Multivariate Analysis

February 17, 2016

Contents

1 Indicator Function **18**

2 Old Datatype package: constructing datatypes from Cartesian Products and Disjoint Sums **21**
2.1 The datatype universe 22
2.2 Freeness: Distinctness of Constructors 24
2.3 Set Constructions 27

3 Bijections between natural numbers and other types **32**
3.1 Type *nat* × *nat* 32
3.2 Type *nat* + *nat* 34
3.3 Type *int* 34
3.4 Type *nat list* 35
3.5 Finite sets of naturals 36
3.5.1 Preliminaries 36
3.5.2 From sets to naturals 37
3.5.3 From naturals to sets 38
3.5.4 Proof of isomorphism 39

4 Encoding (almost) everything into natural numbers **39**
4.1 The class of countable types 40
4.2 Conversion functions 40
4.3 Finite types are countable 40
4.4 Automatically proving countability of old-style datatypes . . 41
4.5 Automatically proving countability of datatypes 43
4.6 More Countable types 44
4.7 The rationals are countably infinite 45

5 Infinite Sets and Related Concepts **46**
5.1 Infinitely Many and Almost All 47
5.2 Enumeration of an Infinite Set 50

6 Countable sets **52**
6.1 Predicate for countable sets . . . 52
6.2 Enumerate a countable set . . . 53
6.3 Closure properties of countability . . . 56
6.4 Misc lemmas . . . 59
6.5 Uncountable . . . 59

7 Pi and Function Sets **60**
7.1 Basic Properties of *Pi* . . . 60
7.2 Composition With a Restricted Domain: *compose* . . . 63
7.3 Bounded Abstraction: *restrict* . . . 63
7.4 Bijections Between Sets . . . 64
7.5 Extensionality . . . 65
7.6 Cardinality . . . 66
7.7 Extensional Function Spaces . . . 67
7.7.1 Injective Extensional Function Spaces . . . 70
7.7.2 Cardinality . . . 71

8 Square root of sum of squares **71**

9 Inner Product Spaces and the Gradient Derivative **75**
9.1 Real inner product spaces . . . 75
9.2 Class instances . . . 80
9.3 Gradient derivative . . . 81

10 Additive group operations on product types **83**
10.1 Operations . . . 83
10.2 Class instances . . . 84

11 Cartesian Products as Vector Spaces **86**
11.1 Product is a real vector space . . . 86
11.2 Product is a topological space . . . 86
11.2.1 Continuity of operations . . . 89
11.2.2 Separation axioms . . . 90
11.3 Product is a metric space . . . 91
11.4 Product is a complete metric space . . . 94
11.5 Product is a normed vector space . . . 94
11.5.1 Pair operations are linear . . . 95
11.5.2 Frechet derivatives involving pairs . . . 96
11.6 Product is an inner product space . . . 97

12 Finite-Dimensional Inner Product Spaces **98**
12.1 Type class of Euclidean spaces . . . 98
12.2 Subclass relationships . . . 100
12.3 Class instances . . . 100

12.3.1 Type *real* . . . 100
12.3.2 Type *complex* . . . 101
12.3.3 Type $'a \times 'b$ . . . 101

13 Elementary linear algebra on Euclidean spaces 102
13.1 Orthogonality. . . . 105
13.2 Linear functions. . . . 106
13.3 Bilinear functions. . . . 109
13.4 Adjoints. . . . 110
13.5 Interlude: Some properties of real sets . . . 111
13.6 A generic notion of "hull" (convex, affine, conic hull and closure). . . . 112
13.7 Archimedean properties and useful consequences . . . 114
13.8 A bit of linear algebra. . . . 116
13.9 Euclidean Spaces as Typeclass . . . 131
13.10 Linearity and Bilinearity continued . . . 132
13.11 We continue. . . . 135
13.12 Infinity norm . . . 158
13.13 Collinearity . . . 163

14 A decision procedure for universal multivariate real arithmetic with addition, multiplication and ordering using semidefinite programming 165

15 General linear decision procedure for normed spaces 165

16 Elementary topology in Euclidean space. 168
16.1 Topological Basis . . . 169
16.2 Countable Basis . . . 172
16.3 Polish spaces . . . 177
16.4 General notion of a topology as a value . . . 177
16.4.1 Main properties of open sets . . . 178
16.4.2 Closed sets . . . 178
16.4.3 Subspace topology . . . 180
16.4.4 The standard Euclidean topology . . . 182
16.5 Open and closed balls . . . 185
16.6 Boxes . . . 188
16.7 Connectedness . . . 197
16.8 Limit points . . . 198
16.9 Interior of a Set . . . 201
16.10 Closure of a Set . . . 204
16.11 Connected components, considered as a connectedness relation or a set . . . 207
16.12 The set of connected components of a set . . . 212

16.13Frontier (aka boundary) . . . 215
16.14Filters and the "eventually true" quantifier . . . 216
16.15Limits . . . 217
16.16Infimum Distance . . . 225
16.17More properties of closed balls . . . 228
16.18Boundedness . . . 235
16.19Compactness . . . 241
16.19.1 Bolzano-Weierstrass property . . . 241
16.19.2 Sequential compactness . . . 252
16.19.3 Totally bounded . . . 258
16.19.4 Heine-Borel theorem . . . 258
16.19.5 Complete the chain of compactness variants . . . 259
16.20Metric spaces with the Heine-Borel property . . . 260
16.20.1 Completeness . . . 264
16.21Relations among convergence and absolute convergence for power series. . . . 271
16.22Bounded closed nest property (proof does not use Heine-Borel) 272
16.23Continuity . . . 276
16.23.1 Structural rules for pointwise continuity . . . 283
16.23.2 Structural rules for setwise continuity . . . 284
16.23.3 Structural rules for uniform continuity . . . 284
16.24Topological stuff lifted from and dropped to R . . . 299
16.25Pasted sets . . . 300
16.26Separation between points and sets . . . 309
16.27Closure of halfspaces and hyperplanes . . . 310
16.28Intervals . . . 315
16.29Homeomorphisms . . . 326
16.30Some properties of a canonical subspace . . . 334
16.31Affine transformations of intervals . . . 336
16.32Banach fixed point theorem (not really topological...) . . . 337
16.33Edelstein fixed point theorem . . . 341

17 Convexity in real vector spaces **346**
17.1 Convexity . . . 346
17.2 Explicit expressions for convexity in terms of arbitrary sums . 349
17.3 Functions that are convex on a set . . . 352
17.4 Arithmetic operations on sets preserve convexity . . . 354
17.5 Convexity of real functions . . . 364

18 Algebraic operations on sets **366**

19 Convex sets, functions and related things. 374
19.1 Affine set and affine hull . . . 382
19.1.1 Some explicit formulations (from Lars Schewe) . . . 382
19.1.2 Stepping theorems and hence small special cases . . . 387
19.1.3 Some relations between affine hull and subspaces . . . 390
19.1.4 Parallel affine sets . . . 392
19.1.5 Subspace parallel to an affine set . . . 394
19.2 Cones . . . 396
19.2.1 Conic hull . . . 397
19.3 Affine dependence and consequential theorems (from Lars Schewe) . . . 400
19.4 Connectedness of convex sets . . . 402
19.5 Convex hull . . . 404
19.5.1 Convex hull is "preserved" by a linear function . . . 405
19.5.2 Stepping theorems for convex hulls of finite sets . . . 406
19.5.3 Explicit expression for convex hull . . . 409
19.5.4 Another formulation from Lars Schewe . . . 412
19.5.5 A stepping theorem for that expansion . . . 414
19.5.6 Hence some special cases . . . 416
19.6 Relations among closure notions and corresponding hulls . . . 417
19.7 Some Properties of Affine Dependent Sets . . . 421
19.8 Affine Dimension of a Set . . . 425
19.9 Caratheodory's theorem. . . . 437
19.10 Relative interior of a set . . . 441
19.10.1 Relative open sets . . . 447
19.10.2 Relative interior preserves under linear transformations 450
19.11 Some Properties of subset of standard basis . . . 453
19.12 Openness and compactness are preserved by convex hull operation. . . . 454
19.13 Extremal points of a simplex are some vertices. . . . 460
19.14 Closest point of a convex set is unique, with a continuous projection. . . . 464
19.14.1 Various point-to-set separating/supporting hyperplane theorems. . . . 467
19.14.2 Now set-to-set for closed/compact sets . . . 469
19.14.3 General case without assuming closure and getting non-strict separation . . . 471
19.15 More convexity generalities . . . 473
19.16 Moving and scaling convex hulls. . . . 474
19.17 Convexity of cone hulls . . . 474
19.18 Convex set as intersection of halfspaces . . . 476
19.19 Radon's theorem (from Lars Schewe) . . . 476
19.20 Helly's theorem . . . 479

19.21 Homeomorphism of all convex compact sets with nonempty interior . . . 482
19.22 Epigraphs of convex functions . . . 492
19.22.1 Use this to derive general bound property of convex function . . . 493
19.23 Convexity of general and special intervals . . . 493
19.24 On *real*, *is-interval*, *convex* and *connected* are all equivalent. . . 494
19.25 Another intermediate value theorem formulation . . . 496
19.26 A bound within a convex hull, and so an interval . . . 497
19.27 Bounded convex function on open set is continuous . . . 501
19.28 Upper bound on a ball implies upper and lower bounds . . . 503
19.28.1 Hence a convex function on an open set is continuous 504
19.29 Line segments, Starlike Sets, etc. . . . 506
19.29.1 More lemmas, especially for working with the underlying formula . . . 510
19.29.2 Betweenness . . . 513
19.30 Shrinking towards the interior of a convex set . . . 515
19.31 Some obvious but surprisingly hard simplex lemmas . . . 517
19.32 Relative interior of convex set . . . 526
19.32.1 Relative interior and closure under common operations 539
19.32.2 Relative interior of convex cone . . . 552
19.33 Convexity on direct sums . . . 558
19.34 Explicit formulas for interior and relative interior of convex hull . . . 565
19.35 Similar results for closure and (relative or absolute) frontier. . 571
19.36 Coplanarity, and collinearity in terms of affine hull . . . 574
19.37 The infimum of the distance between two sets . . . 579

20 Results connected with topological dimension. 583
20.1 The key "counting" observation, somewhat abstracted. . . . 585
20.2 The odd/even result for faces of complete vertices, generalized. 586
20.3 Reduced labelling . . . 607
20.4 The main result for the unit cube . . . 612
20.5 Retractions . . . 620
20.6 Preservation of fixpoints under (more general notion of) retraction . . . 620
20.7 The Brouwer theorem for any set with nonempty interior . . 622

21 Continuous paths and path-connected sets 625
21.1 Paths and Arcs . . . 625
21.2 Invariance theorems . . . 626
21.3 Basic lemmas about paths . . . 628

22 Path Images **631**
22.1 Simple paths with the endpoints removed 633
22.2 The operations on paths 633

23 Choosing a subpath of an existing path **636**
23.1 There is a subpath to the frontier 640
23.2 Reparametrizing a closed curve to start at some chosen point 643
23.3 Special case of straight-line paths 645
23.4 Bounding a point away from a path 646

24 Path component, considered as a "joinability" relation (from Tom Hales) **647**
24.1 Path connectedness of a space 649
24.2 Some useful lemmas about path-connectedness 649
24.3 Sphere is path-connected 656
24.4 Relations between components and path components 660
24.5 Existence of unbounded components 661

25 The "inside" and "outside" of a set **663**

26 Homotopy of maps p,q : X=¿Y with property P of all intermediate maps. **677**
26.1 Trivial properties. 679
26.2 Homotopy of paths, maintaining the same endpoints. 684
26.3 Group properties for homotopy of paths 687
26.4 Homotopy of loops without requiring preservation of endpoints. 689
26.5 Relations between the two variants of homotopy 690
26.6 Homotopy of "nearby" function, paths and loops. 693
26.7 Homotopy and subpaths 695

27 A generic phantom type **697**

28 Cardinality of types **698**
28.1 Preliminary lemmas 698
28.2 Cardinalities of types 699
28.3 Classes with at least 1 and 2 701
28.4 A type class for deciding finiteness of types 702
28.5 A type class for computing the cardinality of types 702
28.6 Instantiations for *card-UNIV* 702
28.7 Code setup for sets 706

29 Numeral Syntax for Types **709**
29.1 Numeral Types 709
29.2 Locales for for modular arithmetic subtypes 711
29.3 Ring class instances 713

29.4 Order instances . . . 715
29.5 Code setup and type classes for code generation . . . 716
29.6 Syntax . . . 719
29.7 Examples . . . 720

30 Definition of finite Cartesian product types. 720
30.1 Finite Cartesian products, with indexing and lambdas. . . . 720
30.2 Group operations and class instances . . . 721
30.3 Real vector space . . . 722
30.4 Topological space . . . 723
30.5 Metric space . . . 725
30.6 Normed vector space . . . 728
30.7 Inner product space . . . 729
30.8 Euclidean space . . . 730

31 Operator Norm 731

32 Uniform Limit and Uniform Convergence 736

33 Bounded Linear Function 747
33.1 Intro rules for *bounded-linear* . . . 747
33.2 declaration of derivative/continuous/tendsto introduction rules for bounded linear functions . . . 748
33.3 type of bounded linear functions . . . 748
33.4 type class instantiations . . . 749
33.5 On Euclidean Space . . . 754
33.6 concrete bounded linear functions . . . 758

34 Multivariate calculus in Euclidean space 761
34.1 Derivatives . . . 761
34.1.1 Combining theorems. . . . 761
34.2 Derivative with composed bilinear function. . . . 762
34.2.1 Caratheodory characterization . . . 763
34.2.2 Limit transformation for derivatives . . . 764
34.3 Differentiability . . . 765
34.4 Frechet derivative and Jacobian matrix . . . 766
34.5 Differentiability implies continuity . . . 766
34.6 The chain rule . . . 767
34.7 Composition rules stated just for differentiability . . . 767
34.8 Uniqueness of derivative . . . 768
34.9 The traditional Rolle theorem in one dimension . . . 771
34.10 One-dimensional mean value theorem . . . 774
34.11 More general bound theorems . . . 776
34.12 Differentiability of inverse function (most basic form) . . . 784

34.13Proving surjectivity via Brouwer fixpoint theorem 789
34.14Uniformly convergent sequence of derivatives 798
34.15Differentiation of a series 805
34.16Relation between convexity and derivative 813
34.17Partial derivatives 815

35 Kurzweil-Henstock Gauge Integration in many dimensions. 819
35.1 Sundries 821
35.2 Some useful lemmas about intervals. 823
35.3 Bounds on intervals where they exist. 827
35.4 Content (length, area, volume...) of an interval. 829
35.5 The notion of a gauge — simply an open set containing the point. 833
35.6 Divisions. 834
35.7 Tagged (partial) divisions. 850
35.8 Fine-ness of a partition w.r.t. a gauge. 855
35.9 Gauge integral. Define on compact intervals first, then use a limit. 855
35.10Some basic combining lemmas. 857
35.11The set we're concerned with must be closed. 858
35.12General bisection principle for intervals; might be useful elsewhere. 858
35.13Cousin's lemma. 865
35.14Basic theorems about integrals. 865
35.15Cauchy-type criterion for integrability. 878
35.16Additivity of integral on abutting intervals. 880
35.17A sort of converse, integrability on subintervals. 889
35.18Generalized notion of additivity. 892
35.19Using additivity of lifted function to encode definedness. . . . 893
35.20Two key instances of additivity. 895
35.21Points of division of a partition. 897
35.22Preservation by divisions and tagged divisions. 900
35.23Additivity of content. 907
35.24Finally, the integral of a constant 907
35.25Bounds on the norm of Riemann sums and the integral itself. 908
35.26Similar theorems about relationship among components. . . . 910
35.27Uniform limit of integrable functions is integrable. 914
35.28Negligible sets. 917
35.29Negligibility of hyperplane. 917
35.30A technical lemma about "refinement" of division. 923
35.31Hence the main theorem about negligible sets. 925
35.32Some other trivialities about negligible sets. 932
35.33Finite case of the spike theorem is quite commonly needed. . 934
35.34In particular, the boundary of an interval is negligible. 935

35.35Integrability of continuous functions. 936
35.36Specialization of additivity to one dimension. 939
35.37Special case of additivity we need for the FCT. 942
35.38A useful lemma allowing us to factor out the content size. . . 943
35.39Fundamental theorem of calculus. 944
35.40Taylor series expansion 946
35.41Attempt a systematic general set of "offset" results for components. 948
35.42Only need trivial subintervals if the interval itself is trivial. . 949
35.43Integrability on subintervals. 951
35.44Combining adjacent intervals in 1 dimension. 952
35.45Reduce integrability to "local" integrability. 953
35.46Second FCT or existence of antiderivative. 954
35.47Combined fundamental theorem of calculus. 956
35.48General "twiddling" for interval-to-interval function image. . 957
35.49Special case of a basic affine transformation. 960
35.50Special case of stretching coordinate axes separately. 962
35.51even more special cases. 964
35.52Stronger form of FCT; quite a tedious proof. 965
35.53Stronger form with finite number of exceptional points. . . . 976
35.54This doesn't directly involve integration, but that gives an easy proof. 985
35.55Generalize a bit to any convex set. 986
35.56Integrating characteristic function of an interval 989
35.57More lemmas that are useful later 998
35.58Continuity of the integral (for a 1-dimensional interval). . . . 1001
35.59A straddling criterion for integrability 1004
35.60Adding integrals over several sets 1008
35.61Also tagged divisions 1012
35.62Henstock's lemma 1013
35.63Geometric progression 1020
35.64Monotone convergence (bounded interval first) 1021
35.65Absolute integrability (this is the same as Lebesgue integrability) 1034
35.66differentiation under the integral sign 1062
35.67Exchange uniform limit and integral 1069
35.68Dominated convergence 1070
35.69Compute a double integral using iterated integrals and switching the order of integration 1076

36 Instantiates the finite Cartesian product of Euclidean spaces as a Euclidean space. 1084
36.1 Basic componentwise operations on vectors. 1085

36.2 A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space. 1086
36.3 Some frequently useful arithmetic lemmas over vectors. 1088
36.4 Matrix operations 1091
36.5 lambda skolemization on cartesian products 1095
36.6 Convex Euclidean Space 1107
36.7 Derivative 1108
36.8 Component of the differential must be zero if it exists at a local maximum or minimum for that corresponding component. 1108
36.9 Lemmas for working on (*real*, *1*) *vec* 1108
36.10 The collapse of the general concepts to dimension one. 1110
36.11 Explicit vector construction from lists. 1110

37 Fashoda meet theorem **1111**
37.1 Bijections between intervals. 1112
37.2 Fashoda meet theorem 1113
37.3 Some slightly ad hoc lemmas I use below 1123
37.4 Useful Fashoda corollary pointed out to me by Tom Hales . . 1127

38 Continuity and iterations (of set transformers) **1131**
38.1 Continuity for complete lattices 1131

39 Extended natural numbers (i.e. with infinity) **1139**
39.1 Type definition 1139
39.2 Constructors and numbers 1140
39.3 Addition 1142
39.4 Multiplication 1143
39.5 Numerals 1144
39.6 Subtraction 1145
39.7 Ordering 1145
39.8 Cancellation simprocs 1149
39.9 Well-ordering 1151
39.10 Complete Lattice 1152
39.11 Traditional theorem names 1153

40 Liminf and Limsup on complete lattices **1153**
40.0.1 *Liminf* and *Limsup* 1153
40.1 More Limits 1162

41 Extended real number line **1164**
41.1 Definition and basic properties 1167
41.1.1 Addition 1170
41.1.2 Linear order on *ereal* 1172
41.1.3 Multiplication 1180

41.1.4 Power . . . 1188
41.1.5 Subtraction . . . 1189
41.1.6 Division . . . 1193
41.2 Complete lattice . . . 1197
41.2.1 Topological space . . . 1199
41.3 Relation to *enat* . . . 1209
41.4 Limits on *ereal* . . . 1211
41.4.1 Convergent sequences . . . 1214
41.4.2 Sums . . . 1225
41.4.3 Continuity . . . 1239
41.4.4 liminf and limsup . . . 1242
41.4.5 Tests for code generator . . . 1244

42 Limits on the Extended real number line 1244
42.1 monoset . . . 1251
42.2 Relate extended reals and the indicator function . . . 1254

43 Permutations, both general and specifically on finite sets. 1255
43.1 Transpositions . . . 1255
43.2 Basic consequences of the definition . . . 1255
43.3 Group properties . . . 1257
43.4 The number of permutations on a finite set . . . 1257
43.5 Permutations of index set for iterated operations . . . 1260
43.6 Various combinations of transpositions with 2, 1 and 0 common elements . . . 1261
43.7 Permutations as transposition sequences . . . 1261
43.8 Some closure properties of the set of permutations, with lengths 1261
43.9 The identity map only has even transposition sequences . . . 1263
43.10 Therefore we have a welldefined notion of parity . . . 1266
43.11 And it has the expected composition properties . . . 1267
43.12 A more abstract characterization of permutations . . . 1268
43.13 Relation to "permutes" . . . 1270
43.14 Hence a sort of induction principle composing by swaps . . . 1271
43.15 Sign of a permutation as a real number . . . 1271
43.16 More lemmas about permutations . . . 1272
43.17 Sum over a set of permutations (could generalize to iteration) 1275

44 Traces, Determinant of square matrices and some properties 1277
44.1 Trace . . . 1277

45 Pointwise order on product types **1302**
45.1 Pointwise ordering . . . 1302
45.2 Binary infimum and supremum . . . 1303
45.3 Top and bottom elements . . . 1304
45.4 Complete lattice operations . . . 1305
45.5 Complete distributive lattices . . . 1307
45.6 An ordering on euclidean spaces that will allow us to talk about intervals . . . 1307

46 Non-negative, non-positive integers and reals **1313**
46.1 Non-positive integers . . . 1314
46.2 Non-negative reals . . . 1316
46.3 Non-positive reals . . . 1318

47 Complex Analysis Basics **1320**
47.1 General lemmas . . . 1320
47.2 DERIV stuff . . . 1322
47.3 Some limit theorems about real part of real series etc. . . . 1323
47.4 Complex number lemmas . . . 1324
47.5 Holomorphic functions . . . 1325
47.6 Caratheodory characterization. . . . 1328
47.7 Holomorphic . . . 1328
47.8 Analyticity on a set . . . 1331
47.9 analyticity at a point. . . . 1336
47.10 Combining theorems for derivative with "analytic at" hypotheses . . . 1336
47.11 Complex differentiation of sequences and series . . . 1337
47.12 Bound theorem . . . 1340
47.13 Inverse function theorem for complex derivatives. . . . 1340
47.14 Taylor on Complex Numbers . . . 1341
47.15 Polynomal function extremal theorem, from HOL Light . . . 1345

48 Bounded Continuous Functions **1347**
48.1 Definition . . . 1347
48.2 Complete Space . . . 1350
48.3 Supremum norm for a normed vector space . . . 1351
48.4 Continuously Extended Functions . . . 1355

49 The Bernstein-Weierstrass and Stone-Weierstrass Theorems **1358**
49.1 Bernstein polynomials . . . 1358
49.2 Explicit Bernstein version of the 1D Weierstrass approximation theorem . . . 1359
49.3 General Stone-Weierstrass theorem . . . 1361
49.4 Polynomial functions . . . 1374

49.5 Stone-Weierstrass theorem for polynomial functions 1379
49.6 Polynomial functions as paths 1382

50 Radius of Convergence and Summation Tests 1384
50.1 Rounded dual logarithm 1384
50.2 Convergence tests for infinite sums 1386
50.2.1 Root test 1386
50.2.2 Cauchy's condensation test 1388
50.2.3 Summability of powers 1391
50.2.4 Kummer's test 1393
50.2.5 Ratio test 1395
50.2.6 Raabe's test 1395
50.3 Radius of convergence 1396

51 Complex Transcendental Functions 1404
51.1 The Exponential Function is Differentiable and Continuous . 1405
51.2 Euler and de Moivre formulas. 1406
51.3 Relationships between real and complex trig functions 1407
51.4 Get a nice real/imaginary separation in Euler's formula. . . . 1408
51.5 More on the Polar Representation of Complex Numbers . . . 1408
51.6 Taylor series for complex exponential, sine and cosine. 1414
51.7 The argument of a complex number 1417
51.8 Analytic properties of tangent function 1423
51.9 Complex logarithms (the conventional principal value) 1423
51.10Relation to Real Logarithm 1424
51.11The Unwinding Number and the Ln-product Formula 1426
51.12Derivative of Ln away from the branch cut 1426
51.13Quadrant-type results for Ln 1427
51.14More Properties of Ln 1429
51.15Relation between Ln and Arg, and hence continuity of Arg . 1432
51.16Complex Powers 1437
51.17Some Limits involving Logarithms 1439
51.18Relation between Square Root and exp/ln, hence its derivative 1442
51.19Complex arctangent 1444
51.20Real arctangent 1450
51.21Inverse Sine 1453
51.22Inverse Cosine 1457
51.23Upper and Lower Bounds for Inverse Sine and Cosine 1460
51.24Interrelations between Arcsin and Arccos 1461
51.25Relationship with Arcsin on the Real Numbers 1463
51.26Relationship with Arccos on the Real Numbers 1464
51.27Some interrelationships among the real inverse trig functions. 1465
51.28continuity results for arcsin and arccos. 1466
51.29Roots of unity 1468

52 Complex path integrals and Cauchy's integral theorem 1470
52.1 Piecewise differentiable functions 1470
52.1.1 The concept of continuously differentiable 1474
52.2 Valid paths, and their start and finish 1481
52.2.1 In particular, all results for paths apply 1481
52.3 Contour Integrals along a path 1482
52.4 Reversing a path 1483
52.5 Joining two paths together 1485
52.6 Shifting the starting point of a (closed) path 1489
52.7 More about straight-line paths 1493
52.8 Relation to subpath construction 1494
52.9 Segments via convex hulls 1498
52.10Arithmetical combining theorems 1500
52.11Operations on path integrals 1502
52.12Arithmetic theorems for path integrability 1504
52.13Reversing a path integral 1504
52.14Reversing the order in a double path integral 1508
52.15The key quadrisection step 1510
52.16Cauchy's theorem for triangles 1512
52.17Version needing function holomorphic in interior only 1517
52.18Version allowing finite number of exceptional points 1523
52.19Cauchy's theorem for an open starlike set 1525
52.20Cauchy's theorem for a convex set 1528
52.21Generalize integrability to local primitives 1531
52.22Constancy of a function from a connected set into a finite, disconnected or discrete set 1541
52.23Winding Numbers 1544
52.24The winding number is an integer 1552
52.25The version with complex integers and equality 1554
52.26Continuity of winding number and invariance on connected sets. 1557
52.27The winding number is constant on a connected region 1561
52.28Winding number is zero "outside" a curve, in various senses . 1562
52.29Cauchy's integral formula, again for a convex enclosing set. . 1568
52.30Homotopy forms of Cauchy's theorem 1569
52.31More winding number properties 1574
52.32Partial circle path 1576
52.33Special case of one complete circle 1582
52.34Uniform convergence of path integral 1587
52.35General stepping result for derivative formulas. 1589
52.36Existence of all higher derivatives. 1595
52.37Morera's theorem. 1596
52.38Combining theorems for higher derivatives including Leibniz rule. 1598

52.39A holomorphic function is analytic, i.e. has local power series. 1605
52.40The Liouville theorem and the Fundamental Theorem of Algebra. 1608
52.41Weierstrass convergence theorem. 1610
52.42Some more simple/convenient versions for applications. . . . 1613
52.43On analytic functions defined by a series. 1615
52.44Equality between holomorphic functions, on open ball then connected set. 1620
52.45Some basic lemmas about poles/singularities. 1621
52.46General, homology form of Cauchy's theorem. 1625

53 Generalised Binomial Theorem **1637**

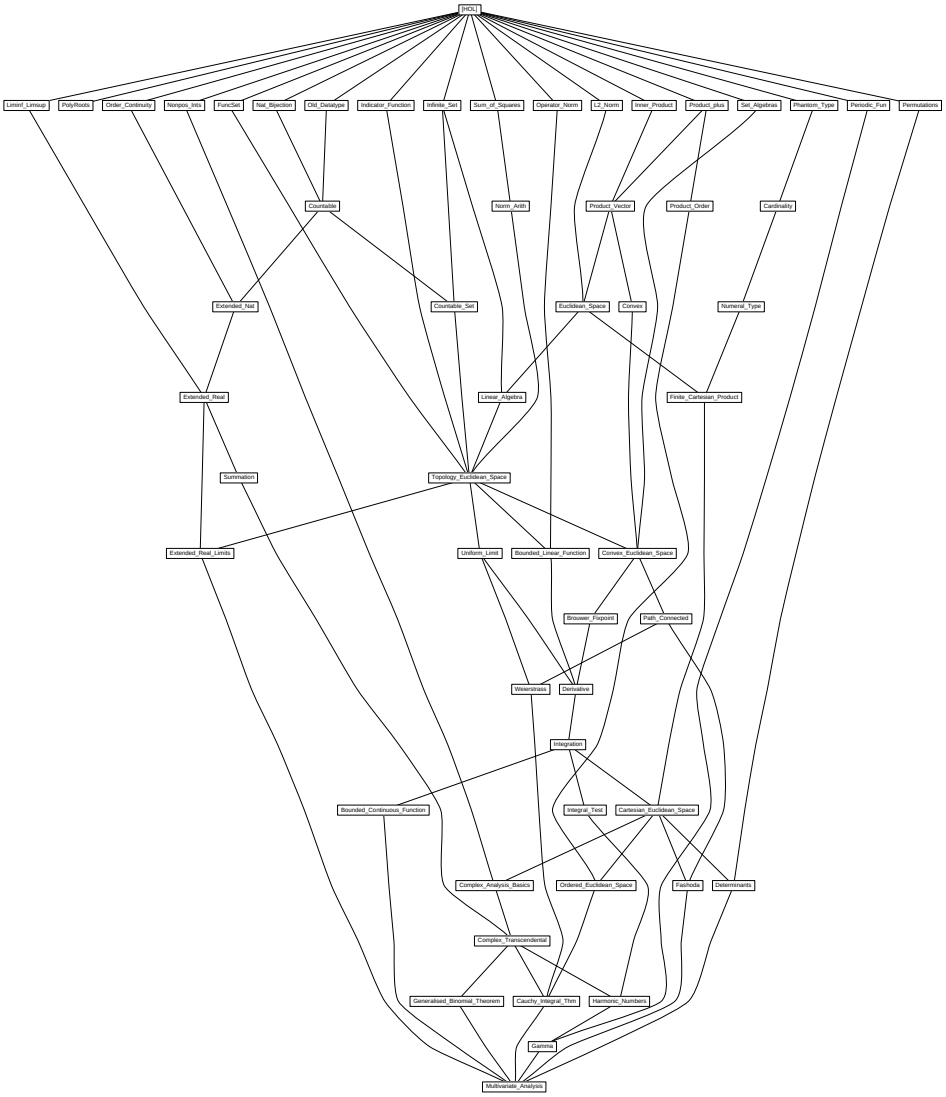
54 Integral Test for Summability **1643**

55 Harmonic Numbers **1645**
55.1 The Harmonic numbers 1646
55.2 The Euler–Mascheroni constant 1647
55.3 Bounds on the Euler–Mascheroni constant 1650

56 Periodic Functions **1657**

57 The Gamma Function **1660**
57.1 Definitions 1664
57.2 Convergence of the Euler series form 1666
57.3 Basic properties 1681
57.4 Differentiability 1684
57.5 Continuity 1685
57.6 Beta function 1697
57.7 Legendre duplication theorem 1698
57.8 Limits and residues 1711
57.9 Alternative definitions 1712
57.9.1 Variant of the Euler form 1712
57.9.2 Weierstrass form 1713
57.9.3 Binomial coefficient form 1715
57.10The Weierstra product formula for the sine 1716
57.11The Solution to the Basel problem 1717

58 polynomial functions: extremal behaviour and root counts 1720
58.1 Geometric progressions 1720
58.2 Basics about polynomial functions: extremal behaviour and root counts. 1721


HOL
Lim
PolyRoots
Order_Continuity
Nonpos_Ints
FuncSet
Nat_Bijection
Old_Datatype
Indicator_Function
Infinite_Set
Sum_of_Squares
Operator_Norm
L2_Norm
Inner_Product
Product_plus
Set_Algebras
Phantom_Type
Pseudo_Fun
Permutations
Countable
Norm_Arith
Product_Vector
Product_Order
Cardinality
Extended_Nat
Countable_Set
Euclidean_Space
Convex
Numeral_Type
Extended_Real
Linear_Algebra
Finite_Cartesian_Product
Summation
Topology_Euclidean_Space
Extended_Real_Limits
Uniform_Limit
Bounded_Linear_Function
Convex_Euclidean_Space
Brouwer_Fixpoint
Path_Connected
Weierstrass
Derivative
Integration
Bounded_Continuous_Function
Integral_Test
Cartesian_Euclidean_Space
Complex_Analysis_Basics
Ordered_Euclidean_Space
Fashoda
Determinants
Complex_Transcendental
Generalised_Binomial_Theorem
Cauchy_Integral_Thm
Harmonic_Numbers
Gamma
Multivariate_Analysis

1 Indicator Function

theory *Indicator-Function*
imports *Complex-Main*
begin

definition *indicator S x* = (*if x* ∈ *S then 1 else 0*)

lemma *indicator-simps*[*simp*]:
x ∈ *S* ⟹ *indicator S x* = *1*
x ∉ *S* ⟹ *indicator S x* = *0*
unfolding *indicator-def* **by** *auto*

lemma *indicator-pos-le*[*intro*, *simp*]: (*0*::'*a*::*linordered-semidom*) ≤ *indicator S x*
and *indicator-le-1*[*intro*, *simp*]: *indicator S x* ≤ (*1*::'*a*::*linordered-semidom*)
unfolding *indicator-def* **by** *auto*

lemma *indicator-abs-le-1*: |*indicator S x*| ≤ (*1*::'*a*::*linordered-idom*)
unfolding *indicator-def* **by** *auto*

lemma *indicator-eq-0-iff*: *indicator A x* = (*0*::-::*zero-neq-one*) ⟷ *x* ∉ *A*
by (*auto simp*: *indicator-def*)

lemma *indicator-eq-1-iff*: *indicator A x* = (*1*::-::*zero-neq-one*) ⟷ *x* ∈ *A*
by (*auto simp*: *indicator-def*)

lemma *split-indicator*: *P* (*indicator S x*) ⟷ ((*x* ∈ *S* ⟶ *P 1*) ∧ (*x* ∉ *S* ⟶ *P 0*))
unfolding *indicator-def* **by** *auto*

lemma *split-indicator-asm*: *P* (*indicator S x*) ⟷ (¬ (*x* ∈ *S* ∧ ¬ *P 1* ∨ *x* ∉ *S* ∧ ¬ *P 0*))
unfolding *indicator-def* **by** *auto*

lemma *indicator-inter-arith*: *indicator* (*A* ∩ *B*) *x* = *indicator A x* ∗ (*indicator B x*::'*a*::*semiring-1*)
unfolding *indicator-def* **by** (*auto simp*: *min-def max-def*)

lemma *indicator-union-arith*: *indicator* (*A* ∪ *B*) *x* = *indicator A x* + *indicator B x* − *indicator A x* ∗ (*indicator B x*::'*a*::*ring-1*)
unfolding *indicator-def* **by** (*auto simp*: *min-def max-def*)

lemma *indicator-inter-min*: *indicator* (*A* ∩ *B*) *x* = *min* (*indicator A x*) (*indicator B x*::'*a*::*linordered-semidom*)
and *indicator-union-max*: *indicator* (*A* ∪ *B*) *x* = *max* (*indicator A x*) (*indicator B x*::'*a*::*linordered-semidom*)
unfolding *indicator-def* **by** (*auto simp*: *min-def max-def*)

lemma *indicator-disj-union*: *A* ∩ *B* = {} ⟹ *indicator* (*A* ∪ *B*) *x* = (*indicator*

A x + indicator B x::*'a*::*linordered-semidom*)
by (*auto split*: *split-indicator*)

lemma *indicator-compl*: *indicator* (− *A*) *x* = *1* − (*indicator A x*::*'a*::*ring-1*)
and *indicator-diff*: *indicator* (*A* − *B*) *x* = *indicator A x* ∗ (*1* − *indicator B x*::*'a*::*ring-1*)
unfolding *indicator-def* **by** (*auto simp*: *min-def max-def*)

lemma *indicator-times*: *indicator* (*A* × *B*) *x* = *indicator A* (*fst x*) ∗ (*indicator B* (*snd x*)::*'a*::*semiring-1*)
unfolding *indicator-def* **by** (*cases x*) *auto*

lemma *indicator-sum*: *indicator* (*A* <+> *B*) *x* = (*case x of Inl x* ⇒ *indicator A x* | *Inr x* ⇒ *indicator B x*)
unfolding *indicator-def* **by** (*cases x*) *auto*

lemma *indicator-image*: *inj f* ⟹ *indicator* (*f* ‘ *X*) (*f x*) = (*indicator X x*::-::*zero-neq-one*)
by (*auto simp*: *indicator-def inj-on-def*)

lemma *indicator-vimage*: *indicator* (*f* −‘ *A*) *x* = *indicator A* (*f x*)
by(*auto split*: *split-indicator*)

lemma
fixes *f* :: *'a* ⇒ *'b*::*semiring-1* **assumes** *finite A*
shows *setsum-mult-indicator*[*simp*]: (∑ *x* ∈ *A*. *f x* ∗ *indicator B x*) = (∑ *x* ∈ *A* ∩ *B*. *f x*)
and *setsum-indicator-mult*[*simp*]: (∑ *x* ∈ *A*. *indicator B x* ∗ *f x*) = (∑ *x* ∈ *A* ∩ *B*. *f x*)
unfolding *indicator-def*
using *assms* **by** (*auto intro*!: *setsum.mono-neutral-cong-right split*: *split-if-asm*)

lemma *setsum-indicator-eq-card*:
assumes *finite A*
shows (∑ *x* ∈ *A*. *indicator B x*) = *card* (*A Int B*)
using *setsum-mult-indicator*[*OF assms*, *of* %*x*. *1*::*nat*]
unfolding *card-eq-setsum* **by** *simp*

lemma *setsum-indicator-scaleR*[*simp*]:
finite A ⟹
(∑ *x* ∈ *A*. *indicator* (*B x*) (*g x*) ∗*R f x*) = (∑ *x* ∈ {*x*∈*A*. *g x* ∈ *B x*}. *f x*::*'a*::*real-vector*)
using *assms* **by** (*auto intro*!: *setsum.mono-neutral-cong-right split*: *split-if-asm simp*: *indicator-def*)

lemma *LIMSEQ-indicator-incseq*:
assumes *incseq A*
shows (λ*i*. *indicator* (*A i*) *x* :: *'a* :: {*topological-space*, *one*, *zero*}) ⟶ *indicator* (⋃ *i*. *A i*) *x*
proof *cases*

assume $\exists i.\ x \in A\ i$
then obtain i **where** $x \in A\ i$
by *auto*
then have
$\bigwedge n.\ (indicator\ (A\ (n + i))\ x :: 'a) = 1$
$(indicator\ (\bigcup i.\ A\ i)\ x :: 'a) = 1$
using *incseqD*[*OF* ‹*incseq A*›, *of i n* + *i* **for** *n*] ‹$x \in A\ i$› **by** (*auto simp*: *indicator-def*)
then show *?thesis*
by (*rule-tac LIMSEQ-offset*[*of* - *i*]) *simp*
qed (*auto simp*: *indicator-def*)

lemma *LIMSEQ-indicator-UN*:
$(\lambda k.\ indicator\ (\bigcup i{<}k.\ A\ i)\ x :: 'a :: \{topological\text{-}space, one, zero\}) \longrightarrow indicator\ (\bigcup i.\ A\ i)\ x$
proof −
have $(\lambda k.\ indicator\ (\bigcup i{<}k.\ A\ i)\ x{::}'a) \longrightarrow indicator\ (\bigcup k.\ \bigcup i{<}k.\ A\ i)\ x$
by (*intro LIMSEQ-indicator-incseq*) (*auto simp*: *incseq-def intro*: *less-le-trans*)
also have $(\bigcup k.\ \bigcup i{<}k.\ A\ i) = (\bigcup i.\ A\ i)$
by *auto*
finally show *?thesis* **.**
qed

lemma *LIMSEQ-indicator-decseq*:
assumes *decseq A*
shows $(\lambda i.\ indicator\ (A\ i)\ x :: 'a :: \{topological\text{-}space, one, zero\}) \longrightarrow indicator\ (\bigcap i.\ A\ i)\ x$
proof *cases*
assume $\exists i.\ x \notin A\ i$
then obtain i **where** $x \notin A\ i$
by *auto*
then have
$\bigwedge n.\ (indicator\ (A\ (n + i))\ x :: 'a) = 0$
$(indicator\ (\bigcap i.\ A\ i)\ x :: 'a) = 0$
using *decseqD*[*OF* ‹*decseq A*›, *of i n* + *i* **for** *n*] ‹$x \notin A\ i$› **by** (*auto simp*: *indicator-def*)
then show *?thesis*
by (*rule-tac LIMSEQ-offset*[*of* - *i*]) *simp*
qed (*auto simp*: *indicator-def*)

lemma *LIMSEQ-indicator-INT*:
$(\lambda k.\ indicator\ (\bigcap i{<}k.\ A\ i)\ x :: 'a :: \{topological\text{-}space, one, zero\}) \longrightarrow indicator\ (\bigcap i.\ A\ i)\ x$
proof −
have $(\lambda k.\ indicator\ (\bigcap i{<}k.\ A\ i)\ x{::}'a) \longrightarrow indicator\ (\bigcap k.\ \bigcap i{<}k.\ A\ i)\ x$
by (*intro LIMSEQ-indicator-decseq*) (*auto simp*: *decseq-def intro*: *less-le-trans*)
also have $(\bigcap k.\ \bigcap i{<}k.\ A\ i) = (\bigcap i.\ A\ i)$
by *auto*
finally show *?thesis* **.**

qed

lemma *indicator-add*:
$A \cap B = \{\} \Longrightarrow (indicator\ A\ x::-::monoid\text{-}add) + indicator\ B\ x = indicator\ (A \cup B)\ x$
unfolding *indicator-def* **by** *auto*

lemma *of-real-indicator*: *of-real* (*indicator A x*) = *indicator A x*
by (*simp split*: *split-indicator*)

lemma *real-of-nat-indicator*: *real* (*indicator A x* :: *nat*) = *indicator A x*
by (*simp split*: *split-indicator*)

lemma *abs-indicator*: $|indicator\ A\ x :: 'a::linordered\text{-}idom| = indicator\ A\ x$
by (*simp split*: *split-indicator*)

lemma *mult-indicator-subset*:
$A \subseteq B \Longrightarrow indicator\ A\ x * indicator\ B\ x = (indicator\ A\ x :: 'a::\{comm\text{-}semiring\text{-}1\})$
by (*auto split*: *split-indicator simp*: *fun-eq-iff*)

lemma *indicator-sums*:
assumes $\bigwedge i\ j.\ i \neq j \Longrightarrow A\ i \cap A\ j = \{\}$
shows $(\lambda i.\ indicator\ (A\ i)\ x::real)\ sums\ indicator\ (\bigcup i.\ A\ i)\ x$
proof *cases*
assume $\exists i.\ x \in A\ i$
then obtain *i* **where** *i*: $x \in A\ i$ **..**
with *assms* **have** $(\lambda i.\ indicator\ (A\ i)\ x::real)\ sums\ (\sum i \in \{i\}.\ indicator\ (A\ i)\ x)$
by (*intro sums-finite*) (*auto split*: *split-indicator*)
also have $(\sum i \in \{i\}.\ indicator\ (A\ i)\ x) = indicator\ (\bigcup i.\ A\ i)\ x$
using *i* **by** (*auto split*: *split-indicator*)
finally show *?thesis* **.**
qed *simp*

end

2 Old Datatype package: constructing datatypes from Cartesian Products and Disjoint Sums

theory *Old-Datatype*
imports *../Main*
keywords *old-datatype* :: *thy-decl*
begin

ML-file ~~/*src/HOL/Tools/datatype-realizer.ML*

2.1 The datatype universe

definition *Node* = {*p*. *EX f x k*. *p* = (*f* :: *nat* => *'b* + *nat*, *x* ::*'a* + *nat*) & *f k* = *Inr 0*}

typedef (*'a*, *'b*) *node* = *Node* :: ((*nat* => *'b* + *nat*) * (*'a* + *nat*)) *set*
morphisms *Rep-Node Abs-Node*
unfolding *Node-def* **by** *auto*

Datatypes will be represented by sets of type *node*

type-synonym *'a item* = (*'a*, *unit*) *node set*
type-synonym (*'a*, *'b*) *dtree* = (*'a*, *'b*) *node set*

definition *Push* :: [(*'b* + *nat*), *nat* => (*'b* + *nat*)] => (*nat* => (*'b* + *nat*))

where *Push* == (%*b h*. *case-nat b h*)

definition *Push-Node* :: [(*'b* + *nat*), (*'a*, *'b*) *node*] => (*'a*, *'b*) *node*
where *Push-Node* == (%*n x*. *Abs-Node* (*apfst* (*Push n*) (*Rep-Node x*)))

definition *Atom* :: (*'a* + *nat*) => (*'a*, *'b*) *dtree*
where *Atom* == (%*x*. {*Abs-Node*((%*k*. *Inr 0*, *x*))})
definition *Scons* :: [(*'a*, *'b*) *dtree*, (*'a*, *'b*) *dtree*] => (*'a*, *'b*) *dtree*
where *Scons M N* == (*Push-Node* (*Inr 1*) ‘ *M*) *Un* (*Push-Node* (*Inr* (*Suc 1*)) ‘ *N*)

definition *Leaf* :: *'a* => (*'a*, *'b*) *dtree*
where *Leaf* == *Atom o Inl*
definition *Numb* :: *nat* => (*'a*, *'b*) *dtree*
where *Numb* == *Atom o Inr*

definition *In0* :: (*'a*, *'b*) *dtree* => (*'a*, *'b*) *dtree*
where *In0*(*M*) == *Scons* (*Numb 0*) *M*
definition *In1* :: (*'a*, *'b*) *dtree* => (*'a*, *'b*) *dtree*
where *In1*(*M*) == *Scons* (*Numb 1*) *M*

definition *Lim* :: (*'b* => (*'a*, *'b*) *dtree*) => (*'a*, *'b*) *dtree*
where *Lim f* == ⋃{*z*. *? x*. *z* = *Push-Node* (*Inl x*) ‘ (*f x*)}

definition *ndepth* :: (*'a*, *'b*) *node* => *nat*
where *ndepth*(*n*) == (%(*f*,*x*). *LEAST k*. *f k* = *Inr 0*) (*Rep-Node n*)
definition *ntrunc* :: [*nat*, (*'a*, *'b*) *dtree*] => (*'a*, *'b*) *dtree*

where *ntrunc k N == {n. n:N & ndepth(n)<k}*

definition *uprod* :: *[('a, 'b) dtree set, ('a, 'b) dtree set]=> ('a, 'b) dtree set*
where *uprod A B == UN x:A. UN y:B. { Scons x y }*
definition *usum* :: *[('a, 'b) dtree set, ('a, 'b) dtree set]=> ('a, 'b) dtree set*
where *usum A B == In0`A Un In1`B*

definition *Split* :: *[[('a, 'b) dtree, ('a, 'b) dtree]=>'c, ('a, 'b) dtree] => 'c*
where *Split c M == THE u. EX x y. M = Scons x y & u = c x y*

definition *Case* :: *[[('a, 'b) dtree]=>'c, [('a, 'b) dtree]=>'c, ('a, 'b) dtree] => 'c*
where *Case c d M == THE u. (EX x . M = In0(x) & u = c(x)) | (EX y . M = In1(y) & u = d(y))*

definition *dprod* :: *[(('a, 'b) dtree * ('a, 'b) dtree)set, (('a, 'b) dtree * ('a, 'b) dtree)set]*
*=> (('a, 'b) dtree * ('a, 'b) dtree)set*
where *dprod r s == UN (x,x'):r. UN (y,y'):s. {(Scons x y, Scons x' y')}*

definition *dsum* :: *[(('a, 'b) dtree * ('a, 'b) dtree)set, (('a, 'b) dtree * ('a, 'b) dtree)set]*
*=> (('a, 'b) dtree * ('a, 'b) dtree)set*
where *dsum r s == (UN (x,x'):r. {(In0(x),In0(x'))}) Un (UN (y,y'):s. {(In1(y),In1(y'))})*

lemma *apfst-convE*:
[| q = apfst f p; !!x y. [| p = (x,y); q = (f(x),y) |] ==> R |] ==> R
by (*force simp add*: *apfst-def*)

lemma *Push-inject1*: *Push i f = Push j g ==> i=j*
apply (*simp add*: *Push-def fun-eq-iff*)
apply (*drule-tac x=0* **in** *spec*, *simp*)
done

lemma *Push-inject2*: *Push i f = Push j g ==> f=g*
apply (*auto simp add*: *Push-def fun-eq-iff*)
apply (*drule-tac x=Suc x* **in** *spec*, *simp*)
done

lemma *Push-inject*:
[| Push i f =Push j g; [| i=j; f=g |] ==> P |] ==> P

by (*blast dest*: *Push-inject1 Push-inject2*)

lemma *Push-neq-K0*: *Push* (*Inr* (*Suc k*)) *f* = (%*z*. *Inr 0*) ==> *P*
by (*auto simp add*: *Push-def fun-eq-iff split*: *nat.split-asm*)

lemmas *Abs-Node-inj* = *Abs-Node-inject* [*THEN* [*2*] *rev-iffD1*]

lemma *Node-K0-I*: (%*k*. *Inr 0*, *a*) : *Node*
by (*simp add*: *Node-def*)

lemma *Node-Push-I*: *p*: *Node* ==> *apfst* (*Push i*) *p* : *Node*
apply (*simp add*: *Node-def Push-def*)
apply (*fast intro*!: *apfst-conv nat.case*(*2*)[*THEN trans*])
done

2.2 Freeness: Distinctness of Constructors

lemma *Scons-not-Atom* [*iff*]: *Scons M N* ≠ *Atom*(*a*)
unfolding *Atom-def Scons-def Push-Node-def One-nat-def*
by (*blast intro*: *Node-K0-I Rep-Node* [*THEN Node-Push-I*]
dest!: *Abs-Node-inj*
elim!: *apfst-convE sym* [*THEN Push-neq-K0*])

lemmas *Atom-not-Scons* [*iff*] = *Scons-not-Atom* [*THEN not-sym*]

lemma *inj-Atom*: *inj*(*Atom*)
apply (*simp add*: *Atom-def*)
apply (*blast intro*!: *inj-onI Node-K0-I dest*!: *Abs-Node-inj*)
done
lemmas *Atom-inject* = *inj-Atom* [*THEN injD*]

lemma *Atom-Atom-eq* [*iff*]: (*Atom*(*a*)=*Atom*(*b*)) = (*a*=*b*)
by (*blast dest*!: *Atom-inject*)

lemma *inj-Leaf*: *inj*(*Leaf*)
apply (*simp add*: *Leaf-def o-def*)
apply (*rule inj-onI*)
apply (*erule Atom-inject* [*THEN Inl-inject*])
done

lemmas *Leaf-inject* [*dest*!] = *inj-Leaf* [*THEN injD*]

lemma *inj-Numb*: *inj*(*Numb*)
apply (*simp add*: *Numb-def o-def*)
apply (*rule inj-onI*)
apply (*erule Atom-inject* [*THEN Inr-inject*])
done

lemmas *Numb-inject* [*dest*!] = *inj-Numb* [*THEN injD*]

lemma *Push-Node-inject*:
[| *Push-Node i m* =*Push-Node j n*; [| *i*=*j*; *m*=*n* |] ==> *P*
|] ==> *P*
apply (*simp add*: *Push-Node-def*)
apply (*erule Abs-Node-inj* [*THEN apfst-convE*])
apply (*rule Rep-Node* [*THEN Node-Push-I*])+
apply (*erule sym* [*THEN apfst-convE*])
apply (*blast intro*: *Rep-Node-inject* [*THEN iffD1*] *trans sym elim*!: *Push-inject*)
done

lemma *Scons-inject-lemma1*: *Scons M N* <= *Scons M' N'* ==> *M*<=*M'*
unfolding *Scons-def One-nat-def*
by (*blast dest*!: *Push-Node-inject*)

lemma *Scons-inject-lemma2*: *Scons M N* <= *Scons M' N'* ==> *N*<=*N'*
unfolding *Scons-def One-nat-def*
by (*blast dest*!: *Push-Node-inject*)

lemma *Scons-inject1*: *Scons M N* = *Scons M' N'* ==> *M*=*M'*
apply (*erule equalityE*)
apply (*iprover intro*: *equalityI Scons-inject-lemma1*)
done

lemma *Scons-inject2*: *Scons M N* = *Scons M' N'* ==> *N*=*N'*
apply (*erule equalityE*)
apply (*iprover intro*: *equalityI Scons-inject-lemma2*)
done

lemma *Scons-inject*:
[| *Scons M N* = *Scons M' N'*; [| *M*=*M'*; *N*=*N'* |] ==> *P* |] ==> *P*
by (*iprover dest*: *Scons-inject1 Scons-inject2*)

lemma *Scons-Scons-eq* [*iff*]: (*Scons M N* = *Scons M' N'*) = (*M*=*M'* & *N*=*N'*)
by (*blast elim*!: *Scons-inject*)

lemma *Scons-not-Leaf* [*iff*]: *Scons M N* ≠ *Leaf*(*a*)
unfolding *Leaf-def o-def* **by** (*rule Scons-not-Atom*)

lemmas *Leaf-not-Scons* [*iff*] = *Scons-not-Leaf* [*THEN not-sym*]

lemma *Scons-not-Numb* [*iff*]: *Scons M N* ≠ *Numb*(*k*)
unfolding *Numb-def o-def* **by** (*rule Scons-not-Atom*)

lemmas *Numb-not-Scons* [*iff*] = *Scons-not-Numb* [*THEN not-sym*]

lemma *Leaf-not-Numb* [*iff*]: *Leaf*(*a*) ≠ *Numb*(*k*)
by (*simp add*: *Leaf-def Numb-def*)

lemmas *Numb-not-Leaf* [*iff*] = *Leaf-not-Numb* [*THEN not-sym*]

lemma *ndepth-K0*: *ndepth* (*Abs-Node*(%*k*. *Inr 0*, *x*)) = *0*
by (*simp add*: *ndepth-def Node-K0-I* [*THEN Abs-Node-inverse*] *Least-equality*)

lemma *ndepth-Push-Node-aux*:
case-nat (*Inr* (*Suc i*)) *f k* = *Inr 0* −−> *Suc*(*LEAST x*. *f x* = *Inr 0*) <= *k*
apply (*induct-tac k*, *auto*)
apply (*erule Least-le*)
done

lemma *ndepth-Push-Node*:
ndepth (*Push-Node* (*Inr* (*Suc i*)) *n*) = *Suc*(*ndepth*(*n*))
apply (*insert Rep-Node* [*of n*, *unfolded Node-def*])
apply (*auto simp add*: *ndepth-def Push-Node-def*
Rep-Node [*THEN Node-Push-I*, *THEN Abs-Node-inverse*])
apply (*rule Least-equality*)
apply (*auto simp add*: *Push-def ndepth-Push-Node-aux*)
apply (*erule LeastI*)
done

lemma *ntrunc-0* [*simp*]: *ntrunc 0 M* = {}
by (*simp add*: *ntrunc-def*)

lemma *ntrunc-Atom* [*simp*]: *ntrunc* (*Suc k*) (*Atom a*) = *Atom*(*a*)
by (*auto simp add*: *Atom-def ntrunc-def ndepth-K0*)

lemma *ntrunc-Leaf* [*simp*]: *ntrunc* (*Suc k*) (*Leaf a*) = *Leaf*(*a*)
unfolding *Leaf-def o-def* **by** (*rule ntrunc-Atom*)

lemma *ntrunc-Numb* [*simp*]: *ntrunc* (*Suc k*) (*Numb i*) = *Numb*(*i*)
unfolding *Numb-def o-def* **by** (*rule ntrunc-Atom*)

lemma *ntrunc-Scons* [*simp*]:
ntrunc (*Suc k*) (*Scons M N*) = *Scons* (*ntrunc k M*) (*ntrunc k N*)
unfolding *Scons-def ntrunc-def One-nat-def*
by (*auto simp add*: *ndepth-Push-Node*)

lemma *ntrunc-one-In0* [*simp*]: *ntrunc* (*Suc 0*) (*In0 M*) = {}
apply (*simp add*: *In0-def*)
apply (*simp add*: *Scons-def*)
done

lemma *ntrunc-In0* [*simp*]: *ntrunc* (*Suc*(*Suc k*)) (*In0 M*) = *In0* (*ntrunc* (*Suc k*) *M*)
by (*simp add*: *In0-def*)

lemma *ntrunc-one-In1* [*simp*]: *ntrunc* (*Suc 0*) (*In1 M*) = {}
apply (*simp add*: *In1-def*)
apply (*simp add*: *Scons-def*)
done

lemma *ntrunc-In1* [*simp*]: *ntrunc* (*Suc*(*Suc k*)) (*In1 M*) = *In1* (*ntrunc* (*Suc k*) *M*)
by (*simp add*: *In1-def*)

2.3 Set Constructions

lemma *uprodI* [*intro*!]: [| *M*:*A*; *N*:*B* |] ==> *Scons M N* : *uprod A B*
by (*simp add*: *uprod-def*)

lemma *uprodE* [*elim*!]:
[| *c* : *uprod A B*;
!!*x y*. [| *x*:*A*; *y*:*B*; *c* = *Scons x y* |] ==> *P*

$|]$ ==> P
by (*auto simp add*: *uprod-def*)

lemma *uprodE2*: [| *Scons M N* : *uprod A B*; [| M:A; N:B |] ==> P |] ==> P
by (*auto simp add*: *uprod-def*)

lemma *usum-In0I* [*intro*]: M:A ==> *In0*(M) : *usum A B*
by (*simp add*: *usum-def*)

lemma *usum-In1I* [*intro*]: N:B ==> *In1*(N) : *usum A B*
by (*simp add*: *usum-def*)

lemma *usumE* [*elim*!]:
[| u : *usum A B*;
!!x. [| x:A; u=*In0*(x) |] ==> P;
!!y. [| y:B; u=*In1*(y) |] ==> P
|] ==> P
by (*auto simp add*: *usum-def*)

lemma *In0-not-In1* [*iff*]: *In0*(M) ≠ *In1*(N)
unfolding *In0-def In1-def One-nat-def* **by** *auto*

lemmas *In1-not-In0* [*iff*] = *In0-not-In1* [*THEN not-sym*]

lemma *In0-inject*: *In0*(M) = *In0*(N) ==> M=N
by (*simp add*: *In0-def*)

lemma *In1-inject*: *In1*(M) = *In1*(N) ==> M=N
by (*simp add*: *In1-def*)

lemma *In0-eq* [*iff*]: (*In0 M* = *In0 N*) = (M=N)
by (*blast dest*!: *In0-inject*)

lemma *In1-eq* [*iff*]: (*In1 M* = *In1 N*) = (M=N)
by (*blast dest*!: *In1-inject*)

lemma *inj-In0*: *inj In0*
by (*blast intro*!: *inj-onI*)

lemma *inj-In1*: *inj In1*
by (*blast intro*!: *inj-onI*)

lemma *Lim-inject*: *Lim f = Lim g ==> f = g*
apply (*simp add*: *Lim-def*)
apply (*rule ext*)
apply (*blast elim*!: *Push-Node-inject*)
done

lemma *ntrunc-subsetI*: *ntrunc k M <= M*
by (*auto simp add*: *ntrunc-def*)

lemma *ntrunc-subsetD*: (!!*k*. *ntrunc k M <= N*) *==> M<=N*
by (*auto simp add*: *ntrunc-def*)

lemma *ntrunc-equality*: (!!*k*. *ntrunc k M = ntrunc k N*) *==> M=N*
apply (*rule equalityI*)
apply (*rule-tac* [!] *ntrunc-subsetD*)
apply (*rule-tac* [!] *ntrunc-subsetI* [*THEN* [*2*] *subset-trans*], *auto*)
done

lemma *ntrunc-o-equality*:
[| !!*k*. (*ntrunc*(*k*) *o h1*) = (*ntrunc*(*k*) *o h2*) |] *==> h1=h2*
apply (*rule ntrunc-equality* [*THEN ext*])
apply (*simp add*: *fun-eq-iff*)
done

lemma *uprod-mono*: [| *A<=A'*; *B<=B'* |] *==> uprod A B <= uprod A' B'*
by (*simp add*: *uprod-def*, *blast*)

lemma *usum-mono*: [| *A<=A'*; *B<=B'* |] *==> usum A B <= usum A' B'*
by (*simp add*: *usum-def*, *blast*)

lemma *Scons-mono*: [| *M<=M'*; *N<=N'* |] *==> Scons M N <= Scons M' N'*
by (*simp add*: *Scons-def*, *blast*)

lemma *In0-mono*: *M<=N ==> In0*(*M*) *<= In0*(*N*)
by (*simp add*: *In0-def Scons-mono*)

lemma *In1-mono*: *M<=N ==> In1*(*M*) *<= In1*(*N*)
by (*simp add*: *In1-def Scons-mono*)

lemma *Split* [*simp*]: *Split c* (*Scons M N*) = *c M N*
by (*simp add*: *Split-def*)

lemma *Case-In0* [*simp*]: *Case c d* (*In0 M*) = *c*(*M*)
by (*simp add*: *Case-def*)

lemma *Case-In1* [*simp*]: *Case c d* (*In1 N*) = *d*(*N*)
by (*simp add*: *Case-def*)

lemma *ntrunc-UN1*: *ntrunc k* (*UN x*. *f*(*x*)) = (*UN x*. *ntrunc k* (*f x*))
by (*simp add*: *ntrunc-def*, *blast*)

lemma *Scons-UN1-x*: *Scons* (*UN x*. *f x*) *M* = (*UN x*. *Scons* (*f x*) *M*)
by (*simp add*: *Scons-def*, *blast*)

lemma *Scons-UN1-y*: *Scons M* (*UN x*. *f x*) = (*UN x*. *Scons M* (*f x*))
by (*simp add*: *Scons-def*, *blast*)

lemma *In0-UN1*: *In0*(*UN x*. *f*(*x*)) = (*UN x*. *In0*(*f*(*x*)))
by (*simp add*: *In0-def Scons-UN1-y*)

lemma *In1-UN1*: *In1*(*UN x*. *f*(*x*)) = (*UN x*. *In1*(*f*(*x*)))
by (*simp add*: *In1-def Scons-UN1-y*)

lemma *dprodI* [*intro*!]:
[| (*M*,*M'*):*r*; (*N*,*N'*):*s* |] ==> (*Scons M N*, *Scons M' N'*) : *dprod r s*
by (*auto simp add*: *dprod-def*)

lemma *dprodE* [*elim*!]:
[| *c* : *dprod r s*;
!!*x y x' y'*. [| (*x*,*x'*) : *r*; (*y*,*y'*) : *s*;
c = (*Scons x y*, *Scons x' y'*) |] ==> *P*
|] ==> *P*
by (*auto simp add*: *dprod-def*)

lemma *dsum-In0I* [*intro*]: (*M*,*M'*):*r* ==> (*In0*(*M*), *In0*(*M'*)) : *dsum r s*
by (*auto simp add*: *dsum-def*)

lemma *dsum-In1I* [*intro*]: (*N*,*N'*):*s* ==> (*In1*(*N*), *In1*(*N'*)) : *dsum r s*
by (*auto simp add*: *dsum-def*)

lemma *dsumE* [*elim*!]:
[| *w* : *dsum r s*;
!!*x x'*. [| (*x*,*x'*) : *r*; *w* = (*In0*(*x*), *In0*(*x'*)) |] ==> *P*;
!!*y y'*. [| (*y*,*y'*) : *s*; *w* = (*In1*(*y*), *In1*(*y'*)) |] ==> *P*
|] ==> *P*
by (*auto simp add*: *dsum-def*)

lemma *dprod-mono*: [| *r*<=*r'*; *s*<=*s'* |] ==> *dprod r s* <= *dprod r' s'*
by *blast*

lemma *dsum-mono*: [| *r*<=*r'*; *s*<=*s'* |] ==> *dsum r s* <= *dsum r' s'*
by *blast*

lemma *dprod-Sigma*: (*dprod* (*A* × *B*) (*C* × *D*)) <= (*uprod A C*) × (*uprod B D*)
by *blast*

lemmas *dprod-subset-Sigma* = *subset-trans* [*OF dprod-mono dprod-Sigma*]

lemma *dprod-subset-Sigma2*:
(*dprod* (*Sigma A B*) (*Sigma C D*)) <= *Sigma* (*uprod A C*) (*Split* (%*x y*. *uprod* (*B x*) (*D y*)))
by *auto*

lemma *dsum-Sigma*: (*dsum* (*A* × *B*) (*C* × *D*)) <= (*usum A C*) × (*usum B D*)
by *blast*

lemmas *dsum-subset-Sigma* = *subset-trans* [*OF dsum-mono dsum-Sigma*]

lemma *Domain-dprod* [*simp*]: *Domain* (*dprod r s*) = *uprod* (*Domain r*) (*Domain s*)
by *auto*

lemma *Domain-dsum* [*simp*]: *Domain* (*dsum r s*) = *usum* (*Domain r*) (*Domain s*)
by *auto*

hides popular names

hide-type (**open**) *node item*
hide-const (**open**) *Push Node Atom Leaf Numb Lim Split Case*

ML-file ~~/*src*/*HOL*/*Tools*/*Old-Datatype*/*old-datatype.ML*
ML-file ~~/*src*/*HOL*/*Tools*/*inductive-realizer.ML*

end

3 Bijections between natural numbers and other types

theory *Nat-Bijection*
imports *Main*
begin

3.1 Type *nat* × *nat*

Triangle numbers: 0, 1, 3, 6, 10, 15, ...

definition *triangle* :: *nat* ⇒ *nat*
where *triangle n* = (*n* ∗ *Suc n*) *div 2*

lemma *triangle-0* [*simp*]: *triangle 0* = *0*
unfolding *triangle-def* **by** *simp*

lemma *triangle-Suc* [*simp*]: *triangle* (*Suc n*) = *triangle n* + *Suc n*
unfolding *triangle-def* **by** *simp*

definition *prod-encode* :: *nat* × *nat* ⇒ *nat*
where *prod-encode* = (λ(*m*, *n*). *triangle* (*m* + *n*) + *m*)

In this auxiliary function, *triangle k* + *m* is an invariant.

fun *prod-decode-aux* :: *nat* ⇒ *nat* ⇒ *nat* × *nat*
where
prod-decode-aux k m =
(*if m* ≤ *k then* (*m*, *k* − *m*) *else prod-decode-aux* (*Suc k*) (*m* − *Suc k*))

declare *prod-decode-aux.simps* [*simp del*]

definition *prod-decode* :: *nat* ⇒ *nat* × *nat*
where *prod-decode* = *prod-decode-aux 0*

lemma *prod-encode-prod-decode-aux*:

prod-encode (*prod-decode-aux k m*) = *triangle k* + *m*
apply (*induct k m rule*: *prod-decode-aux.induct*)
apply (*subst prod-decode-aux.simps*)
apply (*simp add*: *prod-encode-def*)
done

lemma *prod-decode-inverse* [*simp*]: *prod-encode* (*prod-decode n*) = *n*
unfolding *prod-decode-def* **by** (*simp add*: *prod-encode-prod-decode-aux*)

lemma *prod-decode-triangle-add*: *prod-decode* (*triangle k* + *m*) = *prod-decode-aux k m*
apply (*induct k arbitrary*: *m*)
apply (*simp add*: *prod-decode-def*)
apply (*simp only*: *triangle-Suc add.assoc*)
apply (*subst prod-decode-aux.simps*, *simp*)
done

lemma *prod-encode-inverse* [*simp*]: *prod-decode* (*prod-encode x*) = *x*
unfolding *prod-encode-def*
apply (*induct x*)
apply (*simp add*: *prod-decode-triangle-add*)
apply (*subst prod-decode-aux.simps*, *simp*)
done

lemma *inj-prod-encode*: *inj-on prod-encode A*
by (*rule inj-on-inverseI*, *rule prod-encode-inverse*)

lemma *inj-prod-decode*: *inj-on prod-decode A*
by (*rule inj-on-inverseI*, *rule prod-decode-inverse*)

lemma *surj-prod-encode*: *surj prod-encode*
by (*rule surjI*, *rule prod-decode-inverse*)

lemma *surj-prod-decode*: *surj prod-decode*
by (*rule surjI*, *rule prod-encode-inverse*)

lemma *bij-prod-encode*: *bij prod-encode*
by (*rule bijI* [*OF inj-prod-encode surj-prod-encode*])

lemma *bij-prod-decode*: *bij prod-decode*
by (*rule bijI* [*OF inj-prod-decode surj-prod-decode*])

lemma *prod-encode-eq*: *prod-encode x* = *prod-encode y* ⟷ *x* = *y*
by (*rule inj-prod-encode* [*THEN inj-eq*])

lemma *prod-decode-eq*: *prod-decode x* = *prod-decode y* ⟷ *x* = *y*
by (*rule inj-prod-decode* [*THEN inj-eq*])

Ordering properties

lemma *le-prod-encode-1*: $a \leq$ *prod-encode* (a, b)
unfolding *prod-encode-def* **by** *simp*

lemma *le-prod-encode-2*: $b \leq$ *prod-encode* (a, b)
unfolding *prod-encode-def* **by** (*induct b*, *simp-all*)

3.2 Type *nat* + *nat*

definition *sum-encode* :: *nat* + *nat* ⇒ *nat*
where
sum-encode x = (*case x of Inl a* ⇒ *2* ∗ *a* | *Inr b* ⇒ *Suc* (*2* ∗ *b*))

definition *sum-decode* :: *nat* ⇒ *nat* + *nat*
where
sum-decode n = (*if even n then Inl* (*n div 2*) *else Inr* (*n div 2*))

lemma *sum-encode-inverse* [*simp*]: *sum-decode* (*sum-encode x*) = *x*
unfolding *sum-decode-def sum-encode-def*
by (*induct x*) *simp-all*

lemma *sum-decode-inverse* [*simp*]: *sum-encode* (*sum-decode n*) = *n*
by (*simp add*: *even-two-times-div-two sum-decode-def sum-encode-def*)

lemma *inj-sum-encode*: *inj-on sum-encode A*
by (*rule inj-on-inverseI*, *rule sum-encode-inverse*)

lemma *inj-sum-decode*: *inj-on sum-decode A*
by (*rule inj-on-inverseI*, *rule sum-decode-inverse*)

lemma *surj-sum-encode*: *surj sum-encode*
by (*rule surjI*, *rule sum-decode-inverse*)

lemma *surj-sum-decode*: *surj sum-decode*
by (*rule surjI*, *rule sum-encode-inverse*)

lemma *bij-sum-encode*: *bij sum-encode*
by (*rule bijI* [*OF inj-sum-encode surj-sum-encode*])

lemma *bij-sum-decode*: *bij sum-decode*
by (*rule bijI* [*OF inj-sum-decode surj-sum-decode*])

lemma *sum-encode-eq*: *sum-encode x* = *sum-encode y* ⟷ *x* = *y*
by (*rule inj-sum-encode* [*THEN inj-eq*])

lemma *sum-decode-eq*: *sum-decode x* = *sum-decode y* ⟷ *x* = *y*
by (*rule inj-sum-decode* [*THEN inj-eq*])

3.3 Type *int*

definition *int-encode* :: *int* ⇒ *nat*

where
int-encode i = sum-encode (if 0 ≤ i then Inl (nat i) else Inr (nat (− i − 1)))

definition *int-decode* :: *nat ⇒ int*
where
int-decode n = (case sum-decode n of Inl a ⇒ int a | Inr b ⇒ − int b − 1)

lemma *int-encode-inverse* [*simp*]: *int-decode (int-encode x) = x*
unfolding *int-decode-def int-encode-def* **by** *simp*

lemma *int-decode-inverse* [*simp*]: *int-encode (int-decode n) = n*
unfolding *int-decode-def int-encode-def* **using** *sum-decode-inverse* [*of n*]
by (*cases sum-decode n*, *simp-all*)

lemma *inj-int-encode*: *inj-on int-encode A*
by (*rule inj-on-inverseI*, *rule int-encode-inverse*)

lemma *inj-int-decode*: *inj-on int-decode A*
by (*rule inj-on-inverseI*, *rule int-decode-inverse*)

lemma *surj-int-encode*: *surj int-encode*
by (*rule surjI*, *rule int-decode-inverse*)

lemma *surj-int-decode*: *surj int-decode*
by (*rule surjI*, *rule int-encode-inverse*)

lemma *bij-int-encode*: *bij int-encode*
by (*rule bijI* [*OF inj-int-encode surj-int-encode*])

lemma *bij-int-decode*: *bij int-decode*
by (*rule bijI* [*OF inj-int-decode surj-int-decode*])

lemma *int-encode-eq*: *int-encode x = int-encode y ⟷ x = y*
by (*rule inj-int-encode* [*THEN inj-eq*])

lemma *int-decode-eq*: *int-decode x = int-decode y ⟷ x = y*
by (*rule inj-int-decode* [*THEN inj-eq*])

3.4 Type *nat list*

fun *list-encode* :: *nat list ⇒ nat*
where
list-encode [] = 0
| *list-encode (x # xs) = Suc (prod-encode (x, list-encode xs))*

function *list-decode* :: *nat ⇒ nat list*
where
list-decode 0 = []
| *list-decode (Suc n) = (case prod-decode n of (x, y) ⇒ x # list-decode y)*

by *pat-completeness auto*

termination *list-decode*
apply (*relation measure id*, *simp-all*)
apply (*drule arg-cong* [**where** *f*=*prod-encode*])
apply (*drule sym*)
apply (*simp add*: *le-imp-less-Suc le-prod-encode-2*)
done

lemma *list-encode-inverse* [*simp*]: *list-decode* (*list-encode x*) = *x*
by (*induct x rule*: *list-encode.induct*) *simp-all*

lemma *list-decode-inverse* [*simp*]: *list-encode* (*list-decode n*) = *n*
apply (*induct n rule*: *list-decode.induct*, *simp*)
apply (*simp split*: *prod.split*)
apply (*simp add*: *prod-decode-eq* [*symmetric*])
done

lemma *inj-list-encode*: *inj-on list-encode A*
by (*rule inj-on-inverseI*, *rule list-encode-inverse*)

lemma *inj-list-decode*: *inj-on list-decode A*
by (*rule inj-on-inverseI*, *rule list-decode-inverse*)

lemma *surj-list-encode*: *surj list-encode*
by (*rule surjI*, *rule list-decode-inverse*)

lemma *surj-list-decode*: *surj list-decode*
by (*rule surjI*, *rule list-encode-inverse*)

lemma *bij-list-encode*: *bij list-encode*
by (*rule bijI* [*OF inj-list-encode surj-list-encode*])

lemma *bij-list-decode*: *bij list-decode*
by (*rule bijI* [*OF inj-list-decode surj-list-decode*])

lemma *list-encode-eq*: *list-encode x* = *list-encode y* ⟷ *x* = *y*
by (*rule inj-list-encode* [*THEN inj-eq*])

lemma *list-decode-eq*: *list-decode x* = *list-decode y* ⟷ *x* = *y*
by (*rule inj-list-decode* [*THEN inj-eq*])

3.5 Finite sets of naturals

3.5.1 Preliminaries

lemma *finite-vimage-Suc-iff*: *finite* (*Suc* −‘ *F*) ⟷ *finite F*
apply (*safe intro*!: *finite-vimageI inj-Suc*)
apply (*rule finite-subset* [**where** *B*=*insert 0* (*Suc* ‘ *Suc* −‘ *F*)])
apply (*rule subsetI*, *case-tac x*, *simp*, *simp*)

apply (*rule finite-insert* [*THEN iffD2*])
apply (*erule finite-imageI*)
done

lemma *vimage-Suc-insert-0*: *Suc −‘ insert 0 A = Suc −‘ A*
by *auto*

lemma *vimage-Suc-insert-Suc*:
Suc −‘ insert (Suc n) A = insert n (Suc −‘ A)
by *auto*

lemma *div2-even-ext-nat*:
fixes *x y* :: *nat*
assumes *x div 2 = y div 2*
and *even x ⟷ even y*
shows *x = y*
proof −
from ‹*even x ⟷ even y*› **have** *x mod 2 = y mod 2*
by (*simp only*: *even-iff-mod-2-eq-zero*) *auto*
with *assms* **have** *x div 2 * 2 + x mod 2 = y div 2 * 2 + y mod 2*
by *simp*
then show *?thesis*
by *simp*
qed

3.5.2 From sets to naturals

definition *set-encode* :: *nat set ⇒ nat*
where *set-encode = setsum* (*op ^ 2*)

lemma *set-encode-empty* [*simp*]: *set-encode {} = 0*
by (*simp add*: *set-encode-def*)

lemma *set-encode-inf*: ~ *finite A ⟹ set-encode A = 0*
by (*simp add*: *set-encode-def*)

lemma *set-encode-insert* [*simp*]:
⟦*finite A*; *n ∉ A*⟧ *⟹ set-encode* (*insert n A*) = *2^n + set-encode A*
by (*simp add*: *set-encode-def*)

lemma *even-set-encode-iff*: *finite A ⟹ even* (*set-encode A*) *⟷ 0 ∉ A*
unfolding *set-encode-def* **by** (*induct set*: *finite*, *auto*)

lemma *set-encode-vimage-Suc*: *set-encode* (*Suc −‘ A*) = *set-encode A div 2*
apply (*cases finite A*)
apply (*erule finite-induct*, *simp*)
apply (*case-tac x*)
apply (*simp add*: *even-set-encode-iff vimage-Suc-insert-0*)
apply (*simp add*: *finite-vimageI add.commute vimage-Suc-insert-Suc*)

apply (*simp add*: *set-encode-def finite-vimage-Suc-iff*)
done

lemmas *set-encode-div-2* = *set-encode-vimage-Suc* [*symmetric*]

3.5.3 From naturals to sets

definition *set-decode* :: *nat* ⇒ *nat set*
  where *set-decode x* = {*n*. *odd* (*x div 2* ^ *n*)}

lemma *set-decode-0* [*simp*]: *0* ∈ *set-decode x* ⟷ *odd x*
by (*simp add*: *set-decode-def*)

lemma *set-decode-Suc* [*simp*]:
  Suc n ∈ *set-decode x* ⟷ *n* ∈ *set-decode* (*x div 2*)
by (*simp add*: *set-decode-def div-mult2-eq*)

lemma *set-decode-zero* [*simp*]: *set-decode 0* = {}
by (*simp add*: *set-decode-def*)

lemma *set-decode-div-2*: *set-decode* (*x div 2*) = *Suc* −‘ *set-decode x*
by *auto*

lemma *set-decode-plus-power-2*:
  n ∉ *set-decode z* ⟹ *set-decode* (*2* ^ *n* + *z*) = *insert n* (*set-decode z*)
proof (*induct n arbitrary*: *z*)
  case *0* **show** *?case*
  proof (*rule set-eqI*)
    fix *q* **show** *q* ∈ *set-decode* (*2* ^ *0* + *z*) ⟷ *q* ∈ *insert 0* (*set-decode z*)
      by (*induct q*) (*insert 0*, *simp-all*)
  qed
next
  case (*Suc n*) **show** *?case*
  proof (*rule set-eqI*)
    fix *q* **show** *q* ∈ *set-decode* (*2* ^ *Suc n* + *z*) ⟷ *q* ∈ *insert* (*Suc n*) (*set-decode z*)
      by (*induct q*) (*insert Suc*, *simp-all*)
  qed
qed

lemma *finite-set-decode* [*simp*]: *finite* (*set-decode n*)
apply (*induct n rule*: *nat-less-induct*)
apply (*case-tac n* = *0*, *simp*)
apply (*drule-tac x*=*n div 2* **in** *spec*, *simp*)
apply (*simp add*: *set-decode-div-2*)
apply (*simp add*: *finite-vimage-Suc-iff*)
done

3.5.4 Proof of isomorphism

lemma *set-decode-inverse* [*simp*]: *set-encode* (*set-decode n*) = *n*
apply (*induct n rule*: *nat-less-induct*)
apply (*case-tac n* = *0*, *simp*)
apply (*drule-tac x*=*n div 2* **in** *spec*, *simp*)
apply (*simp add*: *set-decode-div-2 set-encode-vimage-Suc*)
apply (*erule div2-even-ext-nat*)
apply (*simp add*: *even-set-encode-iff*)
done

lemma *set-encode-inverse* [*simp*]: *finite A* ⟹ *set-decode* (*set-encode A*) = *A*
apply (*erule finite-induct*, *simp-all*)
apply (*simp add*: *set-decode-plus-power-2*)
done

lemma *inj-on-set-encode*: *inj-on set-encode* (*Collect finite*)
by (*rule inj-on-inverseI* [**where** *g*=*set-decode*], *simp*)

lemma *set-encode-eq*:
⟦*finite A*; *finite B*⟧ ⟹ *set-encode A* = *set-encode B* ⟷ *A* = *B*
by (*rule iffI*, *simp add*: *inj-onD* [*OF inj-on-set-encode*], *simp*)

lemma *subset-decode-imp-le*:
assumes *set-decode m* ⊆ *set-decode n*
shows *m* ≤ *n*
proof −
have *n* = *m* + *set-encode* (*set-decode n* − *set-decode m*)
proof −
obtain *A B* **where** *m* = *set-encode A finite A*
n = *set-encode B finite B*
by (*metis finite-set-decode set-decode-inverse*)
thus *?thesis* **using** *assms*
apply *auto*
apply (*simp add*: *set-encode-def add.commute setsum.subset-diff*)
done
qed
thus *?thesis*
by (*metis le-add1*)
qed

end

4 Encoding (almost) everything into natural numbers

theory *Countable*
imports *Old-Datatype* ~~/*src*/*HOL*/*Rat Nat-Bijection*
begin

4.1 The class of countable types

class *countable* =
  assumes *ex-inj*: $\exists$ *to-nat* :: *'a* $\Rightarrow$ *nat*. *inj to-nat*

lemma *countable-classI*:
  fixes *f* :: *'a* $\Rightarrow$ *nat*
  assumes $\bigwedge x\ y.\ f\ x = f\ y \Longrightarrow x = y$
  shows *OFCLASS*(*'a*, *countable-class*)
proof (*intro-classes*, *rule exI*)
  show *inj f*
    by (*rule injI* [*OF assms*]) *assumption*
qed

4.2 Conversion functions

definition *to-nat* :: *'a*::*countable* $\Rightarrow$ *nat* **where**
  to-nat = (*SOME f*. *inj f*)

definition *from-nat* :: *nat* $\Rightarrow$ *'a*::*countable* **where**
  from-nat = *inv* (*to-nat* :: *'a* $\Rightarrow$ *nat*)

lemma *inj-to-nat* [*simp*]: *inj to-nat*
  by (*rule exE-some* [*OF ex-inj*]) (*simp add*: *to-nat-def*)

lemma *inj-on-to-nat*[*simp*, *intro*]: *inj-on to-nat S*
  using *inj-to-nat* **by** (*auto simp*: *inj-on-def*)

lemma *surj-from-nat* [*simp*]: *surj from-nat*
  unfolding *from-nat-def* **by** (*simp add*: *inj-imp-surj-inv*)

lemma *to-nat-split* [*simp*]: *to-nat x* = *to-nat y* $\longleftrightarrow$ *x* = *y*
  using *injD* [*OF inj-to-nat*] **by** *auto*

lemma *from-nat-to-nat* [*simp*]:
  from-nat (*to-nat x*) = *x*
  by (*simp add*: *from-nat-def*)

4.3 Finite types are countable

subclass (**in** *finite*) *countable*
proof
  have *finite* (*UNIV*::*'a set*) **by** (*rule finite-UNIV*)
  with *finite-conv-nat-seg-image* [*of UNIV*::*'a set*]
  obtain *n* **and** *f* :: *nat* $\Rightarrow$ *'a*
    where *UNIV* = *f* ‘ {*i*. *i* < *n*} **by** *auto*
  then have *surj f* **unfolding** *surj-def* **by** *auto*
  then have *inj* (*inv f*) **by** (*rule surj-imp-inj-inv*)
  then show $\exists$ *to-nat* :: *'a* $\Rightarrow$ *nat*. *inj to-nat* **by** (*rule exI*[*of inj*])
qed

4.4 Automatically proving countability of old-style datatypes

context
begin

qualified inductive *finite-item* :: *'a Old-Datatype.item* ⇒ *bool* **where**
  undefined: *finite-item undefined*
| *In0*: *finite-item x* ⟹ *finite-item* (*Old-Datatype.In0 x*)
| *In1*: *finite-item x* ⟹ *finite-item* (*Old-Datatype.In1 x*)
| *Leaf*: *finite-item* (*Old-Datatype.Leaf a*)
| *Scons*: ⟦*finite-item x*; *finite-item y*⟧ ⟹ *finite-item* (*Old-Datatype.Scons x y*)

qualified function *nth-item* :: *nat* ⇒ (*'a*::*countable*) *Old-Datatype.item*
where
  nth-item 0 = *undefined*
| *nth-item* (*Suc n*) =
  (*case sum-decode n of*
    Inl i ⇒
    (*case sum-decode i of*
      Inl j ⇒ *Old-Datatype.In0* (*nth-item j*)
    | *Inr j* ⇒ *Old-Datatype.In1* (*nth-item j*))
  | *Inr i* ⇒
    (*case sum-decode i of*
      Inl j ⇒ *Old-Datatype.Leaf* (*from-nat j*)
    | *Inr j* ⇒
      (*case prod-decode j of*
        (*a*, *b*) ⇒ *Old-Datatype.Scons* (*nth-item a*) (*nth-item b*))))
by *pat-completeness auto*

lemma *le-sum-encode-Inl*: *x* ≤ *y* ⟹ *x* ≤ *sum-encode* (*Inl y*)
unfolding *sum-encode-def* **by** *simp*

lemma *le-sum-encode-Inr*: *x* ≤ *y* ⟹ *x* ≤ *sum-encode* (*Inr y*)
unfolding *sum-encode-def* **by** *simp*

qualified termination
by (*relation measure id*)
  (*auto simp add*: *sum-encode-eq* [*symmetric*] *prod-encode-eq* [*symmetric*]
    le-imp-less-Suc le-sum-encode-Inl le-sum-encode-Inr
    le-prod-encode-1 le-prod-encode-2)

lemma *nth-item-covers*: *finite-item x* ⟹ ∃ *n*. *nth-item n* = *x*
proof (*induct set*: *finite-item*)
  case *undefined*
  have *nth-item 0* = *undefined* **by** *simp*
  thus *?case* **..**
next
  case (*In0 x*)
  then obtain *n* **where** *nth-item n* = *x* **by** *fast*
 hence *nth-item* (*Suc* (*sum-encode* (*Inl* (*sum-encode* (*Inl n*))))) = *Old-Datatype.In0*

x **by** *simp*
thus *?case* **..**
next
case (*In1 x*)
then obtain *n* **where** *nth-item n = x* **by** *fast*
hence *nth-item* (*Suc* (*sum-encode* (*Inl* (*sum-encode* (*Inr n*))))) = *Old-Datatype.In1 x* **by** *simp*
thus *?case* **..**
next
case (*Leaf a*)
have *nth-item* (*Suc* (*sum-encode* (*Inr* (*sum-encode* (*Inl* (*to-nat a*)))))) = *Old-Datatype.Leaf a*
by *simp*
thus *?case* **..**
next
case (*Scons x y*)
then obtain *i j* **where** *nth-item i = x* **and** *nth-item j = y* **by** *fast*
hence *nth-item*
(*Suc* (*sum-encode* (*Inr* (*sum-encode* (*Inr* (*prod-encode* (*i*, *j*)))))))) = *Old-Datatype.Scons x y*
by *simp*
thus *?case* **..**
qed

theorem *countable-datatype*:
fixes *Rep* :: *'b* ⇒ (*'a*::*countable*) *Old-Datatype.item*
fixes *Abs* :: (*'a*::*countable*) *Old-Datatype.item* ⇒ *'b*
fixes *rep-set* :: (*'a*::*countable*) *Old-Datatype.item* ⇒ *bool*
assumes *type*: *type-definition Rep Abs* (*Collect rep-set*)
assumes *finite-item*: ⋀*x*. *rep-set x* ⟹ *finite-item x*
shows *OFCLASS*(*'b*, *countable-class*)
proof
def *f* ≡ λ*y*. *LEAST n*. *nth-item n = Rep y*
{
fix *y* :: *'b*
have *rep-set* (*Rep y*)
using *type-definition.Rep* [*OF type*] **by** *simp*
hence *finite-item* (*Rep y*)
by (*rule finite-item*)
hence ∃ *n*. *nth-item n = Rep y*
by (*rule nth-item-covers*)
hence *nth-item* (*f y*) = *Rep y*
unfolding *f-def* **by** (*rule LeastI-ex*)
hence *Abs* (*nth-item* (*f y*)) = *y*
using *type-definition.Rep-inverse* [*OF type*] **by** *simp*
}
hence *inj f*
by (*rule inj-on-inverseI*)
thus ∃ *f*::*'b* ⇒ *nat*. *inj f*

by − (*rule exI*)
qed

ML ‹
fun old-countable-datatype-tac ctxt =
SUBGOAL (fn (goal, -) =>
let
val ty-name =
(case goal of
(- $ Const (@{const-name Pure.type}, Type (@{type-name itself}, [Type (n, -)])))) => n
| - => raise Match)
val typedef-info = hd (Typedef.get-info ctxt ty-name)
val typedef-thm = #type-definition (snd typedef-info)
val pred-name =
(case HOLogic.dest-Trueprop (Thm.concl-of typedef-thm) of
(- $ - $ - $ (- $ Const (n, -))) => n
| - => raise Match)
val induct-info = Inductive.the-inductive ctxt pred-name
val pred-names = #names (fst induct-info)
val induct-thms = #inducts (snd induct-info)
val alist = pred-names ~~ induct-thms
val induct-thm = the (AList.lookup (op =) alist pred-name)
val vars = rev (Term.add-vars (Thm.prop-of induct-thm) [])
val insts = vars |> map (fn (-, T) => try (Thm.cterm-of ctxt)
(Const (@{const-name Countable.finite-item}, T)))
val induct-thm' = Thm.instantiate' [] insts induct-thm
val rules = @{thms finite-item.intros}
in
SOLVED' (fn i => EVERY
[resolve-tac ctxt @{thms countable-datatype} i,
resolve-tac ctxt [typedef-thm] i,
eresolve-tac ctxt [induct-thm'] i,
REPEAT (resolve-tac ctxt rules i ORELSE assume-tac ctxt i)]) 1
end)
›

end

4.5 Automatically proving countability of datatypes

ML-file *bnf-lfp-countable.ML*

ML ‹
fun countable-datatype-tac ctxt st =
(case try (fn () => HEADGOAL (old-countable-datatype-tac ctxt) st) () of
SOME res => res
| NONE => BNF-LFP-Countable.countable-datatype-tac ctxt st);

```
(* compatibility *)
fun countable-tac ctxt =
  SELECT-GOAL (countable-datatype-tac ctxt);
›

method-setup countable-datatype = ‹
  Scan.succeed (SIMPLE-METHOD o countable-datatype-tac)
› prove countable class instances for datatypes
```

4.6 More Countable types

Naturals

```
instance nat :: countable
  by (rule countable-classI [of id]) simp
```

Pairs

```
instance prod :: (countable, countable) countable
  by (rule countable-classI [of λ(x, y). prod-encode (to-nat x, to-nat y)])
    (auto simp add: prod-encode-eq)
```

Sums

```
instance sum :: (countable, countable) countable
  by (rule countable-classI [of (λx. case x of Inl a ⇒ to-nat (False, to-nat a)
                                  | Inr b ⇒ to-nat (True, to-nat b))])
    (simp split: sum.split-asm)
```

Integers

```
instance int :: countable
  by (rule countable-classI [of int-encode]) (simp add: int-encode-eq)
```

Options

```
instance option :: (countable) countable
  by countable-datatype
```

Lists

```
instance list :: (countable) countable
  by countable-datatype
```

String literals

```
instance String.literal :: countable
 by (rule countable-classI [of to-nat ∘ String.explode]) (auto simp add: explode-inject)
```

Functions

```
instance fun :: (finite, countable) countable
proof
  obtain xs :: 'a list where xs: set xs = UNIV
    using finite-list [OF finite-UNIV] ..
```

show ∃ *to-nat*::('*a* ⇒ '*b*) ⇒ *nat*. *inj to-nat*
proof
show *inj* (λ*f*. *to-nat* (*map f xs*))
by (*rule injI*, *simp add*: *xs fun-eq-iff*)
qed
qed

Typereps

instance *typerep* :: *countable*
by *countable-datatype*

4.7 The rationals are countably infinite

definition *nat-to-rat-surj* :: *nat* ⇒ *rat* **where**
nat-to-rat-surj n = (*let* (*a*, *b*) = *prod-decode n in Fract* (*int-decode a*) (*int-decode b*))

lemma *surj-nat-to-rat-surj*: *surj nat-to-rat-surj*
unfolding *surj-def*
proof
fix *r*::*rat*
show ∃ *n*. *r* = *nat-to-rat-surj n*
proof (*cases r*)
fix *i j* **assume** [*simp*]: *r* = *Fract i j* **and** *j* > *0*
have *r* = (*let m* = *int-encode i*; *n* = *int-encode j in nat-to-rat-surj* (*prod-encode* (*m*, *n*)))
by (*simp add*: *Let-def nat-to-rat-surj-def*)
thus ∃ *n*. *r* = *nat-to-rat-surj n* **by**(*auto simp*: *Let-def*)
qed
qed

lemma *Rats-eq-range-nat-to-rat-surj*: ℚ = *range nat-to-rat-surj*
by (*simp add*: *Rats-def surj-nat-to-rat-surj*)

context *field-char-0*
begin

lemma *Rats-eq-range-of-rat-o-nat-to-rat-surj*:
ℚ = *range* (*of-rat* ∘ *nat-to-rat-surj*)
using *surj-nat-to-rat-surj*
by (*auto simp*: *Rats-def image-def surj-def*) (*blast intro*: *arg-cong*[**where** *f* = *of-rat*])

lemma *surj-of-rat-nat-to-rat-surj*:
r ∈ ℚ ⟹ ∃ *n*. *r* = *of-rat* (*nat-to-rat-surj n*)
by (*simp add*: *Rats-eq-range-of-rat-o-nat-to-rat-surj image-def*)

end

instance *rat* :: *countable*
proof
  show ∃ *to-nat*::*rat* ⇒ *nat*. *inj to-nat*
  proof
    have *surj nat-to-rat-surj*
      by (*rule surj-nat-to-rat-surj*)
    then show *inj* (*inv nat-to-rat-surj*)
      by (*rule surj-imp-inj-inv*)
  qed
qed

end

5 Infinite Sets and Related Concepts

theory *Infinite-Set*
imports *Main*
begin

The set of natural numbers is infinite.

lemma *infinite-nat-iff-unbounded-le*: *infinite* (*S*::*nat set*) ⟷ (∀ *m*. ∃ *n*≥*m*. *n* ∈ *S*)
  using *frequently-cofinite*[*of* λ*x*. *x* ∈ *S*]
  by (*simp add*: *cofinite-eq-sequentially frequently-def eventually-sequentially*)

lemma *infinite-nat-iff-unbounded*: *infinite* (*S*::*nat set*) ⟷ (∀ *m*. ∃ *n*>*m*. *n* ∈ *S*)
  using *frequently-cofinite*[*of* λ*x*. *x* ∈ *S*]
  by (*simp add*: *cofinite-eq-sequentially frequently-def eventually-at-top-dense*)

lemma *finite-nat-iff-bounded*: *finite* (*S*::*nat set*) ⟷ (∃ *k*. *S* ⊆ {..<*k*})
  using *infinite-nat-iff-unbounded-le*[*of S*] **by** (*simp add*: *subset-eq*) (*metis not-le*)

lemma *finite-nat-iff-bounded-le*: *finite* (*S*::*nat set*) ⟷ (∃ *k*. *S* ⊆ {.. *k*})
  using *infinite-nat-iff-unbounded*[*of S*] **by** (*simp add*: *subset-eq*) (*metis not-le*)

lemma *finite-nat-bounded*: *finite* (*S*::*nat set*) ⟹ ∃ *k*. *S* ⊆ {..<*k*}
  by (*simp add*: *finite-nat-iff-bounded*)

For a set of natural numbers to be infinite, it is enough to know that for any number larger than some k, there is some larger number that is an element of the set.

lemma *unbounded-k-infinite*: ∀ *m*>*k*. ∃ *n*>*m*. *n* ∈ *S* ⟹ *infinite* (*S*::*nat set*)
apply (*clarsimp simp add*: *finite-nat-set-iff-bounded*)
apply (*drule-tac x*=*Suc* (*max m k*) **in** *spec*)
using *less-Suc-eq* **by** *fastforce*

lemma *nat-not-finite*: *finite* (*UNIV*::*nat set*) ⟹ *R*
  by *simp*

lemma *range-inj-infinite*:
inj (*f*::*nat* ⇒ *'a*) ⟹ *infinite* (*range f*)
proof
assume *finite* (*range f*) **and** *inj f*
then have *finite* (*UNIV*::*nat set*)
by (*rule finite-imageD*)
then show *False* **by** *simp*
qed

The set of integers is also infinite.

lemma *infinite-int-iff-infinite-nat-abs*: *infinite* (*S*::*int set*) ⟷ *infinite* ((*nat* ∘ *abs*) ` *S*)
by (*auto simp*: *transfer-nat-int-set-relations o-def image-comp dest*: *finite-image-absD*)

proposition *infinite-int-iff-unbounded-le*: *infinite* (*S*::*int set*) ⟷ (∀ *m*. ∃ *n*. |*n*| ≥ *m* ∧ *n* ∈ *S*)
apply (*simp add*: *infinite-int-iff-infinite-nat-abs infinite-nat-iff-unbounded-le o-def image-def*)
apply (*metis abs-ge-zero nat-le-eq-zle le-nat-iff*)
done

proposition *infinite-int-iff-unbounded*: *infinite* (*S*::*int set*) ⟷ (∀ *m*. ∃ *n*. |*n*| > *m* ∧ *n* ∈ *S*)
apply (*simp add*: *infinite-int-iff-infinite-nat-abs infinite-nat-iff-unbounded o-def image-def*)
apply (*metis* (*full-types*) *nat-le-iff nat-mono not-le*)
done

proposition *finite-int-iff-bounded*: *finite* (*S*::*int set*) ⟷ (∃ *k*. *abs* ` *S* ⊆ {..<*k*})
using *infinite-int-iff-unbounded-le*[*of S*] **by** (*simp add*: *subset-eq*) (*metis not-le*)

proposition *finite-int-iff-bounded-le*: *finite* (*S*::*int set*) ⟷ (∃ *k*. *abs* ` *S* ⊆ {.. *k*})
using *infinite-int-iff-unbounded*[*of S*] **by** (*simp add*: *subset-eq*) (*metis not-le*)

5.1 Infinitely Many and Almost All

We often need to reason about the existence of infinitely many (resp., all but finitely many) objects satisfying some predicate, so we introduce corresponding binders and their proof rules.

lemma *not-INFM* [*simp*]: ¬ (*INFM x*. *P x*) ⟷ (*MOST x*. ¬ *P x*) **by** (*fact not-frequently*)
lemma *not-MOST* [*simp*]: ¬ (*MOST x*. *P x*) ⟷ (*INFM x*. ¬ *P x*) **by** (*fact not-eventually*)

lemma *INFM-const* [*simp*]: (*INFM x*::*'a*. *P*) ⟷ *P* ∧ *infinite* (*UNIV*::*'a set*)
by (*simp add*: *frequently-const-iff*)

lemma *MOST-const* [*simp*]: $(MOST\ x{::}'a.\ P) \longleftrightarrow P \vee finite\ (UNIV{::}'a\ set)$
by (*simp add*: *eventually-const-iff*)

lemma *INFM-imp-distrib*: $(INFM\ x.\ P\ x \longrightarrow Q\ x) \longleftrightarrow ((MOST\ x.\ P\ x) \longrightarrow (INFM\ x.\ Q\ x))$
by (*simp only*: *imp-conv-disj frequently-disj-iff not-eventually*)

lemma *MOST-imp-iff*: $MOST\ x.\ P\ x \Longrightarrow (MOST\ x.\ P\ x \longrightarrow Q\ x) \longleftrightarrow (MOST\ x.\ Q\ x)$
by (*auto intro*: *eventually-rev-mp eventually-mono*)

lemma *INFM-conjI*: $INFM\ x.\ P\ x \Longrightarrow MOST\ x.\ Q\ x \Longrightarrow INFM\ x.\ P\ x \wedge Q\ x$
by (*rule frequently-rev-mp*[*of P*]) (*auto elim*: *eventually-mono*)

Properties of quantifiers with injective functions.

lemma *INFM-inj*: $INFM\ x.\ P\ (f\ x) \Longrightarrow inj\ f \Longrightarrow INFM\ x.\ P\ x$
using *finite-vimageI*[*of* $\{x.\ P\ x\}$ *f*] **by** (*auto simp*: *frequently-cofinite*)

lemma *MOST-inj*: $MOST\ x.\ P\ x \Longrightarrow inj\ f \Longrightarrow MOST\ x.\ P\ (f\ x)$
using *finite-vimageI*[*of* $\{x.\ \neg\ P\ x\}$ *f*] **by** (*auto simp*: *eventually-cofinite*)

Properties of quantifiers with singletons.

lemma *not-INFM-eq* [*simp*]:
$\neg\ (INFM\ x.\ x = a)$
$\neg\ (INFM\ x.\ a = x)$
unfolding *frequently-cofinite* **by** *simp-all*

lemma *MOST-neq* [*simp*]:
$MOST\ x.\ x \neq a$
$MOST\ x.\ a \neq x$
unfolding *eventually-cofinite* **by** *simp-all*

lemma *INFM-neq* [*simp*]:
$(INFM\ x{::}'a.\ x \neq a) \longleftrightarrow infinite\ (UNIV{::}'a\ set)$
$(INFM\ x{::}'a.\ a \neq x) \longleftrightarrow infinite\ (UNIV{::}'a\ set)$
unfolding *frequently-cofinite* **by** *simp-all*

lemma *MOST-eq* [*simp*]:
$(MOST\ x{::}'a.\ x = a) \longleftrightarrow finite\ (UNIV{::}'a\ set)$
$(MOST\ x{::}'a.\ a = x) \longleftrightarrow finite\ (UNIV{::}'a\ set)$
unfolding *eventually-cofinite* **by** *simp-all*

lemma *MOST-eq-imp*:
$MOST\ x.\ x = a \longrightarrow P\ x$
$MOST\ x.\ a = x \longrightarrow P\ x$
unfolding *eventually-cofinite* **by** *simp-all*

Properties of quantifiers over the naturals.

lemma *MOST-nat*: $(\forall_\infty n.\ P\ (n{::}nat)) \longleftrightarrow (\exists\, m.\ \forall\, n{>}m.\ P\ n)$

by (*auto simp add*: *eventually-cofinite finite-nat-iff-bounded-le subset-eq not-le*[*symmetric*])

lemma *MOST-nat-le*: $(\forall_\infty n.\ P\ (n::nat)) \longleftrightarrow (\exists m.\ \forall n{\geq}m.\ P\ n)$
by (*auto simp add*: *eventually-cofinite finite-nat-iff-bounded subset-eq not-le*[*symmetric*])

lemma *INFM-nat*: $(\exists_\infty n.\ P\ (n::nat)) \longleftrightarrow (\forall m.\ \exists n{>}m.\ P\ n)$
by (*simp add*: *frequently-cofinite infinite-nat-iff-unbounded*)

lemma *INFM-nat-le*: $(\exists_\infty n.\ P\ (n::nat)) \longleftrightarrow (\forall m.\ \exists n{\geq}m.\ P\ n)$
by (*simp add*: *frequently-cofinite infinite-nat-iff-unbounded-le*)

lemma *MOST-INFM*: *infinite* $(UNIV::'a\ set) \Longrightarrow MOST\ x::'a.\ P\ x \Longrightarrow INFM\ x::'a.\ P\ x$
by (*simp add*: *eventually-frequently*)

lemma *MOST-Suc-iff*: $(MOST\ n.\ P\ (Suc\ n)) \longleftrightarrow (MOST\ n.\ P\ n)$
by (*simp add*: *cofinite-eq-sequentially eventually-sequentially-Suc*)

lemma
shows *MOST-SucI*: $MOST\ n.\ P\ n \Longrightarrow MOST\ n.\ P\ (Suc\ n)$
and *MOST-SucD*: $MOST\ n.\ P\ (Suc\ n) \Longrightarrow MOST\ n.\ P\ n$
by (*simp-all add*: *MOST-Suc-iff*)

lemma *MOST-ge-nat*: $MOST\ n::nat.\ m \leq n$
by (*simp add*: *cofinite-eq-sequentially eventually-ge-at-top*)

lemma *Inf-many-def*: *Inf-many* $P \longleftrightarrow$ *infinite* $\{x.\ P\ x\}$ **by** (*fact frequently-cofinite*)
lemma *Alm-all-def*: *Alm-all* $P \longleftrightarrow \neg\ (INFM\ x.\ \neg\ P\ x)$ **by** *simp*
lemma *INFM-iff-infinite*: $(INFM\ x.\ P\ x) \longleftrightarrow$ *infinite* $\{x.\ P\ x\}$ **by** (*fact frequently-cofinite*)
lemma *MOST-iff-cofinite*: $(MOST\ x.\ P\ x) \longleftrightarrow$ *finite* $\{x.\ \neg\ P\ x\}$ **by** (*fact eventually-cofinite*)
lemma *INFM-EX*: $(\exists_\infty x.\ P\ x) \Longrightarrow (\exists x.\ P\ x)$ **by** (*fact frequently-ex*)
lemma *ALL-MOST*: $\forall x.\ P\ x \Longrightarrow \forall_\infty x.\ P\ x$ **by** (*fact always-eventually*)
lemma *INFM-mono*: $\exists_\infty x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow \exists_\infty x.\ Q\ x$ **by** (*fact frequently-elim1*)
lemma *MOST-mono*: $\forall_\infty x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow \forall_\infty x.\ Q\ x$ **by** (*fact eventually-mono*)
lemma *INFM-disj-distrib*: $(\exists_\infty x.\ P\ x \vee Q\ x) \longleftrightarrow (\exists_\infty x.\ P\ x) \vee (\exists_\infty x.\ Q\ x)$ **by** (*fact frequently-disj-iff*)
lemma *MOST-rev-mp*: $\forall_\infty x.\ P\ x \Longrightarrow \forall_\infty x.\ P\ x \longrightarrow Q\ x \Longrightarrow \forall_\infty x.\ Q\ x$ **by** (*fact eventually-rev-mp*)
lemma *MOST-conj-distrib*: $(\forall_\infty x.\ P\ x \wedge Q\ x) \longleftrightarrow (\forall_\infty x.\ P\ x) \wedge (\forall_\infty x.\ Q\ x)$ **by** (*fact eventually-conj-iff*)
lemma *MOST-conjI*: $MOST\ x.\ P\ x \Longrightarrow MOST\ x.\ Q\ x \Longrightarrow MOST\ x.\ P\ x \wedge Q\ x$ **by** (*fact eventually-conj*)
lemma *INFM-finite-Bex-distrib*: *finite* $A \Longrightarrow (INFM\ y.\ \exists x{\in}A.\ P\ x\ y) \longleftrightarrow (\exists x{\in}A.\ INFM\ y.\ P\ x\ y)$ **by** (*fact frequently-bex-finite-distrib*)
lemma *MOST-finite-Ball-distrib*: *finite* $A \Longrightarrow (MOST\ y.\ \forall x{\in}A.\ P\ x\ y) \longleftrightarrow (\forall x{\in}A.\ MOST\ y.\ P\ x\ y)$ **by** (*fact eventually-ball-finite-distrib*)

lemma *INFM-E*: *INFM x. P x* $\Longrightarrow$ $(\bigwedge x.\ P\ x \Longrightarrow thesis) \Longrightarrow thesis$ **by** (*fact frequentlyE*)
lemma *MOST-I*: $(\bigwedge x.\ P\ x) \Longrightarrow$ *MOST x. P x* **by** (*rule eventuallyI*)
lemmas *MOST-iff-finiteNeg* = *MOST-iff-cofinite*

5.2 Enumeration of an Infinite Set

The set's element type must be wellordered (e.g. the natural numbers).

Could be generalized to $enumerate'\ S\ n = (SOME\ t.\ t \in s \wedge finite\ \{s \in S.\ s < t\} \wedge card\ \{s \in S.\ s < t\} = n)$.

primrec (**in** *wellorder*) *enumerate* :: *'a set* $\Rightarrow$ *nat* $\Rightarrow$ *'a*
where
  enumerate-0: *enumerate S 0* = $(LEAST\ n.\ n \in S)$
| *enumerate-Suc*: *enumerate S* $(Suc\ n)$ = *enumerate* $(S - \{LEAST\ n.\ n \in S\})\ n$

lemma *enumerate-Suc'*: *enumerate S* $(Suc\ n)$ = *enumerate* $(S - \{$*enumerate S 0*$\})\ n$
  by *simp*

lemma *enumerate-in-set*: *infinite S* $\Longrightarrow$ *enumerate S n* $\in$ *S*
  apply (*induct n arbitrary*: *S*)
   apply (*fastforce intro*: *LeastI dest*!: *infinite-imp-nonempty*)
  apply *simp*
  apply (*metis DiffE infinite-remove*)
  done

declare *enumerate-0* [*simp del*] *enumerate-Suc* [*simp del*]

lemma *enumerate-step*: *infinite S* $\Longrightarrow$ *enumerate S n* $<$ *enumerate S* $(Suc\ n)$
  apply (*induct n arbitrary*: *S*)
   apply (*rule order-le-neq-trans*)
    apply (*simp add*: *enumerate-0 Least-le enumerate-in-set*)
   apply (*simp only*: *enumerate-Suc'*)
   apply (*subgoal-tac enumerate* $(S - \{$*enumerate S 0*$\})\ 0 \in S - \{$*enumerate S 0*$\}$)
    apply (*blast intro*: *sym*)
   apply (*simp add*: *enumerate-in-set del*: *Diff-iff*)
  apply (*simp add*: *enumerate-Suc'*)
  done

lemma *enumerate-mono*: *m* $<$ *n* $\Longrightarrow$ *infinite S* $\Longrightarrow$ *enumerate S m* $<$ *enumerate S n*
  apply (*erule less-Suc-induct*)
  apply (*auto intro*: *enumerate-step*)
  done

lemma *le-enumerate*:

```
  assumes S: infinite S
  shows n ≤ enumerate S n
  using S
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then have n ≤ enumerate S n by simp
  also note enumerate-mono[of n Suc n, OF - ‹infinite S›]
  finally show ?case by simp
qed

lemma enumerate-Suc'':
  fixes S :: 'a::wellorder set
  assumes infinite S
  shows enumerate S (Suc n) = (LEAST s. s ∈ S ∧ enumerate S n < s)
  using assms
proof (induct n arbitrary: S)
  case 0
  then have ∀ s ∈ S. enumerate S 0 ≤ s
    by (auto simp: enumerate.simps intro: Least-le)
  then show ?case
    unfolding enumerate-Suc' enumerate-0[of S − {enumerate S 0}]
    by (intro arg-cong[where f = Least] ext) auto
next
  case (Suc n S)
  show ?case
    using enumerate-mono[OF zero-less-Suc ‹infinite S›, of n] ‹infinite S›
    apply (subst (1 2) enumerate-Suc')
    apply (subst Suc)
    using ‹infinite S›
    apply simp
    apply (intro arg-cong[where f = Least] ext)
    apply (auto simp: enumerate-Suc'[symmetric])
    done
qed

lemma enumerate-Ex:
  assumes S: infinite (S::nat set)
  shows s ∈ S ⟹ ∃ n. enumerate S n = s
proof (induct s rule: less-induct)
  case (less s)
  show ?case
  proof cases
    let ?y = Max {s'∈S. s' < s}
    assume ∃ y∈S. y < s
    then have y: ⋀x. ?y < x ⟷ (∀ s'∈S. s' < s ⟶ s' < x)
      by (subst Max-less-iff) auto
```

```
    then have y-in: ?y ∈ {s'∈S. s' < s}
      by (intro Max-in) auto
    with less.hyps[of ?y] obtain n where enumerate S n = ?y
      by auto
    with S have enumerate S (Suc n) = s
      by (auto simp: y less enumerate-Suc'' intro!: Least-equality)
    then show ?case by auto
  next
    assume *: ¬ (∃ y∈S. y < s)
    then have ∀ t∈S. s ≤ t by auto
    with ‹s ∈ S› show ?thesis
      by (auto intro!: exI[of - 0] Least-equality simp: enumerate-0)
  qed
qed

lemma bij-enumerate:
  fixes S :: nat set
  assumes S: infinite S
  shows bij-betw (enumerate S) UNIV S
proof −
  have ⋀n m. n ≠ m ⟹ enumerate S n ≠ enumerate S m
    using enumerate-mono[OF - ‹infinite S›] by (auto simp: neq-iff)
  then have inj (enumerate S)
    by (auto simp: inj-on-def)
  moreover have ∀ s ∈ S. ∃ i. enumerate S i = s
    using enumerate-Ex[OF S] by auto
  moreover note ‹infinite S›
  ultimately show ?thesis
    unfolding bij-betw-def by (auto intro: enumerate-in-set)
qed

end
```

6 Countable sets

```
theory Countable-Set
imports Countable Infinite-Set
begin
```

6.1 Predicate for countable sets

```
definition countable :: 'a set ⇒ bool where
  countable S ⟷ (∃ f::'a ⇒ nat. inj-on f S)

lemma countableE:
  assumes S: countable S obtains f :: 'a ⇒ nat where inj-on f S
  using S by (auto simp: countable-def)

lemma countableI: inj-on (f::'a ⇒ nat) S ⟹ countable S
```

by (*auto simp*: *countable-def*)

lemma *countableI'*: *inj-on* (*f*::*'a* ⇒ *'b*::*countable*) *S* ⟹ *countable S*
using *comp-inj-on*[*of f S to-nat*] **by** (*auto intro*: *countableI*)

lemma *countableE-bij*:
assumes *S*: *countable S* **obtains** *f* :: *nat* ⇒ *'a* **and** *C* :: *nat set* **where** *bij-betw f C S*
using *S* **by** (*blast elim*: *countableE dest*: *inj-on-imp-bij-betw bij-betw-inv*)

lemma *countableI-bij*: *bij-betw f* (*C*::*nat set*) *S* ⟹ *countable S*
by (*blast intro*: *countableI bij-betw-inv-into bij-betw-imp-inj-on*)

lemma *countable-finite*: *finite S* ⟹ *countable S*
by (*blast dest*: *finite-imp-inj-to-nat-seg countableI*)

lemma *countableI-bij1*: *bij-betw f A B* ⟹ *countable A* ⟹ *countable B*
by (*blast elim*: *countableE-bij intro*: *bij-betw-trans countableI-bij*)

lemma *countableI-bij2*: *bij-betw f B A* ⟹ *countable A* ⟹ *countable B*
by (*blast elim*: *countableE-bij intro*: *bij-betw-trans bij-betw-inv-into countableI-bij*)

lemma *countable-iff-bij*[*simp*]: *bij-betw f A B* ⟹ *countable A* ⟷ *countable B*
by (*blast intro*: *countableI-bij1 countableI-bij2*)

lemma *countable-subset*: *A* ⊆ *B* ⟹ *countable B* ⟹ *countable A*
by (*auto simp*: *countable-def intro*: *subset-inj-on*)

lemma *countableI-type*[*intro*, *simp*]: *countable* (*A*:: *'a* :: *countable set*)
using *countableI*[*of to-nat A*] **by** *auto*

6.2 Enumerate a countable set

lemma *countableE-infinite*:
assumes *countable S infinite S*
obtains *e* :: *'a* ⇒ *nat* **where** *bij-betw e S UNIV*
proof −
obtain *f* :: *'a* ⇒ *nat* **where** *inj-on f S*
using ‹*countable S*› **by** (*rule countableE*)
then have *bij-betw f S* (*f*‘*S*)
unfolding *bij-betw-def* **by** *simp*
moreover
from ‹*inj-on f S*› ‹*infinite S*› **have** *inf-fS*: *infinite* (*f*‘*S*)
by (*auto dest*: *finite-imageD*)
then have *bij-betw* (*the-inv-into UNIV* (*enumerate* (*f*‘*S*))) (*f*‘*S*) *UNIV*
by (*intro bij-betw-the-inv-into bij-enumerate*)
ultimately have *bij-betw* (*the-inv-into UNIV* (*enumerate* (*f*‘*S*)) ∘ *f*) *S UNIV*
by (*rule bij-betw-trans*)
then show *thesis* **..**

qed

lemma *countable-enum-cases*:
assumes *countable S*
obtains (*finite*) *f* :: *'a* ⇒ *nat* **where** *finite S bij-betw f S* {..<*card S*}
| (*infinite*) *f* :: *'a* ⇒ *nat* **where** *infinite S bij-betw f S UNIV*
using *ex-bij-betw-finite-nat*[*of S*] *countableE-infinite* ‹*countable S*›
by (*cases finite S*) (*auto simp add*: *atLeast0LessThan*)

definition *to-nat-on* :: *'a set* ⇒ *'a* ⇒ *nat* **where**
to-nat-on S = (*SOME f*. *if finite S then bij-betw f S* {..< *card S*} *else bij-betw f S UNIV*)

definition *from-nat-into* :: *'a set* ⇒ *nat* ⇒ *'a* **where**
from-nat-into S n = (*if n* ∈ *to-nat-on S* ‘ *S then inv-into S* (*to-nat-on S*) *n else SOME s*. *s*∈*S*)

lemma *to-nat-on-finite*: *finite S* ⟹ *bij-betw* (*to-nat-on S*) *S* {..< *card S*}
using *ex-bij-betw-finite-nat* **unfolding** *to-nat-on-def*
by (*intro someI2-ex*[**where** *Q*=λ*f*. *bij-betw f S* {..<*card S*}]) (*auto simp add*: *atLeast0LessThan*)

lemma *to-nat-on-infinite*: *countable S* ⟹ *infinite S* ⟹ *bij-betw* (*to-nat-on S*) *S UNIV*
using *countableE-infinite* **unfolding** *to-nat-on-def*
by (*intro someI2-ex*[**where** *Q*=λ*f*. *bij-betw f S UNIV*]) *auto*

lemma *bij-betw-from-nat-into-finite*: *finite S* ⟹ *bij-betw* (*from-nat-into S*) {..< *card S*} *S*
unfolding *from-nat-into-def*[*abs-def*]
using *to-nat-on-finite*[*of S*]
apply (*subst bij-betw-cong*)
apply (*split split-if*)
apply (*simp add*: *bij-betw-def*)
apply (*auto cong*: *bij-betw-cong*
intro: *bij-betw-inv-into to-nat-on-finite*)
done

lemma *bij-betw-from-nat-into*: *countable S* ⟹ *infinite S* ⟹ *bij-betw* (*from-nat-into S*) *UNIV S*
unfolding *from-nat-into-def*[*abs-def*]
using *to-nat-on-infinite*[*of S*, *unfolded bij-betw-def*]
by (*auto cong*: *bij-betw-cong intro*: *bij-betw-inv-into to-nat-on-infinite*)

lemma *inj-on-to-nat-on*[*intro*]: *countable A* ⟹ *inj-on* (*to-nat-on A*) *A*
using *to-nat-on-infinite*[*of A*] *to-nat-on-finite*[*of A*]
by (*cases finite A*) (*auto simp*: *bij-betw-def*)

lemma *to-nat-on-inj*[*simp*]:

countable A $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ *b* $\in$ *A* $\Longrightarrow$ *to-nat-on A a* = *to-nat-on A b* $\longleftrightarrow$ *a* = *b*
using *inj-on-to-nat-on*[*of A*] **by** (*auto dest*: *inj-onD*)

lemma *from-nat-into-to-nat-on*[*simp*]: *countable A* $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ *from-nat-into A* (*to-nat-on A a*) = *a*
by (*auto simp*: *from-nat-into-def intro*!: *inv-into-f-f*)

lemma *subset-range-from-nat-into*: *countable A* $\Longrightarrow$ *A* $\subseteq$ *range* (*from-nat-into A*)
by (*auto intro*: *from-nat-into-to-nat-on*[*symmetric*])

lemma *from-nat-into*: *A* $\neq$ {} $\Longrightarrow$ *from-nat-into A n* $\in$ *A*
unfolding *from-nat-into-def* **by** (*metis equals0I inv-into-into someI-ex*)

lemma *range-from-nat-into-subset*: *A* $\neq$ {} $\Longrightarrow$ *range* (*from-nat-into A*) $\subseteq$ *A*
using *from-nat-into*[*of A*] **by** *auto*

lemma *range-from-nat-into*[*simp*]: *A* $\neq$ {} $\Longrightarrow$ *countable A* $\Longrightarrow$ *range* (*from-nat-into A*) = *A*
by (*metis equalityI range-from-nat-into-subset subset-range-from-nat-into*)

lemma *image-to-nat-on*: *countable A* $\Longrightarrow$ *infinite A* $\Longrightarrow$ *to-nat-on A* ' *A* = *UNIV*
using *to-nat-on-infinite*[*of A*] **by** (*simp add*: *bij-betw-def*)

lemma *to-nat-on-surj*: *countable A* $\Longrightarrow$ *infinite A* $\Longrightarrow$ $\exists$ *a*$\in$*A*. *to-nat-on A a* = *n*
by (*metis* (*no-types*) *image-iff iso-tuple-UNIV-I image-to-nat-on*)

lemma *to-nat-on-from-nat-into*[*simp*]: *n* $\in$ *to-nat-on A* ' *A* $\Longrightarrow$ *to-nat-on A* (*from-nat-into A n*) = *n*
by (*simp add*: *f-inv-into-f from-nat-into-def*)

lemma *to-nat-on-from-nat-into-infinite*[*simp*]:
countable A $\Longrightarrow$ *infinite A* $\Longrightarrow$ *to-nat-on A* (*from-nat-into A n*) = *n*
by (*metis image-iff to-nat-on-surj to-nat-on-from-nat-into*)

lemma *from-nat-into-inj*:
countable A $\Longrightarrow$ *m* $\in$ *to-nat-on A* ' *A* $\Longrightarrow$ *n* $\in$ *to-nat-on A* ' *A* $\Longrightarrow$
from-nat-into A m = *from-nat-into A n* $\longleftrightarrow$ *m* = *n*
by (*subst to-nat-on-inj*[*symmetric*, *of A*]) *auto*

lemma *from-nat-into-inj-infinite*[*simp*]:
countable A $\Longrightarrow$ *infinite A* $\Longrightarrow$ *from-nat-into A m* = *from-nat-into A n* $\longleftrightarrow$ *m* = *n*
using *image-to-nat-on*[*of A*] *from-nat-into-inj*[*of A m n*] **by** *simp*

lemma *eq-from-nat-into-iff*:
countable A $\Longrightarrow$ *x* $\in$ *A* $\Longrightarrow$ *i* $\in$ *to-nat-on A* ' *A* $\Longrightarrow$ *x* = *from-nat-into A i* $\longleftrightarrow$ *i* = *to-nat-on A x*
by *auto*

lemma *from-nat-into-surj*: *countable A* $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ $\exists$ *n. from-nat-into A n = a*
by (*rule exI*[*of - to-nat-on A a*]) *simp*

lemma *from-nat-into-inject*[*simp*]:
A $\neq$ {} $\Longrightarrow$ *countable A* $\Longrightarrow$ *B* $\neq$ {} $\Longrightarrow$ *countable B* $\Longrightarrow$ *from-nat-into A = from-nat-into B* $\longleftrightarrow$ *A = B*
by (*metis range-from-nat-into*)

lemma *inj-on-from-nat-into*: *inj-on from-nat-into* ({*A. A* $\neq$ {} $\wedge$ *countable A*})
unfolding *inj-on-def* **by** *auto*

6.3 Closure properties of countability

lemma *countable-SIGMA*[*intro*, *simp*]:
countable I $\Longrightarrow$ ($\bigwedge$*i. i* $\in$ *I* $\Longrightarrow$ *countable* (*A i*)) $\Longrightarrow$ *countable* (*SIGMA i* : *I. A i*)
by (*intro countableI'*[*of* λ(*i*, *a*). (*to-nat-on I i*, *to-nat-on* (*A i*) *a*)]) (*auto simp*: *inj-on-def*)

lemma *countable-image*[*intro*, *simp*]:
assumes *countable A*
shows *countable* (*f*‘*A*)
proof −
obtain *g* :: *'a* $\Rightarrow$ *nat* **where** *inj-on g A*
using *assms* **by** (*rule countableE*)
moreover have *inj-on* (*inv-into A f*) (*f*‘*A*) *inv-into A f* ‘ *f* ‘ *A* $\subseteq$ *A*
by (*auto intro*: *inj-on-inv-into inv-into-into*)
ultimately show *?thesis*
by (*blast dest*: *comp-inj-on subset-inj-on intro*: *countableI*)
qed

lemma *countable-image-inj-on*: *countable* (*f* ‘ *A*) $\Longrightarrow$ *inj-on f A* $\Longrightarrow$ *countable A*
by (*metis countable-image the-inv-into-onto*)

lemma *countable-UN*[*intro*, *simp*]:
fixes *I* :: *'i set* **and** *A* :: *'i => 'a set*
assumes *I*: *countable I*
assumes *A*: $\bigwedge$*i. i* $\in$ *I* $\Longrightarrow$ *countable* (*A i*)
shows *countable* ($\bigcup$ *i*$\in$*I. A i*)
proof −
have ($\bigcup$ *i*$\in$*I. A i*) = *snd* ‘ (*SIGMA i* : *I. A i*) **by** (*auto simp*: *image-iff*)
then show *?thesis* **by** (*simp add*: *assms*)
qed

lemma *countable-Un*[*intro*]: *countable A* $\Longrightarrow$ *countable B* $\Longrightarrow$ *countable* (*A* $\cup$ *B*)
by (*rule countable-UN*[*of* {*True*, *False*} λ*True* $\Rightarrow$ *A* | *False* $\Rightarrow$ *B*, *simplified*])
(*simp split*: *bool.split*)

lemma *countable-Un-iff*[*simp*]: *countable* $(A \cup B) \longleftrightarrow$ *countable* $A \wedge$ *countable* B
by (*metis countable-Un countable-subset inf-sup-ord*(*3*,*4*))

lemma *countable-Plus*[*intro*, *simp*]:
countable $A \Longrightarrow$ *countable* $B \Longrightarrow$ *countable* $(A <+> B)$
by (*simp add*: *Plus-def*)

lemma *countable-empty*[*intro*, *simp*]: *countable* $\{\}$
by (*blast intro*: *countable-finite*)

lemma *countable-insert*[*intro*, *simp*]: *countable* $A \Longrightarrow$ *countable* (*insert a A*)
using *countable-Un*[*of* $\{a\}$ *A*] **by** (*auto simp*: *countable-finite*)

lemma *countable-Int1*[*intro*, *simp*]: *countable* $A \Longrightarrow$ *countable* $(A \cap B)$
by (*force intro*: *countable-subset*)

lemma *countable-Int2*[*intro*, *simp*]: *countable* $B \Longrightarrow$ *countable* $(A \cap B)$
by (*blast intro*: *countable-subset*)

lemma *countable-INT*[*intro*, *simp*]: $i \in I \Longrightarrow$ *countable* $(A\ i) \Longrightarrow$ *countable* $(\bigcap i \in I.\ A\ i)$
by (*blast intro*: *countable-subset*)

lemma *countable-Diff*[*intro*, *simp*]: *countable* $A \Longrightarrow$ *countable* $(A - B)$
by (*blast intro*: *countable-subset*)

lemma *countable-insert-eq* [*simp*]: *countable* (*insert x A*) = *countable A*
by *auto* (*metis Diff-insert-absorb countable-Diff insert-absorb*)

lemma *countable-vimage*: $B \subseteq$ *range f* $\Longrightarrow$ *countable* $(f -` B) \Longrightarrow$ *countable B*
by (*metis Int-absorb2 assms countable-image image-vimage-eq*)

lemma *surj-countable-vimage*: *surj f* $\Longrightarrow$ *countable* $(f -` B) \Longrightarrow$ *countable B*
by (*metis countable-vimage top-greatest*)

lemma *countable-Collect*[*simp*]: *countable* $A \Longrightarrow$ *countable* $\{a \in A.\ \varphi\ a\}$
by (*metis Collect-conj-eq Int-absorb Int-commute Int-def countable-Int1*)

lemma *countable-Image*:
assumes $\bigwedge y.\ y \in Y \Longrightarrow$ *countable* $(X `` \{y\})$
assumes *countable Y*
shows *countable* $(X `` Y)$
proof −
have *countable* $(X `` (\bigcup y \in Y.\ \{y\}))$
unfolding *Image-UN* **by** (*intro countable-UN assms*)
then show *?thesis* **by** *simp*
qed

lemma *countable-relpow*:
fixes $X :: 'a\ rel$
assumes *Image-X*: $\bigwedge Y.\ countable\ Y \Longrightarrow countable\ (X\ ``\ Y)$
assumes Y: $countable\ Y$
shows $countable\ ((X\ \hat{}\hat{}\ i)\ ``\ Y)$
using Y **by** (*induct i arbitrary*: Y) (*auto simp*: *relcomp-Image Image-X*)

lemma *countable-funpow*:
fixes $f :: 'a\ set \Rightarrow 'a\ set$
assumes $\bigwedge A.\ countable\ A \Longrightarrow countable\ (f\ A)$
and $countable\ A$
shows $countable\ ((f\ \hat{}\hat{}\ n)\ A)$
by(*induction n*)(*simp-all add*: *assms*)

lemma *countable-rtrancl*:
$(\bigwedge Y.\ countable\ Y \Longrightarrow countable\ (X\ ``\ Y)) \Longrightarrow countable\ Y \Longrightarrow countable\ (X\hat{}*\ ``\ Y)$
unfolding *rtrancl-is-UN-relpow UN-Image* **by** (*intro countable-UN countableI-type countable-relpow*)

lemma *countable-lists*[*intro*, *simp*]:
assumes A: $countable\ A$ **shows** $countable\ (lists\ A)$
proof −
have $countable\ (lists\ (range\ (from\text{-}nat\text{-}into\ A)))$
by (*auto simp*: *lists-image*)
with A **show** *?thesis*
by (*auto dest*: *subset-range-from-nat-into countable-subset lists-mono*)
qed

lemma *Collect-finite-eq-lists*: $Collect\ finite = set\ `\ lists\ UNIV$
using *finite-list* **by** *auto*

lemma *countable-Collect-finite*: $countable\ (Collect\ (finite::'a::countable\ set \Rightarrow bool))$
by (*simp add*: *Collect-finite-eq-lists*)

lemma *countable-rat*: $countable\ \mathbb{Q}$
unfolding *Rats-def* **by** *auto*

lemma *Collect-finite-subset-eq-lists*: $\{A.\ finite\ A \wedge A \subseteq T\} = set\ `\ lists\ T$
using *finite-list* **by** (*auto simp*: *lists-eq-set*)

lemma *countable-Collect-finite-subset*:
$countable\ T \Longrightarrow countable\ \{A.\ finite\ A \wedge A \subseteq T\}$
unfolding *Collect-finite-subset-eq-lists* **by** *auto*

lemma *countable-set-option* [*simp*]: $countable\ (set\text{-}option\ x)$
by(*cases x*) *auto*

6.4 Misc lemmas

lemma *countable-all*:
assumes *S*: *countable S*
shows $(\forall s{\in}S.\ P\ s) \longleftrightarrow (\forall n{::}nat.\ from\text{-}nat\text{-}into\ S\ n \in S \longrightarrow P\ (from\text{-}nat\text{-}into\ S\ n))$
using *S*[*THEN subset-range-from-nat-into*] **by** *auto*

lemma *finite-sequence-to-countable-set*:
assumes *countable X* **obtains** *F* **where** $\bigwedge i.\ F\ i \subseteq X\ \bigwedge i.\ F\ i \subseteq F\ (Suc\ i)\ \bigwedge i.\ finite\ (F\ i)\ (\bigcup i.\ F\ i) = X$
proof − **show** *thesis*
apply (*rule that*[*of* $\lambda i.$ *if* $X = \{\}$ *then* $\{\}$ *else from-nat-into X* ‘ $\{..i\}$])
apply (*auto simp*: *image-iff Ball-def intro*: *from-nat-into split*: *split-if-asm*)
proof −
fix *x n* **assume** $x \in X\ \forall i\ m.\ m \leq i \longrightarrow x \neq from\text{-}nat\text{-}into\ X\ m$
with *from-nat-into-surj*[*OF* ‹*countable X*› ‹$x \in X$›]
show *False*
by *auto*
qed
qed

6.5 Uncountable

abbreviation *uncountable* **where**
uncountable A ≡ ¬ *countable A*

lemma *uncountable-def*: *uncountable A* ⟷ *A* ≠ {} ∧ ¬ (∃*f*::(*nat* ⇒ *'a*). *range f* = *A*)
by (*auto intro*: *inj-on-inv-into simp*: *countable-def*)
(*metis all-not-in-conv inj-on-iff-surj subset-UNIV*)

lemma *uncountable-bij-betw*: *bij-betw f A B* ⟹ *uncountable B* ⟹ *uncountable A*
unfolding *bij-betw-def* **by** (*metis countable-image*)

lemma *uncountable-infinite*: *uncountable A* ⟹ *infinite A*
by (*metis countable-finite*)

lemma *uncountable-minus-countable*:
uncountable A ⟹ *countable B* ⟹ *uncountable* (*A* − *B*)
using *countable-Un*[*of B A* − *B*] *assms* **by** *auto*

lemma *countable-Diff-eq* [*simp*]: *countable* (*A* − {*x*}) = *countable A*
by (*meson countable-Diff countable-empty countable-insert uncountable-minus-countable*)

end

7 Pi and Function Sets

theory *FuncSet*
imports *Hilbert-Choice Main*
begin

definition $Pi :: 'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \Rightarrow 'b)\ set$
  where $Pi\ A\ B = \{f.\ \forall x.\ x \in A \longrightarrow f\ x \in B\ x\}$

definition $extensional :: 'a\ set \Rightarrow ('a \Rightarrow 'b)\ set$
  where $extensional\ A = \{f.\ \forall x.\ x \notin A \longrightarrow f\ x = undefined\}$

definition $restrict :: ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b$
  where $restrict\ f\ A = (\lambda x.\ if\ x \in A\ then\ f\ x\ else\ undefined)$

abbreviation $funcset :: 'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'b)\ set$ (**infixr** $\rightarrow$ *60*)
  where $A \rightarrow B \equiv Pi\ A\ (\lambda\text{-}.\ B)$

syntax (*ASCII*)
  -Pi :: $pttrn \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'b)\ set$ ((*3PI -:-./ -*) *10*)
  -lam :: $pttrn \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b \Rightarrow ('a \Rightarrow 'b)$ ((*3%-:-./ -*) [*0*,*0*,*3*] *3*)
syntax
  -Pi :: $pttrn \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'b)\ set$ ((*3*Π *-∈-./ -*) *10*)
  -lam :: $pttrn \Rightarrow 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)$ ((*3*λ*-∈-./ -*) [*0*,*0*,*3*] *3*)
translations
  $\Pi\ x{\in}A.\ B \rightleftharpoons CONST\ Pi\ A\ (\lambda x.\ B)$
  $\lambda x{\in}A.\ f \rightleftharpoons CONST\ restrict\ (\lambda x.\ f)\ A$

definition $compose :: 'a\ set \Rightarrow ('b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c)$
  where $compose\ A\ g\ f = (\lambda x{\in}A.\ g\ (f\ x))$

7.1 Basic Properties of *Pi*

lemma *Pi-I*[*intro*!]: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in B\ x) \Longrightarrow f \in Pi\ A\ B$
  by (*simp add*: *Pi-def*)

lemma *Pi-I'*[*simp*]: $(\bigwedge x.\ x \in A \longrightarrow f\ x \in B\ x) \Longrightarrow f \in Pi\ A\ B$
  by (*simp add*:*Pi-def*)

lemma *funcsetI*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in B) \Longrightarrow f \in A \rightarrow B$
  by (*simp add*: *Pi-def*)

lemma *Pi-mem*: $f \in Pi\ A\ B \Longrightarrow x \in A \Longrightarrow f\ x \in B\ x$
  by (*simp add*: *Pi-def*)

lemma *Pi-iff*: $f \in Pi\ I\ X \longleftrightarrow (\forall i{\in}I.\ f\ i \in X\ i)$
  unfolding *Pi-def* **by** *auto*

lemma *PiE* [*elim*]: $f \in Pi\ A\ B \Longrightarrow (f\ x \in B\ x \Longrightarrow Q) \Longrightarrow (x \notin A \Longrightarrow Q) \Longrightarrow Q$
  by (*auto simp*: *Pi-def*)

lemma *Pi-cong*: $(\bigwedge w.\ w \in A \Longrightarrow f\ w = g\ w) \Longrightarrow f \in Pi\ A\ B \longleftrightarrow g \in Pi\ A\ B$
by (*auto simp*: *Pi-def*)

lemma *funcset-id* [*simp*]: $(\lambda x.\ x) \in A \to A$
by *auto*

lemma *funcset-mem*: $f \in A \to B \Longrightarrow x \in A \Longrightarrow f\ x \in B$
by (*simp add*: *Pi-def*)

lemma *funcset-image*: $f \in A \to B \Longrightarrow f\ `\ A \subseteq B$
by *auto*

lemma *image-subset-iff-funcset*: $F\ `\ A \subseteq B \longleftrightarrow F \in A \to B$
by *auto*

lemma *Pi-eq-empty*[*simp*]: $(\Pi\ x \in A.\ B\ x) = \{\} \longleftrightarrow (\exists\, x{\in}A.\ B\ x = \{\})$
apply (*simp add*: *Pi-def*)
apply *auto*

Converse direction requires Axiom of Choice to exhibit a function picking an element from each non-empty $B\ x$

apply (*drule-tac* $x = \lambda u.\ SOME\ y.\ y \in B\ u$ **in** *spec*)
apply *auto*
apply (*cut-tac* $P = \lambda y.\ y \in B\ x$ **in** *some-eq-ex*)
apply *auto*
done

lemma *Pi-empty* [*simp*]: $Pi\ \{\}\ B = UNIV$
by (*simp add*: *Pi-def*)

lemma *Pi-Int*: $Pi\ I\ E \cap Pi\ I\ F = (\Pi\ i{\in}I.\ E\ i \cap F\ i)$
by *auto*

lemma *Pi-UN*:
fixes $A :: nat \Rightarrow 'i \Rightarrow 'a\ set$
assumes *finite I*
and *mono*: $\bigwedge i\ n\ m.\ i \in I \Longrightarrow n \le m \Longrightarrow A\ n\ i \subseteq A\ m\ i$
shows $(\bigcup n.\ Pi\ I\ (A\ n)) = (\Pi\ i{\in}I.\ \bigcup n.\ A\ n\ i)$
proof (*intro set-eqI iffI*)
fix *f*
assume $f \in (\Pi\ i{\in}I.\ \bigcup n.\ A\ n\ i)$
then have $\forall\, i{\in}I.\ \exists\, n.\ f\ i \in A\ n\ i$
by *auto*
from *bchoice*[*OF this*] **obtain** *n* **where** *n*: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in (A\ (n\ i)\ i)$
by *auto*
obtain *k* **where** *k*: $\bigwedge i.\ i \in I \Longrightarrow n\ i \le k$
using ‹*finite I*› *finite-nat-set-iff-bounded-le*[*of n`I*] **by** *auto*
have $f \in Pi\ I\ (A\ k)$

proof (*intro Pi-I*)
fix i
assume $i \in I$
from *mono*[*OF this*, *of n i k*] *k*[*OF this*] *n*[*OF this*]
show $f\ i \in A\ k\ i$ **by** *auto*
qed
then show $f \in (\bigcup n.\ Pi\ I\ (A\ n))$
by *auto*
qed *auto*

lemma *Pi-UNIV* [*simp*]: $A \to UNIV = UNIV$
by (*simp add*: *Pi-def*)

Covariance of Pi-sets in their second argument

lemma *Pi-mono*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x \subseteq C\ x) \Longrightarrow Pi\ A\ B \subseteq Pi\ A\ C$
by *auto*

Contravariance of Pi-sets in their first argument

lemma *Pi-anti-mono*: $A' \subseteq A \Longrightarrow Pi\ A\ B \subseteq Pi\ A'\ B$
by *auto*

lemma *prod-final*:
assumes *1*: $fst \circ f \in Pi\ A\ B$
and *2*: $snd \circ f \in Pi\ A\ C$
shows $f \in (\Pi\ z \in A.\ B\ z \times C\ z)$
proof (*rule Pi-I*)
fix z
assume z: $z \in A$
have $f\ z = (fst\ (f\ z),\ snd\ (f\ z))$
by *simp*
also have $\ldots \in B\ z \times C\ z$
by (*metis SigmaI PiE o-apply 1 2 z*)
finally show $f\ z \in B\ z \times C\ z$ **.**
qed

lemma *Pi-split-domain*[*simp*]: $x \in Pi\ (I \cup J)\ X \longleftrightarrow x \in Pi\ I\ X \wedge x \in Pi\ J\ X$
by (*auto simp*: *Pi-def*)

lemma *Pi-split-insert-domain*[*simp*]: $x \in Pi\ (insert\ i\ I)\ X \longleftrightarrow x \in Pi\ I\ X \wedge x\ i \in X\ i$
by (*auto simp*: *Pi-def*)

lemma *Pi-cancel-fupd-range*[*simp*]: $i \notin I \Longrightarrow x \in Pi\ I\ (B(i := b)) \longleftrightarrow x \in Pi\ I\ B$
by (*auto simp*: *Pi-def*)

lemma *Pi-cancel-fupd*[*simp*]: $i \notin I \Longrightarrow x(i := a) \in Pi\ I\ B \longleftrightarrow x \in Pi\ I\ B$
by (*auto simp*: *Pi-def*)

lemma *Pi-fupd-iff*: $i \in I \Longrightarrow f \in Pi\ I\ (B(i := A)) \longleftrightarrow f \in Pi\ (I - \{i\})\ B \wedge f\ i \in A$
apply *auto*
apply (*drule-tac x=x* **in** *Pi-mem*)
apply (*simp-all split*: *split-if-asm*)
apply (*drule-tac x=i* **in** *Pi-mem*)
apply (*auto dest*!: *Pi-mem*)
done

7.2 Composition With a Restricted Domain: *compose*

lemma *funcset-compose*: $f \in A \to B \Longrightarrow g \in B \to C \Longrightarrow compose\ A\ g\ f \in A \to C$
by (*simp add*: *Pi-def compose-def restrict-def*)

lemma *compose-assoc*:
assumes $f \in A \to B$
and $g \in B \to C$
and $h \in C \to D$
shows $compose\ A\ h\ (compose\ A\ g\ f) = compose\ A\ (compose\ B\ h\ g)\ f$
using *assms* **by** (*simp add*: *fun-eq-iff Pi-def compose-def restrict-def*)

lemma *compose-eq*: $x \in A \Longrightarrow compose\ A\ g\ f\ x = g\ (f\ x)$
by (*simp add*: *compose-def restrict-def*)

lemma *surj-compose*: $f\ `\ A = B \Longrightarrow g\ `\ B = C \Longrightarrow compose\ A\ g\ f\ `\ A = C$
by (*auto simp add*: *image-def compose-eq*)

7.3 Bounded Abstraction: *restrict*

lemma *restrict-cong*: $I = J \Longrightarrow (\bigwedge i.\ i \in J =simp=> f\ i = g\ i) \Longrightarrow restrict\ f\ I = restrict\ g\ J$
by (*auto simp*: *restrict-def fun-eq-iff simp-implies-def*)

lemma *restrict-in-funcset*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in B) \Longrightarrow (\lambda x{\in}A.\ f\ x) \in A \to B$
by (*simp add*: *Pi-def restrict-def*)

lemma *restrictI*[*intro*!]: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in B\ x) \Longrightarrow (\lambda x{\in}A.\ f\ x) \in Pi\ A\ B$
by (*simp add*: *Pi-def restrict-def*)

lemma *restrict-apply*[*simp*]: $(\lambda y{\in}A.\ f\ y)\ x = (if\ x \in A\ then\ f\ x\ else\ undefined)$
by (*simp add*: *restrict-def*)

lemma *restrict-apply'*: $x \in A \Longrightarrow (\lambda y{\in}A.\ f\ y)\ x = f\ x$
by *simp*

lemma *restrict-ext*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x = g\ x) \Longrightarrow (\lambda x{\in}A.\ f\ x) = (\lambda x{\in}A.\ g\ x)$
by (*simp add*: *fun-eq-iff Pi-def restrict-def*)

lemma *restrict-UNIV*: $restrict\ f\ UNIV = f$

by (*simp add*: *restrict-def*)

lemma *inj-on-restrict-eq* [*simp*]: *inj-on* (*restrict f A*) *A* = *inj-on f A*
by (*simp add*: *inj-on-def restrict-def*)

lemma *Id-compose*: $f \in A \to B \Longrightarrow f \in$ *extensional A* $\Longrightarrow$ *compose A* ($\lambda y{\in}B.\ y$) *f* = *f*
by (*auto simp add*: *fun-eq-iff compose-def extensional-def Pi-def*)

lemma *compose-Id*: $g \in A \to B \Longrightarrow g \in$ *extensional A* $\Longrightarrow$ *compose A g* ($\lambda x{\in}A.\ x$) = *g*
by (*auto simp add*: *fun-eq-iff compose-def extensional-def Pi-def*)

lemma *image-restrict-eq* [*simp*]: (*restrict f A*) ‘ *A* = *f* ‘ *A*
by (*auto simp add*: *restrict-def*)

lemma *restrict-restrict*[*simp*]: *restrict* (*restrict f A*) *B* = *restrict f* (*A* ∩ *B*)
unfolding *restrict-def* **by** (*simp add*: *fun-eq-iff*)

lemma *restrict-fupd*[*simp*]: *i* ∉ *I* $\Longrightarrow$ *restrict* (*f* (*i* := *x*)) *I* = *restrict f I*
by (*auto simp*: *restrict-def*)

lemma *restrict-upd*[*simp*]: *i* ∉ *I* $\Longrightarrow$ (*restrict f I*)(*i* := *y*) = *restrict* (*f*(*i* := *y*)) (*insert i I*)
by (*auto simp*: *fun-eq-iff*)

lemma *restrict-Pi-cancel*: *restrict x I* ∈ *Pi I A* ⟷ *x* ∈ *Pi I A*
by (*auto simp*: *restrict-def Pi-def*)

7.4 Bijections Between Sets

The definition of *bij-betw* is in *Fun.thy*, but most of the theorems belong here, or need at least *Hilbert-Choice*.

lemma *bij-betwI*:
assumes *f* ∈ *A* → *B*
and *g* ∈ *B* → *A*
and *g-f*: $\bigwedge x.\ x{\in}A \Longrightarrow g\ (f\ x) = x$
and *f-g*: $\bigwedge y.\ y{\in}B \Longrightarrow f\ (g\ y) = y$
shows *bij-betw f A B*
unfolding *bij-betw-def*
proof
show *inj-on f A*
by (*metis g-f inj-on-def*)
have *f* ‘ *A* ⊆ *B*
using ‹*f* ∈ *A* → *B*› **by** *auto*
moreover
have *B* ⊆ *f* ‘ *A*
by *auto* (*metis Pi-mem* ‹*g* ∈ *B* → *A*› *f-g image-iff*)
ultimately show *f* ‘ *A* = *B*

by *blast*
qed

lemma *bij-betw-imp-funcset*: *bij-betw f A B* $\Longrightarrow$ *f* $\in$ *A* $\rightarrow$ *B*
by (*auto simp add*: *bij-betw-def*)

lemma *inj-on-compose*: *bij-betw f A B* $\Longrightarrow$ *inj-on g B* $\Longrightarrow$ *inj-on* (*compose A g f*) *A*
by (*auto simp add*: *bij-betw-def inj-on-def compose-eq*)

lemma *bij-betw-compose*: *bij-betw f A B* $\Longrightarrow$ *bij-betw g B C* $\Longrightarrow$ *bij-betw* (*compose A g f*) *A C*
apply (*simp add*: *bij-betw-def compose-eq inj-on-compose*)
apply (*auto simp add*: *compose-def image-def*)
done

lemma *bij-betw-restrict-eq* [*simp*]: *bij-betw* (*restrict f A*) *A B* = *bij-betw f A B*
by (*simp add*: *bij-betw-def*)

7.5 Extensionality

lemma *extensional-empty*[*simp*]: *extensional* {} = {λx. *undefined*}
unfolding *extensional-def* **by** *auto*

lemma *extensional-arb*: *f* $\in$ *extensional A* $\Longrightarrow$ *x* $\notin$ *A* $\Longrightarrow$ *f x* = *undefined*
by (*simp add*: *extensional-def*)

lemma *restrict-extensional* [*simp*]: *restrict f A* $\in$ *extensional A*
by (*simp add*: *restrict-def extensional-def*)

lemma *compose-extensional* [*simp*]: *compose A f g* $\in$ *extensional A*
by (*simp add*: *compose-def*)

lemma *extensionalityI*:
assumes *f* $\in$ *extensional A*
and *g* $\in$ *extensional A*
and $\bigwedge x$. *x* $\in$ *A* $\Longrightarrow$ *f x* = *g x*
shows *f* = *g*
using *assms* **by** (*force simp add*: *fun-eq-iff extensional-def*)

lemma *extensional-restrict*: *f* $\in$ *extensional A* $\Longrightarrow$ *restrict f A* = *f*
by (*rule extensionalityI*[*OF restrict-extensional*]) *auto*

lemma *extensional-subset*: *f* $\in$ *extensional A* $\Longrightarrow$ *A* $\subseteq$ *B* $\Longrightarrow$ *f* $\in$ *extensional B*
unfolding *extensional-def* **by** *auto*

lemma *inv-into-funcset*: *f* ' *A* = *B* $\Longrightarrow$ ($\lambda x \in B$. *inv-into A f x*) $\in$ *B* $\rightarrow$ *A*
by (*unfold inv-into-def*) (*fast intro*: *someI2*)

lemma *compose-inv-into-id*: *bij-betw f A B* $\Longrightarrow$ *compose A* ($\lambda y \in B$. *inv-into A f y*) *f* = ($\lambda x \in A$. *x*)
apply (*simp add*: *bij-betw-def compose-def*)
apply (*rule restrict-ext*, *auto*)
done

lemma *compose-id-inv-into*: *f* ' *A* = *B* $\Longrightarrow$ *compose B f* ($\lambda y \in B$. *inv-into A f y*) = ($\lambda x \in B$. *x*)
apply (*simp add*: *compose-def*)
apply (*rule restrict-ext*)
apply (*simp add*: *f-inv-into-f*)
done

lemma *extensional-insert*[*intro*, *simp*]:
assumes *a* $\in$ *extensional* (*insert i I*)
shows *a*(*i* := *b*) $\in$ *extensional* (*insert i I*)
using *assms* **unfolding** *extensional-def* **by** *auto*

lemma *extensional-Int*[*simp*]: *extensional I* $\cap$ *extensional I'* = *extensional* (*I* $\cap$ *I'*)
unfolding *extensional-def* **by** *auto*

lemma *extensional-UNIV*[*simp*]: *extensional UNIV* = *UNIV*
by (*auto simp*: *extensional-def*)

lemma *restrict-extensional-sub*[*intro*]: *A* $\subseteq$ *B* $\Longrightarrow$ *restrict f A* $\in$ *extensional B*
unfolding *restrict-def extensional-def* **by** *auto*

lemma *extensional-insert-undefined*[*intro*, *simp*]:
a $\in$ *extensional* (*insert i I*) $\Longrightarrow$ *a*(*i* := *undefined*) $\in$ *extensional I*
unfolding *extensional-def* **by** *auto*

lemma *extensional-insert-cancel*[*intro*, *simp*]:
a $\in$ *extensional I* $\Longrightarrow$ *a* $\in$ *extensional* (*insert i I*)
unfolding *extensional-def* **by** *auto*

7.6 Cardinality

lemma *card-inj*: *f* $\in$ *A* $\rightarrow$ *B* $\Longrightarrow$ *inj-on f A* $\Longrightarrow$ *finite B* $\Longrightarrow$ *card A* $\leq$ *card B*
by (*rule card-inj-on-le*) *auto*

lemma *card-bij*:
assumes *f* $\in$ *A* $\rightarrow$ *B inj-on f A*
and *g* $\in$ *B* $\rightarrow$ *A inj-on g B*
and *finite A finite B*
shows *card A* = *card B*
using *assms* **by** (*blast intro*: *card-inj order-antisym*)

7.7 Extensional Function Spaces

definition *PiE* :: *'a set* ⇒ (*'a* ⇒ *'b set*) ⇒ (*'a* ⇒ *'b*) *set*
  where *PiE S T* = *Pi S T* ∩ *extensional S*

abbreviation Pi_E *A B* ≡ *PiE A B*

syntax (*ASCII*)
  -PiE :: *pttrn* ⇒ *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set* ((*3PIE -:-./ -*) *10*)
syntax
  -PiE :: *pttrn* ⇒ *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set* ((*3*Π_E *-∈-./ -*) *10*)
translations
  Π_E *x∈A. B* ⇌ *CONST* Pi_E *A* (λ*x. B*)

abbreviation *extensional-funcset* :: *'a set* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) *set* (**infixr** $\rightarrow_E$ *60*)
  where *A* $\rightarrow_E$ *B* ≡ (Π_E *i∈A. B*)

lemma *extensional-funcset-def*: *extensional-funcset S T* = (*S* → *T*) ∩ *extensional S*
  by (*simp add*: *PiE-def*)

lemma *PiE-empty-domain*[*simp*]: *PiE* {} *T* = {λ*x. undefined*}
  unfolding *PiE-def* **by** *simp*

lemma *PiE-UNIV-domain*: *PiE UNIV T* = *Pi UNIV T*
  unfolding *PiE-def* **by** *simp*

lemma *PiE-empty-range*[*simp*]: *i* ∈ *I* ⟹ *F i* = {} ⟹ (Π_E *i∈I. F i*) = {}
  unfolding *PiE-def* **by** *auto*

lemma *PiE-eq-empty-iff*: Pi_E *I F* = {} ⟷ (∃ *i∈I. F i* = {})
proof
  assume Pi_E *I F* = {}
  show ∃ *i∈I. F i* = {}
  proof (*rule ccontr*)
    assume ¬ *?thesis*
    then have ∀ *i*. ∃ *y*. (*i* ∈ *I* ⟶ *y* ∈ *F i*) ∧ (*i* ∉ *I* ⟶ *y* = *undefined*)
      by *auto*
    from *choice*[*OF this*]
    obtain *f* **where** ∀ *x*. (*x* ∈ *I* ⟶ *f x* ∈ *F x*) ∧ (*x* ∉ *I* ⟶ *f x* = *undefined*) **..**
    then have *f* ∈ Pi_E *I F*
      by (*auto simp*: *extensional-def PiE-def*)
    with ‹Pi_E *I F* = {}› **show** *False*
      by *auto*
  qed
qed (*auto simp*: *PiE-def*)

lemma *PiE-arb*: *f* ∈ *PiE S T* ⟹ *x* ∉ *S* ⟹ *f x* = *undefined*
  unfolding *PiE-def* **by** *auto* (*auto dest*!: *extensional-arb*)

lemma *PiE-mem*: $f \in PiE\ S\ T \Longrightarrow x \in S \Longrightarrow f\ x \in T\ x$
unfolding *PiE-def* **by** *auto*

lemma *PiE-fun-upd*: $y \in T\ x \Longrightarrow f \in PiE\ S\ T \Longrightarrow f(x := y) \in PiE\ (insert\ x\ S)\ T$
unfolding *PiE-def extensional-def* **by** *auto*

lemma *fun-upd-in-PiE*: $x \notin S \Longrightarrow f \in PiE\ (insert\ x\ S)\ T \Longrightarrow f(x := undefined) \in PiE\ S\ T$
unfolding *PiE-def extensional-def* **by** *auto*

lemma *PiE-insert-eq*: $PiE\ (insert\ x\ S)\ T = (\lambda(y,\ g).\ g(x := y))\ `\ (T\ x \times PiE\ S\ T)$
proof −
{
fix *f* **assume** $f \in PiE\ (insert\ x\ S)\ T\ x \notin S$
with *assms* **have** $f \in (\lambda(y,\ g).\ g(x := y))\ `\ (T\ x \times PiE\ S\ T)$
by (*auto intro*!: *image-eqI*[**where** $x=(f\ x, f(x := undefined))$] *intro*: *fun-upd-in-PiE PiE-mem*)
}
moreover
{
fix *f* **assume** $f \in PiE\ (insert\ x\ S)\ T\ x \in S$
with *assms* **have** $f \in (\lambda(y,\ g).\ g(x := y))\ `\ (T\ x \times PiE\ S\ T)$
by (*auto intro*!: *image-eqI*[**where** $x=(f\ x,\ f)$] *intro*: *fun-upd-in-PiE PiE-mem simp*: *insert-absorb*)
}
ultimately show *?thesis*
using *assms* **by** (*auto intro*: *PiE-fun-upd*)
qed

lemma *PiE-Int*: $Pi_E\ I\ A \cap Pi_E\ I\ B = Pi_E\ I\ (\lambda x.\ A\ x \cap B\ x)$
by (*auto simp*: *PiE-def*)

lemma *PiE-cong*: $(\bigwedge i.\ i \in I \Longrightarrow A\ i = B\ i) \Longrightarrow Pi_E\ I\ A = Pi_E\ I\ B$
unfolding *PiE-def* **by** (*auto simp*: *Pi-cong*)

lemma *PiE-E* [*elim*]:
assumes $f \in PiE\ A\ B$
obtains $x \in A$ **and** $f\ x \in B\ x$
| $x \notin A$ **and** $f\ x = undefined$
using *assms* **by** (*auto simp*: *Pi-def PiE-def extensional-def*)

lemma *PiE-I*[*intro*!]:
$(\bigwedge x.\ x \in A \Longrightarrow f\ x \in B\ x) \Longrightarrow (\bigwedge x.\ x \notin A \Longrightarrow f\ x = undefined) \Longrightarrow f \in PiE\ A\ B$
by (*simp add*: *PiE-def extensional-def*)

lemma *PiE-mono*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x \subseteq C\ x) \Longrightarrow PiE\ A\ B \subseteq PiE\ A\ C$
by *auto*

lemma *PiE-iff*: $f \in PiE\ I\ X \longleftrightarrow (\forall i{\in}I.\ f\ i \in X\ i) \wedge f \in extensional\ I$
by (*simp add*: *PiE-def Pi-iff*)

lemma *PiE-restrict*[*simp*]: $f \in PiE\ A\ B \Longrightarrow restrict\ f\ A = f$
by (*simp add*: *extensional-restrict PiE-def*)

lemma *restrict-PiE*[*simp*]: $restrict\ f\ I \in PiE\ I\ S \longleftrightarrow f \in Pi\ I\ S$
by (*auto simp*: *PiE-iff*)

lemma *PiE-eq-subset*:
assumes *ne*: $\bigwedge i.\ i \in I \Longrightarrow F\ i \neq \{\}$ $\bigwedge i.\ i \in I \Longrightarrow F'\ i \neq \{\}$
and *eq*: $Pi_E\ I\ F = Pi_E\ I\ F'$
and $i \in I$
shows $F\ i \subseteq F'\ i$
proof
fix x
assume $x \in F\ i$
with *ne* **have** $\forall j.\ \exists y.\ (j \in I \longrightarrow y \in F\ j \wedge (i = j \longrightarrow x = y)) \wedge (j \notin I \longrightarrow y = undefined)$
by *auto*
from *choice*[*OF this*] **obtain** f
where f: $\forall j.\ (j \in I \longrightarrow f\ j \in F\ j \wedge (i = j \longrightarrow x = f\ j)) \wedge (j \notin I \longrightarrow f\ j = undefined)$ **..**
then have $f \in Pi_E\ I\ F$
by (*auto simp*: *extensional-def PiE-def*)
then have $f \in Pi_E\ I\ F'$
using *assms* **by** *simp*
then show $x \in F'\ i$
using $f\ \langle i \in I \rangle$ **by** (*auto simp*: *PiE-def*)
qed

lemma *PiE-eq-iff-not-empty*:
assumes *ne*: $\bigwedge i.\ i \in I \Longrightarrow F\ i \neq \{\}$ $\bigwedge i.\ i \in I \Longrightarrow F'\ i \neq \{\}$
shows $Pi_E\ I\ F = Pi_E\ I\ F' \longleftrightarrow (\forall i{\in}I.\ F\ i = F'\ i)$
proof (*intro iffI ballI*)
fix i
assume *eq*: $Pi_E\ I\ F = Pi_E\ I\ F'$
assume i: $i \in I$
show $F\ i = F'\ i$
using *PiE-eq-subset*[*of I F F'*, *OF ne eq i*]
using *PiE-eq-subset*[*of I F' F*, *OF ne*(*2*,*1*) *eq*[*symmetric*] *i*]
by *auto*
qed (*auto simp*: *PiE-def*)

lemma *PiE-eq-iff*:
$Pi_E\ I\ F = Pi_E\ I\ F' \longleftrightarrow (\forall i{\in}I.\ F\ i = F'\ i) \vee ((\exists i{\in}I.\ F\ i = \{\}) \wedge (\exists i{\in}I.\ F'$

$i = \{\}))$
proof (*intro iffI disjCI*)
  assume *eq*[*simp*]: $Pi_E\ I\ F = Pi_E\ I\ F'$
  assume $\neg\ ((\exists\, i{\in}I.\ F\ i = \{\}) \wedge (\exists\, i{\in}I.\ F'\ i = \{\}))$
  then have $(\forall\, i{\in}I.\ F\ i \neq \{\}) \wedge (\forall\, i{\in}I.\ F'\ i \neq \{\})$
    using *PiE-eq-empty-iff*[*of I F*] *PiE-eq-empty-iff*[*of I F'*] **by** *auto*
  with *PiE-eq-iff-not-empty*[*of I F F'*] **show** $\forall\, i{\in}I.\ F\ i = F'\ i$
    by *auto*
next
  assume $(\forall\, i{\in}I.\ F\ i = F'\ i) \vee (\exists\, i{\in}I.\ F\ i = \{\}) \wedge (\exists\, i{\in}I.\ F'\ i = \{\})$
  then show $Pi_E\ I\ F = Pi_E\ I\ F'$
   using *PiE-eq-empty-iff*[*of I F*] *PiE-eq-empty-iff*[*of I F'*] **by** (*auto simp*: *PiE-def*)
qed

lemma *extensional-funcset-fun-upd-restricts-rangeI*:
  $\forall\, y \in S.\ f\ x \neq f\ y \Longrightarrow f \in (insert\ x\ S) \rightarrow_E T \Longrightarrow f(x := undefined) \in S \rightarrow_E (T - \{f\ x\})$
  unfolding *extensional-funcset-def extensional-def*
  apply *auto*
  apply (*case-tac x = xa*)
  apply *auto*
  done

lemma *extensional-funcset-fun-upd-extends-rangeI*:
  assumes $a \in T\ f \in S \rightarrow_E (T - \{a\})$
  shows $f(x := a) \in insert\ x\ S \rightarrow_E T$
  using *assms* **unfolding** *extensional-funcset-def extensional-def* **by** *auto*

7.7.1 Injective Extensional Function Spaces

lemma *extensional-funcset-fun-upd-inj-onI*:
  assumes $f \in S \rightarrow_E (T - \{a\})$
    and *inj-on f S*
  shows *inj-on* $(f(x := a))\ S$
  using *assms*
  unfolding *extensional-funcset-def* **by** (*auto intro*!: *inj-on-fun-updI*)

lemma *extensional-funcset-extend-domain-inj-on-eq*:
  assumes $x \notin S$
  shows $\{f.\ f \in (insert\ x\ S) \rightarrow_E T \wedge inj\text{-}on\ f\ (insert\ x\ S)\} =$
    $(\lambda(y,\ g).\ g(x{:=}y))\ `\ \{(y,\ g).\ y \in T \wedge g \in S \rightarrow_E (T - \{y\}) \wedge inj\text{-}on\ g\ S\}$
  using *assms*
  apply (*auto del*: *PiE-I PiE-E*)
  apply (*auto intro*: *extensional-funcset-fun-upd-inj-onI*
    extensional-funcset-fun-upd-extends-rangeI del: *PiE-I PiE-E*)
  apply (*auto simp add*: *image-iff inj-on-def*)
  apply (*rule-tac x=xa x* **in** *exI*)
  apply (*auto intro*: *PiE-mem del*: *PiE-I PiE-E*)
  apply (*rule-tac x=xa*(*x := undefined*) **in** *exI*)

apply (*auto intro*!: *extensional-funcset-fun-upd-restricts-rangeI*)
apply (*auto dest*!: *PiE-mem split*: *split-if-asm*)
done

lemma *extensional-funcset-extend-domain-inj-onI*:
assumes $x \notin S$
shows *inj-on* $(\lambda(y,\ g).\ g(x := y))$ $\{(y,\ g).\ y \in T \wedge g \in S \rightarrow_E (T - \{y\}) \wedge$ *inj-on g S*$\}$
using *assms*
apply (*auto intro*!: *inj-onI*)
apply (*metis fun-upd-same*)
apply (*metis assms PiE-arb fun-upd-triv fun-upd-upd*)
done

7.7.2 Cardinality

lemma *finite-PiE*: *finite S* $\Longrightarrow$ $(\bigwedge i.\ i \in S \Longrightarrow$ *finite* $(T\ i)) \Longrightarrow$ *finite* $(\Pi_E\ i \in S.\ T\ i)$
by (*induct S arbitrary*: *T rule*: *finite-induct*) (*simp-all add*: *PiE-insert-eq*)

lemma *inj-combinator*: $x \notin S \Longrightarrow$ *inj-on* $(\lambda(y,\ g).\ g(x := y))$ $(T\ x \times Pi_E\ S\ T)$
proof (*safe intro*!: *inj-onI ext*)
fix *f y g z*
assume $x \notin S$
assume *fg*: $f \in Pi_E\ S\ T\ g \in Pi_E\ S\ T$
assume $f(x := y) = g(x := z)$
then have $*$: $\bigwedge i.\ (f(x := y))\ i = (g(x := z))\ i$
unfolding *fun-eq-iff* **by** *auto*
from *this*[*of x*] **show** $y = z$ **by** *simp*
fix *i* **from** $*$[*of i*] ‹$x \notin S$› *fg* **show** $f\ i = g\ i$
by (*auto split*: *split-if-asm simp*: *PiE-def extensional-def*)
qed

lemma *card-PiE*: *finite S* $\Longrightarrow$ *card* $(\Pi_E\ i \in S.\ T\ i) = (\prod\ i \in S.$ *card* $(T\ i))$
proof (*induct rule*: *finite-induct*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x S*)
then show *?case*
by (*simp add*: *PiE-insert-eq inj-combinator card-image card-cartesian-product*)
qed

end

8 Square root of sum of squares

theory *L2-Norm*
imports *NthRoot*

begin

definition
setL2 f A = *sqrt* $(\sum i{\in}A.\ (f\ i)^2)$

lemma *setL2-cong*:
$[\![A = B;\ \bigwedge x.\ x \in B \Longrightarrow f\ x = g\ x]\!] \Longrightarrow$ *setL2 f A* = *setL2 g B*
unfolding *setL2-def* **by** *simp*

lemma *strong-setL2-cong*:
$[\![A = B;\ \bigwedge x.\ x \in B =simp=> f\ x = g\ x]\!] \Longrightarrow$ *setL2 f A* = *setL2 g B*
unfolding *setL2-def simp-implies-def* **by** *simp*

lemma *setL2-infinite* [*simp*]: ¬ *finite A* $\Longrightarrow$ *setL2 f A* = *0*
unfolding *setL2-def* **by** *simp*

lemma *setL2-empty* [*simp*]: *setL2 f* {} = *0*
unfolding *setL2-def* **by** *simp*

lemma *setL2-insert* [*simp*]:
$[\![$*finite F*; *a* ∉ *F*$]\!] \Longrightarrow$
setL2 f (*insert a F*) = *sqrt* $((f\ a)^2 + (setL2\ f\ F)^2)$
unfolding *setL2-def* **by** (*simp add*: *setsum-nonneg*)

lemma *setL2-nonneg* [*simp*]: *0* ≤ *setL2 f A*
unfolding *setL2-def* **by** (*simp add*: *setsum-nonneg*)

lemma *setL2-0'*: ∀ *a*∈*A*. *f a* = *0* $\Longrightarrow$ *setL2 f A* = *0*
unfolding *setL2-def* **by** *simp*

lemma *setL2-constant*: *setL2* $(\lambda x.\ y)$ *A* = *sqrt* (*of-nat* (*card A*)) * |*y*|
unfolding *setL2-def* **by** (*simp add*: *real-sqrt-mult*)

lemma *setL2-mono*:
assumes $\bigwedge i.\ i \in K \Longrightarrow f\ i \le g\ i$
assumes $\bigwedge i.\ i \in K \Longrightarrow 0 \le f\ i$
shows *setL2 f K* ≤ *setL2 g K*
unfolding *setL2-def*
by (*simp add*: *setsum-nonneg setsum-mono power-mono assms*)

lemma *setL2-strict-mono*:
assumes *finite K* **and** *K* ≠ {}
assumes $\bigwedge i.\ i \in K \Longrightarrow f\ i < g\ i$
assumes $\bigwedge i.\ i \in K \Longrightarrow 0 \le f\ i$
shows *setL2 f K* < *setL2 g K*
unfolding *setL2-def*
by (*simp add*: *setsum-strict-mono power-strict-mono assms*)

lemma *setL2-right-distrib*:

```
  0 ≤ r ⟹ r * setL2 f A = setL2 (λx. r * f x) A
  unfolding setL2-def
  apply (simp add: power-mult-distrib)
  apply (simp add: setsum-right-distrib [symmetric])
  apply (simp add: real-sqrt-mult setsum-nonneg)
  done

lemma setL2-left-distrib:
  0 ≤ r ⟹ setL2 f A * r = setL2 (λx. f x * r) A
  unfolding setL2-def
  apply (simp add: power-mult-distrib)
  apply (simp add: setsum-left-distrib [symmetric])
  apply (simp add: real-sqrt-mult setsum-nonneg)
  done

lemma setL2-eq-0-iff: finite A ⟹ setL2 f A = 0 ⟷ (∀x∈A. f x = 0)
  unfolding setL2-def
  by (simp add: setsum-nonneg setsum-nonneg-eq-0-iff)

lemma setL2-triangle-ineq:
  shows setL2 (λi. f i + g i) A ≤ setL2 f A + setL2 g A
proof (cases finite A)
  case False
  thus ?thesis by simp
next
  case True
  thus ?thesis
  proof (induct set: finite)
    case empty
    show ?case by simp
  next
    case (insert x F)
    hence sqrt ((f x + g x)^2 + (setL2 (λi. f i + g i) F)^2) ≤
          sqrt ((f x + g x)^2 + (setL2 f F + setL2 g F)^2)
      by (intro real-sqrt-le-mono add-left-mono power-mono insert
                setL2-nonneg add-increasing zero-le-power2)
    also have
      ... ≤ sqrt ((f x)^2 + (setL2 f F)^2) + sqrt ((g x)^2 + (setL2 g F)^2)
      by (rule real-sqrt-sum-squares-triangle-ineq)
    finally show ?case
      using insert by simp
  qed
qed

lemma sqrt-sum-squares-le-sum:
  ⟦0 ≤ x; 0 ≤ y⟧ ⟹ sqrt (x^2 + y^2) ≤ x + y
  apply (rule power2-le-imp-le)
  apply (simp add: power2-sum)
  apply simp
```

done

lemma *setL2-le-setsum* [*rule-format*]:
(∀ *i*∈*A*. *0* ≤ *f i*) ⟶ *setL2 f A* ≤ *setsum f A*
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply *simp*
apply *clarsimp*
apply (*erule order-trans* [*OF sqrt-sum-squares-le-sum*])
apply *simp*
apply *simp*
apply *simp*
done

lemma *sqrt-sum-squares-le-sum-abs*: *sqrt* (x^2 + y^2) ≤ |*x*| + |*y*|
apply (*rule power2-le-imp-le*)
apply (*simp add*: *power2-sum*)
apply *simp*
done

lemma *setL2-le-setsum-abs*: *setL2 f A* ≤ (∑ *i*∈*A*. |*f i*|)
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply *simp*
apply *simp*
apply (*rule order-trans* [*OF sqrt-sum-squares-le-sum-abs*])
apply *simp*
apply *simp*
done

lemma *setL2-mult-ineq-lemma*:
fixes *a b c d* :: *real*
shows *2* * (*a* * *c*) * (*b* * *d*) ≤ a^2 * d^2 + b^2 * c^2
proof −
have *0* ≤ (*a* * *d* − *b* * *c*)^2 **by** *simp*
also have ... = a^2 * d^2 + b^2 * c^2 − *2* * (*a* * *d*) * (*b* * *c*)
by (*simp only*: *power2-diff power-mult-distrib*)
also have ... = a^2 * d^2 + b^2 * c^2 − *2* * (*a* * *c*) * (*b* * *d*)
by *simp*
finally show *2* * (*a* * *c*) * (*b* * *d*) ≤ a^2 * d^2 + b^2 * c^2
by *simp*
qed

lemma *setL2-mult-ineq*: (∑ *i*∈*A*. |*f i*| * |*g i*|) ≤ *setL2 f A* * *setL2 g A*
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply *simp*
apply (*rule power2-le-imp-le*, *simp*)
apply (*rule order-trans*)

apply (*rule power-mono*)
apply (*erule add-left-mono*)
apply (*simp add*: *setsum-nonneg*)
apply (*simp add*: *power2-sum*)
apply (*simp add*: *power-mult-distrib*)
apply (*simp add*: *distrib-left distrib-right*)
apply (*rule ord-le-eq-trans*)
apply (*rule setL2-mult-ineq-lemma*)
apply *simp-all*
done

lemma *member-le-setL2*: $[\![finite\ A;\ i \in A]\!] \Longrightarrow f\ i \leq setL2\ f\ A$
apply (*rule-tac s=insert i* $(A - \{i\})$ **and** *t=A* **in** *subst*)
apply *fast*
apply (*subst setL2-insert*)
apply *simp*
apply *simp*
apply *simp*
done

end

9 Inner Product Spaces and the Gradient Derivative

theory *Inner-Product*
imports ~~/*src/HOL/Complex-Main*
begin

9.1 Real inner product spaces

Temporarily relax type constraints for *open*, *uniformity*, *dist*, and *norm*.

setup ‹*Sign.add-const-constraint*
(@{*const-name open*}, *SOME* @{*typ* $'a$::*open set* ⇒ *bool*})›

setup ‹*Sign.add-const-constraint*
(@{*const-name dist*}, *SOME* @{*typ* $'a$::*dist* ⇒ $'a$ ⇒ *real*})›

setup ‹*Sign.add-const-constraint*
(@{*const-name uniformity*}, *SOME* @{*typ* ($'a$::*uniformity* × $'a$) *filter*})›

setup ‹*Sign.add-const-constraint*
(@{*const-name norm*}, *SOME* @{*typ* $'a$::*norm* ⇒ *real*})›

class *real-inner* = *real-vector* + *sgn-div-norm* + *dist-norm* + *uniformity-dist* + *open-uniformity* +
fixes *inner* :: $'a$ ⇒ $'a$ ⇒ *real*
assumes *inner-commute*: *inner x y* = *inner y x*

and *inner-add-left*: $inner\ (x + y)\ z = inner\ x\ z + inner\ y\ z$
and *inner-scaleR-left* [*simp*]: $inner\ (scaleR\ r\ x)\ y = r * (inner\ x\ y)$
and *inner-ge-zero* [*simp*]: $0 \leq inner\ x\ x$
and *inner-eq-zero-iff* [*simp*]: $inner\ x\ x = 0 \longleftrightarrow x = 0$
and *norm-eq-sqrt-inner*: $norm\ x = sqrt\ (inner\ x\ x)$
begin

lemma *inner-zero-left* [*simp*]: $inner\ 0\ x = 0$
using *inner-add-left* [*of 0 0 x*] **by** *simp*

lemma *inner-minus-left* [*simp*]: $inner\ (-x)\ y = -\ inner\ x\ y$
using *inner-add-left* [*of x − x y*] **by** *simp*

lemma *inner-diff-left*: $inner\ (x - y)\ z = inner\ x\ z - inner\ y\ z$
using *inner-add-left* [*of x − y z*] **by** *simp*

lemma *inner-setsum-left*: $inner\ (\sum x{\in}A.\ f\ x)\ y = (\sum x{\in}A.\ inner\ (f\ x)\ y)$
by (*cases finite A*, *induct set*: *finite*, *simp-all add*: *inner-add-left*)

Transfer distributivity rules to right argument.

lemma *inner-add-right*: $inner\ x\ (y + z) = inner\ x\ y + inner\ x\ z$
using *inner-add-left* [*of y z x*] **by** (*simp only*: *inner-commute*)

lemma *inner-scaleR-right* [*simp*]: $inner\ x\ (scaleR\ r\ y) = r * (inner\ x\ y)$
using *inner-scaleR-left* [*of r y x*] **by** (*simp only*: *inner-commute*)

lemma *inner-zero-right* [*simp*]: $inner\ x\ 0 = 0$
using *inner-zero-left* [*of x*] **by** (*simp only*: *inner-commute*)

lemma *inner-minus-right* [*simp*]: $inner\ x\ (-y) = -\ inner\ x\ y$
using *inner-minus-left* [*of y x*] **by** (*simp only*: *inner-commute*)

lemma *inner-diff-right*: $inner\ x\ (y - z) = inner\ x\ y - inner\ x\ z$
using *inner-diff-left* [*of y z x*] **by** (*simp only*: *inner-commute*)

lemma *inner-setsum-right*: $inner\ x\ (\sum y{\in}A.\ f\ y) = (\sum y{\in}A.\ inner\ x\ (f\ y))$
using *inner-setsum-left* [*of f A x*] **by** (*simp only*: *inner-commute*)

lemmas *inner-add* [*algebra-simps*] = *inner-add-left inner-add-right*
lemmas *inner-diff* [*algebra-simps*] = *inner-diff-left inner-diff-right*
lemmas *inner-scaleR* = *inner-scaleR-left inner-scaleR-right*

Legacy theorem names

lemmas *inner-left-distrib* = *inner-add-left*
lemmas *inner-right-distrib* = *inner-add-right*
lemmas *inner-distrib* = *inner-left-distrib inner-right-distrib*

lemma *inner-gt-zero-iff* [*simp*]: $0 < inner\ x\ x \longleftrightarrow x \neq 0$
by (*simp add*: *order-less-le*)

lemma *power2-norm-eq-inner*: $(norm\ x)^2 = inner\ x\ x$
by (*simp add*: *norm-eq-sqrt-inner*)

Identities involving real multiplication and division.

lemma *inner-mult-left*: $inner\ (of\text{-}real\ m * a)\ b = m * (inner\ a\ b)$
by (*metis real-inner-class.inner-scaleR-left scaleR-conv-of-real*)

lemma *inner-mult-right*: $inner\ a\ (of\text{-}real\ m * b) = m * (inner\ a\ b)$
by (*metis real-inner-class.inner-scaleR-right scaleR-conv-of-real*)

lemma *inner-mult-left'*: $inner\ (a * of\text{-}real\ m)\ b = m * (inner\ a\ b)$
by (*simp add*: *of-real-def*)

lemma *inner-mult-right'*: $inner\ a\ (b * of\text{-}real\ m) = (inner\ a\ b) * m$
by (*simp add*: *of-real-def real-inner-class.inner-scaleR-right*)

lemma *Cauchy-Schwarz-ineq*:
$(inner\ x\ y)^2 \leq inner\ x\ x * inner\ y\ y$
proof (*cases*)
assume $y = 0$
thus *?thesis* **by** *simp*
next
assume *y*: $y \neq 0$
let *?r* $= inner\ x\ y\ /\ inner\ y\ y$
have $0 \leq inner\ (x - scaleR\ ?r\ y)\ (x - scaleR\ ?r\ y)$
by (*rule inner-ge-zero*)
also have $\ldots = inner\ x\ x - inner\ y\ x * ?r$
by (*simp add*: *inner-diff*)
also have $\ldots = inner\ x\ x - (inner\ x\ y)^2\ /\ inner\ y\ y$
by (*simp add*: *power2-eq-square inner-commute*)
finally have $0 \leq inner\ x\ x - (inner\ x\ y)^2\ /\ inner\ y\ y$ **.**
hence $(inner\ x\ y)^2\ /\ inner\ y\ y \leq inner\ x\ x$
by (*simp add*: *le-diff-eq*)
thus $(inner\ x\ y)^2 \leq inner\ x\ x * inner\ y\ y$
by (*simp add*: *pos-divide-le-eq y*)
qed

lemma *Cauchy-Schwarz-ineq2*:
$|inner\ x\ y| \leq norm\ x * norm\ y$
proof (*rule power2-le-imp-le*)
have $(inner\ x\ y)^2 \leq inner\ x\ x * inner\ y\ y$
using *Cauchy-Schwarz-ineq* **.**
thus $|inner\ x\ y|^2 \leq (norm\ x * norm\ y)^2$
by (*simp add*: *power-mult-distrib power2-norm-eq-inner*)
show $0 \leq norm\ x * norm\ y$
unfolding *norm-eq-sqrt-inner*
by (*intro mult-nonneg-nonneg real-sqrt-ge-zero inner-ge-zero*)
qed

lemma *norm-cauchy-schwarz*: *inner x y* $\leq$ *norm x* $*$ *norm y*
using *Cauchy-Schwarz-ineq2* [*of x y*] **by** *auto*

subclass *real-normed-vector*
proof
fix *a* :: *real* **and** *x y* :: *'a*
show *norm x = 0* $\longleftrightarrow$ *x = 0*
unfolding *norm-eq-sqrt-inner* **by** *simp*
show *norm* (*x + y*) $\leq$ *norm x + norm y*
proof (*rule power2-le-imp-le*)
have *inner x y* $\leq$ *norm x* $*$ *norm y*
by (*rule norm-cauchy-schwarz*)
thus (*norm* (*x + y*))^2 $\leq$ (*norm x + norm y*)^2
unfolding *power2-sum power2-norm-eq-inner*
by (*simp add*: *inner-add inner-commute*)
show *0* $\leq$ *norm x + norm y*
unfolding *norm-eq-sqrt-inner* **by** *simp*
qed
have *sqrt* (a^2 $*$ *inner x x*) = |*a*| $*$ *sqrt* (*inner x x*)
by (*simp add*: *real-sqrt-mult-distrib*)
then show *norm* (*a* $*_R$ *x*) = |*a*| $*$ *norm x*
unfolding *norm-eq-sqrt-inner*
by (*simp add*: *power2-eq-square mult.assoc*)
qed

end

lemma *inner-divide-left*:
fixes *a* :: *'a* :: {*real-inner*,*real-div-algebra*}
shows *inner* (*a* / *of-real m*) *b* = (*inner a b*) / *m*
by (*metis* (*no-types*) *divide-inverse inner-commute inner-scaleR-right mult.left-neutral mult.right-neutral mult-scaleR-right of-real-inverse scaleR-conv-of-real times-divide-eq-left*)

lemma *inner-divide-right*:
fixes *a* :: *'a* :: {*real-inner*,*real-div-algebra*}
shows *inner a* (*b* / *of-real m*) = (*inner a b*) / *m*
by (*metis inner-commute inner-divide-left*)

Re-enable constraints for *open*, *uniformity*, *dist*, and *norm*.

setup ‹*Sign.add-const-constraint*
(@{*const-name open*}, *SOME* @{*typ 'a*::*topological-space set* $\Rightarrow$ *bool*})›

setup ‹*Sign.add-const-constraint*
(@{*const-name uniformity*}, *SOME* @{*typ* ('*a*::*uniform-space* $\times$ *'a*) *filter*})›

setup ‹*Sign.add-const-constraint*
(@{*const-name dist*}, *SOME* @{*typ 'a*::*metric-space* $\Rightarrow$ *'a* $\Rightarrow$ *real*})›

setup ‹*Sign.add-const-constraint*
(@{*const-name norm*}, *SOME* @{*typ 'a*::*real-normed-vector* ⇒ *real*})›

lemma *bounded-bilinear-inner*:
bounded-bilinear (*inner*::'*a*::*real-inner* ⇒ '*a* ⇒ *real*)
proof
fix *x y z* :: '*a* **and** *r* :: *real*
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
by (*rule inner-add-left*)
show *inner x* (*y* + *z*) = *inner x y* + *inner x z*
by (*rule inner-add-right*)
show *inner* (*scaleR r x*) *y* = *scaleR r* (*inner x y*)
unfolding *real-scaleR-def* **by** (*rule inner-scaleR-left*)
show *inner x* (*scaleR r y*) = *scaleR r* (*inner x y*)
unfolding *real-scaleR-def* **by** (*rule inner-scaleR-right*)
show ∃ *K*. ∀ *x y*::'*a*. *norm* (*inner x y*) ≤ *norm x* ∗ *norm y* ∗ *K*
proof
show ∀ *x y*::'*a*. *norm* (*inner x y*) ≤ *norm x* ∗ *norm y* ∗ *1*
by (*simp add*: *Cauchy-Schwarz-ineq2*)
qed
qed

lemmas *tendsto-inner* [*tendsto-intros*] =
bounded-bilinear.tendsto [*OF bounded-bilinear-inner*]

lemmas *isCont-inner* [*simp*] =
bounded-bilinear.isCont [*OF bounded-bilinear-inner*]

lemmas *has-derivative-inner* [*derivative-intros*] =
bounded-bilinear.FDERIV [*OF bounded-bilinear-inner*]

lemmas *bounded-linear-inner-left* =
bounded-bilinear.bounded-linear-left [*OF bounded-bilinear-inner*]

lemmas *bounded-linear-inner-right* =
bounded-bilinear.bounded-linear-right [*OF bounded-bilinear-inner*]

lemmas *bounded-linear-inner-left-comp* = *bounded-linear-inner-left*[*THEN bounded-linear-compose*]

lemmas *bounded-linear-inner-right-comp* = *bounded-linear-inner-right*[*THEN bounded-linear-compose*]

lemmas *has-derivative-inner-right* [*derivative-intros*] =
bounded-linear.has-derivative [*OF bounded-linear-inner-right*]

lemmas *has-derivative-inner-left* [*derivative-intros*] =
bounded-linear.has-derivative [*OF bounded-linear-inner-left*]

lemma *differentiable-inner* [*simp*]:
f differentiable (*at x within s*) ⟹ *g differentiable at x within s* ⟹ (λ*x*. *inner* (*f*

x) (g x)) *differentiable at x within s*
unfolding *differentiable-def* **by** (*blast intro*: *has-derivative-inner*)

9.2 Class instances

instantiation *real* :: *real-inner*
begin

definition *inner-real-def* [*simp*]: *inner* = *op* ∗

instance
proof
fix *x y z r* :: *real*
show *inner x y* = *inner y x*
unfolding *inner-real-def* **by** (*rule mult.commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner-real-def* **by** (*rule distrib-right*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner-real-def real-scaleR-def* **by** (*rule mult.assoc*)
show *0* ≤ *inner x x*
unfolding *inner-real-def* **by** *simp*
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner-real-def* **by** *simp*
show *norm x* = *sqrt* (*inner x x*)
unfolding *inner-real-def* **by** *simp*
qed

end

instantiation *complex* :: *real-inner*
begin

definition *inner-complex-def*:
inner x y = *Re x* ∗ *Re y* + *Im x* ∗ *Im y*

instance
proof
fix *x y z* :: *complex* **and** *r* :: *real*
show *inner x y* = *inner y x*
unfolding *inner-complex-def* **by** (*simp add*: *mult.commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner-complex-def* **by** (*simp add*: *distrib-right*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner-complex-def* **by** (*simp add*: *distrib-left*)
show *0* ≤ *inner x x*
unfolding *inner-complex-def* **by** *simp*
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner-complex-def*
by (*simp add*: *add-nonneg-eq-0-iff complex-Re-Im-cancel-iff*)

show *norm x = sqrt (inner x x)*
unfolding *inner-complex-def complex-norm-def*
by (*simp add*: *power2-eq-square*)
qed

end

lemma *complex-inner-1* [*simp*]: *inner 1 x = Re x*
unfolding *inner-complex-def* **by** *simp*

lemma *complex-inner-1-right* [*simp*]: *inner x 1 = Re x*
unfolding *inner-complex-def* **by** *simp*

lemma *complex-inner-ii-left* [*simp*]: *inner ii x = Im x*
unfolding *inner-complex-def* **by** *simp*

lemma *complex-inner-ii-right* [*simp*]: *inner x ii = Im x*
unfolding *inner-complex-def* **by** *simp*

9.3 Gradient derivative

definition
gderiv ::
[*'a*::*real-inner* ⇒ *real*, *'a*, *'a*] ⇒ *bool*
((*GDERIV* (-)/ (-)/ :> (-)) [*1000*, *1000*, *60*] *60*)
where
GDERIV f x :> D ⟷ *FDERIV f x :> (λh. inner h D)*

lemma *gderiv-deriv* [*simp*]: *GDERIV f x :> D* ⟷ *DERIV f x :> D*
by (*simp only*: *gderiv-def has-field-derivative-def inner-real-def mult-commute-abs*)

lemma *GDERIV-DERIV-compose*:
⟦*GDERIV f x :> df*; *DERIV g (f x) :> dg*⟧
⟹ *GDERIV (λx. g (f x)) x :> scaleR dg df*
unfolding *gderiv-def has-field-derivative-def*
apply (*drule* (*1*) *has-derivative-compose*)
apply (*simp add*: *ac-simps*)
done

lemma *has-derivative-subst*: ⟦*FDERIV f x :> df*; *df = d*⟧ ⟹ *FDERIV f x :> d*
by *simp*

lemma *GDERIV-subst*: ⟦*GDERIV f x :> df*; *df = d*⟧ ⟹ *GDERIV f x :> d*
by *simp*

lemma *GDERIV-const*: *GDERIV (λx. k) x :> 0*
unfolding *gderiv-def inner-zero-right* **by** (*rule has-derivative-const*)

lemma *GDERIV-add*:

⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *f x* + *g x*) *x :> df* + *dg*
unfolding *gderiv-def inner-add-right* **by** (*rule has-derivative-add*)

lemma *GDERIV-minus*:
GDERIV f x :> df ⟹ *GDERIV* (λ*x*. − *f x*) *x :> − df*
unfolding *gderiv-def inner-minus-right* **by** (*rule has-derivative-minus*)

lemma *GDERIV-diff*:
⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *f x* − *g x*) *x :> df* − *dg*
unfolding *gderiv-def inner-diff-right* **by** (*rule has-derivative-diff*)

lemma *GDERIV-scaleR*:
⟦*DERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *scaleR* (*f x*) (*g x*)) *x*
:> (*scaleR* (*f x*) *dg* + *scaleR df* (*g x*))
unfolding *gderiv-def has-field-derivative-def inner-add-right inner-scaleR-right*
apply (*rule has-derivative-subst*)
apply (*erule* (*1*) *has-derivative-scaleR*)
apply (*simp add*: *ac-simps*)
done

lemma *GDERIV-mult*:
⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *f x* ∗ *g x*) *x :> scaleR* (*f x*) *dg* + *scaleR* (*g x*) *df*
unfolding *gderiv-def*
apply (*rule has-derivative-subst*)
apply (*erule* (*1*) *has-derivative-mult*)
apply (*simp add*: *inner-add ac-simps*)
done

lemma *GDERIV-inverse*:
⟦*GDERIV f x :> df*; *f x* ≠ *0*⟧
⟹ *GDERIV* (λ*x*. *inverse* (*f x*)) *x :> −* (*inverse* (*f x*))^2 ∗$_R$ *df*
apply (*erule GDERIV-DERIV-compose*)
apply (*erule DERIV-inverse* [*folded numeral-2-eq-2*])
done

lemma *GDERIV-norm*:
assumes *x* ≠ *0* **shows** *GDERIV* (λ*x*. *norm x*) *x :> sgn x*
proof −
have *1*: *FDERIV* (λ*x*. *inner x x*) *x :>* (λ*h*. *inner x h* + *inner h x*)
by (*intro has-derivative-inner has-derivative-ident*)
have *2*: (λ*h*. *inner x h* + *inner h x*) = (λ*h*. *inner h* (*scaleR 2 x*))
by (*simp add*: *fun-eq-iff inner-commute*)
have *0* < *inner x x* **using** ‹*x* ≠ *0*› **by** *simp*
then have *3*: *DERIV sqrt* (*inner x x*) *:>* (*inverse* (*sqrt* (*inner x x*)) / *2*)
by (*rule DERIV-real-sqrt*)

have *4*: (*inverse* (*sqrt* (*inner x x*)) / *2*) $*_R$ *2* $*_R$ *x* = *sgn x*
by (*simp add*: *sgn-div-norm norm-eq-sqrt-inner*)
show *?thesis*
unfolding *norm-eq-sqrt-inner*
apply (*rule GDERIV-subst* [*OF - 4*])
apply (*rule GDERIV-DERIV-compose* [**where** *g*=*sqrt* **and** *df*=*scaleR 2 x*])
apply (*subst gderiv-def*)
apply (*rule has-derivative-subst* [*OF - 2*])
apply (*rule 1*)
apply (*rule 3*)
done
qed

lemmas *has-derivative-norm* = *GDERIV-norm* [*unfolded gderiv-def*]

end

10 Additive group operations on product types

theory *Product-plus*
imports *Main*
begin

10.1 Operations

instantiation *prod* :: (*zero*, *zero*) *zero*
begin

definition *zero-prod-def*: *0* = (*0*, *0*)

instance ..
end

instantiation *prod* :: (*plus*, *plus*) *plus*
begin

definition *plus-prod-def*:
x + *y* = (*fst x* + *fst y*, *snd x* + *snd y*)

instance ..
end

instantiation *prod* :: (*minus*, *minus*) *minus*
begin

definition *minus-prod-def*:
x − *y* = (*fst x* − *fst y*, *snd x* − *snd y*)

instance ..

end

instantiation *prod* :: (*uminus*, *uminus*) *uminus*
begin

definition *uminus-prod-def*:
$- x = (- fst\ x, - snd\ x)$

instance ..
end

lemma *fst-zero* [*simp*]: $fst\ 0 = 0$
unfolding *zero-prod-def* **by** *simp*

lemma *snd-zero* [*simp*]: $snd\ 0 = 0$
unfolding *zero-prod-def* **by** *simp*

lemma *fst-add* [*simp*]: $fst\ (x + y) = fst\ x + fst\ y$
unfolding *plus-prod-def* **by** *simp*

lemma *snd-add* [*simp*]: $snd\ (x + y) = snd\ x + snd\ y$
unfolding *plus-prod-def* **by** *simp*

lemma *fst-diff* [*simp*]: $fst\ (x - y) = fst\ x - fst\ y$
unfolding *minus-prod-def* **by** *simp*

lemma *snd-diff* [*simp*]: $snd\ (x - y) = snd\ x - snd\ y$
unfolding *minus-prod-def* **by** *simp*

lemma *fst-uminus* [*simp*]: $fst\ (- x) = - fst\ x$
unfolding *uminus-prod-def* **by** *simp*

lemma *snd-uminus* [*simp*]: $snd\ (- x) = - snd\ x$
unfolding *uminus-prod-def* **by** *simp*

lemma *add-Pair* [*simp*]: $(a, b) + (c, d) = (a + c, b + d)$
unfolding *plus-prod-def* **by** *simp*

lemma *diff-Pair* [*simp*]: $(a, b) - (c, d) = (a - c, b - d)$
unfolding *minus-prod-def* **by** *simp*

lemma *uminus-Pair* [*simp*, *code*]: $- (a, b) = (- a, - b)$
unfolding *uminus-prod-def* **by** *simp*

10.2 Class instances

instance *prod* :: (*semigroup-add*, *semigroup-add*) *semigroup-add*
by *standard* (*simp add*: *prod-eq-iff add.assoc*)

instance *prod* :: (*ab-semigroup-add*, *ab-semigroup-add*) *ab-semigroup-add*
by *standard* (*simp add*: *prod-eq-iff add.commute*)

instance *prod* :: (*monoid-add*, *monoid-add*) *monoid-add*
by *standard* (*simp-all add*: *prod-eq-iff*)

instance *prod* :: (*comm-monoid-add*, *comm-monoid-add*) *comm-monoid-add*
by *standard* (*simp add*: *prod-eq-iff*)

instance *prod* :: (*cancel-semigroup-add*, *cancel-semigroup-add*) *cancel-semigroup-add*
by *standard* (*simp-all add*: *prod-eq-iff*)

instance *prod* :: (*cancel-ab-semigroup-add*, *cancel-ab-semigroup-add*) *cancel-ab-semigroup-add*
by *standard* (*simp-all add*: *prod-eq-iff diff-diff-eq*)

instance *prod* :: (*cancel-comm-monoid-add*, *cancel-comm-monoid-add*) *cancel-comm-monoid-add*
..

instance *prod* :: (*group-add*, *group-add*) *group-add*
by *standard* (*simp-all add*: *prod-eq-iff*)

instance *prod* :: (*ab-group-add*, *ab-group-add*) *ab-group-add*
by *standard* (*simp-all add*: *prod-eq-iff*)

lemma *fst-setsum*: *fst* $(\sum x{\in}A.\ f\ x) = (\sum x{\in}A.\ fst\ (f\ x))$
proof (*cases finite A*)
case *True*
then show *?thesis* **by** *induct simp-all*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *snd-setsum*: *snd* $(\sum x{\in}A.\ f\ x) = (\sum x{\in}A.\ snd\ (f\ x))$
proof (*cases finite A*)
case *True*
then show *?thesis* **by** *induct simp-all*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *setsum-prod*: $(\sum x{\in}A.\ (f\ x,\ g\ x)) = (\sum x{\in}A.\ f\ x,\ \sum x{\in}A.\ g\ x)$
proof (*cases finite A*)
case *True*
then show *?thesis* **by** *induct* (*simp-all add*: *zero-prod-def*)
next
case *False*
then show *?thesis* **by** (*simp add*: *zero-prod-def*)

qed

end

11 Cartesian Products as Vector Spaces

theory *Product-Vector*
imports *Inner-Product Product-plus*
begin

11.1 Product is a real vector space

instantiation *prod* :: (*real-vector*, *real-vector*) *real-vector*
begin

definition *scaleR-prod-def*:
scaleR r A = (*scaleR r* (*fst A*), *scaleR r* (*snd A*))

lemma *fst-scaleR* [*simp*]: *fst* (*scaleR r A*) = *scaleR r* (*fst A*)
unfolding *scaleR-prod-def* **by** *simp*

lemma *snd-scaleR* [*simp*]: *snd* (*scaleR r A*) = *scaleR r* (*snd A*)
unfolding *scaleR-prod-def* **by** *simp*

lemma *scaleR-Pair* [*simp*]: *scaleR r* (*a*, *b*) = (*scaleR r a*, *scaleR r b*)
unfolding *scaleR-prod-def* **by** *simp*

instance
proof
fix *a b* :: *real* **and** *x y* :: *'a* × *'b*
show *scaleR a* (*x* + *y*) = *scaleR a x* + *scaleR a y*
by (*simp add*: *prod-eq-iff scaleR-right-distrib*)
show *scaleR* (*a* + *b*) *x* = *scaleR a x* + *scaleR b x*
by (*simp add*: *prod-eq-iff scaleR-left-distrib*)
show *scaleR a* (*scaleR b x*) = *scaleR* (*a* ∗ *b*) *x*
by (*simp add*: *prod-eq-iff*)
show *scaleR 1 x* = *x*
by (*simp add*: *prod-eq-iff*)
qed

end

11.2 Product is a topological space

instantiation *prod* :: (*topological-space*, *topological-space*) *topological-space*
begin

definition *open-prod-def* [*code del*]:
open (*S* :: (*'a* × *'b*) *set*) ⟷

$(\forall\, x{\in}S.\ \exists\, A\ B.\ open\ A \wedge open\ B \wedge x \in A \times B \wedge A \times B \subseteq S)$

lemma *open-prod-elim*:
assumes *open S* **and** $x \in S$
obtains *A B* **where** *open A* **and** *open B* **and** $x \in A \times B$ **and** $A \times B \subseteq S$
using *assms* **unfolding** *open-prod-def* **by** *fast*

lemma *open-prod-intro*:
assumes $\bigwedge x.\ x \in S \Longrightarrow \exists\, A\ B.\ open\ A \wedge open\ B \wedge x \in A \times B \wedge A \times B \subseteq S$
shows *open S*
using *assms* **unfolding** *open-prod-def* **by** *fast*

instance
proof
show *open* (*UNIV* :: ('*a* × '*b*) *set*)
unfolding *open-prod-def* **by** *auto*
next
fix *S T* :: ('*a* × '*b*) *set*
assume *open S open T*
show *open* (*S* ∩ *T*)
proof (*rule open-prod-intro*)
fix *x* **assume** *x*: $x \in S \cap T$
from *x* **have** $x \in S$ **by** *simp*
obtain *Sa Sb* **where** *A*: *open Sa open Sb x* ∈ *Sa* × *Sb Sa* × *Sb* ⊆ *S*
using ‹*open S*› **and** ‹$x \in S$› **by** (*rule open-prod-elim*)
from *x* **have** $x \in T$ **by** *simp*
obtain *Ta Tb* **where** *B*: *open Ta open Tb x* ∈ *Ta* × *Tb Ta* × *Tb* ⊆ *T*
using ‹*open T*› **and** ‹$x \in T$› **by** (*rule open-prod-elim*)
let *?A* = *Sa* ∩ *Ta* **and** *?B* = *Sb* ∩ *Tb*
have *open ?A* ∧ *open ?B* ∧ *x* ∈ *?A* × *?B* ∧ *?A* × *?B* ⊆ *S* ∩ *T*
using *A B* **by** (*auto simp add*: *open-Int*)
thus ∃ *A B*. *open A* ∧ *open B* ∧ *x* ∈ *A* × *B* ∧ *A* × *B* ⊆ *S* ∩ *T*
by *fast*
qed
next
fix *K* :: ('*a* × '*b*) *set set*
assume ∀ *S*∈*K*. *open S* **thus** *open* (⋃ *K*)
unfolding *open-prod-def* **by** *fast*
qed

end

declare [[*code abort*: *open*::('*a*::*topological-space*∗'*b*::*topological-space*) *set* ⇒ *bool*]]

lemma *open-Times*: *open S* ⟹ *open T* ⟹ *open* (*S* × *T*)
unfolding *open-prod-def* **by** *auto*

lemma *fst-vimage-eq-Times*: *fst* −‘ *S* = *S* × *UNIV*
by *auto*

lemma *snd-vimage-eq-Times*: *snd* $-`$ $S = UNIV \times S$
by *auto*

lemma *open-vimage-fst*: *open* $S \Longrightarrow$ *open* (*fst* $-`$ S)
by (*simp add*: *fst-vimage-eq-Times open-Times*)

lemma *open-vimage-snd*: *open* $S \Longrightarrow$ *open* (*snd* $-`$ S)
by (*simp add*: *snd-vimage-eq-Times open-Times*)

lemma *closed-vimage-fst*: *closed* $S \Longrightarrow$ *closed* (*fst* $-`$ S)
unfolding *closed-open vimage-Compl* [*symmetric*]
by (*rule open-vimage-fst*)

lemma *closed-vimage-snd*: *closed* $S \Longrightarrow$ *closed* (*snd* $-`$ S)
unfolding *closed-open vimage-Compl* [*symmetric*]
by (*rule open-vimage-snd*)

lemma *closed-Times*: *closed* $S \Longrightarrow$ *closed* $T \Longrightarrow$ *closed* $(S \times T)$
proof −
  have $S \times T = (fst -` S) \cap (snd -` T)$ **by** *auto*
  thus *closed* $S \Longrightarrow$ *closed* $T \Longrightarrow$ *closed* $(S \times T)$
    by (*simp add*: *closed-vimage-fst closed-vimage-snd closed-Int*)
qed

lemma *subset-fst-imageI*: $A \times B \subseteq S \Longrightarrow y \in B \Longrightarrow A \subseteq fst ` S$
  unfolding *image-def subset-eq* **by** *force*

lemma *subset-snd-imageI*: $A \times B \subseteq S \Longrightarrow x \in A \Longrightarrow B \subseteq snd ` S$
  unfolding *image-def subset-eq* **by** *force*

lemma *open-image-fst*: **assumes** *open* S **shows** *open* (*fst* $` S$)
proof (*rule openI*)
  fix x **assume** $x \in fst ` S$
  then obtain y **where** $(x, y) \in S$ **by** *auto*
  then obtain A B **where** *open* A *open* B $x \in A$ $y \in B$ $A \times B \subseteq S$
    using ‹*open* S› **unfolding** *open-prod-def* **by** *auto*
  from ‹$A \times B \subseteq S$› ‹$y \in B$› **have** $A \subseteq fst ` S$ **by** (*rule subset-fst-imageI*)
  with ‹*open* A› ‹$x \in A$› **have** *open* $A \wedge x \in A \wedge A \subseteq fst ` S$ **by** *simp*
  then show $\exists T.$ *open* $T \wedge x \in T \wedge T \subseteq fst ` S$ **by** − (*rule exI*)
qed

lemma *open-image-snd*: **assumes** *open* S **shows** *open* (*snd* $` S$)
proof (*rule openI*)
  fix y **assume** $y \in snd ` S$
  then obtain x **where** $(x, y) \in S$ **by** *auto*
  then obtain A B **where** *open* A *open* B $x \in A$ $y \in B$ $A \times B \subseteq S$
    using ‹*open* S› **unfolding** *open-prod-def* **by** *auto*
  from ‹$A \times B \subseteq S$› ‹$x \in A$› **have** $B \subseteq snd ` S$ **by** (*rule subset-snd-imageI*)

with ‹*open B*› ‹*y ∈ B*› **have** *open B ∧ y ∈ B ∧ B ⊆ snd ‘ S* **by** *simp*
then show *∃ T. open T ∧ y ∈ T ∧ T ⊆ snd ‘ S* **by** *−* (*rule exI*)
qed

11.2.1 Continuity of operations

lemma *tendsto-fst* [*tendsto-intros*]:
assumes (*f* ⟶ *a*) *F*
shows ((*λx. fst* (*f x*)) ⟶ *fst a*) *F*
proof (*rule topological-tendstoI*)
fix *S* **assume** *open S* **and** *fst a ∈ S*
then have *open* (*fst −‘ S*) **and** *a ∈ fst −‘ S*
by (*simp-all add*: *open-vimage-fst*)
with *assms* **have** *eventually* (*λx. f x ∈ fst −‘ S*) *F*
by (*rule topological-tendstoD*)
then show *eventually* (*λx. fst* (*f x*) *∈ S*) *F*
by *simp*
qed

lemma *tendsto-snd* [*tendsto-intros*]:
assumes (*f* ⟶ *a*) *F*
shows ((*λx. snd* (*f x*)) ⟶ *snd a*) *F*
proof (*rule topological-tendstoI*)
fix *S* **assume** *open S* **and** *snd a ∈ S*
then have *open* (*snd −‘ S*) **and** *a ∈ snd −‘ S*
by (*simp-all add*: *open-vimage-snd*)
with *assms* **have** *eventually* (*λx. f x ∈ snd −‘ S*) *F*
by (*rule topological-tendstoD*)
then show *eventually* (*λx. snd* (*f x*) *∈ S*) *F*
by *simp*
qed

lemma *tendsto-Pair* [*tendsto-intros*]:
assumes (*f* ⟶ *a*) *F* **and** (*g* ⟶ *b*) *F*
shows ((*λx.* (*f x*, *g x*)) ⟶ (*a*, *b*)) *F*
proof (*rule topological-tendstoI*)
fix *S* **assume** *open S* **and** (*a*, *b*) *∈ S*
then obtain *A B* **where** *open A open B a ∈ A b ∈ B A × B ⊆ S*
unfolding *open-prod-def* **by** *fast*
have *eventually* (*λx. f x ∈ A*) *F*
using ‹(*f* ⟶ *a*) *F*› ‹*open A*› ‹*a ∈ A*›
by (*rule topological-tendstoD*)
moreover
have *eventually* (*λx. g x ∈ B*) *F*
using ‹(*g* ⟶ *b*) *F*› ‹*open B*› ‹*b ∈ B*›
by (*rule topological-tendstoD*)
ultimately
show *eventually* (*λx.* (*f x*, *g x*) *∈ S*) *F*
by (*rule eventually-elim2*)

(*simp add*: *subsetD* [*OF* ‹$A \times B \subseteq S$›])
qed

lemma *continuous-fst*[*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* (λx. *fst* (*f x*))
unfolding *continuous-def* **by** (*rule tendsto-fst*)

lemma *continuous-snd*[*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* (λx. *snd* (*f x*))
unfolding *continuous-def* **by** (*rule tendsto-snd*)

lemma *continuous-Pair*[*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F g* $\Longrightarrow$ *continuous F* (λx. (*f x*, *g x*))
unfolding *continuous-def* **by** (*rule tendsto-Pair*)

lemma *continuous-on-fst*[*continuous-intros*]: *continuous-on s f* $\Longrightarrow$ *continuous-on s* (λx. *fst* (*f x*))
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-fst*)

lemma *continuous-on-snd*[*continuous-intros*]: *continuous-on s f* $\Longrightarrow$ *continuous-on s* (λx. *snd* (*f x*))
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-snd*)

lemma *continuous-on-Pair*[*continuous-intros*]: *continuous-on s f* $\Longrightarrow$ *continuous-on s g* $\Longrightarrow$ *continuous-on s* (λx. (*f x*, *g x*))
unfolding *continuous-on-def* **by** (*auto intro*: *tendsto-Pair*)

lemma *continuous-on-swap*[*continuous-intros*]: *continuous-on A prod.swap*
by (*simp add*: *prod.swap-def continuous-on-fst continuous-on-snd continuous-on-Pair continuous-on-id*)

lemma *isCont-fst* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* (λx. *fst* (*f x*)) *a*
by (*fact continuous-fst*)

lemma *isCont-snd* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* (λx. *snd* (*f x*)) *a*
by (*fact continuous-snd*)

lemma *isCont-Pair* [*simp*]: $[\![$*isCont f a*; *isCont g a*$]\!] \Longrightarrow$ *isCont* (λx. (*f x*, *g x*)) *a*
by (*fact continuous-Pair*)

11.2.2 Separation axioms

instance *prod* :: (*t0-space*, *t0-space*) *t0-space*
proof
fix *x y* :: *'a* $\times$ *'b* **assume** $x \neq y$
hence *fst x* $\neq$ *fst y* $\vee$ *snd x* $\neq$ *snd y*
by (*simp add*: *prod-eq-iff*)
thus $\exists U.$ *open U* $\wedge$ $(x \in U) \neq (y \in U)$
by (*fast dest*: *t0-space elim*: *open-vimage-fst open-vimage-snd*)

qed

instance *prod* :: (*t1-space*, *t1-space*) *t1-space*
proof
  fix *x y* :: *'a* × *'b* **assume** *x* ≠ *y*
  hence *fst x* ≠ *fst y* ∨ *snd x* ≠ *snd y*
    by (*simp add*: *prod-eq-iff*)
  thus ∃ *U*. *open U* ∧ *x* ∈ *U* ∧ *y* ∉ *U*
    by (*fast dest*: *t1-space elim*: *open-vimage-fst open-vimage-snd*)
qed

instance *prod* :: (*t2-space*, *t2-space*) *t2-space*
proof
  fix *x y* :: *'a* × *'b* **assume** *x* ≠ *y*
  hence *fst x* ≠ *fst y* ∨ *snd x* ≠ *snd y*
    by (*simp add*: *prod-eq-iff*)
  thus ∃ *U V*. *open U* ∧ *open V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* ∩ *V* = {}
    by (*fast dest*: *hausdorff elim*: *open-vimage-fst open-vimage-snd*)
qed

lemma *isCont-swap*[*continuous-intros*]: *isCont prod.swap a*
  using *continuous-on-eq-continuous-within continuous-on-swap* **by** *blast*

11.3 Product is a metric space

instantiation *prod* :: (*metric-space*, *metric-space*) *dist*
begin

definition *dist-prod-def*[*code del*]:
  dist x y = *sqrt* ((*dist* (*fst x*) (*fst y*))^2 + (*dist* (*snd x*) (*snd y*))^2)

instance ..
end

instantiation *prod* :: (*metric-space*, *metric-space*) *uniformity-dist*
begin

definition [*code del*]:
  (*uniformity* :: ((*'a* × *'b*) × (*'a* × *'b*)) *filter*) =
    (*INF e*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

instance
  by *standard* (*rule uniformity-prod-def*)
end

declare *uniformity-Abort*[**where** *'a*=*'a* :: *metric-space* × *'b* :: *metric-space*, *code*]

instantiation *prod* :: (*metric-space*, *metric-space*) *metric-space*
begin

lemma *dist-Pair-Pair*: *dist* (*a*, *b*) (*c*, *d*) = *sqrt* ((*dist a c*)^2 + (*dist b d*)^2)
unfolding *dist-prod-def* **by** *simp*

lemma *dist-fst-le*: *dist* (*fst x*) (*fst y*) ≤ *dist x y*
unfolding *dist-prod-def* **by** (*rule real-sqrt-sum-squares-ge1*)

lemma *dist-snd-le*: *dist* (*snd x*) (*snd y*) ≤ *dist x y*
unfolding *dist-prod-def* **by** (*rule real-sqrt-sum-squares-ge2*)

instance
proof
fix *x y* :: *'a* × *'b*
show *dist x y* = *0* ⟷ *x* = *y*
unfolding *dist-prod-def prod-eq-iff* **by** *simp*
next
fix *x y z* :: *'a* × *'b*
show *dist x y* ≤ *dist x z* + *dist y z*
unfolding *dist-prod-def*
by (*intro order-trans* [*OF - real-sqrt-sum-squares-triangle-ineq*]
real-sqrt-le-mono add-mono power-mono dist-triangle2 zero-le-dist)
next
fix *S* :: (*'a* × *'b*) *set*
have ∗: *open S* ⟷ (∀ *x*∈*S*. ∃ *e*>*0*. ∀ *y*. *dist y x* < *e* ⟶ *y* ∈ *S*)
proof
assume *open S* **show** ∀ *x*∈*S*. ∃ *e*>*0*. ∀ *y*. *dist y x* < *e* ⟶ *y* ∈ *S*
proof
fix *x* **assume** *x* ∈ *S*
obtain *A B* **where** *open A open B x* ∈ *A* × *B A* × *B* ⊆ *S*
using ‹*open S*› **and** ‹*x* ∈ *S*› **by** (*rule open-prod-elim*)
obtain *r* **where** *r*: *0* < *r* ∀ *y*. *dist y* (*fst x*) < *r* ⟶ *y* ∈ *A*
using ‹*open A*› **and** ‹*x* ∈ *A* × *B*› **unfolding** *open-dist* **by** *auto*
obtain *s* **where** *s*: *0* < *s* ∀ *y*. *dist y* (*snd x*) < *s* ⟶ *y* ∈ *B*
using ‹*open B*› **and** ‹*x* ∈ *A* × *B*› **unfolding** *open-dist* **by** *auto*
let *?e* = *min r s*
have *0* < *?e* ∧ (∀ *y*. *dist y x* < *?e* ⟶ *y* ∈ *S*)
proof (*intro allI impI conjI*)
show *0* < *min r s* **by** (*simp add*: *r*(*1*) *s*(*1*))
next
fix *y* **assume** *dist y x* < *min r s*
hence *dist y x* < *r* **and** *dist y x* < *s*
by *simp-all*
hence *dist* (*fst y*) (*fst x*) < *r* **and** *dist* (*snd y*) (*snd x*) < *s*
by (*auto intro*: *le-less-trans dist-fst-le dist-snd-le*)
hence *fst y* ∈ *A* **and** *snd y* ∈ *B*
by (*simp-all add*: *r*(*2*) *s*(*2*))
hence *y* ∈ *A* × *B* **by** (*induct y*, *simp*)
with ‹*A* × *B* ⊆ *S*› **show** *y* ∈ *S* **..**
qed

```
    thus ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S ..
  qed
 next
  assume ∗: ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S show open S
  proof (rule open-prod-intro)
   fix x assume x ∈ S
   then obtain e where 0 < e and S: ∀ y. dist y x < e ⟶ y ∈ S
    using ∗ by fast
   def r ≡ e / sqrt 2 and s ≡ e / sqrt 2
   from ‹0 < e› have 0 < r and 0 < s
    unfolding r-def s-def by simp-all
   from ‹0 < e› have e = sqrt (r² + s²)
    unfolding r-def s-def by (simp add: power-divide)
   def A ≡ {y. dist (fst x) y < r} and B ≡ {y. dist (snd x) y < s}
   have open A and open B
    unfolding A-def B-def by (simp-all add: open-ball)
   moreover have x ∈ A × B
    unfolding A-def B-def mem-Times-iff
    using ‹0 < r› and ‹0 < s› by simp
   moreover have A × B ⊆ S
   proof (clarify)
    fix a b assume a ∈ A and b ∈ B
    hence dist a (fst x) < r and dist b (snd x) < s
     unfolding A-def B-def by (simp-all add: dist-commute)
    hence dist (a, b) x < e
     unfolding dist-prod-def ‹e = sqrt (r² + s²)›
     by (simp add: add-strict-mono power-strict-mono)
    thus (a, b) ∈ S
     by (simp add: S)
   qed
   ultimately show ∃ A B. open A ∧ open B ∧ x ∈ A × B ∧ A × B ⊆ S by
fast
  qed
 qed
 show open S = (∀ x∈S. ∀ F (x', y) in uniformity. x' = x ⟶ y ∈ S)
  unfolding ∗ eventually-uniformity-metric
  by (simp del: split-paired-All add: dist-prod-def dist-commute)
qed

end

declare [[code abort: dist::('a::metric-space∗'b::metric-space)⇒('a∗'b) ⇒ real]]

lemma Cauchy-fst: Cauchy X ⟹ Cauchy (λn. fst (X n))
 unfolding Cauchy-def by (fast elim: le-less-trans [OF dist-fst-le])

lemma Cauchy-snd: Cauchy X ⟹ Cauchy (λn. snd (X n))
 unfolding Cauchy-def by (fast elim: le-less-trans [OF dist-snd-le])
```

lemma *Cauchy-Pair*:
assumes *Cauchy X* **and** *Cauchy Y*
shows *Cauchy* (λ*n*. (*X n*, *Y n*))
proof (*rule metric-CauchyI*)
fix *r* :: *real* **assume** *0* < *r*
hence *0* < *r* / *sqrt 2* (**is** *0* < *?s*) **by** *simp*
obtain *M* **where** *M*: ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*X m*) (*X n*) < *?s*
using *metric-CauchyD* [*OF* ‹*Cauchy X*› ‹*0* < *?s*›] **..**
obtain *N* **where** *N*: ∀ *m*≥*N*. ∀ *n*≥*N*. *dist* (*Y m*) (*Y n*) < *?s*
using *metric-CauchyD* [*OF* ‹*Cauchy Y*› ‹*0* < *?s*›] **..**
have ∀ *m*≥*max M N*. ∀ *n*≥*max M N*. *dist* (*X m*, *Y m*) (*X n*, *Y n*) < *r*
using *M N* **by** (*simp add*: *real-sqrt-sum-squares-less dist-Pair-Pair*)
then show ∃ *n0*. ∀ *m*≥*n0*. ∀ *n*≥*n0*. *dist* (*X m*, *Y m*) (*X n*, *Y n*) < *r* **..**
qed

11.4 Product is a complete metric space

instance *prod* :: (*complete-space*, *complete-space*) *complete-space*
proof
fix *X* :: *nat* ⇒ *'a* × *'b* **assume** *Cauchy X*
have *1*: (λ*n*. *fst* (*X n*)) ⟶ *lim* (λ*n*. *fst* (*X n*))
using *Cauchy-fst* [*OF* ‹*Cauchy X*›]
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
have *2*: (λ*n*. *snd* (*X n*)) ⟶ *lim* (λ*n*. *snd* (*X n*))
using *Cauchy-snd* [*OF* ‹*Cauchy X*›]
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
have *X* ⟶ (*lim* (λ*n*. *fst* (*X n*)), *lim* (λ*n*. *snd* (*X n*)))
using *tendsto-Pair* [*OF 1 2*] **by** *simp*
then show *convergent X*
by (*rule convergentI*)
qed

11.5 Product is a normed vector space

instantiation *prod* :: (*real-normed-vector*, *real-normed-vector*) *real-normed-vector*
begin

definition *norm-prod-def*[*code del*]:
norm x = *sqrt* ((*norm* (*fst x*))² + (*norm* (*snd x*))²)

definition *sgn-prod-def*:
sgn (*x*::*'a* × *'b*) = *scaleR* (*inverse* (*norm x*)) *x*

lemma *norm-Pair*: *norm* (*a*, *b*) = *sqrt* ((*norm a*)² + (*norm b*)²)
unfolding *norm-prod-def* **by** *simp*

instance
proof
fix *r* :: *real* **and** *x y* :: *'a* × *'b*
show *norm x* = *0* ⟷ *x* = *0*

unfolding *norm-prod-def*
by (*simp add*: *prod-eq-iff*)
show *norm* (*x* + *y*) ≤ *norm x* + *norm y*
unfolding *norm-prod-def*
apply (*rule order-trans* [*OF - real-sqrt-sum-squares-triangle-ineq*])
apply (*simp add*: *add-mono power-mono norm-triangle-ineq*)
done
show *norm* (*scaleR r x*) = |*r*| * *norm x*
unfolding *norm-prod-def*
apply (*simp add*: *power-mult-distrib*)
apply (*simp add*: *distrib-left* [*symmetric*])
apply (*simp add*: *real-sqrt-mult-distrib*)
done
show *sgn x* = *scaleR* (*inverse* (*norm x*)) *x*
by (*rule sgn-prod-def*)
show *dist x y* = *norm* (*x* − *y*)
unfolding *dist-prod-def norm-prod-def*
by (*simp add*: *dist-norm*)
qed

end

declare [[*code abort*: *norm*::('*a*::*real-normed-vector* * '*b*::*real-normed-vector*) ⇒ *real*]]

instance *prod* :: (*banach*, *banach*) *banach* **..**

11.5.1 Pair operations are linear

lemma *bounded-linear-fst*: *bounded-linear fst*
using *fst-add fst-scaleR*
by (*rule bounded-linear-intro* [**where** *K*=*1*], *simp add*: *norm-prod-def*)

lemma *bounded-linear-snd*: *bounded-linear snd*
using *snd-add snd-scaleR*
by (*rule bounded-linear-intro* [**where** *K*=*1*], *simp add*: *norm-prod-def*)

lemmas *bounded-linear-fst-comp* = *bounded-linear-fst*[*THEN bounded-linear-compose*]

lemmas *bounded-linear-snd-comp* = *bounded-linear-snd*[*THEN bounded-linear-compose*]

lemma *bounded-linear-Pair*:
assumes *f*: *bounded-linear f*
assumes *g*: *bounded-linear g*
shows *bounded-linear* (λ*x*. (*f x*, *g x*))
proof
interpret *f*: *bounded-linear f* **by** *fact*
interpret *g*: *bounded-linear g* **by** *fact*
fix *x y* **and** *r* :: *real*
show (*f* (*x* + *y*), *g* (*x* + *y*)) = (*f x*, *g x*) + (*f y*, *g y*)

by (*simp add*: *f.add g.add*)
show (*f* (*r* $*_R$ *x*), *g* (*r* $*_R$ *x*)) = *r* $*_R$ (*f x*, *g x*)
by (*simp add*: *f.scaleR g.scaleR*)
obtain *Kf* **where** *0* < *Kf* **and** *norm-f*: $\bigwedge x.$ *norm* (*f x*) ≤ *norm x* ∗ *Kf*
using *f.pos-bounded* **by** *fast*
obtain *Kg* **where** *0* < *Kg* **and** *norm-g*: $\bigwedge x.$ *norm* (*g x*) ≤ *norm x* ∗ *Kg*
using *g.pos-bounded* **by** *fast*
have $\forall x.$ *norm* (*f x*, *g x*) ≤ *norm x* ∗ (*Kf* + *Kg*)
apply (*rule allI*)
apply (*simp add*: *norm-Pair*)
apply (*rule order-trans* [*OF sqrt-add-le-add-sqrt*], *simp*, *simp*)
apply (*simp add*: *distrib-left*)
apply (*rule add-mono* [*OF norm-f norm-g*])
done
then show $\exists K.$ $\forall x.$ *norm* (*f x*, *g x*) ≤ *norm x* ∗ *K* **..**
qed

11.5.2 Frechet derivatives involving pairs

lemma *has-derivative-Pair* [*derivative-intros*]:
assumes *f*: (*f has-derivative f′*) (*at x within s*) **and** *g*: (*g has-derivative g′*) (*at x within s*)
shows ((λ*x*. (*f x*, *g x*)) *has-derivative* (λ*h*. (*f′ h*, *g′ h*))) (*at x within s*)
proof (*rule has-derivativeI-sandwich*[*of 1*])
show *bounded-linear* (λ*h*. (*f′ h*, *g′ h*))
using *f g* **by** (*intro bounded-linear-Pair has-derivative-bounded-linear*)
let *?Rf* = λ*y*. *f y* − *f x* − *f′* (*y* − *x*)
let *?Rg* = λ*y*. *g y* − *g x* − *g′* (*y* − *x*)
let *?R* = λ*y*. ((*f y*, *g y*) − (*f x*, *g x*) − (*f′* (*y* − *x*), *g′* (*y* − *x*)))

show ((λ*y*. *norm* (*?Rf y*) / *norm* (*y* − *x*) + *norm* (*?Rg y*) / *norm* (*y* − *x*)) ⟶ *0*) (*at x within s*)
using *f g* **by** (*intro tendsto-add-zero*) (*auto simp*: *has-derivative-iff-norm*)

fix *y* :: *′a* **assume** *y* ≠ *x*
show *norm* (*?R y*) / *norm* (*y* − *x*) ≤ *norm* (*?Rf y*) / *norm* (*y* − *x*) + *norm* (*?Rg y*) / *norm* (*y* − *x*)
unfolding *add-divide-distrib* [*symmetric*]
by (*simp add*: *norm-Pair divide-right-mono order-trans* [*OF sqrt-add-le-add-sqrt*])
qed *simp*

lemmas *has-derivative-fst* [*derivative-intros*] = *bounded-linear.has-derivative* [*OF bounded-linear-fst*]
lemmas *has-derivative-snd* [*derivative-intros*] = *bounded-linear.has-derivative* [*OF bounded-linear-snd*]

lemma *has-derivative-split* [*derivative-intros*]:
((λ*p*. *f* (*fst p*) (*snd p*)) *has-derivative f′*) *F* ⟹ ((λ(*a*, *b*). *f a b*) *has-derivative f′*) *F*

unfolding *split-beta'* **.**

11.6 Product is an inner product space

instantiation *prod* :: (*real-inner*, *real-inner*) *real-inner*
begin

definition *inner-prod-def*:
inner x y = *inner* (*fst x*) (*fst y*) + *inner* (*snd x*) (*snd y*)

lemma *inner-Pair* [*simp*]: *inner* (*a*, *b*) (*c*, *d*) = *inner a c* + *inner b d*
unfolding *inner-prod-def* **by** *simp*

instance
proof
fix *r* :: *real*
fix *x y z* :: *'a*::*real-inner* × *'b*::*real-inner*
show *inner x y* = *inner y x*
unfolding *inner-prod-def*
by (*simp add*: *inner-commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner-prod-def*
by (*simp add*: *inner-add-left*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner-prod-def*
by (*simp add*: *distrib-left*)
show *0* ≤ *inner x x*
unfolding *inner-prod-def*
by (*intro add-nonneg-nonneg inner-ge-zero*)
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner-prod-def prod-eq-iff*
by (*simp add*: *add-nonneg-eq-0-iff*)
show *norm x* = *sqrt* (*inner x x*)
unfolding *norm-prod-def inner-prod-def*
by (*simp add*: *power2-norm-eq-inner*)
qed

end

lemma *inner-Pair-0*: *inner x* (*0*, *b*) = *inner* (*snd x*) *b inner x* (*a*, *0*) = *inner* (*fst x*) *a*
by (*cases x*, *simp*)+

lemma
fixes *x* :: *'a*::*real-normed-vector*
shows *norm-Pair1* [*simp*]: *norm* (*0*,*x*) = *norm x*
and *norm-Pair2* [*simp*]: *norm* (*x*,*0*) = *norm x*
by (*auto simp*: *norm-Pair*)

lemma *norm-commute*: *norm* (*x*,*y*) = *norm* (*y*,*x*)
by (*simp add*: *norm-Pair*)

lemma *norm-fst-le*: *norm x* ≤ *norm* (*x*,*y*)
by (*metis dist-fst-le fst-conv fst-zero norm-conv-dist*)

lemma *norm-snd-le*: *norm y* ≤ *norm* (*x*,*y*)
by (*metis dist-snd-le snd-conv snd-zero norm-conv-dist*)

end

12 Finite-Dimensional Inner Product Spaces

theory *Euclidean-Space*
imports
L2-Norm
~~/*src*/*HOL*/*Library*/*Inner-Product*
~~/*src*/*HOL*/*Library*/*Product-Vector*
begin

12.1 Type class of Euclidean spaces

class *euclidean-space* = *real-inner* +
fixes *Basis* :: *'a set*
assumes *nonempty-Basis* [*simp*]: *Basis* ≠ {}
assumes *finite-Basis* [*simp*]: *finite Basis*
assumes *inner-Basis*:
⟦*u* ∈ *Basis*; *v* ∈ *Basis*⟧ ⟹ *inner u v* = (*if u* = *v then 1 else 0*)
assumes *euclidean-all-zero-iff*:
(∀ *u*∈*Basis*. *inner x u* = *0*) ⟷ (*x* = *0*)

abbreviation *dimension* :: (*'a*::*euclidean-space*) *itself* ⇒ *nat* **where**
dimension TYPE(*'a*) ≡ *card* (*Basis* :: *'a set*)

syntax *-type-dimension* :: *type* => *nat* ((*1DIM*/(*1'*(*-'*))))

translations *DIM*(*'t*) == *CONST dimension* (*TYPE*(*'t*))

lemma (**in** *euclidean-space*) *norm-Basis*[*simp*]: *u* ∈ *Basis* ⟹ *norm u* = *1*
unfolding *norm-eq-sqrt-inner* **by** (*simp add*: *inner-Basis*)

lemma (**in** *euclidean-space*) *inner-same-Basis*[*simp*]: *u* ∈ *Basis* ⟹ *inner u u* = *1*
by (*simp add*: *inner-Basis*)

lemma (**in** *euclidean-space*) *inner-not-same-Basis*: *u* ∈ *Basis* ⟹ *v* ∈ *Basis* ⟹ *u* ≠ *v* ⟹ *inner u v* = *0*
by (*simp add*: *inner-Basis*)

lemma (**in** *euclidean-space*) *sgn-Basis*: $u \in Basis \Longrightarrow sgn\ u = u$
unfolding *sgn-div-norm* **by** (*simp add*: *scaleR-one*)

lemma (**in** *euclidean-space*) *Basis-zero* [*simp*]: $0 \notin Basis$
proof
assume $0 \in Basis$ **thus** *False*
using *inner-Basis* [*of 0 0*] **by** *simp*
qed

lemma (**in** *euclidean-space*) *nonzero-Basis*: $u \in Basis \Longrightarrow u \neq 0$
by *clarsimp*

lemma (**in** *euclidean-space*) *SOME-Basis*: $(SOME\ i.\ i \in Basis) \in Basis$
by (*metis ex-in-conv nonempty-Basis someI-ex*)

lemma (**in** *euclidean-space*) *inner-setsum-left-Basis*[*simp*]:
$b \in Basis \Longrightarrow inner\ (\sum i{\in}Basis.\ f\ i\ *_R\ i)\ b = f\ b$
by (*simp add*: *inner-setsum-left inner-Basis if-distrib comm-monoid-add-class.setsum.If-cases*)

lemma (**in** *euclidean-space*) *euclidean-eqI*:
assumes *b*: $\bigwedge b.\ b \in Basis \Longrightarrow inner\ x\ b = inner\ y\ b$ **shows** $x = y$
proof −
from *b* **have** $\forall\, b{\in}Basis.\ inner\ (x - y)\ b = 0$
by (*simp add*: *inner-diff-left*)
then show $x = y$
by (*simp add*: *euclidean-all-zero-iff*)
qed

lemma (**in** *euclidean-space*) *euclidean-eq-iff*:
$x = y \longleftrightarrow (\forall\, b{\in}Basis.\ inner\ x\ b = inner\ y\ b)$
by (*auto intro*: *euclidean-eqI*)

lemma (**in** *euclidean-space*) *euclidean-representation-setsum*:
$(\sum i{\in}Basis.\ f\ i\ *_R\ i) = b \longleftrightarrow (\forall\, i{\in}Basis.\ f\ i = inner\ b\ i)$
by (*subst euclidean-eq-iff*) *simp*

lemma (**in** *euclidean-space*) *euclidean-representation-setsum'*:
$b = (\sum i{\in}Basis.\ f\ i\ *_R\ i) \longleftrightarrow (\forall\, i{\in}Basis.\ f\ i = inner\ b\ i)$
by (*auto simp add*: *euclidean-representation-setsum*[*symmetric*])

lemma (**in** *euclidean-space*) *euclidean-representation*: $(\sum b{\in}Basis.\ inner\ x\ b\ *_R\ b) = x$
unfolding *euclidean-representation-setsum* **by** *simp*

lemma (**in** *euclidean-space*) *choice-Basis-iff*:
fixes $P :: 'a \Rightarrow real \Rightarrow bool$
shows $(\forall\, i{\in}Basis.\ \exists\, x.\ P\ i\ x) \longleftrightarrow (\exists\, x.\ \forall\, i{\in}Basis.\ P\ i\ (inner\ x\ i))$
unfolding *bchoice-iff*
proof *safe*

fix f **assume** $\forall i{\in}Basis.\ P\ i\ (f\ i)$
then show $\exists x.\ \forall i{\in}Basis.\ P\ i\ (inner\ x\ i)$
by (*auto intro*!: *exI*[*of* - $\sum i{\in}Basis.\ f\ i *_R i$])
qed *auto*

lemma (**in** *euclidean-space*) *euclidean-representation-setsum-fun*:
$(\lambda x.\ \sum b{\in}Basis.\ inner\ (f\ x)\ b *_R b) = f$
by (*rule ext*) (*simp add*: *euclidean-representation-setsum*)

lemma *euclidean-isCont*:
assumes $\bigwedge b.\ b \in Basis \Longrightarrow isCont\ (\lambda x.\ (inner\ (f\ x)\ b) *_R b)\ x$
shows *isCont f x*
apply (*subst euclidean-representation-setsum-fun* [*symmetric*])
apply (*rule isCont-setsum*)
apply (*blast intro*: *assms*)
done

lemma *DIM-positive*: $0 < DIM('a{::}euclidean\text{-}space)$
by (*simp add*: *card-gt-0-iff*)

12.2 Subclass relationships

instance *euclidean-space* $\subseteq$ *perfect-space*
proof
fix $x :: 'a$ **show** $\neg$ *open* $\{x\}$
proof
assume *open* $\{x\}$
then obtain e **where** $0 < e$ **and** e: $\forall y.\ dist\ y\ x < e \longrightarrow y = x$
unfolding *open-dist* **by** *fast*
def $y \equiv x + scaleR\ (e/2)\ (SOME\ b.\ b \in Basis)$
have [*simp*]: $(SOME\ b.\ b \in Basis) \in Basis$
by (*rule someI-ex*) (*auto simp*: *ex-in-conv*)
from ‹$0 < e$› **have** $y \neq x$
unfolding *y-def* **by** (*auto intro*!: *nonzero-Basis*)
from ‹$0 < e$› **have** $dist\ y\ x < e$
unfolding *y-def* **by** (*simp add*: *dist-norm*)
from ‹$y \neq x$› **and** ‹$dist\ y\ x < e$› **show** *False*
using e **by** *simp*
qed
qed

12.3 Class instances

12.3.1 Type *real*

instantiation *real* :: *euclidean-space*
begin

definition
[*simp*]: $Basis = \{1{::}real\}$

instance
by *standard auto*

end

lemma *DIM-real*[*simp*]: *DIM*(*real*) = *1*
by *simp*

12.3.2 Type *complex*

instantiation *complex* :: *euclidean-space*
begin

definition *Basis-complex-def*:
Basis = {*1*, *ii*}

instance
by *standard* (*auto simp add*: *Basis-complex-def intro*: *complex-eqI split*: *split-if-asm*)

end

lemma *DIM-complex*[*simp*]: *DIM*(*complex*) = *2*
unfolding *Basis-complex-def* **by** *simp*

12.3.3 Type $'a \times 'b$

instantiation *prod* :: (*euclidean-space*, *euclidean-space*) *euclidean-space*
begin

definition
Basis = (λ*u*. (*u*, *0*)) ' *Basis* ∪ (λ*v*. (*0*, *v*)) ' *Basis*

lemma *setsum-Basis-prod-eq*:
fixes *f*::('*a**'*b*)⇒('*a**'*b*)
shows *setsum f Basis* = *setsum* (λ*i*. *f* (*i*, *0*)) *Basis* + *setsum* (λ*i*. *f* (*0*, *i*)) *Basis*
proof −
have *inj-on* (λ*u*. (*u*::'*a*, *0*::'*b*)) *Basis inj-on* (λ*u*. (*0*::'*a*, *u*::'*b*)) *Basis*
by (*auto intro*!: *inj-onI Pair-inject*)
thus *?thesis*
unfolding *Basis-prod-def*
by (*subst setsum.union-disjoint*) (*auto simp*: *Basis-prod-def setsum.reindex*)
qed

instance proof
show (*Basis* :: ('*a* × '*b*) *set*) ≠ {}
unfolding *Basis-prod-def* **by** *simp*
next
show *finite* (*Basis* :: ('*a* × '*b*) *set*)

unfolding *Basis-prod-def* **by** *simp*
next
fix *u v* :: *'a* × *'b*
assume *u* ∈ *Basis* **and** *v* ∈ *Basis*
thus *inner u v* = (*if u* = *v then 1 else 0*)
unfolding *Basis-prod-def inner-prod-def*
by (*auto simp add*: *inner-Basis split*: *split-if-asm*)
next
fix *x* :: *'a* × *'b*
show (∀ *u*∈*Basis*. *inner x u* = *0*) ⟷ *x* = *0*
unfolding *Basis-prod-def ball-Un ball-simps*
by (*simp add*: *inner-prod-def prod-eq-iff euclidean-all-zero-iff*)
qed

lemma *DIM-prod*[*simp*]: *DIM*(*'a* × *'b*) = *DIM*(*'a*) + *DIM*(*'b*)
unfolding *Basis-prod-def*
by (*subst card-Un-disjoint*) (*auto intro*!: *card-image arg-cong2*[**where** *f*=*op* +] *inj-onI*)

end

end

13 Elementary linear algebra on Euclidean spaces

theory *Linear-Algebra*
imports
Euclidean-Space
~~/*src*/*HOL*/*Library*/*Infinite-Set*
begin

lemma *cond-application-beta*: (*if b then f else g*) *x* = (*if b then f x else g x*)
by *auto*

notation *inner* (**infix** · *70*)

lemma *square-bound-lemma*:
fixes *x* :: *real*
shows *x* < (*1* + *x*) ∗ (*1* + *x*)
proof −
have (*x* + *1*/*2*)^2 + *3*/*4* > *0*
using *zero-le-power2*[*of x*+*1*/*2*] **by** *arith*
then show *?thesis*
by (*simp add*: *field-simps power2-eq-square*)
qed

lemma *square-continuous*:
fixes *e* :: *real*
shows *e* > *0* ⟹ ∃ *d*. *0* < *d* ∧ (∀ *y*. |*y* − *x*| < *d* ⟶ |*y* ∗ *y* − *x* ∗ *x*| < *e*)

using *isCont-power*[*OF continuous-ident*, *of x*, *unfolded isCont-def LIM-eq*, *rule-format*, *of e 2*]
by (*force simp add*: *power2-eq-square*)

Hence derive more interesting properties of the norm.

lemma *norm-eq-0-dot*: $norm\ x = 0 \longleftrightarrow x \cdot x = (0::real)$
by *simp*

lemma *norm-triangle-sub*:
fixes *x y* :: *'a*::*real-normed-vector*
shows $norm\ x \leq norm\ y + norm\ (x - y)$
using *norm-triangle-ineq*[*of y x* − *y*] **by** (*simp add*: *field-simps*)

lemma *norm-le*: $norm\ x \leq norm\ y \longleftrightarrow x \cdot x \leq y \cdot y$
by (*simp add*: *norm-eq-sqrt-inner*)

lemma *norm-lt*: $norm\ x < norm\ y \longleftrightarrow x \cdot x < y \cdot y$
by (*simp add*: *norm-eq-sqrt-inner*)

lemma *norm-eq*: $norm\ x = norm\ y \longleftrightarrow x \cdot x = y \cdot y$
apply (*subst order-eq-iff*)
apply (*auto simp*: *norm-le*)
done

lemma *norm-eq-1*: $norm\ x = 1 \longleftrightarrow x \cdot x = 1$
by (*simp add*: *norm-eq-sqrt-inner*)

Squaring equations and inequalities involving norms.

lemma *dot-square-norm*: $x \cdot x = (norm\ x)^2$
by (*simp only*: *power2-norm-eq-inner*)

lemma *norm-eq-square*: $norm\ x = a \longleftrightarrow 0 \leq a \wedge x \cdot x = a^2$
by (*auto simp add*: *norm-eq-sqrt-inner*)

lemma *norm-le-square*: $norm\ x \leq a \longleftrightarrow 0 \leq a \wedge x \cdot x \leq a^2$
apply (*simp add*: *dot-square-norm abs-le-square-iff*[*symmetric*])
using *norm-ge-zero*[*of x*]
apply *arith*
done

lemma *norm-ge-square*: $norm\ x \geq a \longleftrightarrow a \leq 0 \vee x \cdot x \geq a^2$
apply (*simp add*: *dot-square-norm abs-le-square-iff*[*symmetric*])
using *norm-ge-zero*[*of x*]
apply *arith*
done

lemma *norm-lt-square*: $norm\ x < a \longleftrightarrow 0 < a \wedge x \cdot x < a^2$
by (*metis not-le norm-ge-square*)

lemma *norm-gt-square*: $norm\ x > a \longleftrightarrow a < 0 \vee x \cdot x > a^2$
by (*metis norm-le-square not-less*)

Dot product in terms of the norm rather than conversely.

lemmas *inner-simps* = *inner-add-left inner-add-right inner-diff-right inner-diff-left inner-scaleR-left inner-scaleR-right*

lemma *dot-norm*: $x \cdot y = ((norm\ (x + y))^2 - (norm\ x)^2 - (norm\ y)^2)\ /\ 2$
unfolding *power2-norm-eq-inner inner-simps inner-commute* **by** *auto*

lemma *dot-norm-neg*: $x \cdot y = (((norm\ x)^2 + (norm\ y)^2) - (norm\ (x - y))^2)\ /\ 2$
unfolding *power2-norm-eq-inner inner-simps inner-commute*
by (*auto simp add*: *algebra-simps*)

Equality of vectors in terms of *op* $\cdot$ products.

lemma *vector-eq*: $x = y \longleftrightarrow x \cdot x = x \cdot y \wedge y \cdot y = x \cdot x$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs*
then show *?rhs* **by** *simp*
next
assume *?rhs*
then have $x \cdot x - x \cdot y = 0 \wedge x \cdot y - y \cdot y = 0$
by *simp*
then have $x \cdot (x - y) = 0 \wedge y \cdot (x - y) = 0$
by (*simp add*: *inner-diff inner-commute*)
then have $(x - y) \cdot (x - y) = 0$
by (*simp add*: *field-simps inner-diff inner-commute*)
then show $x = y$ **by** *simp*
qed

lemma *norm-triangle-half-r*:
$norm\ (y - x1) < e\ /\ 2 \Longrightarrow norm\ (y - x2) < e\ /\ 2 \Longrightarrow norm\ (x1 - x2) < e$
using *dist-triangle-half-r* **unfolding** *dist-norm*[*symmetric*] **by** *auto*

lemma *norm-triangle-half-l*:
assumes $norm\ (x - y) < e\ /\ 2$
and $norm\ (x' - y) < e\ /\ 2$
shows $norm\ (x - x') < e$
using *dist-triangle-half-l*[*OF assms*[*unfolded dist-norm*[*symmetric*]]]
unfolding *dist-norm*[*symmetric*] **.**

lemma *norm-triangle-le*: $norm\ x + norm\ y \leq e \Longrightarrow norm\ (x + y) \leq e$
by (*rule norm-triangle-ineq* [*THEN order-trans*])

lemma *norm-triangle-lt*: $norm\ x + norm\ y < e \Longrightarrow norm\ (x + y) < e$
by (*rule norm-triangle-ineq* [*THEN le-less-trans*])

lemma *setsum-clauses*:

shows *setsum f {} = 0*
and *finite S ⟹ setsum f (insert x S) = (if x ∈ S then setsum f S else f x + setsum f S)*
by (*auto simp add*: *insert-absorb*)

lemma *setsum-norm-le*:
fixes *f* :: *'a ⇒ 'b*::*real-normed-vector*
assumes *fg*: *∀ x ∈ S. norm (f x) ≤ g x*
shows *norm (setsum f S) ≤ setsum g S*
by (*rule order-trans* [*OF norm-setsum setsum-mono*]) (*simp add*: *fg*)

lemma *setsum-norm-bound*:
fixes *f* :: *'a ⇒ 'b*::*real-normed-vector*
assumes *K*: *∀ x ∈ S. norm (f x) ≤ K*
shows *norm (setsum f S) ≤ of-nat (card S) * K*
using *setsum-norm-le*[*OF K*] *setsum-constant*[*symmetric*]
by *simp*

lemma *setsum-group*:
assumes *fS*: *finite S* **and** *fT*: *finite T* **and** *fST*: *f ' S ⊆ T*
shows *setsum (λy. setsum g {x. x ∈ S ∧ f x = y}) T = setsum g S*
apply (*subst setsum-image-gen*[*OF fS*, *of g f*])
apply (*rule setsum.mono-neutral-right*[*OF fT fST*])
apply (*auto intro*: *setsum.neutral*)
done

lemma *vector-eq-ldot*: *(∀ x. x · y = x · z) ⟷ y = z*
proof
assume *∀ x. x · y = x · z*
then have *∀ x. x · (y − z) = 0*
by (*simp add*: *inner-diff*)
then have *(y − z) · (y − z) = 0* **..**
then show *y = z* **by** *simp*
qed *simp*

lemma *vector-eq-rdot*: *(∀ z. x · z = y · z) ⟷ x = y*
proof
assume *∀ z. x · z = y · z*
then have *∀ z. (x − y) · z = 0*
by (*simp add*: *inner-diff*)
then have *(x − y) · (x − y) = 0* **..**
then show *x = y* **by** *simp*
qed *simp*

13.1 Orthogonality.

context *real-inner*
begin

definition *orthogonal x y* $\longleftrightarrow$ *x* $\cdot$ *y* = *0*

lemma *orthogonal-clauses*:
orthogonal a 0
orthogonal a x $\Longrightarrow$ *orthogonal a* (*c* $*_R$ *x*)
orthogonal a x $\Longrightarrow$ *orthogonal a* (− *x*)
orthogonal a x $\Longrightarrow$ *orthogonal a y* $\Longrightarrow$ *orthogonal a* (*x* + *y*)
orthogonal a x $\Longrightarrow$ *orthogonal a y* $\Longrightarrow$ *orthogonal a* (*x* − *y*)
orthogonal 0 a
orthogonal x a $\Longrightarrow$ *orthogonal* (*c* $*_R$ *x*) *a*
orthogonal x a $\Longrightarrow$ *orthogonal* (− *x*) *a*
orthogonal x a $\Longrightarrow$ *orthogonal y a* $\Longrightarrow$ *orthogonal* (*x* + *y*) *a*
orthogonal x a $\Longrightarrow$ *orthogonal y a* $\Longrightarrow$ *orthogonal* (*x* − *y*) *a*
unfolding *orthogonal-def inner-add inner-diff* **by** *auto*

end

lemma *orthogonal-commute*: *orthogonal x y* $\longleftrightarrow$ *orthogonal y x*
by (*simp add*: *orthogonal-def inner-commute*)

13.2 Linear functions.

lemma *linear-iff*:
linear f $\longleftrightarrow$ ($\forall$ *x y*. *f* (*x* + *y*) = *f x* + *f y*) $\wedge$ ($\forall$ *c x*. *f* (*c* $*_R$ *x*) = *c* $*_R$ *f x*)
(**is** *linear f* $\longleftrightarrow$ *?rhs*)
proof
assume *linear f*
then interpret *f*: *linear f* **.**
show *?rhs* **by** (*simp add*: *f.add f.scaleR*)
next
assume *?rhs*
then show *linear f* **by** *unfold-locales simp-all*
qed

lemma *linear-compose-cmul*: *linear f* $\Longrightarrow$ *linear* (λ*x*. *c* $*_R$ *f x*)
by (*simp add*: *linear-iff algebra-simps*)

lemma *linear-compose-neg*: *linear f* $\Longrightarrow$ *linear* (λ*x*. − *f x*)
by (*simp add*: *linear-iff*)

lemma *linear-compose-add*: *linear f* $\Longrightarrow$ *linear g* $\Longrightarrow$ *linear* (λ*x*. *f x* + *g x*)
by (*simp add*: *linear-iff algebra-simps*)

lemma *linear-compose-sub*: *linear f* $\Longrightarrow$ *linear g* $\Longrightarrow$ *linear* (λ*x*. *f x* − *g x*)
by (*simp add*: *linear-iff algebra-simps*)

lemma *linear-compose*: *linear f* $\Longrightarrow$ *linear g* $\Longrightarrow$ *linear* (*g* $\circ$ *f*)
by (*simp add*: *linear-iff*)

lemma *linear-id*: *linear id*
by (*simp add*: *linear-iff id-def*)

lemma *linear-zero*: *linear* (λ*x*. *0*)
by (*simp add*: *linear-iff*)

lemma *linear-compose-setsum*:
assumes *lS*: ∀ *a* ∈ *S*. *linear* (*f a*)
shows *linear* (λ*x*. *setsum* (λ*a*. *f a x*) *S*)
proof (*cases finite S*)
case *True*
then show *?thesis*
using *lS* **by** *induct* (*simp-all add*: *linear-zero linear-compose-add*)
next
case *False*
then show *?thesis*
by (*simp add*: *linear-zero*)
qed

lemma *linear-0*: *linear f* ⟹ *f 0* = *0*
unfolding *linear-iff*
apply *clarsimp*
apply (*erule allE*[**where** *x*=*0*::*'a*])
apply *simp*
done

lemma *linear-cmul*: *linear f* ⟹ *f* (*c* *$_R$ *x*) = *c* *$_R$ *f x*
by (*rule linear.scaleR*)

lemma *linear-neg*: *linear f* ⟹ *f* (− *x*) = − *f x*
using *linear-cmul* [**where** *c*=−*1*] **by** *simp*

lemma *linear-add*: *linear f* ⟹ *f* (*x* + *y*) = *f x* + *f y*
by (*metis linear-iff*)

lemma *linear-sub*: *linear f* ⟹ *f* (*x* − *y*) = *f x* − *f y*
using *linear-add* [*of f x* − *y*] **by** (*simp add*: *linear-neg*)

lemma *linear-setsum*:
assumes *f*: *linear f*
shows *f* (*setsum g S*) = *setsum* (*f* ∘ *g*) *S*
proof (*cases finite S*)
case *True*
then show *?thesis*
by *induct* (*simp-all add*: *linear-0* [*OF f*] *linear-add* [*OF f*])
next
case *False*
then show *?thesis*
by (*simp add*: *linear-0* [*OF f*])

qed

lemma *linear-setsum-mul*:
assumes *lin*: *linear f*
shows *f* (*setsum* (λ*i*. *c i* ∗$_R$ *v i*) *S*) = *setsum* (λ*i*. *c i* ∗$_R$ *f* (*v i*)) *S*
using *linear-setsum*[*OF lin*, *of* λ*i*. *c i* ∗$_R$ *v i* , *unfolded o-def*] *linear-cmul*[*OF lin*]
by *simp*

lemma *linear-injective-0*:
assumes *lin*: *linear f*
shows *inj f* ⟷ (∀ *x*. *f x* = *0* ⟶ *x* = *0*)
proof −
have *inj f* ⟷ (∀ *x y*. *f x* = *f y* ⟶ *x* = *y*)
by (*simp add*: *inj-on-def*)
also have ... ⟷ (∀ *x y*. *f x* − *f y* = *0* ⟶ *x* − *y* = *0*)
by *simp*
also have ... ⟷ (∀ *x y*. *f* (*x* − *y*) = *0* ⟶ *x* − *y* = *0*)
by (*simp add*: *linear-sub*[*OF lin*])
also have ... ⟷ (∀ *x*. *f x* = *0* ⟶ *x* = *0*)
by *auto*
finally show *?thesis* **.**
qed

lemma *linear-scaleR* [*simp*]: *linear* (λ*x*. *scaleR c x*)
by (*simp add*: *linear-iff scaleR-add-right*)

lemma *linear-scaleR-left* [*simp*]: *linear* (λ*r*. *scaleR r x*)
by (*simp add*: *linear-iff scaleR-add-left*)

lemma *injective-scaleR*: *c* ≠ *0* ⟹ *inj* (λ*x*::'*a*::*real-vector*. *scaleR c x*)
by (*simp add*: *inj-on-def*)

lemma *linear-add-cmul*:
assumes *linear f*
shows *f* (*a* ∗$_R$ *x* + *b* ∗$_R$ *y*) = *a* ∗$_R$ *f x* + *b* ∗$_R$ *f y*
using *linear-add*[*of f*] *linear-cmul*[*of f*] *assms* **by** *simp*

lemma *linear-componentwise*:
fixes *f*:: '*a*::*euclidean-space* ⇒ '*b*::*real-inner*
assumes *lf*: *linear f*
shows (*f x*) ∙ *j* = (∑ *i*∈*Basis*. (*x*∙*i*) ∗ (*f i*∙*j*)) (**is** *?lhs* = *?rhs*)
proof −
have *?rhs* = (∑ *i*∈*Basis*. (*x*∙*i*) ∗$_R$ (*f i*))∙*j*
by (*simp add*: *inner-setsum-left*)
then show *?thesis*
unfolding *linear-setsum-mul*[*OF lf*, *symmetric*]
unfolding *euclidean-representation* **..**
qed

13.3 Bilinear functions.

definition *bilinear f* $\longleftrightarrow$ $(\forall x.\ linear\ (\lambda y.\ f\ x\ y)) \wedge (\forall y.\ linear\ (\lambda x.\ f\ x\ y))$

lemma *bilinear-ladd*: *bilinear h* $\Longrightarrow$ $h\ (x + y)\ z = h\ x\ z + h\ y\ z$
by (*simp add*: *bilinear-def linear-iff*)

lemma *bilinear-radd*: *bilinear h* $\Longrightarrow$ $h\ x\ (y + z) = h\ x\ y + h\ x\ z$
by (*simp add*: *bilinear-def linear-iff*)

lemma *bilinear-lmul*: *bilinear h* $\Longrightarrow$ $h\ (c *_R x)\ y = c *_R h\ x\ y$
by (*simp add*: *bilinear-def linear-iff*)

lemma *bilinear-rmul*: *bilinear h* $\Longrightarrow$ $h\ x\ (c *_R y) = c *_R h\ x\ y$
by (*simp add*: *bilinear-def linear-iff*)

lemma *bilinear-lneg*: *bilinear h* $\Longrightarrow$ $h\ (-\ x)\ y = -\ h\ x\ y$
by (*drule bilinear-lmul* [*of - - 1*]) *simp*

lemma *bilinear-rneg*: *bilinear h* $\Longrightarrow$ $h\ x\ (-\ y) = -\ h\ x\ y$
by (*drule bilinear-rmul* [*of - - - 1*]) *simp*

lemma (**in** *ab-group-add*) *eq-add-iff*: $x = x + y \longleftrightarrow y = 0$
using *add-left-imp-eq*[*of x y 0*] **by** *auto*

lemma *bilinear-lzero*:
assumes *bilinear h*
shows $h\ 0\ x = 0$
using *bilinear-ladd* [*OF assms*, *of 0 0 x*] **by** (*simp add*: *eq-add-iff field-simps*)

lemma *bilinear-rzero*:
assumes *bilinear h*
shows $h\ x\ 0 = 0$
using *bilinear-radd* [*OF assms*, *of x 0 0*] **by** (*simp add*: *eq-add-iff field-simps*)

lemma *bilinear-lsub*: *bilinear h* $\Longrightarrow$ $h\ (x - y)\ z = h\ x\ z - h\ y\ z$
using *bilinear-ladd* [*of h x - y*] **by** (*simp add*: *bilinear-lneg*)

lemma *bilinear-rsub*: *bilinear h* $\Longrightarrow$ $h\ z\ (x - y) = h\ z\ x - h\ z\ y$
using *bilinear-radd* [*of h - x - y*] **by** (*simp add*: *bilinear-rneg*)

lemma *bilinear-setsum*:
assumes *bh*: *bilinear h*
and *fS*: *finite S*
and *fT*: *finite T*
shows $h\ (setsum\ f\ S)\ (setsum\ g\ T) = setsum\ (\lambda(i,j).\ h\ (f\ i)\ (g\ j))\ (S \times T)$
proof −
have $h\ (setsum\ f\ S)\ (setsum\ g\ T) = setsum\ (\lambda x.\ h\ (f\ x)\ (setsum\ g\ T))\ S$
apply (*rule linear-setsum*[*unfolded o-def*])
using *bh fS*

apply (*auto simp add*: *bilinear-def*)
done
also have ... = *setsum* (λ*x*. *setsum* (λ*y*. *h* (*f x*) (*g y*)) *T*) *S*
apply (*rule setsum.cong*, *simp*)
apply (*rule linear-setsum*[*unfolded o-def*])
using *bh fT*
apply (*auto simp add*: *bilinear-def*)
done
finally show *?thesis*
unfolding *setsum.cartesian-product* .
qed

13.4 Adjoints.

definition *adjoint f* = (*SOME f'*. ∀ *x y*. *f x* · *y* = *x* · *f' y*)

lemma *adjoint-unique*:
assumes ∀ *x y*. *inner* (*f x*) *y* = *inner x* (*g y*)
shows *adjoint f* = *g*
unfolding *adjoint-def*
proof (*rule some-equality*)
show ∀ *x y*. *inner* (*f x*) *y* = *inner x* (*g y*)
by (*rule assms*)
next
fix *h*
assume ∀ *x y*. *inner* (*f x*) *y* = *inner x* (*h y*)
then have ∀ *x y*. *inner x* (*g y*) = *inner x* (*h y*)
using *assms* **by** *simp*
then have ∀ *x y*. *inner x* (*g y* − *h y*) = *0*
by (*simp add*: *inner-diff-right*)
then have ∀ *y*. *inner* (*g y* − *h y*) (*g y* − *h y*) = *0*
by *simp*
then have ∀ *y*. *h y* = *g y*
by *simp*
then show *h* = *g* **by** (*simp add*: *ext*)
qed

TODO: The following lemmas about adjoints should hold for any Hilbert space (i.e. complete inner product space). (see http://en.wikipedia.org/wiki/Hermitian_adjoint)

lemma *adjoint-works*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'m*::*euclidean-space*
assumes *lf*: *linear f*
shows *x* · *adjoint f y* = *f x* · *y*
proof −
have ∀ *y*. ∃ *w*. ∀ *x*. *f x* · *y* = *x* · *w*
proof (*intro allI exI*)
fix *y* :: *'m* **and** *x*
let *?w* = (∑ *i*∈*Basis*. (*f i* · *y*) ∗$_R$ *i*) :: *'n*

```
    have f x · y = f (∑ i∈Basis. (x · i) *R i) · y
      by (simp add: euclidean-representation)
    also have ... = (∑ i∈Basis. (x · i) *R f i) · y
      unfolding linear-setsum[OF lf]
      by (simp add: linear-cmul[OF lf])
    finally show f x · y = x · ?w
      by (simp add: inner-setsum-left inner-setsum-right mult.commute)
  qed
  then show ?thesis
    unfolding adjoint-def choice-iff
    by (intro someI2-ex[where Q=λf'. x · f' y = f x · y]) auto
qed

lemma adjoint-clauses:
  fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
  assumes lf: linear f
  shows x · adjoint f y = f x · y
    and adjoint f y · x = y · f x
  by (simp-all add: adjoint-works[OF lf] inner-commute)

lemma adjoint-linear:
  fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
  assumes lf: linear f
  shows linear (adjoint f)
  by (simp add: lf linear-iff euclidean-eq-iff[where 'a='n] euclidean-eq-iff[where
'a='m]
    adjoint-clauses[OF lf] inner-distrib)

lemma adjoint-adjoint:
  fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
  assumes lf: linear f
  shows adjoint (adjoint f) = f
  by (rule adjoint-unique, simp add: adjoint-clauses [OF lf])
```

13.5 Interlude: Some properties of real sets

```
lemma seq-mono-lemma:
  assumes ∀(n::nat) ≥ m. (d n :: real) < e n
    and ∀n ≥ m. e n ≤ e m
  shows ∀n ≥ m. d n < e m
  using assms
  apply auto
  apply (erule-tac x=n in allE)
  apply (erule-tac x=n in allE)
  apply auto
  done

lemma infinite-enumerate:
  assumes fS: infinite S
```

shows $\exists r.\ subseq\ r \wedge (\forall n.\ r\ n \in S)$
unfolding *subseq-def*
using *enumerate-in-set*[*OF fS*] *enumerate-mono*[*of - - S*] *fS* **by** *auto*

lemma *approachable-lt-le*: $(\exists (d::real) > 0.\ \forall x.\ f\ x < d \longrightarrow P\ x) \longleftrightarrow (\exists d{>}0.\ \forall x.\ f\ x \leq d \longrightarrow P\ x)$
apply *auto*
apply (*rule-tac x=d/2* **in** *exI*)
apply *auto*
done

lemma *approachable-lt-le2*: — like the above, but pushes aside an extra formula
$(\exists (d::real) > 0.\ \forall x.\ Q\ x \longrightarrow f\ x < d \longrightarrow P\ x) \longleftrightarrow (\exists d{>}0.\ \forall x.\ f\ x \leq d \longrightarrow Q\ x \longrightarrow P\ x)$
apply *auto*
apply (*rule-tac x=d/2* **in** *exI*, *auto*)
done

lemma *triangle-lemma*:
fixes $x\ y\ z :: real$
assumes x: $0 \leq x$
and y: $0 \leq y$
and z: $0 \leq z$
and xy: $x^2 \leq y^2 + z^2$
shows $x \leq y + z$
proof −
have $y^2 + z^2 \leq y^2 + 2 * y * z + z^2$
using $z\ y$ **by** *simp*
with *xy* **have** *th*: $x^2 \leq (y + z)^2$
by (*simp add*: *power2-eq-square field-simps*)
from $y\ z$ **have** yz: $y + z \geq 0$
by *arith*
from *power2-le-imp-le*[*OF th yz*] **show** *?thesis* **.**
qed

13.6 A generic notion of "hull" (convex, affine, conic hull and closure).

definition *hull* :: ('*a set* ⇒ *bool*) ⇒ '*a set* ⇒ '*a set* (**infixl** *hull 75*)
where $S\ hull\ s = \bigcap \{t.\ S\ t \wedge s \subseteq t\}$

lemma *hull-same*: $S\ s \Longrightarrow S\ hull\ s = s$
unfolding *hull-def* **by** *auto*

lemma *hull-in*: $(\bigwedge T.\ Ball\ T\ S \Longrightarrow S\ (\bigcap T)) \Longrightarrow S\ (S\ hull\ s)$
unfolding *hull-def Ball-def* **by** *auto*

lemma *hull-eq*: $(\bigwedge T.\ Ball\ T\ S \Longrightarrow S\ (\bigcap T)) \Longrightarrow (S\ hull\ s) = s \longleftrightarrow S\ s$
using *hull-same*[*of S s*] *hull-in*[*of S s*] **by** *metis*

lemma *hull-hull*: $S\ hull\ (S\ hull\ s) = S\ hull\ s$
unfolding *hull-def* **by** *blast*

lemma *hull-subset*[*intro*]: $s \subseteq (S\ hull\ s)$
unfolding *hull-def* **by** *blast*

lemma *hull-mono*: $s \subseteq t \Longrightarrow (S\ hull\ s) \subseteq (S\ hull\ t)$
unfolding *hull-def* **by** *blast*

lemma *hull-antimono*: $\forall x.\ S\ x \longrightarrow T\ x \Longrightarrow (T\ hull\ s) \subseteq (S\ hull\ s)$
unfolding *hull-def* **by** *blast*

lemma *hull-minimal*: $s \subseteq t \Longrightarrow S\ t \Longrightarrow (S\ hull\ s) \subseteq t$
unfolding *hull-def* **by** *blast*

lemma *subset-hull*: $S\ t \Longrightarrow S\ hull\ s \subseteq t \longleftrightarrow s \subseteq t$
unfolding *hull-def* **by** *blast*

lemma *hull-UNIV*: $S\ hull\ UNIV = UNIV$
unfolding *hull-def* **by** *auto*

lemma *hull-unique*: $s \subseteq t \Longrightarrow S\ t \Longrightarrow (\bigwedge t'.\ s \subseteq t' \Longrightarrow S\ t' \Longrightarrow t \subseteq t') \Longrightarrow (S\ hull\ s = t)$
unfolding *hull-def* **by** *auto*

lemma *hull-induct*: $(\bigwedge x.\ x \in S \Longrightarrow P\ x) \Longrightarrow Q\ \{x.\ P\ x\} \Longrightarrow \forall x \in Q\ hull\ S.\ P\ x$
using *hull-minimal*[*of S* $\{x.\ P\ x\}$ *Q*]
by (*auto simp add*: *subset-eq*)

lemma *hull-inc*: $x \in S \Longrightarrow x \in P\ hull\ S$
by (*metis hull-subset subset-eq*)

lemma *hull-union-subset*: $(S\ hull\ s) \cup (S\ hull\ t) \subseteq (S\ hull\ (s \cup t))$
unfolding *Un-subset-iff* **by** (*metis hull-mono Un-upper1 Un-upper2*)

lemma *hull-union*:
assumes *T*: $\bigwedge T.\ Ball\ T\ S \Longrightarrow S\ (\bigcap T)$
shows $S\ hull\ (s \cup t) = S\ hull\ (S\ hull\ s \cup S\ hull\ t)$
apply *rule*
apply (*rule hull-mono*)
unfolding *Un-subset-iff*
apply (*metis hull-subset Un-upper1 Un-upper2 subset-trans*)
apply (*rule hull-minimal*)
apply (*metis hull-union-subset*)
apply (*metis hull-in T*)
done

lemma *hull-redundant-eq*: $a \in (S\ hull\ s) \longleftrightarrow S\ hull\ (insert\ a\ s) = S\ hull\ s$

unfolding *hull-def* **by** *blast*

lemma *hull-redundant*: $a \in (S\ hull\ s) \Longrightarrow S\ hull\ (insert\ a\ s) = S\ hull\ s$
by (*metis hull-redundant-eq*)

13.7 Archimedean properties and useful consequences

lemma *real-arch-simple*: $\exists\, n{::}nat.\ x \leq real\ n$
by (*rule ex-le-of-nat*)

lemma *real-arch-inv*: $0 < e \longleftrightarrow (\exists\, n{::}nat.\ n \neq 0 \wedge 0 < inverse\ (real\ n) \wedge inverse\ (real\ n) < e)$
using *reals-Archimedean*[*of e*] *less-trans*[*of 0 1 / real n e* **for** *n*::*nat*]
by (*auto simp add*: *field-simps cong*: *conj-cong simp del*: *of-nat-Suc*)

Bernoulli's inequality

lemma *Bernoulli-inequality*:
fixes x :: *real*
assumes $-1 \leq x$
shows $1 + n * x \leq (1 + x)\ \hat{}\ n$
proof (*induct n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
have $1 + Suc\ n * x \leq 1 + (Suc\ n) * x + n * x\hat{}2$
by (*simp add*: *algebra-simps*)
also have $... = (1 + x) * (1 + n * x)$
by (*auto simp*: *power2-eq-square algebra-simps of-nat-Suc*)
also have $... \leq (1 + x)\ \hat{}\ Suc\ n$
using *Suc.hyps assms mult-left-mono* **by** *fastforce*
finally show *?case* **.**
qed

lemma *Bernoulli-inequality-even*:
fixes x :: *real*
assumes *even n*
shows $1 + n * x \leq (1 + x)\ \hat{}\ n$
proof (*cases* $-1 \leq x \vee n{=}0$)
case *True*
then show *?thesis*
by (*auto simp*: *Bernoulli-inequality*)
next
case *False*
then have $real\ n \geq 1$
by *simp*
with *False* **have** $n * x \leq -1$
by (*metis linear minus-zero mult.commute mult.left-neutral mult-left-mono-neg neg-le-iff-le order-trans zero-le-one*)

```
  then have 1 + n * x ≤ 0
    by auto
  also have ... ≤ (1 + x) ^ n
    using assms
    using zero-le-even-power by blast
  finally show ?thesis .
qed

lemma real-arch-pow:
  fixes x :: real
  assumes x: 1 < x
  shows ∃ n. y < x^n
proof −
  from x have x0: x − 1 > 0
    by arith
  from reals-Archimedean3[OF x0, rule-format, of y]
  obtain n :: nat where n: y < real n * (x − 1) by metis
  from x0 have x00: x− 1 ≥ −1 by arith
  from Bernoulli-inequality[OF x00, of n] n
  have y < x^n by auto
  then show ?thesis by metis
qed

lemma real-arch-pow2:
  fixes x :: real
  shows ∃ n. x < 2^ n
  using real-arch-pow[of 2 x] by simp

lemma real-arch-pow-inv:
  fixes x y :: real
  assumes y: y > 0
    and x1: x < 1
  shows ∃ n. x^n < y
proof (cases x > 0)
  case True
  with x1 have ix: 1 < 1/x by (simp add: field-simps)
  from real-arch-pow[OF ix, of 1/y]
  obtain n where n: 1/y < (1/x)^n by blast
  then show ?thesis using y ‹x > 0› 
    by (auto simp add: field-simps)
next
  case False
  with y x1 show ?thesis
    apply auto
    apply (rule exI[where x=1])
    apply auto
    done
qed
```

lemma *forall-pos-mono*:
(⋀*d e*::*real*. *d* < *e* ⟹ *P d* ⟹ *P e*) ⟹
(⋀*n*::*nat*. *n* ≠ *0* ⟹ *P* (*inverse* (*real n*))) ⟹ (⋀*e*. *0* < *e* ⟹ *P e*)
by (*metis real-arch-inv*)

lemma *forall-pos-mono-1*:
(⋀*d e*::*real*. *d* < *e* ⟹ *P d* ⟹ *P e*) ⟹
(⋀*n*. *P* (*inverse* (*real* (*Suc n*)))) ⟹ *0* < *e* ⟹ *P e*
apply (*rule forall-pos-mono*)
apply *auto*
apply (*metis Suc-pred of-nat-Suc*)
done

lemma *real-archimedian-rdiv-eq-0*:
assumes *x0*: *x* ≥ *0*
and *c*: *c* ≥ *0*
and *xc*: ∀(*m*::*nat*) > *0*. *real m* ∗ *x* ≤ *c*
shows *x* = *0*
proof (*rule ccontr*)
assume *x* ≠ *0*
with *x0* **have** *xp*: *x* > *0* **by** *arith*
from *reals-Archimedean3*[*OF xp*, *rule-format*, *of c*]
obtain *n* :: *nat* **where** *n*: *c* < *real n* ∗ *x*
by *blast*
with *xc*[*rule-format*, *of n*] **have** *n* = *0*
by *arith*
with *n c* **show** *False*
by *simp*
qed

13.8 A bit of linear algebra.

definition (**in** *real-vector*) *subspace* :: *'a set* ⇒ *bool*
where *subspace S* ⟷ *0* ∈ *S* ∧ (∀ *x* ∈ *S*. ∀ *y* ∈ *S*. *x* + *y* ∈ *S*) ∧ (∀ *c*. ∀ *x* ∈ *S*. *c* ∗*R* *x* ∈ *S*)

definition (**in** *real-vector*) *span S* = (*subspace hull S*)
definition (**in** *real-vector*) *dependent S* ⟷ (∃ *a* ∈ *S*. *a* ∈ *span* (*S* − {*a*}))
abbreviation (**in** *real-vector*) *independent s* ≡ ¬ *dependent s*

Closure properties of subspaces.

lemma *subspace-UNIV*[*simp*]: *subspace UNIV*
by (*simp add*: *subspace-def*)

lemma (**in** *real-vector*) *subspace-0*: *subspace S* ⟹ *0* ∈ *S*
by (*metis subspace-def*)

lemma (**in** *real-vector*) *subspace-add*: *subspace S* ⟹ *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *x* + *y* ∈ *S*

by (*metis subspace-def*)

lemma (**in** *real-vector*) *subspace-mul*: *subspace S* $\Longrightarrow$ $x \in S \Longrightarrow c *_R x \in S$
by (*metis subspace-def*)

lemma *subspace-neg*: *subspace S* $\Longrightarrow$ $x \in S \Longrightarrow - x \in S$
by (*metis scaleR-minus1-left subspace-mul*)

lemma *subspace-sub*: *subspace S* $\Longrightarrow$ $x \in S \Longrightarrow y \in S \Longrightarrow x - y \in S$
using *subspace-add* [*of S x* $-$ *y*] **by** (*simp add*: *subspace-neg*)

lemma (**in** *real-vector*) *subspace-setsum*:
assumes *sA*: *subspace A*
and *f*: $\forall x{\in}B.\ f\ x \in A$
shows *setsum f B* $\in A$
proof (*cases finite B*)
case *True*
then show *?thesis*
using *f* **by** *induct* (*simp-all add*: *subspace-0* [*OF sA*] *subspace-add* [*OF sA*])
qed (*simp add*: *subspace-0* [*OF sA*])

lemma *subspace-linear-image*:
assumes *lf*: *linear f*
and *sS*: *subspace S*
shows *subspace* (*f* ‘ *S*)
using *lf sS linear-0*[*OF lf*]
unfolding *linear-iff subspace-def*
apply (*auto simp add*: *image-iff*)
apply (*rule-tac x*=*x* + *y* **in** *bexI*)
apply *auto*
apply (*rule-tac x*=*c* $*_R$ *x* **in** *bexI*)
apply *auto*
done

lemma *subspace-linear-vimage*: *linear f* $\Longrightarrow$ *subspace S* $\Longrightarrow$ *subspace* (*f* $-$‘ *S*)
by (*auto simp add*: *subspace-def linear-iff linear-0*[*of f*])

lemma *subspace-linear-preimage*: *linear f* $\Longrightarrow$ *subspace S* $\Longrightarrow$ *subspace* $\{x.\ f\ x \in S\}$
by (*auto simp add*: *subspace-def linear-iff linear-0*[*of f*])

lemma *subspace-trivial*: *subspace* $\{0\}$
by (*simp add*: *subspace-def*)

lemma (**in** *real-vector*) *subspace-inter*: *subspace A* $\Longrightarrow$ *subspace B* $\Longrightarrow$ *subspace* $(A \cap B)$
by (*simp add*: *subspace-def*)

lemma *subspace-Times*: *subspace A* $\Longrightarrow$ *subspace B* $\Longrightarrow$ *subspace* $(A \times B)$

unfolding *subspace-def zero-prod-def* **by** *simp*

Properties of span.

lemma (**in** *real-vector*) *span-mono*: $A \subseteq B \Longrightarrow$ *span* $A \subseteq$ *span* B
by (*metis span-def hull-mono*)

lemma (**in** *real-vector*) *subspace-span*: *subspace* (*span* S)
unfolding *span-def*
apply (*rule hull-in*)
apply (*simp only*: *subspace-def Inter-iff Int-iff subset-eq*)
apply *auto*
done

lemma (**in** *real-vector*) *span-clauses*:
$a \in S \Longrightarrow a \in$ *span* S
$0 \in$ *span* S
$x \in$ *span* $S \Longrightarrow y \in$ *span* $S \Longrightarrow x + y \in$ *span* S
$x \in$ *span* $S \Longrightarrow c *_R x \in$ *span* S
by (*metis span-def hull-subset subset-eq*) (*metis subspace-span subspace-def*)+

lemma *span-unique*:
$S \subseteq T \Longrightarrow$ *subspace* $T \Longrightarrow (\bigwedge T'.\ S \subseteq T' \Longrightarrow$ *subspace* $T' \Longrightarrow T \subseteq T') \Longrightarrow$ *span* $S = T$
unfolding *span-def* **by** (*rule hull-unique*)

lemma *span-minimal*: $S \subseteq T \Longrightarrow$ *subspace* $T \Longrightarrow$ *span* $S \subseteq T$
unfolding *span-def* **by** (*rule hull-minimal*)

lemma (**in** *real-vector*) *span-induct*:
assumes x: $x \in$ *span* S
and P: *subspace* P
and SP: $\bigwedge x.\ x \in S \Longrightarrow x \in P$
shows $x \in P$
proof −
from SP **have** SP': $S \subseteq P$
by (*simp add*: *subset-eq*)
from x *hull-minimal*[**where** S=*subspace*, *OF* SP' P, *unfolded span-def*[*symmetric*]]
show $x \in P$
by (*metis subset-eq*)
qed

lemma *span-empty*[*simp*]: *span* $\{\} = \{0\}$
apply (*simp add*: *span-def*)
apply (*rule hull-unique*)
apply (*auto simp add*: *subspace-def*)
done

lemma (**in** *real-vector*) *independent-empty*[*intro*]: *independent* $\{\}$
by (*simp add*: *dependent-def*)

lemma *dependent-single*[*simp*]: *dependent* $\{x\} \longleftrightarrow x = 0$
unfolding *dependent-def* **by** *auto*

lemma (**in** *real-vector*) *independent-mono*: *independent* $A \Longrightarrow B \subseteq A \Longrightarrow$ *independent* B
apply (*clarsimp simp add*: *dependent-def span-mono*)
apply (*subgoal-tac span* $(B - \{a\}) \leq$ *span* $(A - \{a\})$)
apply *force*
apply (*rule span-mono*)
apply *auto*
done

lemma (**in** *real-vector*) *span-subspace*: $A \subseteq B \Longrightarrow B \leq$ *span* $A \Longrightarrow$ *subspace* B $\Longrightarrow$ *span* $A = B$
by (*metis order-antisym span-def hull-minimal*)

lemma (**in** *real-vector*) *span-induct'*:
assumes *SP*: $\forall x \in S.\ P\ x$
and *P*: *subspace* $\{x.\ P\ x\}$
shows $\forall x \in$ *span* $S.\ P\ x$
using *span-induct SP P* **by** *blast*

inductive-set (**in** *real-vector*) *span-induct-alt-help* **for** $S :: 'a$ *set*
where
span-induct-alt-help-0: $0 \in$ *span-induct-alt-help* S
| *span-induct-alt-help-S*:
$x \in S \Longrightarrow z \in$ *span-induct-alt-help* $S \Longrightarrow$
$(c *_R x + z) \in$ *span-induct-alt-help* S

lemma *span-induct-alt'*:
assumes *h0*: $h\ 0$
and *hS*: $\bigwedge c\ x\ y.\ x \in S \Longrightarrow h\ y \Longrightarrow h\ (c *_R x + y)$
shows $\forall x \in$ *span* $S.\ h\ x$
proof −
{
fix $x :: 'a$
assume *x*: $x \in$ *span-induct-alt-help* S
have $h\ x$
apply (*rule span-induct-alt-help.induct*[*OF x*])
apply (*rule h0*)
apply (*rule hS*)
apply *assumption*
apply *assumption*
done
}
note *th0* = *this*
{
fix x

```
    assume x: x ∈ span S
    have x ∈ span-induct-alt-help S
    proof (rule span-induct[where x=x and S=S])
      show x ∈ span S by (rule x)
    next
      fix x
      assume xS: x ∈ S
      from span-induct-alt-help-S[OF xS span-induct-alt-help-0, of 1]
      show x ∈ span-induct-alt-help S
        by simp
    next
      have 0 ∈ span-induct-alt-help S by (rule span-induct-alt-help-0)
      moreover
      {
        fix x y
        assume h: x ∈ span-induct-alt-help S y ∈ span-induct-alt-help S
        from h have (x + y) ∈ span-induct-alt-help S
          apply (induct rule: span-induct-alt-help.induct)
          apply simp
          unfolding add.assoc
          apply (rule span-induct-alt-help-S)
          apply assumption
          apply simp
          done
      }
      moreover
      {
        fix c x
        assume xt: x ∈ span-induct-alt-help S
        then have (c *R x) ∈ span-induct-alt-help S
          apply (induct rule: span-induct-alt-help.induct)
          apply (simp add: span-induct-alt-help-0)
          apply (simp add: scaleR-right-distrib)
          apply (rule span-induct-alt-help-S)
          apply assumption
          apply simp
          done }
      ultimately show subspace (span-induct-alt-help S)
        unfolding subspace-def Ball-def by blast
    qed
  }
  with th0 show ?thesis by blast
qed

lemma span-induct-alt:
  assumes h0: h 0
    and hS: ⋀c x y. x ∈ S ⟹ h y ⟹ h (c *R x + y)
    and x: x ∈ span S
  shows h x
```

using *span-induct-alt'*[*of h S*] *h0 hS x* **by** *blast*

Individual closure properties.

lemma *span-span*: *span* (*span A*) = *span A*
unfolding *span-def hull-hull* **..**

lemma (**in** *real-vector*) *span-superset*: $x \in S \Longrightarrow x \in span\ S$
by (*metis span-clauses*(*1*))

lemma (**in** *real-vector*) *span-0*: *0* ∈ *span S*
by (*metis subspace-span subspace-0*)

lemma *span-inc*: $S \subseteq span\ S$
by (*metis subset-eq span-superset*)

lemma (**in** *real-vector*) *dependent-0*:
assumes *0* ∈ *A*
shows *dependent A*
unfolding *dependent-def*
using *assms span-0*
by *auto*

lemma (**in** *real-vector*) *span-add*: $x \in span\ S \Longrightarrow y \in span\ S \Longrightarrow x + y \in span\ S$
by (*metis subspace-add subspace-span*)

lemma (**in** *real-vector*) *span-mul*: $x \in span\ S \Longrightarrow c *_R x \in span\ S$
by (*metis subspace-span subspace-mul*)

lemma *span-neg*: $x \in span\ S \Longrightarrow - x \in span\ S$
by (*metis subspace-neg subspace-span*)

lemma *span-sub*: $x \in span\ S \Longrightarrow y \in span\ S \Longrightarrow x - y \in span\ S$
by (*metis subspace-span subspace-sub*)

lemma (**in** *real-vector*) *span-setsum*: $\forall x{\in}A.\ f\ x \in span\ S \Longrightarrow setsum\ f\ A \in span\ S$
by (*rule subspace-setsum* [*OF subspace-span*])

lemma *span-add-eq*: $x \in span\ S \Longrightarrow x + y \in span\ S \longleftrightarrow y \in span\ S$
by (*metis add-minus-cancel scaleR-minus1-left subspace-def subspace-span*)

Mapping under linear image.

lemma *span-linear-image*:
assumes *lf*: *linear f*
shows *span* (*f* ‘ *S*) = *f* ‘ *span S*
proof (*rule span-unique*)
show *f* ‘ *S* ⊆ *f* ‘ *span S*
by (*intro image-mono span-inc*)

```
  show subspace (f ‘ span S)
    using lf subspace-span by (rule subspace-linear-image)
next
  fix T
  assume f ‘ S ⊆ T and subspace T
  then show f ‘ span S ⊆ T
    unfolding image-subset-iff-subset-vimage
    by (intro span-minimal subspace-linear-vimage lf)
qed

lemma span-union: span (A ∪ B) = (λ(a, b). a + b) ‘ (span A × span B)
proof (rule span-unique)
  show A ∪ B ⊆ (λ(a, b). a + b) ‘ (span A × span B)
    by safe (force intro: span-clauses)+
next
  have linear (λ(a, b). a + b)
    by (simp add: linear-iff scaleR-add-right)
  moreover have subspace (span A × span B)
    by (intro subspace-Times subspace-span)
  ultimately show subspace ((λ(a, b). a + b) ‘ (span A × span B))
    by (rule subspace-linear-image)
next
  fix T
  assume A ∪ B ⊆ T and subspace T
  then show (λ(a, b). a + b) ‘ (span A × span B) ⊆ T
    by (auto intro!: subspace-add elim: span-induct)
qed
```

The key breakdown property.

```
lemma span-singleton: span {x} = range (λk. k *R x)
proof (rule span-unique)
  show {x} ⊆ range (λk. k *R x)
    by (fast intro: scaleR-one [symmetric])
  show subspace (range (λk. k *R x))
    unfolding subspace-def
    by (auto intro: scaleR-add-left [symmetric])
next
  fix T
  assume {x} ⊆ T and subspace T
  then show range (λk. k *R x) ⊆ T
    unfolding subspace-def by auto
qed

lemma span-insert: span (insert a S) = {x. ∃k. (x − k *R a) ∈ span S}
proof −
  have span ({a} ∪ S) = {x. ∃k. (x − k *R a) ∈ span S}
    unfolding span-union span-singleton
    apply safe
    apply (rule-tac x=k in exI, simp)
```

```
    apply (erule rev-image-eqI [OF SigmaI [OF rangeI]])
    apply auto
    done
  then show ?thesis by simp
qed

lemma span-breakdown:
  assumes bS: b ∈ S
    and aS: a ∈ span S
  shows ∃ k. a − k *R b ∈ span (S − {b})
  using assms span-insert [of b S − {b}]
  by (simp add: insert-absorb)

lemma span-breakdown-eq: x ∈ span (insert a S) ⟷ (∃ k. x − k *R a ∈ span S)
  by (simp add: span-insert)
```

Hence some "reversal" results.

```
lemma in-span-insert:
  assumes a: a ∈ span (insert b S)
    and na: a ∉ span S
  shows b ∈ span (insert a S)
proof −
  from a obtain k where k: a − k *R b ∈ span S
    unfolding span-insert by fast
  show ?thesis
  proof (cases k = 0)
    case True
    with k have a ∈ span S by simp
    with na show ?thesis by simp
  next
    case False
    from k have (− inverse k) *R (a − k *R b) ∈ span S
      by (rule span-mul)
    then have b − inverse k *R a ∈ span S
      using ‹k ≠ 0› by (simp add: scaleR-diff-right)
    then show ?thesis
      unfolding span-insert by fast
  qed
qed

lemma in-span-delete:
  assumes a: a ∈ span S
    and na: a ∉ span (S − {b})
  shows b ∈ span (insert a (S − {b}))
  apply (rule in-span-insert)
  apply (rule set-rev-mp)
  apply (rule a)
  apply (rule span-mono)
  apply blast
```

apply (*rule na*)
done

Transitivity property.

lemma *span-redundant*: $x \in span\ S \Longrightarrow span\ (insert\ x\ S) = span\ S$
unfolding *span-def* **by** (*rule hull-redundant*)

lemma *span-trans*:
assumes *x*: $x \in span\ S$
and *y*: $y \in span\ (insert\ x\ S)$
shows $y \in span\ S$
using *assms* **by** (*simp only*: *span-redundant*)

lemma *span-insert-0*[*simp*]: $span\ (insert\ 0\ S) = span\ S$
by (*simp only*: *span-redundant span-0*)

An explicit expansion is sometimes needed.

lemma *span-explicit*:
$span\ P = \{y.\ \exists S\ u.\ finite\ S \wedge S \subseteq P \wedge setsum\ (\lambda v.\ u\ v *_R v)\ S = y\}$
(**is** - = *?E* **is** - = $\{y.$ *?h y*$\}$ **is** - = $\{y.\ \exists S\ u.$ *?Q S u y*$\}$)
proof −
{
fix *x*
assume *?h x*
then obtain *S u* **where** *finite S* **and** $S \subseteq P$ **and** $setsum\ (\lambda v.\ u\ v *_R v)\ S = x$
by *blast*
then have $x \in span\ P$
by (*auto intro*: *span-setsum span-mul span-superset*)
}
moreover
have $\forall x \in span\ P.$ *?h x*
proof (*rule span-induct-alt'*)
show *?h 0*
by (*rule exI*[**where** *x*={}], *simp*)
next
fix *c x y*
assume *x*: $x \in P$
assume *hy*: *?h y*
from *hy* **obtain** *S u* **where** *fS*: *finite S* **and** *SP*: $S \subseteq P$
and *u*: $setsum\ (\lambda v.\ u\ v *_R v)\ S = y$ **by** *blast*
let *?S* = *insert x S*
let *?u* = $\lambda y.$ *if* $y = x$ *then* (*if* $x \in S$ *then u y* + *c else c*) *else u y*
from *fS SP x* **have** *th0*: *finite* (*insert x S*) *insert x S* $\subseteq P$
by *blast*+
have *?Q ?S ?u* ($c *_R x + y$)
proof *cases*
assume *xS*: $x \in S$
have $setsum\ (\lambda v.$ *?u v* $*_R$ *v*$)$ *?S* = $(\sum v \in S - \{x\}.\ u\ v *_R v) + (u\ x + c)$

```
*R x
      using xS by (simp add: setsum.remove [OF fS xS] insert-absorb)
    also have ... = (∑ v∈S. u v *R v) + c *R x
      by (simp add: setsum.remove [OF fS xS] algebra-simps)
    also have ... = c*R x + y
      by (simp add: add.commute u)
    finally have setsum (λv. ?u v *R v) ?S = c*R x + y .
    then show ?thesis using th0 by blast
  next
    assume xS: x ∉ S
    have th00: (∑ v∈S. (if v = x then c else u v) *R v) = y
      unfolding u[symmetric]
      apply (rule setsum.cong)
      using xS
      apply auto
      done
    show ?thesis using fS xS th0
      by (simp add: th00 add.commute cong del: if-weak-cong)
  qed
  then show ?h (c*R x + y)
    by fast
 qed
 ultimately show ?thesis by blast
qed

lemma dependent-explicit:
  dependent P ⟷ (∃ S u. finite S ∧ S ⊆ P ∧ (∃ v∈S. u v ≠ 0 ∧ setsum (λv. u
v *R v) S = 0))
  (is ?lhs = ?rhs)
proof −
  {
    assume dP: dependent P
    then obtain a S u where aP: a ∈ P and fS: finite S
      and SP: S ⊆ P − {a} and ua: setsum (λv. u v *R v) S = a
      unfolding dependent-def span-explicit by blast
    let ?S = insert a S
    let ?u = λy. if y = a then − 1 else u y
    let ?v = a
    from aP SP have aS: a ∉ S
      by blast
    from fS SP aP have th0: finite ?S ?S ⊆ P ?v ∈ ?S ?u ?v ≠ 0
      by auto
    have s0: setsum (λv. ?u v *R v) ?S = 0
      using fS aS
      apply simp
      apply (subst (2) ua[symmetric])
      apply (rule setsum.cong)
      apply auto
      done
```

```
    with th0 have ?rhs by fast
  }
  moreover
  {
    fix S u v
    assume fS: finite S
      and SP: S ⊆ P
      and vS: v ∈ S
      and uv: u v ≠ 0
      and u: setsum (λv. u v *R v) S = 0
    let ?a = v
    let ?S = S − {v}
    let ?u = λi. (− u i) / u v
    have th0: ?a ∈ P finite ?S ?S ⊆ P
      using fS SP vS by auto
    have setsum (λv. ?u v *R v) ?S =
      setsum (λv. (− (inverse (u ?a))) *R (u v *R v)) S − ?u v *R v
      using fS vS uv by (simp add: setsum-diff1 field-simps)
    also have ... = ?a
      unfolding scaleR-right.setsum [symmetric] u using uv by simp
    finally have setsum (λv. ?u v *R v) ?S = ?a .
    with th0 have ?lhs
      unfolding dependent-def span-explicit
      apply −
      apply (rule bexI[where x= ?a])
      apply (simp-all del: scaleR-minus-left)
      apply (rule exI[where x= ?S])
      apply (auto simp del: scaleR-minus-left)
      done
  }
  ultimately show ?thesis by blast
qed


lemma span-finite:
  assumes fS: finite S
  shows span S = {y. ∃ u. setsum (λv. u v *R v) S = y}
  (is - = ?rhs)
proof −
  {
    fix y
    assume y: y ∈ span S
    from y obtain S' u where fS': finite S'
      and SS': S' ⊆ S
      and u: setsum (λv. u v *R v) S' = y
      unfolding span-explicit by blast
    let ?u = λx. if x ∈ S' then u x else 0
    have setsum (λv. ?u v *R v) S = setsum (λv. u v *R v) S'
      using SS' fS by (auto intro!: setsum.mono-neutral-cong-right)
```

```
    then have setsum (λv. ?u v *R v) S = y by (metis u)
    then have y ∈ ?rhs by auto
  }
  moreover
  {
    fix y u
    assume u: setsum (λv. u v *R v) S = y
    then have y ∈ span S using fS unfolding span-explicit by auto
  }
  ultimately show ?thesis by blast
qed
```

This is useful for building a basis step-by-step.

```
lemma independent-insert:
  independent (insert a S) ⟷
    (if a ∈ S then independent S else independent S ∧ a ∉ span S)
  (is ?lhs ⟷ ?rhs)
proof (cases a ∈ S)
  case True
  then show ?thesis
    using insert-absorb[OF True] by simp
next
  case False
  show ?thesis
  proof
    assume i: ?lhs
    then show ?rhs
      using False
      apply simp
      apply (rule conjI)
      apply (rule independent-mono)
      apply assumption
      apply blast
      apply (simp add: dependent-def)
      done
  next
    assume i: ?rhs
    show ?lhs
      using i False
      apply (auto simp add: dependent-def)
      by (metis in-span-insert insert-Diff-if insert-Diff-single insert-absorb)
  qed
qed
```

The degenerate case of the Exchange Lemma.

```
lemma spanning-subset-independent:
  assumes BA: B ⊆ A
    and iA: independent A
    and AsB: A ⊆ span B
```

```
  shows A = B
proof
  show B ⊆ A by (rule BA)

  from span-mono[OF BA] span-mono[OF AsB]
  have sAB: span A = span B unfolding span-span by blast

  {
    fix x
    assume x: x ∈ A
    from iA have th0: x ∉ span (A - {x})
      unfolding dependent-def using x by blast
    from x have xsA: x ∈ span A
      by (blast intro: span-superset)
    have A - {x} ⊆ A by blast
    then have th1: span (A - {x}) ⊆ span A
      by (metis span-mono)
    {
      assume xB: x ∉ B
      from xB BA have B ⊆ A - {x}
        by blast
      then have span B ⊆ span (A - {x})
        by (metis span-mono)
      with th1 th0 sAB have x ∉ span A
        by blast
      with x have False
        by (metis span-superset)
    }
    then have x ∈ B by blast
  }
  then show A ⊆ B by blast
qed
```

The general case of the Exchange Lemma, the key to what follows.

```
lemma exchange-lemma:
  assumes f:finite t
    and i: independent s
    and sp: s ⊆ span t
  shows ∃t'. card t' = card t ∧ finite t' ∧ s ⊆ t' ∧ t' ⊆ s ∪ t ∧ s ⊆ span t'
  using f i sp
proof (induct card (t - s) arbitrary: s t rule: less-induct)
  case less
  note ft = ‹finite t› and s = ‹independent s› and sp = ‹s ⊆ span t›
  let ?P = λt'. card t' = card t ∧ finite t' ∧ s ⊆ t' ∧ t' ⊆ s ∪ t ∧ s ⊆ span t'
  let ?ths = ∃t'. ?P t'
  {
    assume s ⊆ t
    then have ?ths
      by (metis ft Un-commute sp sup-ge1)
```

```
  }
  moreover
  {
    assume st: t ⊆ s
    from spanning-subset-independent[OF st s sp] st ft span-mono[OF st]
    have ?ths
      by (metis Un-absorb sp)
  }
  moreover
  {
    assume st: ¬ s ⊆ t ¬ t ⊆ s
    from st(2) obtain b where b: b ∈ t b ∉ s
      by blast
    from b have t − {b} − s ⊂ t − s
      by blast
    then have cardlt: card (t − {b} − s) < card (t − s)
      using ft by (auto intro: psubset-card-mono)
    from b ft have ct0: card t ≠ 0
      by auto
    have ?ths
    proof cases
      assume stb: s ⊆ span (t − {b})
      from ft have ftb: finite (t − {b})
        by auto
      from less(1)[OF cardlt ftb s stb]
      obtain u where u: card u = card (t − {b}) s ⊆ u u ⊆ s ∪ (t − {b}) s ⊆
span u
        and fu: finite u by blast
      let ?w = insert b u
      have th0: s ⊆ insert b u
        using u by blast
      from u(3) b have u ⊆ s ∪ t
        by blast
      then have th1: insert b u ⊆ s ∪ t
        using u b by blast
      have bu: b ∉ u
        using b u by blast
      from u(1) ft b have card u = (card t − 1)
        by auto
      then have th2: card (insert b u) = card t
        using card-insert-disjoint[OF fu bu] ct0 by auto
      from u(4) have s ⊆ span u .
      also have ... ⊆ span (insert b u)
        by (rule span-mono) blast
      finally have th3: s ⊆ span (insert b u) .
      from th0 th1 th2 th3 fu have th: ?P ?w
        by blast
      from th show ?thesis by blast
    next
```

```
    assume stb: ¬ s ⊆ span (t − {b})
    from stb obtain a where a: a ∈ s a ∉ span (t − {b})
      by blast
    have ab: a ≠ b
      using a b by blast
    have at: a ∉ t
      using a ab span-superset[of a t− {b}] by auto
    have mlt: card ((insert a (t − {b})) − s) < card (t − s)
      using cardlt ft a b by auto
    have ft': finite (insert a (t − {b}))
      using ft by auto
    {
      fix x
      assume xs: x ∈ s
      have t: t ⊆ insert b (insert a (t − {b}))
        using b by auto
      from b(1) have b ∈ span t
        by (simp add: span-superset)
      have bs: b ∈ span (insert a (t − {b}))
        apply (rule in-span-delete)
        using a sp unfolding subset-eq
        apply auto
        done
      from xs sp have x ∈ span t
        by blast
      with span-mono[OF t] have x: x ∈ span (insert b (insert a (t − {b}))) ..
      from span-trans[OF bs x] have x ∈ span (insert a (t − {b})) .
    }
    then have sp': s ⊆ span (insert a (t − {b}))
      by blast
    from less(1)[OF mlt ft' s sp'] obtain u where u:
      card u = card (insert a (t − {b}))
      finite u s ⊆ u u ⊆ s ∪ insert a (t − {b})
      s ⊆ span u by blast
    from u a b ft at ct0 have ?P u
      by auto
    then show ?thesis by blast
  qed
 }
 ultimately show ?ths by blast
qed
```

This implies corresponding size bounds.

```
lemma independent-span-bound:
 assumes f: finite t
   and i: independent s
   and sp: s ⊆ span t
 shows finite s ∧ card s ≤ card t
 by (metis exchange-lemma[OF f i sp] finite-subset card-mono)
```

lemma *finite-Atleast-Atmost-nat*[*simp*]: *finite* {*f x* |*x*. *x*∈ (*UNIV*::*'a*::*finite set*)}
proof −
  have *eq*: {*f x* |*x*. *x*∈ *UNIV*} = *f* ‘ *UNIV*
    by *auto*
  show *?thesis* **unfolding** *eq*
    apply (*rule finite-imageI*)
    apply (*rule finite*)
    done
qed

13.9 Euclidean Spaces as Typeclass

lemma *independent-Basis*: *independent Basis*
  unfolding *dependent-def*
  apply (*subst span-finite*)
  apply *simp*
  apply *clarify*
  apply (*drule-tac f*=*inner a* **in** *arg-cong*)
  apply (*simp add*: *inner-Basis inner-setsum-right eq-commute*)
  done

lemma *span-Basis* [*simp*]: *span Basis* = *UNIV*
  unfolding *span-finite* [*OF finite-Basis*]
  by (*fast intro*: *euclidean-representation*)

lemma *in-span-Basis*: *x* ∈ *span Basis*
  unfolding *span-Basis* **..**

lemma *Basis-le-norm*: *b* ∈ *Basis* ⟹ |*x* · *b*| ≤ *norm x*
  by (*rule order-trans* [*OF Cauchy-Schwarz-ineq2*]) *simp*

lemma *norm-bound-Basis-le*: *b* ∈ *Basis* ⟹ *norm x* ≤ *e* ⟹ |*x* · *b*| ≤ *e*
  by (*metis Basis-le-norm order-trans*)

lemma *norm-bound-Basis-lt*: *b* ∈ *Basis* ⟹ *norm x* < *e* ⟹ |*x* · *b*| < *e*
  by (*metis Basis-le-norm le-less-trans*)

lemma *norm-le-l1*: *norm x* ≤ (∑ *b*∈*Basis*. |*x* · *b*|)
  apply (*subst euclidean-representation*[*of x*, *symmetric*])
  apply (*rule order-trans*[*OF norm-setsum*])
  apply (*auto intro*!: *setsum-mono*)
  done

lemma *setsum-norm-allsubsets-bound*:
  fixes *f* :: *'a* ⇒ *'n*::*euclidean-space*
  assumes *fP*: *finite P*
    and *fPs*: ⋀*Q*. *Q* ⊆ *P* ⟹ *norm* (*setsum f Q*) ≤ *e*
  shows (∑ *x*∈*P*. *norm* (*f x*)) ≤ *2* ∗ *real DIM*(*'n*) ∗ *e*

proof −
have $(\sum x{\in}P.\ norm\ (f\ x)) \leq (\sum x{\in}P.\ \sum b{\in}Basis.\ |f\ x \cdot b|)$
by (*rule setsum-mono*) (*rule norm-le-l1*)
also have $(\sum x{\in}P.\ \sum b{\in}Basis.\ |f\ x \cdot b|) = (\sum b{\in}Basis.\ \sum x{\in}P.\ |f\ x \cdot b|)$
by (*rule setsum.commute*)
also have $\ldots \leq$ *of-nat* (*card* (*Basis* :: *'n set*)) $*$ (*2* $*$ *e*)
proof (*rule setsum-bounded-above*)
fix *i* :: *'n*
assume *i*: $i \in Basis$
have *norm* $(\sum x{\in}P.\ |f\ x \cdot i|) \leq$
norm $((\sum x{\in}P \cap -\ \{x.\ f\ x \cdot i < 0\}.\ f\ x) \cdot i) + norm\ ((\sum x{\in}P \cap \{x.\ f\ x \cdot i < 0\}.\ f\ x) \cdot i)$
by (*simp add*: *abs-real-def setsum.If-cases*[*OF fP*] *setsum-negf norm-triangle-ineq4 inner-setsum-left*
del: *real-norm-def*)
also have $\ldots \leq e + e$
unfolding *real-norm-def*
by (*intro add-mono norm-bound-Basis-le i fPs*) *auto*
finally show $(\sum x{\in}P.\ |f\ x \cdot i|) \leq 2{*}e$ **by** *simp*
qed
also have $\ldots$ = *2* $*$ *real DIM*(*'n*) $*$ *e* **by** *simp*
finally show *?thesis* **.**
qed

13.10 Linearity and Bilinearity continued

lemma *linear-bounded*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*real-normed-vector*
assumes *lf*: *linear f*
shows $\exists B.\ \forall x.\ norm\ (f\ x) \leq B * norm\ x$
proof
let *?B* = $\sum b{\in}Basis.\ norm\ (f\ b)$
show $\forall x.\ norm\ (f\ x) \leq$ *?B* $*$ *norm x*
proof
fix *x* :: *'a*
let *?g* = $\lambda b.\ (x \cdot b) *_R f\ b$
have *norm* $(f\ x) = norm\ (f\ (\sum b{\in}Basis.\ (x \cdot b) *_R b))$
unfolding *euclidean-representation* **..**
also have $\ldots$ = *norm* (*setsum ?g Basis*)
by (*simp add*: *linear-setsum* [*OF lf*] *linear-cmul* [*OF lf*])
finally have *th0*: *norm* $(f\ x) = norm\ (setsum\ ?g\ Basis)$ **.**
have *th*: $\forall\ b{\in}Basis.\ norm\ (?g\ b) \leq norm\ (f\ b) * norm\ x$
proof
fix *i* :: *'a*
assume *i*: $i \in Basis$
from *Basis-le-norm*[*OF i*, *of x*]
show *norm* $(?g\ i) \leq norm\ (f\ i) * norm\ x$
unfolding *norm-scaleR*
apply (*subst mult.commute*)

```
      apply (rule mult-mono)
      apply (auto simp add: field-simps)
      done
  qed
  from setsum-norm-le[of - ?g, OF th]
  show norm (f x) ≤ ?B * norm x
    unfolding th0 setsum-left-distrib by metis
 qed
qed

lemma linear-conv-bounded-linear:
 fixes f :: 'a::euclidean-space ⇒ 'b::real-normed-vector
 shows linear f ⟷ bounded-linear f
proof
 assume linear f
 then interpret f: linear f .
 show bounded-linear f
 proof
  have ∃ B. ∀ x. norm (f x) ≤ B * norm x
    using ‹linear f› by (rule linear-bounded)
  then show ∃ K. ∀ x. norm (f x) ≤ norm x * K
    by (simp add: mult.commute)
 qed
next
 assume bounded-linear f
 then interpret f: bounded-linear f .
 show linear f ..
qed

lemmas linear-linear = linear-conv-bounded-linear[symmetric]

lemma linear-bounded-pos:
 fixes f :: 'a::euclidean-space ⇒ 'b::real-normed-vector
 assumes lf: linear f
 shows ∃ B > 0. ∀ x. norm (f x) ≤ B * norm x
proof −
 have ∃ B > 0. ∀ x. norm (f x) ≤ norm x * B
  using lf unfolding linear-conv-bounded-linear
  by (rule bounded-linear.pos-bounded)
 then show ?thesis
  by (simp only: mult.commute)
qed

lemma bounded-linearI':
 fixes f ::'a::euclidean-space ⇒ 'b::real-normed-vector
 assumes ⋀x y. f (x + y) = f x + f y
  and ⋀c x. f (c *R x) = c *R f x
 shows bounded-linear f
 unfolding linear-conv-bounded-linear[symmetric]
```

by (*rule linearI*[*OF assms*])

lemma *bilinear-bounded*:
fixes *h* :: *'m*::*euclidean-space* ⇒ *'n*::*euclidean-space* ⇒ *'k*::*real-normed-vector*
assumes *bh*: *bilinear h*
shows ∃ *B*. ∀ *x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y*
proof (*clarify intro*!: *exI*[*of* - ∑ *i*∈*Basis*. ∑ *j*∈*Basis*. *norm* (*h i j*)])
fix *x* :: *'m*
fix *y* :: *'n*
have *norm* (*h x y*) = *norm* (*h* (*setsum* (λ*i*. (*x* · *i*) ∗$_R$ *i*) *Basis*) (*setsum* (λ*i*. (*y* · *i*) ∗$_R$ *i*) *Basis*))
apply (*subst euclidean-representation*[**where** *'a*=*'m*])
apply (*subst euclidean-representation*[**where** *'a*=*'n*])
apply *rule*
done
also have ... = *norm* (*setsum* (λ (*i*,*j*). *h* ((*x* · *i*) ∗$_R$ *i*) ((*y* · *j*) ∗$_R$ *j*)) (*Basis* × *Basis*))
unfolding *bilinear-setsum*[*OF bh finite-Basis finite-Basis*] **..**
finally have *th*: *norm* (*h x y*) = ... **.**
show *norm* (*h x y*) ≤ (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. *norm* (*h i j*)) ∗ *norm x* ∗ *norm y*
apply (*auto simp add*: *setsum-left-distrib th setsum.cartesian-product*)
apply (*rule setsum-norm-le*)
apply *simp*
apply (*auto simp add*: *bilinear-rmul*[*OF bh*] *bilinear-lmul*[*OF bh*]
field-simps simp del: *scaleR-scaleR*)
apply (*rule mult-mono*)
apply (*auto simp add*: *zero-le-mult-iff Basis-le-norm*)
apply (*rule mult-mono*)
apply (*auto simp add*: *zero-le-mult-iff Basis-le-norm*)
done
qed

lemma *bilinear-conv-bounded-bilinear*:
fixes *h* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space* ⇒ *'c*::*real-normed-vector*
shows *bilinear h* ⟷ *bounded-bilinear h*
proof
assume *bilinear h*
show *bounded-bilinear h*
proof
fix *x y z*
show *h* (*x* + *y*) *z* = *h x z* + *h y z*
using ‹*bilinear h*› **unfolding** *bilinear-def linear-iff* **by** *simp*
next
fix *x y z*
show *h x* (*y* + *z*) = *h x y* + *h x z*
using ‹*bilinear h*› **unfolding** *bilinear-def linear-iff* **by** *simp*
next
fix *r x y*

```
    show h (scaleR r x) y = scaleR r (h x y)
      using ‹bilinear h› unfolding bilinear-def linear-iff
      by simp
  next
    fix r x y
    show h x (scaleR r y) = scaleR r (h x y)
      using ‹bilinear h› unfolding bilinear-def linear-iff
      by simp
  next
    have ∃ B. ∀ x y. norm (h x y) ≤ B * norm x * norm y
      using ‹bilinear h› by (rule bilinear-bounded)
    then show ∃ K. ∀ x y. norm (h x y) ≤ norm x * norm y * K
      by (simp add: ac-simps)
  qed
next
  assume bounded-bilinear h
  then interpret h: bounded-bilinear h .
  show bilinear h
    unfolding bilinear-def linear-conv-bounded-linear
    using h.bounded-linear-left h.bounded-linear-right by simp
qed

lemma bilinear-bounded-pos:
  fixes h :: 'a::euclidean-space ⇒ 'b::euclidean-space ⇒ 'c::real-normed-vector
  assumes bh: bilinear h
  shows ∃ B > 0. ∀ x y. norm (h x y) ≤ B * norm x * norm y
proof −
  have ∃ B > 0. ∀ x y. norm (h x y) ≤ norm x * norm y * B
    using bh [unfolded bilinear-conv-bounded-bilinear]
    by (rule bounded-bilinear.pos-bounded)
  then show ?thesis
    by (simp only: ac-simps)
qed
```

13.11 We continue.

```
lemma independent-bound:
  fixes S :: 'a::euclidean-space set
  shows independent S ⟹ finite S ∧ card S ≤ DIM('a)
  using independent-span-bound[OF finite-Basis, of S] by auto

corollary
  fixes S :: 'a::euclidean-space set
  assumes independent S
 shows independent-imp-finite: finite S and independent-card-le:card S ≤ DIM('a)
using assms independent-bound by auto

lemma dependent-biggerset:
  fixes S :: 'a::euclidean-space set
```

shows (*finite S* $\Longrightarrow$ *card S* $>$ *DIM*('*a*)) $\Longrightarrow$ *dependent S*
by (*metis independent-bound not-less*)

Hence we can create a maximal independent subset.

lemma *maximal-independent-subset-extend*:
fixes *S* :: '*a*::*euclidean-space set*
assumes *sv*: *S* $\subseteq$ *V*
and *iS*: *independent S*
shows $\exists$ *B*. *S* $\subseteq$ *B* $\wedge$ *B* $\subseteq$ *V* $\wedge$ *independent B* $\wedge$ *V* $\subseteq$ *span B*
using *sv iS*
proof (*induct DIM*('*a*) $-$ *card S arbitrary*: *S rule*: *less-induct*)
case *less*
note *sv* = ‹*S* $\subseteq$ *V*› **and** *i* = ‹*independent S*›
let *?P* = λ*B*. *S* $\subseteq$ *B* $\wedge$ *B* $\subseteq$ *V* $\wedge$ *independent B* $\wedge$ *V* $\subseteq$ *span B*
let *?ths* = $\exists$ *x*. *?P x*
let *?d* = *DIM*('*a*)
show *?ths*
proof (*cases V* $\subseteq$ *span S*)
case *True*
then show *?thesis*
using *sv i* **by** *blast*
next
case *False*
then obtain *a* **where** *a*: *a* $\in$ *V a* $\notin$ *span S*
by *blast*
from *a* **have** *aS*: *a* $\notin$ *S*
by (*auto simp add*: *span-superset*)
have *th0*: *insert a S* $\subseteq$ *V*
using *a sv* **by** *blast*
from *independent-insert*[*of a S*] *i a*
have *th1*: *independent* (*insert a S*)
by *auto*
have *mlt*: *?d* $-$ *card* (*insert a S*) $<$ *?d* $-$ *card S*
using *aS a independent-bound*[*OF th1*] **by** *auto*

from *less*(*1*)[*OF mlt th0 th1*]
obtain *B* **where** *B*: *insert a S* $\subseteq$ *B B* $\subseteq$ *V independent B V* $\subseteq$ *span B*
by *blast*
from *B* **have** *?P B* **by** *auto*
then show *?thesis* **by** *blast*
qed
qed

lemma *maximal-independent-subset*:
$\exists$(*B*:: ('*a*::*euclidean-space*) *set*). *B*$\subseteq$ *V* $\wedge$ *independent B* $\wedge$ *V* $\subseteq$ *span B*
by (*metis maximal-independent-subset-extend*[*of* {}:: ('*a*::*euclidean-space*) *set*]
empty-subsetI independent-empty)

Notion of dimension.

definition *dim V* = (*SOME n*. ∃ *B*. *B* ⊆ *V* ∧ *independent B* ∧ *V* ⊆ *span B* ∧ *card B* = *n*)

lemma *basis-exists*:
∃ *B*. (*B* :: ('*a*::*euclidean-space*) *set*) ⊆ *V* ∧ *independent B* ∧ *V* ⊆ *span B* ∧ (*card B* = *dim V*)
unfolding *dim-def some-eq-ex*[*of* λ*n*. ∃ *B*. *B* ⊆ *V* ∧ *independent B* ∧ *V* ⊆ *span B* ∧ (*card B* = *n*)]
using *maximal-independent-subset*[*of V*] *independent-bound*
by *auto*

corollary *dim-le-card*:
fixes *s* :: '*a*::*euclidean-space set*
shows *finite s* ⟹ *dim s* ≤ *card s*
by (*metis basis-exists card-mono*)

Consequences of independence or spanning for cardinality.

lemma *independent-card-le-dim*:
fixes *B* :: '*a*::*euclidean-space set*
assumes *B* ⊆ *V*
and *independent B*
shows *card B* ≤ *dim V*
proof −
from *basis-exists*[*of V*] ‹*B* ⊆ *V*›
obtain *B'* **where** *independent B'*
and *B* ⊆ *span B'*
and *card B'* = *dim V*
by *blast*
with *independent-span-bound*[*OF* - ‹*independent B*› ‹*B* ⊆ *span B'*›] *independent-bound*[*of B'*]
show *?thesis* **by** *auto*
qed

lemma *span-card-ge-dim*:
fixes *B* :: '*a*::*euclidean-space set*
shows *B* ⊆ *V* ⟹ *V* ⊆ *span B* ⟹ *finite B* ⟹ *dim V* ≤ *card B*
by (*metis basis-exists*[*of V*] *independent-span-bound subset-trans*)

lemma *basis-card-eq-dim*:
fixes *V* :: '*a*::*euclidean-space set*
shows *B* ⊆ *V* ⟹ *V* ⊆ *span B* ⟹ *independent B* ⟹ *finite B* ∧ *card B* = *dim V*
by (*metis order-eq-iff independent-card-le-dim span-card-ge-dim independent-bound*)

lemma *dim-unique*:
fixes *B* :: '*a*::*euclidean-space set*
shows *B* ⊆ *V* ⟹ *V* ⊆ *span B* ⟹ *independent B* ⟹ *card B* = *n* ⟹ *dim V* = *n*
by (*metis basis-card-eq-dim*)

More lemmas about dimension.

lemma *dim-UNIV*: *dim* (*UNIV* :: *'a*::*euclidean-space set*) = *DIM*(*'a*)
using *independent-Basis*
by (*intro dim-unique*[*of Basis*]) *auto*

lemma *dim-subset*:
fixes *S* :: *'a*::*euclidean-space set*
shows *S* ⊆ *T* ⟹ *dim S* ≤ *dim T*
using *basis-exists*[*of T*] *basis-exists*[*of S*]
by (*metis independent-card-le-dim subset-trans*)

lemma *dim-subset-UNIV*:
fixes *S* :: *'a*::*euclidean-space set*
shows *dim S* ≤ *DIM*(*'a*)
by (*metis dim-subset subset-UNIV dim-UNIV*)

Converses to those.

lemma *card-ge-dim-independent*:
fixes *B* :: *'a*::*euclidean-space set*
assumes *BV*: *B* ⊆ *V*
and *iB*: *independent B*
and *dVB*: *dim V* ≤ *card B*
shows *V* ⊆ *span B*
proof
fix *a*
assume *aV*: *a* ∈ *V*
{
assume *aB*: *a* ∉ *span B*
then have *iaB*: *independent* (*insert a B*)
using *iB aV BV* **by** (*simp add*: *independent-insert*)
from *aV BV* **have** *th0*: *insert a B* ⊆ *V*
by *blast*
from *aB* **have** *a* ∉*B*
by (*auto simp add*: *span-superset*)
with *independent-card-le-dim*[*OF th0 iaB*] *dVB independent-bound*[*OF iB*]
have *False* **by** *auto*
}
then show *a* ∈ *span B* **by** *blast*
qed

lemma *card-le-dim-spanning*:
assumes *BV*: (*B*:: (*'a*::*euclidean-space*) *set*) ⊆ *V*
and *VB*: *V* ⊆ *span B*
and *fB*: *finite B*
and *dVB*: *dim V* ≥ *card B*
shows *independent B*
proof −
{
fix *a*

```
    assume a: a ∈ B a ∈ span (B − {a})
    from a fB have c0: card B ≠ 0
      by auto
    from a fB have cb: card (B − {a}) = card B − 1
      by auto
    from BV a have th0: B − {a} ⊆ V
      by blast
    {
      fix x
      assume x: x ∈ V
      from a have eq: insert a (B − {a}) = B
        by blast
      from x VB have x': x ∈ span B
        by blast
      from span-trans[OF a(2), unfolded eq, OF x']
      have x ∈ span (B − {a}) .
    }
    then have th1: V ⊆ span (B − {a})
      by blast
    have th2: finite (B − {a})
      using fB by auto
    from span-card-ge-dim[OF th0 th1 th2]
    have c: dim V ≤ card (B − {a}) .
    from c c0 dVB cb have False by simp
  }
  then show ?thesis
    unfolding dependent-def by blast
qed

lemma card-eq-dim:
  fixes B :: 'a::euclidean-space set
  shows B ⊆ V ⟹ card B = dim V ⟹ finite B ⟹ independent B ⟷ V ⊆
span B
  by (metis order-eq-iff card-le-dim-spanning card-ge-dim-independent)
```

More general size bound lemmas.

```
lemma independent-bound-general:
  fixes S :: 'a::euclidean-space set
  shows independent S ⟹ finite S ∧ card S ≤ dim S
  by (metis independent-card-le-dim independent-bound subset-refl)

lemma dependent-biggerset-general:
  fixes S :: 'a::euclidean-space set
  shows (finite S ⟹ card S > dim S) ⟹ dependent S
  using independent-bound-general[of S] by (metis linorder-not-le)

lemma dim-span [simp]:
  fixes S :: 'a::euclidean-space set
  shows dim (span S) = dim S
```

proof −
have *th0*: *dim S* ≤ *dim* (*span S*)
by (*auto simp add*: *subset-eq intro*: *dim-subset span-superset*)
from *basis-exists*[*of S*]
obtain *B* **where** *B*: *B* ⊆ *S independent B S* ⊆ *span B card B* = *dim S*
by *blast*
from *B* **have** *fB*: *finite B card B* = *dim S*
using *independent-bound* **by** *blast*+
have *bSS*: *B* ⊆ *span S*
using *B*(*1*) **by** (*metis subset-eq span-inc*)
have *sssB*: *span S* ⊆ *span B*
using *span-mono*[*OF B*(*3*)] **by** (*simp add*: *span-span*)
from *span-card-ge-dim*[*OF bSS sssB fB*(*1*)] *th0* **show** *?thesis*
using *fB*(*2*) **by** *arith*
qed

lemma *subset-le-dim*:
fixes *S* :: *'a*::*euclidean-space set*
shows *S* ⊆ *span T* ⟹ *dim S* ≤ *dim T*
by (*metis dim-span dim-subset*)

lemma *span-eq-dim*:
fixes *S* :: *'a*::*euclidean-space set*
shows *span S* = *span T* ⟹ *dim S* = *dim T*
by (*metis dim-span*)

lemma *spans-image*:
assumes *lf*: *linear f*
and *VB*: *V* ⊆ *span B*
shows *f* ‘ *V* ⊆ *span* (*f* ‘ *B*)
unfolding *span-linear-image*[*OF lf*] **by** (*metis VB image-mono*)

lemma *dim-image-le*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *lf*: *linear f*
shows *dim* (*f* ‘ *S*) ≤ *dim* (*S*)
proof −
from *basis-exists*[*of S*] **obtain** *B* **where**
B: *B* ⊆ *S independent B S* ⊆ *span B card B* = *dim S* **by** *blast*
from *B* **have** *fB*: *finite B card B* = *dim S*
using *independent-bound* **by** *blast*+
have *dim* (*f* ‘ *S*) ≤ *card* (*f* ‘ *B*)
apply (*rule span-card-ge-dim*)
using *lf B fB*
apply (*auto simp add*: *span-linear-image spans-image subset-image-iff*)
done
also have ... ≤ *dim S*
using *card-image-le*[*OF fB*(*1*)] *fB* **by** *simp*
finally show *?thesis* **.**

qed

Relation between bases and injectivity/surjectivity of map.

lemma *spanning-surjective-image*:
assumes *us*: *UNIV* ⊆ *span S*
and *lf*: *linear f*
and *sf*: *surj f*
shows *UNIV* ⊆ *span* (*f* ‘ *S*)
proof −
have *UNIV* ⊆ *f* ‘ *UNIV*
using *sf* **by** (*auto simp add*: *surj-def*)
also have . . . ⊆ *span* (*f* ‘ *S*)
using *spans-image*[*OF lf us*] **.**
finally show *?thesis* **.**
qed

lemma *independent-injective-image*:
assumes *iS*: *independent S*
and *lf*: *linear f*
and *fi*: *inj f*
shows *independent* (*f* ‘ *S*)
proof −
{
fix *a*
assume *a*: *a* ∈ *S f a* ∈ *span* (*f* ‘ *S* − {*f a*})
have *eq*: *f* ‘ *S* − {*f a*} = *f* ‘ (*S* − {*a*})
using *fi* **by** (*auto simp add*: *inj-on-def*)
from *a* **have** *f a* ∈ *f* ‘ *span* (*S* − {*a*})
unfolding *eq span-linear-image*[*OF lf*, *of S* − {*a*}] **by** *blast*
then have *a* ∈ *span* (*S* − {*a*})
using *fi* **by** (*auto simp add*: *inj-on-def*)
with *a*(*1*) *iS* **have** *False*
by (*simp add*: *dependent-def*)
}
then show *?thesis*
unfolding *dependent-def* **by** *blast*
qed

Picking an orthogonal replacement for a spanning set.

lemma *vector-sub-project-orthogonal*:
fixes *b x* :: *'a*::*euclidean-space*
shows *b* • (*x* − ((*b* • *x*) / (*b* • *b*)) ∗*R b*) = *0*
unfolding *inner-simps* **by** *auto*

lemma *pairwise-orthogonal-insert*:
assumes *pairwise orthogonal S*
and ⋀*y*. *y* ∈ *S* ⟹ *orthogonal x y*
shows *pairwise orthogonal* (*insert x S*)
using *assms* **unfolding** *pairwise-def*

by (*auto simp add*: *orthogonal-commute*)

lemma *basis-orthogonal*:
fixes B :: $'a$::*real-inner set*
assumes *fB*: *finite B*
shows $\exists\, C.$ *finite* $C \wedge$ *card* $C \leq$ *card* $B \wedge$ *span* $C =$ *span* $B \wedge$ *pairwise orthogonal* C
(**is** $\exists\, C.$ *?P B C*)
using *fB*
proof (*induct rule*: *finite-induct*)
case *empty*
then show *?case*
apply (*rule exI*[**where** x={}])
apply (*auto simp add*: *pairwise-def*)
done
next
case (*insert a B*)
note *fB* = ‹*finite B*› **and** *aB* = ‹$a \notin B$›
from ‹$\exists\, C.$ *finite* $C \wedge$ *card* $C \leq$ *card* $B \wedge$ *span* $C =$ *span* $B \wedge$ *pairwise orthogonal* C›
obtain C **where** C: *finite C card* $C \leq$ *card B*
span $C =$ *span B pairwise orthogonal C* **by** *blast*
let *?a* = $a -$ *setsum* $(\lambda x.\ (x \cdot a\ /\ (x \cdot x)) *_R x)\ C$
let *?C* = *insert ?a C*
from *C*(*1*) **have** *fC*: *finite ?C*
by *simp*
from *fB aB C*(*1*,*2*) **have** *cC*: *card ?C* $\leq$ *card* (*insert a B*)
by (*simp add*: *card-insert-if*)
{
fix *x k*
have *th0*: $\bigwedge(a::'a)\ b\ c.\ a - (b - c) = c + (a - b)$
by (*simp add*: *field-simps*)
have $x - k *_R (a - (\sum x \in C.\ (x \cdot a\ /\ (x \cdot x)) *_R x)) \in$ *span* $C \longleftrightarrow x - k *_R a \in$ *span* C
apply (*simp only*: *scaleR-right-diff-distrib th0*)
apply (*rule span-add-eq*)
apply (*rule span-mul*)
apply (*rule span-setsum*)
apply *clarify*
apply (*rule span-mul*)
apply (*rule span-superset*)
apply *assumption*
done
}
then have *SC*: *span ?C* = *span* (*insert a B*)
unfolding *set-eq-iff span-breakdown-eq C*(*3*)[*symmetric*] **by** *auto*
{
fix *y*
assume *yC*: $y \in C$

```
    then have Cy: C = insert y (C - {y})
      by blast
    have fth: finite (C - {y})
      using C by simp
    have orthogonal ?a y
      unfolding orthogonal-def
      unfolding inner-diff inner-setsum-left right-minus-eq
      unfolding setsum.remove [OF ‹finite C› ‹y ∈ C›]
      apply (clarsimp simp add: inner-commute[of y a])
      apply (rule setsum.neutral)
      apply clarsimp
      apply (rule C(4)[unfolded pairwise-def orthogonal-def, rule-format])
      using ‹y ∈ C› by auto
  }
  with ‹pairwise orthogonal C› have CPO: pairwise orthogonal ?C
    by (rule pairwise-orthogonal-insert)
  from fC cC SC CPO have ?P (insert a B) ?C
    by blast
  then show ?case by blast
qed

lemma orthogonal-basis-exists:
  fixes V :: ('a::euclidean-space) set
  shows ∃B. independent B ∧ B ⊆ span V ∧ V ⊆ span B ∧ (card B = dim V)
∧ pairwise orthogonal B
proof -
  from basis-exists[of V] obtain B where
    B: B ⊆ V independent B V ⊆ span B card B = dim V
    by blast
  from B have fB: finite B card B = dim V
    using independent-bound by auto
  from basis-orthogonal[OF fB(1)] obtain C where
    C: finite C card C ≤ card B span C = span B pairwise orthogonal C
    by blast
  from C B have CSV: C ⊆ span V
    by (metis span-inc span-mono subset-trans)
  from span-mono[OF B(3)] C have SVC: span V ⊆ span C
    by (simp add: span-span)
  from card-le-dim-spanning[OF CSV SVC C(1)] C(2,3) fB
  have iC: independent C
    by (simp add: dim-span)
  from C fB have card C ≤ dim V
    by simp
  moreover have dim V ≤ card C
    using span-card-ge-dim[OF CSV SVC C(1)]
    by (simp add: dim-span)
  ultimately have CdV: card C = dim V
    using C(1) by simp
  from C B CSV CdV iC show ?thesis
```

by *auto*
qed

lemma *span-eq*: *span S = span T ⟷ S ⊆ span T ∧ T ⊆ span S*
using *span-inc*[*unfolded subset-eq*] **using** *span-mono*[*of T span S*] *span-mono*[*of S span T*]
by (*auto simp add*: *span-span*)

Low-dimensional subset is in a hyperplane (weak orthogonal complement).

lemma *span-not-univ-orthogonal*:
fixes *S* :: *'a*::*euclidean-space set*
assumes *sU*: *span S ≠ UNIV*
shows ∃ *a*::*'a*. *a ≠ 0 ∧* (∀ *x ∈ span S*. *a · x = 0*)
proof −
from *sU* **obtain** *a* **where** *a*: *a ∉ span S*
by *blast*
from *orthogonal-basis-exists* **obtain** *B* **where**
B: *independent B B ⊆ span S S ⊆ span B card B = dim S pairwise orthogonal B*
by *blast*
from *B* **have** *fB*: *finite B card B = dim S*
using *independent-bound* **by** *auto*
from *span-mono*[*OF B*(*2*)] *span-mono*[*OF B*(*3*)]
have *sSB*: *span S = span B*
by (*simp add*: *span-span*)
let *?a = a − setsum* (*λb*. (*a · b / (b · b*)) *∗R b*) *B*
have *setsum* (*λb*. (*a · b / (b · b*)) *∗R b*) *B ∈ span S*
unfolding *sSB*
apply (*rule span-setsum*)
apply *clarsimp*
apply (*rule span-mul*)
apply (*rule span-superset*)
apply *assumption*
done
with *a* **have** *a0*:*?a ≠ 0*
by *auto*
have ∀ *x*∈*span B*. *?a · x = 0*
proof (*rule span-induct'*)
show *subspace* {*x*. *?a · x = 0*}
by (*auto simp add*: *subspace-def inner-add*)
next
{
fix *x*
assume *x*: *x ∈ B*
from *x* **have** *B'*: *B = insert x* (*B − {x}*)
by *blast*
have *fth*: *finite* (*B − {x}*)
using *fB* **by** *simp*
have *?a · x = 0*

```
      apply (subst B')
      using fB fth
      unfolding setsum-clauses(2)[OF fth]
      apply simp unfolding inner-simps
      apply (clarsimp simp add: inner-add inner-setsum-left)
      apply (rule setsum.neutral, rule ballI)
      unfolding inner-commute
      apply (auto simp add: x field-simps
        intro: B(5)[unfolded pairwise-def orthogonal-def, rule-format])
      done
    }
    then show ∀ x ∈ B. ?a · x = 0
      by blast
  qed
  with a0 show ?thesis
    unfolding sSB by (auto intro: exI[where x=?a])
qed

lemma span-not-univ-subset-hyperplane:
  fixes S :: 'a::euclidean-space set
  assumes SU: span S ≠ UNIV
  shows ∃ a. a ≠0 ∧ span S ⊆ {x. a · x = 0}
  using span-not-univ-orthogonal[OF SU] by auto

lemma lowdim-subset-hyperplane:
  fixes S :: 'a::euclidean-space set
  assumes d: dim S < DIM('a)
  shows ∃ a::'a. a ≠ 0 ∧ span S ⊆ {x. a · x = 0}
proof −
  {
    assume span S = UNIV
    then have dim (span S) = dim (UNIV :: ('a) set)
      by simp
    then have dim S = DIM('a)
      by (simp add: dim-span dim-UNIV)
    with d have False by arith
  }
  then have th: span S ≠ UNIV
    by blast
  from span-not-univ-subset-hyperplane[OF th] show ?thesis .
qed
```

We can extend a linear basis-basis injection to the whole set.

```
lemma linear-indep-image-lemma:
  assumes lf: linear f
    and fB: finite B
    and ifB: independent (f ' B)
    and fi: inj-on f B
    and xsB: x ∈ span B
```

```
    and fx: f x = 0
  shows x = 0
  using fB ifB fi xsB fx
proof (induct arbitrary: x rule: finite-induct[OF fB])
  case 1
  then show ?case by auto
next
  case (2 a b x)
  have fb: finite b using 2.prems by simp
  have th0: f ‘ b ⊆ f ‘ (insert a b)
    apply (rule image-mono)
    apply blast
    done
  from independent-mono[ OF 2.prems(2) th0]
  have ifb: independent (f ‘ b)  .
  have fib: inj-on f b
    apply (rule subset-inj-on [OF 2.prems(3)])
    apply blast
    done
  from span-breakdown[of a insert a b, simplified, OF 2.prems(4)]
  obtain k where k: x − k∗R a ∈ span (b − {a})
    by blast
  have f (x − k∗R a) ∈ span (f ‘ b)
    unfolding span-linear-image[OF lf]
    apply (rule imageI)
    using k span-mono[of b − {a} b]
    apply blast
    done
  then have f x − k∗R f a ∈ span (f ‘ b)
    by (simp add: linear-sub[OF lf] linear-cmul[OF lf])
  then have th: −k ∗R f a ∈ span (f ‘ b)
    using 2.prems(5) by simp
  have xsb: x ∈ span b
  proof (cases k = 0)
    case True
    with k have x ∈ span (b − {a}) by simp
    then show ?thesis using span-mono[of b − {a} b]
      by blast
  next
    case False
    with span-mul[OF th, of − 1/ k]
    have th1: f a ∈ span (f ‘ b)
      by auto
    from inj-on-image-set-diff[OF 2.prems(3), of insert a b  {a}, symmetric]
    have tha: f ‘ insert a b − f ‘ {a} = f ‘ (insert a b − {a}) by blast
    from 2.prems(2) [unfolded dependent-def bex-simps(8), rule-format, of f a]
    have f a ∉ span (f ‘ b) using tha
      using 2.hyps(2)
      2.prems(3) by auto
```

```
    with th1 have False by blast
    then show ?thesis by blast
  qed
  from 2.hyps(3)[OF fb ifb fib xsb 2.prems(5)] show x = 0 .
qed
```

We can extend a linear mapping from basis.

```
lemma linear-independent-extend-lemma:
  fixes f :: 'a::real-vector ⇒ 'b::real-vector
  assumes fi: finite B
    and ib: independent B
  shows ∃ g.
    (∀ x∈ span B. ∀ y∈ span B. g (x + y) = g x + g y) ∧
    (∀ x∈ span B. ∀ c. g (c*R x) = c *R g x) ∧
    (∀ x∈ B. g x = f x)
  using ib fi
proof (induct rule: finite-induct[OF fi])
  case 1
  then show ?case by auto
next
  case (2 a b)
  from 2.prems 2.hyps have ibf: independent b finite b
    by (simp-all add: independent-insert)
  from 2.hyps(3)[OF ibf] obtain g where
    g: ∀ x∈span b. ∀ y∈span b. g (x + y) = g x + g y
    ∀ x∈span b. ∀ c. g (c *R x) = c *R g x ∀ x∈b. g x = f x by blast
  let ?h = λz. SOME k. (z − k *R a) ∈ span b
  {
    fix z
    assume z: z ∈ span (insert a b)
    have th0: z − ?h z *R a ∈ span b
      apply (rule someI-ex)
      unfolding span-breakdown-eq[symmetric]
      apply (rule z)
      done
    {
      fix k
      assume k: z − k *R a ∈ span b
      have eq: z − ?h z *R a − (z − k*R a) = (k − ?h z) *R a
        by (simp add: field-simps scaleR-left-distrib [symmetric])
      from span-sub[OF th0 k] have khz: (k − ?h z) *R a ∈ span b
        by (simp add: eq)
      {
        assume k ≠ ?h z
        then have k0: k − ?h z ≠ 0 by simp
        from k0 span-mul[OF khz, of 1 /(k − ?h z)]
        have a ∈ span b by simp
        with 2.prems(1) 2.hyps(2) have False
          by (auto simp add: dependent-def)
```

```
    }
    then have k = ?h z by blast
   }
   with th0 have z − ?h z ∗R a ∈ span b ∧ (∀k. z − k ∗R a ∈ span b ⟶ k =
?h z)
    by blast
 }
 note h = this
 let ?g = λz. ?h z ∗R f a + g (z − ?h z ∗R a)
 {
  fix x y
  assume x: x ∈ span (insert a b)
   and y: y ∈ span (insert a b)
  have tha: ⋀(x::'a) y a k l. (x + y) − (k + l) ∗R a = (x − k ∗R a) + (y − l
∗R a)
   by (simp add: algebra-simps)
  have addh: ?h (x + y) = ?h x + ?h y
   apply (rule conjunct2[OF h, rule-format, symmetric])
   apply (rule span-add[OF x y])
   unfolding tha
   apply (metis span-add x y conjunct1[OF h, rule-format])
   done
  have ?g (x + y) = ?g x + ?g y
   unfolding addh tha
   g(1)[rule-format,OF conjunct1[OF h, OF x] conjunct1[OF h, OF y]]
   by (simp add: scaleR-left-distrib)}
 moreover
 {
  fix x :: 'a
  fix c :: real
  assume x: x ∈ span (insert a b)
  have tha: ⋀(x::'a) c k a. c ∗R x − (c ∗ k) ∗R a = c ∗R (x − k ∗R a)
   by (simp add: algebra-simps)
  have hc: ?h (c ∗R x) = c ∗ ?h x
   apply (rule conjunct2[OF h, rule-format, symmetric])
   apply (metis span-mul x)
   apply (metis tha span-mul x conjunct1[OF h])
   done
  have ?g (c ∗R x) = c∗R ?g x
   unfolding hc tha g(2)[rule-format, OF conjunct1[OF h, OF x]]
   by (simp add: algebra-simps)
 }
 moreover
 {
  fix x
  assume x: x ∈ insert a b
  {
   assume xa: x = a
   have ha1: 1 = ?h a
```

```
      apply (rule conjunct2[OF h, rule-format])
      apply (metis span-superset insertI1)
      using conjunct1[OF h, OF span-superset, OF insertI1]
      apply (auto simp add: span-0)
      done
    from xa ha1[symmetric] have ?g x = f x
      apply simp
      using g(2)[rule-format, OF span-0, of 0]
      apply simp
      done
  }
  moreover
  {
    assume xb: x ∈ b
    have h0: 0 = ?h x
      apply (rule conjunct2[OF h, rule-format])
      apply (metis  span-superset x)
      apply simp
      apply (metis span-superset xb)
      done
    have ?g x = f x
      by (simp add: h0[symmetric] g(3)[rule-format, OF xb])
  }
  ultimately have ?g x = f x
    using x by blast
 }
 ultimately show ?case
  apply −
  apply (rule exI[where x=?g])
  apply blast
  done
qed

lemma linear-independent-extend:
 fixes B :: 'a::euclidean-space set
 assumes iB: independent B
 shows ∃ g. linear g ∧ (∀ x∈B. g x = f x)
proof −
 from maximal-independent-subset-extend[of B UNIV] iB
 obtain C where C: B ⊆ C independent C ⋀x. x ∈ span C
  by auto

 from C(2) independent-bound[of C] linear-independent-extend-lemma[of C f]
 obtain g where g:
   (∀ x∈ span C. ∀ y∈ span C. g (x + y) = g x + g y) ∧
    (∀ x∈ span C. ∀ c. g (c∗R x) = c ∗R g x) ∧
    (∀ x∈ C. g x = f x) by blast
 from g show ?thesis
  unfolding linear-iff
```

```
    using C
    apply clarsimp
    apply blast
    done
qed
```

Can construct an isomorphism between spaces of same dimension.

```
lemma subspace-isomorphism:
  fixes S :: 'a::euclidean-space set
    and T :: 'b::euclidean-space set
  assumes s: subspace S
    and t: subspace T
    and d: dim S = dim T
  shows ∃f. linear f ∧ f ‘ S = T ∧ inj-on f S
proof −
  from basis-exists[of S] independent-bound
  obtain B where B: B ⊆ S independent B S ⊆ span B card B = dim S and fB:
finite B
    by blast
  from basis-exists[of T] independent-bound
  obtain C where C: C ⊆ T independent C T ⊆ span C card C = dim T and
fC: finite C
    by blast
  from B(4) C(4) card-le-inj[of B C] d
  obtain f where f: f ‘ B ⊆ C inj-on f B using ‹finite B› ‹finite C›
    by auto
  from linear-independent-extend[OF B(2)]
  obtain g where g: linear g ∀x∈ B. g x = f x
    by blast
  from inj-on-iff-eq-card[OF fB, of f] f(2) have card (f ‘ B) = card B
    by simp
  with B(4) C(4) have ceq: card (f ‘ B) = card C
    using d by simp
  have g ‘ B = f ‘ B
    using g(2) by (auto simp add: image-iff)
  also have ... = C using card-subset-eq[OF fC f(1) ceq] .
  finally have gBC: g ‘ B = C .
  have gi: inj-on g B
    using f(2) g(2) by (auto simp add: inj-on-def)
  note g0 = linear-indep-image-lemma[OF g(1) fB, unfolded gBC, OF C(2) gi]
  {
    fix x y
    assume x: x ∈ S and y: y ∈ S and gxy: g x = g y
    from B(3) x y have x′: x ∈ span B and y′: y ∈ span B
      by blast+
    from gxy have th0: g (x − y) = 0
      by (simp add: linear-sub[OF g(1)])
    have th1: x − y ∈ span B
      using x′ y′ by (metis span-sub)
```

```
    have x = y
      using g0[OF th1 th0] by simp
  }
  then have giS: inj-on g S
    unfolding inj-on-def by blast
  from span-subspace[OF B(1,3) s] have g ‘ S = span (g ‘ B)
    by (simp add: span-linear-image[OF g(1)])
  also have ... = span C unfolding gBC ..
  also have ... = T using span-subspace[OF C(1,3) t] .
  finally have gS: g ‘ S = T .
  from g(1) gS giS show ?thesis
    by blast
qed
```

Linear functions are equal on a subspace if they are on a spanning set.

```
lemma subspace-kernel:
  assumes lf: linear f
  shows subspace {x. f x = 0}
  apply (simp add: subspace-def)
  apply (simp add: linear-add[OF lf] linear-cmul[OF lf] linear-0[OF lf])
  done

lemma linear-eq-0-span:
  assumes lf: linear f and f0: ∀x∈B. f x = 0
  shows ∀x ∈ span B. f x = 0
  using f0 subspace-kernel[OF lf]
  by (rule span-induct′)

lemma linear-eq-0:
  assumes lf: linear f
    and SB: S ⊆ span B
    and f0: ∀x∈B. f x = 0
  shows ∀x ∈ S. f x = 0
  by (metis linear-eq-0-span[OF lf] subset-eq SB f0)

lemma linear-eq:
  assumes lf: linear f
    and lg: linear g
    and S: S ⊆ span B
    and fg: ∀ x∈ B. f x = g x
  shows ∀x∈ S. f x = g x
proof −
  let ?h = λx. f x − g x
  from fg have fg′: ∀x∈ B. ?h x = 0 by simp
  from linear-eq-0[OF linear-compose-sub[OF lf lg] S fg′]
  show ?thesis by simp
qed

lemma linear-eq-stdbasis:
```

fixes f :: $'a$::*euclidean-space* ⇒ -
assumes *lf*: *linear f*
and *lg*: *linear g*
and *fg*: ∀ *b*∈*Basis*. *f b* = *g b*
shows $f = g$
using *linear-eq*[*OF lf lg*, *of* - *Basis*] *fg* **by** *auto*

Similar results for bilinear functions.

lemma *bilinear-eq*:
assumes *bf*: *bilinear f*
and *bg*: *bilinear g*
and *SB*: $S \subseteq$ *span B*
and *TC*: $T \subseteq$ *span C*
and *fg*: ∀ *x*∈ *B*. ∀ *y*∈ *C*. *f x y* = *g x y*
shows ∀ *x*∈*S*. ∀ *y*∈*T*. *f x y* = *g x y*
proof −
let *?P* = {*x*. ∀ *y*∈ *span C*. *f x y* = *g x y*}
from *bf bg* **have** *sp*: *subspace ?P*
unfolding *bilinear-def linear-iff subspace-def bf bg*
by (*auto simp add*: *span-0 bilinear-lzero*[*OF bf*] *bilinear-lzero*[*OF bg*] *span-add Ball-def*
intro: *bilinear-ladd*[*OF bf*])

have ∀ *x* ∈ *span B*. ∀ *y*∈ *span C*. *f x y* = *g x y*
apply (*rule span-induct'* [*OF* - *sp*])
apply (*rule ballI*)
apply (*rule span-induct'*)
apply (*simp add*: *fg*)
apply (*auto simp add*: *subspace-def*)
using *bf bg* **unfolding** *bilinear-def linear-iff*
apply (*auto simp add*: *span-0 bilinear-rzero*[*OF bf*] *bilinear-rzero*[*OF bg*] *span-add Ball-def*
intro: *bilinear-ladd*[*OF bf*])
done
then show *?thesis*
using *SB TC* **by** *auto*
qed

lemma *bilinear-eq-stdbasis*:
fixes f :: $'a$::*euclidean-space* ⇒ $'b$::*euclidean-space* ⇒ -
assumes *bf*: *bilinear f*
and *bg*: *bilinear g*
and *fg*: ∀ *i*∈*Basis*. ∀ *j*∈*Basis*. *f i j* = *g i j*
shows $f = g$
using *bilinear-eq*[*OF bf bg equalityD2*[*OF span-Basis*] *equalityD2*[*OF span-Basis*] *fg*] **by** *blast*

Detailed theorems about left and right invertibility in general case.

lemma *linear-injective-left-inverse*:

fixes $f :: 'a{::}euclidean\text{-}space \Rightarrow 'b{::}euclidean\text{-}space$
assumes *lf*: *linear f*
and *fi*: *inj f*
shows $\exists g.\ linear\ g \wedge g \circ f = id$
proof −
from *linear-independent-extend*[*OF independent-injective-image*, *OF independent-Basis*, *OF lf fi*]
obtain $h :: 'b \Rightarrow 'a$ **where** *h*: *linear h* $\forall x \in f\ `\ Basis.\ h\ x = inv\ f\ x$
by *blast*
from *h*(2) **have** *th*: $\forall i{\in}Basis.\ (h \circ f)\ i = id\ i$
using *inv-o-cancel*[*OF fi*, *unfolded fun-eq-iff id-def o-def*]
by *auto*
from *linear-eq-stdbasis*[*OF linear-compose*[*OF lf h*(1)] *linear-id th*]
have $h \circ f = id$ **.**
then show *?thesis*
using *h*(1) **by** *blast*
qed

lemma *linear-surjective-right-inverse*:
fixes $f :: 'a{::}euclidean\text{-}space \Rightarrow 'b{::}euclidean\text{-}space$
assumes *lf*: *linear f*
and *sf*: *surj f*
shows $\exists g.\ linear\ g \wedge f \circ g = id$
proof −
from *linear-independent-extend*[*OF independent-Basis*[**where** $'a='b$],*of inv f*]
obtain $h :: 'b \Rightarrow 'a$ **where** *h*: *linear h* $\forall x{\in}Basis.\ h\ x = inv\ f\ x$
by *blast*
from *h*(2) **have** *th*: $\forall i{\in}Basis.\ (f \circ h)\ i = id\ i$
using *sf* **by** (*auto simp add*: *surj-iff-all*)
from *linear-eq-stdbasis*[*OF linear-compose*[*OF h*(1) *lf*] *linear-id th*]
have $f \circ h = id$ **.**
then show *?thesis*
using *h*(1) **by** *blast*
qed

An injective map $'a \Rightarrow 'b$ is also surjective.

lemma *linear-injective-imp-surjective*:
fixes $f :: 'a{::}euclidean\text{-}space \Rightarrow 'a{::}euclidean\text{-}space$
assumes *lf*: *linear f*
and *fi*: *inj f*
shows *surj f*
proof −
let *?U* = *UNIV* :: $'a$ *set*
from *basis-exists*[*of ?U*] **obtain** *B*
where *B*: $B \subseteq ?U$ *independent B* $?U \subseteq span\ B$ *card B* = *dim ?U*
by *blast*
from *B*(4) **have** *d*: *dim ?U* = *card B*
by *simp*
have *th*: $?U \subseteq span\ (f\ `\ B)$

```
    apply (rule card-ge-dim-independent)
    apply blast
    apply (rule independent-injective-image[OF B(2) lf fi])
    apply (rule order-eq-refl)
    apply (rule sym)
    unfolding d
    apply (rule card-image)
    apply (rule subset-inj-on[OF fi])
    apply blast
    done
  from th show ?thesis
    unfolding span-linear-image[OF lf] surj-def
    using B(3) by blast
qed
```

And vice versa.

```
lemma surjective-iff-injective-gen:
  assumes fS: finite S
    and fT: finite T
    and c: card S = card T
    and ST: f ‘ S ⊆ T
  shows (∀ y ∈ T. ∃ x ∈ S. f x = y) ⟷ inj-on f S
  (is ?lhs ⟷ ?rhs)
proof
  assume h: ?lhs
  {
    fix x y
    assume x: x ∈ S
    assume y: y ∈ S
    assume f: f x = f y
    from x fS have S0: card S ≠ 0
      by auto
    have x = y
    proof (rule ccontr)
      assume xy: ¬ ?thesis
      have th: card S ≤ card (f ‘ (S − {y}))
        unfolding c
        apply (rule card-mono)
        apply (rule finite-imageI)
        using fS apply simp
        using h xy x y f unfolding subset-eq image-iff
        apply auto
        apply (case-tac xa = f x)
        apply (rule bexI[where x=x])
        apply auto
        done
      also have ... ≤ card (S − {y})
        apply (rule card-image-le)
        using fS by simp
```

```
    also have ... ≤ card S − 1 using y fS by simp
    finally show False using S0 by arith
   qed
  }
  then show ?rhs
   unfolding inj-on-def by blast
next
  assume h: ?rhs
  have f ‘ S = T
   apply (rule card-subset-eq[OF fT ST])
   unfolding card-image[OF h]
   apply (rule c)
   done
  then show ?lhs by blast
qed

lemma linear-surjective-imp-injective:
  fixes f :: 'a::euclidean-space ⇒ 'a::euclidean-space
  assumes lf: linear f
   and sf: surj f
  shows inj f
proof −
  let ?U = UNIV :: 'a set
  from basis-exists[of ?U] obtain B
   where B: B ⊆ ?U independent B ?U ⊆ span B and d: card B = dim ?U
   by blast
  {
   fix x
   assume x: x ∈ span B
   assume fx: f x = 0
   from B(2) have fB: finite B
    using independent-bound by auto
   have fBi: independent (f ‘ B)
    apply (rule card-le-dim-spanning[of f ‘ B ?U])
    apply blast
    using sf B(3)
    unfolding span-linear-image[OF lf] surj-def subset-eq image-iff
    apply blast
    using fB apply blast
    unfolding d[symmetric]
    apply (rule card-image-le)
    apply (rule fB)
    done
   have th0: dim ?U ≤ card (f ‘ B)
    apply (rule span-card-ge-dim)
    apply blast
    unfolding span-linear-image[OF lf]
    apply (rule subset-trans[where B = f ‘ UNIV])
    using sf unfolding surj-def
```

```
      apply blast
      apply (rule image-mono)
      apply (rule B(3))
      apply (metis finite-imageI fB)
      done
    moreover have card (f ‘ B) ≤ card B
      by (rule card-image-le, rule fB)
    ultimately have th1: card B = card (f ‘ B)
      unfolding d by arith
    have fiB: inj-on f B
    unfolding surjective-iff-injective-gen[OF fB finite-imageI[OF fB] th1 subset-refl,
symmetric]
      by blast
    from linear-indep-image-lemma[OF lf fB fBi fiB x] fx
    have x = 0 by blast
  }
  then show ?thesis
    unfolding linear-injective-0[OF lf]
    using B(3)
    by blast
qed
```

Hence either is enough for isomorphism.

```
lemma left-right-inverse-eq:
  assumes fg: f ∘ g = id
    and gh: g ∘ h = id
  shows f = h
proof −
  have f = f ∘ (g ∘ h)
    unfolding gh by simp
  also have ... = (f ∘ g) ∘ h
    by (simp add: o-assoc)
  finally show f = h
    unfolding fg by simp
qed

lemma isomorphism-expand:
  f ∘ g = id ∧ g ∘ f = id ⟷ (∀ x. f (g x) = x) ∧ (∀ x. g (f x) = x)
  by (simp add: fun-eq-iff o-def id-def)

lemma linear-injective-isomorphism:
  fixes f :: 'a::euclidean-space ⇒ 'a::euclidean-space
  assumes lf: linear f
    and fi: inj f
  shows ∃ f'. linear f' ∧ (∀ x. f' (f x) = x) ∧ (∀ x. f (f' x) = x)
  unfolding isomorphism-expand[symmetric]
 using linear-surjective-right-inverse[OF lf linear-injective-imp-surjective[OF lf fi]]
    linear-injective-left-inverse[OF lf fi]
  by (metis left-right-inverse-eq)
```

lemma *linear-surjective-isomorphism*:
fixes f :: *'a*::*euclidean-space* ⇒ *'a*::*euclidean-space*
assumes *lf*: *linear f*
and *sf*: *surj f*
shows ∃f'. *linear* f' ∧ (∀ x. f' (f x) = x) ∧ (∀ x. f (f' x) = x)
unfolding *isomorphism-expand*[*symmetric*]
using *linear-surjective-right-inverse*[*OF lf sf*]
linear-injective-left-inverse[*OF lf linear-surjective-imp-injective*[*OF lf sf*]]
by (*metis left-right-inverse-eq*)

Left and right inverses are the same for *'a* ⇒ *'a*.

lemma *linear-inverse-left*:
fixes f :: *'a*::*euclidean-space* ⇒ *'a*::*euclidean-space*
assumes *lf*: *linear f*
and *lf'*: *linear* f'
shows $f \circ f' = id \longleftrightarrow f' \circ f = id$
proof −
{
fix f f':: *'a* ⇒ *'a*
assume *lf*: *linear f linear* f'
assume *f*: $f \circ f' = id$
from *f* **have** *sf*: *surj f*
apply (*auto simp add*: *o-def id-def surj-def*)
apply *metis*
done
from *linear-surjective-isomorphism*[*OF lf*(*1*) *sf*] *lf f*
have $f' \circ f = id$
unfolding *fun-eq-iff o-def id-def* **by** *metis*
}
then show *?thesis*
using *lf lf'* **by** *metis*
qed

Moreover, a one-sided inverse is automatically linear.

lemma *left-inverse-linear*:
fixes f :: *'a*::*euclidean-space* ⇒ *'a*::*euclidean-space*
assumes *lf*: *linear f*
and *gf*: $g \circ f = id$
shows *linear g*
proof −
from *gf* **have** *fi*: *inj f*
apply (*auto simp add*: *inj-on-def o-def id-def fun-eq-iff*)
apply *metis*
done
from *linear-injective-isomorphism*[*OF lf fi*]
obtain h :: *'a* ⇒ *'a* **where** *h*: *linear h* ∀ x. h (f x) = x ∀ x. f (h x) = x
by *blast*
have $h = g$

apply (*rule ext*) **using** *gf h*(*2*,*3*)
apply (*simp add*: *o-def id-def fun-eq-iff*)
apply *metis*
done
with *h*(*1*) **show** *?thesis* **by** *blast*
qed

lemma *inj-linear-imp-inv-linear*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'a*::*euclidean-space*
assumes *linear f inj f* **shows** *linear* (*inv f*)
using *assms inj-iff left-inverse-linear* **by** *blast*

13.12 Infinity norm

definition *infnorm* (*x*::*'a*::*euclidean-space*) = *Sup* {|*x* · *b*| |*b*. *b* ∈ *Basis*}

lemma *infnorm-set-image*:
fixes *x* :: *'a*::*euclidean-space*
shows {|*x* · *i*| |*i*. *i* ∈ *Basis*} = (λ*i*. |*x* · *i*|) ' *Basis*
by *blast*

lemma *infnorm-Max*:
fixes *x* :: *'a*::*euclidean-space*
shows *infnorm x* = *Max* ((λ*i*. |*x* · *i*|) ' *Basis*)
by (*simp add*: *infnorm-def infnorm-set-image cSup-eq-Max del*: *Sup-image-eq*)

lemma *infnorm-set-lemma*:
fixes *x* :: *'a*::*euclidean-space*
shows *finite* {|*x* · *i*| |*i*. *i* ∈ *Basis*}
and {|*x* · *i*| |*i*. *i* ∈ *Basis*} ≠ {}
unfolding *infnorm-set-image*
by *auto*

lemma *infnorm-pos-le*:
fixes *x* :: *'a*::*euclidean-space*
shows *0* ≤ *infnorm x*
by (*simp add*: *infnorm-Max Max-ge-iff ex-in-conv*)

lemma *infnorm-triangle*:
fixes *x* :: *'a*::*euclidean-space*
shows *infnorm* (*x* + *y*) ≤ *infnorm x* + *infnorm y*
proof −
have ∗: ⋀*a b c d* :: *real*. |*a*| ≤ *c* ⟹ |*b*| ≤ *d* ⟹ |*a* + *b*| ≤ *c* + *d*
by *simp*
show *?thesis*
by (*auto simp*: *infnorm-Max inner-add-left intro*!: ∗)
qed

lemma *infnorm-eq-0*:

fixes $x :: 'a$::*euclidean-space*
shows *infnorm* $x = 0 \longleftrightarrow x = 0$
proof −
have *infnorm* $x \leq 0 \longleftrightarrow x = 0$
unfolding *infnorm-Max* **by** (*simp add*: *euclidean-all-zero-iff*)
then show *?thesis*
using *infnorm-pos-le*[*of x*] **by** *simp*
qed

lemma *infnorm-0*: *infnorm* $0 = 0$
by (*simp add*: *infnorm-eq-0*)

lemma *infnorm-neg*: *infnorm* $(- x) =$ *infnorm* x
unfolding *infnorm-def*
apply (*rule cong*[*of Sup Sup*])
apply *blast*
apply *auto*
done

lemma *infnorm-sub*: *infnorm* $(x - y) =$ *infnorm* $(y - x)$
proof −
have $y - x = - (x - y)$ **by** *simp*
then show *?thesis*
by (*metis infnorm-neg*)
qed

lemma *real-abs-sub-infnorm*: $|$*infnorm* $x -$ *infnorm* $y| \leq$ *infnorm* $(x - y)$
proof −
have *th*: $\bigwedge$(*nx*::*real*) *n ny*. $nx \leq n + ny \Longrightarrow ny \leq n + nx \Longrightarrow |nx - ny| \leq n$
by *arith*
from *infnorm-triangle*[*of x − y y*] *infnorm-triangle*[*of x − y* $-x$]
have *ths*: *infnorm* $x \leq$ *infnorm* $(x - y) +$ *infnorm* y
infnorm $y \leq$ *infnorm* $(x - y) +$ *infnorm* x
by (*simp-all add*: *field-simps infnorm-neg*)
from *th*[*OF ths*] **show** *?thesis* **.**
qed

lemma *real-abs-infnorm*: $|$*infnorm* $x| =$ *infnorm* x
using *infnorm-pos-le*[*of x*] **by** *arith*

lemma *Basis-le-infnorm*:
fixes $x :: 'a$::*euclidean-space*
shows $b \in$ *Basis* $\Longrightarrow |x \cdot b| \leq$ *infnorm* x
by (*simp add*: *infnorm-Max*)

lemma *infnorm-mul*: *infnorm* $(a *_R x) = |a| *$ *infnorm* x
unfolding *infnorm-Max*
proof (*safe intro*!: *Max-eqI*)
let *?B* $= (\lambda i. |x \cdot i|)$ ' *Basis*

```
  {
    fix b :: 'a
    assume b ∈ Basis
    then show |a *R x • b| ≤ |a| * Max ?B
      by (simp add: abs-mult mult-left-mono)
  next
    from Max-in[of ?B] obtain b where b ∈ Basis Max ?B = |x • b|
      by (auto simp del: Max-in)
    then show |a| * Max ((λi. |x • i|) ` Basis) ∈ (λi. |a *R x • i|) ` Basis
      by (intro image-eqI[where x=b]) (auto simp: abs-mult)
  }
qed simp

lemma infnorm-mul-lemma: infnorm (a *R x) ≤ |a| * infnorm x
  unfolding infnorm-mul ..

lemma infnorm-pos-lt: infnorm x > 0 ⟷ x ≠ 0
  using infnorm-pos-le[of x] infnorm-eq-0[of x] by arith
```

Prove that it differs only up to a bound from Euclidean norm.

```
lemma infnorm-le-norm: infnorm x ≤ norm x
  by (simp add: Basis-le-norm infnorm-Max)

lemma (in euclidean-space) euclidean-inner: inner x y = (∑ b∈Basis. (x • b) *
(y • b))
  by (subst (1 2) euclidean-representation [symmetric])
    (simp add: inner-setsum-right inner-Basis ac-simps)

lemma norm-le-infnorm:
  fixes x :: 'a::euclidean-space
  shows norm x ≤ sqrt DIM('a) * infnorm x
proof -
  let ?d = DIM('a)
  have real ?d ≥ 0
    by simp
  then have d2: (sqrt (real ?d))^2 = real ?d
    by (auto intro: real-sqrt-pow2)
  have th: sqrt (real ?d) * infnorm x ≥ 0
    by (simp add: zero-le-mult-iff infnorm-pos-le)
  have th1: x • x ≤ (sqrt (real ?d) * infnorm x)^2
    unfolding power-mult-distrib d2
    apply (subst euclidean-inner)
    apply (subst power2-abs[symmetric])
    apply (rule order-trans[OF setsum-bounded-above[where K=|infnorm x|^2]])
    apply (auto simp add: power2-eq-square[symmetric])
    apply (subst power2-abs[symmetric])
    apply (rule power-mono)
    apply (auto simp: infnorm-Max)
    done
```

from *real-le-lsqrt*[*OF inner-ge-zero th th1*]
show *?thesis*
unfolding *norm-eq-sqrt-inner id-def* **.**
qed

lemma *tendsto-infnorm* [*tendsto-intros*]:
assumes $(f \longrightarrow a)\ F$
shows $((\lambda x.\ \mathit{infnorm}\ (f\ x)) \longrightarrow \mathit{infnorm}\ a)\ F$
proof (*rule tendsto-compose* [*OF LIM-I assms*])
fix $r :: \mathit{real}$
assume $r > 0$
then show $\exists s{>}0.\ \forall x.\ x \neq a \wedge \mathit{norm}\ (x - a) < s \longrightarrow \mathit{norm}\ (\mathit{infnorm}\ x - \mathit{infnorm}\ a) < r$
by (*metis real-norm-def le-less-trans real-abs-sub-infnorm infnorm-le-norm*)
qed

Equality in Cauchy-Schwarz and triangle inequalities.

lemma *norm-cauchy-schwarz-eq*: $x \cdot y = \mathit{norm}\ x * \mathit{norm}\ y \longleftrightarrow \mathit{norm}\ x *_R y = \mathit{norm}\ y *_R x$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
{
assume *h*: $x = 0$
then have *?thesis* **by** *simp*
}
moreover
{
assume *h*: $y = 0$
then have *?thesis* **by** *simp*
}
moreover
{
assume *x*: $x \neq 0$ **and** *y*: $y \neq 0$
from *inner-eq-zero-iff*[*of norm y* $*_R$ *x* − *norm x* $*_R$ *y*]
have *?rhs* $\longleftrightarrow$
$(\mathit{norm}\ y * (\mathit{norm}\ y * \mathit{norm}\ x * \mathit{norm}\ x - \mathit{norm}\ x * (x \cdot y)) -$
$\mathit{norm}\ x * (\mathit{norm}\ y * (y \cdot x) - \mathit{norm}\ x * \mathit{norm}\ y * \mathit{norm}\ y) = 0)$
using *x y*
unfolding *inner-simps*
unfolding *power2-norm-eq-inner*[*symmetric*] *power2-eq-square right-minus-eq*
apply (*simp add*: *inner-commute*)
apply (*simp add*: *field-simps*)
apply *metis*
done
also have ... $\longleftrightarrow$ $(2 * \mathit{norm}\ x * \mathit{norm}\ y * (\mathit{norm}\ x * \mathit{norm}\ y - x \cdot y) = 0)$
using *x y*
by (*simp add*: *field-simps inner-commute*)
also have ... $\longleftrightarrow$ *?lhs* **using** *x y*
apply *simp*

apply *metis*
done
finally have *?thesis* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *norm-cauchy-schwarz-abs-eq*:
$|x \cdot y| = norm\ x * norm\ y \longleftrightarrow$
$norm\ x *_R y = norm\ y *_R x \vee norm\ x *_R y = - norm\ y *_R x$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
have *th*: $\bigwedge(x::real)\ a.\ a \geq 0 \Longrightarrow |x| = a \longleftrightarrow x = a \vee x = - a$
by *arith*
have *?rhs* $\longleftrightarrow$ $norm\ x *_R y = norm\ y *_R x \vee norm\ (- x) *_R y = norm\ y *_R (- x)$
by *simp*
also have ... $\longleftrightarrow (x \cdot y = norm\ x * norm\ y \vee (- x) \cdot y = norm\ x * norm\ y)$
unfolding *norm-cauchy-schwarz-eq*[*symmetric*]
unfolding *norm-minus-cancel norm-scaleR* **..**
also have ... $\longleftrightarrow$ *?lhs*
unfolding *th*[*OF mult-nonneg-nonneg*, *OF norm-ge-zero*[*of x*] *norm-ge-zero*[*of y*]] *inner-simps*
by *auto*
finally show *?thesis* **..**
qed

lemma *norm-triangle-eq*:
fixes *x y* :: *'a*::*real-inner*
shows $norm\ (x + y) = norm\ x + norm\ y \longleftrightarrow norm\ x *_R y = norm\ y *_R x$
proof −
{
assume *x*: $x = 0 \vee y = 0$
then have *?thesis*
by (*cases* $x = 0$) *simp-all*
}
moreover
{
assume *x*: $x \neq 0$ **and** *y*: $y \neq 0$
then have $norm\ x \neq 0\ norm\ y \neq 0$
by *simp-all*
then have *n*: $norm\ x > 0\ norm\ y > 0$
using *norm-ge-zero*[*of x*] *norm-ge-zero*[*of y*] **by** *arith*+
have *th*: $\bigwedge(a::real)\ b\ c.\ a + b + c \neq 0 \Longrightarrow a = b + c \longleftrightarrow a^2 = (b + c)^2$
by *algebra*
have $norm\ (x + y) = norm\ x + norm\ y \longleftrightarrow (norm\ (x + y))^2 = (norm\ x + norm\ y)^2$
apply (*rule th*)
using *n norm-ge-zero*[*of x + y*]

apply *arith*
done
also have ... $\longleftrightarrow$ *norm x* $*_R$ *y* = *norm y* $*_R$ *x*
unfolding *norm-cauchy-schwarz-eq*[*symmetric*]
unfolding *power2-norm-eq-inner inner-simps*
by (*simp add*: *power2-norm-eq-inner*[*symmetric*] *power2-eq-square inner-commute field-simps*)
finally have *?thesis* **.**
}
ultimately show *?thesis* **by** *blast*
qed

13.13 Collinearity

definition *collinear* :: *'a*::*real-vector set* $\Rightarrow$ *bool*
where *collinear S* $\longleftrightarrow$ ($\exists$ *u*. $\forall$ *x* $\in$ *S*. $\forall$ *y* $\in$ *S*. $\exists$ *c*. *x* − *y* = *c* $*_R$ *u*)

lemma *collinear-empty* [*iff*]: *collinear* {}
by (*simp add*: *collinear-def*)

lemma *collinear-sing* [*iff*]: *collinear* {*x*}
by (*simp add*: *collinear-def*)

lemma *collinear-2* [*iff*]: *collinear* {*x*, *y*}
apply (*simp add*: *collinear-def*)
apply (*rule exI*[**where** *x*=*x* − *y*])
apply *auto*
apply (*rule exI*[**where** *x*=*1*], *simp*)
apply (*rule exI*[**where** *x*=− *1*], *simp*)
done

lemma *collinear-lemma*: *collinear* {*0*, *x*, *y*} $\longleftrightarrow$ *x* = *0* $\vee$ *y* = *0* $\vee$ ($\exists$ *c*. *y* = *c* $*_R$ *x*)
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
{
assume *x* = *0* $\vee$ *y* = *0*
then have *?thesis*
by (*cases x* = *0*) (*simp-all add*: *collinear-2 insert-commute*)
}
moreover
{
assume *x*: *x* $\neq$ *0* **and** *y*: *y* $\neq$ *0*
have *?thesis*
proof
assume *h*: *?lhs*
then obtain *u* **where** *u*: $\forall$ *x*$\in$ {*0*,*x*,*y*}. $\forall$ *y*$\in$ {*0*,*x*,*y*}. $\exists$ *c*. *x* − *y* = *c* $*_R$ *u*
unfolding *collinear-def* **by** *blast*
from *u*[*rule-format*, *of x 0*] *u*[*rule-format*, *of y 0*]

```
    obtain cx and cy where
      cx: x = cx *R u and cy: y = cy *R u
      by auto
    from cx x have cx0: cx ≠ 0 by auto
    from cy y have cy0: cy ≠ 0 by auto
    let ?d = cy / cx
    from cx cy cx0 have y = ?d *R x
      by simp
    then show ?rhs using x y by blast
  next
    assume h: ?rhs
    then obtain c where c: y = c *R x
      using x y by blast
    show ?lhs
      unfolding collinear-def c
      apply (rule exI[where x=x])
      apply auto
      apply (rule exI[where x=- 1], simp)
      apply (rule exI[where x= -c], simp)
      apply (rule exI[where x=1], simp)
      apply (rule exI[where x=1 - c], simp add: scaleR-left-diff-distrib)
      apply (rule exI[where x=c - 1], simp add: scaleR-left-diff-distrib)
      done
  qed
 }
 ultimately show ?thesis by blast
qed

lemma norm-cauchy-schwarz-equal: |x · y| = norm x * norm y ⟷ collinear {0,
x, y}
 unfolding norm-cauchy-schwarz-abs-eq
 apply (cases x=0, simp-all add: collinear-2)
 apply (cases y=0, simp-all add: collinear-2 insert-commute)
 unfolding collinear-lemma
 apply simp
 apply (subgoal-tac norm x ≠ 0)
 apply (subgoal-tac norm y ≠ 0)
 apply (rule iffI)
 apply (cases norm x *R y = norm y *R x)
 apply (rule exI[where x=(1/norm x) * norm y])
 apply (drule sym)
 unfolding scaleR-scaleR[symmetric]
 apply (simp add: field-simps)
 apply (rule exI[where x=(1/norm x) * - norm y])
 apply clarify
 apply (drule sym)
 unfolding scaleR-scaleR[symmetric]
 apply (simp add: field-simps)
 apply (erule exE)
```

apply (*erule ssubst*)
unfolding *scaleR-scaleR*
unfolding *norm-scaleR*
apply (*subgoal-tac norm x * c = |c| * norm x ∨ norm x * c = - |c| * norm x*)
apply (*auto simp add*: *field-simps*)
done

end

14 A decision procedure for universal multivariate real arithmetic with addition, multiplication and ordering using semidefinite programming

theory *Sum-of-Squares*
imports *Complex-Main*
begin

ML-file *positivstellensatz.ML*
ML-file *Sum-of-Squares/sum-of-squares.ML*
ML-file *Sum-of-Squares/positivstellensatz-tools.ML*
ML-file *Sum-of-Squares/sos-wrapper.ML*

end

15 General linear decision procedure for normed spaces

theory *Norm-Arith*
imports ~~/*src/HOL/Library/Sum-of-Squares*
begin

lemma *norm-cmul-rule-thm*:
fixes *x* :: *'a*::*real-normed-vector*
shows *b* ≥ *norm x* ⟹ |*c*| * *b* ≥ *norm* (*scaleR c x*)
unfolding *norm-scaleR*
apply (*erule mult-left-mono*)
apply *simp*
done

lemma *norm-add-rule-thm*:
fixes *x1 x2* :: *'a*::*real-normed-vector*
shows *norm x1* ≤ *b1* ⟹ *norm x2* ≤ *b2* ⟹ *norm* (*x1* + *x2*) ≤ *b1* + *b2*
by (*rule order-trans* [*OF norm-triangle-ineq add-mono*])

lemma *ge-iff-diff-ge-0*:
fixes a :: *'a*::*linordered-ring*
shows $a \geq b \equiv a - b \geq 0$
by (*simp add*: *field-simps*)

lemma *pth-1*:
fixes x :: *'a*::*real-normed-vector*
shows $x \equiv scaleR\ 1\ x$ **by** *simp*

lemma *pth-2*:
fixes x :: *'a*::*real-normed-vector*
shows $x - y \equiv x + -y$
by (*atomize* (*full*)) *simp*

lemma *pth-3*:
fixes x :: *'a*::*real-normed-vector*
shows $- x \equiv scaleR\ (-1)\ x$
by *simp*

lemma *pth-4*:
fixes x :: *'a*::*real-normed-vector*
shows $scaleR\ 0\ x \equiv 0$
and $scaleR\ c\ 0 = (0::'a)$
by *simp-all*

lemma *pth-5*:
fixes x :: *'a*::*real-normed-vector*
shows $scaleR\ c\ (scaleR\ d\ x) \equiv scaleR\ (c * d)\ x$
by *simp*

lemma *pth-6*:
fixes x :: *'a*::*real-normed-vector*
shows $scaleR\ c\ (x + y) \equiv scaleR\ c\ x + scaleR\ c\ y$
by (*simp add*: *scaleR-right-distrib*)

lemma *pth-7*:
fixes x :: *'a*::*real-normed-vector*
shows $0 + x \equiv x$
and $x + 0 \equiv x$
by *simp-all*

lemma *pth-8*:
fixes x :: *'a*::*real-normed-vector*
shows $scaleR\ c\ x + scaleR\ d\ x \equiv scaleR\ (c + d)\ x$
by (*simp add*: *scaleR-left-distrib*)

lemma *pth-9*:
fixes x :: *'a*::*real-normed-vector*
shows $(scaleR\ c\ x + z) + scaleR\ d\ x \equiv scaleR\ (c + d)\ x + z$

and *scaleR c x* + (*scaleR d x* + *z*) ≡ *scaleR* (*c* + *d*) *x* + *z*
and (*scaleR c x* + *w*) + (*scaleR d x* + *z*) ≡ *scaleR* (*c* + *d*) *x* + (*w* + *z*)
by (*simp-all add*: *algebra-simps*)

lemma *pth-a*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR 0 x* + *y* ≡ *y*
by *simp*

lemma *pth-b*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR c x* + *scaleR d y* ≡ *scaleR c x* + *scaleR d y*
and (*scaleR c x* + *z*) + *scaleR d y* ≡ *scaleR c x* + (*z* + *scaleR d y*)
and *scaleR c x* + (*scaleR d y* + *z*) ≡ *scaleR c x* + (*scaleR d y* + *z*)
and (*scaleR c x* + *w*) + (*scaleR d y* + *z*) ≡ *scaleR c x* + (*w* + (*scaleR d y* + *z*))
by (*simp-all add*: *algebra-simps*)

lemma *pth-c*:
fixes *x* :: *'a*::*real-normed-vector*
shows *scaleR c x* + *scaleR d y* ≡ *scaleR d y* + *scaleR c x*
and (*scaleR c x* + *z*) + *scaleR d y* ≡ *scaleR d y* + (*scaleR c x* + *z*)
and *scaleR c x* + (*scaleR d y* + *z*) ≡ *scaleR d y* + (*scaleR c x* + *z*)
and (*scaleR c x* + *w*) + (*scaleR d y* + *z*) ≡ *scaleR d y* + ((*scaleR c x* + *w*) + *z*)
by (*simp-all add*: *algebra-simps*)

lemma *pth-d*:
fixes *x* :: *'a*::*real-normed-vector*
shows *x* + *0* ≡ *x*
by *simp*

lemma *norm-imp-pos-and-ge*:
fixes *x* :: *'a*::*real-normed-vector*
shows *norm x* ≡ *n* ⟹ *norm x* ≥ *0* ∧ *n* ≥ *norm x*
by *atomize auto*

lemma *real-eq-0-iff-le-ge-0*:
fixes *x* :: *real*
shows *x* = *0* ≡ *x* ≥ *0* ∧ − *x* ≥ *0*
by *arith*

lemma *norm-pths*:
fixes *x* :: *'a*::*real-normed-vector*
shows *x* = *y* ⟷ *norm* (*x* − *y*) ≤ *0*
and *x* ≠ *y* ⟷ ¬ (*norm* (*x* − *y*) ≤ *0*)
using *norm-ge-zero*[*of x* − *y*] **by** *auto*

lemmas *arithmetic-simps* =

arith-simps
add-numeral-special
add-neg-numeral-special
mult-1-left
mult-1-right

ML-file *normarith.ML*

method-setup *norm* = ‹
Scan.succeed (*SIMPLE-METHOD'* o *NormArith.norm-arith-tac*)
› *prove simple linear statements about vector norms*

Hence more metric properties.

lemma *dist-triangle-add*:
fixes *x y x' y'* :: *'a*::*real-normed-vector*
shows *dist* (*x* + *y*) (*x'* + *y'*) ≤ *dist x x'* + *dist y y'*
by *norm*

lemma *dist-triangle-add-half*:
fixes *x x' y y'* :: *'a*::*real-normed-vector*
shows *dist x x'* < *e* / *2* ⟹ *dist y y'* < *e* / *2* ⟹ *dist*(*x* + *y*) (*x'* + *y'*) < *e*
by *norm*

end

16 Elementary topology in Euclidean space.

theory *Topology-Euclidean-Space*
imports
~~/*src*/*HOL*/*Library*/*Indicator-Function*
~~/*src*/*HOL*/*Library*/*Countable-Set*
~~/*src*/*HOL*/*Library*/*FuncSet*
Linear-Algebra
Norm-Arith
begin

lemma *image-affinity-interval*:
fixes *c* :: *'a*::*ordered-real-vector*
shows ((λ*x*. *m* *$_R$ *x* + *c*) ‘ {*a*..*b*}) = (*if* {*a*..*b*}={} *then* {}
else if 0 <= *m then* {*m* *$_R$ *a* + *c* .. *m* *$_R$ *b* + *c*}
else {*m* *$_R$ *b* + *c* .. *m* *$_R$ *a* + *c*})
apply (*case-tac m=0*, *force*)
apply (*auto simp*: *scaleR-left-mono*)
apply (*rule-tac x=inverse m* *$_R$ (*x*−*c*) **in** *rev-image-eqI*, *auto simp*: *pos-le-divideR-eq le-diff-eq scaleR-left-mono-neg*)
apply (*metis diff-le-eq inverse-inverse-eq order.not-eq-order-implies-strict pos-le-divideR-eq positive-imp-inverse-positive*)
apply (*rule-tac x=inverse m* *$_R$ (*x*−*c*) **in** *rev-image-eqI*, *auto simp*: *not-le neg-le-divideR-eq diff-le-eq*)

using *le-diff-eq scaleR-le-cancel-left-neg*
apply *fastforce*
done

lemma *dist-0-norm*:
fixes $x :: 'a$::*real-normed-vector*
shows *dist 0 x* = *norm x*
unfolding *dist-norm* **by** *simp*

lemma *dist-double*: *dist x y* < *d / 2* ⟹ *dist x z* < *d / 2* ⟹ *dist y z* < *d*
using *dist-triangle*[*of y z x*] **by** (*simp add*: *dist-commute*)

lemma *lim-subseq*: *subseq r* ⟹ *s* ⟶ *l* ⟹ (*s* ∘ *r*) ⟶ *l*
by (*rule LIMSEQ-subseq-LIMSEQ*)

lemma *countable-PiE*:
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *countable* (*F i*)) ⟹ *countable* (*PiE I F*)
by (*induct I arbitrary*: *F rule*: *finite-induct*) (*auto simp*: *PiE-insert-eq*)

lemma *Lim-within-open*:
fixes *f* :: $'a$::*topological-space* ⇒ $'b$::*topological-space*
shows *a* ∈ *S* ⟹ *open S* ⟹ (*f* ⟶ *l*)(*at a within S*) ⟷ (*f* ⟶ *l*)(*at a*)
by (*fact tendsto-within-open*)

lemma *Lim-within-open-NO-MATCH*:
fixes *f* :: $'a$::*topological-space* ⇒ $'b$::*topological-space*
shows *a* ∈ *S* ⟹ *NO-MATCH UNIV S* ⟹ *open S* ⟹ (*f* ⟶ *l*)(*at a within S*) ⟷ (*f* ⟶ *l*)(*at a*)
using *tendsto-within-open* **by** *blast*

lemma *continuous-on-union*:
closed s ⟹ *closed t* ⟹ *continuous-on s f* ⟹ *continuous-on t f* ⟹ *continuous-on* (*s* ∪ *t*) *f*
by (*fact continuous-on-closed-Un*)

lemma *continuous-on-cases*:
closed s ⟹ *closed t* ⟹ *continuous-on s f* ⟹ *continuous-on t g* ⟹
∀ *x*. (*x*∈*s* ∧ ¬ *P x*) ∨ (*x* ∈ *t* ∧ *P x*) ⟶ *f x* = *g x* ⟹
continuous-on (*s* ∪ *t*) (λ*x*. *if P x then f x else g x*)
by (*rule continuous-on-If*) *auto*

16.1 Topological Basis

context *topological-space*
begin

definition *topological-basis B* ⟷
(∀ *b*∈*B*. *open b*) ∧ (∀ *x*. *open x* ⟶ (∃ *B'*. *B'* ⊆ *B* ∧ ⋃ *B'* = *x*))

lemma *topological-basis*:
topological-basis B $\longleftrightarrow$ $(\forall x.\ open\ x \longleftrightarrow (\exists B'.\ B' \subseteq B \wedge \bigcup B' = x))$
unfolding *topological-basis-def*
apply *safe*
apply *fastforce*
apply *fastforce*
apply (*erule-tac x=x* **in** *allE*)
apply *simp*
apply (*rule-tac x={x}* **in** *exI*)
apply *auto*
done

lemma *topological-basis-iff*:
assumes $\bigwedge B'.\ B' \in B \Longrightarrow open\ B'$
shows *topological-basis B* $\longleftrightarrow$ $(\forall O'.\ open\ O' \longrightarrow (\forall x{\in}O'.\ \exists B'{\in}B.\ x \in B' \wedge B' \subseteq O'))$
(**is** - $\longleftrightarrow$ *?rhs*)
proof *safe*
fix O' **and** $x::'a$
assume *H*: *topological-basis B open O' x* $\in O'$
then have $(\exists B'{\subseteq}B.\ \bigcup B' = O')$ **by** (*simp add*: *topological-basis-def*)
then obtain B' **where** $B' \subseteq B\ O' = \bigcup B'$ **by** *auto*
then show $\exists B'{\in}B.\ x \in B' \wedge B' \subseteq O'$ **using** *H* **by** *auto*
next
assume *H*: *?rhs*
show *topological-basis B*
using *assms* **unfolding** *topological-basis-def*
proof *safe*
fix $O' :: 'a\ set$
assume *open O'*
with *H* **obtain** *f* **where** $\forall x{\in}O'.\ f\ x \in B \wedge x \in f\ x \wedge f\ x \subseteq O'$
by (*force intro*: *bchoice simp*: *Bex-def*)
then show $\exists B'{\subseteq}B.\ \bigcup B' = O'$
by (*auto intro*: *exI*[**where** $x=\{f\ x\ |x.\ x \in O'\}$])
qed
qed

lemma *topological-basisI*:
assumes $\bigwedge B'.\ B' \in B \Longrightarrow open\ B'$
and $\bigwedge O'\ x.\ open\ O' \Longrightarrow x \in O' \Longrightarrow \exists B'{\in}B.\ x \in B' \wedge B' \subseteq O'$
shows *topological-basis B*
using *assms* **by** (*subst topological-basis-iff*) *auto*

lemma *topological-basisE*:
fixes O'
assumes *topological-basis B*
and *open O'*
and $x \in O'$

obtains B' **where** $B' \in B\ x \in B'\ B' \subseteq O'$
proof *atomize-elim*
from *assms* **have** $\bigwedge B'.\ B' \in B \Longrightarrow$ *open* B'
by (*simp add*: *topological-basis-def*)
with *topological-basis-iff assms*
show $\exists B'.\ B' \in B \wedge x \in B' \wedge B' \subseteq O'$
using *assms* **by** (*simp add*: *Bex-def*)
qed

lemma *topological-basis-open*:
assumes *topological-basis B*
and $X \in B$
shows *open X*
using *assms* **by** (*simp add*: *topological-basis-def*)

lemma *topological-basis-imp-subbasis*:
assumes *B*: *topological-basis B*
shows *open* = *generate-topology B*
proof (*intro ext iffI*)
fix $S :: 'a\ set$
assume *open S*
with *B* **obtain** B' **where** $B' \subseteq B\ S = \bigcup B'$
unfolding *topological-basis-def* **by** *blast*
then show *generate-topology B S*
by (*auto intro*: *generate-topology.intros dest*: *topological-basis-open*)
next
fix $S :: 'a\ set$
assume *generate-topology B S*
then show *open S*
by *induct* (*auto dest*: *topological-basis-open*[*OF B*])
qed

lemma *basis-dense*:
fixes $B :: 'a\ set\ set$
and $f :: 'a\ set \Rightarrow 'a$
assumes *topological-basis B*
and *choosefrom-basis*: $\bigwedge B'.\ B' \neq \{\} \Longrightarrow f\ B' \in B'$
shows $\forall X.$ *open* $X \longrightarrow X \neq \{\} \longrightarrow (\exists B' \in B.\ f\ B' \in X)$
proof (*intro allI impI*)
fix $X :: 'a\ set$
assume *open X* **and** $X \neq \{\}$
from *topological-basisE*[*OF* ‹*topological-basis B*› ‹*open X*› *choosefrom-basis*[*OF* ‹$X \neq \{\}$›]]
obtain B' **where** $B' \in B\ f\ X \in B'\ B' \subseteq X$ **.**
then show $\exists B' \in B.\ f\ B' \in X$
by (*auto intro*!: *choosefrom-basis*)
qed

end

lemma *topological-basis-prod*:
assumes *A*: *topological-basis A*
and *B*: *topological-basis B*
shows *topological-basis* ((λ(*a*, *b*). *a* × *b*) ' (*A* × *B*))
unfolding *topological-basis-def*
proof (*safe*, *simp-all del*: *ex-simps add*: *subset-image-iff ex-simps*(*1*)[*symmetric*])
fix *S* :: ('*a* × '*b*) *set*
assume *open S*
then show ∃ *X*⊆*A* × *B*. (⋃(*a*,*b*)∈*X*. *a* × *b*) = *S*
proof (*safe intro*!: *exI*[*of* - {*x*∈*A* × *B*. *fst x* × *snd x* ⊆ *S*}])
fix *x y*
assume (*x*, *y*) ∈ *S*
from *open-prod-elim*[*OF* ‹*open S*› *this*]
obtain *a b* **where** *a*: *open ax* ∈ *a* **and** *b*: *open b y* ∈ *b* **and** *a* × *b* ⊆ *S*
by (*metis mem-Sigma-iff*)
moreover
from *A a* **obtain** *A0* **where** *A0* ∈ *A x* ∈ *A0 A0* ⊆ *a*
by (*rule topological-basisE*)
moreover
from *B b* **obtain** *B0* **where** *B0* ∈ *B y* ∈ *B0 B0* ⊆ *b*
by (*rule topological-basisE*)
ultimately show (*x*, *y*) ∈ (⋃(*a*, *b*)∈{*X* ∈ *A* × *B*. *fst X* × *snd X* ⊆ *S*}. *a* × *b*)
by (*intro UN-I*[*of* (*A0*, *B0*)]) *auto*
qed *auto*
qed (*metis A B topological-basis-open open-Times*)

16.2 Countable Basis

locale *countable-basis* =
fixes *B* :: '*a*::*topological-space set set*
assumes *is-basis*: *topological-basis B*
and *countable-basis*: *countable B*
begin

lemma *open-countable-basis-ex*:
assumes *open X*
shows ∃ *B'* ⊆ *B*. *X* = ⋃ *B'*
using *assms countable-basis is-basis*
unfolding *topological-basis-def* **by** *blast*

lemma *open-countable-basisE*:
assumes *open X*
obtains *B'* **where** *B'* ⊆ *B X* = ⋃ *B'*
using *assms open-countable-basis-ex*
by (*atomize-elim*) *simp*

lemma *countable-dense-exists*:

$\exists D::'a\ set.\ countable\ D \wedge (\forall X.\ open\ X \longrightarrow X \neq \{\} \longrightarrow (\exists d \in D.\ d \in X))$
proof −
let *?f* = $(\lambda B'.\ SOME\ x.\ x \in B')$
have *countable* (*?f* ‘ *B*) **using** *countable-basis* **by** *simp*
with *basis-dense*[*OF is-basis*, *of ?f*] **show** *?thesis*
by (*intro exI*[**where** *x*=*?f* ‘ *B*]) (*metis* (*mono-tags*) *all-not-in-conv imageI someI*)
qed

lemma *countable-dense-setE*:
obtains *D* :: *'a set*
where *countable D* $\bigwedge X.$ *open X* $\Longrightarrow X \neq \{\} \Longrightarrow \exists d \in D.\ d \in X$
using *countable-dense-exists* **by** *blast*

end

lemma (**in** *first-countable-topology*) *first-countable-basisE*:
obtains *A* **where** *countable A* $\bigwedge a.\ a \in A \Longrightarrow x \in a$ $\bigwedge a.\ a \in A \Longrightarrow$ *open a*
$\bigwedge S.$ *open S* $\Longrightarrow x \in S \Longrightarrow (\exists a{\in}A.\ a \subseteq S)$
using *first-countable-basis*[*of x*]
apply *atomize-elim*
apply (*elim exE*)
apply (*rule-tac x*=*range A* **in** *exI*)
apply *auto*
done

lemma (**in** *first-countable-topology*) *first-countable-basis-Int-stableE*:
obtains *A* **where** *countable A* $\bigwedge a.\ a \in A \Longrightarrow x \in a$ $\bigwedge a.\ a \in A \Longrightarrow$ *open a*
$\bigwedge S.$ *open S* $\Longrightarrow x \in S \Longrightarrow (\exists a{\in}A.\ a \subseteq S)$
$\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \cap b \in A$
proof *atomize-elim*
obtain *A'* **where** *A'*:
countable A'
$\bigwedge a.\ a \in A' \Longrightarrow x \in a$
$\bigwedge a.\ a \in A' \Longrightarrow$ *open a*
$\bigwedge S.$ *open S* $\Longrightarrow x \in S \Longrightarrow \exists a{\in}A'.\ a \subseteq S$
by (*rule first-countable-basisE*) *blast*
def $A \equiv (\lambda N.\ \bigcap((\lambda n.$ *from-nat-into A' n*) ‘ *N*)) ‘ (*Collect finite*::*nat set set*)
then show $\exists A.$ *countable* $A \wedge (\forall a.\ a \in A \longrightarrow x \in a) \wedge (\forall a.\ a \in A \longrightarrow$ *open a*) $\wedge$
$(\forall S.$ *open S* $\longrightarrow x \in S \longrightarrow (\exists a{\in}A.\ a \subseteq S)) \wedge (\forall a\ b.\ a \in A \longrightarrow b \in A \longrightarrow a \cap b \in A)$
proof (*safe intro*!: *exI*[**where** *x*=*A*])
show *countable A*
unfolding *A-def* **by** (*intro countable-image countable-Collect-finite*)
fix *a*
assume $a \in A$
then show $x \in a$ *open a*
using *A'*(*4*)[*OF open-UNIV*] **by** (*auto simp*: *A-def intro*: *A' from-nat-into*)

next
let *?int* = $\lambda N.\ \bigcap$ (*from-nat-into* A' ‘ N)
fix a b
assume $a \in A$ $b \in A$
then obtain N M **where** a = *?int* N b = *?int* M *finite* $(N \cup M)$
by (*auto simp*: *A-def*)
then show $a \cap b \in A$
by (*auto simp*: *A-def intro*!: *image-eqI*[**where** x=$N \cup M$])
next
fix S
assume *open* S $x \in S$
then obtain a **where** a: a∈A' $a \subseteq S$ **using** A' **by** *blast*
then show $\exists\, a$∈$A.\ a \subseteq S$ **using** a A'
by (*intro bexI*[**where** x=a]) (*auto simp*: *A-def intro*: *image-eqI*[**where** x={*to-nat-on* A' a}])
qed
qed

lemma (**in** *topological-space*) *first-countableI*:
assumes *countable* A
and *1*: $\bigwedge a.\ a \in A \Longrightarrow x \in a$ $\bigwedge a.\ a \in A \Longrightarrow$ *open* a
and *2*: $\bigwedge S.$ *open* $S \Longrightarrow x \in S \Longrightarrow \exists\, a$∈$A.\ a \subseteq S$
shows $\exists\, A$::*nat* ⇒ *'a set*. $(\forall\, i.\ x \in A\ i \wedge$ *open* $(A\ i)) \wedge (\forall\, S.$ *open* $S \wedge x \in S \longrightarrow (\exists\, i.\ A\ i \subseteq S)$)
proof (*safe intro*!: *exI*[*of* - *from-nat-into* A])
fix i
have $A \neq \{\}$ **using** *2*[*of UNIV*] **by** *auto*
show $x \in$ *from-nat-into* A i *open* (*from-nat-into* A i)
using *range-from-nat-into-subset*[*OF* ‹$A \neq \{\}$›] *1* **by** *auto*
next
fix S
assume *open* S x∈S **from** *2*[*OF this*]
show $\exists\, i.$ *from-nat-into* A $i \subseteq S$
using *subset-range-from-nat-into*[*OF* ‹*countable* A›] **by** *auto*
qed

instance *prod* :: (*first-countable-topology*, *first-countable-topology*) *first-countable-topology*
proof
fix x :: *'a* × *'b*
obtain A **where** A:
countable A
$\bigwedge a.\ a \in A \Longrightarrow$ *fst* $x \in a$
$\bigwedge a.\ a \in A \Longrightarrow$ *open* a
$\bigwedge S.$ *open* $S \Longrightarrow$ *fst* $x \in S \Longrightarrow \exists\, a$∈$A.\ a \subseteq S$
by (*rule first-countable-basisE*[*of fst* x]) *blast*
obtain B **where** B:
countable B
$\bigwedge a.\ a \in B \Longrightarrow$ *snd* $x \in a$
$\bigwedge a.\ a \in B \Longrightarrow$ *open* a

```
    ⋀S. open S ⟹ snd x ∈ S ⟹ ∃ a∈B. a ⊆ S
   by (rule first-countable-basisE[of snd x]) blast
 show ∃ A::nat ⇒ ('a × 'b) set.
   (∀ i. x ∈ A i ∧ open (A i)) ∧ (∀ S. open S ∧ x ∈ S ⟶ (∃ i. A i ⊆ S))
 proof (rule first-countableI[of (λ(a, b). a × b) ` (A × B)], safe)
   fix a b
   assume x: a ∈ A b ∈ B
   with A(2, 3)[of a] B(2, 3)[of b] show x ∈ a × b and open (a × b)
     unfolding mem-Times-iff
     by (auto intro: open-Times)
 next
   fix S
   assume open S x ∈ S
   then obtain a' b' where a'b': open a' open b' x ∈ a' × b' a' × b' ⊆ S
     by (rule open-prod-elim)
   moreover
   from a'b' A(4)[of a'] B(4)[of b']
   obtain a b where a ∈ A a ⊆ a' b ∈ B b ⊆ b'
     by auto
   ultimately
   show ∃ a∈(λ(a, b). a × b) ` (A × B). a ⊆ S
     by (auto intro!: bexI[of - a × b] bexI[of - a] bexI[of - b])
 qed (simp add: A B)
qed

class second-countable-topology = topological-space +
 assumes ex-countable-subbasis:
   ∃ B::'a::topological-space set set. countable B ∧ open = generate-topology B
begin

lemma ex-countable-basis: ∃ B::'a set set. countable B ∧ topological-basis B
proof −
 from ex-countable-subbasis obtain B where B: countable B open = generate-topology
B
   by blast
 let ?B = Inter ` {b. finite b ∧ b ⊆ B }

 show ?thesis
 proof (intro exI conjI)
   show countable ?B
     by (intro countable-image countable-Collect-finite-subset B)
   {
     fix S
     assume open S
     then have ∃ B'⊆{b. finite b ∧ b ⊆ B}. (⋃ b∈B'. ⋂ b) = S
       unfolding B
     proof induct
       case UNIV
       show ?case by (intro exI[of - {{}}]) simp
```

```
    next
      case (Int a b)
      then obtain x y where x: a = UNION x Inter ⋀i. i ∈ x ⟹ finite i ∧ i
⊆ B
        and y: b = UNION y Inter ⋀i. i ∈ y ⟹ finite i ∧ i ⊆ B
        by blast
      show ?case
        unfolding x y Int-UN-distrib2
        by (intro exI[of - {i ∪ j| i j.  i ∈ x ∧ j ∈ y}]) (auto dest: x(2) y(2))
    next
      case (UN K)
      then have ∀k∈K. ∃B'⊆{b. finite b ∧ b ⊆ B}. UNION B' Inter = k by
auto
      then obtain k where
        ∀ka∈K. k ka ⊆ {b. finite b ∧ b ⊆ B} ∧ UNION (k ka) Inter = ka
        unfolding bchoice-iff ..
      then show ∃B'⊆{b. finite b ∧ b ⊆ B}. UNION B' Inter = ⋃K
        by (intro exI[of - UNION K k]) auto
    next
      case (Basis S)
      then show ?case
        by (intro exI[of - {{S}}]) auto
    qed
    then have (∃B'⊆Inter ` {b. finite b ∧ b ⊆ B}. ⋃B' = S)
      unfolding subset-image-iff by blast }
  then show topological-basis ?B
    unfolding topological-space-class.topological-basis-def
    by (safe intro!: topological-space-class.open-Inter)
      (simp-all add: B generate-topology.Basis subset-eq)
 qed
qed

end

sublocale second-countable-topology <
  countable-basis SOME B. countable B ∧ topological-basis B
  using someI-ex[OF ex-countable-basis]
  by unfold-locales safe

instance prod :: (second-countable-topology, second-countable-topology) second-countable-topology
proof
  obtain A :: 'a set set where countable A topological-basis A
    using ex-countable-basis by auto
  moreover
  obtain B :: 'b set set where countable B topological-basis B
    using ex-countable-basis by auto
  ultimately show ∃B::('a × 'b) set set. countable B ∧ open = generate-topology
B
    by (auto intro!: exI[of - (λ(a, b). a × b) ` (A × B)] topological-basis-prod
```

topological-basis-imp-subbasis)
qed

instance *second-countable-topology* ⊆ *first-countable-topology*
proof
fix *x* :: *'a*
def *B* ≡ *SOME B*::*'a set set*. *countable B* ∧ *topological-basis B*
then have *B*: *countable B topological-basis B*
using *countable-basis is-basis*
by (*auto simp*: *countable-basis is-basis*)
then show ∃ *A*::*nat* ⇒ *'a set*.
(∀ *i*. *x* ∈ *A i* ∧ *open* (*A i*)) ∧ (∀ *S*. *open S* ∧ *x* ∈ *S* ⟶ (∃ *i*. *A i* ⊆ *S*))
by (*intro first-countableI*[*of* {*b*∈*B*. *x* ∈ *b*}])
(*fastforce simp*: *topological-space-class.topological-basis-def*)+
qed

16.3 Polish spaces

Textbooks define Polish spaces as completely metrizable. We assume the topology to be complete for a given metric.

class *polish-space* = *complete-space* + *second-countable-topology*

16.4 General notion of a topology as a value

definition *istopology L* ⟷
L {} ∧ (∀ *S T*. *L S* ⟶ *L T* ⟶ *L* (*S* ∩ *T*)) ∧ (∀ *K*. *Ball K L* ⟶ *L* (⋃ *K*))

typedef *'a topology* = {*L*::(*'a set*) ⇒ *bool*. *istopology L*}
morphisms *openin topology*
unfolding *istopology-def* **by** *blast*

lemma *istopology-open-in*[*intro*]: *istopology*(*openin U*)
using *openin*[*of U*] **by** *blast*

lemma *topology-inverse'*: *istopology U* ⟹ *openin* (*topology U*) = *U*
using *topology-inverse*[*unfolded mem-Collect-eq*] **.**

lemma *topology-inverse-iff*: *istopology U* ⟷ *openin* (*topology U*) = *U*
using *topology-inverse*[*of U*] *istopology-open-in*[*of topology U*] **by** *auto*

lemma *topology-eq*: *T1* = *T2* ⟷ (∀ *S*. *openin T1 S* ⟷ *openin T2 S*)
proof
assume *T1* = *T2*
then show ∀ *S*. *openin T1 S* ⟷ *openin T2 S* **by** *simp*
next
assume *H*: ∀ *S*. *openin T1 S* ⟷ *openin T2 S*
then have *openin T1* = *openin T2* **by** (*simp add*: *fun-eq-iff*)
then have *topology* (*openin T1*) = *topology* (*openin T2*) **by** *simp*
then show *T1* = *T2* **unfolding** *openin-inverse* **.**

qed

Infer the "universe" from union of all sets in the topology.

definition *topspace* $T = \bigcup \{S.\ \mathit{openin}\ T\ S\}$

16.4.1 Main properties of open sets

lemma *openin-clauses*:
fixes $U :: 'a\ \mathit{topology}$
shows
$\mathit{openin}\ U\ \{\}$
$\bigwedge S\ T.\ \mathit{openin}\ U\ S \Longrightarrow \mathit{openin}\ U\ T \Longrightarrow \mathit{openin}\ U\ (S \cap T)$
$\bigwedge K.\ (\forall S \in K.\ \mathit{openin}\ U\ S) \Longrightarrow \mathit{openin}\ U\ (\bigcup K)$
using *openin*[*of U*] **unfolding** *istopology-def mem-Collect-eq* **by** *fast+*

lemma *openin-subset*[*intro*]: $\mathit{openin}\ U\ S \Longrightarrow S \subseteq \mathit{topspace}\ U$
unfolding *topspace-def* **by** *blast*

lemma *openin-empty*[*simp*]: $\mathit{openin}\ U\ \{\}$
by (*simp add*: *openin-clauses*)

lemma *openin-Int*[*intro*]: $\mathit{openin}\ U\ S \Longrightarrow \mathit{openin}\ U\ T \Longrightarrow \mathit{openin}\ U\ (S \cap T)$
using *openin-clauses* **by** *simp*

lemma *openin-Union*[*intro*]: $(\forall S \in K.\ \mathit{openin}\ U\ S) \Longrightarrow \mathit{openin}\ U\ (\bigcup K)$
using *openin-clauses* **by** *simp*

lemma *openin-Un*[*intro*]: $\mathit{openin}\ U\ S \Longrightarrow \mathit{openin}\ U\ T \Longrightarrow \mathit{openin}\ U\ (S \cup T)$
using *openin-Union*[*of* $\{S,T\}$ *U*] **by** *auto*

lemma *openin-topspace*[*intro*, *simp*]: $\mathit{openin}\ U\ (\mathit{topspace}\ U)$
by (*simp add*: *openin-Union topspace-def*)

lemma *openin-subopen*: $\mathit{openin}\ U\ S \longleftrightarrow (\forall x \in S.\ \exists T.\ \mathit{openin}\ U\ T \wedge x \in T \wedge T \subseteq S)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs*
then show *?rhs* **by** *auto*
next
assume *H*: *?rhs*
let *?t* $= \bigcup \{T.\ \mathit{openin}\ U\ T \wedge T \subseteq S\}$
have $\mathit{openin}\ U$ *?t* **by** (*simp add*: *openin-Union*)
also have *?t* $= S$ **using** *H* **by** *auto*
finally show $\mathit{openin}\ U\ S$ **.**
qed

16.4.2 Closed sets

definition $\mathit{closedin}\ U\ S \longleftrightarrow S \subseteq \mathit{topspace}\ U \wedge \mathit{openin}\ U\ (\mathit{topspace}\ U - S)$

lemma *closedin-subset*: *closedin U S* $\Longrightarrow$ *S* $\subseteq$ *topspace U*
by (*metis closedin-def*)

lemma *closedin-empty*[*simp*]: *closedin U* {}
by (*simp add*: *closedin-def*)

lemma *closedin-topspace*[*intro*, *simp*]: *closedin U* (*topspace U*)
by (*simp add*: *closedin-def*)

lemma *closedin-Un*[*intro*]: *closedin U S* $\Longrightarrow$ *closedin U T* $\Longrightarrow$ *closedin U* (*S* $\cup$ *T*)
by (*auto simp add*: *Diff-Un closedin-def*)

lemma *Diff-Inter*[*intro*]: *A* $-$ $\bigcap$ *S* = $\bigcup$ {*A* $-$ *s*|*s*. *s*$\in$*S*}
by *auto*

lemma *closedin-Inter*[*intro*]:
assumes *Ke*: *K* $\neq$ {}
and *Kc*: $\bigwedge$*S*. *S* $\in$*K* $\Longrightarrow$ *closedin U S*
shows *closedin U* ($\bigcap$ *K*)
using *Ke Kc* **unfolding** *closedin-def Diff-Inter* **by** *auto*

lemma *closedin-INT*[*intro*]:
assumes *A* $\neq$ {} $\bigwedge$*x*. *x* $\in$ *A* $\Longrightarrow$ *closedin U* (*B x*)
shows *closedin U* ($\bigcap$ *x*$\in$*A*. *B x*)
unfolding *Inter-image-eq* [*symmetric*]
apply (*rule closedin-Inter*)
using *assms*
apply *auto*
done

lemma *closedin-Int*[*intro*]: *closedin U S* $\Longrightarrow$ *closedin U T* $\Longrightarrow$ *closedin U* (*S* $\cap$ *T*)
using *closedin-Inter*[*of* {*S*,*T*} *U*] **by** *auto*

lemma *openin-closedin-eq*: *openin U S* $\longleftrightarrow$ *S* $\subseteq$ *topspace U* $\wedge$ *closedin U* (*topspace U* $-$ *S*)
apply (*auto simp add*: *closedin-def Diff-Diff-Int inf-absorb2*)
apply (*metis openin-subset subset-eq*)
done

lemma *openin-closedin*: *S* $\subseteq$ *topspace U* $\Longrightarrow$ (*openin U S* $\longleftrightarrow$ *closedin U* (*topspace U* $-$ *S*))
by (*simp add*: *openin-closedin-eq*)

lemma *openin-diff*[*intro*]:
assumes *oS*: *openin U S*
and *cT*: *closedin U T*

shows *openin U* (*S* − *T*)
proof −
have *S* − *T* = *S* ∩ (*topspace U* − *T*) **using** *openin-subset*[*of U S*] *oS cT*
by (*auto simp add*: *topspace-def openin-subset*)
then show *?thesis* **using** *oS cT*
by (*auto simp add*: *closedin-def*)
qed

lemma *closedin-diff*[*intro*]:
assumes *oS*: *closedin U S*
and *cT*: *openin U T*
shows *closedin U* (*S* − *T*)
proof −
have *S* − *T* = *S* ∩ (*topspace U* − *T*)
using *closedin-subset*[*of U S*] *oS cT* **by** (*auto simp add*: *topspace-def*)
then show *?thesis*
using *oS cT* **by** (*auto simp add*: *openin-closedin-eq*)
qed

16.4.3 Subspace topology

definition *subtopology U V* = *topology* (λ*T*. ∃ *S*. *T* = *S* ∩ *V* ∧ *openin U S*)

lemma *istopology-subtopology*: *istopology* (λ*T*. ∃ *S*. *T* = *S* ∩ *V* ∧ *openin U S*)
(**is** *istopology ?L*)
proof −
have *?L* {} **by** *blast*
{
fix *A B*
assume *A*: *?L A* **and** *B*: *?L B*
from *A B* **obtain** *Sa* **and** *Sb* **where** *Sa*: *openin U Sa A* = *Sa* ∩ *V* **and** *Sb*: *openin U Sb B* = *Sb* ∩ *V*
by *blast*
have *A* ∩ *B* = (*Sa* ∩ *Sb*) ∩ *V openin U* (*Sa* ∩ *Sb*)
using *Sa Sb* **by** *blast*+
then have *?L* (*A* ∩ *B*) **by** *blast*
}
moreover
{
fix *K*
assume *K*: *K* ⊆ *Collect ?L*
have *th0*: *Collect ?L* = (λ*S*. *S* ∩ *V*) ‘ *Collect* (*openin U*)
by *blast*
from *K*[*unfolded th0 subset-image-iff*]
obtain *Sk* **where** *Sk*: *Sk* ⊆ *Collect* (*openin U*) *K* = (λ*S*. *S* ∩ *V*) ‘ *Sk*
by *blast*
have ⋃ *K* = (⋃ *Sk*) ∩ *V*
using *Sk* **by** *auto*
moreover have *openin U* (⋃ *Sk*)

```
      using Sk by (auto simp add: subset-eq)
    ultimately have ?L (⋃ K) by blast
  }
  ultimately show ?thesis
    unfolding subset-eq mem-Collect-eq istopology-def by blast
qed

lemma openin-subtopology: openin (subtopology U V) S ⟷ (∃ T. openin U T ∧
S = T ∩ V)
  unfolding subtopology-def topology-inverse'[OF istopology-subtopology]
  by auto

lemma topspace-subtopology: topspace (subtopology U V) = topspace U ∩ V
  by (auto simp add: topspace-def openin-subtopology)

lemma closedin-subtopology: closedin (subtopology U V) S ⟷ (∃ T. closedin U
T ∧ S = T ∩ V)
  unfolding closedin-def topspace-subtopology
  by (auto simp add: openin-subtopology)

lemma openin-subtopology-refl: openin (subtopology U V) V ⟷ V ⊆ topspace
U
  unfolding openin-subtopology
  by auto (metis IntD1 in-mono openin-subset)

lemma subtopology-superset:
  assumes UV: topspace U ⊆ V
  shows subtopology U V = U
proof −
  {
    fix S
    {
      fix T
      assume T: openin U T S = T ∩ V
      from T openin-subset[OF T(1)] UV have eq: S = T
        by blast
      have openin U S
        unfolding eq using T by blast
    }
    moreover
    {
      assume S: openin U S
      then have ∃ T. openin U T ∧ S = T ∩ V
        using openin-subset[OF S] UV by auto
    }
    ultimately have (∃ T. openin U T ∧ S = T ∩ V) ⟷ openin U S
      by blast
  }
  then show ?thesis
```

unfolding *topology-eq openin-subtopology* **by** *blast*
qed

lemma *subtopology-topspace*[*simp*]: *subtopology U* (*topspace U*) = *U*
by (*simp add*: *subtopology-superset*)

lemma *subtopology-UNIV*[*simp*]: *subtopology U UNIV* = *U*
by (*simp add*: *subtopology-superset*)

16.4.4 The standard Euclidean topology

definition *euclidean* :: *'a*::*topological-space topology*
where *euclidean* = *topology open*

lemma *open-openin*: *open S* ⟷ *openin euclidean S*
unfolding *euclidean-def*
apply (*rule cong*[**where** *x*=*S* **and** *y*=*S*])
apply (*rule topology-inverse*[*symmetric*])
apply (*auto simp add*: *istopology-def*)
done

lemma *topspace-euclidean*: *topspace euclidean* = *UNIV*
apply (*simp add*: *topspace-def*)
apply (*rule set-eqI*)
apply (*auto simp add*: *open-openin*[*symmetric*])
done

lemma *topspace-euclidean-subtopology*[*simp*]: *topspace* (*subtopology euclidean S*)
= *S*
by (*simp add*: *topspace-euclidean topspace-subtopology*)

lemma *closed-closedin*: *closed S* ⟷ *closedin euclidean S*
by (*simp add*: *closed-def closedin-def topspace-euclidean open-openin Compl-eq-Diff-UNIV*)

lemma *open-subopen*: *open S* ⟷ (∀ *x*∈*S*. ∃ *T*. *open T* ∧ *x* ∈ *T* ∧ *T* ⊆ *S*)
by (*simp add*: *open-openin openin-subopen*[*symmetric*])

Basic "localization" results are handy for connectedness.

lemma *openin-open*: *openin* (*subtopology euclidean U*) *S* ⟷ (∃ *T*. *open T* ∧ (*S*
= *U* ∩ *T*))
by (*auto simp add*: *openin-subtopology open-openin*[*symmetric*])

lemma *openin-open-Int*[*intro*]: *open S* ⟹ *openin* (*subtopology euclidean U*) (*U*
∩ *S*)
by (*auto simp add*: *openin-open*)

lemma *open-openin-trans*[*trans*]:
open S ⟹ *open T* ⟹ *T* ⊆ *S* ⟹ *openin* (*subtopology euclidean S*) *T*
by (*metis Int-absorb1 openin-open-Int*)

lemma *open-subset*: $S \subseteq T \Longrightarrow$ *open* $S \Longrightarrow$ *openin* (*subtopology euclidean* T) S
by (*auto simp add*: *openin-open*)

lemma *closedin-closed*: *closedin* (*subtopology euclidean* U) $S \longleftrightarrow (\exists\, T.$ *closed* $T \wedge S = U \cap T)$
by (*simp add*: *closedin-subtopology closed-closedin Int-ac*)

lemma *closedin-closed-Int*: *closed* $S \Longrightarrow$ *closedin* (*subtopology euclidean* U) $(U \cap S)$
by (*metis closedin-closed*)

lemma *closed-closedin-trans*:
closed $S \Longrightarrow$ *closed* $T \Longrightarrow T \subseteq S \Longrightarrow$ *closedin* (*subtopology euclidean* S) T
by (*metis closedin-closed inf.absorb2*)

lemma *closed-subset*: $S \subseteq T \Longrightarrow$ *closed* $S \Longrightarrow$ *closedin* (*subtopology euclidean* T) S
by (*auto simp add*: *closedin-closed*)

lemma *openin-euclidean-subtopology-iff*:
fixes S U :: *'a*::*metric-space set*
shows *openin* (*subtopology euclidean* U) $S \longleftrightarrow$
$S \subseteq U \wedge (\forall\, x{\in}S.\ \exists\, e{>}0.\ \forall\, x'{\in}U.\ dist\ x'\ x < e \longrightarrow x'{\in}\ S)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *openin-open open-dist* **by** *blast*
next
def $T \equiv \{x.\ \exists\, a{\in}S.\ \exists\, d{>}0.\ (\forall\, y{\in}U.\ dist\ y\ a < d \longrightarrow y \in S) \wedge dist\ x\ a < d\}$
have *1*: $\forall\, x{\in}T.\ \exists\, e{>}0.\ \forall\, y.\ dist\ y\ x < e \longrightarrow y \in T$
unfolding *T-def*
apply *clarsimp*
apply (*rule-tac* $x{=}d - dist\ x\ a$ **in** *exI*)
apply (*clarsimp simp add*: *less-diff-eq*)
by (*metis dist-commute dist-triangle-lt*)
assume *?rhs* **then have** *2*: $S = U \cap T$
unfolding *T-def*
by *auto* (*metis dist-self*)
from *1 2* **show** *?lhs*
unfolding *openin-open open-dist* **by** *fast*
qed

lemma *connected-open-in*:
connected s $\longleftrightarrow$
$\sim$($\exists$ *e1 e2*. *openin* (*subtopology euclidean s*) *e1* $\wedge$
openin (*subtopology euclidean s*) *e2* $\wedge$
$s \subseteq e1 \cup e2 \wedge e1 \cap e2 = \{\} \wedge e1 \neq \{\} \wedge e2 \neq \{\}$)

apply (*simp add*: *connected-def openin-open*, *safe*)
apply (*simp-all*, *blast+*) — slow
done

lemma *connected-open-in-eq*:
connected s ⟷
~(∃ *e1 e2*. *openin* (*subtopology euclidean s*) *e1* ∧
openin (*subtopology euclidean s*) *e2* ∧
e1 ∪ *e2* = *s* ∧ *e1* ∩ *e2* = {} ∧
e1 ≠ {} ∧ *e2* ≠ {})
apply (*simp add*: *connected-open-in*, *safe*)
apply *blast*
by (*metis Int-lower1 Un-subset-iff openin-open subset-antisym*)

lemma *connected-closed-in*:
connected s ⟷
~(∃ *e1 e2*.
closedin (*subtopology euclidean s*) *e1* ∧
closedin (*subtopology euclidean s*) *e2* ∧
s ⊆ *e1* ∪ *e2* ∧ *e1* ∩ *e2* = {} ∧
e1 ≠ {} ∧ *e2* ≠ {})
proof −
{ fix *A B x x'*
assume *s-sub*: *s* ⊆ *A* ∪ *B*
and *disj*: *A* ∩ *B* ∩ *s* = {}
and *x*: *x* ∈ *s x* ∈ *B* **and** *x'*: *x'* ∈ *s x'* ∈ *A*
and *cl*: *closed A closed B*
assume ∀ *e1*. (∀ *T*. *closed T* ⟶ *e1* ≠ *s* ∩ *T*) ∨ (∀ *e2*. *e1* ∩ *e2* = {} ⟶ *s* ⊆ *e1* ∪ *e2* ⟶ (∀ *T*. *closed T* ⟶ *e2* ≠ *s* ∩ *T*) ∨ *e1* = {} ∨ *e2* = {})
then have ⋀*C D*. *s* ∩ *C* = {} ∨ *s* ∩ *D* = {} ∨ *s* ∩ (*C* ∩ (*s* ∩ *D*)) ≠ {} ∨ ¬ *s* ⊆ *s* ∩ (*C* ∪ *D*) ∨ ¬ *closed C* ∨ ¬ *closed D*
by (*metis* (*no-types*) *Int-Un-distrib Int-assoc*)
moreover have *s* ∩ (*A* ∩ *B*) = {} *s* ∩ (*A* ∪ *B*) = *s s* ∩ *B* ≠ {}
using *disj s-sub x* **by** *blast+*
ultimately have *s* ∩ *A* = {}
using *cl* **by** (*metis inf.left-commute inf-bot-right order-refl*)
then have *False*
using *x'* **by** *blast*
} note ∗ = *this*
show *?thesis*
apply (*simp add*: *connected-closed closedin-closed*)
apply (*safe*; *simp*)
apply *blast*
apply (*blast intro*: ∗)
done
qed

lemma *connected-closed-in-eq*:
connected s ⟷

~(∃ *e1 e2*.
closedin (*subtopology euclidean s*) *e1* ∧
closedin (*subtopology euclidean s*) *e2* ∧
e1 ∪ *e2* = *s* ∧ *e1* ∩ *e2* = {} ∧
e1 ≠ {} ∧ *e2* ≠ {})
apply (*simp add*: *connected-closed-in*, *safe*)
apply *blast*
by (*metis Int-lower1 Un-subset-iff closedin-closed subset-antisym*)

These "transitivity" results are handy too

lemma *openin-trans*[*trans*]:
openin (*subtopology euclidean T*) *S* ⟹ *openin* (*subtopology euclidean U*) *T* ⟹
openin (*subtopology euclidean U*) *S*
unfolding *open-openin openin-open* **by** *blast*

lemma *openin-open-trans*: *openin* (*subtopology euclidean T*) *S* ⟹ *open T* ⟹ *open S*
by (*auto simp add*: *openin-open intro*: *openin-trans*)

lemma *closedin-trans*[*trans*]:
closedin (*subtopology euclidean T*) *S* ⟹ *closedin* (*subtopology euclidean U*) *T* ⟹
closedin (*subtopology euclidean U*) *S*
by (*auto simp add*: *closedin-closed closed-closedin closed-Inter Int-assoc*)

lemma *closedin-closed-trans*: *closedin* (*subtopology euclidean T*) *S* ⟹ *closed T* ⟹ *closed S*
by (*auto simp add*: *closedin-closed intro*: *closedin-trans*)

lemma *openin-subtopology-inter-subset*:
openin (*subtopology euclidean u*) (*u* ∩ *s*) ∧ *v* ⊆ *u* ⟹ *openin* (*subtopology euclidean v*) (*v* ∩ *s*)
by (*auto simp*: *openin-subtopology*)

lemma *openin-open-eq*: *open s* ⟹ (*openin* (*subtopology euclidean s*) *t* ⟷ *open t* ∧ *t* ⊆ *s*)
using *open-subset openin-open-trans openin-subset* **by** *fastforce*

16.5 Open and closed balls

definition *ball* :: *'a*::*metric-space* ⇒ *real* ⇒ *'a set*
where *ball x e* = {*y*. *dist x y* < *e*}

definition *cball* :: *'a*::*metric-space* ⇒ *real* ⇒ *'a set*
where *cball x e* = {*y*. *dist x y* ≤ *e*}

definition *sphere* :: *'a*::*metric-space* ⇒ *real* ⇒ *'a set*
where *sphere x e* = {*y*. *dist x y* = *e*}

lemma *mem-ball* [*simp*]: $y \in ball\ x\ e \longleftrightarrow dist\ x\ y < e$
by (*simp add*: *ball-def*)

lemma *mem-cball* [*simp*]: $y \in cball\ x\ e \longleftrightarrow dist\ x\ y \leq e$
by (*simp add*: *cball-def*)

lemma *mem-sphere* [*simp*]: $y \in sphere\ x\ e \longleftrightarrow dist\ x\ y = e$
by (*simp add*: *sphere-def*)

lemma *ball-trivial* [*simp*]: $ball\ x\ 0 = \{\}$
by (*simp add*: *ball-def*)

lemma *cball-trivial* [*simp*]: $cball\ x\ 0 = \{x\}$
by (*simp add*: *cball-def*)

lemma *mem-ball-0* [*simp*]:
fixes $x :: 'a$::*real-normed-vector*
shows $x \in ball\ 0\ e \longleftrightarrow norm\ x < e$
by (*simp add*: *dist-norm*)

lemma *mem-cball-0* [*simp*]:
fixes $x :: 'a$::*real-normed-vector*
shows $x \in cball\ 0\ e \longleftrightarrow norm\ x \leq e$
by (*simp add*: *dist-norm*)

lemma *centre-in-ball* [*simp*]: $x \in ball\ x\ e \longleftrightarrow 0 < e$
by *simp*

lemma *centre-in-cball* [*simp*]: $x \in cball\ x\ e \longleftrightarrow 0 \leq e$
by *simp*

lemma *ball-subset-cball* [*simp*,*intro*]: $ball\ x\ e \subseteq cball\ x\ e$
by (*simp add*: *subset-eq*)

lemma *sphere-cball* [*simp*,*intro*]: $sphere\ z\ r \subseteq cball\ z\ r$
by *force*

lemma *subset-ball*[*intro*]: $d \leq e \Longrightarrow ball\ x\ d \subseteq ball\ x\ e$
by (*simp add*: *subset-eq*)

lemma *subset-cball*[*intro*]: $d \leq e \Longrightarrow cball\ x\ d \subseteq cball\ x\ e$
by (*simp add*: *subset-eq*)

lemma *ball-max-Un*: $ball\ a\ (max\ r\ s) = ball\ a\ r \cup ball\ a\ s$
by (*simp add*: *set-eq-iff*) *arith*

lemma *ball-min-Int*: $ball\ a\ (min\ r\ s) = ball\ a\ r \cap ball\ a\ s$
by (*simp add*: *set-eq-iff*)

lemma *cball-diff-eq-sphere*: *cball a r − ball a r = {x. dist x a = r}*
by (*auto simp*: *cball-def ball-def dist-commute*)

lemma *open-ball* [*intro*, *simp*]: *open* (*ball x e*)
proof −
have *open* (*dist x −‘ {..<e}*)
by (*intro open-vimage open-lessThan continuous-intros*)
also have *dist x −‘ {..<e} = ball x e*
by *auto*
finally show *?thesis* **.**
qed

lemma *open-contains-ball*: *open S ⟷ (∀ x∈S. ∃ e>0. ball x e ⊆ S)*
unfolding *open-dist subset-eq mem-ball Ball-def dist-commute* **..**

lemma *openE*[*elim?*]:
assumes *open S x∈S*
obtains *e* **where** *e>0 ball x e ⊆ S*
using *assms* **unfolding** *open-contains-ball* **by** *auto*

lemma *open-contains-ball-eq*: *open S ⟹ ∀ x. x∈S ⟷ (∃ e>0. ball x e ⊆ S)*
by (*metis open-contains-ball subset-eq centre-in-ball*)

lemma *ball-eq-empty*[*simp*]: *ball x e = {} ⟷ e ≤ 0*
unfolding *mem-ball set-eq-iff*
apply (*simp add*: *not-less*)
apply (*metis zero-le-dist order-trans dist-self*)
done

lemma *ball-empty*: *e ≤ 0 ⟹ ball x e = {}* **by** *simp*

lemma *euclidean-dist-l2*:
fixes *x y* :: *'a* :: *euclidean-space*
shows *dist x y = setL2* (*λi. dist* (*x · i*) (*y · i*)) *Basis*
unfolding *dist-norm norm-eq-sqrt-inner setL2-def*
by (*subst euclidean-inner*) (*simp add*: *power2-eq-square inner-diff-left*)

lemma *eventually-nhds-ball*: *d > 0 ⟹ eventually* (*λx. x ∈ ball z d*) (*nhds z*)
by (*rule eventually-nhds-in-open*) *simp-all*

lemma *eventually-at-ball*: *d > 0 ⟹ eventually* (*λt. t ∈ ball z d ∧ t ∈ A*) (*at z within A*)
unfolding *eventually-at* **by** (*intro exI*[*of - d*]) (*simp-all add*: *dist-commute*)

lemma *eventually-at-ball'*: *d > 0 ⟹ eventually* (*λt. t ∈ ball z d ∧ t ≠ z ∧ t ∈ A*) (*at z within A*)
unfolding *eventually-at* **by** (*intro exI*[*of - d*]) (*simp-all add*: *dist-commute*)

16.6 Boxes

abbreviation *One* :: *'a*::*euclidean-space*
where $One \equiv \sum Basis$

definition (**in** *euclidean-space*) *eucl-less* (**infix** $<e$ *50*)
where $eucl\text{-}less\ a\ b \longleftrightarrow (\forall i{\in}Basis.\ a \cdot i < b \cdot i)$

definition *box-eucl-less*: $box\ a\ b = \{x.\ a <e\ x \wedge x <e\ b\}$
definition $cbox\ a\ b = \{x.\ \forall i{\in}Basis.\ a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i\}$

lemma *box-def*: $box\ a\ b = \{x.\ \forall i{\in}Basis.\ a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i\}$
and *in-box-eucl-less*: $x \in box\ a\ b \longleftrightarrow a <e\ x \wedge x <e\ b$
and *mem-box*: $x \in box\ a\ b \longleftrightarrow (\forall i{\in}Basis.\ a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i)$
$x \in cbox\ a\ b \longleftrightarrow (\forall i{\in}Basis.\ a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i)$
by (*auto simp*: *box-eucl-less eucl-less-def cbox-def*)

lemma *cbox-Pair-eq*: $cbox\ (a,\ c)\ (b,\ d) = cbox\ a\ b \times cbox\ c\ d$
by (*force simp*: *cbox-def Basis-prod-def*)

lemma *cbox-Pair-iff* [*iff*]: $(x,\ y) \in cbox\ (a,\ c)\ (b,\ d) \longleftrightarrow x \in cbox\ a\ b \wedge y \in cbox\ c\ d$
by (*force simp*: *cbox-Pair-eq*)

lemma *cbox-Pair-eq-0*: $cbox\ (a,\ c)\ (b,\ d) = \{\} \longleftrightarrow cbox\ a\ b = \{\} \vee cbox\ c\ d = \{\}$
by (*force simp*: *cbox-Pair-eq*)

lemma *swap-cbox-Pair* [*simp*]: $prod.swap\ `\ cbox\ (c,\ a)\ (d,\ b) = cbox\ (a,c)\ (b,d)$
by *auto*

lemma *mem-box-real*[*simp*]:
$(x{::}real) \in box\ a\ b \longleftrightarrow a < x \wedge x < b$
$(x{::}real) \in cbox\ a\ b \longleftrightarrow a \leq x \wedge x \leq b$
by (*auto simp*: *mem-box*)

lemma *box-real*[*simp*]:
fixes *a b*:: *real*
shows $box\ a\ b = \{a <..< b\}\ cbox\ a\ b = \{a\ ..\ b\}$
by *auto*

lemma *box-Int-box*:
fixes *a* :: *'a*::*euclidean-space*
shows $box\ a\ b \cap box\ c\ d =$
$box\ (\sum i{\in}Basis.\ max\ (a{\cdot}i)\ (c{\cdot}i) *_R i)\ (\sum i{\in}Basis.\ min\ (b{\cdot}i)\ (d{\cdot}i) *_R i)$
unfolding *set-eq-iff* **and** *Int-iff* **and** *mem-box* **by** *auto*

lemma *rational-boxes*:
fixes *x* :: *'a*::*euclidean-space*
assumes $e > 0$

shows $\exists\, a\ b.\ (\forall\, i{\in}\mathit{Basis}.\ a \cdot i \in \mathbb{Q} \wedge b \cdot i \in \mathbb{Q}\ ) \wedge x \in \mathit{box}\ a\ b \wedge \mathit{box}\ a\ b \subseteq \mathit{ball}\ x\ e$
proof −
def $e' \equiv e\ /\ (2 * \mathit{sqrt}\ (\mathit{real}\ (\mathit{DIM}\ ('a))))$
then have e: $e' > 0$
using *assms* **by** (*auto simp*: *DIM-positive*)
have $\forall\, i.\ \exists\, y.\ y \in \mathbb{Q} \wedge y < x \cdot i \wedge x \cdot i - y < e'$ (**is** $\forall\, i.\ ?th\ i$)
proof
fix i
from *Rats-dense-in-real*[*of* $x \cdot i - e'\ x \cdot i$] e
show *?th i* **by** *auto*
qed
from *choice*[*OF this*] **obtain** a **where**
a: $\forall\, xa.\ a\ xa \in \mathbb{Q} \wedge a\ xa < x \cdot xa \wedge x \cdot xa - a\ xa < e'$ **..**
have $\forall\, i.\ \exists\, y.\ y \in \mathbb{Q} \wedge x \cdot i < y \wedge y - x \cdot i < e'$ (**is** $\forall\, i.\ ?th\ i$)
proof
fix i
from *Rats-dense-in-real*[*of* $x \cdot i\ x \cdot i + e'$] e
show *?th i* **by** *auto*
qed
from *choice*[*OF this*] **obtain** b **where**
b: $\forall\, xa.\ b\ xa \in \mathbb{Q} \wedge x \cdot xa < b\ xa \wedge b\ xa - x \cdot xa < e'$ **..**
let *?a* $= \sum i{\in}\mathit{Basis}.\ a\ i *_R i$ **and** *?b* $= \sum i{\in}\mathit{Basis}.\ b\ i *_R i$
show *?thesis*
proof (*rule exI*[*of* - *?a*], *rule exI*[*of* - *?b*], *safe*)
fix $y :: 'a$
assume $*$: $y \in \mathit{box}\ ?a\ ?b$
have $\mathit{dist}\ x\ y = \mathit{sqrt}\ (\sum i{\in}\mathit{Basis}.\ (\mathit{dist}\ (x \cdot i)\ (y \cdot i))^2)$
unfolding *setL2-def*[*symmetric*] **by** (*rule euclidean-dist-l2*)
also have $\ldots < \mathit{sqrt}\ (\sum (i::'a){\in}\mathit{Basis}.\ e\hat{}2\ /\ \mathit{real}\ (\mathit{DIM}('a)))$
proof (*rule real-sqrt-less-mono*, *rule setsum-strict-mono*)
fix $i :: 'a$
assume i: $i \in \mathit{Basis}$
have $a\ i < y{\cdot}i \wedge y{\cdot}i < b\ i$
using $*\ i$ **by** (*auto simp*: *box-def*)
moreover have $a\ i < x{\cdot}i\ x{\cdot}i - a\ i < e'$
using a **by** *auto*
moreover have $x{\cdot}i < b\ i\ b\ i - x{\cdot}i < e'$
using b **by** *auto*
ultimately have $|x{\cdot}i - y{\cdot}i| < 2 * e'$
by *auto*
then have $\mathit{dist}\ (x \cdot i)\ (y \cdot i) < e/\mathit{sqrt}\ (\mathit{real}\ (\mathit{DIM}('a)))$
unfolding e'*-def* **by** (*auto simp*: *dist-real-def*)
then have $(\mathit{dist}\ (x \cdot i)\ (y \cdot i))^2 < (e/\mathit{sqrt}\ (\mathit{real}\ (\mathit{DIM}('a))))^2$
by (*rule power-strict-mono*) *auto*
then show $(\mathit{dist}\ (x \cdot i)\ (y \cdot i))^2 < e^2\ /\ \mathit{real}\ \mathit{DIM}('a)$
by (*simp add*: *power-divide*)
qed *auto*
also have $\ldots = e$

```
    using ‹0 < e› by simp
  finally show y ∈ ball x e
    by (auto simp: ball-def)
 qed (insert a b, auto simp: box-def)
qed

lemma open-UNION-box:
 fixes M :: 'a::euclidean-space set
 assumes open M
 defines a' ≡ λf :: 'a ⇒ real × real. (∑(i::'a)∈Basis. fst (f i) *R i)
 defines b' ≡ λf :: 'a ⇒ real × real. (∑(i::'a)∈Basis. snd (f i) *R i)
 defines I ≡ {f∈Basis →E ℚ × ℚ. box (a' f) (b' f) ⊆ M}
 shows M = (⋃f∈I. box (a' f) (b' f))
proof −
 have x ∈ (⋃f∈I. box (a' f) (b' f)) if x ∈ M for x
 proof −
  obtain e where e: e > 0 ball x e ⊆ M
    using openE[OF ‹open M› ‹x ∈ M›] by auto
  moreover obtain a b where ab:
    x ∈ box a b
    ∀i ∈ Basis. a • i ∈ ℚ
    ∀i∈Basis. b • i ∈ ℚ
    box a b ⊆ ball x e
    using rational-boxes[OF e(1)] by metis
  ultimately show ?thesis
    by (intro UN-I[of λi∈Basis. (a • i, b • i)])
       (auto simp: euclidean-representation I-def a'-def b'-def)
 qed
 then show ?thesis by (auto simp: I-def)
qed

lemma box-eq-empty:
 fixes a :: 'a::euclidean-space
 shows (box a b = {} ⟷ (∃i∈Basis. b•i ≤ a•i)) (is ?th1)
  and (cbox a b = {} ⟷ (∃i∈Basis. b•i < a•i)) (is ?th2)
proof −
 {
  fix i x
  assume i: i∈Basis and as:b•i ≤ a•i and x:x∈box a b
  then have a • i < x • i ∧ x • i < b • i
    unfolding mem-box by (auto simp: box-def)
  then have a•i < b•i by auto
  then have False using as by auto
 }
 moreover
 {
  assume as: ∀i∈Basis. ¬ (b•i ≤ a•i)
  let ?x = (1/2) *R (a + b)
  {
```

```
    fix i :: 'a
    assume i: i ∈ Basis
    have a•i < b•i
      using as[THEN bspec[where x=i]] i by auto
    then have a•i < ((1/2) *R (a+b)) • i ((1/2) *R (a+b)) • i < b•i
      by (auto simp: inner-add-left)
  }
  then have box a b ≠ {}
    using mem-box(1)[of ?x a b] by auto
 }
 ultimately show ?th1 by blast

 {
  fix i x
  assume i: i ∈ Basis and as:b•i < a•i and x:x∈cbox a b
  then have a • i ≤ x • i ∧ x • i ≤ b • i
    unfolding mem-box by auto
  then have a•i ≤ b•i by auto
  then have False using as by auto
 }
 moreover
 {
  assume as:∀ i∈Basis. ¬ (b•i < a•i)
  let ?x = (1/2) *R (a + b)
  {
   fix i :: 'a
   assume i:i ∈ Basis
   have a•i ≤ b•i
     using as[THEN bspec[where x=i]] i by auto
   then have a•i ≤ ((1/2) *R (a+b)) • i ((1/2) *R (a+b)) • i ≤ b•i
     by (auto simp: inner-add-left)
  }
  then have cbox a b ≠ {}
    using mem-box(2)[of ?x a b] by auto
 }
 ultimately show ?th2 by blast
qed

lemma box-ne-empty:
 fixes a :: 'a::euclidean-space
 shows cbox a b ≠ {} ⟷ (∀ i∈Basis. a•i ≤ b•i)
 and box a b ≠ {} ⟷ (∀ i∈Basis. a•i < b•i)
 unfolding box-eq-empty[of a b] by fastforce+

lemma
 fixes a :: 'a::euclidean-space
 shows cbox-sing: cbox a a = {a}
  and box-sing: box a a = {}
 unfolding set-eq-iff mem-box eq-iff [symmetric]
```

by (*auto intro*!: *euclidean-eqI*[**where** *'a='a*])
(*metis all-not-in-conv nonempty-Basis*)

lemma *subset-box-imp*:
fixes *a* :: *'a*::*euclidean-space*
shows (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) ⟹ *cbox c d* ⊆ *cbox a b*
and (∀ *i*∈*Basis*. *a*•*i* < *c*•*i* ∧ *d*•*i* < *b*•*i*) ⟹ *cbox c d* ⊆ *box a b*
and (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) ⟹ *box c d* ⊆ *cbox a b*
and (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) ⟹ *box c d* ⊆ *box a b*
unfolding *subset-eq*[*unfolded Ball-def*] **unfolding** *mem-box*
by (*best intro*: *order-trans less-le-trans le-less-trans less-imp-le*)+

lemma *box-subset-cbox*:
fixes *a* :: *'a*::*euclidean-space*
shows *box a b* ⊆ *cbox a b*
unfolding *subset-eq* [*unfolded Ball-def*] *mem-box*
by (*fast intro*: *less-imp-le*)

lemma *subset-box*:
fixes *a* :: *'a*::*euclidean-space*
shows *cbox c d* ⊆ *cbox a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* ≤ *d*•*i*) −−> (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) (**is** *?th1*)
and *cbox c d* ⊆ *box a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* ≤ *d*•*i*) −−> (∀ *i*∈*Basis*. *a*•*i* < *c*•*i* ∧ *d*•*i* < *b*•*i*) (**is** *?th2*)
and *box c d* ⊆ *cbox a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* < *d*•*i*) −−> (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) (**is** *?th3*)
and *box c d* ⊆ *box a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* < *d*•*i*) −−> (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) (**is** *?th4*)
proof −
show *?th1*
unfolding *subset-eq* **and** *Ball-def* **and** *mem-box*
by (*auto intro*: *order-trans*)
show *?th2*
unfolding *subset-eq* **and** *Ball-def* **and** *mem-box*
by (*auto intro*: *le-less-trans less-le-trans order-trans less-imp-le*)
{
assume *as*: *box c d* ⊆ *cbox a b* ∀ *i*∈*Basis*. *c*•*i* < *d*•*i*
then have *box c d* ≠ {}
unfolding *box-eq-empty* **by** *auto*
fix *i* :: *'a*
assume *i*: *i* ∈ *Basis*

{
let *?x* = (∑ *j*∈*Basis*. (*if j*=*i then* ((*min* (*a*•*j*) (*d*•*j*))+*c*•*j*)/*2 else* (*c*•*j*+*d*•*j*)/*2*) *∗R j*)::*'a*
assume *as2*: *a*•*i* > *c*•*i*
{
fix *j* :: *'a*
assume *j*: *j* ∈ *Basis*

```
      then have c · j < ?x · j ∧ ?x · j < d · j
        apply (cases j = i)
        using as(2)[THEN bspec[where x=j]] i
        apply (auto simp add: as2)
        done
      }
      then have ?x∈box c d
        using i unfolding mem-box by auto
      moreover
      have ?x ∉ cbox a b
        unfolding mem-box
        apply auto
        apply (rule-tac x=i in bexI)
        using as(2)[THEN bspec[where x=i]] and as2 i
        apply auto
        done
      ultimately have False using as by auto
    }
    then have a•i ≤ c•i by (rule ccontr) auto
    moreover
    {
     let ?x = (∑ j∈Basis. (if j=i then ((max (b•j) (c•j))+d•j)/2 else (c•j+d•j)/2)
*R j)::'a
      assume as2: b•i < d•i
      {
        fix j :: 'a
        assume j∈Basis
        then have d · j > ?x · j ∧ ?x · j > c · j
          apply (cases j = i)
          using as(2)[THEN bspec[where x=j]]
          apply (auto simp add: as2)
          done
      }
      then have ?x∈box c d
        unfolding mem-box by auto
      moreover
      have ?x∉cbox a b
        unfolding mem-box
        apply auto
        apply (rule-tac x=i in bexI)
        using as(2)[THEN bspec[where x=i]] and as2 using i
        apply auto
        done
      ultimately have False using as by auto
    }
    then have b•i ≥ d•i by (rule ccontr) auto
    ultimately
    have a•i ≤ c•i ∧ d•i ≤ b•i by auto
  } note part1 = this
```

show *?th3*
unfolding *subset-eq* **and** *Ball-def* **and** *mem-box*
apply (*rule*, *rule*, *rule*, *rule*)
apply (*rule part1*)
unfolding *subset-eq* **and** *Ball-def* **and** *mem-box*
prefer *4*
apply *auto*
apply (*erule-tac x=xa* **in** *allE*, *erule-tac x=xa* **in** *allE*, *fastforce*)+
done
{
assume *as*: *box c d* ⊆ *box a b* ∀ *i*∈*Basis*. *c*•*i* < *d*•*i*
fix *i* :: *'a*
assume *i*:*i*∈*Basis*
from *as*(*1*) **have** *box c d* ⊆ *cbox a b*
using *box-subset-cbox*[*of a b*] **by** *auto*
then have *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*
using *part1* **and** *as*(*2*) **using** *i* **by** *auto*
} note ∗ = *this*
show *?th4*
unfolding *subset-eq* **and** *Ball-def* **and** *mem-box*
apply (*rule*, *rule*, *rule*, *rule*)
apply (*rule* ∗)
unfolding *subset-eq* **and** *Ball-def* **and** *mem-box*
prefer *4*
apply *auto*
apply (*erule-tac x=xa* **in** *allE*, *simp*)+
done
qed

lemma *inter-interval*:
fixes *a* :: *'a*::*euclidean-space*
shows *cbox a b* ∩ *cbox c d* =
cbox (∑ *i*∈*Basis*. *max* (*a*•*i*) (*c*•*i*) ∗$_R$ *i*) (∑ *i*∈*Basis*. *min* (*b*•*i*) (*d*•*i*) ∗$_R$ *i*)
unfolding *set-eq-iff* **and** *Int-iff* **and** *mem-box*
by *auto*

lemma *disjoint-interval*:
fixes *a*::*'a*::*euclidean-space*
shows *cbox a b* ∩ *cbox c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* < *a*•*i* ∨ *d*•*i* < *c*•*i* ∨ *b*•*i* < *c*•*i* ∨ *d*•*i* < *a*•*i*)) (**is** *?th1*)
and *cbox a b* ∩ *box c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* < *a*•*i* ∨ *d*•*i* ≤ *c*•*i* ∨ *b*•*i* ≤ *c*•*i* ∨ *d*•*i* ≤ *a*•*i*)) (**is** *?th2*)
and *box a b* ∩ *cbox c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* ≤ *a*•*i* ∨ *d*•*i* < *c*•*i* ∨ *b*•*i* ≤ *c*•*i* ∨ *d*•*i* ≤ *a*•*i*)) (**is** *?th3*)
and *box a b* ∩ *box c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* ≤ *a*•*i* ∨ *d*•*i* ≤ *c*•*i* ∨ *b*•*i* ≤ *c*•*i* ∨ *d*•*i* ≤ *a*•*i*)) (**is** *?th4*)
proof −
let *?z* = (∑ *i*∈*Basis*. (((*max* (*a*•*i*) (*c*•*i*)) + (*min* (*b*•*i*) (*d*•*i*))) / *2*) ∗$_R$ *i*)::*'a*
have ∗∗: ⋀*P Q*. (⋀*i* :: *'a*. *i* ∈ *Basis* ⟹ *Q ?z i* ⟹ *P i*) ⟹

$(\bigwedge i\ x :: 'a.\ i \in Basis \Longrightarrow P\ i \Longrightarrow Q\ x\ i) \Longrightarrow (\forall x.\ \exists i{\in}Basis.\ Q\ x\ i) \longleftrightarrow (\exists i{\in}Basis.\ P\ i)$
by *blast*
note $* =$ *set-eq-iff Int-iff empty-iff mem-box ball-conj-distrib*[*symmetric*] *eq-False ball-simps*(*10*)
show *?th1* **unfolding** $*$ **by** (*intro* $**$) *auto*
show *?th2* **unfolding** $*$ **by** (*intro* $**$) *auto*
show *?th3* **unfolding** $*$ **by** (*intro* $**$) *auto*
show *?th4* **unfolding** $*$ **by** (*intro* $**$) *auto*
qed

lemma *UN-box-eq-UNIV*: $(\bigcup i{::}nat.\ box\ (-\ (real\ i *_R One))\ (real\ i *_R One)) = UNIV$
proof −
have $|x \cdot b| < real\text{-}of\text{-}int\ (\lceil Max\ ((\lambda b.\ |x \cdot b|)\ `Basis)\rceil + 1)$
if [*simp*]: $b \in Basis$ **for** $x\ b :: 'a$
proof −
have $|x \cdot b| \le real\text{-}of\text{-}int\ \lceil |x \cdot b|\rceil$
by (*rule le-of-int-ceiling*)
also have $\ldots \le real\text{-}of\text{-}int\ \lceil Max\ ((\lambda b.\ |x \cdot b|)\ `Basis)\rceil$
by (*auto intro*!: *ceiling-mono*)
also have $\ldots < real\text{-}of\text{-}int\ (\lceil Max\ ((\lambda b.\ |x \cdot b|)\ `Basis)\rceil + 1)$
by *simp*
finally show *?thesis* **.**
qed
then have $\exists n{::}nat.\ \forall b{\in}Basis.\ |x \cdot b| < real\ n$ **for** $x :: 'a$
by (*metis order.strict-trans reals-Archimedean2*)
moreover have $\bigwedge x\ b{::}'a.\ \bigwedge n{::}nat.\ |x \cdot b| < real\ n \longleftrightarrow -\ real\ n < x \cdot b \wedge x \cdot b < real\ n$
by *auto*
ultimately show *?thesis*
by (*auto simp*: *box-def inner-setsum-left inner-Basis setsum.If-cases*)
qed

Intervals in general, including infinite and mixtures of open and closed.

definition *is-interval* $(s{::}('a{::}euclidean\text{-}space)\ set) \longleftrightarrow$
$(\forall a{\in}s.\ \forall b{\in}s.\ \forall x.\ (\forall i{\in}Basis.\ ((a{\cdot}i \le x{\cdot}i \wedge x{\cdot}i \le b{\cdot}i) \vee (b{\cdot}i \le x{\cdot}i \wedge x{\cdot}i \le a{\cdot}i))) \longrightarrow x \in s)$

lemma *is-interval-cbox*: *is-interval* (*cbox a* $(b{::}'a{::}euclidean\text{-}space)$) (**is** *?th1*)
and *is-interval-box*: *is-interval* (*box a b*) (**is** *?th2*)
unfolding *is-interval-def mem-box Ball-def atLeastAtMost-iff*
by (*meson order-trans le-less-trans less-le-trans less-trans*)+

lemma *is-interval-empty* [*iff*]: *is-interval* {}
unfolding *is-interval-def* **by** *simp*

lemma *is-interval-univ* [*iff*]: *is-interval UNIV*
unfolding *is-interval-def* **by** *simp*

lemma *mem-is-intervalI*:
assumes *is-interval s*
assumes $a \in s$ $b \in s$
assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i \vee b \cdot i \leq x \cdot i \wedge x \cdot i \leq a \cdot i$
shows $x \in s$
by (*rule assms*(*1*)[*simplified is-interval-def*, *rule-format*, *OF assms*(*2*,*3*,*4*)])

lemma *interval-subst*:
fixes *S*::'*a*::*euclidean-space set*
assumes *is-interval S*
assumes $x \in S$ y $j \in S$
assumes $j \in Basis$
shows $(\sum i{\in}Basis.\ (if\ i = j\ then\ y\ i \cdot i\ else\ x \cdot i) *_R i) \in S$
by (*rule mem-is-intervalI*[*OF assms*(*1*,*2*)]) (*auto simp*: *assms*)

lemma *mem-box-componentwiseI*:
fixes *S*::'*a*::*euclidean-space set*
assumes *is-interval S*
assumes $\bigwedge i.\ i \in Basis \Longrightarrow x \cdot i \in ((\lambda x.\ x \cdot i)\ `\ S)$
shows $x \in S$
proof −
from *assms* **have** $\forall i \in Basis.\ \exists s \in S.\ x \cdot i = s \cdot i$
by *auto*
with *finite-Basis* **obtain** *s* **and** *bs*::'*a list* **where**
s: $\bigwedge i.\ i \in Basis \Longrightarrow x \cdot i = s\ i \cdot i$ $\bigwedge i.\ i \in Basis \Longrightarrow s\ i \in S$ **and**
bs: *set bs* = *Basis distinct bs*
by (*metis finite-distinct-list*)
from *nonempty-Basis s* **obtain** *j* **where** *j*: $j \in Basis$ $s\ j \in S$ **by** *blast*
def $y \equiv$ *rec-list*
(*s j*)
(λj - *Y*. $(\sum i{\in}Basis.\ (if\ i = j\ then\ s\ i \cdot i\ else\ Y \cdot i) *_R i)$)
have $x = (\sum i{\in}Basis.\ (if\ i \in set\ bs\ then\ s\ i \cdot i\ else\ s\ j \cdot i) *_R i)$
using *bs* **by** (*auto simp add*: *s*(*1*)[*symmetric*] *euclidean-representation*)
also have [*symmetric*]: *y bs* = ...
using *bs*(*2*) *bs*(*1*)[*THEN equalityD1*]
by (*induct bs*) (*auto simp*: *y-def euclidean-representation intro*!: *euclidean-eqI*[**where** '*a*='*a*])
also have *y bs* ∈ *S*
using *bs*(*1*)[*THEN equalityD1*]
apply (*induct bs*)
apply (*auto simp*: *y-def j*)
apply (*rule interval-subst*[*OF assms*(*1*)])
apply (*auto simp*: *s*)
done
finally show *?thesis* .
qed

16.7 Connectedness

lemma *connected-local*:
connected S ⟷
¬ (∃ *e1 e2*.
openin (*subtopology euclidean S*) *e1* ∧
openin (*subtopology euclidean S*) *e2* ∧
S ⊆ *e1* ∪ *e2* ∧
e1 ∩ *e2* = {} ∧
e1 ≠ {} ∧
e2 ≠ {})
unfolding *connected-def openin-open*
by *safe blast+*

lemma *exists-diff*:
fixes *P* :: *'a set* ⇒ *bool*
shows (∃ *S*. *P* (− *S*)) ⟷ (∃ *S*. *P S*) (**is** *?lhs* ⟷ *?rhs*)
proof −
{
assume *?lhs*
then have *?rhs* **by** *blast*
}
moreover
{
fix *S*
assume *H*: *P S*
have *S* = − (− *S*) **by** *auto*
with *H* **have** *P* (− (− *S*)) **by** *metis*
}
ultimately show *?thesis* **by** *metis*
qed

lemma *connected-clopen*: *connected S* ⟷
(∀ *T*. *openin* (*subtopology euclidean S*) *T* ∧
closedin (*subtopology euclidean S*) *T* ⟶ *T* = {} ∨ *T* = *S*) (**is** *?lhs* ⟷ *?rhs*)
proof −
have ¬ *connected S* ⟷
(∃ *e1 e2*. *open e1* ∧ *open* (− *e2*) ∧ *S* ⊆ *e1* ∪ (− *e2*) ∧ *e1* ∩ (− *e2*) ∩ *S* = {}
∧ *e1* ∩ *S* ≠ {} ∧ (− *e2*) ∩ *S* ≠ {})
unfolding *connected-def openin-open closedin-closed*
by (*metis double-complement*)
then have *th0*: *connected S* ⟷
¬ (∃ *e2 e1*. *closed e2* ∧ *open e1* ∧ *S* ⊆ *e1* ∪ (− *e2*) ∧ *e1* ∩ (− *e2*) ∩ *S* = {}
∧ *e1* ∩ *S* ≠ {} ∧ (− *e2*) ∩ *S* ≠ {})
(**is** - ⟷ ¬ (∃ *e2 e1*. *?P e2 e1*))
apply (*simp add*: *closed-def*)
apply *metis*
done
have *th1*: *?rhs* ⟷ ¬ (∃ *t' t*. *closed t'*∧*t* = *S*∩*t'* ∧ *t*≠{} ∧ *t*≠*S* ∧ (∃ *t'*. *open t'*
∧ *t* = *S* ∩ *t'*))

(**is** - $\longleftrightarrow \neg$ ($\exists\, t'\ t.$ *?Q* t' *t*))
unfolding *connected-def openin-open closedin-closed* **by** *auto*
{
fix *e2*
{
fix *e1*
have *?P e2 e1* $\longleftrightarrow$ ($\exists\, t.$ *closed e2* $\wedge$ *t* = *S*$\cap$*e2* $\wedge$ *open e1* $\wedge$ *t* = *S*$\cap$*e1* $\wedge$ *t*$\neq${} $\wedge$ *t* $\neq$ *S*)
by *auto*
}
then have ($\exists$ *e1*. *?P e2 e1*) $\longleftrightarrow$ ($\exists\, t.$ *?Q e2 t*)
by *metis*
}
then have $\forall$ *e2*. ($\exists$ *e1*. *?P e2 e1*) $\longleftrightarrow$ ($\exists\, t.$ *?Q e2 t*)
by *blast*
then show *?thesis*
unfolding *th0 th1* **by** *simp*
qed

16.8 Limit points

definition (**in** *topological-space*) *islimpt*:: *'a* $\Rightarrow$ *'a set* $\Rightarrow$ *bool* (**infixr** *islimpt 60*)
where *x islimpt S* $\longleftrightarrow$ ($\forall\, T.$ *x*$\in$*T* $\longrightarrow$ *open T* $\longrightarrow$ ($\exists\, y{\in}S.$ *y*$\in$*T* $\wedge$ *y*$\neq$*x*))

lemma *islimptI*:
assumes $\bigwedge T.$ *x* $\in$ *T* $\Longrightarrow$ *open T* $\Longrightarrow$ $\exists\, y{\in}S.$ *y* $\in$ *T* $\wedge$ *y* $\neq$ *x*
shows *x islimpt S*
using *assms* **unfolding** *islimpt-def* **by** *auto*

lemma *islimptE*:
assumes *x islimpt S* **and** *x* $\in$ *T* **and** *open T*
obtains *y* **where** *y* $\in$ *S* **and** *y* $\in$ *T* **and** *y* $\neq$ *x*
using *assms* **unfolding** *islimpt-def* **by** *auto*

lemma *islimpt-iff-eventually*: *x islimpt S* $\longleftrightarrow \neg$ *eventually* ($\lambda y.$ *y* $\notin$ *S*) (*at x*)
unfolding *islimpt-def eventually-at-topological* **by** *auto*

lemma *islimpt-subset*: *x islimpt S* $\Longrightarrow$ *S* $\subseteq$ *T* $\Longrightarrow$ *x islimpt T*
unfolding *islimpt-def* **by** *fast*

lemma *islimpt-approachable*:
fixes *x* :: *'a*::*metric-space*
shows *x islimpt S* $\longleftrightarrow$ ($\forall$ *e*>*0*. $\exists\, x'{\in}S.$ *x'* $\neq$ *x* $\wedge$ *dist x' x* < *e*)
unfolding *islimpt-iff-eventually eventually-at* **by** *fast*

lemma *islimpt-approachable-le*:
fixes *x* :: *'a*::*metric-space*
shows *x islimpt S* $\longleftrightarrow$ ($\forall$ *e*>*0*. $\exists\, x'{\in}$ *S*. *x'* $\neq$ *x* $\wedge$ *dist x' x* $\leq$ *e*)
unfolding *islimpt-approachable*

using *approachable-lt-le* [**where** $f{=}\lambda y.\ dist\ y\ x$ **and** $P{=}\lambda y.\ y \notin S \vee y = x$,
THEN arg-cong [**where** $f{=}Not$]]
by (*simp add*: *Bex-def conj-commute conj-left-commute*)

lemma *islimpt-UNIV-iff*: $x\ islimpt\ UNIV \longleftrightarrow \neg\ open\ \{x\}$
unfolding *islimpt-def* **by** (*safe*, *fast*, *case-tac* $T = \{x\}$, *fast*, *fast*)

lemma *islimpt-punctured*: $x\ islimpt\ S = x\ islimpt\ (S{-}\{x\})$
unfolding *islimpt-def* **by** *blast*

A perfect space has no isolated points.

lemma *islimpt-UNIV* [*simp*, *intro*]: ($x{::}'a{::}$*perfect-space*) *islimpt UNIV*
unfolding *islimpt-UNIV-iff* **by** (*rule not-open-singleton*)

lemma *perfect-choose-dist*:
fixes $x :: 'a{::}\{perfect\text{-}space, metric\text{-}space\}$
shows $0 < r \Longrightarrow \exists\, a.\ a \neq x \wedge dist\ a\ x < r$
using *islimpt-UNIV* [*of x*]
by (*simp add*: *islimpt-approachable*)

lemma *closed-limpt*: $closed\ S \longleftrightarrow (\forall\, x.\ x\ islimpt\ S \longrightarrow x \in S)$
unfolding *closed-def*
apply (*subst open-subopen*)
apply (*simp add*: *islimpt-def subset-eq*)
apply (*metis ComplE ComplI*)
done

lemma *islimpt-EMPTY*[*simp*]: $\neg\ x\ islimpt\ \{\}$
unfolding *islimpt-def* **by** *auto*

lemma *finite-set-avoid*:
fixes $a :: 'a{::}$*metric-space*
assumes *fS*: *finite S*
shows $\exists\, d{>}0.\ \forall\, x{\in}S.\ x \neq a \longrightarrow d \leq dist\ a\ x$
proof (*induct rule*: *finite-induct*[*OF fS*])
case *1*
then show *?case* **by** (*auto intro*: *zero-less-one*)
next
case (*2 x F*)
from *2* **obtain** *d* **where** *d*: $d > 0\ \forall\, x{\in}F.\ x \neq a \longrightarrow d \leq dist\ a\ x$
by *blast*
show *?case*
proof (*cases* $x = a$)
case *True*
then show *?thesis* **using** *d* **by** *auto*
next
case *False*
let *?d* = *min d* (*dist a x*)
have *dp*: *?d* > *0*

```
      using False d(1) by auto
    from d have d': ∀x∈F. x ≠ a ⟶ ?d ≤ dist a x
      by auto
    with dp False show ?thesis
      by (auto intro!: exI[where x=?d])
  qed
qed

lemma islimpt-Un: x islimpt (S ∪ T) ⟷ x islimpt S ∨ x islimpt T
  by (simp add: islimpt-iff-eventually eventually-conj-iff)

lemma discrete-imp-closed:
  fixes S :: 'a::metric-space set
  assumes e: 0 < e
    and d: ∀x ∈ S. ∀y ∈ S. dist y x < e ⟶ y = x
  shows closed S
proof −
  {
    fix x
    assume C: ∀e>0. ∃x'∈S. x' ≠ x ∧ dist x' x < e
    from e have e2: e/2 > 0 by arith
    from C[rule-format, OF e2] obtain y where y: y ∈ S y ≠ x dist y x < e/2
      by blast
    let ?m = min (e/2) (dist x y)
    from e2 y(2) have mp: ?m > 0
      by simp
    from C[rule-format, OF mp] obtain z where z: z ∈ S z ≠ x dist z x < ?m
      by blast
    have th: dist z y < e using z y
      by (intro dist-triangle-lt [where z=x], simp)
    from d[rule-format, OF y(1) z(1) th] y z
    have False by (auto simp add: dist-commute)}
  then show ?thesis
    by (metis islimpt-approachable closed-limpt [where 'a='a])
qed

lemma closed-of-nat-image: closed (of-nat ‘ A :: 'a :: real-normed-algebra-1 set)
  by (rule discrete-imp-closed[of 1]) (auto simp: dist-of-nat)

lemma closed-of-int-image: closed (of-int ‘ A :: 'a :: real-normed-algebra-1 set)
  by (rule discrete-imp-closed[of 1]) (auto simp: dist-of-int)

lemma closed-Nats [simp]: closed (ℕ :: 'a :: real-normed-algebra-1 set)
  unfolding Nats-def by (rule closed-of-nat-image)

lemma closed-Ints [simp]: closed (ℤ :: 'a :: real-normed-algebra-1 set)
  unfolding Ints-def by (rule closed-of-int-image)
```

16.9 Interior of a Set

definition *interior* $S = \bigcup \{T.\ \mathit{open}\ T \wedge T \subseteq S\}$

lemma *interiorI* [*intro?*]:
assumes *open* T **and** $x \in T$ **and** $T \subseteq S$
shows $x \in \mathit{interior}\ S$
using *assms* **unfolding** *interior-def* **by** *fast*

lemma *interiorE* [*elim?*]:
assumes $x \in \mathit{interior}\ S$
obtains T **where** *open* T **and** $x \in T$ **and** $T \subseteq S$
using *assms* **unfolding** *interior-def* **by** *fast*

lemma *open-interior* [*simp*, *intro*]: *open* (*interior* S)
by (*simp add*: *interior-def open-Union*)

lemma *interior-subset*: $\mathit{interior}\ S \subseteq S$
by (*auto simp add*: *interior-def*)

lemma *interior-maximal*: $T \subseteq S \Longrightarrow \mathit{open}\ T \Longrightarrow T \subseteq \mathit{interior}\ S$
by (*auto simp add*: *interior-def*)

lemma *interior-open*: $\mathit{open}\ S \Longrightarrow \mathit{interior}\ S = S$
by (*intro equalityI interior-subset interior-maximal subset-refl*)

lemma *interior-eq*: $\mathit{interior}\ S = S \longleftrightarrow \mathit{open}\ S$
by (*metis open-interior interior-open*)

lemma *open-subset-interior*: $\mathit{open}\ S \Longrightarrow S \subseteq \mathit{interior}\ T \longleftrightarrow S \subseteq T$
by (*metis interior-maximal interior-subset subset-trans*)

lemma *interior-empty* [*simp*]: $\mathit{interior}\ \{\} = \{\}$
using *open-empty* **by** (*rule interior-open*)

lemma *interior-UNIV* [*simp*]: $\mathit{interior}\ \mathit{UNIV} = \mathit{UNIV}$
using *open-UNIV* **by** (*rule interior-open*)

lemma *interior-interior* [*simp*]: $\mathit{interior}\ (\mathit{interior}\ S) = \mathit{interior}\ S$
using *open-interior* **by** (*rule interior-open*)

lemma *interior-mono*: $S \subseteq T \Longrightarrow \mathit{interior}\ S \subseteq \mathit{interior}\ T$
by (*auto simp add*: *interior-def*)

lemma *interior-unique*:
assumes $T \subseteq S$ **and** *open* T
assumes $\bigwedge T'.\ T' \subseteq S \Longrightarrow \mathit{open}\ T' \Longrightarrow T' \subseteq T$
shows $\mathit{interior}\ S = T$
by (*intro equalityI assms interior-subset open-interior interior-maximal*)

lemma *interior-singleton* [*simp*]:
  fixes *a* :: *'a*::*perfect-space* **shows** *interior* {*a*} = {}
 apply (*rule interior-unique*, *simp-all*)
 using *not-open-singleton subset-singletonD* **by** *fastforce*

lemma *interior-Int* [*simp*]: *interior* (*S* ∩ *T*) = *interior S* ∩ *interior T*
 by (*intro equalityI Int-mono Int-greatest interior-mono Int-lower1*
  Int-lower2 interior-maximal interior-subset open-Int open-interior)

lemma *mem-interior*: *x* ∈ *interior S* ⟷ (∃ *e*>*0*. *ball x e* ⊆ *S*)
 using *open-contains-ball-eq* [**where** *S*=*interior S*]
 by (*simp add*: *open-subset-interior*)

lemma *eventually-nhds-in-nhd*: *x* ∈ *interior s* ⟹ *eventually* (λ*y*. *y* ∈ *s*) (*nhds x*)
 using *interior-subset*[*of s*] **by** (*subst eventually-nhds*) *blast*

lemma *interior-limit-point* [*intro*]:
 fixes *x* :: *'a*::*perfect-space*
 assumes *x*: *x* ∈ *interior S*
 shows *x islimpt S*
 using *x islimpt-UNIV* [*of x*]
 unfolding *interior-def islimpt-def*
 apply (*clarsimp*, *rename-tac T T'*)
 apply (*drule-tac x*=*T* ∩ *T'* **in** *spec*)
 apply (*auto simp add*: *open-Int*)
 done

lemma *interior-closed-Un-empty-interior*:
 assumes *cS*: *closed S*
  and *iT*: *interior T* = {}
 shows *interior* (*S* ∪ *T*) = *interior S*
proof
 show *interior S* ⊆ *interior* (*S* ∪ *T*)
  by (*rule interior-mono*) (*rule Un-upper1*)
 show *interior* (*S* ∪ *T*) ⊆ *interior S*
 proof
  fix *x*
  assume *x* ∈ *interior* (*S* ∪ *T*)
  then obtain *R* **where** *open R x* ∈ *R R* ⊆ *S* ∪ *T* **..**
  show *x* ∈ *interior S*
  proof (*rule ccontr*)
   assume *x* ∉ *interior S*
   with ‹*x* ∈ *R*› ‹*open R*› **obtain** *y* **where** *y* ∈ *R* − *S*
    unfolding *interior-def* **by** *fast*
   from ‹*open R*› ‹*closed S*› **have** *open* (*R* − *S*)
    by (*rule open-Diff*)
   from ‹*R* ⊆ *S* ∪ *T*› **have** *R* − *S* ⊆ *T*
    by *fast*

```
    from ‹y ∈ R − S› ‹open (R − S)› ‹R − S ⊆ T› ‹interior T = {}› show False
      unfolding interior-def by fast
   qed
  qed
qed

lemma interior-Times: interior (A × B) = interior A × interior B
proof (rule interior-unique)
  show interior A × interior B ⊆ A × B
    by (intro Sigma-mono interior-subset)
  show open (interior A × interior B)
    by (intro open-Times open-interior)
  fix T
  assume T ⊆ A × B and open T
  then show T ⊆ interior A × interior B
  proof safe
    fix x y
    assume (x, y) ∈ T
    then obtain C D where open C open D C × D ⊆ T x ∈ C y ∈ D
      using ‹open T› unfolding open-prod-def by fast
    then have open C open D C ⊆ A D ⊆ B x ∈ C y ∈ D
      using ‹T ⊆ A × B› by auto
    then show x ∈ interior A and y ∈ interior B
      by (auto intro: interiorI)
  qed
qed

lemma interior-Ici:
  fixes x :: 'a :: {dense-linorder, linorder-topology}
  assumes b < x
  shows interior { x ..} = { x <..}
proof (rule interior-unique)
  fix T assume T ⊆ {x ..} open T
  moreover have x ∉ T
  proof
    assume x ∈ T
    obtain y where y < x {y <.. x} ⊆ T
      using open-left[OF ‹open T› ‹x ∈ T› ‹b < x›] by auto
    with dense[OF ‹y < x›] obtain z where z ∈ T z < x
      by (auto simp: subset-eq Ball-def)
    with ‹T ⊆ {x ..}› show False by auto
  qed
  ultimately show T ⊆ {x <..}
    by (auto simp: subset-eq less-le)
qed auto

lemma interior-Iic:
  fixes x :: 'a :: {dense-linorder, linorder-topology}
  assumes x < b
```

shows *interior* $\{.. x\} = \{..< x\}$
proof (*rule interior-unique*)
fix T **assume** $T \subseteq \{.. x\}$ *open* T
moreover have $x \notin T$
proof
assume $x \in T$
obtain y **where** $x < y$ $\{x ..< y\} \subseteq T$
using *open-right*[*OF* ‹*open* T› ‹$x \in T$› ‹$x < b$›] **by** *auto*
with *dense*[*OF* ‹$x < y$›] **obtain** z **where** $z \in T$ $x < z$
by (*auto simp*: *subset-eq Ball-def less-le*)
with ‹$T \subseteq \{.. x\}$› **show** *False* **by** *auto*
qed
ultimately show $T \subseteq \{..< x\}$
by (*auto simp*: *subset-eq less-le*)
qed *auto*

16.10 Closure of a Set

definition *closure* $S = S \cup \{x \mid x.\ x$ *islimpt* $S\}$

lemma *interior-closure*: *interior* $S = -$ (*closure* $(-S)$)
unfolding *interior-def closure-def islimpt-def* **by** *auto*

lemma *closure-interior*: *closure* $S = -$ *interior* $(-S)$
unfolding *interior-closure* **by** *simp*

lemma *closed-closure*[*simp*, *intro*]: *closed* (*closure* S)
unfolding *closure-interior* **by** (*simp add*: *closed-Compl*)

lemma *closure-subset*: $S \subseteq$ *closure* S
unfolding *closure-def* **by** *simp*

lemma *closure-hull*: *closure* $S =$ *closed hull* S
unfolding *hull-def closure-interior interior-def* **by** *auto*

lemma *closure-eq*: *closure* $S = S \longleftrightarrow$ *closed* S
unfolding *closure-hull* **using** *closed-Inter* **by** (*rule hull-eq*)

lemma *closure-closed* [*simp*]: *closed* $S \Longrightarrow$ *closure* $S = S$
unfolding *closure-eq* **.**

lemma *closure-closure* [*simp*]: *closure* (*closure* S) = *closure* S
unfolding *closure-hull* **by** (*rule hull-hull*)

lemma *closure-mono*: $S \subseteq T \Longrightarrow$ *closure* $S \subseteq$ *closure* T
unfolding *closure-hull* **by** (*rule hull-mono*)

lemma *closure-minimal*: $S \subseteq T \Longrightarrow$ *closed* $T \Longrightarrow$ *closure* $S \subseteq T$
unfolding *closure-hull* **by** (*rule hull-minimal*)

lemma *closure-unique*:
  assumes $S \subseteq T$
    and *closed* T
    and $\bigwedge T'.\ S \subseteq T' \Longrightarrow$ *closed* $T' \Longrightarrow T \subseteq T'$
  shows *closure* $S = T$
  using *assms* **unfolding** *closure-hull* **by** (*rule hull-unique*)

lemma *closure-empty* [*simp*]: *closure* $\{\} = \{\}$
  using *closed-empty* **by** (*rule closure-closed*)

lemma *closure-UNIV* [*simp*]: *closure UNIV* $=$ *UNIV*
  using *closed-UNIV* **by** (*rule closure-closed*)

lemma *closure-union* [*simp*]: *closure* $(S \cup T) =$ *closure* $S \cup$ *closure* T
  unfolding *closure-interior* **by** *simp*

lemma *closure-eq-empty* [*iff*]: *closure* $S = \{\} \longleftrightarrow S = \{\}$
  using *closure-empty closure-subset*[*of* S]
  by *blast*

lemma *closure-subset-eq*: *closure* $S \subseteq S \longleftrightarrow$ *closed* S
  using *closure-eq*[*of* S] *closure-subset*[*of* S]
  by *simp*

lemma *open-inter-closure-eq-empty*:
  open $S \Longrightarrow (S \cap$ *closure* $T) = \{\} \longleftrightarrow S \cap T = \{\}$
  using *open-subset-interior*[*of* $S - T$]
  using *interior-subset*[*of* $- T$]
  unfolding *closure-interior*
  by *auto*

lemma *open-inter-closure-subset*:
  open $S \Longrightarrow (S \cap (closure\ T)) \subseteq closure(S \cap T)$
proof
  fix x
  assume *as*: *open* S $x \in S \cap$ *closure* T
  {
    assume $*$: x *islimpt* T
    have x *islimpt* $(S \cap T)$
    proof (*rule islimptI*)
      fix A
      assume $x \in A$ *open* A
      with *as* **have** $x \in A \cap S$ *open* $(A \cap S)$
        by (*simp-all add*: *open-Int*)
      with $*$ **obtain** y **where** $y \in T$ $y \in A \cap S$ $y \neq x$
        by (*rule islimptE*)
      then have $y \in S \cap T$ $y \in A \wedge y \neq x$
        by *simp-all*

```
      then show ∃y∈(S ∩ T). y ∈ A ∧ y ≠ x ..
    qed
  }
  then show x ∈ closure (S ∩ T) using as
    unfolding closure-def
    by blast
qed

lemma closure-complement: closure (− S) = − interior S
  unfolding closure-interior by simp

lemma interior-complement: interior (− S) = − closure S
  unfolding closure-interior by simp

lemma closure-Times: closure (A × B) = closure A × closure B
proof (rule closure-unique)
  show A × B ⊆ closure A × closure B
    by (intro Sigma-mono closure-subset)
  show closed (closure A × closure B)
    by (intro closed-Times closed-closure)
  fix T
  assume A × B ⊆ T and closed T
  then show closure A × closure B ⊆ T
    apply (simp add: closed-def open-prod-def, clarify)
    apply (rule ccontr)
    apply (drule-tac x=(a, b) in bspec, simp, clarify, rename-tac C D)
    apply (simp add: closure-interior interior-def)
    apply (drule-tac x=C in spec)
    apply (drule-tac x=D in spec)
    apply auto
    done
qed

lemma islimpt-in-closure: (x islimpt S) = (x:closure(S−{x}))
  unfolding closure-def using islimpt-punctured by blast

lemma connected-imp-connected-closure: connected s ⟹ connected (closure s)
   by (rule connectedI) (meson closure-subset open-Int open-inter-closure-eq-empty
subset-trans connectedD)

lemma limpt-of-limpts:
     fixes x :: 'a::metric-space
     shows x islimpt {y. y islimpt s} ⟹ x islimpt s
  apply (clarsimp simp add: islimpt-approachable)
  apply (drule-tac x=e/2 in spec)
  apply (auto simp: simp del: less-divide-eq-numeral1)
  apply (drule-tac x=dist x' x in spec)
  apply (auto simp: zero-less-dist-iff simp del: less-divide-eq-numeral1)
  apply (erule rev-bexI)
```

by (*metis dist-commute dist-triangle-half-r less-trans less-irrefl*)

lemma *closed-limpts*: *closed* {*x*::*'a*::*metric-space*. *x islimpt s*}
using *closed-limpt limpt-of-limpts* **by** *blast*

lemma *limpt-of-closure*:
fixes *x* :: *'a*::*metric-space*
shows *x islimpt closure s* ⟷ *x islimpt s*
by (*auto simp*: *closure-def islimpt-Un dest*: *limpt-of-limpts*)

lemma *closed-in-limpt*:
closedin (*subtopology euclidean t*) *s* ⟷ *s* ⊆ *t* ∧ (∀ *x*. *x islimpt s* ∧ *x* ∈ *t* ⟶ *x* ∈ *s*)
apply (*simp add*: *closedin-closed*, *safe*)
apply (*simp add*: *closed-limpt islimpt-subset*)
apply (*rule-tac x*=*closure s* **in** *exI*)
apply *simp*
apply (*force simp*: *closure-def*)
done

lemma *closedin-closed-eq*:
closed s ⟹ (*closedin* (*subtopology euclidean s*) *t* ⟷ *closed t* ∧ *t* ⊆ *s*)
by (*meson closed-in-limpt closed-subset closedin-closed-trans*)

lemma *bdd-below-closure*:
fixes *A* :: *real set*
assumes *bdd-below A*
shows *bdd-below* (*closure A*)
proof −
from *assms* **obtain** *m* **where** ⋀*x*. *x* ∈ *A* ⟹ *m* ≤ *x* **unfolding** *bdd-below-def*
by *auto*
hence *A* ⊆ {*m*..} **by** *auto*
hence *closure A* ⊆ {*m*..} **using** *closed-real-atLeast* **by** (*rule closure-minimal*)
thus *?thesis* **unfolding** *bdd-below-def* **by** *auto*
qed

16.11 Connected components, considered as a connectedness relation or a set

definition
connected-component s x y ≡ ∃ *t*. *connected t* ∧ *t* ⊆ *s* ∧ *x* ∈ *t* ∧ *y* ∈ *t*

abbreviation
connected-component-set s x ≡ *Collect* (*connected-component s x*)

lemma *connected-componentI*:
⟦*connected t*; *t* ⊆ *s*; *x* ∈ *t*; *y* ∈ *t*⟧ ⟹ *connected-component s x y*
by (*auto simp*: *connected-component-def*)

lemma *connected-component-in*: *connected-component s x y* $\Longrightarrow$ $x \in s \wedge y \in s$
by (*auto simp*: *connected-component-def*)

lemma *connected-component-refl*: $x \in s$ $\Longrightarrow$ *connected-component s x x*
apply (*auto simp*: *connected-component-def*)
using *connected-sing* **by** *blast*

lemma *connected-component-refl-eq* [*simp*]: *connected-component s x x* $\longleftrightarrow$ $x \in s$
by (*auto simp*: *connected-component-refl*) (*auto simp*: *connected-component-def*)

lemma *connected-component-sym*: *connected-component s x y* $\Longrightarrow$ *connected-component s y x*
by (*auto simp*: *connected-component-def*)

lemma *connected-component-trans*:
$[\![$*connected-component s x y*; *connected-component s y z*$]\!]$ $\Longrightarrow$ *connected-component s x z*
unfolding *connected-component-def*
by (*metis Int-iff Un-iff Un-subset-iff equals0D connected-Un*)

lemma *connected-component-of-subset*: $[\![$*connected-component s x y*; $s \subseteq t]\!]$ $\Longrightarrow$ *connected-component t x y*
by (*auto simp*: *connected-component-def*)

lemma *connected-component-Union*: *connected-component-set s x* $= \bigcup \{t.$ *connected t* $\wedge$ $x \in t \wedge t \subseteq s\}$
by (*auto simp*: *connected-component-def*)

lemma *connected-connected-component* [*iff*]: *connected* (*connected-component-set s x*)
by (*auto simp*: *connected-component-Union intro*: *connected-Union*)

lemma *connected-iff-eq-connected-component-set*: *connected s* $\longleftrightarrow$ $(\forall x \in s.$ *connected-component-set s x* $= s)$
proof (*cases s*={})
case *True* **then show** *?thesis* **by** *simp*
next
case *False*
then obtain *x* **where** $x \in s$ **by** *auto*
show *?thesis*
proof
assume *connected s*
then show $\forall x \in s.$ *connected-component-set s x* $= s$
by (*force simp*: *connected-component-def*)
next
assume $\forall x \in s.$ *connected-component-set s x* $= s$
then show *connected s*
by (*metis* ‹$x \in s$› *connected-connected-component*)
qed

qed

lemma *connected-component-subset*: *connected-component-set s x* ⊆ *s*
using *connected-component-in* **by** *blast*

lemma *connected-component-eq-self*: ⟦*connected s*; *x* ∈ *s*⟧ ⟹ *connected-component-set s x* = *s*
by (*simp add*: *connected-iff-eq-connected-component-set*)

lemma *connected-iff-connected-component*:
connected s ⟷ (∀ *x* ∈ *s*. ∀ *y* ∈ *s*. *connected-component s x y*)
using *connected-component-in* **by** (*auto simp*: *connected-iff-eq-connected-component-set*)

lemma *connected-component-maximal*:
⟦*x* ∈ *t*; *connected t*; *t* ⊆ *s*⟧ ⟹ *t* ⊆ (*connected-component-set s x*)
using *connected-component-eq-self connected-component-of-subset* **by** *blast*

lemma *connected-component-mono*:
s ⊆ *t* ⟹ (*connected-component-set s x*) ⊆ (*connected-component-set t x*)
by (*simp add*: *Collect-mono connected-component-of-subset*)

lemma *connected-component-eq-empty* [*simp*]: *connected-component-set s x* = {}
⟷ (*x* ∉ *s*)
using *connected-component-refl* **by** (*fastforce simp*: *connected-component-in*)

lemma *connected-component-set-empty* [*simp*]: *connected-component-set* {} *x* =
{}
using *connected-component-eq-empty* **by** *blast*

lemma *connected-component-eq*:
y ∈ *connected-component-set s x*
⟹ (*connected-component-set s y* = *connected-component-set s x*)
by (*metis* (*no-types*, *lifting*) *Collect-cong connected-component-sym connected-component-trans mem-Collect-eq*)

lemma *closed-connected-component*:
assumes *s*: *closed s* **shows** *closed* (*connected-component-set s x*)
proof (*cases x* ∈ *s*)
case *False* **then show** *?thesis*
by (*metis connected-component-eq-empty closed-empty*)
next
case *True*
show *?thesis*
unfolding *closure-eq* [*symmetric*]
proof
show *closure* (*connected-component-set s x*) ⊆ *connected-component-set s x*
apply (*rule connected-component-maximal*)
apply (*simp add*: *closure-def True*)
apply (*simp add*: *connected-imp-connected-closure*)

apply (*simp add*: *s closure-minimal connected-component-subset*)
done
next
show *connected-component-set s x* ⊆ *closure* (*connected-component-set s x*)
by (*simp add*: *closure-subset*)
qed
qed

lemma *connected-component-disjoint*:
(*connected-component-set s a*) ∩ (*connected-component-set s b*) = {} ⟷
a ∉ *connected-component-set s b*
apply (*auto simp*: *connected-component-eq*)
using *connected-component-eq connected-component-sym* **by** *blast*

lemma *connected-component-nonoverlap*:
(*connected-component-set s a*) ∩ (*connected-component-set s b*) = {} ⟷
(*a* ∉ *s* ∨ *b* ∉ *s* ∨ *connected-component-set s a* ≠ *connected-component-set s b*)
apply (*auto simp*: *connected-component-in*)
using *connected-component-refl-eq* **apply** *blast*
apply (*metis connected-component-eq mem-Collect-eq*)
apply (*metis connected-component-eq mem-Collect-eq*)
done

lemma *connected-component-overlap*:
(*connected-component-set s a* ∩ *connected-component-set s b* ≠ {}) =
(*a* ∈ *s* ∧ *b* ∈ *s* ∧ *connected-component-set s a* = *connected-component-set s b*)
by (*auto simp*: *connected-component-nonoverlap*)

lemma *connected-component-sym-eq*: *connected-component s x y* ⟷ *connected-component s y x*
using *connected-component-sym* **by** *blast*

lemma *connected-component-eq-eq*:
connected-component-set s x = *connected-component-set s y* ⟷
x ∉ *s* ∧ *y* ∉ *s* ∨ *x* ∈ *s* ∧ *y* ∈ *s* ∧ *connected-component s x y*
apply (*case-tac y* ∈ *s*)
apply (*simp add*:)
apply (*metis connected-component-eq connected-component-eq-empty connected-component-refl-eq mem-Collect-eq*)
apply (*case-tac x* ∈ *s*)
apply (*simp add*:)
apply (*metis connected-component-eq-empty*)
using *connected-component-eq-empty* **by** *blast*

lemma *connected-iff-connected-component-eq*:
connected s ⟷
(∀ *x* ∈ *s*. ∀ *y* ∈ *s*. *connected-component-set s x* = *connected-component-set s y*)
by (*simp add*: *connected-component-eq-eq connected-iff-connected-component*)

lemma *connected-component-idemp*:
connected-component-set (*connected-component-set s x*) *x* = *connected-component-set s x*
apply (*rule subset-antisym*)
apply (*simp add*: *connected-component-subset*)
by (*metis connected-component-eq-empty connected-component-maximal connected-component-refl-eq connected-connected-component mem-Collect-eq set-eq-subset*)

lemma *connected-component-unique*:
⟦*x* ∈ *c*; *c* ⊆ *s*; *connected c*;
⋀*c'*. *x* ∈ *c'* ∧ *c'* ⊆ *s* ∧ *connected c'*
⟹ *c'* ⊆ *c*⟧
⟹ *connected-component-set s x* = *c*
apply (*rule subset-antisym*)
apply (*meson connected-component-maximal connected-component-subset connected-connected-component contra-subsetD*)
by (*simp add*: *connected-component-maximal*)

lemma *joinable-connected-component-eq*:
⟦*connected t*; *t* ⊆ *s*;
connected-component-set s x ∩ *t* ≠ {};
connected-component-set s y ∩ *t* ≠ {}⟧
⟹ *connected-component-set s x* = *connected-component-set s y*
apply (*simp add*: *ex-in-conv* [*symmetric*])
apply (*rule connected-component-eq*)
by (*metis* (*no-types*, *hide-lams*) *connected-component-eq-eq connected-component-in connected-component-maximal subsetD mem-Collect-eq*)

lemma *Union-connected-component*: ⋃(*connected-component-set s* ' *s*) = *s*
apply (*rule subset-antisym*)
apply (*simp add*: *SUP-least connected-component-subset*)
using *connected-component-refl-eq*
by *force*

lemma *complement-connected-component-unions*:
s − *connected-component-set s x* =
⋃(*connected-component-set s* ' *s* − {*connected-component-set s x*})
apply (*subst Union-connected-component* [*symmetric*], *auto*)
apply (*metis connected-component-eq-eq connected-component-in*)
by (*metis connected-component-eq mem-Collect-eq*)

lemma *connected-component-intermediate-subset*:
⟦*connected-component-set u a* ⊆ *t*; *t* ⊆ *u*⟧
⟹ *connected-component-set t a* = *connected-component-set u a*
apply (*case-tac a* ∈ *u*)
apply (*simp add*: *connected-component-maximal connected-component-mono subset-antisym*)

using *connected-component-eq-empty* **by** *blast*

16.12 The set of connected components of a set

definition *components*:: *'a*::*topological-space set* ⇒ *'a set set* **where**
components s ≡ *connected-component-set s ' s*

lemma *components-iff*: *s* ∈ *components u* ⟷ (∃ *x*. *x* ∈ *u* ∧ *s* = *connected-component-set u x*)
by (*auto simp*: *components-def*)

lemma *Union-components*: *u* = ⋃(*components u*)
apply (*rule subset-antisym*)
apply (*metis Union-connected-component components-def set-eq-subset*)
using *Union-connected-component components-def* **by** *fastforce*

lemma *pairwise-disjoint-components*: *pairwise* (λ*X Y*. *X* ∩ *Y* = {}) (*components u*)
apply (*simp add*: *pairwise-def*)
apply (*auto simp*: *components-iff*)
apply (*metis connected-component-eq-eq connected-component-in*)+
done

lemma *in-components-nonempty*: *c* ∈ *components s* ⟹ *c* ≠ {}
by (*metis components-iff connected-component-eq-empty*)

lemma *in-components-subset*: *c* ∈ *components s* ⟹ *c* ⊆ *s*
using *Union-components* **by** *blast*

lemma *in-components-connected*: *c* ∈ *components s* ⟹ *connected c*
by (*metis components-iff connected-connected-component*)

lemma *in-components-maximal*:
c ∈ *components s* ⟷
(*c* ≠ {} ∧ *c* ⊆ *s* ∧ *connected c* ∧ (∀ *d*. *d* ≠ {} ∧ *c* ⊆ *d* ∧ *d* ⊆ *s* ∧ *connected d* ⟶ *d* = *c*))
apply (*rule iffI*)
apply (*simp add*: *in-components-nonempty in-components-connected*)
apply (*metis* (*full-types*) *components-iff connected-component-eq-self connected-component-intermediate-subset connected-component-refl in-components-subset mem-Collect-eq rev-subsetD*)
by (*metis bot.extremum-uniqueI components-iff connected-component-eq-empty connected-component-maximal connected-component-subset connected-connected-component subset-emptyI*)

lemma *joinable-components-eq*:
connected t ∧ *t* ⊆ *s* ∧ *c1* ∈ *components s* ∧ *c2* ∈ *components s* ∧ *c1* ∩ *t* ≠ {} ∧ *c2* ∩ *t* ≠ {} ⟹ *c1* = *c2*
by (*metis* (*full-types*) *components-iff joinable-connected-component-eq*)

lemma *closed-components*: $[\![closed\ s;\ c \in components\ s]\!] \Longrightarrow closed\ c$
by (*metis closed-connected-component components-iff*)

lemma *components-nonoverlap*:
$[\![c \in components\ s;\ c' \in components\ s]\!] \Longrightarrow (c \cap c' = \{\}) \longleftrightarrow (c \neq c')$
apply (*auto simp*: *in-components-nonempty components-iff*)
using *connected-component-refl* **apply** *blast*
apply (*metis connected-component-eq-eq connected-component-in*)
by (*metis connected-component-eq mem-Collect-eq*)

lemma *components-eq*: $[\![c \in components\ s;\ c' \in components\ s]\!] \Longrightarrow (c = c' \longleftrightarrow c \cap c' \neq \{\})$
by (*metis components-nonoverlap*)

lemma *components-eq-empty* [*simp*]: $components\ s = \{\} \longleftrightarrow s = \{\}$
by (*simp add*: *components-def*)

lemma *components-empty* [*simp*]: $components\ \{\} = \{\}$
by *simp*

lemma *connected-eq-connected-components-eq*: $connected\ s \longleftrightarrow (\forall c \in components\ s.\ \forall c' \in components\ s.\ c = c')$
by (*metis* (*no-types*, *hide-lams*) *components-iff connected-component-eq-eq connected-iff-connected-component*

lemma *components-eq-sing-iff*: $components\ s = \{s\} \longleftrightarrow connected\ s \wedge s \neq \{\}$
apply (*rule iffI*)
using *in-components-connected* **apply** *fastforce*
apply *safe*
using *Union-components* **apply** *fastforce*
apply (*metis components-iff connected-component-eq-self*)
using *in-components-maximal* **by** *auto*

lemma *components-eq-sing-exists*: $(\exists a.\ components\ s = \{a\}) \longleftrightarrow connected\ s \wedge s \neq \{\}$
apply (*rule iffI*)
using *connected-eq-connected-components-eq* **apply** *fastforce*
by (*metis components-eq-sing-iff*)

lemma *connected-eq-components-subset-sing*: $connected\ s \longleftrightarrow components\ s \subseteq \{s\}$
by (*metis Union-components components-empty components-eq-sing-iff connected-empty insert-subset order-refl subset-singletonD*)

lemma *connected-eq-components-subset-sing-exists*: $connected\ s \longleftrightarrow (\exists a.\ components\ s \subseteq \{a\})$
by (*metis components-eq-sing-exists connected-eq-components-subset-sing empty-iff subset-iff subset-singletonD*)

lemma *in-components-self*: $s \in components\ s \longleftrightarrow connected\ s \wedge s \neq \{\}$
by (*metis components-empty components-eq-sing-iff empty-iff in-components-connected*

insertI1)

lemma *components-maximal*: ⟦*c* ∈ *components s*; *connected t*; *t* ⊆ *s*; *c* ∩ *t* ≠ {}⟧ ⟹ *t* ⊆ *c*
apply (*simp add*: *components-def ex-in-conv* [*symmetric*], *clarify*)
by (*meson connected-component-def connected-component-trans*)

lemma *exists-component-superset*: ⟦*t* ⊆ *s*; *s* ≠ {}; *connected t*⟧ ⟹ ∃*c*. *c* ∈ *components s* ∧ *t* ⊆ *c*
apply (*case-tac t* = {})
apply *force*
by (*metis components-def ex-in-conv connected-component-maximal contra-subsetD image-eqI*)

lemma *components-intermediate-subset*: ⟦*s* ∈ *components u*; *s* ⊆ *t*; *t* ⊆ *u*⟧ ⟹ *s* ∈ *components t*
apply (*auto simp*: *components-iff*)
by (*metis connected-component-eq-empty connected-component-intermediate-subset*)

lemma *in-components-unions-complement*: *c* ∈ *components s* ⟹ *s* − *c* = ⋃(*components s* − {*c*})
by (*metis complement-connected-component-unions components-def components-iff*)

lemma *connected-intermediate-closure*:
assumes *cs*: *connected s* **and** *st*: *s* ⊆ *t* **and** *ts*: *t* ⊆ *closure s*
shows *connected t*
proof (*rule connectedI*)
fix *A B*
assume *A*: *open A* **and** *B*: *open B* **and** *Alap*: *A* ∩ *t* ≠ {} **and** *Blap*: *B* ∩ *t* ≠ {}
and *disj*: *A* ∩ *B* ∩ *t* = {} **and** *cover*: *t* ⊆ *A* ∪ *B*
have *disjs*: *A* ∩ *B* ∩ *s* = {}
using *disj st* **by** *auto*
have *A* ∩ *closure s* ≠ {}
using *Alap Int-absorb1 ts* **by** *blast*
then have *Alaps*: *A* ∩ *s* ≠ {}
by (*simp add*: *A open-inter-closure-eq-empty*)
have *B* ∩ *closure s* ≠ {}
using *Blap Int-absorb1 ts* **by** *blast*
then have *Blaps*: *B* ∩ *s* ≠ {}
by (*simp add*: *B open-inter-closure-eq-empty*)
then show *False*
using *cs* [*unfolded connected-def*] *A B disjs Alaps Blaps cover st*
by *blast*
qed

lemma *closed-in-connected-component*: *closedin* (*subtopology euclidean s*) (*connected-component-set s x*)
proof (*cases connected-component-set s x* = {})

case *True* **then show** *?thesis*
by (*metis closedin-empty*)
next
case *False*
then obtain *y* **where** *y*: *connected-component s x y*
by *blast*
have *1*: *connected-component-set s x ⊆ s ∩ closure* (*connected-component-set s x*)
by (*auto simp*: *closure-def connected-component-in*)
have *2*: *connected-component s x y ⟹ s ∩ closure* (*connected-component-set s x*) *⊆ connected-component-set s x*
apply (*rule connected-component-maximal*)
apply (*simp add*:)
using *closure-subset connected-component-in* **apply** *fastforce*
using *1 connected-intermediate-closure* **apply** *blast+*
done
show *?thesis* **using** *y*
apply (*simp add*: *Topology-Euclidean-Space.closedin-closed*)
using *1 2* **by** *auto*
qed

16.13 Frontier (aka boundary)

definition *frontier S = closure S − interior S*

lemma *frontier-closed*: *closed* (*frontier S*)
by (*simp add*: *frontier-def closed-Diff*)

lemma *frontier-closures*: *frontier S* = (*closure S*) ∩ (*closure*(− *S*))
by (*auto simp add*: *frontier-def interior-closure*)

lemma *frontier-straddle*:
fixes *a* :: *'a*::*metric-space*
shows *a ∈ frontier S ⟷* (∀ *e*>*0*. (∃ *x*∈*S*. *dist a x* < *e*) ∧ (∃ *x*. *x ∉ S ∧ dist a x* < *e*))
unfolding *frontier-def closure-interior*
by (*auto simp add*: *mem-interior subset-eq ball-def*)

lemma *frontier-subset-closed*: *closed S ⟹ frontier S ⊆ S*
by (*metis frontier-def closure-closed Diff-subset*)

lemma *frontier-empty*[*simp*]: *frontier* {} = {}
by (*simp add*: *frontier-def*)

lemma *frontier-subset-eq*: *frontier S ⊆ S ⟷ closed S*
proof −
{
assume *frontier S ⊆ S*
then have *closure S ⊆ S*

using *interior-subset* **unfolding** *frontier-def* **by** *auto*
then have *closed S*
using *closure-subset-eq* **by** *auto*
}
then show *?thesis* **using** *frontier-subset-closed*[*of S*] **..**
qed

lemma *frontier-complement* [*simp*]: *frontier* (− *S*) = *frontier S*
by (*auto simp add*: *frontier-def closure-complement interior-complement*)

lemma *frontier-disjoint-eq*: *frontier S* ∩ *S* = {} ⟷ *open S*
using *frontier-complement frontier-subset-eq*[*of* − *S*]
unfolding *open-closed* **by** *auto*

16.14 Filters and the "eventually true" quantifier

definition *indirection* :: *'a*::*real-normed-vector* ⇒ *'a* ⇒ *'a filter*
(**infixr** *indirection 70*)
where *a indirection v* = *at a within* {*b*. ∃ *c*≥*0*. *b* − *a* = *scaleR c v*}

Identify Trivial limits, where we can't approach arbitrarily closely.

lemma *trivial-limit-within*: *trivial-limit* (*at a within S*) ⟷ ¬ *a islimpt S*
proof
assume *trivial-limit* (*at a within S*)
then show ¬ *a islimpt S*
unfolding *trivial-limit-def*
unfolding *eventually-at-topological*
unfolding *islimpt-def*
apply (*clarsimp simp add*: *set-eq-iff*)
apply (*rename-tac T*, *rule-tac x*=*T* **in** *exI*)
apply (*clarsimp*, *drule-tac x*=*y* **in** *bspec*, *simp-all*)
done
next
assume ¬ *a islimpt S*
then show *trivial-limit* (*at a within S*)
unfolding *trivial-limit-def eventually-at-topological islimpt-def*
by *metis*
qed

lemma *trivial-limit-at-iff*: *trivial-limit* (*at a*) ⟷ ¬ *a islimpt UNIV*
using *trivial-limit-within* [*of a UNIV*] **by** *simp*

lemma *trivial-limit-at*:
fixes *a* :: *'a*::*perfect-space*
shows ¬ *trivial-limit* (*at a*)
by (*rule at-neq-bot*)

lemma *trivial-limit-at-infinity*:
¬ *trivial-limit* (*at-infinity* :: (*'a*::{*real-normed-vector*,*perfect-space*}) *filter*)

unfolding *trivial-limit-def eventually-at-infinity*
apply *clarsimp*
apply (*subgoal-tac* $\exists x::'a.\ x \neq 0$, *clarify*)
apply (*rule-tac* x=*scaleR* (b / *norm* x) x **in** *exI*, *simp*)
apply (*cut-tac islimpt-UNIV* [*of* $0::'a$, *unfolded islimpt-def*])
apply (*drule-tac* x=*UNIV* **in** *spec*, *simp*)
done

lemma *not-trivial-limit-within*: $\neg$ *trivial-limit* (*at* x *within* S) = ($x \in$ *closure* ($S - \{x\}$))
using *islimpt-in-closure*
by (*metis trivial-limit-within*)

lemma *at-within-eq-bot-iff*: (*at* c *within* A = *bot*) $\longleftrightarrow$ ($c \notin$ *closure* ($A - \{c\}$))
using *not-trivial-limit-within*[*of* c A] **by** *blast*

Some property holds "sufficiently close" to the limit point.

lemma *trivial-limit-eventually*: *trivial-limit net* $\Longrightarrow$ *eventually* P *net*
by *simp*

lemma *trivial-limit-eq*: *trivial-limit net* $\longleftrightarrow$ ($\forall P.$ *eventually* P *net*)
by (*simp add*: *filter-eq-iff*)

16.15 Limits

lemma *Lim*:
($f \longrightarrow l$) *net* $\longleftrightarrow$
trivial-limit net $\vee$
($\forall e{>}0.$ *eventually* ($\lambda x.$ *dist* ($f\ x$) $l < e$) *net*)
unfolding *tendsto-iff trivial-limit-eq* **by** *auto*

Show that they yield usual definitions in the various cases.

lemma *Lim-within-le*: ($f \longrightarrow l$)(*at* a *within* S) $\longleftrightarrow$
($\forall e{>}0.\ \exists d{>}0.\ \forall x{\in}S.\ 0 < dist\ x\ a \wedge dist\ x\ a \leq d \longrightarrow dist\ (f\ x)\ l < e$)
by (*auto simp add*: *tendsto-iff eventually-at-le*)

lemma *Lim-within*: ($f \longrightarrow l$) (*at* a *within* S) $\longleftrightarrow$
($\forall e > 0.\ \exists d{>}0.\ \forall x \in S.\ 0 < dist\ x\ a \wedge dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l < e$)
by (*auto simp add*: *tendsto-iff eventually-at*)

corollary *Lim-withinI* [*intro?*]:
assumes $\bigwedge e.\ e > 0 \Longrightarrow \exists d{>}0.\ \forall x \in S.\ 0 < dist\ x\ a \wedge dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l \leq e$
shows ($f \longrightarrow l$) (*at* a *within* S)
apply (*simp add*: *Lim-within*, *clarify*)
apply (*rule ex-forward* [*OF assms* [*OF half-gt-zero*]], *auto*)
done

lemma *Lim-at*: ($f \longrightarrow l$) (*at* a) $\longleftrightarrow$

$(\forall\, e>0.\ \exists\, d>0.\ \forall\, x.\ 0 < dist\ x\ a \wedge dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp add*: *tendsto-iff eventually-at2*)

lemma *Lim-at-infinity*:
$(f \longrightarrow l)$ *at-infinity* $\longleftrightarrow$ $(\forall\, e>0.\ \exists\, b.\ \forall\, x.\ norm\ x \geq b \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp add*: *tendsto-iff eventually-at-infinity*)

corollary *Lim-at-infinityI* [*intro?*]:
assumes $\bigwedge e.\ e > 0 \Longrightarrow \exists\, B.\ \forall\, x.\ norm\ x \geq B \longrightarrow dist\ (f\ x)\ l \leq e$
shows $(f \longrightarrow l)$ *at-infinity*
apply (*simp add*: *Lim-at-infinity*, *clarify*)
apply (*rule ex-forward* [*OF assms* [*OF half-gt-zero*]], *auto*)
done

lemma *Lim-eventually*: *eventually* $(\lambda x.\ f\ x = l)$ *net* $\Longrightarrow$ $(f \longrightarrow l)$ *net*
by (*rule topological-tendstoI*, *auto elim*: *eventually-mono*)

The expected monotonicity property.

lemma *Lim-Un*:
assumes $(f \longrightarrow l)$ $(at\ x\ within\ S)$ $(f \longrightarrow l)$ $(at\ x\ within\ T)$
shows $(f \longrightarrow l)$ $(at\ x\ within\ (S \cup T))$
using *assms* **unfolding** *at-within-union* **by** (*rule filterlim-sup*)

lemma *Lim-Un-univ*:
$(f \longrightarrow l)$ $(at\ x\ within\ S) \Longrightarrow (f \longrightarrow l)$ $(at\ x\ within\ T) \Longrightarrow$
$S \cup T = UNIV \Longrightarrow (f \longrightarrow l)$ $(at\ x)$
by (*metis Lim-Un*)

Interrelations between restricted and unrestricted limits.

lemma *Lim-at-imp-Lim-at-within*:
$(f \longrightarrow l)$ $(at\ x) \Longrightarrow (f \longrightarrow l)$ $(at\ x\ within\ S)$
by (*metis order-refl filterlim-mono subset-UNIV at-le*)

lemma *eventually-within-interior*:
assumes $x \in interior\ S$
shows *eventually* P $(at\ x\ within\ S) \longleftrightarrow$ *eventually* P $(at\ x)$
(**is** *?lhs* = *?rhs*)
proof
from *assms* **obtain** T **where** T: *open* T $x \in T$ $T \subseteq S$ **..**
{
assume *?lhs*
then obtain A **where** *open* A **and** $x \in A$ **and** $\forall\, y{\in}A.\ y \neq x \longrightarrow y \in S \longrightarrow P\ y$
unfolding *eventually-at-topological*
by *auto*
with T **have** *open* $(A \cap T)$ **and** $x \in A \cap T$ **and** $\forall\, y \in A \cap T.\ y \neq x \longrightarrow P\ y$
by *auto*
then show *?rhs*
unfolding *eventually-at-topological* **by** *auto*

```
  next
    assume ?rhs
    then show ?lhs
      by (auto elim: eventually-mono simp: eventually-at-filter)
  }
qed

lemma at-within-interior:
  x ∈ interior S ⟹ at x within S = at x
  unfolding filter-eq-iff by (intro allI eventually-within-interior)

lemma Lim-within-LIMSEQ:
  fixes a :: 'a::first-countable-topology
  assumes ∀ S. (∀ n. S n ≠ a ∧ S n ∈ T) ∧ S ⟶ a ⟶ (λn. X (S n)) ⟶
L
  shows (X ⟶ L) (at a within T)
  using assms unfolding tendsto-def [where l=L]
  by (simp add: sequentially-imp-eventually-within)

lemma Lim-right-bound:
  fixes f :: 'a :: {linorder-topology, conditionally-complete-linorder, no-top} ⇒
    'b::{linorder-topology, conditionally-complete-linorder}
  assumes mono: ⋀a b. a ∈ I ⟹ b ∈ I ⟹ x < a ⟹ a ≤ b ⟹ f a ≤ f b
    and bnd: ⋀a. a ∈ I ⟹ x < a ⟹ K ≤ f a
  shows (f ⟶ Inf (f ‘ ({x<..} ∩ I))) (at x within ({x<..} ∩ I))
proof (cases {x<..} ∩ I = {})
  case True
  then show ?thesis by simp
next
  case False
  show ?thesis
  proof (rule order-tendstoI)
    fix a
    assume a: a < Inf (f ‘ ({x<..} ∩ I))
    {
      fix y
      assume y ∈ {x<..} ∩ I
      with False bnd have Inf (f ‘ ({x<..} ∩ I)) ≤ f y
        by (auto intro!: cInf-lower bdd-belowI2 simp del: Inf-image-eq)
      with a have a < f y
        by (blast intro: less-le-trans)
    }
    then show eventually (λx. a < f x) (at x within ({x<..} ∩ I))
      by (auto simp: eventually-at-filter intro: exI[of - 1] zero-less-one)
  next
    fix a
    assume Inf (f ‘ ({x<..} ∩ I)) < a
    from cInf-lessD[OF - this] False obtain y where y: x < y y ∈ I f y < a
      by auto
```

```
    then have eventually (λx. x ∈ I ⟶ f x < a) (at-right x)
      unfolding eventually-at-right[OF ‹x < y›] by (metis less-imp-le le-less-trans
mono)
    then show eventually (λx. f x < a) (at x within ({x<..} ∩ I))
      unfolding eventually-at-filter by eventually-elim simp
  qed
qed
```

Another limit point characterization.

```
lemma islimpt-sequential:
  fixes x :: 'a::first-countable-topology
  shows x islimpt S ⟷ (∃f. (∀n::nat. f n ∈ S - {x}) ∧ (f ⟶ x) sequentially)
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  from countable-basis-at-decseq[of x] obtain A where A:
      ⋀i. open (A i)
      ⋀i. x ∈ A i
      ⋀S. open S ⟹ x ∈ S ⟹ eventually (λi. A i ⊆ S) sequentially
    by blast
  def f ≡ λn. SOME y. y ∈ S ∧ y ∈ A n ∧ x ≠ y
  {
    fix n
    from ‹?lhs› have ∃y. y ∈ S ∧ y ∈ A n ∧ x ≠ y
      unfolding islimpt-def using A(1,2)[of n] by auto
    then have f n ∈ S ∧ f n ∈ A n ∧ x ≠ f n
      unfolding f-def by (rule someI-ex)
    then have f n ∈ S f n ∈ A n x ≠ f n by auto
  }
  then have ∀n. f n ∈ S - {x} by auto
  moreover have (λn. f n) ⟶⟶ x
  proof (rule topological-tendstoI)
    fix S
    assume open S x ∈ S
    from A(3)[OF this] ‹⋀n. f n ∈ A n›
    show eventually (λx. f x ∈ S) sequentially
      by (auto elim!: eventually-mono)
  qed
  ultimately show ?rhs by fast
next
  assume ?rhs
  then obtain f :: nat ⇒ 'a where f: ⋀n. f n ∈ S - {x} and lim: f ⟶⟶ x
    by auto
  show ?lhs
    unfolding islimpt-def
  proof safe
    fix T
    assume open T x ∈ T
    from lim[THEN topological-tendstoD, OF this] f
```

show $\exists\, y{\in}S.\ y \in T \wedge y \neq x$
unfolding *eventually-sequentially* **by** *auto*
qed
qed

lemma *Lim-null*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
shows $(f \longrightarrow l)\ net \longleftrightarrow ((\lambda x.\ f(x) - l) \longrightarrow 0)\ net$
by (*simp add*: *Lim dist-norm*)

lemma *Lim-null-comparison*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
assumes *eventually* $(\lambda x.\ norm\ (f\ x) \leq g\ x)\ net$ $(g \longrightarrow 0)\ net$
shows $(f \longrightarrow 0)\ net$
using *assms*(*2*)
proof (*rule metric-tendsto-imp-tendsto*)
show *eventually* $(\lambda x.\ dist\ (f\ x)\ 0 \leq dist\ (g\ x)\ 0)\ net$
using *assms*(*1*) **by** (*rule eventually-mono*) (*simp add*: *dist-norm*)
qed

lemma *Lim-transform-bound*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
and $g :: 'a \Rightarrow 'c$::*real-normed-vector*
assumes *eventually* $(\lambda n.\ norm\ (f\ n) \leq norm\ (g\ n))\ net$
and $(g \longrightarrow 0)\ net$
shows $(f \longrightarrow 0)\ net$
using *assms*(*1*) *tendsto-norm-zero* [*OF assms*(*2*)]
by (*rule Lim-null-comparison*)

Deducing things about the limit from the elements.

lemma *Lim-in-closed-set*:
assumes *closed S*
and *eventually* $(\lambda x.\ f(x) \in S)\ net$
and $\neg$ *trivial-limit net* $(f \longrightarrow l)\ net$
shows $l \in S$
proof (*rule ccontr*)
assume $l \notin S$
with ‹*closed S*› **have** *open* $(-\ S)\ l \in -\ S$
by (*simp-all add*: *open-Compl*)
with *assms*(*4*) **have** *eventually* $(\lambda x.\ f\ x \in -\ S)\ net$
by (*rule topological-tendstoD*)
with *assms*(*2*) **have** *eventually* $(\lambda x.\ False)\ net$
by (*rule eventually-elim2*) *simp*
with *assms*(*3*) **show** *False*
by (*simp add*: *eventually-False*)
qed

Need to prove closed(cball(x,e)) before deducing this as a corollary.

lemma *Lim-dist-ubound*:

assumes $\neg$(*trivial-limit net*)
and ($f \longrightarrow l$) *net*
and *eventually* ($\lambda x.$ *dist a* ($f\ x$) $\leq e$) *net*
shows *dist a l* $\leq e$
using *assms* **by** (*fast intro*: *tendsto-le tendsto-intros*)

lemma *Lim-norm-ubound*:
fixes f :: *'a* $\Rightarrow$ *'b*::*real-normed-vector*
assumes $\neg$(*trivial-limit net*) ($f \longrightarrow l$) *net eventually* ($\lambda x.$ *norm*($f\ x$) $\leq e$) *net*
shows *norm*(l) $\leq e$
using *assms* **by** (*fast intro*: *tendsto-le tendsto-intros*)

lemma *Lim-norm-lbound*:
fixes f :: *'a* $\Rightarrow$ *'b*::*real-normed-vector*
assumes $\neg$ *trivial-limit net*
and ($f \longrightarrow l$) *net*
and *eventually* ($\lambda x.$ $e \leq$ *norm* ($f\ x$)) *net*
shows $e \leq$ *norm l*
using *assms* **by** (*fast intro*: *tendsto-le tendsto-intros*)

Limit under bilinear function

lemma *Lim-bilinear*:
assumes ($f \longrightarrow l$) *net*
and ($g \longrightarrow m$) *net*
and *bounded-bilinear h*
shows (($\lambda x.$ *h* ($f\ x$) ($g\ x$)) $\longrightarrow$ (*h l m*)) *net*
using ‹*bounded-bilinear h*› ‹($f \longrightarrow l$) *net*› ‹($g \longrightarrow m$) *net*›
by (*rule bounded-bilinear.tendsto*)

These are special for limits out of the same vector space.

lemma *Lim-within-id*: (*id* $\longrightarrow$ *a*) (*at a within s*)
unfolding *id-def* **by** (*rule tendsto-ident-at*)

lemma *Lim-at-id*: (*id* $\longrightarrow$ *a*) (*at a*)
unfolding *id-def* **by** (*rule tendsto-ident-at*)

lemma *Lim-at-zero*:
fixes *a* :: *'a*::*real-normed-vector*
and *l* :: *'b*::*topological-space*
shows ($f \longrightarrow l$) (*at a*) $\longleftrightarrow$ (($\lambda x.$ $f(a + x)$) $\longrightarrow$ *l*) (*at 0*)
using *LIM-offset-zero LIM-offset-zero-cancel* **..**

It's also sometimes useful to extract the limit point from the filter.

abbreviation *netlimit* :: *'a*::*t2-space filter* $\Rightarrow$ *'a*
where *netlimit F* $\equiv$ *Lim F* ($\lambda x.$ *x*)

lemma *netlimit-within*: $\neg$ *trivial-limit* (*at a within S*) $\Longrightarrow$ *netlimit* (*at a within S*) = *a*
by (*rule tendsto-Lim*) (*auto intro*: *tendsto-intros*)

lemma *netlimit-at*:
fixes $a :: 'a$::{*perfect-space*,*t2-space*}
shows *netlimit* (*at a*) = *a*
using *netlimit-within* [*of a UNIV*] **by** *simp*

lemma *lim-within-interior*:
$x \in$ *interior S* $\Longrightarrow$ (*f* $\longrightarrow$ *l*) (*at x within S*) $\longleftrightarrow$ (*f* $\longrightarrow$ *l*) (*at x*)
by (*metis at-within-interior*)

lemma *netlimit-within-interior*:
fixes *x* :: $'a$::{*t2-space*,*perfect-space*}
assumes $x \in$ *interior S*
shows *netlimit* (*at x within S*) = *x*
using *assms* **by** (*metis at-within-interior netlimit-at*)

lemma *netlimit-at-vector*:
fixes *a* :: $'a$::*real-normed-vector*
shows *netlimit* (*at a*) = *a*
proof (*cases* $\exists x.\ x \neq a$)
case *True* **then obtain** *x* **where** *x*: $x \neq a$ **..**
have ¬ *trivial-limit* (*at a*)
unfolding *trivial-limit-def eventually-at dist-norm*
apply *clarsimp*
apply (*rule-tac x*=*a* + *scaleR* (*d* / *2*) (*sgn* (*x* − *a*)) **in** *exI*)
apply (*simp add*: *norm-sgn sgn-zero-iff x*)
done
then show *?thesis*
by (*rule netlimit-within* [*of a UNIV*])
qed *simp*

Useful lemmas on closure and set of possible sequential limits.

lemma *closure-sequential*:
fixes *l* :: $'a$::*first-countable-topology*
shows *l* $\in$ *closure S* $\longleftrightarrow$ ($\exists x.\ (\forall n.\ x\ n \in S) \wedge$ (*x* $\longrightarrow$ *l*) *sequentially*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
moreover
{
assume $l \in S$
then have *?rhs* **using** *tendsto-const*[*of l sequentially*] **by** *auto*
}
moreover
{
assume *l islimpt S*
then have *?rhs* **unfolding** *islimpt-sequential* **by** *auto*
}
ultimately show *?rhs*

unfolding *closure-def* **by** *auto*
next
assume *?rhs*
then show *?lhs* **unfolding** *closure-def islimpt-sequential* **by** *auto*
qed

lemma *closed-sequential-limits*:
fixes S :: *'a::first-countable-topology set*
shows *closed* $S \longleftrightarrow (\forall x\ l.\ (\forall n.\ x\ n \in S) \wedge (x \longrightarrow l)\ \textit{sequentially} \longrightarrow l \in S)$
by (*metis closure-sequential closure-subset-eq subset-iff*)

lemma *closure-approachable*:
fixes S :: *'a::metric-space set*
shows $x \in \textit{closure}\ S \longleftrightarrow (\forall e{>}0.\ \exists y{\in}S.\ \textit{dist}\ y\ x < e)$
apply (*auto simp add*: *closure-def islimpt-approachable*)
apply (*metis dist-self*)
done

lemma *closed-approachable*:
fixes S :: *'a::metric-space set*
shows *closed* $S \Longrightarrow (\forall e{>}0.\ \exists y{\in}S.\ \textit{dist}\ y\ x < e) \longleftrightarrow x \in S$
by (*metis closure-closed closure-approachable*)

lemma *closure-contains-Inf*:
fixes S :: *real set*
assumes $S \neq \{\}$ *bdd-below* S
shows *Inf* $S \in \textit{closure}\ S$
proof −
have $*$: $\forall x{\in}S.\ \textit{Inf}\ S \leq x$
using *cInf-lower*[*of - S*] *assms* **by** *metis*
{
fix e :: *real*
assume $e > 0$
then have *Inf* $S <$ *Inf* $S + e$ **by** *simp*
with *assms* **obtain** x **where** $x \in S\ x < \textit{Inf}\ S + e$
by (*subst* (*asm*) *cInf-less-iff*) *auto*
with $*$ **have** $\exists x{\in}S.\ \textit{dist}\ x\ (\textit{Inf}\ S) < e$
by (*intro bexI*[*of - x*]) (*auto simp add*: *dist-real-def*)
}
then show *?thesis* **unfolding** *closure-approachable* **by** *auto*
qed

lemma *closed-contains-Inf*:
fixes S :: *real set*
shows $S \neq \{\} \Longrightarrow \textit{bdd-below}\ S \Longrightarrow \textit{closed}\ S \Longrightarrow \textit{Inf}\ S \in S$
by (*metis closure-contains-Inf closure-closed assms*)

lemma *closed-subset-contains-Inf*:
fixes $A\ C$ :: *real set*

shows *closed C* $\Longrightarrow$ $A \subseteq C \Longrightarrow A \neq \{\} \Longrightarrow$ *bdd-below A* $\Longrightarrow$ *Inf A* $\in C$
by (*metis closure-contains-Inf closure-minimal subset-eq*)

lemma *atLeastAtMost-subset-contains-Inf*:
fixes *A* :: *real set* **and** *a b* :: *real*
shows $A \neq \{\} \Longrightarrow a \leq b \Longrightarrow A \subseteq \{a..b\} \Longrightarrow$ *Inf A* $\in \{a..b\}$
by (*rule closed-subset-contains-Inf*)
(*auto intro*: *closed-real-atLeastAtMost intro*!: *bdd-belowI*[*of A a*])

lemma *not-trivial-limit-within-ball*:
$\neg$ *trivial-limit* (*at x within S*) $\longleftrightarrow$ ($\forall e{>}0.\ S \cap$ *ball x e* $- \{x\} \neq \{\}$)
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
show *?rhs* **if** *?lhs*
proof −
{
fix *e* :: *real*
assume *e* > *0*
then obtain *y* **where** $y \in S - \{x\}$ **and** *dist y x* < *e*
using ‹*?lhs*› *not-trivial-limit-within*[*of x S*] *closure-approachable*[*of x S* − $\{x\}$]
by *auto*
then have $y \in S \cap$ *ball x e* $- \{x\}$
unfolding *ball-def* **by** (*simp add*: *dist-commute*)
then have $S \cap$ *ball x e* $- \{x\} \neq \{\}$ **by** *blast*
}
then show *?thesis* **by** *auto*
qed
show *?lhs* **if** *?rhs*
proof −
{
fix *e* :: *real*
assume *e* > *0*
then obtain *y* **where** $y \in S \cap$ *ball x e* $- \{x\}$
using ‹*?rhs*› **by** *blast*
then have $y \in S - \{x\}$ **and** *dist y x* < *e*
unfolding *ball-def* **by** (*simp-all add*: *dist-commute*)
then have $\exists y \in S - \{x\}.$ *dist y x* < *e*
by *auto*
}
then show *?thesis*
using *not-trivial-limit-within*[*of x S*] *closure-approachable*[*of x S* − $\{x\}$]
by *auto*
qed
qed

16.16 Infimum Distance

definition *infdist x A* = (*if A* = $\{\}$ *then 0 else INF a*:*A*. *dist x a*)

lemma *bdd-below-infdist*[*intro*, *simp*]: *bdd-below* (*dist x*‘*A*)
by (*auto intro*!: *zero-le-dist*)

lemma *infdist-notempty*: $A \neq \{\} \Longrightarrow$ *infdist x A* = (*INF a*:*A*. *dist x a*)
by (*simp add*: *infdist-def*)

lemma *infdist-nonneg*: *0* ≤ *infdist x A*
by (*auto simp add*: *infdist-def intro*: *cINF-greatest*)

lemma *infdist-le*: *a* ∈ *A* ⟹ *infdist x A* ≤ *dist x a*
by (*auto intro*: *cINF-lower simp add*: *infdist-def*)

lemma *infdist-le2*: *a* ∈ *A* ⟹ *dist x a* ≤ *d* ⟹ *infdist x A* ≤ *d*
by (*auto intro*!: *cINF-lower2 simp add*: *infdist-def*)

lemma *infdist-zero*[*simp*]: *a* ∈ *A* ⟹ *infdist a A* = *0*
by (*auto intro*!: *antisym infdist-nonneg infdist-le2*)

lemma *infdist-triangle*: *infdist x A* ≤ *infdist y A* + *dist x y*
proof (*cases A* = {})
case *True*
then show *?thesis* **by** (*simp add*: *infdist-def*)
next
case *False*
then obtain *a* **where** *a* ∈ *A* **by** *auto*
have *infdist x A* ≤ *Inf* {*dist x y* + *dist y a* |*a*. *a* ∈ *A*}
proof (*rule cInf-greatest*)
from ‹*A* ≠ {}› **show** {*dist x y* + *dist y a* |*a*. *a* ∈ *A*} ≠ {}
by *simp*
fix *d*
assume *d* ∈ {*dist x y* + *dist y a* |*a*. *a* ∈ *A*}
then obtain *a* **where** *d*: *d* = *dist x y* + *dist y a a* ∈ *A*
by *auto*
show *infdist x A* ≤ *d*
unfolding *infdist-notempty*[*OF* ‹*A* ≠ {}›]
proof (*rule cINF-lower2*)
show *a* ∈ *A* **by** *fact*
show *dist x a* ≤ *d*
unfolding *d* **by** (*rule dist-triangle*)
qed *simp*
qed
also have ... = *dist x y* + *infdist y A*
proof (*rule cInf-eq*, *safe*)
fix *a*
assume *a* ∈ *A*
then show *dist x y* + *infdist y A* ≤ *dist x y* + *dist y a*
by (*auto intro*: *infdist-le*)
next

```
    fix i
    assume inf: ⋀d. d ∈ {dist x y + dist y a |a. a ∈ A} ⟹ i ≤ d
    then have i − dist x y ≤ infdist y A
      unfolding infdist-notempty[OF ‹A ≠ {}›] using ‹a ∈ A›
      by (intro cINF-greatest) (auto simp: field-simps)
    then show i ≤ dist x y + infdist y A
      by simp
  qed
  finally show ?thesis by simp
qed

lemma in-closure-iff-infdist-zero:
  assumes A ≠ {}
  shows x ∈ closure A ⟷ infdist x A = 0
proof
  assume x ∈ closure A
  show infdist x A = 0
  proof (rule ccontr)
    assume infdist x A ≠ 0
    with infdist-nonneg[of x A] have infdist x A > 0
      by auto
    then have ball x (infdist x A) ∩ closure A = {}
      apply auto
       apply (metis ‹x ∈ closure A› closure-approachable dist-commute infdist-le
not-less)
      done
    then have x ∉ closure A
      by (metis ‹0 < infdist x A› centre-in-ball disjoint-iff-not-equal)
    then show False using ‹x ∈ closure A› by simp
  qed
next
  assume x: infdist x A = 0
  then obtain a where a ∈ A
    by atomize-elim (metis all-not-in-conv assms)
  show x ∈ closure A
    unfolding closure-approachable
    apply safe
  proof (rule ccontr)
    fix e :: real
    assume e > 0
    assume ¬ (∃ y∈A. dist y x < e)
    then have infdist x A ≥ e using ‹a ∈ A›
      unfolding infdist-def
      by (force simp: dist-commute intro: cINF-greatest)
    with x ‹e > 0› show False by auto
  qed
qed

lemma in-closed-iff-infdist-zero:
```

assumes *closed A* $A \neq \{\}$
shows $x \in A \longleftrightarrow$ *infdist x A* $= 0$
proof −
have $x \in$ *closure A* $\longleftrightarrow$ *infdist x A* $= 0$
by (*rule in-closure-iff-infdist-zero*) *fact*
with *assms* **show** *?thesis* **by** *simp*
qed

lemma *tendsto-infdist* [*tendsto-intros*]:
assumes *f*: $(f \longrightarrow l)$ *F*
shows $((\lambda x.$ *infdist* $(f\ x)$ *A*$) \longrightarrow$ *infdist l A*$)$ *F*
proof (*rule tendstoI*)
fix *e* ::*real*
assume $e > 0$
from *tendstoD*[*OF f this*]
show *eventually* $(\lambda x.$ *dist* $($*infdist* $(f\ x)$ *A*$)$ $($*infdist l A*$) < e)$ *F*
proof (*eventually-elim*)
fix *x*
from *infdist-triangle*[*of l A f x*] *infdist-triangle*[*of f x A l*]
have *dist* $($*infdist* $(f\ x)$ *A*$)$ $($*infdist l A*$) \leq$ *dist* $(f\ x)$ *l*
by (*simp add*: *dist-commute dist-real-def*)
also assume *dist* $(f\ x)$ $l < e$
finally show *dist* $($*infdist* $(f\ x)$ *A*$)$ $($*infdist l A*$) < e$ **.**
qed
qed

Some other lemmas about sequences.

lemma *sequentially-offset*:
assumes *eventually* $(\lambda i.$ *P i*$)$ *sequentially*
shows *eventually* $(\lambda i.$ *P* $(i + k))$ *sequentially*
using *assms* **by** (*rule eventually-sequentially-seg* [*THEN iffD2*])

lemma *seq-offset-neg*:
$(f \longrightarrow l)$ *sequentially* $\Longrightarrow$ $((\lambda i.\ f(i - k)) \longrightarrow l)$ *sequentially*
apply (*erule filterlim-compose*)
apply (*simp add*: *filterlim-def le-sequentially eventually-filtermap eventually-sequentially*)
apply *arith*
done

lemma *seq-harmonic*: $((\lambda n.$ *inverse* $($*real n*$)) \longrightarrow 0)$ *sequentially*
using *LIMSEQ-inverse-real-of-nat* **by** (*rule LIMSEQ-imp-Suc*)

16.17 More properties of closed balls

lemma *closed-cball* [*iff*]: *closed* $($*cball x e*$)$
proof −
have *closed* $($*dist x* $-`$ $\{..e\})$
by (*intro closed-vimage closed-atMost continuous-intros*)
also have *dist x* $-`$ $\{..e\} =$ *cball x e*

```
    by auto
  finally show ?thesis .
qed

lemma open-contains-cball: open S ⟷ (∀ x∈S. ∃ e>0.  cball x e ⊆ S)
proof −
  {
    fix x and e::real
    assume x∈S e>0 ball x e ⊆ S
    then have ∃ d>0. cball x d ⊆ S unfolding subset-eq by (rule-tac x=e/2 in
exI, auto)
  }
  moreover
  {
    fix x and e::real
    assume x∈S e>0 cball x e ⊆ S
    then have ∃ d>0. ball x d ⊆ S
      unfolding subset-eq
      apply(rule-tac x=e/2 in exI)
      apply auto
      done
  }
  ultimately show ?thesis
    unfolding open-contains-ball by auto
qed

lemma open-contains-cball-eq: open S ⟹ (∀ x. x ∈ S ⟷ (∃ e>0. cball x e ⊆
S))
  by (metis open-contains-cball subset-eq order-less-imp-le centre-in-cball)

lemma mem-interior-cball: x ∈ interior S ⟷ (∃ e>0. cball x e ⊆ S)
  apply (simp add: interior-def, safe)
  apply (force simp add: open-contains-cball)
  apply (rule-tac x=ball x e in exI)
  apply (simp add: subset-trans [OF ball-subset-cball])
  done

lemma islimpt-ball:
  fixes x y :: 'a::{real-normed-vector,perfect-space}
  shows y islimpt ball x e ⟷ 0 < e ∧ y ∈ cball x e
  (is ?lhs ⟷ ?rhs)
proof
  show ?rhs if ?lhs
  proof
    {
      assume e ≤ 0
      then have ∗: ball x e = {}
        using ball-eq-empty[of x e] by auto
      have False using ‹?lhs›
```

```
      unfolding ∗ using islimpt-EMPTY[of y] by auto
    }
    then show e > 0 by (metis not-less)
    show y ∈ cball x e
      using closed-cball[of x e] islimpt-subset[of y ball x e cball x e]
        ball-subset-cball[of x e] ‹?lhs›
      unfolding closed-limpt by auto
  qed
  show ?lhs if ?rhs
  proof −
    from that have e > 0 by auto
    {
      fix d :: real
      assume d > 0
      have ∃x′∈ball x e. x′ ≠ y ∧ dist x′ y < d
      proof (cases d ≤ dist x y)
        case True
        then show ∃x′∈ball x e. x′ ≠ y ∧ dist x′ y < d
        proof (cases x = y)
          case True
          then have False
            using ‹d ≤ dist x y› ‹d>0› by auto
          then show ∃x′∈ball x e. x′ ≠ y ∧ dist x′ y < d
            by auto
        next
          case False
          have dist x (y − (d / (2 ∗ dist y x)) ∗R (y − x)) =
            norm (x − y + (d / (2 ∗ norm (y − x))) ∗R (y − x))
         unfolding mem-cball mem-ball dist-norm diff-diff-eq2 diff-add-eq[symmetric]
            by auto
          also have ... = |− 1 + d / (2 ∗ norm (x − y))| ∗ norm (x − y)
            using scaleR-left-distrib[of − 1 d / (2 ∗ norm (y − x)), symmetric, of
y − x]
            unfolding scaleR-minus-left scaleR-one
            by (auto simp add: norm-minus-commute)
          also have ... = |− norm (x − y) + d / 2|
            unfolding abs-mult-pos[of norm (x − y), OF norm-ge-zero[of x − y]]
            unfolding distrib-right using ‹x≠y› by auto
          also have ... ≤ e − d/2 using ‹d ≤ dist x y› and ‹d>0› and ‹?rhs›
            by (auto simp add: dist-norm)
          finally have y − (d / (2 ∗ dist y x)) ∗R (y − x) ∈ ball x e using ‹d>0›
            by auto
          moreover
          have (d / (2∗dist y x)) ∗R (y − x) ≠ 0
            using ‹x≠y›[unfolded dist-nz] ‹d>0› unfolding scaleR-eq-0-iff
            by (auto simp add: dist-commute)
          moreover
          have dist (y − (d / (2 ∗ dist y x)) ∗R (y − x)) y < d
            unfolding dist-norm
```

```
        apply simp
        unfolding norm-minus-cancel
        using ‹d > 0› ‹x≠y›[unfolded dist-nz] dist-commute[of x y]
        unfolding dist-norm
        apply auto
        done
      ultimately show ∃x'∈ball x e. x' ≠ y ∧ dist x' y < d
        apply (rule-tac x = y - (d / (2*dist y x)) *R (y - x) in bexI)
        apply auto
        done
    qed
  next
    case False
    then have d > dist x y by auto
    show ∃x' ∈ ball x e. x' ≠ y ∧ dist x' y < d
    proof (cases x = y)
      case True
      obtain z where **: z ≠ y dist z y < min e d
        using perfect-choose-dist[of min e d y]
        using ‹d > 0› ‹e>0› by auto
      show ∃x'∈ball x e. x' ≠ y ∧ dist x' y < d
        unfolding ‹x = y›
        using ‹z ≠ y› **
        apply (rule-tac x=z in bexI)
        apply (auto simp add: dist-commute)
        done
    next
      case False
      then show ∃x'∈ball x e. x' ≠ y ∧ dist x' y < d
        using ‹d>0› ‹d > dist x y› ‹?rhs›
        apply (rule-tac x=x in bexI)
        apply auto
        done
    qed
  qed
 }
 then show ?thesis
  unfolding mem-cball islimpt-approachable mem-ball by auto
qed
qed

lemma closure-ball-lemma:
 fixes x y :: 'a::real-normed-vector
 assumes x ≠ y
 shows y islimpt ball x (dist x y)
proof (rule islimptI)
 fix T
 assume y ∈ T open T
 then obtain r where 0 < r ∀z. dist z y < r ⟶ z ∈ T
```

unfolding *open-dist* **by** *fast*

def $k \equiv min\ 1\ (r\ /\ (2 * dist\ x\ y))$
def $z \equiv y + scaleR\ k\ (x - y)$
have *z-def2*: $z = x + scaleR\ (1 - k)\ (y - x)$
unfolding *z-def* **by** (*simp add*: *algebra-simps*)
have $dist\ z\ y < r$
unfolding *z-def k-def* **using** ‹$0 < r$›
by (*simp add*: *dist-norm min-def*)
then have $z \in T$
using ‹$\forall z.\ dist\ z\ y < r \longrightarrow z \in T$› **by** *simp*
have $dist\ x\ z < dist\ x\ y$
unfolding *z-def2 dist-norm*
apply (*simp add*: *norm-minus-commute*)
apply (*simp only*: *dist-norm* [*symmetric*])
apply (*subgoal-tac* $|1 - k| * dist\ x\ y < 1 * dist\ x\ y$, *simp*)
apply (*rule mult-strict-right-mono*)
apply (*simp add*: *k-def* ‹$0 < r$› ‹$x \neq y$›)
apply (*simp add*: ‹$x \neq y$›)
done
then have $z \in ball\ x\ (dist\ x\ y)$
by *simp*
have $z \neq y$
unfolding *z-def k-def* **using** ‹$x \neq y$› ‹$0 < r$›
by (*simp add*: *min-def*)
show $\exists z{\in}ball\ x\ (dist\ x\ y).\ z \in T \wedge z \neq y$
using ‹$z \in ball\ x\ (dist\ x\ y)$› ‹$z \in T$› ‹$z \neq y$›
by *fast*
qed

lemma *closure-ball* [*simp*]:
fixes $x :: 'a{::}real\text{-}normed\text{-}vector$
shows $0 < e \Longrightarrow closure\ (ball\ x\ e) = cball\ x\ e$
apply (*rule equalityI*)
apply (*rule closure-minimal*)
apply (*rule ball-subset-cball*)
apply (*rule closed-cball*)
apply (*rule subsetI*, *rename-tac y*)
apply (*simp add*: *le-less* [**where** $'a{=}real$])
apply (*erule disjE*)
apply (*rule subsetD* [*OF closure-subset*], *simp*)
apply (*simp add*: *closure-def*)
apply *clarify*
apply (*rule closure-ball-lemma*)
apply (*simp add*: *zero-less-dist-iff*)
done

lemma *interior-cball* [*simp*]:

```
  fixes x :: 'a::{real-normed-vector, perfect-space}
  shows interior (cball x e) = ball x e
proof (cases e ≥ 0)
  case False note cs = this
  from cs have null: ball x e = {}
    using ball-empty[of e x] by auto
  moreover
  {
    fix y
    assume y ∈ cball x e
    then have False
        by (metis ball-eq-empty null cs dist-eq-0-iff dist-le-zero-iff empty-subsetI
mem-cball subset-antisym subset-ball)
  }
  then have cball x e = {} by auto
  then have interior (cball x e) = {}
    using interior-empty by auto
  ultimately show ?thesis by blast
next
  case True note cs = this
  have ball x e ⊆ cball x e
    using ball-subset-cball by auto
  moreover
  {
    fix S y
    assume as: S ⊆ cball x e open S y∈S
    then obtain d where d>0 and d: ∀ x'. dist x' y < d ⟶ x' ∈ S
      unfolding open-dist by blast
    then obtain xa where xa-y: xa ≠ y and xa: dist xa y < d
      using perfect-choose-dist [of d] by auto
    have xa ∈ S
      using d[THEN spec[where x = xa]]
      using xa by (auto simp add: dist-commute)
    then have xa-cball: xa ∈ cball x e
      using as(1) by auto
    then have y ∈ ball x e
    proof (cases x = y)
      case True
      then have e > 0 using cs order.order-iff-strict xa-cball xa-y by fastforce
      then show y ∈ ball x e
        using ‹x = y › by simp
    next
      case False
      have dist (y + (d / 2 / dist y x) *R (y − x)) y < d
        unfolding dist-norm
        using ‹d>0› norm-ge-zero[of y − x] ‹x ≠ y› by auto
      then have *: y + (d / 2 / dist y x) *R (y − x) ∈ cball x e
        using d as(1)[unfolded subset-eq] by blast
      have y − x ≠ 0 using ‹x ≠ y› by auto
```

```
    hence **:d / (2 * norm (y - x)) > 0
      unfolding zero-less-norm-iff[symmetric] using ‹d>0› by auto
    have dist (y + (d / 2 / dist y x) *R (y - x)) x =
      norm (y + (d / (2 * norm (y - x))) *R y - (d / (2 * norm (y - x))) *R
x - x)
      by (auto simp add: dist-norm algebra-simps)
    also have ... = norm ((1 + d / (2 * norm (y - x))) *R (y - x))
      by (auto simp add: algebra-simps)
    also have ... = |1 + d / (2 * norm (y - x))| * norm (y - x)
      using ** by auto
    also have ... = (dist y x) + d/2
      using ** by (auto simp add: distrib-right dist-norm)
    finally have e ≥ dist x y +d/2
      using *[unfolded mem-cball] by (auto simp add: dist-commute)
    then show y ∈ ball x e
      unfolding mem-ball using ‹d>0› by auto
  qed
 }
 then have ∀ S ⊆ cball x e. open S ⟶ S ⊆ ball x e
  by auto
 ultimately show ?thesis
  using interior-unique[of ball x e cball x e]
  using open-ball[of x e]
  by auto
qed

lemma frontier-ball:
 fixes a :: 'a::real-normed-vector
 shows 0 < e ⟹ frontier(ball a e) = {x. dist a x = e}
 apply (simp add: frontier-def closure-ball interior-open order-less-imp-le)
 apply (simp add: set-eq-iff)
 apply arith
 done

lemma frontier-cball:
 fixes a :: 'a::{real-normed-vector, perfect-space}
 shows frontier (cball a e) = {x. dist a x = e}
 apply (simp add: frontier-def interior-cball closed-cball order-less-imp-le)
 apply (simp add: set-eq-iff)
 apply arith
 done

lemma cball-eq-empty [simp]: cball x e = {} ⟷ e < 0
 apply (simp add: set-eq-iff not-le)
 apply (metis zero-le-dist dist-self order-less-le-trans)
 done

lemma cball-empty [simp]: e < 0 ⟹ cball x e = {}
 by (simp add: cball-eq-empty)
```

lemma *cball-eq-sing*:
fixes $x :: 'a::\{metric\text{-}space,perfect\text{-}space\}$
shows $cball\ x\ e = \{x\} \longleftrightarrow e = 0$
proof (*rule linorder-cases*)
assume e: $0 < e$
obtain a **where** $a \neq x$ $dist\ a\ x < e$
using *perfect-choose-dist* [*OF* e] **by** *auto*
then have $a \neq x$ $dist\ x\ a \leq e$
by (*auto simp add*: *dist-commute*)
with e **show** *?thesis* **by** (*auto simp add*: *set-eq-iff*)
qed *auto*

lemma *cball-sing*:
fixes $x :: 'a::metric\text{-}space$
shows $e = 0 \Longrightarrow cball\ x\ e = \{x\}$
by (*auto simp add*: *set-eq-iff*)

lemma *ball-divide-subset*: $d \geq 1 \Longrightarrow ball\ x\ (e/d) \subseteq ball\ x\ e$
apply (*cases* $e \leq 0$)
apply (*simp add*: *ball-empty divide-simps*)
apply (*rule subset-ball*)
apply (*simp add*: *divide-simps*)
done

lemma *ball-divide-subset-numeral*: $ball\ x\ (e\ /\ numeral\ w) \subseteq ball\ x\ e$
using *ball-divide-subset one-le-numeral* **by** *blast*

lemma *cball-divide-subset*: $d \geq 1 \Longrightarrow cball\ x\ (e/d) \subseteq cball\ x\ e$
apply (*cases* $e < 0$)
apply (*simp add*: *divide-simps*)
apply (*rule subset-cball*)
apply (*metis divide-1 frac-le not-le order-refl zero-less-one*)
done

lemma *cball-divide-subset-numeral*: $cball\ x\ (e\ /\ numeral\ w) \subseteq cball\ x\ e$
using *cball-divide-subset one-le-numeral* **by** *blast*

16.18 Boundedness

definition (**in** *metric-space*) *bounded* :: $'a\ set \Rightarrow bool$
where $bounded\ S \longleftrightarrow (\exists x\ e.\ \forall y{\in}S.\ dist\ x\ y \leq e)$

lemma *bounded-subset-cball*: $bounded\ S \longleftrightarrow (\exists e\ x.\ S \subseteq cball\ x\ e \wedge 0 \leq e)$
unfolding *bounded-def subset-eq* **by** *auto* (*meson order-trans zero-le-dist*)

lemma *bounded-subset-ballD*:
assumes *bounded S* **shows** $\exists r.\ 0 < r \wedge S \subseteq ball\ x\ r$
proof −

obtain *e*::*real* **and** *y* **where** $S \subseteq$ *cball y e* $0 \leq e$
using *assms* **by** (*auto simp*: *bounded-subset-cball*)
then show *?thesis*
apply (*rule-tac x*=*dist x y* + *e* + *1* **in** *exI*)
apply (*simp add*: *add.commute add-pos-nonneg*)
apply (*erule subset-trans*)
apply (*clarsimp simp add*: *cball-def*)
by (*metis add-le-cancel-right add-strict-increasing dist-commute dist-triangle-le zero-less-one*)
qed

lemma *bounded-any-center*: *bounded S* $\longleftrightarrow$ $(\exists e.\ \forall y{\in}S.$ *dist a y* $\leq e)$
unfolding *bounded-def*
by *auto* (*metis add.commute add-le-cancel-right dist-commute dist-triangle-le*)

lemma *bounded-iff*: *bounded S* $\longleftrightarrow$ $(\exists a.\ \forall x{\in}S.$ *norm x* $\leq a)$
unfolding *bounded-any-center* [**where** *a*=*0*]
by (*simp add*: *dist-norm*)

lemma *bdd-above-norm*: *bdd-above* (*norm* ' *X*) $\longleftrightarrow$ *bounded X*
by (*simp add*: *bounded-iff bdd-above-def*)

lemma *bounded-realI*:
assumes $\forall x{\in}s.$ $|x{::}real| \leq B$
shows *bounded s*
unfolding *bounded-def dist-real-def*
by (*metis abs-minus-commute assms diff-0-right*)

lemma *bounded-empty* [*simp*]: *bounded* {}
by (*simp add*: *bounded-def*)

lemma *bounded-subset*: *bounded T* $\Longrightarrow$ $S \subseteq T \Longrightarrow$ *bounded S*
by (*metis bounded-def subset-eq*)

lemma *bounded-interior*[*intro*]: *bounded S* $\Longrightarrow$ *bounded*(*interior S*)
by (*metis bounded-subset interior-subset*)

lemma *bounded-closure*[*intro*]:
assumes *bounded S*
shows *bounded* (*closure S*)
proof −
from *assms* **obtain** *x* **and** *a* **where** *a*: $\forall y{\in}S.$ *dist x y* $\leq a$
unfolding *bounded-def* **by** *auto*
{
fix *y*
assume *y* $\in$ *closure S*
then obtain *f* **where** *f*: $\forall n.$ *f n* $\in S$ (*f* $\longrightarrow$ *y*) *sequentially*
unfolding *closure-sequential* **by** *auto*
have $\forall n.$ *f n* $\in S \longrightarrow$ *dist x* (*f n*) $\leq a$ **using** *a* **by** *simp*

```
    then have eventually (λn. dist x (f n) ≤ a) sequentially
      by (simp add: f(1))
    have dist x y ≤ a
      apply (rule Lim-dist-ubound [of sequentially f])
      apply (rule trivial-limit-sequentially)
      apply (rule f(2))
      apply fact
      done
  }
  then show ?thesis
    unfolding bounded-def by auto
qed

lemma bounded-cball[simp,intro]: bounded (cball x e)
  apply (simp add: bounded-def)
  apply (rule-tac x=x in exI)
  apply (rule-tac x=e in exI)
  apply auto
  done

lemma bounded-ball[simp,intro]: bounded (ball x e)
  by (metis ball-subset-cball bounded-cball bounded-subset)

lemma bounded-Un[simp]: bounded (S ∪ T) ⟷ bounded S ∧ bounded T
  apply (auto simp add: bounded-def)
  by (metis Un-iff add-le-cancel-left dist-triangle le-max-iff-disj max.order-iff)

lemma bounded-Union[intro]: finite F ⟹ ∀ S∈F. bounded S ⟹ bounded (⋃ F)
  by (induct rule: finite-induct[of F]) auto

lemma bounded-UN [intro]: finite A ⟹ ∀ x∈A. bounded (B x) ⟹ bounded
(⋃ x∈A. B x)
  by (induct set: finite) auto

lemma bounded-insert [simp]: bounded (insert x S) ⟷ bounded S
proof −
  have ∀ y∈{x}. dist x y ≤ 0
    by simp
  then have bounded {x}
    unfolding bounded-def by fast
  then show ?thesis
    by (metis insert-is-Un bounded-Un)
qed

lemma finite-imp-bounded [intro]: finite S ⟹ bounded S
  by (induct set: finite) simp-all

lemma bounded-pos: bounded S ⟷ (∃ b>0. ∀ x∈ S. norm x ≤ b)
  apply (simp add: bounded-iff)
```

apply (*subgoal-tac* $\bigwedge x$ (y::*real*). $0 < 1 + |y| \wedge (x \leq y \longrightarrow x \leq 1 + |y|)$)
apply *metis*
apply *arith*
done

lemma *bounded-pos-less*: *bounded* $S \longleftrightarrow (\exists\, b{>}0.\ \forall\, x{\in}\ S.\ norm\ x < b)$
apply (*simp add*: *bounded-pos*)
apply (*safe*; *rule-tac* $x{=}b{+}1$ **in** *exI*; *force*)
done

lemma *Bseq-eq-bounded*:
fixes f :: *nat* $\Rightarrow$ $'a$::*real-normed-vector*
shows *Bseq* $f \longleftrightarrow$ *bounded* (*range* f)
unfolding *Bseq-def bounded-pos* **by** *auto*

lemma *bounded-Int*[*intro*]: *bounded* $S \vee$ *bounded* $T \Longrightarrow$ *bounded* $(S \cap T)$
by (*metis Int-lower1 Int-lower2 bounded-subset*)

lemma *bounded-diff*[*intro*]: *bounded* $S \Longrightarrow$ *bounded* $(S - T)$
by (*metis Diff-subset bounded-subset*)

lemma *not-bounded-UNIV*[*simp*, *intro*]:
$\neg$ *bounded* (*UNIV* :: $'a$::{*real-normed-vector*, *perfect-space*} *set*)
proof (*auto simp add*: *bounded-pos not-le*)
obtain x :: $'a$ **where** $x \neq 0$
using *perfect-choose-dist* [*OF zero-less-one*] **by** *fast*
fix b :: *real*
assume b: $b > 0$
have $b1$: $b + 1 \geq 0$
using b **by** *simp*
with ‹$x \neq 0$› **have** $b < norm\ (scaleR\ (b + 1)\ (sgn\ x))$
by (*simp add*: *norm-sgn*)
then show $\exists\, x{::}'a.\ b < norm\ x$ **..**
qed

corollary *cobounded-imp-unbounded*:
fixes S :: $'a$::{*real-normed-vector*, *perfect-space*} *set*
shows *bounded* $(-S) \Longrightarrow {}^{\sim}$ (*bounded* S)
using *bounded-Un* [*of* S $-S$] **by** (*simp add*: *sup-compl-top*)

lemma *bounded-linear-image*:
assumes *bounded* S
and *bounded-linear* f
shows *bounded* (f ‘ S)
proof −
from *assms*(*1*) **obtain** b **where** b: $b > 0\ \forall\, x{\in}S.\ norm\ x \leq b$
unfolding *bounded-pos* **by** *auto*
from *assms*(*2*) **obtain** B **where** B: $B > 0\ \forall\, x.\ norm\ (f\ x) \leq B * norm\ x$
using *bounded-linear.pos-bounded* **by** (*auto simp add*: *ac-simps*)

```
  {
    fix x
    assume x ∈ S
    then have norm x ≤ b
      using b by auto
    then have norm (f x) ≤ B * b
      using B(2)
      apply (erule-tac x=x in allE)
      apply (metis B(1) B(2) order-trans mult-le-cancel-left-pos)
      done
  }
  then show ?thesis
    unfolding bounded-pos
    apply (rule-tac x=b*B in exI)
    using b B by (auto simp add: mult.commute)
qed

lemma bounded-scaling:
  fixes S :: 'a::real-normed-vector set
  shows bounded S ⟹ bounded ((λx. c *R x) ` S)
  apply (rule bounded-linear-image)
  apply assumption
  apply (rule bounded-linear-scaleR-right)
  done

lemma bounded-translation:
  fixes S :: 'a::real-normed-vector set
  assumes bounded S
  shows bounded ((λx. a + x) ` S)
proof −
  from assms obtain b where b: b > 0 ∀x∈S. norm x ≤ b
    unfolding bounded-pos by auto
  {
    fix x
    assume x ∈ S
    then have norm (a + x) ≤ b + norm a
      using norm-triangle-ineq[of a x] b by auto
  }
  then show ?thesis
    unfolding bounded-pos
    using norm-ge-zero[of a] b(1) and add-strict-increasing[of b 0 norm a]
    by (auto intro!: exI[of - b + norm a])
qed

lemma bounded-uminus [simp]:
  fixes X :: 'a::euclidean-space set
  shows bounded (uminus ` X) ⟷ bounded X
by (auto simp: bounded-def dist-norm; rule-tac x=−x in exI; force simp add:
add.commute norm-minus-commute)
```

Some theorems on sups and infs using the notion "bounded".

lemma *bounded-real*: *bounded* (*S*::*real set*) $\longleftrightarrow$ ($\exists\, a.\ \forall\, x{\in}S.\ |x| \leq a$)
by (*simp add*: *bounded-iff*)

lemma *bounded-imp-bdd-above*: *bounded S* $\Longrightarrow$ *bdd-above* (*S* :: *real set*)
by (*auto simp*: *bounded-def bdd-above-def dist-real-def*)
(*metis abs-le-D1 abs-minus-commute diff-le-eq*)

lemma *bounded-imp-bdd-below*: *bounded S* $\Longrightarrow$ *bdd-below* (*S* :: *real set*)
by (*auto simp*: *bounded-def bdd-below-def dist-real-def*)
(*metis abs-le-D1 add.commute diff-le-eq*)

lemma *bounded-inner-imp-bdd-above*:
assumes *bounded s*
shows *bdd-above* ((λ*x*. *x* · *a*) ‘ *s*)
by (*simp add*: *assms bounded-imp-bdd-above bounded-linear-image bounded-linear-inner-left*)

lemma *bounded-inner-imp-bdd-below*:
assumes *bounded s*
shows *bdd-below* ((λ*x*. *x* · *a*) ‘ *s*)
by (*simp add*: *assms bounded-imp-bdd-below bounded-linear-image bounded-linear-inner-left*)

lemma *bounded-has-Sup*:
fixes *S* :: *real set*
assumes *bounded S*
and *S* ≠ {}
shows ∀ *x*∈*S*. *x* ≤ *Sup S*
and ∀ *b*. (∀ *x*∈*S*. *x* ≤ *b*) ⟶ *Sup S* ≤ *b*
proof
show ∀ *b*. (∀ *x*∈*S*. *x* ≤ *b*) ⟶ *Sup S* ≤ *b*
using *assms* **by** (*metis cSup-least*)
qed (*metis cSup-upper assms*(*1*) *bounded-imp-bdd-above*)

lemma *Sup-insert*:
fixes *S* :: *real set*
shows *bounded S* ⟹ *Sup* (*insert x S*) = (*if S* = {} *then x else max x* (*Sup S*))
by (*auto simp*: *bounded-imp-bdd-above sup-max cSup-insert-If*)

lemma *Sup-insert-finite*:
fixes *S* :: *'a*::*conditionally-complete-linorder set*
shows *finite S* ⟹ *Sup* (*insert x S*) = (*if S* = {} *then x else max x* (*Sup S*))
by (*simp add*: *cSup-insert sup-max*)

lemma *bounded-has-Inf*:
fixes *S* :: *real set*
assumes *bounded S*
and *S* ≠ {}
shows ∀ *x*∈*S*. *x* ≥ *Inf S*
and ∀ *b*. (∀ *x*∈*S*. *x* ≥ *b*) ⟶ *Inf S* ≥ *b*

proof
show $\forall b.\ (\forall x{\in}S.\ x \geq b) \longrightarrow Inf\ S \geq b$
using *assms* **by** (*metis cInf-greatest*)
qed (*metis cInf-lower assms*(*1*) *bounded-imp-bdd-below*)

lemma *Inf-insert*:
fixes S :: *real set*
shows *bounded* $S \Longrightarrow Inf\ (insert\ x\ S) = (if\ S = \{\}\ then\ x\ else\ min\ x\ (Inf\ S))$
by (*auto simp*: *bounded-imp-bdd-below inf-min cInf-insert-If*)

lemma *Inf-insert-finite*:
fixes S :: *'a*::*conditionally-complete-linorder set*
shows *finite* $S \Longrightarrow Inf\ (insert\ x\ S) = (if\ S = \{\}\ then\ x\ else\ min\ x\ (Inf\ S))$
by (*simp add*: *cInf-eq-Min*)

lemma *finite-imp-less-Inf*:
fixes a :: *'a*::*conditionally-complete-linorder*
shows $[\![finite\ X;\ x \in X;\ \bigwedge x.\ x{\in}X \Longrightarrow a < x]\!] \Longrightarrow a < Inf\ X$
by (*induction X rule*: *finite-induct*) (*simp-all add*: *cInf-eq-Min Inf-insert-finite*)

lemma *finite-less-Inf-iff*:
fixes a :: *'a* :: *conditionally-complete-linorder*
shows $[\![finite\ X;\ X \neq \{\}]\!] \Longrightarrow a < Inf\ X \longleftrightarrow (\forall x \in X.\ a < x)$
by (*auto simp*: *cInf-eq-Min*)

lemma *finite-imp-Sup-less*:
fixes a :: *'a*::*conditionally-complete-linorder*
shows $[\![finite\ X;\ x \in X;\ \bigwedge x.\ x{\in}X \Longrightarrow a > x]\!] \Longrightarrow a > Sup\ X$
by (*induction X rule*: *finite-induct*) (*simp-all add*: *cSup-eq-Max Sup-insert-finite*)

lemma *finite-Sup-less-iff*:
fixes a :: *'a* :: *conditionally-complete-linorder*
shows $[\![finite\ X;\ X \neq \{\}]\!] \Longrightarrow a > Sup\ X \longleftrightarrow (\forall x \in X.\ a > x)$
by (*auto simp*: *cSup-eq-Max*)

16.19 Compactness

16.19.1 Bolzano-Weierstrass property

lemma *heine-borel-imp-bolzano-weierstrass*:
assumes *compact s*
and *infinite t*
and $t \subseteq s$
shows $\exists x \in s.\ x\ islimpt\ t$
proof (*rule ccontr*)
assume $\neg\ (\exists x \in s.\ x\ islimpt\ t)$
then obtain f **where** f: $\forall x{\in}s.\ x \in f\ x \wedge open\ (f\ x) \wedge (\forall y{\in}t.\ y \in f\ x \longrightarrow y = x)$
unfolding *islimpt-def*
using *bchoice*[*of s* $\lambda\ x\ T.\ x \in T \wedge open\ T \wedge (\forall y{\in}t.\ y \in T \longrightarrow y = x)$]

by *auto*
obtain g **where** g: $g \subseteq \{t.\ \exists x.\ x \in s \wedge t = f\ x\}$ *finite* g $s \subseteq \bigcup g$
using *assms*(*1*)[*unfolded compact-eq-heine-borel*, *THEN spec*[**where** x=$\{t.\ \exists x.\ x \in s \wedge t = f\ x\}$]]
using f **by** *auto*
from *g*(*1*,*3*) **have** g':$\forall x \in g.\ \exists xa \in s.\ x = f\ xa$
by *auto*
{
fix x y
assume $x \in t$ $y \in t$ $f\ x = f\ y$
then have $x \in f\ x$ $y \in f\ x \longrightarrow y = x$
using f[*THEN bspec*[**where** x=x]] **and** ‹$t \subseteq s$› **by** *auto*
then have $x = y$
using ‹$f\ x = f\ y$› **and** f[*THEN bspec*[**where** x=y]] **and** ‹$y \in t$› **and** ‹$t \subseteq s$›
by *auto*
}
then have *inj-on* f t
unfolding *inj-on-def* **by** *simp*
then have *infinite* (f ‘ t)
using *assms*(*2*) **using** *finite-imageD* **by** *auto*
moreover
{
fix x
assume $x \in t$ $f\ x \notin g$
from *g*(*3*) *assms*(*3*) ‹$x \in t$› **obtain** h **where** $h \in g$ **and** $x \in h$
by *auto*
then obtain y **where** $y \in s$ $h = f\ y$
using g'[*THEN bspec*[**where** x=h]] **by** *auto*
then have $y = x$
using f[*THEN bspec*[**where** x=y]] **and** ‹$x \in t$› **and** ‹$x \in h$›[*unfolded* ‹$h = f\ y$›]
by *auto*
then have *False*
using ‹$f\ x \notin g$› ‹$h \in g$› **unfolding** ‹$h = f\ y$›
by *auto*
}
then have f ‘ $t \subseteq g$ **by** *auto*
ultimately show *False*
using *g*(*2*) **using** *finite-subset* **by** *auto*
qed

lemma *acc-point-range-imp-convergent-subsequence*:
fixes l :: $'a$ :: *first-countable-topology*
assumes l: $\forall U.\ l \in U \longrightarrow$ *open* $U \longrightarrow$ *infinite* ($U \cap$ *range* f)
shows $\exists r.$ *subseq* $r \wedge (f \circ r) \longrightarrow l$
proof −
from *countable-basis-at-decseq*[*of* l]
obtain A **where** A:
$\bigwedge i.$ *open* (A i)
$\bigwedge i.$ $l \in A$ i

```
    ⋀S. open S ⟹ l ∈ S ⟹ eventually (λi. A i ⊆ S) sequentially
   by blast
 def s ≡ λn i. SOME j. i < j ∧ f j ∈ A (Suc n)
 {
  fix n i
  have infinite (A (Suc n) ∩ range f − f‘{.. i})
   using l A by auto
  then have ∃x. x ∈ A (Suc n) ∩ range f − f‘{.. i}
   unfolding ex-in-conv by (intro notI) simp
  then have ∃j. f j ∈ A (Suc n) ∧ j ∉ {.. i}
   by auto
  then have ∃a. i < a ∧ f a ∈ A (Suc n)
   by (auto simp: not-le)
  then have i < s n i f (s n i) ∈ A (Suc n)
   unfolding s-def by (auto intro: someI2-ex)
 }
 note s = this
 def r ≡ rec-nat (s 0 0) s
 have subseq r
  by (auto simp: r-def s subseq-Suc-iff)
 moreover
 have (λn. f (r n)) ⟶ l
 proof (rule topological-tendstoI)
  fix S
  assume open S l ∈ S
  with A(3) have eventually (λi. A i ⊆ S) sequentially
   by auto
  moreover
  {
   fix i
   assume Suc 0 ≤ i
   then have f (r i) ∈ A i
    by (cases i) (simp-all add: r-def s)
  }
  then have eventually (λi. f (r i) ∈ A i) sequentially
   by (auto simp: eventually-sequentially)
  ultimately show eventually (λi. f (r i) ∈ S) sequentially
   by eventually-elim auto
 qed
 ultimately show ∃r. subseq r ∧ (f ∘ r) ⟶ l
  by (auto simp: convergent-def comp-def)
qed

lemma sequence-infinite-lemma:
 fixes f :: nat ⇒ 'a::t1-space
 assumes ∀n. f n ≠ l
  and (f ⟶ l) sequentially
 shows infinite (range f)
proof
```

```
  assume finite (range f)
  then have closed (range f)
    by (rule finite-imp-closed)
  then have open (− range f)
    by (rule open-Compl)
  from assms(1) have l ∈ − range f
    by auto
  from assms(2) have eventually (λn. f n ∈ − range f) sequentially
    using ‹open (− range f)› ‹l ∈ − range f›
    by (rule topological-tendstoD)
  then show False
    unfolding eventually-sequentially
    by auto
qed

lemma closure-insert:
  fixes x :: 'a::t1-space
  shows closure (insert x s) = insert x (closure s)
  apply (rule closure-unique)
  apply (rule insert-mono [OF closure-subset])
  apply (rule closed-insert [OF closed-closure])
  apply (simp add: closure-minimal)
  done

lemma islimpt-insert:
  fixes x :: 'a::t1-space
  shows x islimpt (insert a s) ⟷ x islimpt s
proof
  assume ∗: x islimpt (insert a s)
  show x islimpt s
  proof (rule islimptI)
    fix t
    assume t: x ∈ t open t
    show ∃ y∈s. y ∈ t ∧ y ≠ x
    proof (cases x = a)
      case True
      obtain y where y ∈ insert a s y ∈ t y ≠ x
        using ∗ t by (rule islimptE)
      with ‹x = a› show ?thesis by auto
    next
      case False
      with t have t': x ∈ t − {a} open (t − {a})
        by (simp-all add: open-Diff)
      obtain y where y ∈ insert a s y ∈ t − {a} y ≠ x
        using ∗ t' by (rule islimptE)
      then show ?thesis by auto
    qed
  qed
next
```

assume *x islimpt s*
then show *x islimpt* (*insert a s*)
by (*rule islimpt-subset*) *auto*
qed

lemma *islimpt-finite*:
fixes *x* :: *'a*::*t1-space*
shows *finite s* $\Longrightarrow$ $\neg$ *x islimpt s*
by (*induct set*: *finite*) (*simp-all add*: *islimpt-insert*)

lemma *islimpt-union-finite*:
fixes *x* :: *'a*::*t1-space*
shows *finite s* $\Longrightarrow$ *x islimpt* (*s* $\cup$ *t*) $\longleftrightarrow$ *x islimpt t*
by (*simp add*: *islimpt-Un islimpt-finite*)

lemma *islimpt-eq-acc-point*:
fixes *l* :: *'a* :: *t1-space*
shows *l islimpt S* $\longleftrightarrow$ ($\forall$ *U*. *l*$\in$*U* $\longrightarrow$ *open U* $\longrightarrow$ *infinite* (*U* $\cap$ *S*))
proof (*safe intro*!: *islimptI*)
fix *U*
assume *l islimpt S l* $\in$ *U open U finite* (*U* $\cap$ *S*)
then have *l islimpt S l* $\in$ (*U* $-$ (*U* $\cap$ *S* $-$ {*l*})) *open* (*U* $-$ (*U* $\cap$ *S* $-$ {*l*}))
by (*auto intro*: *finite-imp-closed*)
then show *False*
by (*rule islimptE*) *auto*
next
fix *T*
assume $*$: $\forall$ *U*. *l*$\in$*U* $\longrightarrow$ *open U* $\longrightarrow$ *infinite* (*U* $\cap$ *S*) *l* $\in$ *T open T*
then have *infinite* (*T* $\cap$ *S* $-$ {*l*})
by *auto*
then have $\exists$ *x*. *x* $\in$ (*T* $\cap$ *S* $-$ {*l*})
unfolding *ex-in-conv* **by** (*intro notI*) *simp*
then show $\exists$ *y*$\in$*S*. *y* $\in$ *T* $\wedge$ *y* $\neq$ *l*
by *auto*
qed

lemma *islimpt-range-imp-convergent-subsequence*:
fixes *l* :: *'a* :: {*t1-space*, *first-countable-topology*}
assumes *l*: *l islimpt* (*range f*)
shows $\exists$ *r*. *subseq r* $\wedge$ (*f* $\circ$ *r*) $\longrightarrow$ *l*
using *l* **unfolding** *islimpt-eq-acc-point*
by (*rule acc-point-range-imp-convergent-subsequence*)

lemma *sequence-unique-limpt*:
fixes *f* :: *nat* $\Rightarrow$ *'a*::*t2-space*
assumes (*f* $\longrightarrow$ *l*) *sequentially*
and *l'* *islimpt* (*range f*)
shows *l'* = *l*
proof (*rule ccontr*)

```
  assume l' ≠ l
  obtain s t where open s open t l' ∈ s l ∈ t s ∩ t = {}
    using hausdorff [OF ‹l' ≠ l›] by auto
  have eventually (λn. f n ∈ t) sequentially
    using assms(1) ‹open t› ‹l ∈ t› by (rule topological-tendstoD)
  then obtain N where ∀ n≥N. f n ∈ t
    unfolding eventually-sequentially by auto

  have UNIV = {..<N} ∪ {N..}
    by auto
  then have l' islimpt (f ‘ ({..<N} ∪ {N..}))
    using assms(2) by simp
  then have l' islimpt (f ‘ {..<N} ∪ f ‘ {N..})
    by (simp add: image-Un)
  then have l' islimpt (f ‘ {N..})
    by (simp add: islimpt-union-finite)
  then obtain y where y ∈ f ‘ {N..} y ∈ s y ≠ l'
    using ‹l' ∈ s› ‹open s› by (rule islimptE)
  then obtain n where N ≤ n f n ∈ s f n ≠ l'
    by auto
  with ‹∀ n≥N. f n ∈ t› have f n ∈ s ∩ t
    by simp
  with ‹s ∩ t = {}› show False
    by simp
qed

lemma bolzano-weierstrass-imp-closed:
  fixes s :: 'a::{first-countable-topology,t2-space} set
  assumes ∀ t. infinite t ∧ t ⊆ s --> (∃ x ∈ s. x islimpt t)
  shows closed s
proof -
  {
    fix x l
    assume as: ∀ n::nat. x n ∈ s (x ⟶ l) sequentially
    then have l ∈ s
    proof (cases ∀ n. x n ≠ l)
      case False
      then show l∈s using as(1) by auto
    next
      case True note cas = this
      with as(2) have infinite (range x)
        using sequence-infinite-lemma[of x l] by auto
      then obtain l' where l'∈s l' islimpt (range x)
        using assms[THEN spec[where x=range x]] as(1) by auto
      then show l∈s using sequence-unique-limpt[of x l l']
        using as cas by auto
    qed
  }
  then show ?thesis
```

unfolding *closed-sequential-limits* **by** *fast*
qed

lemma *compact-imp-bounded*:
assumes *compact U*
shows *bounded U*
proof −
have *compact U* ∀ *x*∈*U*. *open* (*ball x 1*) *U* ⊆ (⋃ *x*∈*U*. *ball x 1*)
using *assms* **by** *auto*
then obtain *D* **where** *D*: *D* ⊆ *U finite D U* ⊆ (⋃ *x*∈*D*. *ball x 1*)
by (*rule compactE-image*)
from ‹*finite D*› **have** *bounded* (⋃ *x*∈*D*. *ball x 1*)
by (*simp add*: *bounded-UN*)
then show *bounded U* **using** ‹*U* ⊆ (⋃ *x*∈*D*. *ball x 1*)›
by (*rule bounded-subset*)
qed

In particular, some common special cases.

lemma *compact-union* [*intro*]:
assumes *compact s*
and *compact t*
shows *compact* (*s* ∪ *t*)
proof (*rule compactI*)
fix *f*
assume ∗: *Ball f open s* ∪ *t* ⊆ ⋃ *f*
from ∗ ‹*compact s*› **obtain** *s*′ **where** *s*′ ⊆ *f* ∧ *finite s*′ ∧ *s* ⊆ ⋃ *s*′
unfolding *compact-eq-heine-borel* **by** (*auto elim*!: *allE*[*of* - *f*])
moreover
from ∗ ‹*compact t*› **obtain** *t*′ **where** *t*′ ⊆ *f* ∧ *finite t*′ ∧ *t* ⊆ ⋃ *t*′
unfolding *compact-eq-heine-borel* **by** (*auto elim*!: *allE*[*of* - *f*])
ultimately show ∃*f*′⊆*f*. *finite f*′ ∧ *s* ∪ *t* ⊆ ⋃ *f*′
by (*auto intro*!: *exI*[*of* - *s*′ ∪ *t*′])
qed

lemma *compact-Union* [*intro*]: *finite S* ⟹ (⋀*T*. *T* ∈ *S* ⟹ *compact T*) ⟹ *compact* (⋃ *S*)
by (*induct set*: *finite*) *auto*

lemma *compact-UN* [*intro*]:
finite A ⟹ (⋀*x*. *x* ∈ *A* ⟹ *compact* (*B x*)) ⟹ *compact* (⋃ *x*∈*A*. *B x*)
unfolding *SUP-def* **by** (*rule compact-Union*) *auto*

lemma *closed-inter-compact* [*intro*]:
assumes *closed s*
and *compact t*
shows *compact* (*s* ∩ *t*)
using *compact-inter-closed* [*of t s*] *assms*
by (*simp add*: *Int-commute*)

lemma *compact-inter* [*intro*]:
fixes $s\ t :: 'a :: t2\text{-}space\ set$
assumes *compact s*
and *compact t*
shows *compact* $(s \cap t)$
using *assms* **by** (*intro compact-inter-closed compact-imp-closed*)

lemma *compact-sing* [*simp*]: *compact* $\{a\}$
unfolding *compact-eq-heine-borel* **by** *auto*

lemma *compact-insert* [*simp*]:
assumes *compact s*
shows *compact* (*insert x s*)
proof −
have *compact* $(\{x\} \cup s)$
using *compact-sing assms* **by** (*rule compact-union*)
then show *?thesis* **by** *simp*
qed

lemma *finite-imp-compact*: *finite* $s \Longrightarrow$ *compact s*
by (*induct set*: *finite*) *simp-all*

lemma *open-delete*:
fixes $s :: 'a::t1\text{-}space\ set$
shows *open* $s \Longrightarrow$ *open* $(s - \{x\})$
by (*simp add*: *open-Diff*)

Compactness expressed with filters

lemma *closure-iff-nhds-not-empty*:
$x \in$ *closure* $X \longleftrightarrow (\forall A.\ \forall S \subseteq A.$ *open* $S \longrightarrow x \in S \longrightarrow X \cap A \neq \{\})$
proof *safe*
assume *x*: $x \in$ *closure* X
fix $S\ A$
assume *open* $S\ x \in S\ X \cap A = \{\}\ S \subseteq A$
then have $x \notin$ *closure* $(-S)$
by (*auto simp*: *closure-complement subset-eq*[*symmetric*] *intro*: *interiorI*)
with *x* **have** $x \in$ *closure* $X -$ *closure* $(-S)$
by *auto*
also have $\ldots \subseteq$ *closure* $(X \cap S)$
using ‹*open S*› *open-inter-closure-subset*[*of S X*] **by** (*simp add*: *closed-Compl ac-simps*)
finally have $X \cap S \neq \{\}$ **by** *auto*
then show *False* **using** ‹$X \cap A = \{\}$› ‹$S \subseteq A$› **by** *auto*
next
assume $\forall A\ S.\ S \subseteq A \longrightarrow$ *open* $S \longrightarrow x \in S \longrightarrow X \cap A \neq \{\}$
from *this*[*THEN spec*, *of* $-\ X$, *THEN spec*, *of* $-$ *closure X*]
show $x \in$ *closure X*
by (*simp add*: *closure-subset open-Compl*)
qed

lemma *compact-filter*:
compact U $\longleftrightarrow$ ($\forall F.\ F \neq bot \longrightarrow eventually\ (\lambda x.\ x \in U)\ F \longrightarrow (\exists x{\in}U.\ inf\ (nhds\ x)\ F \neq bot)$)
proof (*intro allI iffI impI compact-fip*[*THEN iffD2*] *notI*)
fix *F*
assume *compact U*
assume *F*: $F \neq bot$ *eventually* ($\lambda x.\ x \in U$) *F*
then have $U \neq \{\}$
by (*auto simp*: *eventually-False*)

def $Z \equiv closure\ `\ \{A.\ eventually\ (\lambda x.\ x \in A)\ F\}$
then have $\forall z{\in}Z.\ closed\ z$
by *auto*
moreover
have *ev-Z*: $\bigwedge z.\ z \in Z \Longrightarrow eventually\ (\lambda x.\ x \in z)\ F$
unfolding *Z-def* **by** (*auto elim*: *eventually-mono intro*: *set-mp*[*OF closure-subset*])
have ($\forall B \subseteq Z.\ finite\ B \longrightarrow U \cap \bigcap B \neq \{\}$)
proof (*intro allI impI*)
fix *B* **assume** *finite B* $B \subseteq Z$
with ‹*finite B*› *ev-Z F*(*2*) **have** *eventually* ($\lambda x.\ x \in U \cap (\bigcap B)$) *F*
by (*auto simp*: *eventually-ball-finite-distrib eventually-conj-iff*)
with *F* **show** $U \cap \bigcap B \neq \{\}$
by (*intro notI*) (*simp add*: *eventually-False*)
qed
ultimately have $U \cap \bigcap Z \neq \{\}$
using ‹*compact U*› **unfolding** *compact-fip* **by** *blast*
then obtain *x* **where** $x \in U$ **and** *x*: $\bigwedge z.\ z \in Z \Longrightarrow x \in z$
by *auto*

have $\bigwedge P.\ eventually\ P\ (inf\ (nhds\ x)\ F) \Longrightarrow P \neq bot$
unfolding *eventually-inf eventually-nhds*
proof *safe*
fix *P Q R S*
assume *eventually R F open S x* $\in$ *S*
with *open-inter-closure-eq-empty*[*of S* $\{x.\ R\ x\}$] *x*[*of closure* $\{x.\ R\ x\}$]
have $S \cap \{x.\ R\ x\} \neq \{\}$ **by** (*auto simp*: *Z-def*)
moreover assume *Ball S Q* $\forall x.\ Q\ x \wedge R\ x \longrightarrow bot\ x$
ultimately show *False* **by** (*auto simp*: *set-eq-iff*)
qed
with ‹$x \in U$› **show** $\exists x{\in}U.\ inf\ (nhds\ x)\ F \neq bot$
by (*metis eventually-bot*)
next
fix *A*
assume *A*: $\forall a{\in}A.\ closed\ a\ \forall B{\subseteq}A.\ finite\ B \longrightarrow U \cap \bigcap B \neq \{\}\ U \cap \bigcap A = \{\}$
def $F \equiv INF\ a{:}insert\ U\ A.\ principal\ a$
have $F \neq bot$
unfolding *F-def*
proof (*rule INF-filter-not-bot*)

fix X **assume** $X \subseteq$ *insert* U A *finite* X
moreover with $A(2)$[*THEN spec*, *of* $X - \{U\}$] **have** $U \cap \bigcap(X - \{U\}) \neq \{\}$
by *auto*
ultimately show (*INF* a:X. *principal* a) $\neq$ *bot*
by (*auto simp add*: *INF-principal-finite principal-eq-bot-iff*)
qed
moreover
have $F \leq$ *principal* U
unfolding *F-def* **by** *auto*
then have *eventually* ($\lambda x.\ x \in U$) F
by (*auto simp*: *le-filter-def eventually-principal*)
moreover
assume $\forall F.\ F \neq$ *bot* $\longrightarrow$ *eventually* ($\lambda x.\ x \in U$) $F \longrightarrow (\exists x{\in}U.$ *inf* (*nhds* x) $F \neq$ *bot*)
ultimately obtain x **where** $x \in U$ **and** x: *inf* (*nhds* x) $F \neq$ *bot*
by *auto*

{ fix V **assume** $V \in A$
then have $F \leq$ *principal* V
unfolding *F-def* **by** (*intro INF-lower2*[*of* V]) *auto*
then have V: *eventually* ($\lambda x.\ x \in V$) F
by (*auto simp*: *le-filter-def eventually-principal*)
have $x \in$ *closure* V
unfolding *closure-iff-nhds-not-empty*
proof (*intro impI allI*)
fix S A
assume *open* S $x \in S$ $S \subseteq A$
then have *eventually* ($\lambda x.\ x \in A$) (*nhds* x)
by (*auto simp*: *eventually-nhds*)
with V **have** *eventually* ($\lambda x.\ x \in V \cap A$) (*inf* (*nhds* x) F)
by (*auto simp*: *eventually-inf*)
with x **show** $V \cap A \neq \{\}$
by (*auto simp del*: *Int-iff simp add*: *trivial-limit-def*)
qed
then have $x \in V$
using ‹$V \in A$› $A(1)$ **by** *simp*
}
with ‹$x{\in}U$› **have** $x \in U \cap \bigcap A$ **by** *auto*
with ‹$U \cap \bigcap A = \{\}$› **show** *False* **by** *auto*
qed

definition *countably-compact* $U \longleftrightarrow$
$(\forall A.$ *countable* $A \longrightarrow (\forall a{\in}A.$ *open* $a) \longrightarrow U \subseteq \bigcup A \longrightarrow (\exists T{\subseteq}A.$ *finite* $T \wedge U \subseteq \bigcup T))$

lemma *countably-compactE*:
assumes *countably-compact* s **and** $\forall t{\in}C.$ *open* t **and** $s \subseteq \bigcup C$ *countable* C
obtains C' **where** $C' \subseteq C$ **and** *finite* C' **and** $s \subseteq \bigcup C'$

using *assms* **unfolding** *countably-compact-def* **by** *metis*

lemma *countably-compactI*:
assumes $\bigwedge C.\ \forall\, t{\in}C.\ \mathit{open}\ t \Longrightarrow s \subseteq \bigcup C \Longrightarrow \mathit{countable}\ C \Longrightarrow (\exists\, C'{\subseteq}C.\ \mathit{finite}\ C' \wedge s \subseteq \bigcup C')$
shows *countably-compact s*
using *assms* **unfolding** *countably-compact-def* **by** *metis*

lemma *compact-imp-countably-compact*: *compact U* $\Longrightarrow$ *countably-compact U*
by (*auto simp*: *compact-eq-heine-borel countably-compact-def*)

lemma *countably-compact-imp-compact*:
assumes *countably-compact U*
and *ccover*: *countable B* $\forall\, b{\in}B.\ \mathit{open}\ b$
and *basis*: $\bigwedge T\ x.\ \mathit{open}\ T \Longrightarrow x \in T \Longrightarrow x \in U \Longrightarrow \exists\, b{\in}B.\ x \in b \wedge b \cap U \subseteq T$
shows *compact U*
using ‹*countably-compact U*›
unfolding *compact-eq-heine-borel countably-compact-def*
proof *safe*
fix *A*
assume *A*: $\forall\, a{\in}A.\ \mathit{open}\ a\ U \subseteq \bigcup A$
assume $*$: $\forall\, A.\ \mathit{countable}\ A \longrightarrow (\forall\, a{\in}A.\ \mathit{open}\ a) \longrightarrow U \subseteq \bigcup A \longrightarrow (\exists\, T{\subseteq}A.\ \mathit{finite}\ T \wedge U \subseteq \bigcup T)$

moreover def $C \equiv \{b{\in}B.\ \exists\, a{\in}A.\ b \cap U \subseteq a\}$
ultimately have *countable C* $\forall\, a{\in}C.\ \mathit{open}\ a$
unfolding *C-def* **using** *ccover* **by** *auto*
moreover
have $\bigcup A \cap U \subseteq \bigcup C$
proof *safe*
fix *x a*
assume $x \in U\ x \in a\ a \in A$
with *basis*[*of a x*] *A* **obtain** *b* **where** $b \in B\ x \in b\ b \cap U \subseteq a$
by *blast*
with ‹$a \in A$› **show** $x \in \bigcup C$
unfolding *C-def* **by** *auto*
qed
then have $U \subseteq \bigcup C$ **using** ‹$U \subseteq \bigcup A$› **by** *auto*
ultimately obtain *T* **where** *T*: $T{\subseteq}C$ *finite* $T\ U \subseteq \bigcup T$
using $*$ **by** *metis*
then have $\forall\, t{\in}T.\ \exists\, a{\in}A.\ t \cap U \subseteq a$
by (*auto simp*: *C-def*)
then obtain *f* **where** $\forall\, t{\in}T.\ f\ t \in A \wedge t \cap U \subseteq f\ t$
unfolding *bchoice-iff Bex-def* **..**
with *T* **show** $\exists\, T{\subseteq}A.\ \mathit{finite}\ T \wedge U \subseteq \bigcup T$
unfolding *C-def* **by** (*intro exI*[*of - f*‘*T*]) *fastforce*
qed

lemma *countably-compact-imp-compact-second-countable*:
countably-compact U $\Longrightarrow$ *compact* (*U* :: *'a* :: *second-countable-topology set*)
proof (*rule countably-compact-imp-compact*)
fix *T* **and** *x* :: *'a*
assume *open T x* $\in$ *T*
from *topological-basisE*[*OF is-basis this*] **obtain** *b* **where**
b $\in$ (*SOME B*. *countable B* $\wedge$ *topological-basis B*) *x* $\in$ *b b* $\subseteq$ *T* **.**
then show $\exists$ *b*$\in$*SOME B*. *countable B* $\wedge$ *topological-basis B*. *x* $\in$ *b* $\wedge$ *b* $\cap$ *U* $\subseteq$ *T*
by *blast*
qed (*insert countable-basis topological-basis-open*[*OF is-basis*], *auto*)

lemma *countably-compact-eq-compact*:
countably-compact U $\longleftrightarrow$ *compact* (*U* :: *'a* :: *second-countable-topology set*)
using *countably-compact-imp-compact-second-countable compact-imp-countably-compact*
by *blast*

16.19.2 Sequential compactness

definition *seq-compact* :: *'a*::*topological-space set* $\Rightarrow$ *bool*
where *seq-compact S* $\longleftrightarrow$
($\forall$ *f*. ($\forall$ *n*. *f n* $\in$ *S*) $\longrightarrow$ ($\exists$ *l*$\in$*S*. $\exists$ *r*. *subseq r* $\wedge$ ((*f* $\circ$ *r*) $\longrightarrow$ *l*) *sequentially*))

lemma *seq-compactI*:
assumes $\bigwedge$*f*. $\forall$ *n*. *f n* $\in$ *S* $\Longrightarrow$ $\exists$ *l*$\in$*S*. $\exists$ *r*. *subseq r* $\wedge$ ((*f* $\circ$ *r*) $\longrightarrow$ *l*) *sequentially*
shows *seq-compact S*
unfolding *seq-compact-def* **using** *assms* **by** *fast*

lemma *seq-compactE*:
assumes *seq-compact S* $\forall$ *n*. *f n* $\in$ *S*
obtains *l r* **where** *l* $\in$ *S subseq r* ((*f* $\circ$ *r*) $\longrightarrow$ *l*) *sequentially*
using *assms* **unfolding** *seq-compact-def* **by** *fast*

lemma *closed-sequentially*:
assumes *closed s* **and** $\forall$ *n*. *f n* $\in$ *s* **and** *f* $\longrightarrow$ *l*
shows *l* $\in$ *s*
proof (*rule ccontr*)
assume *l* $\notin$ *s*
with ‹*closed s*› **and** ‹*f* $\longrightarrow$ *l*› **have** *eventually* (λ*n*. *f n* $\in$ $-$ *s*) *sequentially*
by (*fast intro*: *topological-tendstoD*)
with ‹$\forall$ *n*. *f n* $\in$ *s*› **show** *False*
by *simp*
qed

lemma *seq-compact-inter-closed*:
assumes *seq-compact s* **and** *closed t*
shows *seq-compact* (*s* $\cap$ *t*)
proof (*rule seq-compactI*)
fix *f* **assume** $\forall$ *n*::*nat*. *f n* $\in$ *s* $\cap$ *t*

hence $\forall n.\ f\ n \in s$ **and** $\forall n.\ f\ n \in t$
by *simp-all*
from ‹*seq-compact s*› **and** ‹$\forall n.\ f\ n \in s$›
obtain $l\ r$ **where** $l \in s$ **and** r: *subseq r* **and** l: $(f \circ r) \longrightarrow l$
by (*rule seq-compactE*)
from ‹$\forall n.\ f\ n \in t$› **have** $\forall n.\ (f \circ r)\ n \in t$
by *simp*
from ‹*closed t*› **and** *this* **and** l **have** $l \in t$
by (*rule closed-sequentially*)
with ‹$l \in s$› **and** r **and** l **show** $\exists l{\in}s \cap t.\ \exists r.\ subseq\ r \wedge (f \circ r) \longrightarrow l$
by *fast*
qed

lemma *seq-compact-closed-subset*:
assumes *closed s* **and** $s \subseteq t$ **and** *seq-compact t*
shows *seq-compact s*
using *assms seq-compact-inter-closed* [*of t s*] **by** (*simp add*: *Int-absorb1*)

lemma *seq-compact-imp-countably-compact*:
fixes U :: *'a* :: *first-countable-topology set*
assumes *seq-compact U*
shows *countably-compact U*
proof (*safe intro*!: *countably-compactI*)
fix *A*
assume *A*: $\forall a{\in}A.$ *open a* $U \subseteq \bigcup A$ *countable A*
have *subseq*: $\bigwedge X.$ *range* $X \subseteq U \Longrightarrow \exists r\ x.\ x \in U \wedge subseq\ r \wedge (X \circ r) \longrightarrow x$
using ‹*seq-compact U*› **by** (*fastforce simp*: *seq-compact-def subset-eq*)
show $\exists T{\subseteq}A.$ *finite* $T \wedge U \subseteq \bigcup T$
proof *cases*
assume *finite A*
with *A* **show** *?thesis* **by** *auto*
next
assume *infinite A*
then have $A \neq \{\}$ **by** *auto*
show *?thesis*
proof (*rule ccontr*)
assume $\neg\ (\exists T{\subseteq}A.$ *finite* $T \wedge U \subseteq \bigcup T)$
then have $\forall T.\ \exists x.\ T \subseteq A \wedge$ *finite* $T \longrightarrow (x \in U - \bigcup T)$
by *auto*
then obtain X' **where** T: $\bigwedge T.\ T \subseteq A \Longrightarrow$ *finite* $T \Longrightarrow X'\ T \in U - \bigcup T$
by *metis*
def $X \equiv \lambda n.\ X'$ (*from-nat-into A* ‘ $\{..\ n\}$)
have X: $\bigwedge n.\ X\ n \in U - (\bigcup i{\leq}n.$ *from-nat-into A i*)
using ‹$A \neq \{\}$› **unfolding** *X-def SUP-def* **by** (*intro T*) (*auto intro*: *from-nat-into*)
then have *range* $X \subseteq U$
by *auto*
with *subseq*[*of X*] **obtain** $r\ x$ **where** $x \in U$ **and** r: *subseq r* $(X \circ r) \longrightarrow$

x
by *auto*
from ‹*x*∈*U*› ‹*U* ⊆ ⋃ *A*› *from-nat-into-surj*[*OF* ‹*countable A*›]
obtain *n* **where** *x* ∈ *from-nat-into A n* **by** *auto*
with *r*(*2*) *A*(*1*) *from-nat-into*[*OF* ‹*A* ≠ {}›, *of n*]
have *eventually* (λ*i*. *X* (*r i*) ∈ *from-nat-into A n*) *sequentially*
unfolding *tendsto-def* **by** (*auto simp*: *comp-def*)
then obtain *N* **where** ⋀*i*. *N* ≤ *i* ⟹ *X* (*r i*) ∈ *from-nat-into A n*
by (*auto simp*: *eventually-sequentially*)
moreover from *X* **have** ⋀*i*. *n* ≤ *r i* ⟹ *X* (*r i*) ∉ *from-nat-into A n*
by *auto*
moreover from ‹*subseq r*›[*THEN seq-suble*, *of max n N*] **have** ∃ *i*. *n* ≤ *r i* ∧ *N* ≤ *i*
by (*auto intro*!: *exI*[*of - max n N*])
ultimately show *False*
by *auto*
qed
qed
qed

lemma *compact-imp-seq-compact*:
fixes *U* :: *'a* :: *first-countable-topology set*
assumes *compact U*
shows *seq-compact U*
unfolding *seq-compact-def*
proof *safe*
fix *X* :: *nat* ⇒ *'a*
assume ∀ *n*. *X n* ∈ *U*
then have *eventually* (λ*x*. *x* ∈ *U*) (*filtermap X sequentially*)
by (*auto simp*: *eventually-filtermap*)
moreover
have *filtermap X sequentially* ≠ *bot*
by (*simp add*: *trivial-limit-def eventually-filtermap*)
ultimately
obtain *x* **where** *x* ∈ *U* **and** *x*: *inf* (*nhds x*) (*filtermap X sequentially*) ≠ *bot* (**is** *?F* ≠ -)
using ‹*compact U*› **by** (*auto simp*: *compact-filter*)

from *countable-basis-at-decseq*[*of x*]
obtain *A* **where** *A*:
⋀*i*. *open* (*A i*)
⋀*i*. *x* ∈ *A i*
⋀*S*. *open S* ⟹ *x* ∈ *S* ⟹ *eventually* (λ*i*. *A i* ⊆ *S*) *sequentially*
by *blast*
def *s* ≡ λ*n i*. *SOME j*. *i* < *j* ∧ *X j* ∈ *A* (*Suc n*)
{
fix *n i*
have ∃ *a*. *i* < *a* ∧ *X a* ∈ *A* (*Suc n*)
proof (*rule ccontr*)

```
    assume ¬ (∃ a>i. X a ∈ A (Suc n))
    then have ⋀a. Suc i ≤ a ⟹ X a ∉ A (Suc n)
      by auto
    then have eventually (λx. x ∉ A (Suc n)) (filtermap X sequentially)
      by (auto simp: eventually-filtermap eventually-sequentially)
    moreover have eventually (λx. x ∈ A (Suc n)) (nhds x)
      using A(1,2)[of Suc n] by (auto simp: eventually-nhds)
    ultimately have eventually (λx. False) ?F
      by (auto simp add: eventually-inf)
    with x show False
      by (simp add: eventually-False)
  qed
  then have i < s n i X (s n i) ∈ A (Suc n)
    unfolding s-def by (auto intro: someI2-ex)
 }
 note s = this
 def r ≡ rec-nat (s 0 0) s
 have subseq r
  by (auto simp: r-def s subseq-Suc-iff)
 moreover
 have (λn. X (r n)) ⟶ x
 proof (rule topological-tendstoI)
  fix S
  assume open S x ∈ S
  with A(3) have eventually (λi. A i ⊆ S) sequentially
    by auto
  moreover
  {
   fix i
   assume Suc 0 ≤ i
   then have X (r i) ∈ A i
    by (cases i) (simp-all add: r-def s)
  }
  then have eventually (λi. X (r i) ∈ A i) sequentially
   by (auto simp: eventually-sequentially)
  ultimately show eventually (λi. X (r i) ∈ S) sequentially
   by eventually-elim auto
 qed
 ultimately show ∃x ∈ U. ∃r. subseq r ∧ (X ∘ r) ⟶ x
  using ‹x ∈ U› by (auto simp: convergent-def comp-def)
qed

lemma countably-compact-imp-acc-point:
 assumes countably-compact s
  and countable t
  and infinite t
  and t ⊆ s
 shows ∃x∈s. ∀ U. x∈U ∧ open U ⟶ infinite (U ∩ t)
proof (rule ccontr)
```

def $C \equiv (\lambda F.\ \mathit{interior}\ (F \cup (-\ t)))\ `\ \{F.\ \mathit{finite}\ F \wedge F \subseteq t\ \}$
note ‹*countably-compact s*›
moreover have $\forall\, t{\in}C.\ \mathit{open}\ t$
by (*auto simp*: *C-def*)
moreover
assume $\neg\ (\exists\, x{\in}s.\ \forall\, U.\ x{\in}U \wedge \mathit{open}\ U \longrightarrow \mathit{infinite}\ (U \cap t))$
then have *s*: $\bigwedge x.\ x \in s \Longrightarrow \exists\, U.\ x{\in}U \wedge \mathit{open}\ U \wedge \mathit{finite}\ (U \cap t)$ **by** *metis*
have $s \subseteq \bigcup C$
using ‹$t \subseteq s$›
unfolding *C-def Union-image-eq*
apply (*safe dest*!: *s*)
apply (*rule-tac a*=$U \cap t$ **in** *UN-I*)
apply (*auto intro*!: *interiorI simp add*: *finite-subset*)
done
moreover
from ‹*countable t*› **have** *countable C*
unfolding *C-def* **by** (*auto intro*: *countable-Collect-finite-subset*)
ultimately
obtain *D* **where** $D \subseteq C$ *finite D* $s \subseteq \bigcup D$
by (*rule countably-compactE*)
then obtain *E* **where** *E*: $E \subseteq \{F.\ \mathit{finite}\ F \wedge F \subseteq t\ \}$ *finite E*
and *s*: $s \subseteq (\bigcup F{\in}E.\ \mathit{interior}\ (F \cup (-\ t)))$
by (*metis* (*lifting*) *Union-image-eq finite-subset-image C-def*)
from *s* ‹$t \subseteq s$› **have** $t \subseteq \bigcup E$
using *interior-subset* **by** *blast*
moreover have *finite* $(\bigcup E)$
using *E* **by** *auto*
ultimately show *False* **using** ‹*infinite t*›
by (*auto simp*: *finite-subset*)
qed

lemma *countable-acc-point-imp-seq-compact*:
fixes *s* :: *'a*::*first-countable-topology set*
assumes $\forall\, t.\ \mathit{infinite}\ t \wedge \mathit{countable}\ t \wedge t \subseteq s \longrightarrow$
$(\exists\, x{\in}s.\ \forall\, U.\ x{\in}U \wedge \mathit{open}\ U \longrightarrow \mathit{infinite}\ (U \cap t))$
shows *seq-compact s*
proof −
{
fix *f* :: *nat* ⇒ *'a*
assume *f*: $\forall\, n.\ f\ n \in s$
have $\exists\, l{\in}s.\ \exists\, r.\ \mathit{subseq}\ r \wedge ((f \circ r) \longrightarrow l)\ \mathit{sequentially}$
proof (*cases finite* (*range f*))
case *True*
obtain *l* **where** *infinite* $\{n.\ f\ n = f\ l\}$
using *pigeonhole-infinite*[*OF - True*] **by** *auto*
then obtain *r* **where** *subseq r* **and** *fr*: $\forall\, n.\ f\ (r\ n) = f\ l$
using *infinite-enumerate* **by** *blast*
then have *subseq r* $\wedge$ $(f \circ r) \longrightarrow f\ l$
by (*simp add*: *fr o-def*)

with *f* **show** $\exists\, l{\in}s.\ \exists\, r.\ subseq\ r \wedge (f \circ r) \longrightarrow l$
by *auto*
next
case *False*
with *f assms* **have** $\exists\, x{\in}s.\ \forall\, U.\ x{\in}U \wedge open\ U \longrightarrow infinite\ (U \cap range\ f)$
by *auto*
then obtain *l* **where** $l \in s\ \forall\, U.\ l{\in}U \wedge open\ U \longrightarrow infinite\ (U \cap range\ f)$
..
from *this*(*2*) **have** $\exists\, r.\ subseq\ r \wedge ((f \circ r) \longrightarrow l)\ sequentially$
using *acc-point-range-imp-convergent-subsequence*[*of l f*] **by** *auto*
with ‹$l \in s$› **show** $\exists\, l{\in}s.\ \exists\, r.\ subseq\ r \wedge ((f \circ r) \longrightarrow l)\ sequentially$ **..**
qed
}
then show *?thesis*
unfolding *seq-compact-def* **by** *auto*
qed

lemma *seq-compact-eq-countably-compact*:
fixes *U* :: *'a* :: *first-countable-topology set*
shows *seq-compact U* $\longleftrightarrow$ *countably-compact U*
using
countable-acc-point-imp-seq-compact
countably-compact-imp-acc-point
seq-compact-imp-countably-compact
by *metis*

lemma *seq-compact-eq-acc-point*:
fixes *s* :: *'a* :: *first-countable-topology set*
shows *seq-compact s* $\longleftrightarrow$
$(\forall\, t.\ infinite\ t \wedge countable\ t \wedge t \subseteq s \longrightarrow (\exists\, x{\in}s.\ \forall\, U.\ x{\in}U \wedge open\ U \longrightarrow infinite\ (U \cap t)))$
using
countable-acc-point-imp-seq-compact[*of s*]
countably-compact-imp-acc-point[*of s*]
seq-compact-imp-countably-compact[*of s*]
by *metis*

lemma *seq-compact-eq-compact*:
fixes *U* :: *'a* :: *second-countable-topology set*
shows *seq-compact U* $\longleftrightarrow$ *compact U*
using *seq-compact-eq-countably-compact countably-compact-eq-compact* **by** *blast*

lemma *bolzano-weierstrass-imp-seq-compact*:
fixes *s* :: *'a*::{*t1-space*, *first-countable-topology*} *set*
shows $\forall\, t.\ infinite\ t \wedge t \subseteq s \longrightarrow (\exists\, x \in s.\ x\ islimpt\ t) \Longrightarrow seq\text{-}compact\ s$
by (*rule countable-acc-point-imp-seq-compact*) (*metis islimpt-eq-acc-point*)

16.19.3 Totally bounded

lemma *cauchy-def*: *Cauchy s* $\longleftrightarrow$ $(\forall e{>}0.\ \exists N.\ \forall m\ n.\ m \geq N \wedge n \geq N \longrightarrow$ *dist*$(s\ m)(s\ n) < e)$
unfolding *Cauchy-def* **by** *metis*

lemma *seq-compact-imp-totally-bounded*:
assumes *seq-compact s*
shows $\forall e{>}0.\ \exists k.$ *finite* $k \wedge k \subseteq s \wedge s \subseteq (\bigcup x{\in}k.$ *ball* $x\ e)$
proof −
{ fix *e*::*real* **assume** $e > 0$ **assume** $*$: $\bigwedge k.$ *finite* $k \Longrightarrow k \subseteq s \Longrightarrow \neg\ s \subseteq (\bigcup x{\in}k.$ *ball* $x\ e)$
let *?Q* $= \lambda x\ n\ r.\ r \in s \wedge (\forall m < (n{::}nat).\ \neg\ ($*dist* $(x\ m)\ r < e))$
have $\exists x.\ \forall n{::}nat.$ *?Q* $x\ n\ (x\ n)$
proof (*rule dependent-wellorder-choice*)
fix $n\ x$ **assume** $\bigwedge y.\ y < n \Longrightarrow$ *?Q* $x\ y\ (x\ y)$
then have $\neg\ s \subseteq (\bigcup x{\in}x\ `\ \{0..{<}n\}.$ *ball* $x\ e)$
using $*[$*of* $x\ `\ \{0\ ..{<}\ n\}]$ **by** (*auto simp*: *subset-eq*)
then obtain z **where** $z{:}z{\in}s\ z \notin (\bigcup x{\in}x\ `\ \{0..{<}n\}.$ *ball* $x\ e)$
unfolding *subset-eq* **by** *auto*
show $\exists r.$ *?Q* $x\ n\ r$
using z **by** *auto*
qed *simp*
then obtain x **where** $\forall n{::}nat.\ x\ n \in s$ **and** $x{:}\bigwedge n\ m.\ m < n \Longrightarrow \neg\ ($*dist* $(x\ m)\ (x\ n) < e)$
by *blast*
then obtain $l\ r$ **where** $l \in s$ **and** $r{:}$*subseq* r **and** $((x \circ r) \longrightarrow l)$ *sequentially*
using *assms* **by** (*metis seq-compact-def*)
from *this*(*3*) **have** *Cauchy* $(x \circ r)$
using *LIMSEQ-imp-Cauchy* **by** *auto*
then obtain $N{::}nat$ **where** $\bigwedge m\ n.\ N \leq m \Longrightarrow N \leq n \Longrightarrow$ *dist* $((x \circ r)\ m)\ ((x \circ r)\ n) < e$
unfolding *cauchy-def* **using** ‹$e > 0$› **by** *blast*
then have *False*
using $x[$*of* $r\ N\ r\ (N{+}1)]\ r$ **by** (*auto simp*: *subseq-def*) **}**
then show *?thesis*
by *metis*
qed

16.19.4 Heine-Borel theorem

lemma *seq-compact-imp-heine-borel*:
fixes $s :: {'a} ::$ *metric-space set*
assumes *seq-compact s*
shows *compact s*
proof −
from *seq-compact-imp-totally-bounded*[*OF* ‹*seq-compact s*›]
obtain f **where** $f{:}\ \forall e{>}0.$ *finite* $(f\ e) \wedge f\ e \subseteq s \wedge s \subseteq (\bigcup x{\in}f\ e.$ *ball* $x\ e)$
unfolding *choice-iff'* **..**
def $K \equiv (\lambda(x,\ r).$ *ball* $x\ r)\ `\ ((\bigcup e \in \mathbb{Q} \cap \{0\ <..\}.\ f\ e) \times \mathbb{Q})$

```
  have countably-compact s
    using ‹seq-compact s› by (rule seq-compact-imp-countably-compact)
  then show compact s
  proof (rule countably-compact-imp-compact)
    show countable K
      unfolding K-def using f
      by (auto intro: countable-finite countable-subset countable-rat
            intro!: countable-image countable-SIGMA countable-UN)
    show ∀ b∈K. open b by (auto simp: K-def)
  next
    fix T x
    assume T: open T x ∈ T and x: x ∈ s
    from openE[OF T] obtain e where 0 < e ball x e ⊆ T
      by auto
    then have 0 < e / 2 ball x (e / 2) ⊆ T
      by auto
    from Rats-dense-in-real[OF ‹0 < e / 2›] obtain r where r ∈ ℚ 0 < r r < e
/ 2
      by auto
    from f[rule-format, of r] ‹0 < r› ‹x ∈ s› obtain k where k ∈ f r x ∈ ball k r
      unfolding Union-image-eq by auto
    from ‹r ∈ ℚ› ‹0 < r› ‹k ∈ f r› have ball k r ∈ K
      by (auto simp: K-def)
    then show ∃ b∈K. x ∈ b ∧ b ∩ s ⊆ T
    proof (rule bexI[rotated], safe)
      fix y
      assume y ∈ ball k r
      with ‹r < e / 2› ‹x ∈ ball k r› have dist x y < e
        by (intro dist-double[where x = k and d=e]) (auto simp: dist-commute)
      with ‹ball x e ⊆ T› show y ∈ T
        by auto
    next
      show x ∈ ball k r by fact
    qed
  qed
qed

lemma compact-eq-seq-compact-metric:
  compact (s :: 'a::metric-space set) ⟷ seq-compact s
  using compact-imp-seq-compact seq-compact-imp-heine-borel by blast

lemma compact-def:
  compact (S :: 'a::metric-space set) ⟷
  (∀ f. (∀ n. f n ∈ S) ⟶ (∃ l∈S. ∃ r. subseq r ∧ (f ∘ r) ⟶⟶ l))
  unfolding compact-eq-seq-compact-metric seq-compact-def by auto
```

16.19.5 Complete the chain of compactness variants

```
lemma compact-eq-bolzano-weierstrass:
```

fixes s :: $'a$::*metric-space set*
shows *compact* $s \longleftrightarrow (\forall t.\ \textit{infinite}\ t \wedge t \subseteq s \longrightarrow (\exists x \in s.\ x\ \textit{islimpt}\ t))$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
using *heine-borel-imp-bolzano-weierstrass*[*of s*] **by** *auto*
next
assume *?rhs*
then show *?lhs*
unfolding *compact-eq-seq-compact-metric* **by** (*rule bolzano-weierstrass-imp-seq-compact*)
qed

lemma *bolzano-weierstrass-imp-bounded*:
$\forall t.\ \textit{infinite}\ t \wedge t \subseteq s \longrightarrow (\exists x \in s.\ x\ \textit{islimpt}\ t) \Longrightarrow \textit{bounded}\ s$
using *compact-imp-bounded* **unfolding** *compact-eq-bolzano-weierstrass* **.**

16.20 Metric spaces with the Heine-Borel property

A metric space (or topological vector space) is said to have the Heine-Borel property if every closed and bounded subset is compact.

class *heine-borel* = *metric-space* +
assumes *bounded-imp-convergent-subsequence*:
bounded (*range f*) $\Longrightarrow \exists l\ r.$ *subseq* $r \wedge ((f \circ r) \longrightarrow l)$ *sequentially*

lemma *bounded-closed-imp-seq-compact*:
fixes *s*::$'a$::*heine-borel set*
assumes *bounded s*
and *closed s*
shows *seq-compact s*
proof (*unfold seq-compact-def*, *clarify*)
fix *f* :: *nat* $\Rightarrow$ $'a$
assume *f*: $\forall n.\ f\ n \in s$
with ‹*bounded s*› **have** *bounded* (*range f*)
by (*auto intro*: *bounded-subset*)
obtain *l r* **where** *r*: *subseq r* **and** *l*: $((f \circ r) \longrightarrow l)$ *sequentially*
using *bounded-imp-convergent-subsequence* [*OF* ‹*bounded* (*range f*)›] **by** *auto*
from *f* **have** *fr*: $\forall n.\ (f \circ r)\ n \in s$
by *simp*
have $l \in s$ **using** ‹*closed s*› *fr l*
by (*rule closed-sequentially*)
show $\exists l{\in}s.\ \exists r.$ *subseq* $r \wedge ((f \circ r) \longrightarrow l)$ *sequentially*
using ‹$l \in s$› *r l* **by** *blast*
qed

lemma *compact-eq-bounded-closed*:
fixes *s* :: $'a$::*heine-borel set*
shows *compact s* $\longleftrightarrow$ *bounded s* $\wedge$ *closed s*
(**is** *?lhs* = *?rhs*)

proof
assume *?lhs*
then show *?rhs*
using *compact-imp-closed compact-imp-bounded*
by *blast*
next
assume *?rhs*
then show *?lhs*
using *bounded-closed-imp-seq-compact*[*of s*]
unfolding *compact-eq-seq-compact-metric*
by *auto*
qed

lemma *compact-components*:
fixes *s* :: *'a*::*heine-borel set*
shows ⟦*compact s*; *c* ∈ *components s*⟧ ⟹ *compact c*
by (*meson bounded-subset closed-components in-components-subset compact-eq-bounded-closed*)

lemma *bounded-increasing-convergent*:
fixes *s* :: *nat* ⇒ *real*
shows *bounded* {*s n*| *n*. *True*} ⟹ ∀ *n*. *s n* ≤ *s* (*Suc n*) ⟹ ∃ *l*. *s* ⟶ *l*
using *Bseq-mono-convergent*[*of s*] *incseq-Suc-iff*[*of s*]
by (*auto simp*: *image-def Bseq-eq-bounded convergent-def incseq-def*)

instance *real* :: *heine-borel*
proof
fix *f* :: *nat* ⇒ *real*
assume *f*: *bounded* (*range f*)
obtain *r* **where** *r*: *subseq r monoseq* (*f* ∘ *r*)
unfolding *comp-def* **by** (*metis seq-monosub*)
then have *Bseq* (*f* ∘ *r*)
unfolding *Bseq-eq-bounded* **using** *f* **by** (*force intro*: *bounded-subset*)
with *r* **show** ∃ *l r*. *subseq r* ∧ (*f* ∘ *r*) ⟶ *l*
using *Bseq-monoseq-convergent*[*of f* ∘ *r*] **by** (*auto simp*: *convergent-def*)
qed

lemma *compact-lemma-general*:
fixes *f* :: *nat* ⇒ *'a*
fixes *proj*::*'a* ⇒ *'b* ⇒ *'c*::*heine-borel* (**infixl** *proj 60*)
fixes *unproj*:: (*'b* ⇒ *'c*) ⇒ *'a*
assumes *finite-basis*: *finite basis*
assumes *bounded-proj*: ⋀*k*. *k* ∈ *basis* ⟹ *bounded* ((λ*x*. *x proj k*) ‘ *range f*)
assumes *proj-unproj*: ⋀*e k*. *k* ∈ *basis* ⟹ (*unproj e*) *proj k* = *e k*
assumes *unproj-proj*: ⋀*x*. *unproj* (λ*k*. *x proj k*) = *x*
shows ∀ *d*⊆*basis*. ∃ *l*::*'a*. ∃ *r*.
subseq r ∧ (∀ *e*>*0*. *eventually* (λ*n*. ∀ *i*∈*d*. *dist* (*f* (*r n*) *proj i*) (*l proj i*) < *e*)
sequentially)
proof *safe*

fix $d :: 'b\ set$
assume d: $d \subseteq basis$
with *finite-basis* **have** *finite d*
by (*blast intro*: *finite-subset*)
from *this d* **show** $\exists\, l::'a.\ \exists\, r.\ subseq\ r \wedge$
$(\forall\, e{>}0.\ eventually\ (\lambda n.\ \forall\, i{\in}d.\ dist\ (f\ (r\ n)\ proj\ i)\ (l\ proj\ i) < e)\ sequentially)$
proof (*induct d*)
case *empty*
then show *?case*
unfolding *subseq-def* **by** *auto*
next
case (*insert k d*)
have *k*[*intro*]: $k \in basis$
using *insert* **by** *auto*
have *s'*: *bounded* ((λ*x*. *x proj k*) ‘ *range f*)
using *k*
by (*rule bounded-proj*)
obtain *l1*::'*a* **and** *r1* **where** *r1*: *subseq r1*
and *lr1*: ∀ *e* > *0*. *eventually* (λ*n*. ∀ *i*∈*d*. *dist* (*f* (*r1 n*) *proj i*) (*l1 proj i*) < *e*) *sequentially*
using *insert*(*3*) **using** *insert*(*4*) **by** *auto*
have *f'*: ∀ *n*. *f* (*r1 n*) *proj k* ∈ (λ*x*. *x proj k*) ‘ *range f*
by *simp*
have *bounded* (*range* (λ*i*. *f* (*r1 i*) *proj k*))
by (*metis* (*lifting*) *bounded-subset f' image-subsetI s'*)
then obtain *l2 r2* **where** *r2*:*subseq r2* **and** *lr2*:((λ*i*. *f* (*r1* (*r2 i*)) *proj k*) ⟶ *l2*) *sequentially*
using *bounded-imp-convergent-subsequence*[*of* λ*i*. *f* (*r1 i*) *proj k*]
by (*auto simp*: *o-def*)
def *r* ≡ *r1* ∘ *r2*
have *r*:*subseq r*
using *r1* **and** *r2* **unfolding** *r-def o-def subseq-def* **by** *auto*
moreover
def *l* ≡ *unproj* (λ*i*. *if i* = *k then l2 else l1 proj i*)::'*a*
{
fix *e*::*real*
assume *e* > *0*
from *lr1* ‹*e* > *0*› **have** *N1*: *eventually* (λ*n*. ∀ *i*∈*d*. *dist* (*f* (*r1 n*) *proj i*) (*l1 proj i*) < *e*) *sequentially*
by *blast*
from *lr2* ‹*e* > *0*› **have** *N2*:*eventually* (λ*n*. *dist* (*f* (*r1* (*r2 n*)) *proj k*) *l2* < *e*) *sequentially*
by (*rule tendstoD*)
from *r2 N1* **have** *N1'*: *eventually* (λ*n*. ∀ *i*∈*d*. *dist* (*f* (*r1* (*r2 n*)) *proj i*) (*l1 proj i*) < *e*) *sequentially*
by (*rule eventually-subseq*)
have *eventually* (λ*n*. ∀ *i*∈(*insert k d*). *dist* (*f* (*r n*) *proj i*) (*l proj i*) < *e*) *sequentially*
using *N1' N2*

```
    by eventually-elim (insert insert.prems, auto simp: l-def r-def o-def proj-unproj)
   }
   ultimately show ?case by auto
  qed
qed

lemma compact-lemma:
  fixes f :: nat ⇒ 'a::euclidean-space
  assumes bounded (range f)
  shows ∀ d⊆Basis. ∃ l::'a. ∃ r.
     subseq r ∧ (∀ e>0. eventually (λn. ∀ i∈d. dist (f (r n) • i) (l • i) < e)
sequentially)
  by (rule compact-lemma-general[where unproj=λe. ∑ i∈Basis. e i *R i])
     (auto intro!: assms bounded-linear-inner-left bounded-linear-image
       simp: euclidean-representation)

instance euclidean-space ⊆ heine-borel
proof
  fix f :: nat ⇒ 'a
  assume f: bounded (range f)
  then obtain l::'a and r where r: subseq r
   and l: ∀ e>0. eventually (λn. ∀ i∈Basis. dist (f (r n) • i) (l • i) < e) sequentially
    using compact-lemma [OF f] by blast
  {
   fix e::real
   assume e > 0
   hence e / real-of-nat DIM('a) > 0 by (simp add: DIM-positive)
  with l have eventually (λn. ∀ i∈Basis. dist (f (r n) • i) (l • i) < e / (real-of-nat
DIM('a))) sequentially
     by simp
   moreover
   {
    fix n
    assume n: ∀ i∈Basis. dist (f (r n) • i) (l • i) < e / (real-of-nat DIM('a))
    have dist (f (r n)) l ≤ (∑ i∈Basis. dist (f (r n) • i) (l • i))
     apply (subst euclidean-dist-l2)
     using zero-le-dist
     apply (rule setL2-le-setsum)
     done
    also have ... < (∑ i∈(Basis::'a set). e / (real-of-nat DIM('a)))
     apply (rule setsum-strict-mono)
     using n
     apply auto
     done
    finally have dist (f (r n)) l < e
     by auto
   }
   ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
    by (rule eventually-mono)
```

```
  }
  then have ∗: ((f ∘ r) ⟶ l) sequentially
    unfolding o-def tendsto-iff by simp
  with r show ∃ l r. subseq r ∧ ((f ∘ r) ⟶ l) sequentially
    by auto
qed

lemma bounded-fst: bounded s ⟹ bounded (fst ‘ s)
  unfolding bounded-def
  by (metis (erased, hide-lams) dist-fst-le image-iff order-trans)

lemma bounded-snd: bounded s ⟹ bounded (snd ‘ s)
  unfolding bounded-def
  by (metis (no-types, hide-lams) dist-snd-le image-iff order.trans)

instance prod :: (heine-borel, heine-borel) heine-borel
proof
  fix f :: nat ⇒ 'a × 'b
  assume f: bounded (range f)
  then have bounded (fst ‘ range f)
    by (rule bounded-fst)
  then have s1: bounded (range (fst ∘ f))
    by (simp add: image-comp)
  obtain l1 r1 where r1: subseq r1 and l1: (λn. fst (f (r1 n))) ⟶ l1
    using bounded-imp-convergent-subsequence [OF s1] unfolding o-def by fast
  from f have s2: bounded (range (snd ∘ f ∘ r1))
    by (auto simp add: image-comp intro: bounded-snd bounded-subset)
  obtain l2 r2 where r2: subseq r2 and l2: ((λn. snd (f (r1 (r2 n)))) ⟶ l2)
sequentially
    using bounded-imp-convergent-subsequence [OF s2]
    unfolding o-def by fast
  have l1': ((λn. fst (f (r1 (r2 n))))) ⟶ l1) sequentially
    using LIMSEQ-subseq-LIMSEQ [OF l1 r2] unfolding o-def .
  have l: ((f ∘ (r1 ∘ r2)) ⟶ (l1, l2)) sequentially
    using tendsto-Pair [OF l1' l2] unfolding o-def by simp
  have r: subseq (r1 ∘ r2)
    using r1 r2 unfolding subseq-def by simp
  show ∃ l r. subseq r ∧ ((f ∘ r) ⟶ l) sequentially
    using l r by fast
qed
```

16.20.1 Completeness

```
lemma (in metric-space) completeI:
  assumes ⋀f. ∀ n. f n ∈ s ⟹ Cauchy f ⟹ ∃ l∈s. f ⟶ l
  shows complete s
  using assms unfolding complete-def by fast

lemma (in metric-space) completeE:
```

assumes *complete s* **and** $\forall n.\ f\ n \in s$ **and** *Cauchy f*
obtains l **where** $l \in s$ **and** $f \longrightarrow l$
using *assms* **unfolding** *complete-def* **by** *fast*

lemma *compact-imp-complete*:
fixes $s :: 'a$::*metric-space set*
assumes *compact s*
shows *complete s*
proof −
{
fix *f*
assume *as*: $(\forall n$::*nat*. $f\ n \in s)$ *Cauchy f*
from *as*(*1*) **obtain** $l\ r$ **where** *lr*: $l \in s$ *subseq r* $(f \circ r) \longrightarrow l$
using *assms* **unfolding** *compact-def* **by** *blast*

note $lr' =$ *seq-suble* [*OF lr*(*2*)]
{
fix e :: *real*
assume $e > 0$
from *as*(*2*) **obtain** N **where** $N{:}\forall m\ n.\ N \leq m \wedge N \leq n \longrightarrow dist\ (f\ m)\ (f\ n) < e/2$
unfolding *cauchy-def*
using ‹$e > 0$›
apply (*erule-tac* $x{=}e/2$ **in** *allE*)
apply *auto*
done
from *lr*(*3*)[*unfolded lim-sequentially*, *THEN spec*[**where** $x{=}e/2$]]
obtain M **where** $M{:}\forall n{\geq}M.\ dist\ ((f \circ r)\ n)\ l < e/2$
using ‹$e > 0$› **by** *auto*
{
fix n :: *nat*
assume n: $n \geq max\ N\ M$
have $dist\ ((f \circ r)\ n)\ l < e/2$
using $n\ M$ **by** *auto*
moreover have $r\ n \geq N$
using lr'[*of n*] n **by** *auto*
then have $dist\ (f\ n)\ ((f \circ r)\ n) < e\ /\ 2$
using N **and** n **by** *auto*
ultimately have $dist\ (f\ n)\ l < e$
using *dist-triangle-half-r*[*of f* $(r\ n)$ $f\ n\ e\ l$]
by (*auto simp add*: *dist-commute*)
}
then have $\exists N.\ \forall n{\geq}N.\ dist\ (f\ n)\ l < e$ **by** *blast*
}
then have $\exists l{\in}s.\ (f \longrightarrow l)$ *sequentially* **using** ‹$l{\in}s$›
unfolding *lim-sequentially* **by** *auto*
}
then show *?thesis* **unfolding** *complete-def* **by** *auto*

qed

lemma *nat-approx-posE*:
fixes *e*::*real*
assumes *0 < e*
obtains *n* :: *nat* **where** *1 / (Suc n) < e*
proof *atomize-elim*
have *1 / real (Suc (nat ⌈1/e⌉)) < 1 / ⌈1/e⌉*
by (*rule divide-strict-left-mono*) (*auto simp*: ‹*0 < e*›)
also have *1 / ⌈1/e⌉ ≤ 1 / (1/e)*
by (*rule divide-left-mono*) (*auto simp*: ‹*0 < e*› *ceiling-correct*)
also have *... = e* **by** *simp*
finally show *∃ n. 1 / real (Suc n) < e* **..**
qed

lemma *compact-eq-totally-bounded*:
compact s ⟷ complete s ∧ (∀ e>0. ∃ k. finite k ∧ s ⊆ (⋃ x∈k. ball x e))
(**is** - *⟷ ?rhs*)
proof
assume *assms*: *?rhs*
then obtain *k* **where** *k*: *⋀e. 0 < e ⟹ finite (k e) ⋀e. 0 < e ⟹ s ⊆ (⋃ x∈k e. ball x e)*
by (*auto simp*: *choice-iff'*)

show *compact s*
proof *cases*
assume *s = {}*
then show *compact s* **by** (*simp add*: *compact-def*)
next
assume *s ≠ {}*
show *?thesis*
unfolding *compact-def*
proof *safe*
fix *f* :: *nat ⇒ 'a*
assume *f*: *∀ n. f n ∈ s*

def *e ≡ λn. 1 / (2 * Suc n)*
then have [*simp*]: *⋀n. 0 < e n* **by** *auto*
def *B ≡ λn U. SOME b. infinite {n. f n ∈ b} ∧ (∃ x. b ⊆ ball x (e n) ∩ U)*
{
fix *n U*
assume *infinite {n. f n ∈ U}*
then have *∃ b∈k (e n). infinite {i∈{n. f n ∈ U}. f i ∈ ball b (e n)}*
using *k f* **by** (*intro pigeonhole-infinite-rel*) (*auto simp*: *subset-eq*)
then obtain *a* **where**
a ∈ k (e n)
infinite {i ∈ {n. f n ∈ U}. f i ∈ ball a (e n)} **..**
then have *∃ b. infinite {i. f i ∈ b} ∧ (∃ x. b ⊆ ball x (e n) ∩ U)*
by (*intro exI*[*of* - *ball a (e n) ∩ U*] *exI*[*of* - *a*]) (*auto simp*: *ac-simps*)

```
    from someI-ex[OF this]
    have infinite {i. f i ∈ B n U} ∃x. B n U ⊆ ball x (e n) ∩ U
      unfolding B-def by auto
  }
  note B = this

  def F ≡ rec-nat (B 0 UNIV) B
  {
    fix n
    have infinite {i. f i ∈ F n}
      by (induct n) (auto simp: F-def B)
  }
  then have F: ⋀n. ∃x. F (Suc n) ⊆ ball x (e n) ∩ F n
    using B by (simp add: F-def)
  then have F-dec: ⋀m n. m ≤ n ⟹ F n ⊆ F m
    using decseq-SucI[of F] by (auto simp: decseq-def)

  obtain sel where sel: ⋀k i. i < sel k i ⋀k i. f (sel k i) ∈ F k
  proof (atomize-elim, unfold all-conj-distrib[symmetric], intro choice allI)
    fix k i
    have infinite ({n. f n ∈ F k} - {.. i})
      using ‹infinite {n. f n ∈ F k}› by auto
    from infinite-imp-nonempty[OF this]
    show ∃x>i. f x ∈ F k
      by (simp add: set-eq-iff not-le conj-commute)
  qed

  def t ≡ rec-nat (sel 0 0) (λn i. sel (Suc n) i)
  have subseq t
    unfolding subseq-Suc-iff by (simp add: t-def sel)
  moreover have ∀i. (f ∘ t) i ∈ s
    using f by auto
  moreover
  {
    fix n
    have (f ∘ t) n ∈ F n
      by (cases n) (simp-all add: t-def sel)
  }
  note t = this

  have Cauchy (f ∘ t)
  proof (safe intro!: metric-CauchyI exI elim!: nat-approx-posE)
    fix r :: real and N n m
    assume 1 / Suc N < r Suc N ≤ n Suc N ≤ m
    then have (f ∘ t) n ∈ F (Suc N) (f ∘ t) m ∈ F (Suc N) 2 * e N < r
      using F-dec t by (auto simp: e-def field-simps of-nat-Suc)
    with F[of N] obtain x where dist x ((f ∘ t) n) < e N dist x ((f ∘ t) m)
< e N
      by (auto simp: subset-eq)
```

```
      with dist-triangle[of (f ∘ t) m (f ∘ t) n x] ‹2 * e N < r›
      show dist ((f ∘ t) m) ((f ∘ t) n) < r
        by (simp add: dist-commute)
    qed

    ultimately show ∃l∈s. ∃r. subseq r ∧ (f ∘ r) ⟶ l
      using assms unfolding complete-def by blast
  qed
 qed
qed (metis compact-imp-complete compact-imp-seq-compact seq-compact-imp-totally-bounded)

lemma cauchy: Cauchy s ⟷ (∀e>0.∃ N::nat. ∀n≥N. dist(s n)(s N) < e) (is
?lhs = ?rhs)
proof −
 {
  assume ?rhs
  {
   fix e::real
   assume e>0
   with ‹?rhs› obtain N where N:∀n≥N. dist (s n) (s N) < e/2
    by (erule-tac x=e/2 in allE) auto
   {
    fix n m
    assume nm:N ≤ m ∧ N ≤ n
    then have dist (s m) (s n) < e using N
     using dist-triangle-half-l[of s m s N e s n]
     by blast
   }
   then have ∃N. ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (s m) (s n) < e
    by blast
  }
  then have ?lhs
   unfolding cauchy-def
   by blast
 }
 then show ?thesis
  unfolding cauchy-def
  using dist-triangle-half-l
  by blast
qed

lemma cauchy-imp-bounded:
 assumes Cauchy s
 shows bounded (range s)
proof −
 from assms obtain N :: nat where ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (s m) (s
n) < 1
  unfolding cauchy-def
  apply (erule-tac x= 1 in allE)
```

```
    apply auto
    done
  then have N:∀ n. N ≤ n ⟶ dist (s N) (s n) < 1 by auto
  moreover
  have bounded (s ‘ {0..N})
    using finite-imp-bounded[of s ‘ {1..N}] by auto
  then obtain a where a:∀ x∈s ‘ {0..N}. dist (s N) x ≤ a
    unfolding bounded-any-center [where a=s N] by auto
  ultimately show ?thesis
    unfolding bounded-any-center [where a=s N]
    apply (rule-tac x=max a 1 in exI)
    apply auto
    apply (erule-tac x=y in allE)
    apply (erule-tac x=y in ballE)
    apply auto
    done
qed

instance heine-borel < complete-space
proof
  fix f :: nat ⇒ 'a assume Cauchy f
  then have bounded (range f)
    by (rule cauchy-imp-bounded)
  then have compact (closure (range f))
    unfolding compact-eq-bounded-closed by auto
  then have complete (closure (range f))
    by (rule compact-imp-complete)
  moreover have ∀ n. f n ∈ closure (range f)
    using closure-subset [of range f] by auto
  ultimately have ∃ l∈closure (range f). (f ⟶⟶ l) sequentially
    using ‹Cauchy f› unfolding complete-def by auto
  then show convergent f
    unfolding convergent-def by auto
qed

instance euclidean-space ⊆ banach ..

lemma complete-UNIV: complete (UNIV :: ('a::complete-space) set)
proof (rule completeI)
  fix f :: nat ⇒ 'a assume Cauchy f
  then have convergent f by (rule Cauchy-convergent)
  then show ∃ l∈UNIV. f ⟶⟶ l unfolding convergent-def by simp
qed

lemma complete-imp-closed:
  fixes s :: 'a::metric-space set
  assumes complete s
  shows closed s
proof (unfold closed-sequential-limits, clarify)
```

fix $f\ x$ **assume** $\forall\, n.\ f\ n \in s$ **and** $f \longrightarrow x$
from ‹$f \longrightarrow x$› **have** *Cauchy f*
by (*rule LIMSEQ-imp-Cauchy*)
with ‹*complete s*› **and** ‹$\forall\, n.\ f\ n \in s$› **obtain** l **where** $l \in s$ **and** $f \longrightarrow l$
by (*rule completeE*)
from ‹$f \longrightarrow x$› **and** ‹$f \longrightarrow l$› **have** $x = l$
by (*rule LIMSEQ-unique*)
with ‹$l \in s$› **show** $x \in s$
by *simp*
qed

lemma *complete-inter-closed*:
fixes s :: *'a*::*metric-space set*
assumes *complete s* **and** *closed t*
shows *complete* $(s \cap t)$
proof (*rule completeI*)
fix f **assume** $\forall\, n.\ f\ n \in s \cap t$ **and** *Cauchy f*
then have $\forall\, n.\ f\ n \in s$ **and** $\forall\, n.\ f\ n \in t$
by *simp-all*
from ‹*complete s*› **obtain** l **where** $l \in s$ **and** $f \longrightarrow l$
using ‹$\forall\, n.\ f\ n \in s$› **and** ‹*Cauchy f*› **by** (*rule completeE*)
from ‹*closed t*› **and** ‹$\forall\, n.\ f\ n \in t$› **and** ‹$f \longrightarrow l$› **have** $l \in t$
by (*rule closed-sequentially*)
with ‹$l \in s$› **and** ‹$f \longrightarrow l$› **show** $\exists\, l{\in}s \cap t.\ f \longrightarrow l$
by *fast*
qed

lemma *complete-closed-subset*:
fixes s :: *'a*::*metric-space set*
assumes *closed s* **and** $s \subseteq t$ **and** *complete t*
shows *complete s*
using *assms complete-inter-closed* [*of t s*] **by** (*simp add*: *Int-absorb1*)

lemma *complete-eq-closed*:
fixes s :: (*'a*::*complete-space*) *set*
shows *complete s* $\longleftrightarrow$ *closed s*
proof
assume *closed s* **then show** *complete s*
using *subset-UNIV complete-UNIV* **by** (*rule complete-closed-subset*)
next
assume *complete s* **then show** *closed s*
by (*rule complete-imp-closed*)
qed

lemma *convergent-eq-cauchy*:
fixes s :: *nat* $\Rightarrow$ *'a*::*complete-space*
shows $(\exists\, l.\ (s \longrightarrow l)\ \mathit{sequentially}) \longleftrightarrow \mathit{Cauchy}\ s$
unfolding *Cauchy-convergent-iff convergent-def* **..**

lemma *convergent-imp-bounded*:
fixes $s :: nat \Rightarrow 'a::metric\text{-}space$
shows $(s \longrightarrow l)$ *sequentially* $\Longrightarrow$ *bounded* (*range* s)
by (*intro cauchy-imp-bounded LIMSEQ-imp-Cauchy*)

lemma *compact-cball*[*simp*]:
fixes $x :: 'a::heine\text{-}borel$
shows *compact* (*cball* x e)
using *compact-eq-bounded-closed bounded-cball closed-cball*
by *blast*

lemma *compact-frontier-bounded*[*intro*]:
fixes $s :: 'a::heine\text{-}borel\ set$
shows *bounded* $s \Longrightarrow$ *compact* (*frontier* s)
unfolding *frontier-def*
using *compact-eq-bounded-closed*
by *blast*

lemma *compact-frontier*[*intro*]:
fixes $s :: 'a::heine\text{-}borel\ set$
shows *compact* $s \Longrightarrow$ *compact* (*frontier* s)
using *compact-eq-bounded-closed compact-frontier-bounded*
by *blast*

lemma *frontier-subset-compact*:
fixes $s :: 'a::heine\text{-}borel\ set$
shows *compact* $s \Longrightarrow$ *frontier* $s \subseteq s$
using *frontier-subset-closed compact-eq-bounded-closed*
by *blast*

16.21 Relations among convergence and absolute convergence for power series.

lemma *summable-imp-bounded*:
fixes $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
shows *summable* $f \Longrightarrow$ *bounded* (*range* f)
by (*frule summable-LIMSEQ-zero*) (*simp add*: *convergent-imp-bounded*)

lemma *summable-imp-sums-bounded*:
summable $f \Longrightarrow$ *bounded* (*range* $(\lambda n.\ setsum\ f\ \{..<n\})$)
by (*auto simp*: *summable-def sums-def dest*: *convergent-imp-bounded*)

lemma *power-series-conv-imp-absconv-weak*:
fixes $a:: nat \Rightarrow 'a::\{real\text{-}normed\text{-}div\text{-}algebra, banach\}$ **and** $w :: 'a$
assumes *sum*: *summable* $(\lambda n.\ a\ n * z\ \hat{}\ n)$ **and** *no*: *norm* $w <$ *norm* z
shows *summable* $(\lambda n.\ of\text{-}real(norm(a\ n)) * w\ \hat{}\ n)$
proof −
obtain M **where** M: $\bigwedge x.\ norm\ (a\ x * z\ \hat{}\ x) \leq M$
using *summable-imp-bounded* [*OF sum*] **by** (*force simp add*: *bounded-iff*)

then have $*$: *summable* ($\lambda n.$ *norm* (*a n*) $*$ *norm w* ^ *n*)
by (*rule-tac M=M* **in** *Abel-lemma*) (*auto simp*: *norm-mult norm-power intro*: *no*)
show *?thesis*
apply (*rule series-comparison-complex* [*of* ($\lambda n.$ *of-real*(*norm*(*a n*) $*$ *norm w* ^ *n*))])
apply (*simp only*: *summable-complex-of-real* $*$)
apply (*auto simp*: *norm-mult norm-power*)
done
qed

16.22 Bounded closed nest property (proof does not use Heine-Borel)

lemma *bounded-closed-nest*:
fixes *s* :: *nat* $\Rightarrow$ ('*a*::*heine-borel*) *set*
assumes $\forall n.$ *closed* (*s n*)
and $\forall n.$ *s n* $\neq$ {}
and $\forall m\ n.$ *m* $\leq$ *n* $\longrightarrow$ *s n* $\subseteq$ *s m*
and *bounded* (*s 0*)
shows $\exists a.\ \forall n.$ *a* $\in$ *s n*
proof −
from *assms*(*2*) **obtain** *x* **where** *x*: $\forall n.$ *x n* $\in$ *s n*
using *choice*[*of* $\lambda n\ x.$ *x* $\in$ *s n*] **by** *auto*
from *assms*(*4*,*1*) **have** *seq-compact* (*s 0*)
by (*simp add*: *bounded-closed-imp-seq-compact*)
then obtain *l r* **where** *lr*: *l* $\in$ *s 0 subseq r* (*x* $\circ$ *r*) $\longrightarrow$ *l*
using *x* **and** *assms*(*3*) **unfolding** *seq-compact-def* **by** *blast*
have $\forall n.$ *l* $\in$ *s n*
proof
fix *n* :: *nat*
have *closed* (*s n*)
using *assms*(*1*) **by** *simp*
moreover have $\forall i.$ (*x* $\circ$ *r*) *i* $\in$ *s i*
using *x* **and** *assms*(*3*) **and** *lr*(*2*) [*THEN seq-suble*] **by** *auto*
then have $\forall i.$ (*x* $\circ$ *r*) (*i* + *n*) $\in$ *s n*
using *assms*(*3*) **by** (*fast intro*!: *le-add2*)
moreover have ($\lambda i.$ (*x* $\circ$ *r*) (*i* + *n*)) $\longrightarrow$ *l*
using *lr*(*3*) **by** (*rule LIMSEQ-ignore-initial-segment*)
ultimately show *l* $\in$ *s n*
by (*rule closed-sequentially*)
qed
then show *?thesis* **..**
qed

Decreasing case does not even need compactness, just completeness.

lemma *decreasing-closed-nest*:
fixes *s* :: *nat* $\Rightarrow$ ('*a*::*complete-space*) *set*
assumes

```
    ∀n. closed (s n)
    ∀n. s n ≠ {}
    ∀m n. m ≤ n ⟶ s n ⊆ s m
    ∀e>0. ∃n. ∀x∈s n. ∀y∈s n. dist x y < e
  shows ∃a. ∀n. a ∈ s n
proof −
  have ∀n. ∃x. x ∈ s n
    using assms(2) by auto
  then have ∃t. ∀n. t n ∈ s n
    using choice[of λn x. x ∈ s n] by auto
  then obtain t where t: ∀n. t n ∈ s n by auto
  {
    fix e :: real
    assume e > 0
    then obtain N where N:∀x∈s N. ∀y∈s N. dist x y < e
      using assms(4) by auto
    {
      fix m n :: nat
      assume N ≤ m ∧ N ≤ n
      then have t m ∈ s N t n ∈ s N
        using assms(3) t unfolding  subset-eq t by blast+
      then have dist (t m) (t n) < e
        using N by auto
    }
    then have ∃N. ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (t m) (t n) < e
      by auto
  }
  then have Cauchy t
    unfolding cauchy-def by auto
  then obtain l where l:(t ⟶⟶ l) sequentially
    using complete-UNIV unfolding complete-def by auto
  {
    fix n :: nat
    {
      fix e :: real
      assume e > 0
      then obtain N :: nat where N: ∀n≥N. dist (t n) l < e
        using l[unfolded lim-sequentially] by auto
      have t (max n N) ∈ s n
        using assms(3)
        unfolding subset-eq
        apply (erule-tac x=n in allE)
        apply (erule-tac x=max n N in allE)
        using t
        apply auto
        done
      then have ∃y∈s n. dist y l < e
        apply (rule-tac x=t (max n N) in bexI)
        using N
```

apply *auto*
done
}
then have $l \in s\ n$
using *closed-approachable*[*of s n l*] *assms*(*1*) **by** *auto*
}
then show *?thesis* **by** *auto*
qed

Strengthen it to the intersection actually being a singleton.

lemma *decreasing-closed-nest-sing*:
fixes $s :: nat \Rightarrow 'a$::*complete-space set*
assumes
$\forall n.\ closed(s\ n)$
$\forall n.\ s\ n \neq \{\}$
$\forall m\ n.\ m \leq n \longrightarrow s\ n \subseteq s\ m$
$\forall e{>}0.\ \exists n.\ \forall x \in (s\ n).\ \forall\ y{\in}(s\ n).\ dist\ x\ y < e$
shows $\exists a.\ \bigcap(range\ s) = \{a\}$
proof −
obtain *a* **where** *a*: $\forall n.\ a \in s\ n$
using *decreasing-closed-nest*[*of s*] **using** *assms* **by** *auto*
{
fix *b*
assume *b*: $b \in \bigcap(range\ s)$
{
fix *e* :: *real*
assume $e > 0$
then have $dist\ a\ b < e$
using *assms*(*4*) **and** *b* **and** *a* **by** *blast*
}
then have $dist\ a\ b = 0$
by (*metis dist-eq-0-iff dist-nz less-le*)
}
with *a* **have** $\bigcap(range\ s) = \{a\}$
unfolding *image-def* **by** *auto*
then show *?thesis* **..**
qed

Cauchy-type criteria for uniform convergence.

lemma *uniformly-convergent-eq-cauchy*:
fixes *s*::$nat \Rightarrow 'b \Rightarrow 'a$::*complete-space*
shows
$(\exists l.\ \forall e{>}0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e) \longleftrightarrow$
$(\forall e{>}0.\ \exists N.\ \forall m\ n\ x.\ N \leq m \wedge N \leq n \wedge P\ x \longrightarrow dist\ (s\ m\ x)\ (s\ n\ x) < e)$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *l* **where** *l*:$\forall e{>}0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist\ (s\ n\ x)\ (l\ x)$
$< e$

```
    by auto
  {
    fix e :: real
    assume e > 0
    then obtain N :: nat where N: ∀ n x. N ≤ n ∧ P x ⟶ dist (s n x) (l x) <
e / 2
      using l[THEN spec[where x=e/2]] by auto
    {
      fix n m :: nat and x :: 'b
      assume N ≤ m ∧ N ≤ n ∧ P x
      then have dist (s m x) (s n x) < e
        using N[THEN spec[where x=m], THEN spec[where x=x]]
        using N[THEN spec[where x=n], THEN spec[where x=x]]
        using dist-triangle-half-l[of s m x l x e s n x] by auto
    }
    then have ∃ N. ∀ m n x. N ≤ m ∧ N ≤ n ∧ P x  --> dist (s m x) (s n x)
< e  by auto
  }
  then show ?rhs by auto
next
  assume ?rhs
  then have ∀ x. P x ⟶ Cauchy (λn. s n x)
    unfolding cauchy-def
    apply auto
    apply (erule-tac x=e in allE)
    apply auto
    done
  then obtain l where l: ∀ x. P x ⟶ ((λn. s n x) ⟶⟶ l x) sequentially
    unfolding convergent-eq-cauchy[symmetric]
    using choice[of λx l. P x ⟶ ((λn. s n x) ⟶⟶ l) sequentially]
    by auto
  {
    fix e :: real
    assume e > 0
    then obtain N where N:∀ m n x. N ≤ m ∧ N ≤ n ∧ P x ⟶ dist (s m x)
(s n x) < e/2
      using ‹?rhs›[THEN spec[where x=e/2]] by auto
    {
      fix x
      assume P x
      then obtain M where M:∀ n≥M. dist (s n x) (l x) < e/2
        using l[THEN spec[where x=x], unfolded lim-sequentially] and ‹e > 0›
        by (auto elim!: allE[where x=e/2])
      fix n :: nat
      assume n ≥ N
      then have dist(s n x)(l x) < e
        using ‹P x›and N[THEN spec[where x=n], THEN spec[where x=N+M],
THEN spec[where x=x]]
          using M[THEN spec[where x=N+M]] and dist-triangle-half-l[of s n x s
```

$(N+M)$ x e l x]
by (*auto simp add*: *dist-commute*)
}
then have $\exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e$
by *auto*
}
then show *?lhs* **by** *auto*
qed

lemma *uniformly-cauchy-imp-uniformly-convergent*:
fixes $s :: nat \Rightarrow 'a \Rightarrow 'b$::*complete-space*
assumes $\forall e>0.\exists N.\ \forall m\ (n::nat)\ x.\ N \leq m \wedge N \leq n \wedge P\ x \dashrightarrow dist(s\ m\ x)(s\ n\ x) < e$
and $\forall x.\ P\ x \dashrightarrow (\forall e>0.\ \exists N.\ \forall n.\ N \leq n \longrightarrow dist(s\ n\ x)(l\ x) < e)$
shows $\forall e>0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e$
proof −
obtain l' **where** l:$\forall e>0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge P\ x \longrightarrow dist\ (s\ n\ x)\ (l'\ x) < e$
using *assms*(*1*) **unfolding** *uniformly-convergent-eq-cauchy*[*symmetric*] **by** *auto*
moreover
{
fix x
assume $P\ x$
then have $l\ x = l'\ x$
using *tendsto-unique*[*OF trivial-limit-sequentially*, *of* $\lambda n.\ s\ n\ x$ l x l' x]
using l **and** *assms*(*2*) **unfolding** *lim-sequentially* **by** *blast*
}
ultimately show *?thesis* **by** *auto*
qed

16.23 Continuity

Derive the epsilon-delta forms, which we often use as "definitions"

lemma *continuous-within-eps-delta*:
continuous (*at* x *within* s) $f \longleftrightarrow (\forall e>0.\ \exists d>0.\ \forall x'\in s.\ \ dist\ x'\ x < d \dashrightarrow dist\ (f\ x')\ (f\ x) < e)$
unfolding *continuous-within* **and** *Lim-within*
apply *auto*
apply (*metis dist-nz dist-self*)
apply *blast*
done

corollary *continuous-at-eps-delta*:
continuous (*at* x) $f \longleftrightarrow (\forall e > 0.\ \exists d > 0.\ \forall x'.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e)$
using *continuous-within-eps-delta* [*of* x *UNIV* f] **by** *simp*

lemma *continuous-at-right-real-increasing*:
fixes $f :: real \Rightarrow real$
assumes *nondecF*: $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$

shows *continuous* (*at-right a*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. *f* (*a* + *d*) − *f a* < *e*)
apply (*simp add*: *greaterThan-def dist-real-def continuous-within Lim-within-le*)
apply (*intro all-cong ex-cong*)
apply *safe*
apply (*erule-tac x*=*a* + *d* **in** *allE*)
apply *simp*
apply (*simp add*: *nondecF field-simps*)
apply (*drule nondecF*)
apply *simp*
done

lemma *continuous-at-left-real-increasing*:
assumes *nondecF*: ⋀ *x y*. *x* ≤ *y* ⟹ *f x* ≤ ((*f y*) :: *real*)
shows (*continuous* (*at-left* (*a* :: *real*)) *f*) = (∀ *e* > *0*. ∃ *delta* > *0*. *f a* − *f* (*a* − *delta*) < *e*)
apply (*simp add*: *lessThan-def dist-real-def continuous-within Lim-within-le*)
apply (*intro all-cong ex-cong*)
apply *safe*
apply (*erule-tac x*=*a* − *d* **in** *allE*)
apply *simp*
apply (*simp add*: *nondecF field-simps*)
apply (*cut-tac x*=*a* − *d* **and** *y*=*x* **in** *nondecF*)
apply *simp-all*
done

Versions in terms of open balls.

lemma *continuous-within-ball*:
continuous (*at x within s*) *f* ⟷
(∀ *e* > *0*. ∃ *d* > *0*. *f* ‘ (*ball x d* ∩ *s*) ⊆ *ball* (*f x*) *e*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{
fix *e* :: *real*
assume *e* > *0*
then obtain *d* **where** *d*: *d*>*0* ∀ *xa*∈*s*. *0* < *dist xa x* ∧ *dist xa x* < *d* ⟶ *dist* (*f xa*) (*f x*) < *e*
using ‹*?lhs*›[*unfolded continuous-within Lim-within*] **by** *auto*
{
fix *y*
assume *y* ∈ *f* ‘ (*ball x d* ∩ *s*)
then have *y* ∈ *ball* (*f x*) *e*
using *d*(*2*)
apply (*auto simp add*: *dist-commute*)
apply (*erule-tac x*=*xa* **in** *ballE*)
apply *auto*
using ‹*e* > *0*›
apply *auto*
done

```
    }
    then have ∃ d>0. f ‘ (ball x d ∩ s) ⊆ ball (f x) e
      using ‹d > 0›
      unfolding subset-eq ball-def by (auto simp add: dist-commute)
  }
  then show ?rhs by auto
next
  assume ?rhs
  then show ?lhs
    unfolding continuous-within Lim-within ball-def subset-eq
    apply (auto simp add: dist-commute)
    apply (erule-tac x=e in allE)
    apply auto
    done
qed

lemma continuous-at-ball:
  continuous (at x) f ⟷ (∀ e>0. ∃ d>0. f ‘ (ball x d) ⊆ ball (f x) e) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding continuous-at Lim-at subset-eq Ball-def Bex-def image-iff mem-ball
    apply auto
    apply (erule-tac x=e in allE)
    apply auto
    apply (rule-tac x=d in exI)
    apply auto
    apply (erule-tac x=xa in allE)
    apply (auto simp add: dist-commute)
    done
next
  assume ?rhs
  then show ?lhs
    unfolding continuous-at Lim-at subset-eq Ball-def Bex-def image-iff mem-ball
    apply auto
    apply (erule-tac x=e in allE)
    apply auto
    apply (rule-tac x=d in exI)
    apply auto
    apply (erule-tac x=f xa in allE)
    apply (auto simp add: dist-commute)
    done
qed
```

Define setwise continuity in terms of limits within the set.

```
lemma continuous-on-iff:
  continuous-on s f ⟷
    (∀ x∈s. ∀ e>0. ∃ d>0. ∀ x'∈s. dist x' x < d ⟶ dist (f x') (f x) < e)
```

unfolding *continuous-on-def Lim-within*
by (*metis dist-pos-lt dist-self*)

lemma *continuous-within-E*:
assumes *continuous* (*at x within s*) *f e>0*
obtains *d* **where** *d>0* $\bigwedge x'.$ $[\![x' \in s;\ dist\ x'\ x \leq d]\!] \Longrightarrow dist\ (f\ x')\ (f\ x) < e$
using *assms* **apply** (*simp add*: *continuous-within-eps-delta*)
apply (*drule spec* [*of* - *e*], *clarify*)
apply (*rule-tac d=d/2* **in** *that*, *auto*)
done

lemma *continuous-onI* [*intro?*]:
assumes $\bigwedge x\ e.$ $[\![e > 0;\ x \in s]\!] \Longrightarrow \exists\, d>0.\ \forall\, x' \in s.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) \leq e$
shows *continuous-on s f*
apply (*simp add*: *continuous-on-iff*, *clarify*)
apply (*rule ex-forward* [*OF assms* [*OF half-gt-zero*]], *auto*)
done

lemma *uniformly-continuous-on-def*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
shows *uniformly-continuous-on s f* ⟷
$(\forall\, e>0.\ \exists\, d>0.\ \forall\, x \in s.\ \forall\, x' \in s.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e)$
unfolding *uniformly-continuous-on-uniformity*
uniformity-dist filterlim-INF filterlim-principal eventually-inf-principal
by (*force simp*: *Ball-def uniformity-dist*[*symmetric*] *eventually-uniformity-metric*)

Some simple consequential lemmas.

lemma *continuous-onD*:
assumes *continuous-on s f x∈s e>0*
obtains *d* **where** *d>0* $\bigwedge x'.$ $[\![x' \in s;\ dist\ x'\ x < d]\!] \Longrightarrow dist\ (f\ x')\ (f\ x) < e$
using *assms* [*unfolded continuous-on-iff*] **by** *blast*

lemma *uniformly-continuous-onE*:
assumes *uniformly-continuous-on s f 0 < e*
obtains *d* **where** *d>0* $\bigwedge x\ x'.$ $[\![x \in s;\ x' \in s;\ dist\ x'\ x < d]\!] \Longrightarrow dist\ (f\ x')\ (f\ x) < e$
using *assms*
by (*auto simp*: *uniformly-continuous-on-def*)

lemma *continuous-at-imp-continuous-within*:
continuous (*at x*) *f* ⟹ *continuous* (*at x within s*) *f*
unfolding *continuous-within continuous-at* **using** *Lim-at-imp-Lim-at-within* **by** *auto*

lemma *Lim-trivial-limit*: *trivial-limit net* ⟹ (*f* ⟶ *l*) *net*
by *simp*

lemmas *continuous-on* = *continuous-on-def* — legacy theorem name

lemma *continuous-within-subset*:
continuous (*at x within s*) *f* ⟹ *t* ⊆ *s* ⟹ *continuous* (*at x within t*) *f*
unfolding *continuous-within* **by**(*metis tendsto-within-subset*)

lemma *continuous-on-interior*:
continuous-on s f ⟹ *x* ∈ *interior s* ⟹ *continuous* (*at x*) *f*
by (*metis continuous-on-eq-continuous-at continuous-on-subset interiorE*)

lemma *continuous-on-eq*:
⟦*continuous-on s f*; ⋀*x*. *x* ∈ *s* ⟹ *f x* = *g x*⟧ ⟹ *continuous-on s g*
unfolding *continuous-on-def tendsto-def eventually-at-topological*
by *simp*

Characterization of various kinds of continuity in terms of sequences.

lemma *continuous-within-sequentially*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*topological-space*
shows *continuous* (*at a within s*) *f* ⟷
(∀ *x*. (∀ *n*::*nat*. *x n* ∈ *s*) ∧ (*x* ⟶ *a*) *sequentially*
⟶ ((*f* ∘ *x*) ⟶ *f a*) *sequentially*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{
fix *x* :: *nat* ⇒ *'a*
assume *x*: ∀ *n*. *x n* ∈ *s* ∀ *e*>*0*. *eventually* (λ*n*. *dist* (*x n*) *a* < *e*) *sequentially*
fix *T* :: *'b set*
assume *open T* **and** *f a* ∈ *T*
with ‹*?lhs*› **obtain** *d* **where** *d*>*0* **and** *d*:∀ *x*∈*s*. *0* < *dist x a* ∧ *dist x a* < *d*
⟶ *f x* ∈ *T*
unfolding *continuous-within tendsto-def eventually-at* **by** *auto*
have *eventually* (λ*n*. *dist* (*x n*) *a* < *d*) *sequentially*
using *x*(*2*) ‹*d*>*0*› **by** *simp*
then have *eventually* (λ*n*. (*f* ∘ *x*) *n* ∈ *T*) *sequentially*
proof *eventually-elim*
case (*elim n*)
then show *?case*
using *d x*(*1*) ‹*f a* ∈ *T*› **by** *auto*
qed
}
then show *?rhs*
unfolding *tendsto-iff tendsto-def* **by** *simp*
next
assume *?rhs*
then show *?lhs*
unfolding *continuous-within tendsto-def* [**where** *l*=*f a*]
by (*simp add*: *sequentially-imp-eventually-within*)
qed

lemma *continuous-at-sequentially*:
fixes f :: *'a::metric-space* $\Rightarrow$ *'b::topological-space*
shows *continuous* (*at a*) $f \longleftrightarrow$
$(\forall x.\ (x \longrightarrow a)\ \mathit{sequentially} \longrightarrow ((f \circ x) \longrightarrow f\ a)\ \mathit{sequentially})$
using *continuous-within-sequentially*[*of a UNIV f*] **by** *simp*

lemma *continuous-on-sequentially*:
fixes f :: *'a::metric-space* $\Rightarrow$ *'b::topological-space*
shows *continuous-on s f* $\longleftrightarrow$
$(\forall x.\ \forall a \in s.\ (\forall n.\ x(n) \in s) \wedge (x \longrightarrow a)\ \mathit{sequentially}$
$\longrightarrow ((f \circ x) \longrightarrow f\ a)\ \mathit{sequentially})$
(**is** *?lhs* = *?rhs*)
proof
assume *?rhs*
then show *?lhs*
using *continuous-within-sequentially*[*of - s f*]
unfolding *continuous-on-eq-continuous-within*
by *auto*
next
assume *?lhs*
then show *?rhs*
unfolding *continuous-on-eq-continuous-within*
using *continuous-within-sequentially*[*of - s f*]
by *auto*
qed

lemma *uniformly-continuous-on-sequentially*:
uniformly-continuous-on s f $\longleftrightarrow$ $(\forall x\ y.\ (\forall n.\ x\ n \in s) \wedge (\forall n.\ y\ n \in s) \wedge$
$(\lambda n.\ \mathit{dist}\ (x\ n)\ (y\ n)) \longrightarrow 0 \longrightarrow (\lambda n.\ \mathit{dist}\ (f(x\ n))\ (f(y\ n))) \longrightarrow 0)$ (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{
fix $x\ y$
assume x: $\forall n.\ x\ n \in s$
and y: $\forall n.\ y\ n \in s$
and *xy*: $((\lambda n.\ \mathit{dist}\ (x\ n)\ (y\ n)) \longrightarrow 0)$ *sequentially*
{
fix e :: *real*
assume $e > 0$
then obtain d **where** $d > 0$ **and** d: $\forall x{\in}s.\ \forall x'{\in}s.\ \mathit{dist}\ x'\ x < d \longrightarrow \mathit{dist}\ (f\ x')\ (f\ x) < e$
using ‹*?lhs*›[*unfolded uniformly-continuous-on-def*, *THEN spec*[**where** $x{=}e$]]
by *auto*
obtain N **where** N: $\forall n{\geq}N.\ \mathit{dist}\ (x\ n)\ (y\ n) < d$
using *xy*[*unfolded lim-sequentially dist-norm*] **and** ‹$d{>}0$› **by** *auto*
{
fix n
assume $n{\geq}N$

```
      then have dist (f (x n)) (f (y n)) < e
        using N[THEN spec[where x=n]]
        using d[THEN bspec[where x=x n], THEN bspec[where x=y n]]
        using x and y
        unfolding dist-commute
        by simp
     }
     then have ∃N. ∀n≥N. dist (f (x n)) (f (y n)) < e
      by auto
   }
   then have ((λn. dist (f(x n)) (f(y n))) ⟶ 0) sequentially
    unfolding lim-sequentially and dist-real-def by auto
  }
  then show ?rhs by auto
next
  assume ?rhs
  {
   assume ¬ ?lhs
   then obtain e where e > 0 ∀d>0. ∃x∈s. ∃x'∈s. dist x' x < d ∧ ¬ dist (f
x') (f x) < e
    unfolding uniformly-continuous-on-def by auto
   then obtain fa where fa:
    ∀x. 0 < x ⟶ fst (fa x) ∈ s ∧ snd (fa x) ∈ s ∧ dist (fst (fa x)) (snd (fa x))
< x ∧ ¬ dist (f (fst (fa x))) (f (snd (fa x))) < e
    using choice[of λd x. d>0 ⟶ fst x ∈ s ∧ snd x ∈ s ∧ dist (snd x) (fst x)
< d ∧ ¬ dist (f (snd x)) (f (fst x)) < e]
    unfolding Bex-def
    by (auto simp add: dist-commute)
   def x ≡ λn::nat. fst (fa (inverse (real n + 1)))
   def y ≡ λn::nat. snd (fa (inverse (real n + 1)))
   have xyn: ∀n. x n ∈ s ∧ y n ∈ s
    and xy0: ∀n. dist (x n) (y n) < inverse (real n + 1)
    and fxy:∀n. ¬ dist (f (x n)) (f (y n)) < e
    unfolding x-def and y-def using fa
    by auto
   {
    fix e :: real
    assume e > 0
    then obtain N :: nat where N ≠ 0 and N: 0 < inverse (real N) ∧ inverse
(real N) < e
     unfolding real-arch-inv[of e] by auto
    {
     fix n :: nat
     assume n ≥ N
     then have inverse (real n + 1) < inverse (real N)
      using of-nat-0-le-iff and ‹N≠0› by auto
     also have ... < e using N by auto
     finally have inverse (real n + 1) < e by auto
     then have dist (x n) (y n) < e
```

```
        using xy0[THEN spec[where x=n]] by auto
      }
      then have ∃N. ∀n≥N. dist (x n) (y n) < e by auto
    }
    then have ∀e>0. ∃N. ∀n≥N. dist (f (x n)) (f (y n)) < e
      using ‹?rhs›[THEN spec[where x=x], THEN spec[where x=y]] and xyn
      unfolding lim-sequentially dist-real-def by auto
    then have False using fxy and ‹e>0› by auto
  }
  then show ?lhs
    unfolding uniformly-continuous-on-def by blast
qed
```

The usual transformation theorems.

```
lemma continuous-transform-within:
  fixes f g :: 'a::metric-space ⇒ 'b::topological-space
  assumes continuous (at x within s) f
    and 0 < d
    and x ∈ s
    and ⋀x'. ⟦x' ∈ s; dist x' x < d⟧ ⟹ f x' = g x'
  shows continuous (at x within s) g
  using assms
  unfolding continuous-within
  by (force simp add: intro: Lim-transform-within)
```

16.23.1 Structural rules for pointwise continuity

```
lemmas continuous-within-id = continuous-ident

lemmas continuous-at-id = continuous-ident

lemma continuous-infdist[continuous-intros]:
  assumes continuous F f
  shows continuous F (λx. infdist (f x) A)
  using assms unfolding continuous-def by (rule tendsto-infdist)

lemma continuous-infnorm[continuous-intros]:
  continuous F f ⟹ continuous F (λx. infnorm (f x))
  unfolding continuous-def by (rule tendsto-infnorm)

lemma continuous-inner[continuous-intros]:
  assumes continuous F f
    and continuous F g
  shows continuous F (λx. inner (f x) (g x))
  using assms unfolding continuous-def by (rule tendsto-inner)

lemmas continuous-at-inverse = isCont-inverse
```

16.23.2 Structural rules for setwise continuity

lemma *continuous-on-infnorm*[*continuous-intros*]:
continuous-on s f ⟹ *continuous-on s* (λ*x*. *infnorm* (*f x*))
unfolding *continuous-on* **by** (*fast intro*: *tendsto-infnorm*)

lemma *continuous-on-inner*[*continuous-intros*]:
fixes *g* :: *'a*::*topological-space* ⇒ *'b*::*real-inner*
assumes *continuous-on s f*
and *continuous-on s g*
shows *continuous-on s* (λ*x*. *inner* (*f x*) (*g x*))
using *bounded-bilinear-inner assms*
by (*rule bounded-bilinear.continuous-on*)

16.23.3 Structural rules for uniform continuity

lemma *uniformly-continuous-on-dist*[*continuous-intros*]:
fixes *f g* :: *'a*::*metric-space* ⇒ *'b*::*metric-space*
assumes *uniformly-continuous-on s f*
and *uniformly-continuous-on s g*
shows *uniformly-continuous-on s* (λ*x*. *dist* (*f x*) (*g x*))
proof −
{
fix *a b c d* :: *'b*
have |*dist a b* − *dist c d*| ≤ *dist a c* + *dist b d*
using *dist-triangle2* [*of a b c*] *dist-triangle2* [*of b c d*]
using *dist-triangle3* [*of c d a*] *dist-triangle* [*of a d b*]
by *arith*
} note *le* = *this*
{
fix *x y*
assume *f*: (λ*n*. *dist* (*f* (*x n*)) (*f* (*y n*))) ⟶ *0*
assume *g*: (λ*n*. *dist* (*g* (*x n*)) (*g* (*y n*))) ⟶ *0*
have (λ*n*. |*dist* (*f* (*x n*)) (*g* (*x n*)) − *dist* (*f* (*y n*)) (*g* (*y n*))|) ⟶ *0*
by (*rule Lim-transform-bound* [*OF - tendsto-add-zero* [*OF f g*]],
simp add: *le*)
}
then show *?thesis*
using *assms* **unfolding** *uniformly-continuous-on-sequentially*
unfolding *dist-real-def* **by** *simp*
qed

lemma *uniformly-continuous-on-norm*[*continuous-intros*]:
fixes *f* :: *'a* :: *metric-space* ⇒ *'b* :: *real-normed-vector*
assumes *uniformly-continuous-on s f*
shows *uniformly-continuous-on s* (λ*x*. *norm* (*f x*))
unfolding *norm-conv-dist* **using** *assms*
by (*intro uniformly-continuous-on-dist uniformly-continuous-on-const*)

lemma (**in** *bounded-linear*) *uniformly-continuous-on*[*continuous-intros*]:

fixes *g* :: -::*metric-space* ⇒ -
assumes *uniformly-continuous-on s g*
shows *uniformly-continuous-on s* (λ*x*. *f* (*g x*))
using *assms* **unfolding** *uniformly-continuous-on-sequentially*
unfolding *dist-norm tendsto-norm-zero-iff diff* [*symmetric*]
by (*auto intro*: *tendsto-zero*)

lemma *uniformly-continuous-on-cmul*[*continuous-intros*]:
fixes *f* :: 'a::*metric-space* ⇒ 'b::*real-normed-vector*
assumes *uniformly-continuous-on s f*
shows *uniformly-continuous-on s* (λ*x*. *c* ∗*R* *f*(*x*))
using *bounded-linear-scaleR-right assms*
by (*rule bounded-linear.uniformly-continuous-on*)

lemma *dist-minus*:
fixes *x y* :: 'a::*real-normed-vector*
shows *dist* (− *x*) (− *y*) = *dist x y*
unfolding *dist-norm minus-diff-minus norm-minus-cancel* **..**

lemma *uniformly-continuous-on-minus*[*continuous-intros*]:
fixes *f* :: 'a::*metric-space* ⇒ 'b::*real-normed-vector*
shows *uniformly-continuous-on s f* ⟹ *uniformly-continuous-on s* (λ*x*. − *f x*)
unfolding *uniformly-continuous-on-def dist-minus* **.**

lemma *uniformly-continuous-on-add*[*continuous-intros*]:
fixes *f g* :: 'a::*metric-space* ⇒ 'b::*real-normed-vector*
assumes *uniformly-continuous-on s f*
and *uniformly-continuous-on s g*
shows *uniformly-continuous-on s* (λ*x*. *f x* + *g x*)
using *assms*
unfolding *uniformly-continuous-on-sequentially*
unfolding *dist-norm tendsto-norm-zero-iff add-diff-add*
by (*auto intro*: *tendsto-add-zero*)

lemma *uniformly-continuous-on-diff*[*continuous-intros*]:
fixes *f* :: 'a::*metric-space* ⇒ 'b::*real-normed-vector*
assumes *uniformly-continuous-on s f*
and *uniformly-continuous-on s g*
shows *uniformly-continuous-on s* (λ*x*. *f x* − *g x*)
using *assms uniformly-continuous-on-add* [*of s f* − *g*]
by (*simp add*: *fun-Compl-def uniformly-continuous-on-minus*)

lemmas *continuous-at-compose* = *isCont-o*

Continuity in terms of open preimages.

lemma *continuous-at-open*:
continuous (*at x*) *f* ⟷ (∀ *t*. *open t* ∧ *f x* ∈ *t* −−> (∃ *s*. *open s* ∧ *x* ∈ *s* ∧ (∀ *x*′ ∈ *s*. (*f x*′) ∈ *t*)))
unfolding *continuous-within-topological* [*of x UNIV f*]

unfolding *imp-conjL*
by (*intro all-cong imp-cong ex-cong conj-cong refl*) *auto*

lemma *continuous-imp-tendsto*:
assumes *continuous* (*at x0*) *f*
and $x \longrightarrow x0$
shows $(f \circ x) \longrightarrow (f\ x0)$
proof (*rule topological-tendstoI*)
fix *S*
assume *open S f x0* $\in S$
then obtain *T* **where** *T-def*: *open T x0* $\in T\ \forall x{\in}T.\ f\ x \in S$
using *assms continuous-at-open* **by** *metis*
then have *eventually* $(\lambda n.\ x\ n \in T)$ *sequentially*
using *assms T-def* **by** (*auto simp*: *tendsto-def*)
then show *eventually* $(\lambda n.\ (f \circ x)\ n \in S)$ *sequentially*
using *T-def* **by** (*auto elim*!: *eventually-mono*)
qed

lemma *continuous-on-open*:
continuous-on s f $\longleftrightarrow$
$(\forall t.$ *openin* (*subtopology euclidean* (*f* ' *s*)) *t* $\longrightarrow$
openin (*subtopology euclidean s*) $\{x \in s.\ f\ x \in t\}$)
unfolding *continuous-on-open-invariant openin-open Int-def vimage-def Int-commute*
by (*simp add*: *imp-ex imageI conj-commute eq-commute cong*: *conj-cong*)

Similarly in terms of closed sets.

lemma *continuous-on-closed*:
continuous-on s f $\longleftrightarrow$
$(\forall t.$ *closedin* (*subtopology euclidean* (*f* ' *s*)) *t* $\longrightarrow$
closedin (*subtopology euclidean s*) $\{x \in s.\ f\ x \in t\}$)
unfolding *continuous-on-closed-invariant closedin-closed Int-def vimage-def Int-commute*
by (*simp add*: *imp-ex imageI conj-commute eq-commute cong*: *conj-cong*)

Half-global and completely global cases.

lemma *continuous-openin-preimage*:
assumes *continuous-on s f open t*
shows *openin* (*subtopology euclidean s*) $\{x \in s.\ f\ x \in t\}$
proof −
have $*$: $\forall x.\ x \in s \wedge f\ x \in t \longleftrightarrow x \in s \wedge f\ x \in (t \cap f\ `\ s)$
by *auto*
have *openin* (*subtopology euclidean* (*f* ' *s*)) $(t \cap f\ `\ s)$
using *openin-open-Int*[*of t f* ' *s*, *OF assms*(*2*)] **unfolding** *openin-open* **by** *auto*
then show *?thesis*
using *assms*(*1*)[*unfolded continuous-on-open*, *THEN spec*[**where** $x{=}t \cap f\ `\ s$]]
using $*$ **by** *auto*
qed

lemma *continuous-closedin-preimage*:
assumes *continuous-on s f* **and** *closed t*

shows *closedin* (*subtopology euclidean s*) {*x* ∈ *s*. *f x* ∈ *t*}
proof −
have ∗: ∀ *x*. *x* ∈ *s* ∧ *f x* ∈ *t* ⟷ *x* ∈ *s* ∧ *f x* ∈ (*t* ∩ *f* ' *s*)
by *auto*
have *closedin* (*subtopology euclidean* (*f* ' *s*)) (*t* ∩ *f* ' *s*)
using *closedin-closed-Int*[*of t f* ' *s*, *OF assms*(*2*)] **unfolding** *Int-commute*
by *auto*
then show *?thesis*
using *assms*(*1*)[*unfolded continuous-on-closed*, *THEN spec*[**where** *x*=*t* ∩ *f* ' *s*]]
using ∗ **by** *auto*
qed

lemma *continuous-open-preimage*:
assumes *continuous-on s f*
and *open s*
and *open t*
shows *open* {*x* ∈ *s*. *f x* ∈ *t*}
proof−
obtain *T* **where** *T*: *open T* {*x* ∈ *s*. *f x* ∈ *t*} = *s* ∩ *T*
using *continuous-openin-preimage*[*OF assms*(*1*,*3*)] **unfolding** *openin-open* **by** *auto*
then show *?thesis*
using *open-Int*[*of s T*, *OF assms*(*2*)] **by** *auto*
qed

lemma *continuous-closed-preimage*:
assumes *continuous-on s f*
and *closed s*
and *closed t*
shows *closed* {*x* ∈ *s*. *f x* ∈ *t*}
proof−
obtain *T* **where** *closed T* {*x* ∈ *s*. *f x* ∈ *t*} = *s* ∩ *T*
using *continuous-closedin-preimage*[*OF assms*(*1*,*3*)]
unfolding *closedin-closed* **by** *auto*
then show *?thesis* **using** *closed-Int*[*of s T*, *OF assms*(*2*)] **by** *auto*
qed

lemma *continuous-open-preimage-univ*:
∀ *x*. *continuous* (*at x*) *f* ⟹ *open s* ⟹ *open* {*x*. *f x* ∈ *s*}
using *continuous-open-preimage*[*of UNIV f s*] *open-UNIV continuous-at-imp-continuous-on*
by *auto*

lemma *continuous-closed-preimage-univ*:
(∀ *x*. *continuous* (*at x*) *f*) ⟹ *closed s* ⟹ *closed* {*x*. *f x* ∈ *s*}
using *continuous-closed-preimage*[*of UNIV f s*] *closed-UNIV continuous-at-imp-continuous-on*
by *auto*

lemma *continuous-open-vimage*: ∀ *x*. *continuous* (*at x*) *f* ⟹ *open s* ⟹ *open* (*f*

$-$ ‘ s)
unfolding *vimage-def* **by** (*rule continuous-open-preimage-univ*)

lemma *continuous-closed-vimage*: $\forall x.$ *continuous* (*at* x) $f \Longrightarrow$ *closed* $s \Longrightarrow$ *closed* (f $-$ ‘ s)
unfolding *vimage-def* **by** (*rule continuous-closed-preimage-univ*)

lemma *interior-image-subset*:
assumes $\forall x.$ *continuous* (*at* x) f
and *inj* f
shows *interior* (f ‘ s) $\subseteq$ f ‘ (*interior* s)
proof
fix x **assume** $x \in$ *interior* (f ‘ s)
then obtain T **where** *as*: *open* T $x \in T$ $T \subseteq f$ ‘ s **..**
then have $x \in f$ ‘ s **by** *auto*
then obtain y **where** y: $y \in s$ $x = f\ y$ **by** *auto*
have *open* (*vimage* f T)
using *assms*(*1*) ‹*open* T› **by** (*rule continuous-open-vimage*)
moreover have $y \in$ *vimage* f T
using ‹$x = f\ y$› ‹$x \in T$› **by** *simp*
moreover have *vimage* f $T \subseteq s$
using ‹$T \subseteq$ *image* f s› ‹*inj* f› **unfolding** *inj-on-def subset-eq* **by** *auto*
ultimately have $y \in$ *interior* s **..**
with ‹$x = f\ y$› **show** $x \in f$ ‘ *interior* s **..**
qed

Equality of continuous functions on closure and related results.

lemma *continuous-closed-in-preimage-constant*:
fixes f :: - $\Rightarrow$ *'b*::*t1-space*
shows *continuous-on* s $f \Longrightarrow$ *closedin* (*subtopology euclidean* s) $\{x \in s.\ f\ x = a\}$
using *continuous-closedin-preimage*[*of* s f $\{a\}$] **by** *auto*

lemma *continuous-closed-preimage-constant*:
fixes f :: - $\Rightarrow$ *'b*::*t1-space*
shows *continuous-on* s $f \Longrightarrow$ *closed* $s \Longrightarrow$ *closed* $\{x \in s.\ f\ x = a\}$
using *continuous-closed-preimage*[*of* s f $\{a\}$] **by** *auto*

lemma *continuous-constant-on-closure*:
fixes f :: - $\Rightarrow$ *'b*::*t1-space*
assumes *continuous-on* (*closure* s) f
and $\forall x \in s.\ f\ x = a$
shows $\forall x \in$ (*closure* s). $f\ x = a$
using *continuous-closed-preimage-constant*[*of closure* s f a]
assms closure-minimal[*of* s $\{x \in$ *closure* $s.\ f\ x = a\}$] *closure-subset*
unfolding *subset-eq*
by *auto*

lemma *image-closure-subset*:
assumes *continuous-on* (*closure* s) f

```
    and closed t
    and (f ' s) ⊆ t
  shows f ' (closure s) ⊆ t
proof −
  have s ⊆ {x ∈ closure s. f x ∈ t}
    using assms(3) closure-subset by auto
  moreover have closed {x ∈ closure s. f x ∈ t}
    using continuous-closed-preimage[OF assms(1)] and assms(2) by auto
  ultimately have closure s = {x ∈ closure s . f x ∈ t}
    using closure-minimal[of s {x ∈ closure s. f x ∈ t}] by auto
  then show ?thesis by auto
qed

lemma continuous-on-closure-norm-le:
  fixes f :: 'a::metric-space ⇒ 'b::real-normed-vector
  assumes continuous-on (closure s) f
    and ∀ y ∈ s. norm(f y) ≤ b
    and x ∈ (closure s)
  shows norm (f x) ≤ b
proof −
  have ∗: f ' s ⊆ cball 0 b
    using assms(2)[unfolded mem-cball-0[symmetric]] by auto
  show ?thesis
    using image-closure-subset[OF assms(1) closed-cball[of 0 b] ∗] assms(3)
    unfolding subset-eq
    apply (erule-tac x=f x in ballE)
    apply (auto simp add: dist-norm)
    done
qed

lemma isCont-indicator:
  fixes x :: 'a::t2-space
  shows isCont (indicator A :: 'a ⇒ real) x = (x ∉ frontier A)
proof auto
  fix x
  assume cts-at: isCont (indicator A :: 'a ⇒ real) x and fr: x ∈ frontier A
  with continuous-at-open have 1: ∀ V::real set. open V ∧ indicator A x ∈ V ⟶
    (∃ U::'a set. open U ∧ x ∈ U ∧ (∀ y∈U. indicator A y ∈ V)) by auto
  show False
  proof (cases x ∈ A)
    assume x: x ∈ A
    hence indicator A x ∈ ({0<..<2} :: real set) by simp
   hence ∃ U. open U ∧ x ∈ U ∧ (∀ y∈U. indicator A y ∈ ({0<..<2} :: real set))
      using 1 open-greaterThanLessThan by blast
    then guess U .. note U = this
    hence ∀ y∈U. indicator A y > (0::real)
      unfolding greaterThanLessThan-def by auto
    hence U ⊆ A using indicator-eq-0-iff by force
    hence x ∈ interior A using U interiorI by auto
```

```
    thus ?thesis using fr unfolding frontier-def by simp
  next
    assume x: x ∉ A
    hence indicator A x ∈ ({-1<..<1} :: real set) by simp
    hence ∃ U. open U ∧ x ∈ U ∧ (∀ y∈U. indicator A y ∈ ({-1<..<1} :: real
set))
      using 1 open-greaterThanLessThan by blast
    then guess U .. note U = this
    hence ∀ y∈U. indicator A y < (1::real)
      unfolding greaterThanLessThan-def by auto
    hence U ⊆ -A by auto
    hence x ∈ interior (-A) using U interiorI by auto
    thus ?thesis using fr interior-complement unfolding frontier-def by auto
  qed
next
  assume nfr: x ∉ frontier A
  hence x ∈ interior A ∨ x ∈ interior (-A)
    by (auto simp: frontier-def closure-interior)
  thus isCont ((indicator A)::'a ⇒ real) x
  proof
    assume int: x ∈ interior A
    hence ∃ U. open U ∧ x ∈ U ∧ U ⊆ A unfolding interior-def by auto
    then guess U .. note U = this
    hence ∀ y∈U. indicator A y = (1::real) unfolding indicator-def by auto
      hence continuous-on U (indicator A) by (simp add: continuous-on-const
indicator-eq-1-iff)
    thus ?thesis using U continuous-on-eq-continuous-at by auto
  next
    assume ext: x ∈ interior (-A)
    hence ∃ U. open U ∧ x ∈ U ∧ U ⊆ -A unfolding interior-def by auto
    then guess U .. note U = this
    hence ∀ y∈U. indicator A y = (0::real) unfolding indicator-def by auto
     hence continuous-on U (indicator A) by (smt U continuous-on-topological
indicator-def)
    thus ?thesis using U continuous-on-eq-continuous-at by auto
  qed
qed
```

Making a continuous function avoid some value in a neighbourhood.

```
lemma continuous-within-avoid:
  fixes f :: 'a::metric-space ⇒ 'b::t1-space
  assumes continuous (at x within s) f
    and f x ≠ a
  shows ∃ e>0. ∀ y ∈ s. dist x y < e --> f y ≠ a
proof -
  obtain U where open U and f x ∈ U and a ∉ U
    using t1-space [OF ‹f x ≠ a›] by fast
  have (f ⟶ f x) (at x within s)
    using assms(1) by (simp add: continuous-within)
```

then have *eventually* $(\lambda y.\ f\ y \in U)$ $(at\ x\ within\ s)$
using ‹*open U*› **and** ‹$f\ x \in U$›
unfolding *tendsto-def* **by** *fast*
then have *eventually* $(\lambda y.\ f\ y \neq a)$ $(at\ x\ within\ s)$
using ‹$a \notin U$› **by** (*fast elim*: *eventually-mono*)
then show *?thesis*
using ‹$f\ x \neq a$› **by** (*auto simp*: *dist-commute zero-less-dist-iff eventually-at*)
qed

lemma *continuous-at-avoid*:
fixes f :: *'a*::*metric-space* ⇒ *'b*::*t1-space*
assumes *continuous* $(at\ x)\ f$
and $f\ x \neq a$
shows $\exists e{>}0.\ \forall y.\ dist\ x\ y < e \longrightarrow f\ y \neq a$
using *assms continuous-within-avoid*[*of x UNIV f a*] **by** *simp*

lemma *continuous-on-avoid*:
fixes f :: *'a*::*metric-space* ⇒ *'b*::*t1-space*
assumes *continuous-on s f*
and $x \in s$
and $f\ x \neq a$
shows $\exists e{>}0.\ \forall y \in s.\ dist\ x\ y < e \longrightarrow f\ y \neq a$
using *assms*(*1*)[*unfolded continuous-on-eq-continuous-within*, *THEN bspec*[**where** *x*=*x*],
OF assms(*2*)] *continuous-within-avoid*[*of x s f a*]
using *assms*(*3*)
by *auto*

lemma *continuous-on-open-avoid*:
fixes f :: *'a*::*metric-space* ⇒ *'b*::*t1-space*
assumes *continuous-on s f*
and *open s*
and $x \in s$
and $f\ x \neq a$
shows $\exists e{>}0.\ \forall y.\ dist\ x\ y < e \longrightarrow f\ y \neq a$
using *assms*(*1*)[*unfolded continuous-on-eq-continuous-at*[*OF assms*(*2*)], *THEN bspec*[**where** *x*=*x*], *OF assms*(*3*)]
using *continuous-at-avoid*[*of x f a*] *assms*(*4*)
by *auto*

Proving a function is constant by proving open-ness of level set.

lemma *continuous-levelset-open-in-cases*:
fixes f :: - ⇒ *'b*::*t1-space*
shows *connected s* ⟹ *continuous-on s f* ⟹
openin (*subtopology euclidean s*) $\{x \in s.\ f\ x = a\}$
⟹ $(\forall x \in s.\ f\ x \neq a) \vee (\forall x \in s.\ f\ x = a)$
unfolding *connected-clopen*
using *continuous-closed-in-preimage-constant* **by** *auto*

lemma *continuous-levelset-open-in*:
fixes f :: - ⇒ *'b*::*t1-space*
shows *connected* $s \Longrightarrow$ *continuous-on* $s\ f \Longrightarrow$
openin (*subtopology euclidean* s) $\{x \in s.\ f\ x = a\} \Longrightarrow$
$(\exists x \in s.\ f\ x = a) \Longrightarrow (\forall x \in s.\ f\ x = a)$
using *continuous-levelset-open-in-cases*[*of* $s\ f$]
by *meson*

lemma *continuous-levelset-open*:
fixes f :: - ⇒ *'b*::*t1-space*
assumes *connected* s
and *continuous-on* $s\ f$
and *open* $\{x \in s.\ f\ x = a\}$
and $\exists x \in s.\ \ f\ x = a$
shows $\forall x \in s.\ f\ x = a$
using *continuous-levelset-open-in*[*OF assms*(*1*,*2*), *of a*, *unfolded openin-open*]
using *assms* (*3*,*4*)
by *fast*

Some arithmetical combinations (more to prove).

lemma *open-scaling*[*intro*]:
fixes s :: *'a*::*real-normed-vector set*
assumes $c \neq 0$
and *open* s
shows *open*$((\lambda x.\ c *_R x)\ `\ s)$
proof −
{
fix x
assume $x \in s$
then obtain e **where** $e>0$
and e:$\forall x'.$ *dist* $x'\ x < e \longrightarrow x' \in s$ **using** *assms*(*2*)[*unfolded open-dist*, *THEN bspec*[**where** x=x]]
by *auto*
have $e * |c| > 0$
using *assms*(*1*)[*unfolded zero-less-abs-iff*[*symmetric*]] ‹$e>0$› **by** *auto*
moreover
{
fix y
assume *dist* $y\ (c *_R x) < e * |c|$
then have *norm* $((1\ /\ c) *_R y - x) < e$
unfolding *dist-norm*
using *norm-scaleR*[*of* $c\ (1\ /\ c) *_R y - x$, *unfolded scaleR-right-diff-distrib*, *unfolded scaleR-scaleR*] *assms*(*1*)
assms(*1*)[*unfolded zero-less-abs-iff*[*symmetric*]] **by** (*simp del*:*zero-less-abs-iff*)
then have $y \in op *_R c\ `\ s$
using *rev-image-eqI*[*of* $(1\ /\ c) *_R y\ s\ y\ op *_R c$]
using e[*THEN spec*[**where** x=$(1\ /\ c) *_R y$]]
using *assms*(*1*)
unfolding *dist-norm scaleR-scaleR*

```
      by auto
    }
    ultimately have ∃ e>0. ∀ x'. dist x' (c *R x) < e ⟶ x' ∈ op *R c ` s
      apply (rule-tac x=e * |c| in exI)
      apply auto
      done
  }
  then show ?thesis unfolding open-dist by auto
qed

lemma minus-image-eq-vimage:
  fixes A :: 'a::ab-group-add set
  shows (λx. - x) ` A = (λx. - x) -` A
  by (auto intro!: image-eqI [where f=λx. - x])

lemma open-negations:
  fixes s :: 'a::real-normed-vector set
  shows open s ⟹ open ((λx. - x) ` s)
  using open-scaling [of - 1 s] by simp

lemma open-translation:
  fixes s :: 'a::real-normed-vector set
  assumes open s
  shows open((λx. a + x) ` s)
proof -
  {
    fix x
    have continuous (at x) (λx. x - a)
      by (intro continuous-diff continuous-at-id continuous-const)
  }
  moreover have {x. x - a ∈ s} = op + a ` s
    by force
  ultimately show ?thesis using continuous-open-preimage-univ[of λx. x - a s]
    using assms by auto
qed

lemma open-affinity:
  fixes s :: 'a::real-normed-vector set
  assumes open s  c ≠ 0
  shows open ((λx. a + c *R x) ` s)
proof -
  have *: (λx. a + c *R x) = (λx. a + x) ∘ (λx. c *R x)
    unfolding o-def ..
  have op + a ` op *R c ` s = (op + a ∘ op *R c) ` s
    by auto
  then show ?thesis
    using assms open-translation[of op *R c ` s a]
    unfolding *
    by auto
```

qed

lemma *interior-translation*:
fixes $s :: 'a$::*real-normed-vector set*
shows *interior* $((\lambda x.\ a + x)\ `\ s) = (\lambda x.\ a + x)\ `\ (\mathit{interior}\ s)$
proof (*rule set-eqI*, *rule*)
fix x
assume $x \in \mathit{interior}\ (\mathit{op} + a\ `\ s)$
then obtain e **where** $e > 0$ **and** e: *ball* $x\ e \subseteq \mathit{op} + a\ `\ s$
unfolding *mem-interior* **by** *auto*
then have *ball* $(x - a)\ e \subseteq s$
unfolding *subset-eq Ball-def mem-ball dist-norm*
by (*auto simp add*: *diff-diff-eq*)
then show $x \in \mathit{op} + a\ `\ \mathit{interior}\ s$
unfolding *image-iff*
apply (*rule-tac* $x{=}x - a$ **in** *bexI*)
unfolding *mem-interior*
using ‹$e > 0$›
apply *auto*
done
next
fix x
assume $x \in \mathit{op} + a\ `\ \mathit{interior}\ s$
then obtain $y\ e$ **where** $e > 0$ **and** e: *ball* $y\ e \subseteq s$ **and** y: $x = a + y$
unfolding *image-iff Bex-def mem-interior* **by** *auto*
{
fix z
have $*$: $a + y - z = y + a - z$ **by** *auto*
assume $z \in \mathit{ball}\ x\ e$
then have $z - a \in s$
using e[*unfolded subset-eq*, *THEN bspec*[**where** $x{=}z - a$]]
unfolding *mem-ball dist-norm y group-add-class.diff-diff-eq2* $*$
by *auto*
then have $z \in \mathit{op} + a\ `\ s$
unfolding *image-iff* **by** (*auto intro*!: *bexI*[**where** $x{=}z - a$])
}
then have *ball* $x\ e \subseteq \mathit{op} + a\ `\ s$
unfolding *subset-eq* **by** *auto*
then show $x \in \mathit{interior}\ (\mathit{op} + a\ `\ s)$
unfolding *mem-interior* **using** ‹$e > 0$› **by** *auto*
qed

Topological properties of linear functions.

lemma *linear-lim-0*:
assumes *bounded-linear f*
shows $(f \longrightarrow 0)\ (\mathit{at}\ (0))$
proof −
interpret f: *bounded-linear f* **by** *fact*
have $(f \longrightarrow f\ 0)\ (\mathit{at}\ 0)$

using *tendsto-ident-at* **by** (*rule f.tendsto*)
then show *?thesis* **unfolding** *f.zero* **.**
qed

lemma *linear-continuous-at*:
assumes *bounded-linear f*
shows *continuous* (*at a*) *f*
unfolding *continuous-at* **using** *assms*
apply (*rule bounded-linear.tendsto*)
apply (*rule tendsto-ident-at*)
done

lemma *linear-continuous-within*:
bounded-linear f $\Longrightarrow$ *continuous* (*at x within s*) *f*
using *continuous-at-imp-continuous-within*[*of x f s*] **using** *linear-continuous-at*[*of f*] **by** *auto*

lemma *linear-continuous-on*:
bounded-linear f $\Longrightarrow$ *continuous-on s f*
using *continuous-at-imp-continuous-on*[*of s f*] **using** *linear-continuous-at*[*of f*] **by** *auto*

Also bilinear functions, in composition form.

lemma *bilinear-continuous-at-compose*:
continuous (*at x*) *f* $\Longrightarrow$ *continuous* (*at x*) *g* $\Longrightarrow$ *bounded-bilinear h* $\Longrightarrow$
continuous (*at x*) ($\lambda x.$ *h* (*f x*) (*g x*))
unfolding *continuous-at*
using *Lim-bilinear*[*of f f x* (*at x*) *g g x h*]
by *auto*

lemma *bilinear-continuous-within-compose*:
continuous (*at x within s*) *f* $\Longrightarrow$ *continuous* (*at x within s*) *g* $\Longrightarrow$ *bounded-bilinear h* $\Longrightarrow$
continuous (*at x within s*) ($\lambda x.$ *h* (*f x*) (*g x*))
by (*rule Limits.bounded-bilinear.continuous*)

lemma *bilinear-continuous-on-compose*:
continuous-on s f $\Longrightarrow$ *continuous-on s g* $\Longrightarrow$ *bounded-bilinear h* $\Longrightarrow$
continuous-on s ($\lambda x.$ *h* (*f x*) (*g x*))
by (*rule Limits.bounded-bilinear.continuous-on*)

Preservation of compactness and connectedness under continuous function.

lemma *compact-eq-openin-cover*:
compact S $\longleftrightarrow$
($\forall C.$ ($\forall c{\in}C.$ *openin* (*subtopology euclidean S*) *c*) $\wedge$ $S \subseteq \bigcup C \longrightarrow$
($\exists D{\subseteq}C.$ *finite D* $\wedge$ $S \subseteq \bigcup D$))
proof *safe*
fix *C*
assume *compact S* **and** $\forall c{\in}C.$ *openin* (*subtopology euclidean S*) *c* **and** $S \subseteq \bigcup C$

then have $\forall\, c{\in}\{T.\ \mathit{open}\ T \wedge S \cap T \in C\}.\ \mathit{open}\ c$ **and** $S \subseteq \bigcup\{T.\ \mathit{open}\ T \wedge S \cap T \in C\}$
unfolding *openin-open* **by** *force+*
with ‹*compact S*› **obtain** D **where** $D \subseteq \{T.\ \mathit{open}\ T \wedge S \cap T \in C\}$ **and** *finite* D **and** $S \subseteq \bigcup D$
by (*rule compactE*)
then have *image* $(\lambda T.\ S \cap T)\ D \subseteq C \wedge$ *finite* (*image* $(\lambda T.\ S \cap T)\ D) \wedge S \subseteq \bigcup$(*image* $(\lambda T.\ S \cap T)\ D$)
by *auto*
then show $\exists\, D{\subseteq}C.$ *finite* $D \wedge S \subseteq \bigcup D$ **..**
next
assume *1*: $\forall\, C.\ (\forall\, c{\in}C.\ \mathit{openin}\ (\mathit{subtopology}\ \mathit{euclidean}\ S)\ c) \wedge S \subseteq \bigcup C \longrightarrow (\exists\, D{\subseteq}C.\ \mathit{finite}\ D \wedge S \subseteq \bigcup D)$
show *compact S*
proof (*rule compactI*)
fix C
let *?C* = *image* $(\lambda T.\ S \cap T)\ C$
assume $\forall\, t{\in}C.\ \mathit{open}\ t$ **and** $S \subseteq \bigcup C$
then have $(\forall\, c{\in}?C.\ \mathit{openin}\ (\mathit{subtopology}\ \mathit{euclidean}\ S)\ c) \wedge S \subseteq \bigcup ?C$
unfolding *openin-open* **by** *auto*
with *1* **obtain** D **where** $D \subseteq ?C$ **and** *finite D* **and** $S \subseteq \bigcup D$
by *metis*
let *?D* = *inv-into C* $(\lambda T.\ S \cap T)$ ‘ D
have *?D* $\subseteq C \wedge$ *finite ?D* $\wedge\ S \subseteq \bigcup ?D$
proof (*intro conjI*)
from ‹$D \subseteq ?C$› **show** *?D* $\subseteq C$
by (*fast intro*: *inv-into-into*)
from ‹*finite D*› **show** *finite ?D*
by (*rule finite-imageI*)
from ‹$S \subseteq \bigcup D$› **show** $S \subseteq \bigcup ?D$
apply (*rule subset-trans*)
apply *clarsimp*
apply (*frule subsetD* [*OF* ‹$D \subseteq ?C$›, *THEN f-inv-into-f*])
apply (*erule rev-bexI*, *fast*)
done
qed
then show $\exists\, D{\subseteq}C.$ *finite* $D \wedge S \subseteq \bigcup D$ **..**
qed
qed

lemma *connected-continuous-image*:
assumes *continuous-on s f*
and *connected s*
shows *connected*(*f* ‘ *s*)
proof −
{
fix T
assume *as*:
$T \neq \{\}$

```
      T ≠ f ‘ s
      openin (subtopology euclidean (f ‘ s)) T
      closedin (subtopology euclidean (f ‘ s)) T
    have {x ∈ s. f x ∈ T} = {} ∨ {x ∈ s. f x ∈ T} = s
      using assms(1)[unfolded continuous-on-open, THEN spec[where x=T]]
      using assms(1)[unfolded continuous-on-closed, THEN spec[where x=T]]
      using assms(2)[unfolded connected-clopen, THEN spec[where x={x ∈ s. f x
∈ T}]] as(3,4) by auto
    then have False using as(1,2)
      using as(4)[unfolded closedin-def topspace-euclidean-subtopology] by auto
  }
  then show ?thesis
    unfolding connected-clopen by auto
qed

lemma connected-linear-image:
  fixes f :: 'a::euclidean-space ⇒ 'b::real-normed-vector
  assumes linear f and connected s
  shows connected (f ‘ s)
using connected-continuous-image assms linear-continuous-on linear-conv-bounded-linear
by blast
```

Continuity implies uniform continuity on a compact domain.

```
lemma compact-uniformly-continuous:
  fixes f :: 'a :: metric-space ⇒ 'b :: metric-space
  assumes f: continuous-on s f
    and s: compact s
  shows uniformly-continuous-on s f
  unfolding uniformly-continuous-on-def
proof (cases, safe)
  fix e :: real
  assume 0 < e s ≠ {}
  def [simp]: R ≡ {(y, d). y ∈ s ∧ 0 < d ∧ ball y d ∩ s ⊆ {x ∈ s. f x ∈ ball (f
y) (e/2) } }
  let ?b = (λ(y, d). ball y (d/2))
  have (∀ r∈R. open (?b r)) s ⊆ (⋃ r∈R. ?b r)
  proof safe
    fix y
    assume y ∈ s
    from continuous-openin-preimage[OF f open-ball]
    obtain T where open T and T: {x ∈ s. f x ∈ ball (f y) (e/2)} = T ∩ s
      unfolding openin-subtopology open-openin by metis
    then obtain d where ball y d ⊆ T 0 < d
      using ‹0 < e› ‹y ∈ s› by (auto elim!: openE)
    with T ‹y ∈ s› show y ∈ (⋃ r∈R. ?b r)
      by (intro UN-I[of (y, d)]) auto
  qed auto
  with s obtain D where D: finite D D ⊆ R s ⊆ (⋃ (y, d)∈D. ball y (d/2))
    by (rule compactE-image)
```

```
  with ‹s ≠ {}› have [simp]: ⋀x. x < Min (snd ‘ D) ⟷ (∀(y, d)∈D. x < d)
    by (subst Min-gr-iff) auto
  show ∃d>0. ∀x∈s. ∀x'∈s. dist x' x < d ⟶ dist (f x') (f x) < e
  proof (rule, safe)
    fix x x'
    assume in-s: x' ∈ s x ∈ s
    with D obtain y d where x: x ∈ ball y (d/2) (y, d) ∈ D
      by blast
    moreover assume dist x x' < Min (snd‘D) / 2
    ultimately have dist y x' < d
      by (intro dist-double[where x=x and d=d]) (auto simp: dist-commute)
    with D x in-s show  dist (f x) (f x') < e
       by (intro dist-double[where x=f y and d=e]) (auto simp: dist-commute
subset-eq)
  qed (insert D, auto)
qed auto
```

A uniformly convergent limit of continuous functions is continuous.

```
lemma continuous-uniform-limit:
  fixes f :: 'a ⇒ 'b::metric-space ⇒ 'c::metric-space
  assumes ¬ trivial-limit F
    and eventually (λn. continuous-on s (f n)) F
    and ∀e>0. eventually (λn. ∀x∈s. dist (f n x) (g x) < e) F
  shows continuous-on s g
proof −
  {
    fix x and e :: real
    assume x∈s e>0
    have eventually (λn. ∀x∈s. dist (f n x) (g x) < e / 3) F
      using ‹e>0› assms(3)[THEN spec[where x=e/3]] by auto
    from eventually-happens [OF eventually-conj [OF this assms(2)]]
    obtain n where n:∀x∈s. dist (f n x) (g x) < e / 3  continuous-on s (f n)
      using assms(1) by blast
    have e / 3 > 0 using ‹e>0› by auto
    then obtain d where d>0 and d:∀x'∈s. dist x' x < d ⟶ dist (f n x') (f n
x) < e / 3
      using n(2)[unfolded continuous-on-iff, THEN bspec[where x=x], OF ‹x∈s›,
THEN spec[where x=e/3]] by blast
    {
      fix y
      assume y ∈ s and dist y x < d
      then have dist (f n y) (f n x) < e / 3
        by (rule d [rule-format])
      then have dist (f n y) (g x) < 2 ∗ e / 3
        using dist-triangle [of f n y g x f n x]
        using n(1)[THEN bspec[where x=x], OF ‹x∈s›]
        by auto
      then have dist (g y) (g x) < e
        using n(1)[THEN bspec[where x=y], OF ‹y∈s›]
```

using *dist-triangle3* [*of g y g x f n y*]
by *auto*
}
then have ∃ *d*>*0*. ∀ *x'*∈*s*. *dist x' x* < *d* ⟶ *dist* (*g x'*) (*g x*) < *e*
using ‹*d*>*0*› **by** *auto*
}
then show *?thesis*
unfolding *continuous-on-iff* **by** *auto*
qed

16.24 Topological stuff lifted from and dropped to R

lemma *open-real*:
fixes *s* :: *real set*
shows *open s* ⟷ (∀ *x* ∈ *s*. ∃ *e*>*0*. ∀ *x'*. |*x'* − *x*| < *e* −−> *x'* ∈ *s*)
unfolding *open-dist dist-norm* **by** *simp*

lemma *islimpt-approachable-real*:
fixes *s* :: *real set*
shows *x islimpt s* ⟷ (∀ *e*>*0*. ∃ *x'*∈ *s*. *x'* ≠ *x* ∧ |*x'* − *x*| < *e*)
unfolding *islimpt-approachable dist-norm* **by** *simp*

lemma *closed-real*:
fixes *s* :: *real set*
shows *closed s* ⟷ (∀ *x*. (∀ *e*>*0*. ∃ *x'* ∈ *s*. *x'* ≠ *x* ∧ |*x'* − *x*| < *e*) ⟶ *x* ∈ *s*)
unfolding *closed-limpt islimpt-approachable dist-norm* **by** *simp*

lemma *continuous-at-real-range*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *real*
shows *continuous* (*at x*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. ∀ *x'*. *norm*(*x'* − *x*) < *d* −−> |*f x'* − *f x*| < *e*)
unfolding *continuous-at*
unfolding *Lim-at*
unfolding *dist-norm*
apply *auto*
apply (*erule-tac x*=*e* **in** *allE*)
apply *auto*
apply (*rule-tac x*=*d* **in** *exI*)
apply *auto*
apply (*erule-tac x*=*x'* **in** *allE*)
apply *auto*
apply (*erule-tac x*=*e* **in** *allE*)
apply *auto*
done

lemma *continuous-on-real-range*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *real*
shows *continuous-on s f* ⟷
(∀ *x* ∈ *s*. ∀ *e*>*0*. ∃ *d*>*0*. (∀ *x'* ∈ *s*. *norm*(*x'* − *x*) < *d* ⟶ |*f x'* − *f x*| < *e*))

unfolding *continuous-on-iff dist-norm* **by** *simp*

Hence some handy theorems on distance, diameter etc. of/from a set.

lemma *distance-attains-sup*:
assumes *compact s* $s \neq \{\}$
shows $\exists x{\in}s.\ \forall y{\in}s.\ dist\ a\ y \leq dist\ a\ x$
proof (*rule continuous-attains-sup* [*OF assms*])
{
fix *x*
assume $x{\in}s$
have (*dist a* $\longrightarrow$ *dist a x*) (*at x within s*)
by (*intro tendsto-dist tendsto-const tendsto-ident-at*)
}
then show *continuous-on s* (*dist a*)
unfolding *continuous-on* **..**
qed

For *minimal* distance, we only need closure, not compactness.

lemma *distance-attains-inf*:
fixes *a* :: $'a$::*heine-borel*
assumes *closed s*
and $s \neq \{\}$
shows $\exists x{\in}s.\ \forall y{\in}s.\ dist\ a\ x \leq dist\ a\ y$
proof −
from *assms*(*2*) **obtain** *b* **where** $b \in s$ **by** *auto*
let *?B* = $s \cap$ *cball a* (*dist b a*)
have *?B* $\neq \{\}$ **using** ‹$b \in s$›
by (*auto simp add*: *dist-commute*)
moreover have *continuous-on ?B* (*dist a*)
by (*auto intro*!: *continuous-at-imp-continuous-on continuous-dist continuous-at-id continuous-const*)
moreover have *compact ?B*
by (*intro closed-inter-compact* ‹*closed s*› *compact-cball*)
ultimately obtain *x* **where** $x \in$ *?B* $\forall y{\in}$*?B*. *dist a x* $\leq$ *dist a y*
by (*metis continuous-attains-inf*)
then show *?thesis* **by** *fastforce*
qed

16.25 Pasted sets

lemma *bounded-Times*:
assumes *bounded s bounded t*
shows *bounded* ($s \times t$)
proof −
obtain *x y a b* **where** $\forall z{\in}s.\ dist\ x\ z \leq a\ \forall z{\in}t.\ dist\ y\ z \leq b$
using *assms* [*unfolded bounded-def*] **by** *auto*
then have $\forall z{\in}s \times t.\ dist\ (x,\ y)\ z \leq sqrt\ (a^2 + b^2)$
by (*auto simp add*: *dist-Pair-Pair real-sqrt-le-mono add-mono power-mono*)
then show *?thesis* **unfolding** *bounded-any-center* [**where** $a{=}(x,\ y)$] **by** *auto*

qed

lemma *mem-Times-iff*: $x \in A \times B \longleftrightarrow fst\ x \in A \wedge snd\ x \in B$
by (*induct x*) *simp*

lemma *seq-compact-Times*: *seq-compact s* $\Longrightarrow$ *seq-compact t* $\Longrightarrow$ *seq-compact* (*s* $\times$ *t*)
unfolding *seq-compact-def*
apply *clarify*
apply (*drule-tac x=fst* $\circ$ *f* **in** *spec*)
apply (*drule mp*, *simp add*: *mem-Times-iff*)
apply (*clarify*, *rename-tac l1 r1*)
apply (*drule-tac x=snd* $\circ$ *f* $\circ$ *r1* **in** *spec*)
apply (*drule mp*, *simp add*: *mem-Times-iff*)
apply (*clarify*, *rename-tac l2 r2*)
apply (*rule-tac x=*(*l1*, *l2*) **in** *rev-bexI*, *simp*)
apply (*rule-tac x=r1* $\circ$ *r2* **in** *exI*)
apply (*rule conjI*, *simp add*: *subseq-def*)
apply (*drule-tac f=r2* **in** *LIMSEQ-subseq-LIMSEQ*, *assumption*)
apply (*drule* (*1*) *tendsto-Pair*) **back**
apply (*simp add*: *o-def*)
done

lemma *compact-Times*:
assumes *compact s compact t*
shows *compact* (*s* $\times$ *t*)
proof (*rule compactI*)
fix *C*
assume *C*: $\forall t{\in}C.\ open\ t\ s \times t \subseteq \bigcup C$
have $\forall x{\in}s.\ \exists a.\ open\ a \wedge x \in a \wedge (\exists d{\subseteq}C.\ finite\ d \wedge a \times t \subseteq \bigcup d)$
proof
fix *x*
assume $x \in s$
have $\forall y{\in}t.\ \exists a\ b\ c.\ c \in C \wedge open\ a \wedge open\ b \wedge x \in a \wedge y \in b \wedge a \times b \subseteq c$ (**is** $\forall y{\in}t.\ ?P\ y$)
proof
fix *y*
assume $y \in t$
with ‹$x \in s$› *C* **obtain** *c* **where** $c \in C$ $(x, y) \in c$ *open c* **by** *auto*
then show *?P y* **by** (*auto elim*!: *open-prod-elim*)
qed
then obtain *a b c* **where** *b*: $\bigwedge y.\ y \in t \Longrightarrow open\ (b\ y)$
and *c*: $\bigwedge y.\ y \in t \Longrightarrow c\ y \in C \wedge open\ (a\ y) \wedge open\ (b\ y) \wedge x \in a\ y \wedge y \in b\ y \wedge a\ y \times b\ y \subseteq c\ y$
by *metis*
then have $\forall y{\in}t.\ open\ (b\ y)\ t \subseteq (\bigcup y{\in}t.\ b\ y)$ **by** *auto*
from *compactE-image*[*OF* ‹*compact t*› *this*] **obtain** *D* **where** *D*: $D \subseteq t$ *finite* $D\ t \subseteq (\bigcup y{\in}D.\ b\ y)$
by *auto*

```
    moreover from D c have (⋂y∈D. a y) × t ⊆ (⋃y∈D. c y)
      by (fastforce simp: subset-eq)
    ultimately show ∃a. open a ∧ x ∈ a ∧ (∃d⊆C. finite d ∧ a × t ⊆ ⋃d)
     using c by (intro exI[of - c‘D] exI[of - ⋂(a‘D)] conjI) (auto intro!: open-INT)
  qed
  then obtain a d where a: ∀x∈s. open (a x) s ⊆ (⋃x∈s. a x)
    and d: ⋀x. x ∈ s ⟹ d x ⊆ C ∧ finite (d x) ∧ a x × t ⊆ ⋃d x
    unfolding subset-eq UN-iff by metis
  moreover
  from compactE-image[OF ‹compact s› a]
  obtain e where e: e ⊆ s finite e and s: s ⊆ (⋃x∈e. a x)
    by auto
  moreover
  {
    from s have s × t ⊆ (⋃x∈e. a x × t)
      by auto
    also have ... ⊆ (⋃x∈e. ⋃d x)
      using d ‹e ⊆ s› by (intro UN-mono) auto
    finally have s × t ⊆ (⋃x∈e. ⋃d x) .
  }
  ultimately show ∃C'⊆C. finite C' ∧ s × t ⊆ ⋃C'
    by (intro exI[of - (⋃x∈e. d x)]) (auto simp add: subset-eq)
qed
```

Hence some useful properties follow quite easily.

```
lemma compact-scaling:
  fixes s :: 'a::real-normed-vector set
  assumes compact s
  shows compact ((λx. c *R x) ‘ s)
proof −
  let ?f = λx. scaleR c x
  have *: bounded-linear ?f by (rule bounded-linear-scaleR-right)
  show ?thesis
    using compact-continuous-image[of s ?f] continuous-at-imp-continuous-on[of s
?f]
    using linear-continuous-at[OF *] assms
    by auto
qed

lemma compact-negations:
  fixes s :: 'a::real-normed-vector set
  assumes compact s
  shows compact ((λx. − x) ‘ s)
  using compact-scaling [OF assms, of − 1] by auto

lemma compact-sums:
  fixes s t :: 'a::real-normed-vector set
  assumes compact s
    and compact t
```

shows *compact* $\{x + y \mid x\ y.\ x \in s \wedge y \in t\}$
proof −
have $*$: $\{x + y \mid x\ y.\ x \in s \wedge y \in t\} = (\lambda z.\ fst\ z + snd\ z)$ ‘ $(s \times t)$
apply *auto*
unfolding *image-iff*
apply (*rule-tac* x=(xa, y) **in** *bexI*)
apply *auto*
done
have *continuous-on* $(s \times t)$ $(\lambda z.\ fst\ z + snd\ z)$
unfolding *continuous-on* **by** (*rule ballI*) (*intro tendsto-intros*)
then show *?thesis*
unfolding $*$ **using** *compact-continuous-image compact-Times* [*OF assms*] **by** *auto*
qed

lemma *compact-differences*:
fixes $s\ t$ :: $'a$::*real-normed-vector set*
assumes *compact s*
and *compact t*
shows *compact* $\{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
proof−
have $\{x - y \mid x\ y.\ x{\in}s \wedge y \in t\} = \{x + y \mid x\ y.\ x \in s \wedge y \in (uminus\ ‘\ t)\}$
apply *auto*
apply (*rule-tac* x= *xa* **in** *exI*)
apply *auto*
done
then show *?thesis*
using *compact-sums*[*OF assms*(*1*) *compact-negations*[*OF assms*(*2*)]] **by** *auto*
qed

lemma *compact-translation*:
fixes s :: $'a$::*real-normed-vector set*
assumes *compact s*
shows *compact* $((\lambda x.\ a + x)\ ‘\ s)$
proof −
have $\{x + y \mid x\ y.\ x \in s \wedge y \in \{a\}\} = (\lambda x.\ a + x)\ ‘\ s$
by *auto*
then show *?thesis*
using *compact-sums*[*OF assms compact-sing*[*of a*]] **by** *auto*
qed

lemma *compact-affinity*:
fixes s :: $'a$::*real-normed-vector set*
assumes *compact s*
shows *compact* $((\lambda x.\ a + c *_R x)\ ‘\ s)$
proof −
have $op + a\ ‘\ op *_R\ c\ ‘\ s = (\lambda x.\ a + c *_R x)\ ‘\ s$
by *auto*
then show *?thesis*

using *compact-translation*[*OF compact-scaling*[*OF assms*], *of a c*] **by** *auto*
qed

Hence we get the following.

lemma *compact-sup-maxdistance*:
fixes s :: *'a*::*metric-space set*
assumes *compact s*
and $s \neq \{\}$
shows $\exists x{\in}s.\ \exists y{\in}s.\ \forall u{\in}s.\ \forall v{\in}s.\ dist\ u\ v \leq dist\ x\ y$
proof −
have *compact* $(s \times s)$
using ‹*compact s*› **by** (*intro compact-Times*)
moreover have $s \times s \neq \{\}$
using ‹$s \neq \{\}$› **by** *auto*
moreover have *continuous-on* $(s \times s)$ $(\lambda x.\ dist\ (fst\ x)\ (snd\ x))$
by (*intro continuous-at-imp-continuous-on ballI continuous-intros*)
ultimately show *?thesis*
using *continuous-attains-sup*[*of* $s \times s$ $\lambda x.\ dist\ (fst\ x)\ (snd\ x)$] **by** *auto*
qed

We can state this in terms of diameter of a set.

definition *diameter* :: *'a*::*metric-space set* ⇒ *real* **where**
diameter S = (*if S* = {} *then 0 else SUP* (*x*,*y*):*S*×*S*. *dist x y*)

lemma *diameter-bounded-bound*:
fixes s :: *'a* :: *metric-space set*
assumes *s*: *bounded s x* ∈ *s y* ∈ *s*
shows *dist x y* ≤ *diameter s*
proof −
from *s* **obtain** *z d* **where** *z*: $\bigwedge x.\ x \in s \Longrightarrow dist\ z\ x \leq d$
unfolding *bounded-def* **by** *auto*
have *bdd-above* (*case-prod dist* ‘ (*s*×*s*))
proof (*intro bdd-aboveI*, *safe*)
fix *a b*
assume *a* ∈ *s b* ∈ *s*
with *z*[*of a*] *z*[*of b*] *dist-triangle*[*of a b z*]
show *dist a b* ≤ *2* ∗ *d*
by (*simp add*: *dist-commute*)
qed
moreover have (*x*,*y*) ∈ *s*×*s* **using** *s* **by** *auto*
ultimately have *dist x y* ≤ (*SUP* (*x*,*y*):*s*×*s*. *dist x y*)
by (*rule cSUP-upper2*) *simp*
with ‹*x* ∈ *s*› **show** *?thesis*
by (*auto simp add*: *diameter-def*)
qed

lemma *diameter-lower-bounded*:
fixes s :: *'a* :: *metric-space set*
assumes *s*: *bounded s*

```
    and d: 0 < d d < diameter s
  shows ∃ x∈s. ∃ y∈s. d < dist x y
proof (rule ccontr)
  assume contr: ¬ ?thesis
  moreover have s ≠ {}
    using d by (auto simp add: diameter-def)
  ultimately have diameter s ≤ d
    by (auto simp: not-less diameter-def intro!: cSUP-least)
  with ‹d < diameter s› show False by auto
qed

lemma diameter-bounded:
  assumes bounded s
  shows ∀ x∈s. ∀ y∈s. dist x y ≤ diameter s
    and ∀ d>0. d < diameter s ⟶ (∃ x∈s. ∃ y∈s. dist x y > d)
  using diameter-bounded-bound[of s] diameter-lower-bounded[of s] assms
  by auto

lemma diameter-compact-attained:
  assumes compact s
    and s ≠ {}
  shows ∃ x∈s. ∃ y∈s. dist x y = diameter s
proof −
  have b: bounded s using assms(1)
    by (rule compact-imp-bounded)
  then obtain x y where xys: x∈s y∈s
    and xy: ∀ u∈s. ∀ v∈s. dist u v ≤ dist x y
    using compact-sup-maxdistance[OF assms] by auto
  then have diameter s ≤ dist x y
    unfolding diameter-def
    apply clarsimp
    apply (rule cSUP-least)
    apply fast+
    done
  then show ?thesis
    by (metis b diameter-bounded-bound order-antisym xys)
qed
```

Related results with closure as the conclusion.

```
lemma closed-scaling:
  fixes s :: 'a::real-normed-vector set
  assumes closed s
  shows closed ((λx. c *R x) ‘ s)
proof (cases c = 0)
  case True then show ?thesis
    by (auto simp add: image-constant-conv)
next
  case False
  from assms have closed ((λx. inverse c *R x) −‘ s)
```

```
    by (simp add: continuous-closed-vimage)
  also have (λx. inverse c *R x) −‘ s = (λx. c *R x) ‘ s
    using ‹c ≠ 0› by (auto elim: image-eqI [rotated])
  finally show ?thesis .
qed

lemma closed-negations:
  fixes s :: 'a::real-normed-vector set
  assumes closed s
  shows closed ((λx. −x) ‘ s)
  using closed-scaling[OF assms, of − 1] by simp

lemma compact-closed-sums:
  fixes s :: 'a::real-normed-vector set
  assumes compact s and closed t
  shows closed {x + y | x y. x ∈ s ∧ y ∈ t}
proof −
  let ?S = {x + y |x y. x ∈ s ∧ y ∈ t}
  {
    fix x l
    assume as: ∀ n. x n ∈ ?S  (x ⟶ l) sequentially
    from as(1) obtain f where f: ∀ n. x n = fst (f n) + snd (f n)  ∀ n. fst (f n)
∈ s  ∀ n. snd (f n) ∈ t
      using choice[of λn y. x n = (fst y) + (snd y) ∧ fst y ∈ s ∧ snd y ∈ t] by
auto
    obtain l' r where l'∈s and r: subseq r and lr: (((λn. fst (f n)) ∘ r) ⟶ l')
sequentially
      using assms(1)[unfolded compact-def, THEN spec[where x=λ n. fst (f n)]]
using f(2) by auto
    have ((λn. snd (f (r n))) ⟶ l − l') sequentially
      using tendsto-diff[OF LIMSEQ-subseq-LIMSEQ[OF as(2) r] lr] and f(1)
      unfolding o-def
      by auto
    then have l − l' ∈ t
      using assms(2)[unfolded closed-sequential-limits,
        THEN spec[where x=λ n. snd (f (r n))],
        THEN spec[where x=l − l']]
      using f(3)
      by auto
    then have l ∈ ?S
      using ‹l' ∈ s›
      apply auto
      apply (rule-tac x=l' in exI)
      apply (rule-tac x=l − l' in exI)
      apply auto
      done
  }
  then show ?thesis
    unfolding closed-sequential-limits by fast
```

qed

lemma *closed-compact-sums*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *closed s*
and *compact t*
shows *closed* $\{x + y \mid x\ y.\ x \in s \wedge y \in t\}$
proof −
have $\{x + y \mid x\ y.\ x \in t \wedge y \in s\} = \{x + y \mid x\ y.\ x \in s \wedge y \in t\}$
apply *auto*
apply (*rule-tac x*=*y* **in** *exI*)
apply *auto*
apply (*rule-tac x*=*y* **in** *exI*)
apply *auto*
done
then show *?thesis*
using *compact-closed-sums*[*OF assms*(*2*,*1*)] **by** *simp*
qed

lemma *compact-closed-differences*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *compact s*
and *closed t*
shows *closed* $\{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
proof −
have $\{x + y \mid x\ y.\ x \in s \wedge y \in$ *uminus* ‘ $t\} = \{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
apply *auto*
apply (*rule-tac x*=*xa* **in** *exI*)
apply *auto*
apply (*rule-tac x*=*xa* **in** *exI*)
apply *auto*
done
then show *?thesis*
using *compact-closed-sums*[*OF assms*(*1*) *closed-negations*[*OF assms*(*2*)]] **by** *auto*
qed

lemma *closed-compact-differences*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *closed s*
and *compact t*
shows *closed* $\{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
proof −
have $\{x + y \mid x\ y.\ x \in s \wedge y \in$ *uminus* ‘ $t\} = \{x - y \mid x\ y.\ x \in s \wedge y \in t\}$
apply *auto*
apply (*rule-tac x*=*xa* **in** *exI*)
apply *auto*
apply (*rule-tac x*=*xa* **in** *exI*)
apply *auto*

```
    done
 then show ?thesis
  using closed-compact-sums[OF assms(1) compact-negations[OF assms(2)]] by
simp
qed

lemma closed-translation:
  fixes a :: 'a::real-normed-vector
  assumes closed s
  shows closed ((λx. a + x) ‘ s)
proof −
  have {a + y |y. y ∈ s} = (op + a ‘ s) by auto
  then show ?thesis
    using compact-closed-sums[OF compact-sing[of a] assms] by auto
qed

lemma translation-Compl:
  fixes a :: 'a::ab-group-add
  shows (λx. a + x) ‘ (− t) = − ((λx. a + x) ‘ t)
  apply (auto simp add: image-iff)
  apply (rule-tac x=x − a in bexI)
  apply auto
  done

lemma translation-UNIV:
  fixes a :: 'a::ab-group-add
  shows range (λx. a + x) = UNIV
  apply (auto simp add: image-iff)
  apply (rule-tac x=x − a in exI)
  apply auto
  done

lemma translation-diff:
  fixes a :: 'a::ab-group-add
  shows (λx. a + x) ‘ (s − t) = ((λx. a + x) ‘ s) − ((λx. a + x) ‘ t)
  by auto

lemma closure-translation:
  fixes a :: 'a::real-normed-vector
  shows closure ((λx. a + x) ‘ s) = (λx. a + x) ‘ (closure s)
proof −
  have ∗: op + a ‘ (− s) = − op + a ‘ s
    apply auto
    unfolding image-iff
    apply (rule-tac x=x − a in bexI)
    apply auto
    done
  show ?thesis
    unfolding closure-interior translation-Compl
```

using *interior-translation*[*of* $a - s$]
unfolding $*$
by *auto*
qed

lemma *frontier-translation*:
fixes $a :: 'a$::*real-normed-vector*
shows *frontier*$((\lambda x.\ a + x)\ `\ s) = (\lambda x.\ a + x)\ `\ (\mathit{frontier}\ s)$
unfolding *frontier-def translation-diff interior-translation closure-translation*
by *auto*

16.26 Separation between points and sets

lemma *separate-point-closed*:
fixes $s :: 'a$::*heine-borel set*
assumes *closed s*
and $a \notin s$
shows $\exists d{>}0.\ \forall x{\in}s.\ d \leq \mathit{dist}\ a\ x$
proof (*cases* $s = \{\}$)
case *True*
then show *?thesis* **by**(*auto intro*!: *exI*[**where** *x=1*])
next
case *False*
from *assms* **obtain** x **where** $x{\in}s\ \forall y{\in}s.\ \mathit{dist}\ a\ x \leq \mathit{dist}\ a\ y$
using ‹$s \neq \{\}$› *distance-attains-inf* [*of s a*] **by** *blast*
with ‹$x{\in}s$› **show** *?thesis* **using** *dist-pos-lt*[*of a x*] **and**‹$a \notin s$›
by *blast*
qed

lemma *separate-compact-closed*:
fixes $s\ t :: 'a$::*heine-borel set*
assumes *compact s*
and *t*: *closed t* $s \cap t = \{\}$
shows $\exists d{>}0.\ \forall x{\in}s.\ \forall y{\in}t.\ d \leq \mathit{dist}\ x\ y$
proof *cases*
assume $s \neq \{\} \wedge t \neq \{\}$
then have $s \neq \{\}\ t \neq \{\}$ **by** *auto*
let *?inf* $= \lambda x.\ \mathit{infdist}\ x\ t$
have *continuous-on s ?inf*
by (*auto intro*!: *continuous-at-imp-continuous-on continuous-infdist continuous-at-id*)
then obtain x **where** x: $x \in s\ \forall y{\in}s.\ \mathit{?inf}\ x \leq \mathit{?inf}\ y$
using *continuous-attains-inf*[*OF* ‹*compact s*› ‹$s \neq \{\}$›] **by** *auto*
then have $0 <$ *?inf x*
using t ‹$t \neq \{\}$› *in-closed-iff-infdist-zero* **by** (*auto simp*: *less-le infdist-nonneg*)
moreover have $\forall x'{\in}s.\ \forall y{\in}t.\ \mathit{?inf}\ x \leq \mathit{dist}\ x'\ y$
using x **by** (*auto intro*: *order-trans infdist-le*)
ultimately show *?thesis* **by** *auto*
qed (*auto intro*!: *exI*[*of - 1*])

lemma *separate-closed-compact*:
fixes *s t* :: *'a*::*heine-borel set*
assumes *closed s*
and *compact t*
and $s \cap t = \{\}$
shows $\exists\, d{>}0.\ \forall\, x{\in}s.\ \forall\, y{\in}t.\ d \leq dist\ x\ y$
proof −
have ∗: $t \cap s = \{\}$
using *assms*(*3*) **by** *auto*
show *?thesis*
using *separate-compact-closed*[*OF assms*(*2*,*1*) ∗]
apply *auto*
apply (*rule-tac x*=*d* **in** *exI*)
apply *auto*
apply (*erule-tac x*=*y* **in** *ballE*)
apply (*auto simp add*: *dist-commute*)
done
qed

16.27 Closure of halfspaces and hyperplanes

lemma *isCont-open-vimage*:
assumes $\bigwedge x.\ isCont\ f\ x$
and *open s*
shows *open* (*f* −‘ *s*)
proof −
from *assms*(*1*) **have** *continuous-on UNIV f*
unfolding *isCont-def continuous-on-def* **by** *simp*
then have *open* $\{x \in UNIV.\ f\ x \in s\}$
using *open-UNIV* ‹*open s*› **by** (*rule continuous-open-preimage*)
then show *open* (*f* −‘ *s*)
by (*simp add*: *vimage-def*)
qed

lemma *isCont-closed-vimage*:
assumes $\bigwedge x.\ isCont\ f\ x$
and *closed s*
shows *closed* (*f* −‘ *s*)
using *assms* **unfolding** *closed-def vimage-Compl* [*symmetric*]
by (*rule isCont-open-vimage*)

lemma *open-Collect-less*:
fixes *f g* :: *'a*::*t2-space* ⇒ *real*
assumes *f*: $\bigwedge x.\ isCont\ f\ x$
and *g*: $\bigwedge x.\ isCont\ g\ x$
shows *open* $\{x.\ f\ x < g\ x\}$
proof −
have *open* ((λ*x*. *g x* − *f x*) −‘ {*0*<..})
using *isCont-diff* [*OF g f*] *open-real-greaterThan*

by (*rule isCont-open-vimage*)
also have $((\lambda x.\ g\ x - f\ x) -` \{0<..\}) = \{x.\ f\ x < g\ x\}$
by *auto*
finally show *?thesis* **.**
qed

lemma *closed-Collect-le*:
fixes $f\ g :: 'a::t2\text{-}space \Rightarrow real$
assumes f: $\bigwedge x.\ isCont\ f\ x$
and g: $\bigwedge x.\ isCont\ g\ x$
shows *closed* $\{x.\ f\ x \leq g\ x\}$
proof −
have *closed* $((\lambda x.\ g\ x - f\ x) -` \{0..\})$
using *isCont-diff* [*OF g f*] *closed-real-atLeast*
by (*rule isCont-closed-vimage*)
also have $((\lambda x.\ g\ x - f\ x) -` \{0..\}) = \{x.\ f\ x \leq g\ x\}$
by *auto*
finally show *?thesis* **.**
qed

lemma *closed-Collect-eq*:
fixes $f\ g :: 'a::t2\text{-}space \Rightarrow 'b::t2\text{-}space$
assumes f: $\bigwedge x.\ isCont\ f\ x$
and g: $\bigwedge x.\ isCont\ g\ x$
shows *closed* $\{x.\ f\ x = g\ x\}$
proof −
have *open* $\{(x::'b,\ y::'b).\ x \neq y\}$
unfolding *open-prod-def* **by** (*auto dest*!: *hausdorff*)
then have *closed* $\{(x::'b,\ y::'b).\ x = y\}$
unfolding *closed-def split-def Collect-neg-eq* **.**
with *isCont-Pair* [*OF f g*]
have *closed* $((\lambda x.\ (f\ x,\ g\ x)) -` \{(x,\ y).\ x = y\})$
by (*rule isCont-closed-vimage*)
also have $\ldots = \{x.\ f\ x = g\ x\}$ **by** *auto*
finally show *?thesis* **.**
qed

lemma *continuous-on-closed-Collect-le*:
fixes $f\ g :: 'a::t2\text{-}space \Rightarrow real$
assumes f: *continuous-on s f* **and** g: *continuous-on s g* **and** s: *closed s*
shows *closed* $\{x \in s.\ f\ x \leq g\ x\}$
proof −
have *closed* $((\lambda x.\ g\ x - f\ x) -` \{0..\} \cap s)$
using *closed-real-atLeast continuous-on-diff* [*OF g f*]
by (*simp add*: *continuous-on-closed-vimage* [*OF s*])
also have $((\lambda x.\ g\ x - f\ x) -` \{0..\} \cap s) = \{x \in s.\ f\ x \leq g\ x\}$
by *auto*
finally show *?thesis* **.**
qed

lemma *continuous-at-inner*: *continuous* (*at x*) (*inner a*)
unfolding *continuous-at* **by** (*intro tendsto-intros*)

lemma *closed-halfspace-le*: *closed* {*x*. *inner a x* ≤ *b*}
by (*simp add*: *closed-Collect-le*)

lemma *closed-halfspace-ge*: *closed* {*x*. *inner a x* ≥ *b*}
by (*simp add*: *closed-Collect-le*)

lemma *closed-hyperplane*: *closed* {*x*. *inner a x* = *b*}
by (*simp add*: *closed-Collect-eq*)

lemma *closed-halfspace-component-le*: *closed* {*x*::'*a*::*euclidean-space*. *x*•*i* ≤ *a*}
by (*simp add*: *closed-Collect-le*)

lemma *closed-halfspace-component-ge*: *closed* {*x*::'*a*::*euclidean-space*. *x*•*i* ≥ *a*}
by (*simp add*: *closed-Collect-le*)

lemma *closed-interval-left*:
fixes *b* :: '*a*::*euclidean-space*
shows *closed* {*x*::'*a*. ∀ *i*∈*Basis*. *x*•*i* ≤ *b*•*i*}
by (*simp add*: *Collect-ball-eq closed-INT closed-Collect-le*)

lemma *closed-interval-right*:
fixes *a* :: '*a*::*euclidean-space*
shows *closed* {*x*::'*a*. ∀ *i*∈*Basis*. *a*•*i* ≤ *x*•*i*}
by (*simp add*: *Collect-ball-eq closed-INT closed-Collect-le*)

lemma *continuous-le-on-closure*:
fixes *a*::*real*
assumes *f*: *continuous-on* (*closure s*) *f*
and *x*: *x* ∈ *closure*(*s*)
and *xlo*: ⋀*x*. *x* ∈ *s* ==> *f*(*x*) ≤ *a*
shows *f*(*x*) ≤ *a*
using *image-closure-subset* [*OF f*]
using *image-closure-subset* [*OF f*] *closed-halfspace-le* [*of 1*::*real a*] *assms*
by *force*

lemma *continuous-ge-on-closure*:
fixes *a*::*real*
assumes *f*: *continuous-on* (*closure s*) *f*
and *x*: *x* ∈ *closure*(*s*)
and *xlo*: ⋀*x*. *x* ∈ *s* ==> *f*(*x*) ≥ *a*
shows *f*(*x*) ≥ *a*
using *image-closure-subset* [*OF f*] *closed-halfspace-ge* [*of a 1*::*real*] *assms*
by *force*

Openness of halfspaces.

lemma *open-halfspace-lt*: *open* $\{x.\ inner\ a\ x < b\}$
by (*simp add*: *open-Collect-less*)

lemma *open-halfspace-gt*: *open* $\{x.\ inner\ a\ x > b\}$
by (*simp add*: *open-Collect-less*)

lemma *open-halfspace-component-lt*: *open* $\{x::'a::euclidean\text{-}space.\ x{\cdot}i < a\}$
by (*simp add*: *open-Collect-less*)

lemma *open-halfspace-component-gt*: *open* $\{x::'a::euclidean\text{-}space.\ x{\cdot}i > a\}$
by (*simp add*: *open-Collect-less*)

This gives a simple derivation of limit component bounds.

lemma *Lim-component-le*:
fixes $f :: 'a \Rightarrow 'b::euclidean\text{-}space$
assumes $(f \longrightarrow l)\ net$
and $\neg\ (trivial\text{-}limit\ net)$
and $eventually\ (\lambda x.\ f(x){\cdot}i \leq b)\ net$
shows $l{\cdot}i \leq b$
by (*rule tendsto-le*[*OF assms*(*2*) *tendsto-const tendsto-inner*[*OF assms*(*1*) *tendsto-const*] *assms*(*3*)])

lemma *Lim-component-ge*:
fixes $f :: 'a \Rightarrow 'b::euclidean\text{-}space$
assumes $(f \longrightarrow l)\ net$
and $\neg\ (trivial\text{-}limit\ net)$
and $eventually\ (\lambda x.\ b \leq (f\ x){\cdot}i)\ net$
shows $b \leq l{\cdot}i$
by (*rule tendsto-le*[*OF assms*(*2*) *tendsto-inner*[*OF assms*(*1*) *tendsto-const*] *tendsto-const assms*(*3*)])

lemma *Lim-component-eq*:
fixes $f :: 'a \Rightarrow 'b::euclidean\text{-}space$
assumes *net*: $(f \longrightarrow l)\ net\ \neg\ trivial\text{-}limit\ net$
and *ev*:*eventually* $(\lambda x.\ f(x){\cdot}i = b)\ net$
shows $l{\cdot}i = b$
using *ev*[*unfolded order-eq-iff eventually-conj-iff*]
using *Lim-component-ge*[*OF net*, *of b i*]
using *Lim-component-le*[*OF net*, *of i b*]
by *auto*

Limits relative to a union.

lemma *eventually-within-Un*:
$eventually\ P\ (at\ x\ within\ (s \cup t)) \longleftrightarrow$
$eventually\ P\ (at\ x\ within\ s) \wedge eventually\ P\ (at\ x\ within\ t)$
unfolding *eventually-at-filter*
by (*auto elim*!: *eventually-rev-mp*)

lemma *Lim-within-union*:

$(f \longrightarrow l)\ (at\ x\ within\ (s \cup t)) \longleftrightarrow$
$(f \longrightarrow l)\ (at\ x\ within\ s) \wedge (f \longrightarrow l)\ (at\ x\ within\ t)$
unfolding *tendsto-def*
by (*auto simp add*: *eventually-within-Un*)

lemma *Lim-topological*:
$(f \longrightarrow l)\ net \longleftrightarrow$
trivial-limit net $\vee$ $(\forall S.\ open\ S \longrightarrow l \in S \longrightarrow eventually\ (\lambda x.\ f\ x \in S)\ net)$
unfolding *tendsto-def trivial-limit-eq* **by** *auto*

Continuity relative to a union.

lemma *continuous-on-union-local*:
$[\![closedin\ (subtopology\ euclidean\ (s \cup t))\ s;\ closedin\ (subtopology\ euclidean\ (s \cup t))\ t;$
continuous-on s f; *continuous-on t f*$]\!]$
$\Longrightarrow$ *continuous-on* $(s \cup t)$ *f*
unfolding *continuous-on closed-in-limpt*
by (*metis Lim-trivial-limit Lim-within-union Un-iff trivial-limit-within*)

lemma *continuous-on-cases-local*:
$[\![closedin\ (subtopology\ euclidean\ (s \cup t))\ s;\ closedin\ (subtopology\ euclidean\ (s \cup t))\ t;$
continuous-on s f; *continuous-on t g*;
$\bigwedge x.\ [\![x \in s \wedge \sim P\ x \vee x \in t \wedge P\ x]\!] \Longrightarrow f\ x = g\ x]\!]$
$\Longrightarrow$ *continuous-on* $(s \cup t)\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ g\ x)$
by (*rule continuous-on-union-local*) (*auto intro*: *continuous-on-eq*)

lemma *continuous-on-cases-le*:
fixes $h :: 'a :: topological\text{-}space \Rightarrow real$
assumes *continuous-on* $\{t \in s.\ h\ t \leq a\}\ f$
and *continuous-on* $\{t \in s.\ a \leq h\ t\}\ g$
and *h*: *continuous-on s h*
and $\bigwedge t.\ [\![t \in s;\ h\ t = a]\!] \Longrightarrow f\ t = g\ t$
shows *continuous-on s* $(\lambda t.\ if\ h\ t \leq a\ then\ f(t)\ else\ g(t))$
proof −
have *s*: $s = \{t \in s.\ h\ t \in atMost\ a\} \cup \{t \in s.\ h\ t \in atLeast\ a\}$
by *force*
have *1*: *closedin* (*subtopology euclidean s*) $\{t \in s.\ h\ t \in atMost\ a\}$
by (*rule continuous-closedin-preimage* [*OF h closed-atMost*])
have *2*: *closedin* (*subtopology euclidean s*) $\{t \in s.\ h\ t \in atLeast\ a\}$
by (*rule continuous-closedin-preimage* [*OF h closed-atLeast*])
show *?thesis*
apply (*rule continuous-on-subset* [*of s*, *OF - order-refl*])
apply (*subst s*)
apply (*rule continuous-on-cases-local*)
using *1 2 s assms* **apply** *auto*
done
qed

lemma *continuous-on-cases-1*:
fixes s :: *real set*
assumes *continuous-on* $\{t \in s.\ t \leq a\}$ f
and *continuous-on* $\{t \in s.\ a \leq t\}$ g
and $a \in s \Longrightarrow f\ a = g\ a$
shows *continuous-on* s $(\lambda t.\ \textit{if}\ t \leq a\ \textit{then}\ f(t)\ \textit{else}\ g(t))$
using *assms*
by (*auto simp*: *continuous-on-id intro*: *continuous-on-cases-le* [**where** $h = id$, *simplified*])

Some more convenient intermediate-value theorem formulations.

lemma *connected-ivt-hyperplane*:
assumes *connected* s
and $x \in s$
and $y \in s$
and *inner* $a\ x \leq b$
and $b \leq$ *inner* $a\ y$
shows $\exists z \in s.$ *inner* $a\ z = b$
proof (*rule ccontr*)
assume *as*:¬ $(\exists z{\in}s.$ *inner* $a\ z = b)$
let *?A* $= \{x.$ *inner* $a\ x < b\}$
let *?B* $= \{x.$ *inner* $a\ x > b\}$
have *open ?A open ?B*
using *open-halfspace-lt* **and** *open-halfspace-gt* **by** *auto*
moreover
have $?A \cap ?B = \{\}$ **by** *auto*
moreover
have $s \subseteq ?A \cup ?B$ **using** *as* **by** *auto*
ultimately
show *False*
using *assms*(*1*)[*unfolded connected-def not-ex*,
THEN spec[**where** *x*=*?A*], *THEN spec*[**where** *x*=*?B*]]
using *assms*(*2*−*5*)
by *auto*
qed

lemma *connected-ivt-component*:
fixes *x*::*'a*::*euclidean-space*
shows *connected* $s \Longrightarrow$
$x \in s \Longrightarrow y \in s \Longrightarrow$
$x \cdot k \leq a \Longrightarrow a \leq y \cdot k \Longrightarrow (\exists z{\in}s.\ \ z \cdot k = a)$
using *connected-ivt-hyperplane*[*of s x y k*::*'a a*]
by (*auto simp*: *inner-commute*)

16.28 Intervals

lemma *open-box*[*intro*]: *open* (*box a b*)
proof −
have *open* $(\bigcap i{\in}Basis.\ (op \cdot i) -` \{a \cdot i <..< b \cdot i\})$

by (*auto intro*!: *continuous-open-vimage continuous-inner continuous-at-id continuous-const*)
also have $(\bigcap i{\in}Basis.\ (op \cdot i) -` \{a \cdot i <..< b \cdot i\}) = box\ a\ b$
by (*auto simp add*: *box-def inner-commute*)
finally show *?thesis* **.**
qed

instance *euclidean-space* $\subseteq$ *second-countable-topology*
proof
def $a \equiv \lambda f :: 'a \Rightarrow (real \times real).\ \sum i{\in}Basis.\ fst\ (f\ i) *_R i$
then have *a*: $\bigwedge f.\ (\sum i{\in}Basis.\ fst\ (f\ i) *_R i) = a\ f$
by *simp*
def $b \equiv \lambda f :: 'a \Rightarrow (real \times real).\ \sum i{\in}Basis.\ snd\ (f\ i) *_R i$
then have *b*: $\bigwedge f.\ (\sum i{\in}Basis.\ snd\ (f\ i) *_R i) = b\ f$
by *simp*
def $B \equiv (\lambda f.\ box\ (a\ f)\ (b\ f))\ `\ (Basis \to_E (\mathbb{Q} \times \mathbb{Q}))$

have *Ball B open* **by** (*simp add*: *B-def open-box*)
moreover have $(\forall A.\ open\ A \longrightarrow (\exists B'{\subseteq}B.\ \bigcup B' = A))$
proof *safe*
fix *A*::*'a set*
assume *open A*
show $\exists B'{\subseteq}B.\ \bigcup B' = A$
apply (*rule exI*[*of* - $\{b{\in}B.\ b \subseteq A\}$])
apply (*subst* (*3*) *open-UNION-box*[*OF* ‹*open A*›])
apply (*auto simp add*: *a b B-def*)
done
qed
ultimately
have *topological-basis B*
unfolding *topological-basis-def* **by** *blast*
moreover
have *countable B*
unfolding *B-def*
by (*intro countable-image countable-PiE finite-Basis countable-SIGMA countable-rat*)
ultimately show $\exists B::'a\ set\ set.\ countable\ B \wedge open = generate\text{-}topology\ B$
by (*blast intro*: *topological-basis-imp-subbasis*)
qed

instance *euclidean-space* $\subseteq$ *polish-space* **..**

lemma *closed-cbox*[*intro*]:
fixes *a b* :: *'a*::*euclidean-space*
shows *closed* (*cbox a b*)
proof −
have *closed* $(\bigcap i{\in}Basis.\ (\lambda x.\ x{\cdot}i) -` \{a{\cdot}i\ ..\ b{\cdot}i\})$
by (*intro closed-INT ballI continuous-closed-vimage allI*
linear-continuous-at closed-real-atLeastAtMost finite-Basis bounded-linear-inner-left)
also have $(\bigcap i{\in}Basis.\ (\lambda x.\ x{\cdot}i) -` \{a{\cdot}i\ ..\ b{\cdot}i\}) = cbox\ a\ b$
by (*auto simp add*: *cbox-def*)

```
  finally show closed (cbox a b) .
qed

lemma interior-cbox [intro]:
  fixes a b :: 'a::euclidean-space
  shows interior (cbox a b) = box a b (is ?L = ?R)
proof(rule subset-antisym)
  show ?R ⊆ ?L
    using box-subset-cbox open-box
    by (rule interior-maximal)
  {
    fix x
    assume x ∈ interior (cbox a b)
    then obtain s where s: open s x ∈ s s ⊆ cbox a b ..
    then obtain e where e>0 and e:∀x'. dist x' x < e ⟶ x' ∈ cbox a b
      unfolding open-dist and subset-eq by auto
    {
      fix i :: 'a
      assume i: i ∈ Basis
      have dist (x - (e / 2) *R i) x < e
        and dist (x + (e / 2) *R i) x < e
        unfolding dist-norm
        apply auto
        unfolding norm-minus-cancel
        using norm-Basis[OF i] ‹e>0›
        apply auto
        done
      then have a • i ≤ (x - (e / 2) *R i) • i and (x + (e / 2) *R i) • i ≤ b • i
        using e[THEN spec[where x=x - (e/2) *R i]]
          and e[THEN spec[where x=x + (e/2) *R i]]
        unfolding mem-box
        using i
        by blast+
      then have a • i < x • i and x • i < b • i
        using ‹e>0› i
        by (auto simp: inner-diff-left inner-Basis inner-add-left)
    }
    then have x ∈ box a b
      unfolding mem-box by auto
  }
  then show ?L ⊆ ?R ..
qed

lemma bounded-cbox:
  fixes a :: 'a::euclidean-space
  shows bounded (cbox a b)
proof -
  let ?b = ∑i∈Basis. |a•i| + |b•i|
  {
```

```
    fix x :: 'a
    assume x: ∀ i∈Basis. a • i ≤ x • i ∧ x • i ≤ b • i
    {
      fix i :: 'a
      assume i ∈ Basis
      then have |x•i| ≤ |a•i| + |b•i|
        using x[THEN bspec[where x=i]] by auto
    }
    then have (∑ i∈Basis. |x • i|) ≤ ?b
      apply -
      apply (rule setsum-mono)
      apply auto
      done
    then have norm x ≤ ?b
      using norm-le-l1[of x] by auto
  }
  then show ?thesis
    unfolding cbox-def bounded-iff by auto
qed

lemma bounded-box:
  fixes a :: 'a::euclidean-space
  shows bounded (box a b)
  using bounded-cbox[of a b]
  using box-subset-cbox[of a b]
  using bounded-subset[of cbox a b box a b]
  by simp

lemma not-interval-univ:
  fixes a :: 'a::euclidean-space
  shows cbox a b ≠ UNIV box a b ≠ UNIV
  using bounded-box[of a b] bounded-cbox[of a b] by auto

lemma compact-cbox:
  fixes a :: 'a::euclidean-space
  shows compact (cbox a b)
  using bounded-closed-imp-seq-compact[of cbox a b] using bounded-cbox[of a b]
  by (auto simp: compact-eq-seq-compact-metric)

lemma box-midpoint:
  fixes a :: 'a::euclidean-space
  assumes box a b ≠ {}
  shows ((1/2) *R (a + b)) ∈ box a b
proof -
  {
    fix i :: 'a
    assume i ∈ Basis
    then have a • i < ((1 / 2) *R (a + b)) • i ∧ ((1 / 2) *R (a + b)) • i < b • i
     using assms[unfolded box-ne-empty, THEN bspec[where x=i]] by (auto simp:
```

```
inner-add-left)
  }
  then show ?thesis unfolding mem-box by auto
qed

lemma open-cbox-convex:
  fixes x :: 'a::euclidean-space
  assumes x: x ∈ box a b
    and y: y ∈ cbox a b
    and e: 0 < e e ≤ 1
  shows (e *R x + (1 − e) *R y) ∈ box a b
proof −
  {
    fix i :: 'a
    assume i: i ∈ Basis
    have a · i = e * (a · i) + (1 − e) * (a · i)
      unfolding left-diff-distrib by simp
    also have ... < e * (x · i) + (1 − e) * (y · i)
      apply (rule add-less-le-mono)
      using e unfolding mult-less-cancel-left and mult-le-cancel-left
      apply simp-all
      using x unfolding mem-box using i
      apply simp
      using y unfolding mem-box using i
      apply simp
      done
    finally have a · i < (e *R x + (1 − e) *R y) · i
      unfolding inner-simps by auto
    moreover
    {
      have b · i = e * (b·i) + (1 − e) * (b·i)
        unfolding left-diff-distrib by simp
      also have ... > e * (x · i) + (1 − e) * (y · i)
        apply (rule add-less-le-mono)
        using e unfolding mult-less-cancel-left and mult-le-cancel-left
        apply simp-all
        using x
        unfolding mem-box
        using i
        apply simp
        using y
        unfolding mem-box
        using i
        apply simp
        done
      finally have (e *R x + (1 − e) *R y) · i < b · i
        unfolding inner-simps by auto
    }
    ultimately have a · i < (e *R x + (1 − e) *R y) · i ∧ (e *R x + (1 − e)
```

```
*R y) · i < b · i
      by auto
  }
  then show ?thesis
    unfolding mem-box by auto
qed

lemma closure-box:
  fixes a :: 'a::euclidean-space
   assumes box a b ≠ {}
  shows closure (box a b) = cbox a b
proof −
  have ab: a <e b
    using assms by (simp add: eucl-less-def box-ne-empty)
  let ?c = (1 / 2) *R (a + b)
  {
    fix x
    assume as:x ∈ cbox a b
    def f ≡ λn::nat. x + (inverse (real n + 1)) *R (?c − x)
    {
      fix n
      assume fn: f n <e b ⟶ a <e f n ⟶ f n = x and xc: x ≠ ?c
      have ∗: 0 < inverse (real n + 1) inverse (real n + 1) ≤ 1
        unfolding inverse-le-1-iff by auto
      have (inverse (real n + 1)) *R ((1 / 2) *R (a + b)) + (1 − inverse (real n
+ 1)) *R x =
        x + (inverse (real n + 1)) *R (((1 / 2) *R (a + b)) − x)
        by (auto simp add: algebra-simps)
      then have f n <e b and a <e f n
        using open-cbox-convex[OF box-midpoint[OF assms] as ∗]
        unfolding f-def by (auto simp: box-def eucl-less-def)
      then have False
        using fn unfolding f-def using xc by auto
    }
    moreover
    {
      assume ¬ (f ⟶⟶ x) sequentially
      {
        fix e :: real
        assume e > 0
        then have ∃N::nat. inverse (real (N + 1)) < e
          using real-arch-inv[of e]
          apply (auto simp add: Suc-pred')
          apply (metis Suc-pred' of-nat-Suc)
          done
        then obtain N :: nat where N: inverse (real (N + 1)) < e
          by auto
        have inverse (real n + 1) < e if N ≤ n for n
          by (auto intro!: that le-less-trans [OF - N])
```

```
      then have ∃N::nat. ∀n≥N. inverse (real n + 1) < e by auto
    }
    then have ((λn. inverse (real n + 1)) ⟶ 0) sequentially
      unfolding lim-sequentially by(auto simp add: dist-norm)
    then have (f ⟶ x) sequentially
      unfolding f-def
      using tendsto-add[OF tendsto-const, of λn::nat. (inverse (real n + 1)) *R
((1 / 2) *R (a + b) − x) 0 sequentially x]
      using tendsto-scaleR [OF - tendsto-const, of λn::nat. inverse (real n + 1)
0 sequentially ((1 / 2) *R (a + b) − x)]
      by auto
  }
  ultimately have x ∈ closure (box a b)
    using as and box-midpoint[OF assms]
    unfolding closure-def
    unfolding islimpt-sequential
    by (cases x=?c) (auto simp: in-box-eucl-less)
 }
 then show ?thesis
  using closure-minimal[OF box-subset-cbox, of a b] by blast
qed

lemma bounded-subset-box-symmetric:
 fixes s::('a::euclidean-space) set
 assumes bounded s
 shows ∃a. s ⊆ box (−a) a
proof −
 obtain b where b>0 and b: ∀x∈s. norm x ≤ b
  using assms[unfolded bounded-pos] by auto
 def a ≡ (∑i∈Basis. (b + 1) *R i)::'a
 {
  fix x
  assume x ∈ s
  fix i :: 'a
  assume i: i ∈ Basis
  then have (−a)•i < x•i and x•i < a•i
    using b[THEN bspec[where x=x], OF ‹x∈s›]
    using Basis-le-norm[OF i, of x]
    unfolding inner-simps and a-def
    by auto
 }
 then show ?thesis
  by (auto intro: exI[where x=a] simp add: box-def)
qed

lemma bounded-subset-open-interval:
 fixes s :: ('a::euclidean-space) set
 shows bounded s ⟹ (∃a b. s ⊆ box a b)
 by (auto dest!: bounded-subset-box-symmetric)
```

lemma *bounded-subset-cbox-symmetric*:
fixes s :: ('a::*euclidean-space*) *set*
assumes *bounded* s
shows $\exists a.\ s \subseteq cbox\ (-a)\ a$
proof −
obtain a **where** $s \subseteq box\ (-a)\ a$
using *bounded-subset-box-symmetric*[*OF assms*] **by** *auto*
then show *?thesis*
using *box-subset-cbox*[*of* $-a$ a] **by** *auto*
qed

lemma *bounded-subset-cbox*:
fixes s :: ('a::*euclidean-space*) *set*
shows *bounded* $s \Longrightarrow \exists a\ b.\ s \subseteq cbox\ a\ b$
using *bounded-subset-cbox-symmetric*[*of* s] **by** *auto*

lemma *frontier-cbox*:
fixes $a\ b$ :: 'a::*euclidean-space*
shows *frontier* $(cbox\ a\ b) = cbox\ a\ b - box\ a\ b$
unfolding *frontier-def* **unfolding** *interior-cbox* **and** *closure-closed*[*OF closed-cbox*]
..

lemma *frontier-box*:
fixes $a\ b$ :: 'a::*euclidean-space*
shows *frontier* $(box\ a\ b) = (if\ box\ a\ b = \{\}\ then\ \{\}\ else\ cbox\ a\ b - box\ a\ b)$
proof (*cases* $box\ a\ b = \{\}$)
case *True*
then show *?thesis*
using *frontier-empty* **by** *auto*
next
case *False*
then show *?thesis*
unfolding *frontier-def* **and** *closure-box*[*OF False*] **and** *interior-open*[*OF open-box*]
by *auto*
qed

lemma *inter-interval-mixed-eq-empty*:
fixes a :: 'a::*euclidean-space*
assumes $box\ c\ d \neq \{\}$
shows $box\ a\ b \cap cbox\ c\ d = \{\} \longleftrightarrow box\ a\ b \cap box\ c\ d = \{\}$
unfolding *closure-box*[*OF assms*, *symmetric*]
unfolding *open-inter-closure-eq-empty*[*OF open-box*] **..**

lemma *diameter-cbox*:
fixes $a\ b$::'a::*euclidean-space*
shows $(\forall i \in Basis.\ a \cdot i \leq b \cdot i) \Longrightarrow diameter\ (cbox\ a\ b) = dist\ a\ b$
by (*force simp add*: *diameter-def SUP-def simp del*: *Sup-image-eq intro*!: *cSup-eq-maximum setL2-mono*

simp: *euclidean-dist-l2*[**where** *'a='a*] *cbox-def dist-norm*)

lemma *eucl-less-eq-halfspaces*:
fixes *a* :: *'a*::*euclidean-space*
shows $\{x.\ x <e\ a\} = (\bigcap i{\in}Basis.\ \{x.\ x \cdot i < a \cdot i\})$
$\{x.\ a <e\ x\} = (\bigcap i{\in}Basis.\ \{x.\ a \cdot i < x \cdot i\})$
by (*auto simp*: *eucl-less-def*)

lemma *eucl-le-eq-halfspaces*:
fixes *a* :: *'a*::*euclidean-space*
shows $\{x.\ \forall i{\in}Basis.\ x \cdot i \le a \cdot i\} = (\bigcap i{\in}Basis.\ \{x.\ x \cdot i \le a \cdot i\})$
$\{x.\ \forall i{\in}Basis.\ a \cdot i \le x \cdot i\} = (\bigcap i{\in}Basis.\ \{x.\ a \cdot i \le x \cdot i\})$
by *auto*

lemma *open-Collect-eucl-less*[*simp*, *intro*]:
fixes *a* :: *'a*::*euclidean-space*
shows *open* $\{x.\ x <e\ a\}$
open $\{x.\ a <e\ x\}$
by (*auto simp*: *eucl-less-eq-halfspaces open-halfspace-component-lt open-halfspace-component-gt*)

lemma *closed-Collect-eucl-le*[*simp*, *intro*]:
fixes *a* :: *'a*::*euclidean-space*
shows *closed* $\{x.\ \forall i{\in}Basis.\ a \cdot i \le x \cdot i\}$
closed $\{x.\ \forall i{\in}Basis.\ x \cdot i \le a \cdot i\}$
unfolding *eucl-le-eq-halfspaces*
by (*simp-all add*: *closed-INT closed-Collect-le*)

lemma *image-affinity-cbox*: **fixes** *m*::*real*
fixes *a b c* :: *'a*::*euclidean-space*
shows $(\lambda x.\ m *_R x + c)$ ` *cbox a b* =
(*if cbox a b* = {} *then* {}
else (*if* $0 \le m$ *then cbox* $(m *_R a + c)$ $(m *_R b + c)$
else cbox $(m *_R b + c)$ $(m *_R a + c)$)))
proof (*cases* $m = 0$)
case *True*
{
fix *x*
assume $\forall i{\in}Basis.\ x \cdot i \le c \cdot i\ \forall i{\in}Basis.\ c \cdot i \le x \cdot i$
then have $x = c$
apply −
apply (*subst euclidean-eq-iff*)
apply (*auto intro*: *order-antisym*)
done
}
moreover have $c \in cbox\ (m *_R a + c)\ (m *_R b + c)$
unfolding *True* **by** (*auto simp add*: *cbox-sing*)
ultimately show *?thesis* **using** *True* **by** (*auto simp*: *cbox-def*)
next
case *False*

```
  {
    fix y
    assume ∀i∈Basis. a • i ≤ y • i ∀i∈Basis. y • i ≤ b • i m > 0
    then have ∀i∈Basis. (m *R a + c) • i ≤ (m *R y + c) • i and ∀i∈Basis.
(m *R y + c) • i ≤ (m *R b + c) • i
      by (auto simp: inner-distrib)
  }
  moreover
  {
    fix y
    assume ∀i∈Basis. a • i ≤ y • i ∀i∈Basis. y • i ≤ b • i m < 0
    then have ∀i∈Basis. (m *R b + c) • i ≤ (m *R y + c) • i and ∀i∈Basis.
(m *R y + c) • i ≤ (m *R a + c) • i
      by (auto simp add: mult-left-mono-neg inner-distrib)
  }
  moreover
  {
    fix y
    assume m > 0 and ∀i∈Basis. (m *R a + c) • i ≤ y • i and ∀i∈Basis. y • i
≤ (m *R b + c) • i
    then have y ∈ (λx. m *R x + c) ‘ cbox a b
      unfolding image-iff Bex-def mem-box
      apply (intro exI[where x=(1 / m) *R (y − c)])
    apply (auto simp add: pos-le-divide-eq pos-divide-le-eq mult.commute inner-distrib
inner-diff-left)
      done
  }
  moreover
  {
    fix y
    assume ∀i∈Basis. (m *R b + c) • i ≤ y • i ∀i∈Basis. y • i ≤ (m *R a + c)
• i m < 0
    then have y ∈ (λx. m *R x + c) ‘ cbox a b
      unfolding image-iff Bex-def mem-box
      apply (intro exI[where x=(1 / m) *R (y − c)])
    apply (auto simp add: neg-le-divide-eq neg-divide-le-eq mult.commute inner-distrib
inner-diff-left)
      done
  }
  ultimately show ?thesis using False by (auto simp: cbox-def)
qed

lemma image-smult-cbox:(λx. m *R (x::-::euclidean-space)) ‘ cbox a b =
  (if cbox a b = {} then {} else if 0 ≤ m then cbox (m *R a) (m *R b) else cbox
(m *R b) (m *R a))
  using image-affinity-cbox[of m 0 a b] by auto

lemma islimpt-greaterThanLessThan1:
  fixes a b::'a::{linorder-topology, dense-order}
```

assumes $a < b$
shows a *islimpt* $\{a<..<b\}$
proof (*rule islimptI*)
fix T
assume *open* T $a \in T$
from *open-right*[*OF this* ‹$a < b$›]
obtain c **where** c: $a < c$ $\{a..<c\} \subseteq T$ **by** *auto*
with *assms dense*[*of a min c b*]
show $\exists y{\in}\{a<..<b\}.\ y \in T \wedge y \neq a$
by (*metis atLeastLessThan-iff greaterThanLessThan-iff min-less-iff-conj not-le order.strict-implies-order subset-eq*)
qed

lemma *islimpt-greaterThanLessThan2*:
fixes a b::'a::{*linorder-topology*, *dense-order*}
assumes $a < b$
shows b *islimpt* $\{a<..<b\}$
proof (*rule islimptI*)
fix T
assume *open* T $b \in T$
from *open-left*[*OF this* ‹$a < b$›]
obtain c **where** c: $c < b$ $\{c<..b\} \subseteq T$ **by** *auto*
with *assms dense*[*of max a c b*]
show $\exists y{\in}\{a<..<b\}.\ y \in T \wedge y \neq b$
by (*metis greaterThanAtMost-iff greaterThanLessThan-iff max-less-iff-conj not-le order.strict-implies-order subset-eq*)
qed

lemma *closure-greaterThanLessThan*[*simp*]:
fixes a b::'a::{*linorder-topology*, *dense-order*}
shows $a < b \Longrightarrow$ *closure* $\{a <..< b\} = \{a\ ..\ b\}$ (**is** - $\Longrightarrow$ *?l* = *?r*)
proof
have *?l* $\subseteq$ *closure ?r*
by (*rule closure-mono*) *auto*
thus *closure* $\{a<..<b\} \subseteq \{a..b\}$ **by** *simp*
qed (*auto simp*: *closure-def order.order-iff-strict islimpt-greaterThanLessThan1 islimpt-greaterThanLessThan2*)

lemma *closure-greaterThan*[*simp*]:
fixes a b::'a::{*no-top*, *linorder-topology*, *dense-order*}
shows *closure* $\{a<..\} = \{a..\}$
proof −
from *gt-ex* **obtain** b **where** $a < b$ **by** *auto*
hence $\{a<..\} = \{a<..<b\} \cup \{b..\}$ **by** *auto*
also have *closure* ... $= \{a..\}$ **using** ‹$a < b$› **unfolding** *closure-union*
by *auto*
finally show *?thesis* **.**
qed

lemma *closure-lessThan*[*simp*]:
fixes *b*::*'a*::{*no-bot*, *linorder-topology*, *dense-order*}
shows *closure* {..<*b*} = {..*b*}
proof −
from *lt-ex* **obtain** *a* **where** *a* < *b* **by** *auto*
hence {..<*b*} = {*a*<..<*b*} ∪ {..*a*} **by** *auto*
also have *closure* ... = {..*b*} **using** ‹*a* < *b*› **unfolding** *closure-union*
by *auto*
finally show *?thesis* **.**
qed

lemma *closure-atLeastLessThan*[*simp*]:
fixes *a b*::*'a*::{*linorder-topology*, *dense-order*}
assumes *a* < *b*
shows *closure* {*a* ..< *b*} = {*a* .. *b*}
proof −
from *assms* **have** {*a* ..< *b*} = {*a*} ∪ {*a* <..< *b*} **by** *auto*
also have *closure* ... = {*a* .. *b*} **unfolding** *closure-union*
by (*auto simp add*: *assms less-imp-le*)
finally show *?thesis* **.**
qed

lemma *closure-greaterThanAtMost*[*simp*]:
fixes *a b*::*'a*::{*linorder-topology*, *dense-order*}
assumes *a* < *b*
shows *closure* {*a* <.. *b*} = {*a* .. *b*}
proof −
from *assms* **have** {*a* <.. *b*} = {*b*} ∪ {*a* <..< *b*} **by** *auto*
also have *closure* ... = {*a* .. *b*} **unfolding** *closure-union*
by (*auto simp add*: *assms less-imp-le*)
finally show *?thesis* **.**
qed

16.29 Homeomorphisms

definition *homeomorphism s t f g* ⟷
(∀ *x*∈*s*. (*g*(*f x*) = *x*)) ∧ (*f* ‘ *s* = *t*) ∧ *continuous-on s f* ∧
(∀ *y*∈*t*. (*f*(*g y*) = *y*)) ∧ (*g* ‘ *t* = *s*) ∧ *continuous-on t g*

definition *homeomorphic* :: *'a*::*topological-space set* ⇒ *'b*::*topological-space set* ⇒ *bool*
(**infixr** *homeomorphic 60*)
where *s homeomorphic t* ≡ (∃ *f g*. *homeomorphism s t f g*)

lemma *homeomorphic-refl*: *s homeomorphic s*
unfolding *homeomorphic-def*
unfolding *homeomorphism-def*
using *continuous-on-id*
apply (*rule-tac x* = (λ*x*. *x*) **in** *exI*)

apply (*rule-tac* $x = (\lambda x.\ x)$ **in** *exI*)
apply *blast*
done

lemma *homeomorphic-sym*: *s homeomorphic t* $\longleftrightarrow$ *t homeomorphic s*
unfolding *homeomorphic-def*
unfolding *homeomorphism-def*
by *blast*

lemma *homeomorphic-trans*:
assumes *s homeomorphic t*
and *t homeomorphic u*
shows *s homeomorphic u*
proof −
obtain *f1 g1* **where** *fg1*: $\forall x{\in}s.\ g1\ (f1\ x) = x\ \ f1\ `\ s = t$
continuous-on s f1 $\forall y{\in}t.\ f1\ (g1\ y) = y\ g1\ `\ t = s$ *continuous-on t g1*
using *assms*(*1*) **unfolding** *homeomorphic-def homeomorphism-def* **by** *auto*
obtain *f2 g2* **where** *fg2*: $\forall x{\in}t.\ g2\ (f2\ x) = x\ \ f2\ `\ t = u$ *continuous-on t f2*
$\forall y{\in}u.\ f2\ (g2\ y) = y\ g2\ `\ u = t$ *continuous-on u g2*
using *assms*(*2*) **unfolding** *homeomorphic-def homeomorphism-def* **by** *auto*
{
fix *x*
assume $x{\in}s$
then have $(g1 \circ g2)\ ((f2 \circ f1)\ x) = x$
using *fg1*(*1*)[*THEN bspec*[**where** *x*=*x*]] **and** *fg2*(*1*)[*THEN bspec*[**where** *x*=*f1 x*]] **and** *fg1*(*2*)
by *auto*
}
moreover have $(f2 \circ f1)\ `\ s = u$
using *fg1*(*2*) *fg2*(*2*) **by** *auto*
moreover have *continuous-on s* $(f2 \circ f1)$
using *continuous-on-compose*[*OF fg1*(*3*)] **and** *fg2*(*3*) **unfolding** *fg1*(*2*) **by** *auto*
moreover
{
fix *y*
assume $y{\in}u$
then have $(f2 \circ f1)\ ((g1 \circ g2)\ y) = y$
using *fg2*(*4*)[*THEN bspec*[**where** *x*=*y*]] **and** *fg1*(*4*)[*THEN bspec*[**where** *x*=*g2 y*]] **and** *fg2*(*5*)
by *auto*
}
moreover have $(g1 \circ g2)\ `\ u = s$ **using** *fg1*(*5*) *fg2*(*5*) **by** *auto*
moreover have *continuous-on u* $(g1 \circ g2)$
using *continuous-on-compose*[*OF fg2*(*6*)] **and** *fg1*(*6*)
unfolding *fg2*(*5*)
by *auto*
ultimately show *?thesis*
unfolding *homeomorphic-def homeomorphism-def*

```
    apply (rule-tac x=f2 ∘ f1 in exI)
    apply (rule-tac x=g1 ∘ g2 in exI)
    apply auto
    done
qed

lemma homeomorphic-minimal:
  s homeomorphic t ⟷
    (∃f g. (∀x∈s. f(x) ∈ t ∧ (g(f(x)) = x)) ∧
          (∀y∈t. g(y) ∈ s ∧ (f(g(y)) = y)) ∧
          continuous-on s f ∧ continuous-on t g)
  unfolding homeomorphic-def homeomorphism-def
  apply auto
  apply (rule-tac x=f in exI)
  apply (rule-tac x=g in exI)
  apply auto
  apply (rule-tac x=f in exI)
  apply (rule-tac x=g in exI)
  apply auto
  unfolding image-iff
  apply (erule-tac x=g x in ballE)
  apply (erule-tac x=x in ballE)
  apply auto
  apply (rule-tac x=g x in bexI)
  apply auto
  apply (erule-tac x=f x in ballE)
  apply (erule-tac x=x in ballE)
  apply auto
  apply (rule-tac x=f x in bexI)
  apply auto
  done
```

Relatively weak hypotheses if a set is compact.

```
lemma homeomorphism-compact:
  fixes f :: 'a::topological-space ⇒ 'b::t2-space
  assumes compact s continuous-on s f  f ` s = t  inj-on f s
  shows ∃g. homeomorphism s t f g
proof −
  def g ≡ λx. SOME y. y∈s ∧ f y = x
  have g: ∀x∈s. g (f x) = x
    using assms(3) assms(4)[unfolded inj-on-def] unfolding g-def by auto
  {
    fix y
    assume y ∈ t
    then obtain x where x:f x = y x∈s
      using assms(3) by auto
    then have g (f x) = x using g by auto
    then have f (g y) = y unfolding x(1)[symmetric] by auto
  }
```

```
  then have g':∀x∈t. f (g x) = x by auto
  moreover
  {
    fix x
    have x∈s ⟹ x ∈ g ` t
      using g[THEN bspec[where x=x]]
      unfolding image-iff
      using assms(3)
      by (auto intro!: bexI[where x=f x])
    moreover
    {
      assume x∈g ` t
      then obtain y where y:y∈t g y = x by auto
      then obtain x' where x':x'∈s f x' = y
        using assms(3) by auto
      then have x ∈ s
        unfolding g-def
        using someI2[of λb. b∈s ∧ f b = y x' λx. x∈s]
        unfolding y(2)[symmetric] and g-def
        by auto
    }
    ultimately have x∈s ⟷ x ∈ g ` t ..
  }
  then have g ` t = s by auto
  ultimately show ?thesis
    unfolding homeomorphism-def homeomorphic-def
    apply (rule-tac x=g in exI)
    using g and assms(3) and continuous-on-inv[OF assms(2,1), of g, unfolded
assms(3)] and assms(2)
    apply auto
    done
qed

lemma homeomorphic-compact:
  fixes f :: 'a::topological-space ⇒ 'b::t2-space
  shows compact s ⟹ continuous-on s f ⟹ (f ` s = t) ⟹ inj-on f s ⟹ s
homeomorphic t
  unfolding homeomorphic-def by (metis homeomorphism-compact)
```

Preservation of topological properties.

```
lemma homeomorphic-compactness: s homeomorphic t ⟹ (compact s ⟷ com-
pact t)
  unfolding homeomorphic-def homeomorphism-def
  by (metis compact-continuous-image)
```

Results on translation, scaling etc.

```
lemma homeomorphic-scaling:
  fixes s :: 'a::real-normed-vector set
  assumes c ≠ 0
```

shows *s homeomorphic* $((\lambda x.\ c *_R x)\ `\ s)$
unfolding *homeomorphic-minimal*
apply (*rule-tac* $x{=}\lambda x.\ c *_R x$ **in** *exI*)
apply (*rule-tac* $x{=}\lambda x.\ (1\ /\ c) *_R x$ **in** *exI*)
using *assms*
apply (*auto simp add*: *continuous-intros*)
done

lemma *homeomorphic-translation*:
fixes *s* :: *'a*::*real-normed-vector set*
shows *s homeomorphic* $((\lambda x.\ a + x)\ `\ s)$
unfolding *homeomorphic-minimal*
apply (*rule-tac* $x{=}\lambda x.\ a + x$ **in** *exI*)
apply (*rule-tac* $x{=}\lambda x.\ -a + x$ **in** *exI*)
using *continuous-on-add* [*OF continuous-on-const continuous-on-id*, *of s a*]
continuous-on-add [*OF continuous-on-const continuous-on-id*, *of plus a* ` *s* − *a*]
apply *auto*
done

lemma *homeomorphic-affinity*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes $c \neq 0$
shows *s homeomorphic* $((\lambda x.\ a + c *_R x)\ `\ s)$
proof −
have ∗: $op + a\ `\ op *_R c\ `\ s = (\lambda x.\ a + c *_R x)\ `\ s$ **by** *auto*
show *?thesis*
using *homeomorphic-trans*
using *homeomorphic-scaling*[*OF assms*, *of s*]
using *homeomorphic-translation*[*of* $(\lambda x.\ c *_R x)\ `\ s\ a$]
unfolding ∗
by *auto*
qed

lemma *homeomorphic-balls*:
fixes *a b* ::*'a*::*real-normed-vector*
assumes $0 < d\ \ 0 < e$
shows (*ball a d*) *homeomorphic* (*ball b e*) (**is** *?th*)
and (*cball a d*) *homeomorphic* (*cball b e*) (**is** *?cth*)
proof −
show *?th* **unfolding** *homeomorphic-minimal*
apply(*rule-tac* $x{=}\lambda x.\ b + (e/d) *_R (x - a)$ **in** *exI*)
apply(*rule-tac* $x{=}\lambda x.\ a + (d/e) *_R (x - b)$ **in** *exI*)
using *assms*
apply (*auto intro*!: *continuous-intros*
simp: *dist-commute dist-norm pos-divide-less-eq mult-strict-left-mono*)
done
show *?cth* **unfolding** *homeomorphic-minimal*
apply(*rule-tac* $x{=}\lambda x.\ b + (e/d) *_R (x - a)$ **in** *exI*)

```
    apply(rule-tac x=λx. a + (d/e) *R (x - b) in exI)
    using assms
    apply (auto intro!: continuous-intros
      simp: dist-commute dist-norm pos-divide-le-eq mult-strict-left-mono)
    done
qed
```

"Isometry" (up to constant bounds) of injective linear map etc.

```
lemma cauchy-isometric:
  assumes e: e > 0
    and s: subspace s
    and f: bounded-linear f
    and normf: ∀x∈s. norm (f x) ≥ e * norm x
    and xs: ∀n. x n ∈ s
    and cf: Cauchy (f ∘ x)
  shows Cauchy x
proof -
  interpret f: bounded-linear f by fact
  {
    fix d :: real
    assume d > 0
    then obtain N where N:∀n≥N. norm (f (x n) - f (x N)) < e * d
      using cf[unfolded cauchy o-def dist-norm, THEN spec[where x=e*d]] e
      by auto
    {
      fix n
      assume n≥N
      have e * norm (x n - x N) ≤ norm (f (x n - x N))
        using subspace-sub[OF s, of x n x N]
        using xs[THEN spec[where x=N]] and xs[THEN spec[where x=n]]
        using normf[THEN bspec[where x=x n - x N]]
        by auto
      also have norm (f (x n - x N)) < e * d
        using ‹N ≤ n› N unfolding f.diff[symmetric] by auto
      finally have norm (x n - x N) < d using ‹e>0› by simp
    }
    then have ∃N. ∀n≥N. norm (x n - x N) < d by auto
  }
  then show ?thesis unfolding cauchy and dist-norm by auto
qed

lemma complete-isometric-image:
  assumes 0 < e
    and s: subspace s
    and f: bounded-linear f
    and normf: ∀x∈s. norm(f x) ≥ e * norm(x)
    and cs: complete s
  shows complete (f ‘ s)
proof -
```

```
  {
    fix g
    assume as:∀ n::nat. g n ∈ f ‘ s and cfg:Cauchy g
    then obtain x where ∀ n. x n ∈ s ∧ g n = f (x n)
      using choice[of λ n xa. xa ∈ s ∧ g n = f xa]
      by auto
    then have x:∀ n. x n ∈ s  ∀ n. g n = f (x n)
      by auto
    then have f ∘ x = g
      unfolding fun-eq-iff
      by auto
    then obtain l where l∈s and l:(x ⟶ l) sequentially
      using cs[unfolded complete-def, THEN spec[where x=x]]
      using cauchy-isometric[OF ‹0 < e› s f normf] and cfg and x(1)
      by auto
    then have ∃ l∈f ‘ s. (g ⟶ l) sequentially
      using linear-continuous-at[OF f, unfolded continuous-at-sequentially, THEN
spec[where x=x], of l]
      unfolding ‹f ∘ x = g›
      by auto
  }
  then show ?thesis
    unfolding complete-def by auto
qed

lemma injective-imp-isometric:
  fixes f :: 'a::euclidean-space ⇒ 'b::euclidean-space
  assumes s: closed s subspace s
    and f: bounded-linear f ∀ x∈s. f x = 0 ⟶ x = 0
  shows ∃ e>0. ∀ x∈s. norm (f x) ≥ e * norm x
proof (cases s ⊆ {0::'a})
  case True
  {
    fix x
    assume x ∈ s
    then have x = 0 using True by auto
    then have norm x ≤ norm (f x) by auto
  }
  then show ?thesis by (auto intro!: exI[where x=1])
next
  interpret f: bounded-linear f by fact
  case False
  then obtain a where a: a ≠ 0 a ∈ s
    by auto
  from False have s ≠ {}
    by auto
  let ?S = {f x| x. (x ∈ s ∧ norm x = norm a)}
  let ?S' = {x::'a. x∈s ∧ norm x = norm a}
  let ?S'' = {x::'a. norm x = norm a}
```

```
  have ?S'' = frontier(cball 0 (norm a))
    unfolding frontier-cball and dist-norm by auto
  then have compact ?S''
    using compact-frontier[OF compact-cball, of 0 norm a] by auto
  moreover have ?S' = s ∩ ?S'' by auto
  ultimately have compact ?S'
    using closed-inter-compact[of s ?S''] using s(1) by auto
  moreover have ∗:f ‘ ?S' = ?S by auto
  ultimately have compact ?S
   using compact-continuous-image[OF linear-continuous-on[OF f(1)], of ?S'] by
auto
  then have closed ?S using compact-imp-closed by auto
  moreover have ?S ≠ {} using a by auto
  ultimately obtain b' where b'∈?S ∀ y∈?S. norm b' ≤ norm y
    using distance-attains-inf[of ?S 0] unfolding dist-0-norm by auto
  then obtain b where b∈s
    and ba: norm b = norm a
    and b: ∀ x∈{x ∈ s. norm x = norm a}. norm (f b) ≤ norm (f x)
    unfolding ∗[symmetric] unfolding image-iff by auto

  let ?e = norm (f b) / norm b
  have norm b > 0 using ba and a and norm-ge-zero by auto
  moreover have norm (f b) > 0
    using f(2)[THEN bspec[where x=b], OF ‹b∈s›]
    using ‹norm b >0›
    unfolding zero-less-norm-iff
    by auto
  ultimately have 0 < norm (f b) / norm b by simp
  moreover
  {
    fix x
    assume x∈s
    then have norm (f b) / norm b ∗ norm x ≤ norm (f x)
    proof (cases x=0)
      case True
      then show norm (f b) / norm b ∗ norm x ≤ norm (f x) by auto
    next
      case False
      then have ∗: 0 < norm a / norm x
        using ‹a≠0›
        unfolding zero-less-norm-iff[symmetric] by simp
      have ∀ c. ∀ x∈s. c ∗R x ∈ s
        using s[unfolded subspace-def] by auto
      then have (norm a / norm x) ∗R x ∈ {x ∈ s. norm x = norm a}
        using ‹x∈s› and ‹x≠0› by auto
      then show norm (f b) / norm b ∗ norm x ≤ norm (f x)
        using b[THEN bspec[where x=(norm a / norm x) ∗R x]]
        unfolding f.scaleR and ba using ‹x≠0› ‹a≠0›
```

```
      by (auto simp add: mult.commute pos-le-divide-eq pos-divide-le-eq)
    qed
  }
  ultimately show ?thesis by auto
qed

lemma closed-injective-image-subspace:
  fixes f :: 'a::euclidean-space ⇒ 'b::euclidean-space
  assumes subspace s bounded-linear f ∀ x∈s. f x = 0 ⟶ x = 0 closed s
  shows closed(f ‘ s)
proof −
  obtain e where e > 0 and e: ∀ x∈s. e * norm x ≤ norm (f x)
    using injective-imp-isometric[OF assms(4,1,2,3)] by auto
  show ?thesis
    using complete-isometric-image[OF ‹e>0› assms(1,2) e] and assms(4)
    unfolding complete-eq-closed[symmetric] by auto
qed
```

16.30 Some properties of a canonical subspace

```
lemma subspace-substandard:
  subspace {x::'a::euclidean-space. (∀ i∈Basis. P i ⟶ x•i = 0)}
  unfolding subspace-def by (auto simp: inner-add-left)

lemma closed-substandard:
  closed {x::'a::euclidean-space. ∀ i∈Basis. P i −−> x•i = 0} (is closed ?A)
proof −
  let ?D = {i∈Basis. P i}
  have closed (⋂ i∈?D. {x::'a. x•i = 0})
    by (simp add: closed-INT closed-Collect-eq)
  also have (⋂ i∈?D. {x::'a. x•i = 0}) = ?A
    by auto
  finally show closed ?A .
qed

lemma dim-substandard:
  assumes d: d ⊆ Basis
  shows dim {x::'a::euclidean-space. ∀ i∈Basis. i ∉ d ⟶ x•i = 0} = card d (is
dim ?A = -)
proof (rule dim-unique)
  show d ⊆ ?A
    using d by (auto simp: inner-Basis)
  show independent d
    using independent-mono [OF independent-Basis d] .
  show ?A ⊆ span d
  proof (clarify)
    fix x assume x: ∀ i∈Basis. i ∉ d ⟶ x • i = 0
    have finite d
      using finite-subset [OF d finite-Basis] .
```

then have $(\sum i{\in}d.\ (x \cdot i) *_R i) \in span\ d$
by (*simp add*: *span-setsum span-clauses*)
also have $(\sum i{\in}d.\ (x \cdot i) *_R i) = (\sum i{\in}Basis.\ (x \cdot i) *_R i)$
by (*rule setsum.mono-neutral-cong-left* [*OF finite-Basis d*]) (*auto simp add*: *x*)
finally show $x \in span\ d$
unfolding *euclidean-representation* **.**
qed
qed *simp*

Hence closure and completeness of all subspaces.

lemma *ex-card*:
assumes $n \leq card\ A$
shows $\exists S{\subseteq}A.\ card\ S = n$
proof *cases*
assume *finite A*
from *ex-bij-betw-nat-finite*[*OF this*] **obtain** *f* **where** *f*: *bij-betw f* $\{0..<card\ A\}$ *A* **..**
moreover from *f* ‹$n \leq card\ A$› **have** $\{..<n\} \subseteq \{..<card\ A\}$ *inj-on f* $\{..<n\}$
by (*auto simp*: *bij-betw-def intro*: *subset-inj-on*)
ultimately have $f\ `\ \{..<n\} \subseteq A$ *card* $(f\ `\ \{..<n\}) = n$
by (*auto simp*: *bij-betw-def card-image*)
then show *?thesis* **by** *blast*
next
assume ¬ *finite A*
with ‹$n \leq card\ A$› **show** *?thesis* **by** *force*
qed

lemma *closed-subspace*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *subspace s*
shows *closed s*
proof −
have *dim s* $\leq$ *card* (*Basis* :: *'a set*)
using *dim-subset-UNIV* **by** *auto*
with *ex-card*[*OF this*] **obtain** *d* :: *'a set* **where** *t*: *card d* = *dim s* **and** *d*: $d \subseteq Basis$
by *auto*
let *?t* = $\{x::'a.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x{\cdot}i = 0\}$
have $\exists f.\ linear\ f \wedge f\ `\ \{x::'a.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\} = s \wedge$
inj-on f $\{x::'a.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\}$
using *dim-substandard*[*of d*] *t d assms*
by (*intro subspace-isomorphism*[*OF subspace-substandard*[*of* $\lambda i.\ i \notin d$]]) (*auto simp*: *inner-Basis*)
then obtain *f* **where** *f*:
linear f
$f\ `\ \{x.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\} = s$
inj-on f $\{x.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\}$
by *blast*

interpret *f*: *bounded-linear f*
using *f* **unfolding** *linear-conv-bounded-linear* **by** *auto*
{
fix *x*
have *x*∈*?t* ⟹ *f x* = *0* ⟹ *x* = *0*
using *f.zero d f(3)*[*THEN inj-onD*, *of x 0*] **by** *auto*
}
moreover have *closed ?t* **using** *closed-substandard* **.**
moreover have *subspace ?t* **using** *subspace-substandard* **.**
ultimately show *?thesis*
using *closed-injective-image-subspace*[*of ?t f*]
unfolding *f(2)* **using** *f(1)* **unfolding** *linear-conv-bounded-linear* **by** *auto*
qed

lemma *complete-subspace*:
fixes *s* :: (*'a*::*euclidean-space*) *set*
shows *subspace s* ⟹ *complete s*
using *complete-eq-closed closed-subspace* **by** *auto*

lemma *dim-closure*:
fixes *s* :: (*'a*::*euclidean-space*) *set*
shows *dim*(*closure s*) = *dim s* (**is** *?dc* = *?d*)
proof −
have *?dc* ≤ *?d* **using** *closure-minimal*[*OF span-inc*, *of s*]
using *closed-subspace*[*OF subspace-span*, *of s*]
using *dim-subset*[*of closure s span s*]
unfolding *dim-span*
by *auto*
then show *?thesis* **using** *dim-subset*[*OF closure-subset*, *of s*]
by *auto*
qed

16.31 Affine transformations of intervals

lemma *real-affinity-le*:
0 < (*m*::*'a*::*linordered-field*) ⟹ (*m* ∗ *x* + *c* ≤ *y* ⟷ *x* ≤ *inverse*(*m*) ∗ *y* + −(*c* / *m*))
by (*simp add*: *field-simps*)

lemma *real-le-affinity*:
0 < (*m*::*'a*::*linordered-field*) ⟹ (*y* ≤ *m* ∗ *x* + *c* ⟷ *inverse*(*m*) ∗ *y* + −(*c* / *m*) ≤ *x*)
by (*simp add*: *field-simps*)

lemma *real-affinity-lt*:
0 < (*m*::*'a*::*linordered-field*) ⟹ (*m* ∗ *x* + *c* < *y* ⟷ *x* < *inverse*(*m*) ∗ *y* + −(*c* / *m*))
by (*simp add*: *field-simps*)

lemma *real-lt-affinity*:
$0 < (m::'a::linordered\text{-}field) \Longrightarrow (y < m * x + c \longleftrightarrow inverse(m) * y + -(c / m) < x)$
by (*simp add*: *field-simps*)

lemma *real-affinity-eq*:
$(m::'a::linordered\text{-}field) \neq 0 \Longrightarrow (m * x + c = y \longleftrightarrow x = inverse(m) * y + -(c / m))$
by (*simp add*: *field-simps*)

lemma *real-eq-affinity*:
$(m::'a::linordered\text{-}field) \neq 0 \Longrightarrow (y = m * x + c \longleftrightarrow inverse(m) * y + -(c / m) = x)$
by (*simp add*: *field-simps*)

16.32 Banach fixed point theorem (not really topological...)

theorem *banach-fix*:
assumes *s*: *complete s* $s \neq \{\}$
and *c*: $0 \leq c$ $c < 1$
and *f*: $(f \text{`} s) \subseteq s$
and *lipschitz*: $\forall x{\in}s.\ \forall y{\in}s.\ dist\ (f\ x)\ (f\ y) \leq c * dist\ x\ y$
shows $\exists!x{\in}s.\ f\ x = x$
proof −
have $1 - c > 0$ **using** *c* **by** *auto*

from *s*(*2*) **obtain** *z0* **where** $z0 \in s$ **by** *auto*
def $z \equiv \lambda n.\ (f\ \hat{}\hat{}\ n)\ z0$
{
fix $n :: nat$
have $z\ n \in s$ **unfolding** *z-def*
proof (*induct n*)
case *0*
then show *?case* **using** ‹*z0* ∈ *s*› **by** *auto*
next
case *Suc*
then show *?case* **using** *f* **by** *auto* **qed**
} **note** *z-in-s* = *this*

def $d \equiv dist\ (z\ 0)\ (z\ 1)$

have *fzn*:$\bigwedge n.\ f\ (z\ n) = z\ (Suc\ n)$ **unfolding** *z-def* **by** *auto*
{
fix $n :: nat$
have $dist\ (z\ n)\ (z\ (Suc\ n)) \leq (c\ \hat{}\ n) * d$
proof (*induct n*)
case *0*
then show *?case*
unfolding *d-def* **by** *auto*

```
    next
      case (Suc m)
      then have c * dist (z m) (z (Suc m)) ≤ c ^ Suc m * d
        using ‹0 ≤ c›
        using mult-left-mono[of dist (z m) (z (Suc m)) c ^ m * d c]
        by auto
      then show ?case
       using lipschitz[THEN bspec[where x=z m], OF z-in-s, THEN bspec[where
x=z (Suc m)], OF z-in-s]
        unfolding fzn and mult-le-cancel-left
        by auto
    qed
  } note cf-z = this

  {
    fix n m :: nat
    have (1 - c) * dist (z m) (z (m+n)) ≤ (c ^ m) * d * (1 - c ^ n)
    proof (induct n)
      case 0
      show ?case by auto
    next
      case (Suc k)
      have (1 - c) * dist (z m) (z (m + Suc k)) ≤
          (1 - c) * (dist (z m) (z (m + k)) + dist (z (m + k)) (z (Suc (m + k))))
        using dist-triangle and c by (auto simp add: dist-triangle)
      also have ... ≤ (1 - c) * (dist (z m) (z (m + k)) + c ^ (m + k) * d)
        using cf-z[of m + k] and c by auto
      also have ... ≤ c ^ m * d * (1 - c ^ k) + (1 - c) * c ^ (m + k) * d
        using Suc by (auto simp add: field-simps)
      also have ... = (c ^ m) * (d * (1 - c ^ k) + (1 - c) * c ^ k * d)
        unfolding power-add by (auto simp add: field-simps)
      also have ... ≤ (c ^ m) * d * (1 - c ^ Suc k)
        using c by (auto simp add: field-simps)
      finally show ?case by auto
    qed
  } note cf-z2 = this
  {
    fix e :: real
    assume e > 0
    then have ∃ N. ∀ m n. N ≤ m ∧ N ≤ n ⟶ dist (z m) (z n) < e
    proof (cases d = 0)
      case True
      have *: ⋀x. ((1 - c) * x ≤ 0) = (x ≤ 0) using ‹1 - c > 0›
        by (metis mult-zero-left mult.commute real-mult-le-cancel-iff1)
      from True have ⋀n. z n = z0 using cf-z2[of 0] and c unfolding z-def
        by (simp add: *)
      then show ?thesis using ‹e>0› by auto
    next
      case False
```

```
    then have d>0 unfolding d-def using zero-le-dist[of z 0 z 1]
      by (metis False d-def less-le)
    hence 0 < e * (1 - c) / d
      using ‹e>0› and ‹1-c>0› by auto
    then obtain N where N:c ^ N < e * (1 - c) / d
      using real-arch-pow-inv[of e * (1 - c) / d c] and c by auto
    {
      fix m n::nat
      assume m>n and as:m≥N n≥N
      have *:c ^ n ≤ c ^ N using ‹n≥N› and c
        using power-decreasing[OF ‹n≥N›, of c] by auto
      have 1 - c ^ (m - n) > 0
        using c and power-strict-mono[of c 1 m - n] using ‹m>n› by auto
      hence **: d * (1 - c ^ (m - n)) / (1 - c) > 0
        using ‹d>0› ‹0 < 1 - c› by auto

      have dist (z m) (z n) ≤ c ^ n * d * (1 - c ^ (m - n)) / (1 - c)
        using cf-z2[of n m - n] and ‹m>n›
        unfolding pos-le-divide-eq[OF ‹1-c>0›]
        by (auto simp add: mult.commute dist-commute)
      also have ... ≤ c ^ N * d * (1 - c ^ (m - n)) / (1 - c)
        using mult-right-mono[OF * order-less-imp-le[OF **]]
        unfolding mult.assoc by auto
      also have ... < (e * (1 - c) / d) * d * (1 - c ^ (m - n)) / (1 - c)
        using mult-strict-right-mono[OF N **] unfolding mult.assoc by auto
      also have ... = e * (1 - c ^ (m - n))
        using c and ‹d>0› and ‹1 - c > 0› by auto
      also have ... ≤ e using c and ‹1 - c ^ (m - n) > 0› and ‹e>0›
        using mult-right-le-one-le[of e 1 - c ^ (m - n)] by auto
      finally have  dist (z m) (z n) < e by auto
    } note * = this
    {
      fix m n :: nat
      assume as: N ≤ m N ≤ n
      then have dist (z n) (z m) < e
      proof (cases n = m)
        case True
        then show ?thesis using ‹e>0› by auto
      next
        case False
        then show ?thesis using as and *[of n m] *[of m n]
          unfolding nat-neq-iff by (auto simp add: dist-commute)
      qed
    }
    then show ?thesis by auto
  qed
}
then have Cauchy z
  unfolding cauchy-def by auto
```

```
  then obtain x where x∈s and x:(z ⟶ x) sequentially
    using s(1)[unfolded compact-def complete-def, THEN spec[where x=z]] and
z-in-s by auto

  def e ≡ dist (f x) x
  have e = 0
  proof (rule ccontr)
    assume e ≠ 0
    then have e > 0
      unfolding e-def using zero-le-dist[of f x x]
      by (metis dist-eq-0-iff dist-nz e-def)
    then obtain N where N:∀ n≥N. dist (z n) x < e / 2
      using x[unfolded lim-sequentially, THEN spec[where x=e/2]] by auto
    then have N':dist (z N) x < e / 2 by auto

    have *: c * dist (z N) x ≤ dist (z N) x
      unfolding mult-le-cancel-right2
      using zero-le-dist[of z N x] and c
      by (metis dist-eq-0-iff dist-nz order-less-asym less-le)
    have dist (f (z N)) (f x) ≤ c * dist (z N) x
      using lipschitz[THEN bspec[where x=z N], THEN bspec[where x=x]]
      using z-in-s[of N] ‹x∈s›
      using c
      by auto
    also have ... < e / 2
      using N' and c using * by auto
    finally show False
      unfolding fzn
     using N[THEN spec[where x=Suc N]] and dist-triangle-half-r[of z (Suc N)
f x e x]
      unfolding e-def
      by auto
  qed
  then have f x = x unfolding e-def by auto
  moreover
  {
    fix y
    assume f y = y y∈s
    then have dist x y ≤ c * dist x y
      using lipschitz[THEN bspec[where x=x], THEN bspec[where x=y]]
      using ‹x∈s› and ‹f x = x›
      by auto
    then have dist x y = 0
      unfolding mult-le-cancel-right1
      using c and zero-le-dist[of x y]
      by auto
    then have y = x by auto
  }
  ultimately show ?thesis using ‹x∈s› by blast+
```

qed

16.33 Edelstein fixed point theorem

theorem *edelstein-fix*:
fixes $s :: 'a::metric\text{-}space\ set$
assumes s: $compact\ s$ $s \neq \{\}$
and gs: $(g\ `\ s) \subseteq s$
and $dist$: $\forall x{\in}s.\ \forall y{\in}s.\ x \neq y \longrightarrow dist\ (g\ x)\ (g\ y) < dist\ x\ y$
shows $\exists!x{\in}s.\ g\ x = x$
proof −
let $?D = (\lambda x.\ (x,\ x))\ `\ s$
have D: $compact\ ?D$ $?D \neq \{\}$
by (*rule compact-continuous-image*)
(*auto intro*!: *s continuous-Pair continuous-within-id simp*: *continuous-on-eq-continuous-within*)

have $\bigwedge x\ y\ e.\ x \in s \Longrightarrow y \in s \Longrightarrow 0 < e \Longrightarrow dist\ y\ x < e \Longrightarrow dist\ (g\ y)\ (g\ x) < e$
using *dist* **by** *fastforce*
then have *continuous-on s g*
unfolding *continuous-on-iff* **by** *auto*
then have *cont*: *continuous-on ?D* $(\lambda x.\ dist\ ((g \circ fst)\ x)\ (snd\ x))$
unfolding *continuous-on-eq-continuous-within*
by (*intro continuous-dist ballI continuous-within-compose*)
(*auto intro*!: *continuous-fst continuous-snd continuous-within-id simp*: *image-image*)

obtain a **where** $a \in s$ **and** le: $\bigwedge x.\ x \in s \Longrightarrow dist\ (g\ a)\ a \leq dist\ (g\ x)\ x$
using *continuous-attains-inf*[*OF D cont*] **by** *auto*

have $g\ a = a$
proof (*rule ccontr*)
assume $g\ a \neq a$
with ‹$a \in s$› *gs* **have** $dist\ (g\ (g\ a))\ (g\ a) < dist\ (g\ a)\ a$
by (*intro dist*[*rule-format*]) *auto*
moreover have $dist\ (g\ a)\ a \leq dist\ (g\ (g\ a))\ (g\ a)$
using ‹$a \in s$› *gs* **by** (*intro le*) *auto*
ultimately show *False* **by** *auto*
qed
moreover have $\bigwedge x.\ x \in s \Longrightarrow g\ x = x \Longrightarrow x = a$
using *dist*[*THEN bspec*[**where** $x{=}a$]] ‹$g\ a = a$› **and** ‹$a{\in}s$› **by** *auto*
ultimately show $\exists!x{\in}s.\ g\ x = x$ **using** ‹$a \in s$› **by** *blast*
qed

lemma *cball-subset-cball-iff*:
fixes $a :: 'a :: euclidean\text{-}space$
shows $cball\ a\ r \subseteq cball\ a'\ r' \longleftrightarrow dist\ a\ a' + r \leq r' \vee r < 0$
(**is** *?lhs* = *?rhs*)
proof

assume *?lhs*
then show *?rhs*
proof (*cases r < 0*)
case *True* **then show** *?rhs* **by** *simp*
next
case *False*
then have [*simp*]: *r ≥ 0* **by** *simp*
have *norm (a − a′) + r ≤ r′*
proof (*cases a = a′*)
case *True* **then show** *?thesis*
using *subsetD* [**where** *c = a + r ∗R (SOME i. i ∈ Basis)*, *OF* ‹*?lhs*›] *subsetD* [**where** *c = a*, *OF* ‹*?lhs*›]
by (*force simp add*: *SOME-Basis dist-norm*)
next
case *False*
have *norm (a′ − (a + (r / norm (a − a′)) ∗R (a − a′))) = norm (a′ − a − (r / norm (a − a′)) ∗R (a − a′))*
by (*simp add*: *algebra-simps*)
also have *... = norm ((−1 − (r / norm (a − a′))) ∗R (a − a′))*
by (*simp add*: *algebra-simps*)
also have *... = |− norm (a − a′) − r|*
using ‹*a ≠ a′*› **by** (*simp add*: *abs-mult-pos field-simps*)
finally have [*simp*]: *norm (a′ − (a + (r / norm (a − a′)) ∗R (a − a′))) = |norm (a − a′) + r|* **by** *linarith*
show *?thesis*
using *subsetD* [**where** *c = a′ + (1 + r / norm(a − a′)) ∗R (a − a′)*, *OF* ‹*?lhs*›] ‹*a ≠ a′*›
by (*simp add*: *dist-norm scaleR-add-left*)
qed
then show *?rhs* **by** (*simp add*: *dist-norm*)
qed
next
assume *?rhs* **then show** *?lhs*
apply (*auto simp*: *ball-def dist-norm*)
apply (*metis add.commute add-le-cancel-right dist-norm dist-triangle-alt order-trans*)
done
qed

lemma *cball-subset-ball-iff*:
fixes *a* :: *′a* :: *euclidean-space*
shows *cball a r ⊆ ball a′ r′ ⟷ dist a a′ + r < r′ ∨ r < 0*
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
proof (*cases r < 0*)
case *True* **then show** *?rhs* **by** *simp*
next
case *False*

```
    then have [simp]: r ≥ 0 by simp
    have norm (a − a′) + r < r′
    proof (cases a = a′)
      case True then show ?thesis
        using subsetD [where c = a + r *R (SOME i. i ∈ Basis), OF ‹?lhs›]
subsetD [where c = a, OF ‹?lhs›]
        by (force simp add: SOME-Basis dist-norm)
    next
      case False
      { assume norm (a − a′) + r ≥ r′
        then have |r′ − norm (a − a′)| ≤ r
          apply (simp split: abs-split)
          by (metis ‹0 ≤ r› ‹?lhs› centre-in-cball dist-commute dist-norm less-asym
mem-ball subset-eq)
        then have False
        using subsetD [where c = a + (r′ / norm(a − a′) − 1) *R (a − a′), OF
‹?lhs›] ‹a ≠ a′›
        apply (simp add: dist-norm field-simps)
        apply (simp add: diff-divide-distrib scaleR-left-diff-distrib)
        done
      }
      then show ?thesis by force
    qed
    then show ?rhs by (simp add: dist-norm)
  qed
next
  assume ?rhs then show ?lhs
    apply (auto simp: ball-def dist-norm )
   apply (metis add.commute add-le-cancel-right dist-norm dist-triangle-alt le-less-trans)
    done
qed

lemma ball-subset-cball-iff:
  fixes a :: 'a :: euclidean-space
  shows ball a r ⊆ cball a′ r′ ⟷ dist a a′ + r ≤ r′ ∨ r ≤ 0
        (is ?lhs = ?rhs)
proof (cases r ≤ 0)
  case True then show ?thesis
    using dist-not-less-zero less-le-trans by force
next
  case False show ?thesis
  proof
    assume ?lhs
    then have (cball a r ⊆ cball a′ r′)
      by (metis False closed-cball closure-ball closure-closed closure-mono not-less)
    then show ?rhs
      using False cball-subset-cball-iff by fastforce
  next
    assume ?rhs with False show ?lhs
```

```
    using ball-subset-cball cball-subset-cball-iff by blast
  qed
qed

lemma ball-subset-ball-iff:
  fixes a :: 'a :: euclidean-space
  shows ball a r ⊆ ball a' r' ⟷ dist a a' + r ≤ r' ∨ r ≤ 0
      (is ?lhs = ?rhs)
proof (cases r ≤ 0)
  case True then show ?thesis
    using dist-not-less-zero less-le-trans by force
next
  case False show ?thesis
  proof
    assume ?lhs
    then have 0 < r'
     by (metis (no-types) False ‹?lhs› centre-in-ball dist-norm le-less-trans mem-ball
norm-ge-zero not-less set-mp)
    then have (cball a r ⊆ cball a' r')
      by (metis False‹?lhs› closure-ball closure-mono not-less)
    then show ?rhs
      using False cball-subset-cball-iff by fastforce
  next
  assume ?rhs then show ?lhs
    apply (auto simp: ball-def)
     apply (metis add.commute add-le-cancel-right dist-commute dist-triangle-lt
not-le order-trans)
    using dist-not-less-zero order.strict-trans2 apply blast
    done
  qed
qed


lemma ball-eq-ball-iff:
  fixes x :: 'a :: euclidean-space
  shows ball x d = ball y e ⟷ d ≤ 0 ∧ e ≤ 0 ∨ x=y ∧ d=e
      (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
  proof (cases d ≤ 0 ∨ e ≤ 0)
    case True
      with ‹?lhs› show ?rhs
       by safe (simp-all only: ball-eq-empty [of y e, symmetric] ball-eq-empty [of x
d, symmetric])
  next
    case False
    with ‹?lhs› show ?rhs
    apply (auto simp add: set-eq-subset ball-subset-ball-iff dist-norm norm-minus-commute
```

algebra-simps)
apply (*metis add-le-same-cancel1 le-add-same-cancel1 norm-ge-zero norm-pths*(*2*) *order-trans*)
apply (*metis add-increasing2 add-le-imp-le-right eq-iff norm-ge-zero*)
done
qed
next
assume *?rhs* **then show** *?lhs*
by (*auto simp add*: *set-eq-subset ball-subset-ball-iff*)
qed

lemma *cball-eq-cball-iff*:
fixes *x* :: *'a* :: *euclidean-space*
shows *cball x d* = *cball y e* ⟷ *d* < *0* ∧ *e* < *0* ∨ *x*=*y* ∧ *d*=*e*
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
proof (*cases d* < *0* ∨ *e* < *0*)
case *True*
with ‹*?lhs*› **show** *?rhs*
by *safe* (*simp-all only*: *cball-eq-empty* [*of y e*, *symmetric*] *cball-eq-empty* [*of x d*, *symmetric*])
next
case *False*
with ‹*?lhs*› **show** *?rhs*
apply (*auto simp add*: *set-eq-subset cball-subset-cball-iff dist-norm norm-minus-commute algebra-simps*)
apply (*metis add-le-same-cancel1 le-add-same-cancel1 norm-ge-zero norm-pths*(*2*) *order-trans*)
apply (*metis add-increasing2 add-le-imp-le-right eq-iff norm-ge-zero*)
done
qed
next
assume *?rhs* **then show** *?lhs*
by (*auto simp add*: *set-eq-subset cball-subset-cball-iff*)
qed

lemma *ball-eq-cball-iff*:
fixes *x* :: *'a* :: *euclidean-space*
shows *ball x d* = *cball y e* ⟷ *d* ≤ *0* ∧ *e* < *0* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
apply (*auto simp add*: *set-eq-subset ball-subset-cball-iff cball-subset-ball-iff algebra-simps*)
apply (*metis add-increasing2 add-le-cancel-right add-less-same-cancel1 dist-not-less-zero less-le-trans zero-le-dist*)
apply (*metis add-less-same-cancel1 dist-not-less-zero less-le-trans not-le*)
using ‹*?lhs*› *ball-eq-empty cball-eq-empty* **apply** *blast*+

done
next
assume *?rhs* **then show** *?lhs* **by** *auto*
qed

lemma *cball-eq-ball-iff*:
fixes $x :: 'a :: euclidean\text{-}space$
shows $cball\ x\ d = ball\ y\ e \longleftrightarrow d < 0 \wedge e \leq 0$
using *ball-eq-cball-iff* **by** *blast*

no-notation
eucl-less (**infix** $<e$ *50*)

end

17 Convexity in real vector spaces

theory *Convex*
imports *Product-Vector*
begin

17.1 Convexity

definition *convex* :: *'a::real-vector set* ⇒ *bool*
where $convex\ s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u{\geq}0.\ \forall v{\geq}0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$

lemma *convexI*:
assumes $\bigwedge x\ y\ u\ v.\ x \in s \Longrightarrow y \in s \Longrightarrow 0 \leq u \Longrightarrow 0 \leq v \Longrightarrow u + v = 1 \Longrightarrow u *_R x + v *_R y \in s$
shows *convex s*
using *assms* **unfolding** *convex-def* **by** *fast*

lemma *convexD*:
assumes *convex s* **and** $x \in s$ **and** $y \in s$ **and** $0 \leq u$ **and** $0 \leq v$ **and** $u + v = 1$
shows $u *_R x + v *_R y \in s$
using *assms* **unfolding** *convex-def* **by** *fast*

lemma *convex-alt*:
$convex\ s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u.\ 0 \leq u \wedge u \leq 1 \longrightarrow ((1 - u) *_R x + u *_R y) \in s)$
(**is** - $\longleftrightarrow$ *?alt*)
proof
assume *alt*[*rule-format*]: *?alt*
{
fix *x y* **and** *u v* :: *real*
assume *mem*: $x \in s\ y \in s$
assume $0 \leq u\ 0 \leq v$
moreover

```
    assume u + v = 1
    then have u = 1 − v by auto
    ultimately have u *R x + v *R y ∈ s
      using alt[OF mem] by auto
  }
  then show convex s
    unfolding convex-def by auto
qed (auto simp: convex-def)

lemma convexD-alt:
  assumes convex s a ∈ s b ∈ s 0 ≤ u u ≤ 1
  shows ((1 − u) *R a + u *R b) ∈ s
  using assms unfolding convex-alt by auto

lemma mem-convex-alt:
  assumes convex S x ∈ S y ∈ S u ≥ 0 v ≥ 0 u + v > 0
  shows ((u/(u+v)) *R x + (v/(u+v)) *R y) ∈ S
  apply (rule convexD)
  using assms
  apply (simp-all add: zero-le-divide-iff add-divide-distrib [symmetric])
  done

lemma convex-empty[intro,simp]: convex {}
  unfolding convex-def by simp

lemma convex-singleton[intro,simp]: convex {a}
  unfolding convex-def by (auto simp: scaleR-left-distrib[symmetric])

lemma convex-UNIV[intro,simp]: convex UNIV
  unfolding convex-def by auto

lemma convex-Inter: (∀ s∈f. convex s) ⟹ convex(⋂f)
  unfolding convex-def by auto

lemma convex-Int: convex s ⟹ convex t ⟹ convex (s ∩ t)
  unfolding convex-def by auto

lemma convex-INT: ∀ i∈A. convex (B i) ⟹ convex (⋂ i∈A. B i)
  unfolding convex-def by auto

lemma convex-Times: convex s ⟹ convex t ⟹ convex (s × t)
  unfolding convex-def by auto

lemma convex-halfspace-le: convex {x. inner a x ≤ b}
  unfolding convex-def
  by (auto simp: inner-add intro!: convex-bound-le)

lemma convex-halfspace-ge: convex {x. inner a x ≥ b}
proof −
```

```
  have ∗: {x. inner a x ≥ b} = {x. inner (−a) x ≤ −b}
    by auto
  show ?thesis
    unfolding ∗ using convex-halfspace-le[of −a −b] by auto
qed

lemma convex-hyperplane: convex {x. inner a x = b}
proof −
  have ∗: {x. inner a x = b} = {x. inner a x ≤ b} ∩ {x. inner a x ≥ b}
    by auto
  show ?thesis using convex-halfspace-le convex-halfspace-ge
    by (auto intro!: convex-Int simp: ∗)
qed

lemma convex-halfspace-lt: convex {x. inner a x < b}
  unfolding convex-def
  by (auto simp: convex-bound-lt inner-add)

lemma convex-halfspace-gt: convex {x. inner a x > b}
   using convex-halfspace-lt[of −a −b] by auto

lemma convex-real-interval [iff]:
  fixes a b :: real
  shows convex {a..} and convex {..b}
    and convex {a<..} and convex {..<b}
    and convex {a..b} and convex {a<..b}
    and convex {a..<b} and convex {a<..<b}
proof −
  have {a..} = {x. a ≤ inner 1 x}
    by auto
  then show 1: convex {a..}
    by (simp only: convex-halfspace-ge)
  have {..b} = {x. inner 1 x ≤ b}
    by auto
  then show 2: convex {..b}
    by (simp only: convex-halfspace-le)
  have {a<..} = {x. a < inner 1 x}
    by auto
  then show 3: convex {a<..}
    by (simp only: convex-halfspace-gt)
  have {..<b} = {x. inner 1 x < b}
    by auto
  then show 4: convex {..<b}
    by (simp only: convex-halfspace-lt)
  have {a..b} = {a..} ∩ {..b}
    by auto
  then show convex {a..b}
    by (simp only: convex-Int 1 2)
  have {a<..b} = {a<..} ∩ {..b}
```

by *auto*
then show *convex* $\{a<..b\}$
by (*simp only*: *convex-Int 3 2*)
have $\{a..<b\} = \{a..\} \cap \{..<b\}$
by *auto*
then show *convex* $\{a..<b\}$
by (*simp only*: *convex-Int 1 4*)
have $\{a<..<b\} = \{a<..\} \cap \{..<b\}$
by *auto*
then show *convex* $\{a<..<b\}$
by (*simp only*: *convex-Int 3 4*)
qed

lemma *convex-Reals*: *convex* $\mathbb{R}$
by (*simp add*: *convex-def scaleR-conv-of-real*)

17.2 Explicit expressions for convexity in terms of arbitrary sums

lemma *convex-setsum*:
fixes C :: *'a*::*real-vector set*
assumes *finite s*
and *convex C*
and $(\sum\ i \in s.\ a\ i) = 1$
assumes $\bigwedge i.\ i \in s \Longrightarrow a\ i \geq 0$
and $\bigwedge i.\ i \in s \Longrightarrow y\ i \in C$
shows $(\sum\ j \in s.\ a\ j *_R y\ j) \in C$
using *assms*(*1*,*3*,*4*,*5*)
proof (*induct arbitrary*: *a set*: *finite*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert i s*) **note** *IH* = *this*(*3*)
have *a i* + *setsum a s* = *1*
and *0* $\leq$ *a i*
and $\forall j{\in}s.\ 0 \leq a\ j$
and *y i* $\in$ *C*
and $\forall j{\in}s.\ y\ j \in C$
using *insert.hyps*(*1*,*2*) *insert.prems* **by** *simp-all*
then have *0* $\leq$ *setsum a s*
by (*simp add*: *setsum-nonneg*)
have $a\ i *_R y\ i + (\sum j{\in}s.\ a\ j *_R y\ j) \in C$
proof (*cases*)
assume *z*: *setsum a s* = *0*
with ‹*a i* + *setsum a s* = *1*› **have** *a i* = *1*
by *simp*
from *setsum-nonneg-0* [*OF* ‹*finite s*› - *z*] ‹$\forall j{\in}s.\ 0 \leq a\ j$› **have** $\forall j{\in}s.\ a\ j = 0$
by *simp*
show *?thesis* **using** ‹*a i* = *1*› **and** ‹$\forall j{\in}s.\ a\ j = 0$› **and** ‹*y i* $\in$ *C*›

```
    by simp
  next
    assume nz: setsum a s ≠ 0
    with ‹0 ≤ setsum a s› have 0 < setsum a s
      by simp
    then have (∑j∈s. (a j / setsum a s) *R y j) ∈ C
      using ‹∀j∈s. 0 ≤ a j› and ‹∀j∈s. y j ∈ C›
      by (simp add: IH setsum-divide-distrib [symmetric])
    from ‹convex C› and ‹y i ∈ C› and this and ‹0 ≤ a i›
      and ‹0 ≤ setsum a s› and ‹a i + setsum a s = 1›
    have a i *R y i + setsum a s *R (∑j∈s. (a j / setsum a s) *R y j) ∈ C
      by (rule convexD)
    then show ?thesis
      by (simp add: scaleR-setsum-right nz)
  qed
  then show ?case using ‹finite s› and ‹i ∉ s›
    by simp
qed

lemma convex:
  convex s ⟷ (∀(k::nat) u x. (∀i. 1≤i ∧ i≤k ⟶ 0 ≤ u i ∧ x i ∈s) ∧ (setsum
u {1..k} = 1)
      ⟶ setsum (λi. u i *R x i) {1..k} ∈ s)
proof safe
  fix k :: nat
  fix u :: nat ⇒ real
  fix x
  assume convex s
    ∀i. 1 ≤ i ∧ i ≤ k ⟶ 0 ≤ u i ∧ x i ∈ s
    setsum u {1..k} = 1
  with convex-setsum[of {1 .. k} s] show (∑j∈{1 .. k}. u j *R x j) ∈ s
    by auto
next
  assume *: ∀k u x. (∀ i :: nat. 1 ≤ i ∧ i ≤ k ⟶ 0 ≤ u i ∧ x i ∈ s) ∧ setsum
u {1..k} = 1
    ⟶ (∑i = 1..k. u i *R (x i :: 'a)) ∈ s
  {
    fix μ :: real
    fix x y :: 'a
    assume xy: x ∈ s y ∈ s
    assume mu: μ ≥ 0 μ ≤ 1
    let ?u = λi. if (i :: nat) = 1 then μ else 1 − μ
    let ?x = λi. if (i :: nat) = 1 then x else y
    have {1 :: nat .. 2} ∩ − {x. x = 1} = {2}
      by auto
    then have card: card ({1 :: nat .. 2} ∩ − {x. x = 1}) = 1
      by simp
    then have setsum ?u {1 .. 2} = 1
      using setsum.If-cases[of {(1 :: nat) .. 2} λ x. x = 1 λ x. μ λ x. 1 − μ]
```

```
    by auto
  with *[rule-format, of 2 ?u ?x] have s: (∑j ∈ {1..2}. ?u j *R ?x j) ∈ s
    using mu xy by auto
  have grarr: (∑j ∈ {Suc (Suc 0)..2}. ?u j *R ?x j) = (1 - μ) *R y
    using setsum-head-Suc[of Suc (Suc 0) 2 λ j. (1 - μ) *R y] by auto
  from setsum-head-Suc[of Suc 0 2 λ j. ?u j *R ?x j, simplified this]
  have (∑j ∈ {1..2}. ?u j *R ?x j) = μ *R x + (1 - μ) *R y
    by auto
  then have (1 - μ) *R y + μ *R x ∈ s
    using s by (auto simp: add.commute)
 }
 then show convex s
  unfolding convex-alt by auto
qed


lemma convex-explicit:
 fixes s :: 'a::real-vector set
 shows convex s ⟷
  (∀ t u. finite t ∧ t ⊆ s ∧ (∀ x∈t. 0 ≤ u x) ∧ setsum u t = 1 ⟶ setsum (λx.
u x *R x) t ∈ s)
proof safe
 fix t
 fix u :: 'a ⇒ real
 assume convex s
  and finite t
  and t ⊆ s ∀ x∈t. 0 ≤ u x setsum u t = 1
 then show (∑ x∈t. u x *R x) ∈ s
  using convex-setsum[of t s u λ x. x] by auto
next
 assume *: ∀ t. ∀ u. finite t ∧ t ⊆ s ∧ (∀ x∈t. 0 ≤ u x) ∧
  setsum u t = 1 ⟶ (∑ x∈t. u x *R x) ∈ s
 show convex s
  unfolding convex-alt
 proof safe
  fix x y
  fix μ :: real
  assume **: x ∈ s y ∈ s 0 ≤ μ μ ≤ 1
  show (1 - μ) *R x + μ *R y ∈ s
  proof (cases x = y)
   case False
   then show ?thesis
    using *[rule-format, of {x, y} λ z. if z = x then 1 - μ else μ] **
     by auto
  next
   case True
   then show ?thesis
    using *[rule-format, of {x, y} λ z. 1] **
     by (auto simp: field-simps real-vector.scale-left-diff-distrib)
```

qed
qed
qed

lemma *convex-finite*:
assumes *finite s*
shows *convex s* $\longleftrightarrow$ $(\forall u.\ (\forall x{\in}s.\ 0 \leq u\ x) \wedge setsum\ u\ s = 1 \longrightarrow setsum\ (\lambda x.\ u\ x *_R x)\ s \in s)$
unfolding *convex-explicit*
proof *safe*
fix *t u*
assume *sum*: $\forall u.\ (\forall x{\in}s.\ 0 \leq u\ x) \wedge setsum\ u\ s = 1 \longrightarrow (\sum x{\in}s.\ u\ x *_R x) \in s$
and *as*: *finite t* $t \subseteq s$ $\forall x{\in}t.\ 0 \leq u\ x$ *setsum u t* = (*1*::*real*)
have $*$: $s \cap t = t$
using *as*(*2*) **by** *auto*
have *if-distrib-arg*: $\bigwedge P\ f\ g\ x.\ (if\ P\ then\ f\ else\ g)\ x = (if\ P\ then\ f\ x\ else\ g\ x)$
by *simp*
show $(\sum x{\in}t.\ u\ x *_R x) \in s$
using *sum*[*THEN spec*[**where** $x{=}\lambda x.\ if\ x{\in}t\ then\ u\ x\ else\ 0$]] *as* $*$
by (*auto simp*: *assms setsum.If-cases if-distrib if-distrib-arg*)
qed (*erule-tac x*=*s* **in** *allE*, *erule-tac x*=*u* **in** *allE*, *auto*)

17.3 Functions that are convex on a set

definition *convex-on* :: *'a*::*real-vector set* $\Rightarrow$ (*'a* $\Rightarrow$ *real*) $\Rightarrow$ *bool*
where *convex-on s f* $\longleftrightarrow$
$(\forall x{\in}s.\ \forall y{\in}s.\ \forall u{\geq}0.\ \forall v{\geq}0.\ u + v = 1 \longrightarrow f\ (u *_R x + v *_R y) \leq u * f\ x + v * f\ y)$

lemma *convex-onI* [*intro?*]:
assumes $\bigwedge t\ x\ y.\ t > 0 \Longrightarrow t < 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
shows *convex-on A f*
unfolding *convex-on-def*
proof *clarify*
fix *x y u v* **assume** *A*: $x \in A$ $y \in A$ $(u{::}real) \geq 0$ $v \geq 0$ $u + v = 1$
from *A*(*5*) **have** [*simp*]: $v = 1 - u$ **by** (*simp add*: *algebra-simps*)
from *A*(*1*−*4*) **show** $f\ (u *_R x + v *_R y) \leq u * f\ x + v * f\ y$ **using** *assms*[*of u y x*]
by (*cases* $u = 0 \vee u = 1$) (*auto simp*: *algebra-simps*)
qed

lemma *convex-on-linorderI* [*intro?*]:
fixes *A* :: (*'a*::{*linorder*,*real-vector*}) *set*
assumes $\bigwedge t\ x\ y.\ t > 0 \Longrightarrow t < 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x < y \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
shows *convex-on A f*
proof

fix *t x y* **assume** *A*: $x \in A$ $y \in A$ $(t::real) > 0$ $t < 1$
case (*goal1 t x y*)
with *goal1 assms*[*of t x y*] *assms*[*of 1 − t y x*]
show *?case* **by** (*cases x y rule*: *linorder-cases*) (*auto simp*: *algebra-simps*)
qed

lemma *convex-onD*:
assumes *convex-on A f*
shows $\bigwedge t\ x\ y.\ t \geq 0 \Longrightarrow t \leq 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
using *assms* **unfolding** *convex-on-def* **by** *auto*

lemma *convex-onD-Icc*:
assumes *convex-on* $\{x..y\}$ $f\ x \leq (y :: - :: \{real\text{-}vector,preorder\})$
shows $\bigwedge t.\ t \geq 0 \Longrightarrow t \leq 1 \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
using *assms*(*2*) **by** (*intro convex-onD*[*OF assms*(*1*)]) *simp-all*

lemma *convex-on-subset*: *convex-on t f* $\Longrightarrow$ $s \subseteq t \Longrightarrow$ *convex-on s f*
unfolding *convex-on-def* **by** *auto*

lemma *convex-on-add* [*intro*]:
assumes *convex-on s f*
and *convex-on s g*
shows *convex-on s* $(\lambda x.\ f\ x + g\ x)$
proof −
{
fix *x y*
assume $x \in s$ $y \in s$
moreover
fix *u v* :: *real*
assume $0 \leq u$ $0 \leq v$ $u + v = 1$
ultimately
have $f\ (u *_R x + v *_R y) + g\ (u *_R x + v *_R y) \leq (u * f\ x + v * f\ y) + (u * g\ x + v * g\ y)$
using *assms* **unfolding** *convex-on-def* **by** (*auto simp*: *add-mono*)
then have $f\ (u *_R x + v *_R y) + g\ (u *_R x + v *_R y) \leq u * (f\ x + g\ x) + v * (f\ y + g\ y)$
by (*simp add*: *field-simps*)
}
then show *?thesis*
unfolding *convex-on-def* **by** *auto*
qed

lemma *convex-on-cmul* [*intro*]:
fixes *c* :: *real*
assumes $0 \leq c$
and *convex-on s f*
shows *convex-on s* $(\lambda x.\ c * f\ x)$

proof −
have ∗: ⋀*u c fx v fy* :: *real*. *u* ∗ (*c* ∗ *fx*) + *v* ∗ (*c* ∗ *fy*) = *c* ∗ (*u* ∗ *fx* + *v* ∗ *fy*)
by (*simp add*: *field-simps*)
show *?thesis* **using** *assms*(*2*) **and** *mult-left-mono* [*OF* - *assms*(*1*)]
unfolding *convex-on-def* **and** ∗ **by** *auto*
qed

lemma *convex-lower*:
assumes *convex-on s f*
and *x* ∈ *s*
and *y* ∈ *s*
and *0* ≤ *u*
and *0* ≤ *v*
and *u* + *v* = *1*
shows *f* (*u* ∗*R x* + *v* ∗*R y*) ≤ *max* (*f x*) (*f y*)
proof −
let *?m* = *max* (*f x*) (*f y*)
have *u* ∗ *f x* + *v* ∗ *f y* ≤ *u* ∗ *max* (*f x*) (*f y*) + *v* ∗ *max* (*f x*) (*f y*)
using *assms*(*4*,*5*) **by** (*auto simp*: *mult-left-mono add-mono*)
also have ... = *max* (*f x*) (*f y*)
using *assms*(*6*) **by** (*simp add*: *distrib-right* [*symmetric*])
finally show *?thesis*
using *assms* **unfolding** *convex-on-def* **by** *fastforce*
qed

lemma *convex-on-dist* [*intro*]:
fixes *s* :: *'a*::*real-normed-vector set*
shows *convex-on s* (λ*x*. *dist a x*)
proof (*auto simp*: *convex-on-def dist-norm*)
fix *x y*
assume *x* ∈ *s y* ∈ *s*
fix *u v* :: *real*
assume *0* ≤ *u*
assume *0* ≤ *v*
assume *u* + *v* = *1*
have *a* = *u* ∗*R a* + *v* ∗*R a*
unfolding *scaleR-left-distrib*[*symmetric*] **and** ‹*u* + *v* = *1*› **by** *simp*
then have ∗: *a* − (*u* ∗*R x* + *v* ∗*R y*) = (*u* ∗*R* (*a* − *x*)) + (*v* ∗*R* (*a* − *y*))
by (*auto simp*: *algebra-simps*)
show *norm* (*a* − (*u* ∗*R x* + *v* ∗*R y*)) ≤ *u* ∗ *norm* (*a* − *x*) + *v* ∗ *norm* (*a* − *y*)
unfolding ∗ **using** *norm-triangle-ineq*[*of u* ∗*R* (*a* − *x*) *v* ∗*R* (*a* − *y*)]
using ‹*0* ≤ *u*› ‹*0* ≤ *v*› **by** *auto*
qed

17.4 Arithmetic operations on sets preserve convexity

lemma *convex-linear-image*:
assumes *linear f*
and *convex s*

```
  shows convex (f ` s)
proof −
  interpret f: linear f by fact
  from ‹convex s› show convex (f ` s)
    by (simp add: convex-def f.scaleR [symmetric] f.add [symmetric])
qed

lemma convex-linear-vimage:
  assumes linear f
    and convex s
  shows convex (f −` s)
proof −
  interpret f: linear f by fact
  from ‹convex s› show convex (f −` s)
    by (simp add: convex-def f.add f.scaleR)
qed

lemma convex-scaling:
  assumes convex s
  shows convex ((λx. c *R x) ` s)
proof −
  have linear (λx. c *R x)
    by (simp add: linearI scaleR-add-right)
  then show ?thesis
    using ‹convex s› by (rule convex-linear-image)
qed

lemma convex-scaled:
  assumes convex s
  shows convex ((λx. x *R c) ` s)
proof −
  have linear (λx. x *R c)
    by (simp add: linearI scaleR-add-left)
  then show ?thesis
    using ‹convex s› by (rule convex-linear-image)
qed

lemma convex-negations:
  assumes convex s
  shows convex ((λx. − x) ` s)
proof −
  have linear (λx. − x)
    by (simp add: linearI)
  then show ?thesis
    using ‹convex s› by (rule convex-linear-image)
qed

lemma convex-sums:
  assumes convex s
```

```
    and convex t
  shows convex {x + y| x y. x ∈ s ∧ y ∈ t}
proof −
  have linear (λ(x, y). x + y)
    by (auto intro: linearI simp: scaleR-add-right)
  with assms have convex ((λ(x, y). x + y) ‘ (s × t))
    by (intro convex-linear-image convex-Times)
  also have ((λ(x, y). x + y) ‘ (s × t)) = {x + y| x y. x ∈ s ∧ y ∈ t}
    by auto
  finally show ?thesis .
qed

lemma convex-differences:
  assumes convex s convex t
  shows convex {x − y| x y. x ∈ s ∧ y ∈ t}
proof −
  have {x − y| x y. x ∈ s ∧ y ∈ t} = {x + y |x y. x ∈ s ∧ y ∈ uminus ‘ t}
    by (auto simp: diff-conv-add-uminus simp del: add-uminus-conv-diff)
  then show ?thesis
    using convex-sums[OF assms(1) convex-negations[OF assms(2)]] by auto
qed

lemma convex-translation:
  assumes convex s
  shows convex ((λx. a + x) ‘ s)
proof −
  have {a + y |y. y ∈ s} = (λx. a + x) ‘ s
    by auto
  then show ?thesis
    using convex-sums[OF convex-singleton[of a] assms] by auto
qed

lemma convex-affinity:
  assumes convex s
  shows convex ((λx. a + c *R x) ‘ s)
proof −
  have (λx. a + c *R x) ‘ s = op + a ‘ op *R c ‘ s
    by auto
  then show ?thesis
    using convex-translation[OF convex-scaling[OF assms], of a c] by auto
qed

lemma pos-is-convex: convex {0 :: real <..}
  unfolding convex-alt
proof safe
  fix y x μ :: real
  assume *: y > 0 x > 0 μ ≥ 0 μ ≤ 1
  {
    assume μ = 0
```

```
    then have μ *R x + (1 - μ) *R y = y by simp
    then have μ *R x + (1 - μ) *R y > 0 using * by simp
  }
  moreover
  {
    assume μ = 1
    then have μ *R x + (1 - μ) *R y > 0 using * by simp
  }
  moreover
  {
    assume μ ≠ 1 μ ≠ 0
    then have μ > 0 (1 - μ) > 0 using * by auto
    then have μ *R x + (1 - μ) *R y > 0 using *
      by (auto simp: add-pos-pos)
  }
  ultimately show (1 - μ) *R y + μ *R x > 0
    using assms by fastforce
qed

lemma convex-on-setsum:
  fixes a :: 'a ⇒ real
    and y :: 'a ⇒ 'b::real-vector
    and f :: 'b ⇒ real
  assumes finite s s ≠ {}
    and convex-on C f
    and convex C
    and (∑ i ∈ s. a i) = 1
    and ⋀i. i ∈ s ⟹ a i ≥ 0
    and ⋀i. i ∈ s ⟹ y i ∈ C
  shows f (∑ i ∈ s. a i *R y i) ≤ (∑ i ∈ s. a i * f (y i))
  using assms
proof (induct s arbitrary: a rule: finite-ne-induct)
  case (singleton i)
  then have ai: a i = 1 by auto
  then show ?case by auto
next
  case (insert i s)
  then have convex-on C f by simp
  from this[unfolded convex-on-def, rule-format]
  have conv: ⋀x y μ. x ∈ C ⟹ y ∈ C ⟹ 0 ≤ μ ⟹ μ ≤ 1 ⟹
      f (μ *R x + (1 - μ) *R y) ≤ μ * f x + (1 - μ) * f y
    by simp
  show ?case
  proof (cases a i = 1)
    case True
    then have (∑ j ∈ s. a j) = 0
      using insert by auto
    then have ⋀j. j ∈ s ⟹ a j = 0
      using setsum-nonneg-0[where 'b=real] insert by fastforce
```

then show *?thesis*
using *insert* **by** *auto*
next
case *False*
from *insert* **have** *yai*: $y\ i \in C\ a\ i \geq 0$
by *auto*
have *fis*: *finite* (*insert i s*)
using *insert* **by** *auto*
then have *ai1*: $a\ i \leq 1$
using *setsum-nonneg-leq-bound*[*of insert i s a*] *insert* **by** *simp*
then have $a\ i < 1$
using *False* **by** *auto*
then have *i0*: $1 - a\ i > 0$
by *auto*
let *?a* = $\lambda j.\ a\ j\ /\ (1 - a\ i)$
have *a-nonneg*: $?a\ j \geq 0$ **if** $j \in s$ **for** j
using *i0 insert that* **by** *fastforce*
have $(\sum\ j \in insert\ i\ s.\ a\ j) = 1$
using *insert* **by** *auto*
then have $(\sum\ j \in s.\ a\ j) = 1 - a\ i$
using *setsum.insert insert* **by** *fastforce*
then have $(\sum\ j \in s.\ a\ j)\ /\ (1 - a\ i) = 1$
using *i0* **by** *auto*
then have *a1*: $(\sum\ j \in s.\ ?a\ j) = 1$
unfolding *setsum-divide-distrib* **by** *simp*
have *convex C* **using** *insert* **by** *auto*
then have *asum*: $(\sum\ j \in s.\ ?a\ j\ *_R\ y\ j) \in C$
using *insert convex-setsum*[*OF* ‹*finite s*› ‹*convex C*› *a1 a-nonneg*] **by** *auto*
have *asum-le*: $f\ (\sum\ j \in s.\ ?a\ j\ *_R\ y\ j) \leq (\sum\ j \in s.\ ?a\ j * f\ (y\ j))$
using *a-nonneg a1 insert* **by** *blast*
have $f\ (\sum\ j \in insert\ i\ s.\ a\ j\ *_R\ y\ j) = f\ ((\sum\ j \in s.\ a\ j\ *_R\ y\ j) + a\ i\ *_R\ y\ i)$
using *setsum.insert*[*of s i* $\lambda\ j.\ a\ j\ *_R\ y\ j$, *OF* ‹*finite s*› ‹$i \notin s$›] *insert*
by (*auto simp only*: *add.commute*)
also have $\ldots = f\ (((1 - a\ i) * inverse\ (1 - a\ i))\ *_R\ (\sum\ j \in s.\ a\ j\ *_R\ y\ j)$
$+\ a\ i\ *_R\ y\ i)$
using *i0* **by** *auto*
also have $\ldots = f\ ((1 - a\ i)\ *_R\ (\sum\ j \in s.\ (a\ j * inverse\ (1 - a\ i))\ *_R\ y\ j)$
$+\ a\ i\ *_R\ y\ i)$
using *scaleR-right.setsum*[*of inverse* $(1 - a\ i)$ $\lambda\ j.\ a\ j\ *_R\ y\ j\ s$, *symmetric*]
by (*auto simp*: *algebra-simps*)
also have $\ldots = f\ ((1 - a\ i)\ *_R\ (\sum\ j \in s.\ ?a\ j\ *_R\ y\ j) + a\ i\ *_R\ y\ i)$
by (*auto simp*: *divide-inverse*)
also have $\ldots \leq (1 - a\ i)\ *_R\ f\ ((\sum\ j \in s.\ ?a\ j\ *_R\ y\ j)) + a\ i * f\ (y\ i)$
using *conv*[*of y i* $(\sum\ j \in s.\ ?a\ j\ *_R\ y\ j)$ *a i*, *OF yai*(1) *asum yai*(2) *ai1*]
by (*auto simp*: *add.commute*)
also have $\ldots \leq (1 - a\ i) * (\sum\ j \in s.\ ?a\ j * f\ (y\ j)) + a\ i * f\ (y\ i)$
using *add-right-mono*[*OF mult-left-mono*[*of - - 1 − a i*,
OF asum-le less-imp-le[*OF i0*]], *of a i* $*\ f\ (y\ i)$] **by** *simp*

```
    also have ... = (∑ j ∈ s. (1 − a i) * ?a j * f (y j)) + a i * f (y i)
      unfolding setsum-right-distrib[of 1 − a i λ j. ?a j * f (y j)] using i0 by
auto
    also have ... = (∑ j ∈ s. a j * f (y j)) + a i * f (y i)
      using i0 by auto
    also have ... = (∑ j ∈ insert i s. a j * f (y j))
      using insert by auto
    finally show ?thesis
      by simp
  qed
qed

lemma convex-on-alt:
  fixes C :: 'a::real-vector set
  assumes convex C
  shows convex-on C f ⟷
    (∀ x ∈ C. ∀ y ∈ C. ∀ μ :: real. μ ≥ 0 ∧ μ ≤ 1 ⟶
      f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y)
proof safe
  fix x y
  fix μ :: real
  assume *: convex-on C f x ∈ C y ∈ C 0 ≤ μ μ ≤ 1
  from this[unfolded convex-on-def, rule-format]
  have ⋀u v. 0 ≤ u ⟹ 0 ≤ v ⟹ u + v = 1 ⟹ f (u *R x + v *R y) ≤ u * f
x + v * f y
    by auto
  from this[of μ 1 − μ, simplified] *
  show f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y
    by auto
next
  assume *: ∀ x∈C. ∀ y∈C. ∀ μ. 0 ≤ μ ∧ μ ≤ 1 ⟶
    f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y
  {
    fix x y
    fix u v :: real
    assume **: x ∈ C y ∈ C u ≥ 0 v ≥ 0 u + v = 1
    then have[simp]: 1 − u = v by auto
    from *[rule-format, of x y u]
    have f (u *R x + v *R y) ≤ u * f x + v * f y
      using ** by auto
  }
  then show convex-on C f
    unfolding convex-on-def by auto
qed

lemma convex-on-diff:
  fixes f :: real ⇒ real
  assumes f: convex-on I f
    and I: x ∈ I y ∈ I
```

```
    and t: x < t t < y
  shows (f x − f t) / (x − t) ≤ (f x − f y) / (x − y)
    and (f x − f y) / (x − y) ≤ (f t − f y) / (t − y)
proof −
  def a ≡ (t − y) / (x − y)
  with t have 0 ≤ a 0 ≤ 1 − a
    by (auto simp: field-simps)
  with f ‹x ∈ I› ‹y ∈ I› have cvx: f (a ∗ x + (1 − a) ∗ y) ≤ a ∗ f x + (1 − a)
∗ f y
    by (auto simp: convex-on-def)
  have a ∗ x + (1 − a) ∗ y = a ∗ (x − y) + y
    by (simp add: field-simps)
  also have ... = t
    unfolding a-def using ‹x < t› ‹t < y› by simp
  finally have f t ≤ a ∗ f x + (1 − a) ∗ f y
    using cvx by simp
  also have ... = a ∗ (f x − f y) + f y
    by (simp add: field-simps)
  finally have f t − f y ≤ a ∗ (f x − f y)
    by simp
  with t show (f x − f t) / (x − t) ≤ (f x − f y) / (x − y)
    by (simp add: le-divide-eq divide-le-eq field-simps a-def)
  with t show (f x − f y) / (x − y) ≤ (f t − f y) / (t − y)
    by (simp add: le-divide-eq divide-le-eq field-simps)
qed

lemma pos-convex-function:
  fixes f :: real ⇒ real
  assumes convex C
    and leq: ⋀x y. x ∈ C ⟹ y ∈ C ⟹ f′ x ∗ (y − x) ≤ f y − f x
  shows convex-on C f
  unfolding convex-on-alt[OF assms(1)]
  using assms
proof safe
  fix x y μ :: real
  let ?x = μ ∗R x + (1 − μ) ∗R y
  assume ∗: convex C x ∈ C y ∈ C μ ≥ 0 μ ≤ 1
  then have 1 − μ ≥ 0 by auto
  then have xpos: ?x ∈ C
    using ∗ unfolding convex-alt by fastforce
  have geq: μ ∗ (f x − f ?x) + (1 − μ) ∗ (f y − f ?x) ≥
      μ ∗ f′ ?x ∗ (x − ?x) + (1 − μ) ∗ f′ ?x ∗ (y − ?x)
    using add-mono[OF mult-left-mono[OF leq[OF xpos ∗(2)] ‹μ ≥ 0›]
      mult-left-mono[OF leq[OF xpos ∗(3)] ‹1 − μ ≥ 0›]]
    by auto
  then have μ ∗ f x + (1 − μ) ∗ f y − f ?x ≥ 0
    by (auto simp: field-simps)
  then show f (μ ∗R x + (1 − μ) ∗R y) ≤ μ ∗ f x + (1 − μ) ∗ f y
    using convex-on-alt by auto
```

qed

lemma *atMostAtLeast-subset-convex*:
fixes C :: *real set*
assumes *convex* C
and $x \in C$ $y \in C$ $x < y$
shows $\{x \,..\, y\} \subseteq C$
proof *safe*
fix z **assume** z: $z \in \{x \,..\, y\}$
have *less*: $z \in C$ **if** $*$: $x < z$ $z < y$
proof −
let $?\mu = (y - z) \,/\, (y - x)$
have $0 \leq ?\mu$ $?\mu \leq 1$
using *assms* $*$ **by** (*auto simp*: *field-simps*)
then have *comb*: $?\mu * x + (1 - ?\mu) * y \in C$
using *assms iffD1*[*OF convex-alt*, *rule-format*, *of C y x* $?\mu$]
by (*simp add*: *algebra-simps*)
have $?\mu * x + (1 - ?\mu) * y = (y - z) * x \,/\, (y - x) + (1 - (y - z) \,/\, (y - x)) * y$
by (*auto simp*: *field-simps*)
also have $\ldots = ((y - z) * x + (y - x - (y - z)) * y) \,/\, (y - x)$
using *assms* **unfolding** *add-divide-distrib* **by** (*auto simp*: *field-simps*)
also have $\ldots = z$
using *assms* **by** (*auto simp*: *field-simps*)
finally show *?thesis*
using *comb* **by** *auto*
qed
show $z \in C$ **using** *z less assms*
unfolding *atLeastAtMost-iff le-less* **by** *auto*
qed

lemma *f''-imp-f'*:
fixes f :: *real* $\Rightarrow$ *real*
assumes *convex* C
and f': $\bigwedge x.\ x \in C \Longrightarrow DERIV\ f\ x :> (f'\ x)$
and f'': $\bigwedge x.\ x \in C \Longrightarrow DERIV\ f'\ x :> (f''\ x)$
and *pos*: $\bigwedge x.\ x \in C \Longrightarrow f''\ x \geq 0$
and $x \in C$ $y \in C$
shows $f'\ x * (y - x) \leq f\ y - f\ x$
using *assms*
proof −
{
fix x y :: *real*
assume $*$: $x \in C$ $y \in C$ $y > x$
then have *ge*: $y - x > 0$ $y - x \geq 0$
by *auto*
from $*$ **have** *le*: $x - y < 0$ $x - y \leq 0$
by *auto*
then obtain *z1* **where** *z1*: $z1 > x$ $z1 < y$ $f\ y - f\ x = (y - x) * f'\ z1$

using *subsetD*[*OF atMostAtLeast-subset-convex*[*OF* ‹*convex C*› ‹$x \in C$› ‹$y \in C$› ‹$x < y$›],
THEN f', *THEN MVT2*[*OF* ‹$x < y$›, *rule-format*, *unfolded atLeastAtMost-iff*[*symmetric*]]]
by *auto*
then have *z1* $\in C$
using *atMostAtLeast-subset-convex* ‹*convex C*› ‹$x \in C$› ‹$y \in C$› ‹$x < y$›
by *fastforce*
from *z1* **have** *z1'*: $f\ x - f\ y = (x - y) * f'\ z1$
by (*simp add*: *field-simps*)
obtain *z2* **where** *z2*: $z2 > x\ z2 < z1\ f'\ z1 - f'\ x = (z1 - x) * f''\ z2$
using *subsetD*[*OF atMostAtLeast-subset-convex*[*OF* ‹*convex C*› ‹$x \in C$› ‹*z1* $\in C$› ‹$x < z1$›],
THEN f'', *THEN MVT2*[*OF* ‹$x < z1$›, *rule-format*, *unfolded atLeastAtMost-iff*[*symmetric*]]]
z1
by *auto*
obtain *z3* **where** *z3*: $z3 > z1\ z3 < y\ f'\ y - f'\ z1 = (y - z1) * f''\ z3$
using *subsetD*[*OF atMostAtLeast-subset-convex*[*OF* ‹*convex C*› ‹*z1* $\in C$› ‹*y* $\in C$› ‹$z1 < y$›],
THEN f'', *THEN MVT2*[*OF* ‹$z1 < y$›, *rule-format*, *unfolded atLeastAtMost-iff*[*symmetric*]]]
z1
by *auto*
have $f'\ y - (f\ x - f\ y)\ /\ (x - y) = f'\ y - f'\ z1$
using $*$ *z1'* **by** *auto*
also have $\ldots = (y - z1) * f''\ z3$
using *z3* **by** *auto*
finally have *cool'*: $f'\ y - (f\ x - f\ y)\ /\ (x - y) = (y - z1) * f''\ z3$
by *simp*
have *A'*: $y - z1 \geq 0$
using *z1* **by** *auto*
have *z3* $\in C$
using *z3* $*$ *atMostAtLeast-subset-convex* ‹*convex C*› ‹$x \in C$› ‹*z1* $\in C$› ‹$x < z1$›
by *fastforce*
then have *B'*: $f''\ z3 \geq 0$
using *assms* **by** *auto*
from *A' B'* **have** $(y - z1) * f''\ z3 \geq 0$
by *auto*
from *cool' this* **have** $f'\ y - (f\ x - f\ y)\ /\ (x - y) \geq 0$
by *auto*
from *mult-right-mono-neg*[*OF this le*(*2*)]
have $f'\ y * (x - y) - (f\ x - f\ y)\ /\ (x - y) * (x - y) \leq 0 * (x - y)$
by (*simp add*: *algebra-simps*)
then have $f'\ y * (x - y) - (f\ x - f\ y) \leq 0$
using *le* **by** *auto*
then have *res*: $f'\ y * (x - y) \leq f\ x - f\ y$
by *auto*
have $(f\ y - f\ x)\ /\ (y - x) - f'\ x = f'\ z1 - f'\ x$
using $*$ *z1* **by** *auto*
also have $\ldots = (z1 - x) * f''\ z2$

```
    using z2 by auto
  finally have cool: (f y − f x) / (y − x) − f' x = (z1 − x) * f'' z2
    by simp
  have A: z1 − x ≥ 0
    using z1 by auto
  have z2 ∈ C
    using z2 z1 * atMostAtLeast-subset-convex ‹convex C› ‹z1 ∈ C› ‹y ∈ C› ‹z1
< y›
    by fastforce
  then have B: f'' z2 ≥ 0
    using assms by auto
  from A B have (z1 − x) * f'' z2 ≥ 0
    by auto
  with cool have (f y − f x) / (y − x) − f' x ≥ 0
    by auto
  from mult-right-mono[OF this ge(2)]
  have (f y − f x) / (y − x) * (y − x) − f' x * (y − x) ≥ 0 * (y − x)
    by (simp add: algebra-simps)
  then have f y − f x − f' x * (y − x) ≥ 0
    using ge by auto
  then have f y − f x ≥ f' x * (y − x) f' y * (x − y) ≤ f x − f y
    using res by auto
 } note less-imp = this
 {
  fix x y :: real
  assume x ∈ C y ∈ C x ≠ y
  then havef y − f x ≥ f' x * (y − x)
  unfolding neq-iff using less-imp by auto
 }
 moreover
 {
  fix x y :: real
  assume x ∈ C y ∈ C x = y
  then have f y − f x ≥ f' x * (y − x) by auto
 }
 ultimately show ?thesis using assms by blast
qed

lemma f''-ge0-imp-convex:
 fixes f :: real ⇒ real
 assumes conv: convex C
  and f': ⋀x. x ∈ C ⟹ DERIV f x :> (f' x)
  and f'': ⋀x. x ∈ C ⟹ DERIV f' x :> (f'' x)
  and pos: ⋀x. x ∈ C ⟹ f'' x ≥ 0
 shows convex-on C f
 using f''-imp-f'[OF conv f' f'' pos] assms pos-convex-function
 by fastforce

lemma minus-log-convex:
```

fixes $b :: real$
assumes $b > 1$
shows *convex-on* $\{0 <..\}$ $(\lambda\ x.\ -\ log\ b\ x)$
proof −
have $\bigwedge z.\ z > 0 \Longrightarrow DERIV\ (log\ b)\ z :> 1\ /\ (ln\ b * z)$
using *DERIV-log* **by** *auto*
then have f': $\bigwedge z.\ z > 0 \Longrightarrow DERIV\ (\lambda\ z.\ -\ log\ b\ z)\ z :> -\ 1\ /\ (ln\ b * z)$
by (*auto simp*: *DERIV-minus*)
have $\bigwedge z :: real.\ z > 0 \Longrightarrow DERIV\ inverse\ z :> -\ (inverse\ z\ \hat{}\ Suc\ (Suc\ 0))$
using *less-imp-neq*[*THEN not-sym*, *THEN DERIV-inverse*] **by** *auto*
from *this*[*THEN DERIV-cmult*, *of* - − 1 / *ln b*]
have $\bigwedge z :: real.\ z > 0 \Longrightarrow$
DERIV $(\lambda\ z.\ (-\ 1\ /\ ln\ b) * inverse\ z)\ z :> (-\ 1\ /\ ln\ b) * (-\ (inverse\ z\ \hat{}\ Suc$
$(Suc\ 0)))$
by *auto*
then have $f''0$: $\bigwedge z$::*real*. $z > 0 \Longrightarrow$
DERIV $(\lambda\ z.\ -\ 1\ /\ (ln\ b * z))\ z :> 1\ /\ (ln\ b * z * z)$
unfolding *inverse-eq-divide* **by** (*auto simp*: *mult.assoc*)
have f''-*ge0*: $\bigwedge z$::*real*. $z > 0 \Longrightarrow 1\ /\ (ln\ b * z * z) \geq 0$
using ‹$b > 1$› **by** (*auto intro*!: *less-imp-le*)
from f''-*ge0-imp-convex*[*OF pos-is-convex*,
unfolded greaterThan-iff, *OF* f' $f''0$ f''-*ge0*]
show *?thesis* **by** *auto*
qed

17.5 Convexity of real functions

lemma *convex-on-realI*:
assumes *connected A*
assumes $\bigwedge x.\ x \in A \Longrightarrow (f\ has\text{-}real\text{-}derivative\ f'\ x)\ (at\ x)$
assumes $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \leq y \Longrightarrow f'\ x \leq f'\ y$
shows *convex-on A f*
proof (*rule convex-on-linorderI*)
fix $t\ x\ y :: real$
assume t: $t > 0\ t < 1$ **and** xy: $x \in A\ y \in A\ x < y$
def $z \equiv (1 - t) * x + t * y$
with ‹*connected A*› **and** xy **have** ivl: $\{x..y\} \subseteq A$ **using** *connected-contains-Icc*
by *blast*

from $xy\ t$ **have** xz: $z > x$ **by** (*simp add*: *z-def algebra-simps*)
have $y - z = (1 - t) * (y - x)$ **by** (*simp add*: *z-def algebra-simps*)
also from $xy\ t$ **have** *...* > 0 **by** (*intro mult-pos-pos*) *simp-all*
finally have yz: $z < y$ **by** *simp*

from *assms xz yz ivl t* **have** $\exists \xi.\ \xi > x \wedge \xi < z \wedge f\ z - f\ x = (z - x) * f'\ \xi$
by (*intro MVT2*) (*auto intro*!: *assms*(*2*))
then obtain ξ **where** ξ: $\xi > x\ \xi < z\ f'\ \xi = (f\ z - f\ x)\ /\ (z - x)$ **by** *auto*
from *assms xz yz ivl t* **have** $\exists \eta.\ \eta > z \wedge \eta < y \wedge f\ y - f\ z = (y - z) * f'\ \eta$
by (*intro MVT2*) (*auto intro*!: *assms*(*2*))

then obtain η **where** η: $\eta > z$ $\eta < y$ $f'\,\eta = (f\,y - f\,z) \,/\, (y - z)$ **by** *auto*

from $\eta(3)$ **have** $(f\,y - f\,z) \,/\, (y - z) = f'\,\eta$ **..**
also from ξ η *ivl* **have** $\xi \in A$ $\eta \in A$ **by** *auto*
with ξ η **have** $f'\,\eta \geq f'\,\xi$ **by** (*intro assms*(3)) *auto*
also from $\xi(3)$ **have** $f'\,\xi = (f\,z - f\,x) \,/\, (z - x)$ **.**
finally have $(f\,y - f\,z) * (z - x) \geq (f\,z - f\,x) * (y - z)$
using *xz yz* **by** (*simp add*: *field-simps*)
also have $z - x = t * (y - x)$ **by** (*simp add*: *z-def algebra-simps*)
also have $y - z = (1 - t) * (y - x)$ **by** (*simp add*: *z-def algebra-simps*)
finally have $(f\,y - f\,z) * t \geq (f\,z - f\,x) * (1 - t)$ **using** *xy* **by** *simp*
thus $(1 - t) * f\,x + t * f\,y \geq f\,((1 - t) *_R x + t *_R y)$
by (*simp add*: *z-def algebra-simps*)
qed

lemma *convex-on-inverse*:
assumes $A \subseteq \{0<..\}$
shows *convex-on A* (*inverse* :: *real* ⇒ *real*)
proof (*rule convex-on-subset*[*OF - assms*], *intro convex-on-realI*[*of - - λx. −inverse* (*x^2*)])
fix *u v* :: *real* **assume** $u \in \{0<..\}$ $v \in \{0<..\}$ $u \leq v$
with *assms* **show** $-inverse\ (u\hat{}2) \leq -inverse\ (v\hat{}2)$
by (*intro le-imp-neg-le le-imp-inverse-le power-mono*) (*simp-all*)
qed (*insert assms*, *auto intro*!: *derivative-eq-intros simp*: *divide-simps power2-eq-square*)

lemma *convex-onD-Icc'*:
assumes *convex-on* $\{x..y\}$ f $c \in \{x..y\}$
defines $d \equiv y - x$
shows $f\,c \leq (f\,y - f\,x) \,/\, d * (c - x) + f\,x$
proof (*cases y x rule*: *linorder-cases*)
assume *less*: $x < y$
hence *d*: $d > 0$ **by** (*simp add*: *d-def*)
from *assms*(2) *less* **have** *A*: $0 \leq (c - x) \,/\, d$ $(c - x) \,/\, d \leq 1$
by (*simp-all add*: *d-def divide-simps*)
have $f\,c = f\,(x + (c - x) * 1)$ **by** *simp*
also from *less* **have** $1 = ((y - x) \,/\, d)$ **by** (*simp add*: *d-def*)
also from *d* **have** $x + (c - x) * \ldots = (1 - (c - x) \,/\, d) *_R x + ((c - x) \,/\, d) *_R y$
by (*simp add*: *field-simps*)
also have $f\ \ldots \leq (1 - (c - x) \,/\, d) * f\,x + (c - x) \,/\, d * f\,y$ **using** *assms less*
by (*intro convex-onD-Icc*) *simp-all*
also from *d* **have** $\ldots = (f\,y - f\,x) \,/\, d * (c - x) + f\,x$ **by** (*simp add*: *field-simps*)
finally show *?thesis* **.**
qed (*insert assms*(2), *simp-all*)

lemma *convex-onD-Icc''*:
assumes *convex-on* $\{x..y\}$ f $c \in \{x..y\}$
defines $d \equiv y - x$
shows $f\,c \leq (f\,x - f\,y) \,/\, d * (y - c) + f\,y$

proof (*cases y x rule*: *linorder-cases*)
assume *less*: $x < y$
hence *d*: $d > 0$ **by** (*simp add*: *d-def*)
from *assms*(*2*) *less* **have** *A*: $0 \leq (y - c) / d\ (y - c) / d \leq 1$
by (*simp-all add*: *d-def divide-simps*)
have $f\ c = f\ (y - (y - c) * 1)$ **by** *simp*
also from *less* **have** $1 = ((y - x) / d)$ **by** (*simp add*: *d-def*)
also from *d* **have** $y - (y - c) * \ldots = (1 - (1 - (y - c) / d)) *_R x + (1 - (y - c) / d) *_R y$
by (*simp add*: *field-simps*)
also have $f \ldots \leq (1 - (1 - (y - c) / d)) * f\ x + (1 - (y - c) / d) * f\ y$
using *assms less*
by (*intro convex-onD-Icc*) (*simp-all add*: *field-simps*)
also from *d* **have** $\ldots = (f\ x - f\ y) / d * (y - c) + f\ y$ **by** (*simp add*: *field-simps*)
finally show *?thesis* **.**
qed (*insert assms*(*2*), *simp-all*)

end

18 Algebraic operations on sets

theory *Set-Algebras*
imports *Main*
begin

This library lifts operations like addition and multiplication to sets. It was designed to support asymptotic calculations. See the comments at the top of theory *BigO*.

instantiation *set* :: (*plus*) *plus*
begin

definition *plus-set* :: *'a*::*plus set* ⇒ *'a set* ⇒ *'a set* **where**
set-plus-def: $A + B = \{c.\ \exists\, a{\in}A.\ \exists\, b{\in}B.\ c = a + b\}$

instance ..

end

instantiation *set* :: (*times*) *times*
begin

definition *times-set* :: *'a*::*times set* ⇒ *'a set* ⇒ *'a set* **where**
set-times-def: $A * B = \{c.\ \exists\, a{\in}A.\ \exists\, b{\in}B.\ c = a * b\}$

instance ..

end

instantiation *set* :: (*zero*) *zero*
begin

definition
set-zero[*simp*]: (*0*::'*a*::*zero set*) = {*0*}

instance ..

end

instantiation *set* :: (*one*) *one*
begin

definition
set-one[*simp*]: (*1*::'*a*::*one set*) = {*1*}

instance ..

end

definition *elt-set-plus* :: '*a*::*plus* ⇒ '*a set* ⇒ '*a set* (**infixl** +*o* *70*) **where**
a +*o* *B* = {*c*. ∃ *b*∈*B*. *c* = *a* + *b*}

definition *elt-set-times* :: '*a*::*times* ⇒ '*a set* ⇒ '*a set* (**infixl** ∗*o* *80*) **where**
a ∗*o* *B* = {*c*. ∃ *b*∈*B*. *c* = *a* ∗ *b*}

abbreviation (*input*) *elt-set-eq* :: '*a* ⇒ '*a set* ⇒ *bool* (**infix** =*o* *50*) **where**
x =*o* *A* ≡ *x* ∈ *A*

instance *set* :: (*semigroup-add*) *semigroup-add*
by *standard* (*force simp add*: *set-plus-def add.assoc*)

instance *set* :: (*ab-semigroup-add*) *ab-semigroup-add*
by *standard* (*force simp add*: *set-plus-def add.commute*)

instance *set* :: (*monoid-add*) *monoid-add*
by *standard* (*simp-all add*: *set-plus-def*)

instance *set* :: (*comm-monoid-add*) *comm-monoid-add*
by *standard* (*simp-all add*: *set-plus-def*)

instance *set* :: (*semigroup-mult*) *semigroup-mult*
by *standard* (*force simp add*: *set-times-def mult.assoc*)

instance *set* :: (*ab-semigroup-mult*) *ab-semigroup-mult*
by *standard* (*force simp add*: *set-times-def mult.commute*)

instance *set* :: (*monoid-mult*) *monoid-mult*

by *standard* (*simp-all add*: *set-times-def*)

instance *set* :: (*comm-monoid-mult*) *comm-monoid-mult*
by *standard* (*simp-all add*: *set-times-def*)

lemma *set-plus-intro* [*intro*]: $a \in C \Longrightarrow b \in D \Longrightarrow a + b \in C + D$
by (*auto simp add*: *set-plus-def*)

lemma *set-plus-elim*:
assumes $x \in A + B$
obtains *a b* **where** $x = a + b$ **and** $a \in A$ **and** $b \in B$
using *assms* **unfolding** *set-plus-def* **by** *fast*

lemma *set-plus-intro2* [*intro*]: $b \in C \Longrightarrow a + b \in a +o C$
by (*auto simp add*: *elt-set-plus-def*)

lemma *set-plus-rearrange*:
$((a::'a::comm\text{-}monoid\text{-}add) +o C) + (b +o D) = (a + b) +o (C + D)$
apply (*auto simp add*: *elt-set-plus-def set-plus-def ac-simps*)
apply (*rule-tac x* = *ba* + *bb* **in** *exI*)
apply (*auto simp add*: *ac-simps*)
apply (*rule-tac x* = *aa* + *a* **in** *exI*)
apply (*auto simp add*: *ac-simps*)
done

lemma *set-plus-rearrange2*: $(a::'a::semigroup\text{-}add) +o (b +o C) = (a + b) +o C$
by (*auto simp add*: *elt-set-plus-def add.assoc*)

lemma *set-plus-rearrange3*: $((a::'a::semigroup\text{-}add) +o B) + C = a +o (B + C)$
apply (*auto simp add*: *elt-set-plus-def set-plus-def*)
apply (*blast intro*: *ac-simps*)
apply (*rule-tac x* = *a* + *aa* **in** *exI*)
apply (*rule conjI*)
apply (*rule-tac x* = *aa* **in** *bexI*)
apply *auto*
apply (*rule-tac x* = *ba* **in** *bexI*)
apply (*auto simp add*: *ac-simps*)
done

theorem *set-plus-rearrange4*: $C + ((a::'a::comm\text{-}monoid\text{-}add) +o D) = a +o (C + D)$
apply (*auto simp add*: *elt-set-plus-def set-plus-def ac-simps*)
apply (*rule-tac x* = *aa* + *ba* **in** *exI*)
apply (*auto simp add*: *ac-simps*)
done

lemmas *set-plus-rearranges* = *set-plus-rearrange set-plus-rearrange2*
set-plus-rearrange3 set-plus-rearrange4

lemma *set-plus-mono* [*intro*!]: $C \subseteq D \Longrightarrow a +o C \subseteq a +o D$
by (*auto simp add*: *elt-set-plus-def*)

lemma *set-plus-mono2* [*intro*]: $(C::'a::plus\ set) \subseteq D \Longrightarrow E \subseteq F \Longrightarrow C + E \subseteq D + F$
by (*auto simp add*: *set-plus-def*)

lemma *set-plus-mono3* [*intro*]: $a \in C \Longrightarrow a +o D \subseteq C + D$
by (*auto simp add*: *elt-set-plus-def set-plus-def*)

lemma *set-plus-mono4* [*intro*]: $(a::'a::comm\text{-}monoid\text{-}add) \in C \Longrightarrow a +o D \subseteq D + C$
by (*auto simp add*: *elt-set-plus-def set-plus-def ac-simps*)

lemma *set-plus-mono5*: $a \in C \Longrightarrow B \subseteq D \Longrightarrow a +o B \subseteq C + D$
apply (*subgoal-tac* $a +o B \subseteq a +o D$)
apply (*erule order-trans*)
apply (*erule set-plus-mono3*)
apply (*erule set-plus-mono*)
done

lemma *set-plus-mono-b*: $C \subseteq D \Longrightarrow x \in a +o C \Longrightarrow x \in a +o D$
apply (*frule set-plus-mono*)
apply *auto*
done

lemma *set-plus-mono2-b*: $C \subseteq D \Longrightarrow E \subseteq F \Longrightarrow x \in C + E \Longrightarrow x \in D + F$
apply (*frule set-plus-mono2*)
prefer *2*
apply *force*
apply *assumption*
done

lemma *set-plus-mono3-b*: $a \in C \Longrightarrow x \in a +o D \Longrightarrow x \in C + D$
apply (*frule set-plus-mono3*)
apply *auto*
done

lemma *set-plus-mono4-b*: $(a::'a::comm\text{-}monoid\text{-}add) : C \Longrightarrow x \in a +o D \Longrightarrow x \in D + C$
apply (*frule set-plus-mono4*)
apply *auto*
done

lemma *set-zero-plus* [*simp*]: $(0::'a::comm\text{-}monoid\text{-}add) +o C = C$
by (*auto simp add*: *elt-set-plus-def*)

lemma *set-zero-plus2*: $(0::'a::comm\text{-}monoid\text{-}add) \in A \Longrightarrow B \subseteq A + B$
apply (*auto simp add*: *set-plus-def*)

apply (*rule-tac* $x = 0$ **in** *bexI*)
apply (*rule-tac* $x = x$ **in** *bexI*)
apply (*auto simp add*: *ac-simps*)
done

lemma *set-plus-imp-minus*: $(a::'a::ab\text{-}group\text{-}add) : b +o C \Longrightarrow (a - b) \in C$
by (*auto simp add*: *elt-set-plus-def ac-simps*)

lemma *set-minus-imp-plus*: $(a::'a::ab\text{-}group\text{-}add) - b : C \Longrightarrow a \in b +o C$
apply (*auto simp add*: *elt-set-plus-def ac-simps*)
apply (*subgoal-tac* $a = (a + - b) + b$)
apply (*rule bexI*, *assumption*)
apply (*auto simp add*: *ac-simps*)
done

lemma *set-minus-plus*: $(a::'a::ab\text{-}group\text{-}add) - b \in C \longleftrightarrow a \in b +o C$
by (*rule iffI*, *rule set-minus-imp-plus*, *assumption*, *rule set-plus-imp-minus*)

lemma *set-times-intro* [*intro*]: $a \in C \Longrightarrow b \in D \Longrightarrow a * b \in C * D$
by (*auto simp add*: *set-times-def*)

lemma *set-times-elim*:
assumes $x \in A * B$
obtains $a\ b$ **where** $x = a * b$ **and** $a \in A$ **and** $b \in B$
using *assms* **unfolding** *set-times-def* **by** *fast*

lemma *set-times-intro2* [*intro*!]: $b \in C \Longrightarrow a * b \in a *o C$
by (*auto simp add*: *elt-set-times-def*)

lemma *set-times-rearrange*:
$((a::'a::comm\text{-}monoid\text{-}mult) *o C) * (b *o D) = (a * b) *o (C * D)$
apply (*auto simp add*: *elt-set-times-def set-times-def*)
apply (*rule-tac* $x = ba * bb$ **in** *exI*)
apply (*auto simp add*: *ac-simps*)
apply (*rule-tac* $x = aa * a$ **in** *exI*)
apply (*auto simp add*: *ac-simps*)
done

lemma *set-times-rearrange2*:
$(a::'a::semigroup\text{-}mult) *o (b *o C) = (a * b) *o C$
by (*auto simp add*: *elt-set-times-def mult.assoc*)

lemma *set-times-rearrange3*:
$((a::'a::semigroup\text{-}mult) *o B) * C = a *o (B * C)$
apply (*auto simp add*: *elt-set-times-def set-times-def*)
apply (*blast intro*: *ac-simps*)
apply (*rule-tac* $x = a * aa$ **in** *exI*)
apply (*rule conjI*)
apply (*rule-tac* $x = aa$ **in** *bexI*)

```
   apply auto
  apply (rule-tac x = ba in bexI)
   apply (auto simp add: ac-simps)
  done

theorem set-times-rearrange4:
  C * ((a::'a::comm-monoid-mult) *o D) = a *o (C * D)
  apply (auto simp add: elt-set-times-def set-times-def ac-simps)
   apply (rule-tac x = aa * ba in exI)
   apply (auto simp add: ac-simps)
  done

lemmas set-times-rearranges = set-times-rearrange set-times-rearrange2
  set-times-rearrange3 set-times-rearrange4

lemma set-times-mono [intro]: C ⊆ D ⟹ a *o C ⊆ a *o D
  by (auto simp add: elt-set-times-def)

lemma set-times-mono2 [intro]: (C::'a::times set) ⊆ D ⟹ E ⊆ F ⟹ C * E ⊆
D * F
  by (auto simp add: set-times-def)

lemma set-times-mono3 [intro]: a ∈ C ⟹ a *o D ⊆ C * D
  by (auto simp add: elt-set-times-def set-times-def)

lemma set-times-mono4 [intro]: (a::'a::comm-monoid-mult) : C ⟹ a *o D ⊆ D
* C
  by (auto simp add: elt-set-times-def set-times-def ac-simps)

lemma set-times-mono5: a ∈ C ⟹ B ⊆ D ⟹ a *o B ⊆ C * D
  apply (subgoal-tac a *o B ⊆ a *o D)
   apply (erule order-trans)
   apply (erule set-times-mono3)
  apply (erule set-times-mono)
  done

lemma set-times-mono-b: C ⊆ D ⟹ x ∈ a *o C ⟹ x ∈ a *o D
  apply (frule set-times-mono)
  apply auto
  done

lemma set-times-mono2-b: C ⊆ D ⟹ E ⊆ F ⟹ x ∈ C * E ⟹ x ∈ D * F
  apply (frule set-times-mono2)
   prefer 2
   apply force
  apply assumption
  done

lemma set-times-mono3-b: a ∈ C ⟹ x ∈ a *o D ⟹ x ∈ C * D
```

apply (*frule set-times-mono3*)
apply *auto*
done

lemma *set-times-mono4-b*: (*a*::'*a*::*comm-monoid-mult*) ∈ *C* ⟹ *x* ∈ *a* ∗*o* *D* ⟹ *x* ∈ *D* ∗ *C*
apply (*frule set-times-mono4*)
apply *auto*
done

lemma *set-one-times* [*simp*]: (*1*::'*a*::*comm-monoid-mult*) ∗*o* *C* = *C*
by (*auto simp add*: *elt-set-times-def*)

lemma *set-times-plus-distrib*:
(*a*::'*a*::*semiring*) ∗*o* (*b* +*o* *C*) = (*a* ∗ *b*) +*o* (*a* ∗*o* *C*)
by (*auto simp add*: *elt-set-plus-def elt-set-times-def ring-distribs*)

lemma *set-times-plus-distrib2*:
(*a*::'*a*::*semiring*) ∗*o* (*B* + *C*) = (*a* ∗*o* *B*) + (*a* ∗*o* *C*)
apply (*auto simp add*: *set-plus-def elt-set-times-def ring-distribs*)
apply *blast*
apply (*rule-tac x* = *b* + *bb* **in** *exI*)
apply (*auto simp add*: *ring-distribs*)
done

lemma *set-times-plus-distrib3*: ((*a*::'*a*::*semiring*) +*o* *C*) ∗ *D* ⊆ *a* ∗*o* *D* + *C* ∗ *D*
apply (*auto simp add*:
elt-set-plus-def elt-set-times-def set-times-def
set-plus-def ring-distribs)
apply *auto*
done

lemmas *set-times-plus-distribs* =
set-times-plus-distrib
set-times-plus-distrib2

lemma *set-neg-intro*: (*a*::'*a*::*ring-1*) ∈ (− *1*) ∗*o* *C* ⟹ − *a* ∈ *C*
by (*auto simp add*: *elt-set-times-def*)

lemma *set-neg-intro2*: (*a*::'*a*::*ring-1*) ∈ *C* ⟹ − *a* ∈ (− *1*) ∗*o* *C*
by (*auto simp add*: *elt-set-times-def*)

lemma *set-plus-image*: *S* + *T* = (λ(*x*, *y*). *x* + *y*) ‘ (*S* × *T*)
unfolding *set-plus-def* **by** (*fastforce simp*: *image-iff*)

lemma *set-times-image*: *S* ∗ *T* = (λ(*x*, *y*). *x* ∗ *y*) ‘ (*S* × *T*)
unfolding *set-times-def* **by** (*fastforce simp*: *image-iff*)

lemma *finite-set-plus*: *finite s* ⟹ *finite t* ⟹ *finite* (*s* + *t*)

unfolding *set-plus-image* **by** *simp*

lemma *finite-set-times*: *finite s* $\Longrightarrow$ *finite t* $\Longrightarrow$ *finite* $(s * t)$
unfolding *set-times-image* **by** *simp*

lemma *set-setsum-alt*:
assumes *fin*: *finite I*
shows *setsum S I* $= \{$*setsum s I* $|s. \forall i \in I.\ s\ i \in S\ i\}$
(**is** - = *?setsum I*)
using *fin*
proof *induct*
case *empty*
then show *?case* **by** *simp*
next
case (*insert x F*)
have *setsum S* (*insert x F*) = *S x* + *?setsum F*
using *insert.hyps* **by** *auto*
also have ... $= \{s\ x +$ *setsum s F* $|s. \forall\ i \in$ *insert x F*. *s i* $\in S\ i\}$
unfolding *set-plus-def*
proof *safe*
fix *y s*
assume $y \in S\ x\ \forall i \in F.\ s\ i \in S\ i$
then show $\exists s'.\ y +$ *setsum s F* $= s'\ x +$ *setsum s' F* $\wedge\ (\forall i \in$ *insert x F*. *s' i* $\in S\ i)$
using *insert.hyps*
by (*intro exI*[*of* - $\lambda i.$ *if i* $\in$ *F then s i else y*]) (*auto simp add*: *set-plus-def*)
qed *auto*
finally show *?case*
using *insert.hyps* **by** *auto*
qed

lemma *setsum-set-cond-linear*:
fixes *f* :: *'a*::*comm-monoid-add set* $\Rightarrow$ *'b*::*comm-monoid-add set*
assumes [*intro*!]: $\bigwedge A\ B.\ P\ A \Longrightarrow P\ B \Longrightarrow P\ (A + B)\ P\ \{0\}$
and *f*: $\bigwedge A\ B.\ P\ A \Longrightarrow P\ B \Longrightarrow f\ (A + B) = f\ A + f\ B\ f\ \{0\} = \{0\}$
assumes *all*: $\bigwedge i.\ i \in I \Longrightarrow P\ (S\ i)$
shows *f* (*setsum S I*) = *setsum* (*f* $\circ$ *S*) *I*
proof (*cases finite I*)
case *True*
from *this all* **show** *?thesis*
proof *induct*
case *empty*
then show *?case* **by** (*auto intro*!: *f*)
next
case (*insert x F*)
from ‹*finite F*› ‹$\bigwedge i.\ i \in$ *insert x F* $\Longrightarrow P\ (S\ i)$› **have** *P* (*setsum S F*)
by *induct auto*
with *insert* **show** *?case*
by (*simp*, *subst f*) *auto*

```
  qed
next
  case False
  then show ?thesis by (auto intro!: f)
qed

lemma setsum-set-linear:
  fixes f :: 'a::comm-monoid-add set ⇒ 'b::comm-monoid-add set
  assumes ⋀A B. f(A) + f(B) = f(A + B) f {0} = {0}
  shows f (setsum S I) = setsum (f ∘ S) I
  using setsum-set-cond-linear[of λx. True f I S] assms by auto

lemma set-times-Un-distrib:
  A * (B ∪ C) = A * B ∪ A * C
  (A ∪ B) * C = A * C ∪ B * C
  by (auto simp: set-times-def)

lemma set-times-UNION-distrib:
  A * UNION I M = (⋃i∈I. A * M i)
  UNION I M * A = (⋃i∈I. M i * A)
  by (auto simp: set-times-def)

end
```

19 Convex sets, functions and related things.

```
theory Convex-Euclidean-Space
imports
  Topology-Euclidean-Space
  ~~/src/HOL/Library/Convex
  ~~/src/HOL/Library/Set-Algebras
begin

lemma independent-injective-on-span-image:
  assumes iS: independent S
    and lf: linear f
    and fi: inj-on f (span S)
  shows independent (f ' S)
proof −
  {
    fix a
    assume a: a ∈ S f a ∈ span (f ' S − {f a})
    have eq: f ' S − {f a} = f ' (S − {a})
      using fi a span-inc by (auto simp add: inj-on-def)
    from a have f a ∈ f ' span (S −{a})
      unfolding eq span-linear-image [OF lf, of S − {a}] by blast
    moreover have span (S − {a}) ⊆ span S
      using span-mono[of S − {a} S] by auto
    ultimately have a ∈ span (S − {a})
```

```
    using fi a span-inc by (auto simp add: inj-on-def)
  with a(1) iS have False
    by (simp add: dependent-def)
 }
 then show ?thesis
  unfolding dependent-def by blast
qed

lemma dim-image-eq:
 fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
 assumes lf: linear f
  and fi: inj-on f (span S)
 shows dim (f ‘ S) = dim (S::'n::euclidean-space set)
proof −
 obtain B where B: B ⊆ S independent B S ⊆ span B card B = dim S
  using basis-exists[of S] by auto
 then have span S = span B
  using span-mono[of B S] span-mono[of S span B] span-span[of B] by auto
 then have independent (f ‘ B)
  using independent-injective-on-span-image[of B f] B assms by auto
 moreover have card (f ‘ B) = card B
  using assms card-image[of f B] subset-inj-on[of f span S B] B span-inc by auto
 moreover have (f ‘ B) ⊆ (f ‘ S)
  using B by auto
 ultimately have dim (f ‘ S) ≥ dim S
  using independent-card-le-dim[of f ‘ B f ‘ S] B by auto
 then show ?thesis
  using dim-image-le[of f S] assms by auto
qed

lemma linear-injective-on-subspace-0:
 assumes lf: linear f
  and subspace S
 shows inj-on f S ⟷ (∀ x ∈ S. f x = 0 ⟶ x = 0)
proof −
 have inj-on f S ⟷ (∀ x ∈ S. ∀ y ∈ S. f x = f y ⟶ x = y)
  by (simp add: inj-on-def)
 also have ... ⟷ (∀ x ∈ S. ∀ y ∈ S. f x − f y = 0 ⟶ x − y = 0)
  by simp
 also have ... ⟷ (∀ x ∈ S. ∀ y ∈ S. f (x − y) = 0 ⟶ x − y = 0)
  by (simp add: linear-sub[OF lf])
 also have ... ⟷ (∀ x ∈ S. f x = 0 ⟶ x = 0)
  using ‹subspace S› subspace-def[of S] subspace-sub[of S] by auto
 finally show ?thesis .
qed

lemma subspace-Inter: ∀ s ∈ f. subspace s ⟹ subspace (⋂f)
 unfolding subspace-def by auto
```

lemma *span-eq*[*simp*]: *span s = s ⟷ subspace s*
unfolding *span-def* **by** (*rule hull-eq*) (*rule subspace-Inter*)

lemma *substdbasis-expansion-unique*:
assumes *d*: *d ⊆ Basis*
shows (∑ *i*∈*d*. *f i* ∗*R i*) = (*x*::′*a*::*euclidean-space*) ⟷
(∀ *i*∈*Basis*. (*i* ∈ *d* ⟶ *f i* = *x* ∙ *i*) ∧ (*i* ∉ *d* ⟶ *x* ∙ *i* = *0*))
proof −
have ∗: ⋀*x a b P*. *x* ∗ (*if P then a else b*) = (*if P then x* ∗ *a else x* ∗ *b*)
by *auto*
have ∗∗: *finite d*
by (*auto intro*: *finite-subset*[*OF assms*])
have ∗∗∗: ⋀*i*. *i* ∈ *Basis* ⟹ (∑ *i*∈*d*. *f i* ∗*R i*) ∙ *i* = (∑ *x*∈*d*. *if x* = *i then f x else 0*)
using *d*
by (*auto intro*!: *setsum.cong simp*: *inner-Basis inner-setsum-left*)
show *?thesis*
unfolding *euclidean-eq-iff*[**where** ′*a*=′*a*] **by** (*auto simp*: *setsum.delta*[*OF* ∗∗] ∗∗∗)
qed

lemma *independent-substdbasis*: *d ⊆ Basis* ⟹ *independent d*
by (*rule independent-mono*[*OF independent-Basis*])

lemma *dim-cball*:
assumes *e* > *0*
shows *dim* (*cball* (*0* :: ′*n*::*euclidean-space*) *e*) = *DIM*(′*n*)
proof −
{
fix *x* :: ′*n*::*euclidean-space*
def *y* ≡ (*e* / *norm x*) ∗*R x*
then have *y* ∈ *cball 0 e*
using *cball-def dist-norm*[*of 0 y*] *assms* **by** *auto*
moreover have ∗: *x* = (*norm x* / *e*) ∗*R y*
using *y-def assms* **by** *simp*
moreover from ∗ **have** *x* = (*norm x*/*e*) ∗*R y*
by *auto*
ultimately have *x* ∈ *span* (*cball 0 e*)
using *span-mul*[*of y cball 0 e norm x*/*e*] *span-inc*[*of cball 0 e*] **by** *auto*
}
then have *span* (*cball 0 e*) = (*UNIV* :: ′*n*::*euclidean-space set*)
by *auto*
then show *?thesis*
using *dim-span*[*of cball* (*0* :: ′*n*::*euclidean-space*) *e*] **by** (*auto simp add*: *dim-UNIV*)
qed

lemma *indep-card-eq-dim-span*:
fixes *B* :: ′*n*::*euclidean-space set*
assumes *independent B*

shows *finite B ∧ card B = dim (span B)*
using *assms basis-card-eq-dim*[*of B span B*] *span-inc* **by** *auto*

lemma *setsum-not-0*: *setsum f A ≠ 0 ⟹ ∃ a ∈ A. f a ≠ 0*
by (*rule ccontr*) *auto*

lemma *subset-translation-eq* [*simp*]:
fixes *a* :: *'a*::*real-vector* **shows** *op + a ` s ⊆ op + a ` t ⟷ s ⊆ t*
by *auto*

lemma *translate-inj-on*:
fixes *A* :: *'a*::*ab-group-add set*
shows *inj-on* (*λx. a + x*) *A*
unfolding *inj-on-def* **by** *auto*

lemma *translation-assoc*:
fixes *a b* :: *'a*::*ab-group-add*
shows (*λx. b + x*) ` ((*λx. a + x*) ` *S*) = (*λx.* (*a + b*) *+ x*) ` *S*
by *auto*

lemma *translation-invert*:
fixes *a* :: *'a*::*ab-group-add*
assumes (*λx. a + x*) ` *A* = (*λx. a + x*) ` *B*
shows *A* = *B*
proof −
have (*λx. −a + x*) ` ((*λx. a + x*) ` *A*) = (*λx. − a + x*) ` ((*λx. a + x*) ` *B*)
using *assms* **by** *auto*
then show *?thesis*
using *translation-assoc*[*of −a a A*] *translation-assoc*[*of −a a B*] **by** *auto*
qed

lemma *translation-galois*:
fixes *a* :: *'a*::*ab-group-add*
shows *T* = ((*λx. a + x*) ` *S*) ⟷ *S* = ((*λx.* (*− a*) *+ x*) ` *T*)
using *translation-assoc*[*of −a a S*]
apply *auto*
using *translation-assoc*[*of a −a T*]
apply *auto*
done

lemma *convex-translation-eq* [*simp*]: *convex* ((*λx. a + x*) ` *s*) ⟷ *convex s*
by (*metis convex-translation translation-galois*)

lemma *translation-inverse-subset*:
assumes ((*λx. − a + x*) ` *V*) ≤ (*S* :: *'n*::*ab-group-add set*)
shows *V* ≤ ((*λx. a + x*) ` *S*)
proof −
{
fix *x*

```
    assume x ∈ V
    then have x−a ∈ S using assms by auto
    then have x ∈ {a + v |v. v ∈ S}
      apply auto
      apply (rule exI[of - x−a])
      apply simp
      done
    then have x ∈ ((λx. a+x) ‘ S) by auto
  }
  then show ?thesis by auto
qed

lemma convex-linear-image-eq [simp]:
    fixes f :: 'a::real-vector ⇒ 'b::real-vector
    shows ⟦linear f; inj f⟧ ⟹ convex (f ‘ s) ⟷ convex s
   by (metis (no-types) convex-linear-image convex-linear-vimage inj-vimage-image-eq)

lemma basis-to-basis-subspace-isomorphism:
  assumes s: subspace (S:: ('n::euclidean-space) set)
    and t: subspace (T :: ('m::euclidean-space) set)
    and d: dim S = dim T
    and B: B ⊆ S independent B S ⊆ span B card B = dim S
    and C: C ⊆ T independent C T ⊆ span C card C = dim T
  shows ∃f. linear f ∧ f ‘ B = C ∧ f ‘ S = T ∧ inj-on f S
proof −
  from B independent-bound have fB: finite B
    by blast
  from C independent-bound have fC: finite C
    by blast
  from B(4) C(4) card-le-inj[of B C] d obtain f where
    f: f ‘ B ⊆ C inj-on f B using ‹finite B› ‹finite C› by auto
  from linear-independent-extend[OF B(2)] obtain g where
    g: linear g ∀x ∈ B. g x = f x by blast
  from inj-on-iff-eq-card[OF fB, of f] f(2)
  have card (f ‘ B) = card B by simp
  with B(4) C(4) have ceq: card (f ‘ B) = card C using d
    by simp
  have g ‘ B = f ‘ B using g(2)
    by (auto simp add: image-iff)
  also have ... = C using card-subset-eq[OF fC f(1) ceq] .
  finally have gBC: g ‘ B = C .
  have gi: inj-on g B using f(2) g(2)
    by (auto simp add: inj-on-def)
  note g0 = linear-indep-image-lemma[OF g(1) fB, unfolded gBC, OF C(2) gi]
  {
    fix x y
    assume x: x ∈ S and y: y ∈ S and gxy: g x = g y
    from B(3) x y have x': x ∈ span B and y': y ∈ span B
      by blast+
```

```
    from gxy have th0: g (x − y) = 0
      by (simp add: linear-sub[OF g(1)])
    have th1: x − y ∈ span B using x′ y′
      by (metis span-sub)
    have x = y using g0[OF th1 th0] by simp
  }
  then have giS: inj-on g S unfolding inj-on-def by blast
  from span-subspace[OF B(1,3) s]
  have g ‘ S = span (g ‘ B)
    by (simp add: span-linear-image[OF g(1)])
  also have ... = span C
    unfolding gBC ..
  also have ... = T
    using span-subspace[OF C(1,3) t] .
  finally have gS: g ‘ S = T .
  from g(1) gS giS gBC show ?thesis
    by blast
qed

lemma closure-bounded-linear-image-subset:
  assumes f: bounded-linear f
  shows f ‘ closure S ⊆ closure (f ‘ S)
  using linear-continuous-on [OF f] closed-closure closure-subset
  by (rule image-closure-subset)

lemma closure-linear-image-subset:
  fixes f :: 'm::euclidean-space ⇒ 'n::real-normed-vector
  assumes linear f
  shows f ‘ (closure S) ⊆ closure (f ‘ S)
  using assms unfolding linear-conv-bounded-linear
  by (rule closure-bounded-linear-image-subset)

lemma closed-injective-linear-image:
    fixes f :: 'a::euclidean-space ⇒ 'b::euclidean-space
    assumes S: closed S and f: linear f inj f
    shows closed (f ‘ S)
proof −
  obtain g where g: linear g g ∘ f = id
    using linear-injective-left-inverse [OF f] by blast
  then have confg: continuous-on (range f) g
    using linear-continuous-on linear-conv-bounded-linear by blast
  have [simp]: g ‘ f ‘ S = S
    using g by (simp add: image-comp)
  have cgf: closed (g ‘ f ‘ S)
    by (simp add: ‹g ∘ f = id› S image-comp)
  have [simp]: {x ∈ range f. g x ∈ S} = f ‘ S
    using g by (simp add: o-def id-def image-def) metis
  show ?thesis
    apply (rule closedin-closed-trans [of range f])
```

apply (*rule continuous-closedin-preimage* [*OF confg cgf*, *simplified*])
apply (*rule closed-injective-image-subspace*)
using *f*
apply (*auto simp*: *linear-linear linear-injective-0*)
done
qed

lemma *closed-injective-linear-image-eq*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *f*: *linear f inj f*
shows (*closed*(*image f s*) ⟷ *closed s*)
by (*metis closed-injective-linear-image closure-eq closure-linear-image-subset closure-subset-eq f*(*1*) *f*(*2*) *inj-image-subset-iff*)

lemma *closure-injective-linear-image*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
shows ⟦*linear f*; *inj f*⟧ ⟹ *f* ‘ (*closure S*) = *closure* (*f* ‘ *S*)
apply (*rule subset-antisym*)
apply (*simp add*: *closure-linear-image-subset*)
by (*simp add*: *closure-minimal closed-injective-linear-image closure-subset image-mono*)

lemma *closure-bounded-linear-image*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
shows ⟦*linear f*; *bounded S*⟧ ⟹ *f* ‘ (*closure S*) = *closure* (*f* ‘ *S*)
apply (*rule subset-antisym*, *simp add*: *closure-linear-image-subset*)
apply (*rule closure-minimal*, *simp add*: *closure-subset image-mono*)
by (*meson bounded-closure closed-closure compact-continuous-image compact-eq-bounded-closed linear-continuous-on linear-conv-bounded-linear*)

lemma *closure-scaleR*:
fixes *S* :: *'a*::*real-normed-vector set*
shows (*op* ∗*R c*) ‘ (*closure S*) = *closure* ((*op* ∗*R c*) ‘ *S*)
proof
show (*op* ∗*R c*) ‘ (*closure S*) ⊆ *closure* ((*op* ∗*R c*) ‘ *S*)
using *bounded-linear-scaleR-right*
by (*rule closure-bounded-linear-image-subset*)
show *closure* ((*op* ∗*R c*) ‘ *S*) ⊆ (*op* ∗*R c*) ‘ (*closure S*)
by (*intro closure-minimal image-mono closure-subset closed-scaling closed-closure*)
qed

lemma *fst-linear*: *linear fst*
unfolding *linear-iff* **by** (*simp add*: *algebra-simps*)

lemma *snd-linear*: *linear snd*
unfolding *linear-iff* **by** (*simp add*: *algebra-simps*)

lemma *fst-snd-linear*: *linear* (λ(*x*,*y*). *x* + *y*)
unfolding *linear-iff* **by** (*simp add*: *algebra-simps*)

lemma *scaleR-2*:
fixes x :: *'a::real-vector*
shows *scaleR 2 x* = $x + x$
unfolding *one-add-one* [*symmetric*] *scaleR-left-distrib* **by** *simp*

lemma *vector-choose-size*:
$0 \leq c \Longrightarrow \exists x::'a::$*euclidean-space*. *norm* $x = c$
apply (*rule exI* [**where** x=$c *_R$ (*SOME i*. $i \in$ *Basis*)])
apply (*auto simp*: *SOME-Basis*)
done

lemma *setsum-delta-notmem*:
assumes $x \notin s$
shows *setsum* (λy. *if* ($y = x$) *then P x else Q y*) s = *setsum Q s*
and *setsum* (λy. *if* ($x = y$) *then P x else Q y*) s = *setsum Q s*
and *setsum* (λy. *if* ($y = x$) *then P y else Q y*) s = *setsum Q s*
and *setsum* (λy. *if* ($x = y$) *then P y else Q y*) s = *setsum Q s*
apply (*rule-tac* [!] *setsum.cong*)
using *assms*
apply *auto*
done

lemma *setsum-delta''*:
fixes *s::'a::real-vector set*
assumes *finite s*
shows ($\sum x \in s$. (*if* $y = x$ *then f x else* 0) $*_R$ x) = (*if* $y \in s$ *then* (*f y*) $*_R$ *y else* 0)
proof −
have $*$: $\bigwedge x\ y$. (*if* $y = x$ *then f x else* (0::*real*)) $*_R$ x = (*if* x=y *then* (*f x*) $*_R$ x *else* 0)
by *auto*
show *?thesis*
unfolding $*$ **using** *setsum.delta*[*OF assms*, *of y* λx. *f x* $*_R$ x] **by** *auto*
qed

lemma *if-smult*: (*if P then x else* (*y*::*real*)) $*_R$ v = (*if P then x* $*_R$ v *else y* $*_R$ v)
by (*fact if-distrib*)

lemma *dist-triangle-eq*:
fixes $x\ y\ z$ :: *'a::real-inner*
shows *dist x z* = *dist x y* + *dist y z* $\longleftrightarrow$
norm ($x - y$) $*_R$ ($y - z$) = *norm* ($y - z$) $*_R$ ($x - y$)
proof −
have $*$: $x - y + (y - z) = x - z$ **by** *auto*
show *?thesis* **unfolding** *dist-norm norm-triangle-eq*[*of* $x - y$ $y - z$, *unfolded* $*$]
by (*auto simp add*:*norm-minus-commute*)
qed

lemma *norm-minus-eqI*: $x = - y \Longrightarrow$ *norm x* = *norm y* **by** *auto*

lemma *Min-grI*:
assumes *finite A* $A \neq \{\}$ $\forall a{\in}A.\ x < a$
shows $x < Min\ A$
unfolding *Min-gr-iff*[*OF assms*(*1*,*2*)] **using** *assms*(*3*) **by** *auto*

lemma *norm-lt*: $norm\ x < norm\ y \longleftrightarrow inner\ x\ x < inner\ y\ y$
unfolding *norm-eq-sqrt-inner* **by** *simp*

lemma *norm-le*: $norm\ x \leq norm\ y \longleftrightarrow inner\ x\ x \leq inner\ y\ y$
unfolding *norm-eq-sqrt-inner* **by** *simp*

19.1 Affine set and affine hull

definition *affine* :: *'a::real-vector set* ⇒ *bool*
where $affine\ s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$

lemma *affine-alt*: $affine\ s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u{::}real.\ (1 - u) *_R x + u *_R y \in s)$
unfolding *affine-def* **by** (*metis eq-diff-eq'*)

lemma *affine-empty*[*intro*]: *affine* {}
unfolding *affine-def* **by** *auto*

lemma *affine-sing*[*intro*]: *affine* {*x*}
unfolding *affine-alt* **by** (*auto simp add*: *scaleR-left-distrib* [*symmetric*])

lemma *affine-UNIV*[*intro*]: *affine UNIV*
unfolding *affine-def* **by** *auto*

lemma *affine-Inter*[*intro*]: $(\forall s{\in}f.\ affine\ s) \Longrightarrow affine\ (\bigcap f)$
unfolding *affine-def* **by** *auto*

lemma *affine-Int*[*intro*]: $affine\ s \Longrightarrow affine\ t \Longrightarrow affine\ (s \cap t)$
unfolding *affine-def* **by** *auto*

lemma *affine-affine-hull* [*simp*]: *affine*(*affine hull s*)
unfolding *hull-def*
using *affine-Inter*[*of* {*t*. *affine t* ∧ *s* ⊆ *t*}] **by** *auto*

lemma *affine-hull-eq*[*simp*]: $(affine\ hull\ s = s) \longleftrightarrow affine\ s$
by (*metis affine-affine-hull hull-same*)

19.1.1 Some explicit formulations (from Lars Schewe)

lemma *affine*:
fixes *V*::*'a*::*real-vector set*
shows *affine V* ⟷
(∀ *s u*. *finite s* ∧ *s* ≠ {} ∧ *s* ⊆ *V* ∧ *setsum u s* = *1* ⟶ (*setsum* (λ*x*. (*u x*) *$*_R$ *x*)) *s* ∈ *V*)
unfolding *affine-def*

apply *rule*
apply(*rule*, *rule*, *rule*)
apply(*erule conjE*)+
defer
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix *x y u v*
assume *as*: $x \in V$ $y \in V$ $u + v = (1::real)$
$\forall s\ u.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq V \wedge$ *setsum* $u\ s = 1 \longrightarrow (\sum x{\in}s.\ u\ x *_R x) \in V$
then show $u *_R x + v *_R y \in V$
apply (*cases* $x = y$)
using *as*(*4*)[*THEN spec*[**where** x={x,y}], *THEN spec*[**where** x=$\lambda w.$ *if* $w = x$ *then u else v*]]
and *as*(*1*−*3*)
apply (*auto simp add*: *scaleR-left-distrib*[*symmetric*])
done
next
fix *s u*
assume *as*: $\forall x{\in}V.\ \forall y{\in}V.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in V$
finite s $s \neq \{\}$ $s \subseteq V$ *setsum u s* = (*1*::*real*)
def $n \equiv$ *card s*
have *card s* = *0* ∨ *card s* = *1* ∨ *card s* = *2* ∨ *card s* > *2* **by** *auto*
then show $(\sum x{\in}s.\ u\ x *_R x) \in V$
proof (*auto simp only*: *disjE*)
assume *card s* = *2*
then have *card s* = *Suc* (*Suc 0*)
by *auto*
then obtain *a b* **where** $s = \{a, b\}$
unfolding *card-Suc-eq* **by** *auto*
then show *?thesis*
using *as*(*1*)[*THEN bspec*[**where** x=a], *THEN bspec*[**where** x=b]] **using** *as*(*4*,*5*)
by (*auto simp add*: *setsum-clauses*(*2*))
next
assume *card s* > *2*
then show *?thesis* **using** *as* **and** *n-def*
proof (*induct n arbitrary*: *u s*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
fix *s* :: *'a set* **and** *u* :: *'a* ⇒ *real*
assume *IA*:
$\bigwedge u\ s.\ [\![2 < card\ s; \forall x{\in}V.\ \forall y{\in}V.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in V;$ *finite s*;
$s \neq \{\}; s \subseteq V;$ *setsum u s* = *1*; $n =$ *card s* $]\!] \Longrightarrow (\sum x{\in}s.\ u\ x *_R x) \in V$
and *as*:
Suc n = *card s* *2* < *card s* $\forall x{\in}V.\ \forall y{\in}V.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in V$

```
        finite s s ≠ {} s ⊆ V setsum u s = 1
    have ∃x∈s. u x ≠ 1
    proof (rule ccontr)
      assume ¬ ?thesis
      then have setsum u s = real-of-nat (card s)
        unfolding card-eq-setsum by auto
      then show False
        using as(7) and ‹card s > 2›
            by (metis One-nat-def less-Suc0 Zero-not-Suc of-nat-1 of-nat-eq-iff
numeral-2-eq-2)
    qed
    then obtain x where x:x ∈ s u x ≠ 1 by auto

    have c: card (s − {x}) = card s − 1
      apply (rule card-Diff-singleton)
      using ‹x∈s› as(4)
      apply auto
      done
    have ∗: s = insert x (s − {x}) finite (s − {x})
      using ‹x∈s› and as(4) by auto
    have ∗∗: setsum u (s − {x}) = 1 − u x
      using setsum-clauses(2)[OF ∗(2), of u x, unfolded ∗(1)[symmetric] as(7)]
by auto
    have ∗∗∗: inverse (1 − u x) ∗ setsum u (s − {x}) = 1
      unfolding ∗∗ using ‹u x ≠ 1› by auto
    have (∑xa∈s − {x}. (inverse (1 − u x) ∗ u xa) ∗R xa) ∈ V
    proof (cases card (s − {x}) > 2)
      case True
      then have s − {x} ≠ {} card (s − {x}) = n
        unfolding c and as(1)[symmetric]
      proof (rule-tac ccontr)
        assume ¬ s − {x} ≠ {}
        then have card (s − {x}) = 0 unfolding card-0-eq[OF ∗(2)] by simp
        then show False using True by auto
      qed auto
      then show ?thesis
        apply (rule-tac IA[of s − {x} λy. (inverse (1 − u x) ∗ u y)])
        unfolding setsum-right-distrib[symmetric]
        using as and ∗∗∗ and True
        apply auto
        done
    next
      case False
      then have card (s − {x}) = Suc (Suc 0)
        using as(2) and c by auto
      then obtain a b where (s − {x}) = {a, b} a≠b
        unfolding card-Suc-eq by auto
      then show ?thesis
        using as(3)[THEN bspec[where x=a], THEN bspec[where x=b]]
```

```
        using *** *(2) and ‹s ⊆ V›
        unfolding setsum-right-distrib
        by (auto simp add: setsum-clauses(2))
      qed
      then have u x + (1 - u x) = 1 ⟹
        u x *R x + (1 - u x) *R ((∑ xa∈s - {x}. u xa *R xa) /R (1 - u x)) ∈
V
        apply -
        apply (rule as(3)[rule-format])
        unfolding  Real-Vector-Spaces.scaleR-right.setsum
        using x(1) as(6)
        apply auto
        done
      then show (∑ x∈s. u x *R x) ∈ V
        unfolding scaleR-scaleR[symmetric] and scaleR-right.setsum [symmetric]
        apply (subst *)
        unfolding setsum-clauses(2)[OF *(2)]
        using ‹u x ≠ 1›
        apply auto
        done
    qed
  next
    assume card s = 1
    then obtain a where s={a}
      by (auto simp add: card-Suc-eq)
    then show ?thesis
      using as(4,5) by simp
  qed (insert ‹s≠{}› ‹finite s›, auto)
qed

lemma affine-hull-explicit:
  affine hull p =
    {y. ∃ s u. finite s ∧ s ≠ {} ∧ s ⊆ p ∧ setsum u s = 1 ∧ setsum (λv. (u v) *R
v) s = y}
  apply (rule hull-unique)
  apply (subst subset-eq)
  prefer 3
  apply rule
  unfolding mem-Collect-eq
  apply (erule exE)+
  apply (erule conjE)+
  prefer 2
  apply rule
proof -
  fix x
  assume x∈p
  then show ∃ s u. finite s ∧ s ≠ {} ∧ s ⊆ p ∧ setsum u s = 1 ∧ (∑ v∈s. u v
*R v) = x
    apply (rule-tac x={x} in exI)
```

apply (*rule-tac x*=$\lambda x.\ 1$ **in** *exI*)
apply *auto*
done
next
fix *t x s u*
assume *as*: $p \subseteq t$ *affine t finite s* $s \neq \{\}$
$s \subseteq p$ *setsum u s* $= 1$ $(\sum v{\in}s.\ u\ v *_R v) = x$
then show $x \in t$
using *as*(*2*)[*unfolded affine*, *THEN spec*[**where** *x*=*s*], *THEN spec*[**where** *x*=*u*]]
by *auto*
next
show *affine* $\{y.\ \exists s\ u.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq p \wedge$ *setsum u s* $= 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = y\}$
unfolding *affine-def*
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
unfolding *mem-Collect-eq*
proof −
fix *u v* :: *real*
assume *uv*: $u + v = 1$
fix *x*
assume $\exists s\ u.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq p \wedge$ *setsum u s* $= 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = x$
then obtain *sx ux* **where**
x: *finite sx* $sx \neq \{\}$ $sx \subseteq p$ *setsum ux sx* $= 1$ $(\sum v{\in}sx.\ ux\ v *_R v) = x$
by *auto*
fix *y*
assume $\exists s\ u.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq p \wedge$ *setsum u s* $= 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = y$
then obtain *sy uy* **where**
y: *finite sy* $sy \neq \{\}$ $sy \subseteq p$ *setsum uy sy* $= 1$ $(\sum v{\in}sy.\ uy\ v *_R v) = y$ **by** *auto*
have *xy*: *finite* $(sx \cup sy)$
using *x*(*1*) *y*(*1*) **by** *auto*
have $**$: $(sx \cup sy) \cap sx = sx$ $(sx \cup sy) \cap sy = sy$
by *auto*
show $\exists s\ ua.$ *finite* $s \wedge s \neq \{\} \wedge s \subseteq p \wedge$
setsum ua s $= 1 \wedge (\sum v{\in}s.\ ua\ v *_R v) = u *_R x + v *_R y$
apply (*rule-tac x*=*sx* $\cup$ *sy* **in** *exI*)
apply (*rule-tac x*=$\lambda a.$ (*if* $a{\in}sx$ *then* $u * ux\ a$ *else* 0) + (*if* $a{\in}sy$ *then* $v * uy\ a$ *else* 0) **in** *exI*)
unfolding *scaleR-left-distrib setsum.distrib if-smult scaleR-zero-left*
$**$ *setsum.inter-restrict*[*OF xy*, *symmetric*]
unfolding *scaleR-scaleR*[*symmetric*] *Real-Vector-Spaces.scaleR-right.setsum*[*symmetric*]
and *setsum-right-distrib*[*symmetric*]
unfolding *x y*
using *x*(*1*−*3*) *y*(*1*−*3*) *uv*
apply *simp*
done

qed
qed

lemma *affine-hull-finite*:
assumes *finite s*
shows *affine hull s* = {*y*. ∃ *u*. *setsum u s* = *1* ∧ *setsum* (λ*v*. *u v* ∗$_R$ *v*) *s* = *y*}
unfolding *affine-hull-explicit* **and** *set-eq-iff* **and** *mem-Collect-eq*
apply (*rule*, *rule*)
apply (*erule exE*)+
apply (*erule conjE*)+
defer
apply (*erule exE*)
apply (*erule conjE*)
proof −
fix *x u*
assume *setsum u s* = *1* (∑ *v*∈*s*. *u v* ∗$_R$ *v*) = *x*
then show ∃ *sa u*. *finite sa* ∧
¬ (∀ *x*. (*x* ∈ *sa*) = (*x* ∈ {})) ∧ *sa* ⊆ *s* ∧ *setsum u sa* = *1* ∧ (∑ *v*∈*sa*. *u v* ∗$_R$ *v*) = *x*
apply (*rule-tac x*=*s* **in** *exI*, *rule-tac x*=*u* **in** *exI*)
using *assms*
apply *auto*
done
next
fix *x t u*
assume *t* ⊆ *s*
then have ∗: *s* ∩ *t* = *t*
by *auto*
assume *finite t* ¬ (∀ *x*. (*x* ∈ *t*) = (*x* ∈ {})) *setsum u t* = *1* (∑ *v*∈*t*. *u v* ∗$_R$ *v*) = *x*
then show ∃ *u*. *setsum u s* = *1* ∧ (∑ *v*∈*s*. *u v* ∗$_R$ *v*) = *x*
apply (*rule-tac x*=λ*x*. *if x*∈*t then u x else 0* **in** *exI*)
unfolding *if-smult scaleR-zero-left* **and** *setsum.inter-restrict*[*OF assms*, *symmetric*] **and** ∗
apply *auto*
done
qed

19.1.2 Stepping theorems and hence small special cases

lemma *affine-hull-empty*[*simp*]: *affine hull* {} = {}
by (*rule hull-unique*) *auto*

lemma *affine-hull-finite-step*:
fixes *y* :: *'a*::*real-vector*
shows
(∃ *u*. *setsum u* {} = *w* ∧ *setsum* (λ*x*. *u x* ∗$_R$ *x*) {} = *y*) ⟷ *w* = *0* ∧ *y* = *0* (**is** *?th1*)
and

```
    finite s ⟹
      (∃ u. setsum u (insert a s) = w ∧ setsum (λx. u x *R x) (insert a s) = y)
⟷
      (∃ v u. setsum u s = w − v ∧ setsum (λx. u x *R x) s = y − v *R a) (is -
⟹ ?lhs = ?rhs)
proof −
  show ?th1 by simp
  assume fin: finite s
  show ?lhs = ?rhs
  proof
    assume ?lhs
    then obtain u where u: setsum u (insert a s) = w ∧ (∑x∈insert a s. u x
*R x) = y
      by auto
    show ?rhs
    proof (cases a ∈ s)
      case True
      then have ∗: insert a s = s by auto
      show ?thesis
        using u[unfolded ∗]
        apply(rule-tac x=0 in exI)
        apply auto
        done
    next
      case False
      then show ?thesis
        apply (rule-tac x=u a in exI)
        using u and fin
        apply auto
        done
    qed
  next
    assume ?rhs
    then obtain v u where vu: setsum u s = w − v  (∑x∈s. u x *R x) = y − v
*R a
      by auto
    have ∗: ⋀x M. (if x = a then v else M) *R x = (if x = a then v *R x else M
*R x)
      by auto
    show ?lhs
    proof (cases a ∈ s)
      case True
      then show ?thesis
        apply (rule-tac x=λx. (if x=a then v else 0) + u x in exI)
        unfolding setsum-clauses(2)[OF fin]
        apply simp
        unfolding scaleR-left-distrib and setsum.distrib
        unfolding vu and ∗ and scaleR-zero-left
        apply (auto simp add: setsum.delta[OF fin])
```

```
      done
    next
      case False
      then have **:
        ⋀x. x ∈ s ⟹ u x = (if x = a then v else u x)
        ⋀x. x ∈ s ⟹ u x *R x = (if x = a then v *R x else u x *R x) by auto
      from False show ?thesis
        apply (rule-tac x=λx. if x=a then v else u x in exI)
        unfolding setsum-clauses(2)[OF fin] and * using vu
       using setsum.cong [of s - λx. u x *R x λx. if x = a then v *R x else u x *R
x, OF - **(2)]
        using setsum.cong [of s - u λx. if x = a then v else u x, OF - **(1)]
        apply auto
        done
    qed
  qed
qed

lemma affine-hull-2:
  fixes a b :: 'a::real-vector
  shows affine hull {a,b} = {u *R a + v *R b| u v. (u + v = 1)}
  (is ?lhs = ?rhs)
proof −
  have *:
    ⋀x y z. z = x − y ⟷ y + z = (x::real)
    ⋀x y z. z = x − y ⟷ y + z = (x::'a) by auto
  have ?lhs = {y. ∃u. setsum u {a, b} = 1 ∧ (∑v∈{a, b}. u v *R v) = y}
    using affine-hull-finite[of {a,b}] by auto
  also have ... = {y. ∃v u. u b = 1 − v ∧ u b *R b = y − v *R a}
    by (simp add: affine-hull-finite-step(2)[of {b} a])
  also have ... = ?rhs unfolding * by auto
  finally show ?thesis by auto
qed

lemma affine-hull-3:
  fixes a b c :: 'a::real-vector
  shows affine hull {a,b,c} = { u *R a + v *R b + w *R c| u v w. u + v + w =
1}
proof −
  have *:
    ⋀x y z. z = x − y ⟷ y + z = (x::real)
    ⋀x y z. z = x − y ⟷ y + z = (x::'a) by auto
  show ?thesis
    apply (simp add: affine-hull-finite affine-hull-finite-step)
    unfolding *
    apply auto
    apply (rule-tac x=v in exI)
    apply (rule-tac x=va in exI)
    apply auto
```

```
    apply (rule-tac x=u in exI)
    apply force
    done
qed

lemma mem-affine:
  assumes affine S x ∈ S y ∈ S u + v = 1
  shows u *R x + v *R y ∈ S
  using assms affine-def[of S] by auto

lemma mem-affine-3:
  assumes affine S x ∈ S y ∈ S z ∈ S u + v + w = 1
  shows u *R x + v *R y + w *R z ∈ S
proof −
  have u *R x + v *R y + w *R z ∈ affine hull {x, y, z}
    using affine-hull-3[of x y z] assms by auto
  moreover
  have affine hull {x, y, z} ⊆ affine hull S
    using hull-mono[of {x, y, z} S] assms by auto
  moreover
  have affine hull S = S
    using assms affine-hull-eq[of S] by auto
  ultimately show ?thesis by auto
qed

lemma mem-affine-3-minus:
  assumes affine S x ∈ S y ∈ S z ∈ S
  shows x + v *R (y−z) ∈ S
  using mem-affine-3[of S x y z 1 v −v] assms
  by (simp add: algebra-simps)

corollary mem-affine-3-minus2:
    ⟦affine S; x ∈ S; y ∈ S; z ∈ S⟧ ⟹ x − v *R (y−z) ∈ S
  by (metis add-uminus-conv-diff mem-affine-3-minus real-vector.scale-minus-left)
```

19.1.3 Some relations between affine hull and subspaces

```
lemma affine-hull-insert-subset-span:
  affine hull (insert a s) ⊆ {a + v| v . v ∈ span {x − a | x . x ∈ s}}
  unfolding subset-eq Ball-def
  unfolding affine-hull-explicit span-explicit mem-Collect-eq
  apply (rule, rule)
  apply (erule exE)+
  apply (erule conjE)+
proof −
  fix x t u
  assume as: finite t t ≠ {} t ⊆ insert a s setsum u t = 1 (∑ v∈t. u v *R v) = x
  have (λx. x − a) ‘ (t − {a}) ⊆ {x − a |x. x ∈ s}
    using as(3) by auto
```

then show $\exists v.\ x = a + v \wedge (\exists S\ u.\ \textit{finite}\ S \wedge S \subseteq \{x - a\ |x.\ x \in s\} \wedge (\sum v{\in}S.\ u\ v *_R v) = v)$
apply (*rule-tac* $x{=}x - a$ **in** *exI*)
apply (*rule conjI*, *simp*)
apply (*rule-tac* $x{=}(\lambda x.\ x - a)$ ‘ $(t - \{a\})$ **in** *exI*)
apply (*rule-tac* $x{=}\lambda x.\ u\ (x + a)$ **in** *exI*)
apply (*rule conjI*) **using** *as(1)* **apply** *simp*
apply (*erule conjI*)
using *as(1)*
apply (*simp add*: *setsum.reindex*[*unfolded inj-on-def*] *scaleR-right-diff-distrib* *setsum-subtractf scaleR-left.setsum*[*symmetric*] *setsum-diff1 scaleR-left-diff-distrib*)
unfolding *as*
apply *simp*
done
qed

lemma *affine-hull-insert-span*:
assumes $a \notin s$
shows *affine hull* (*insert a s*) = $\{a + v \mid v\ .\ v \in \textit{span}\ \{x - a \mid x.\ \ x \in s\}\}$
apply (*rule*, *rule affine-hull-insert-subset-span*)
unfolding *subset-eq Ball-def*
unfolding *affine-hull-explicit* **and** *mem-Collect-eq*
proof (*rule*, *rule*, *erule exE*, *erule conjE*)
fix *y v*
assume $y = a + v\ v \in \textit{span}\ \{x - a\ |x.\ x \in s\}$
then obtain *t u* **where** *obt*: *finite t* $t \subseteq \{x - a\ |x.\ x \in s\}\ a + (\sum v{\in}t.\ u\ v *_R v) = y$
unfolding *span-explicit* **by** *auto*
def $f \equiv (\lambda x.\ x + a)$ ‘ *t*
have *f*: *finite f* $f \subseteq s\ (\sum v{\in}f.\ u\ (v - a) *_R (v - a)) = y - a$
unfolding *f-def* **using** *obt* **by** (*auto simp add*: *setsum.reindex*[*unfolded inj-on-def*])
have $*$: $f \cap \{a\} = \{\}\ f \cap - \{a\} = f$
using *f(2) assms* **by** *auto*
show $\exists sa\ u.\ \textit{finite}\ sa \wedge sa \neq \{\} \wedge sa \subseteq \textit{insert}\ a\ s \wedge \textit{setsum}\ u\ sa = 1 \wedge (\sum v{\in}sa.\ u\ v *_R v) = y$
apply (*rule-tac* $x = \textit{insert}\ a\ f$ **in** *exI*)
apply (*rule-tac* $x = \lambda x.\ \textit{if}\ x{=}a\ \textit{then}\ 1 - \textit{setsum}\ (\lambda x.\ u\ (x - a))\ f\ \textit{else}\ u\ (x - a)$ **in** *exI*)
using *assms* **and** *f*
unfolding *setsum-clauses(2)*[*OF f(1)*] **and** *if-smult*
unfolding *setsum.If-cases*[*OF f(1)*, *of* $\lambda x.\ x = a$]
apply (*auto simp add*: *setsum-subtractf scaleR-left.setsum algebra-simps* $*$)
done
qed

lemma *affine-hull-span*:
assumes $a \in s$
shows *affine hull* $s = \{a + v \mid v.\ v \in \textit{span}\ \{x - a \mid x.\ x \in s - \{a\}\}\}$
using *affine-hull-insert-span*[*of a* $s - \{a\}$, *unfolded insert-Diff*[*OF assms*]] **by**

auto

19.1.4 Parallel affine sets

definition *affine-parallel* :: *'a::real-vector set* ⇒ *'a::real-vector set* ⇒ *bool*
where *affine-parallel S T* ⟷ (∃ *a*. *T* = (λ*x*. *a* + *x*) ‘ *S*)

lemma *affine-parallel-expl-aux*:
fixes *S T* :: *'a::real-vector set*
assumes ∀ *x*. *x* ∈ *S* ⟷ *a* + *x* ∈ *T*
shows *T* = (λ*x*. *a* + *x*) ‘ *S*
proof −
{
fix *x*
assume *x* ∈ *T*
then have (− *a*) + *x* ∈ *S*
using *assms* **by** *auto*
then have *x* ∈ ((λ*x*. *a* + *x*) ‘ *S*)
using *imageI*[*of* −*a*+*x S* (λ*x*. *a*+*x*)] **by** *auto*
}
moreover have *T* ≥ (λ*x*. *a* + *x*) ‘ *S*
using *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *affine-parallel-expl*: *affine-parallel S T* ⟷ (∃ *a*. ∀ *x*. *x* ∈ *S* ⟷ *a* + *x* ∈ *T*)
unfolding *affine-parallel-def*
using *affine-parallel-expl-aux*[*of S - T*] **by** *auto*

lemma *affine-parallel-reflex*: *affine-parallel S S*
unfolding *affine-parallel-def*
apply (*rule exI*[*of - 0*])
apply *auto*
done

lemma *affine-parallel-commut*:
assumes *affine-parallel A B*
shows *affine-parallel B A*
proof −
from *assms* **obtain** *a* **where** *B*: *B* = (λ*x*. *a* + *x*) ‘ *A*
unfolding *affine-parallel-def* **by** *auto*
have [*simp*]: (λ*x*. *x* − *a*) = *plus* (− *a*) **by** (*simp add*: *fun-eq-iff*)
from *B* **show** *?thesis*
using *translation-galois* [*of B a A*]
unfolding *affine-parallel-def* **by** *auto*
qed

lemma *affine-parallel-assoc*:

assumes *affine-parallel A B*
and *affine-parallel B C*
shows *affine-parallel A C*
proof −
from *assms* **obtain** *ab* **where** $B = (\lambda x.\ ab + x)\ `\ A$
unfolding *affine-parallel-def* **by** *auto*
moreover
from *assms* **obtain** *bc* **where** $C = (\lambda x.\ bc + x)\ `\ B$
unfolding *affine-parallel-def* **by** *auto*
ultimately show *?thesis*
using *translation-assoc*[*of bc ab A*] **unfolding** *affine-parallel-def* **by** *auto*
qed

lemma *affine-translation-aux*:
fixes $a :: 'a$::*real-vector*
assumes *affine* $((\lambda x.\ a + x)\ `\ S)$
shows *affine S*
proof −
{
fix *x y u v*
assume *xy*: $x \in S\ y \in S\ (u :: real) + v = 1$
then have $(a + x) \in ((\lambda x.\ a + x)\ `\ S)\ (a + y) \in ((\lambda x.\ a + x)\ `\ S)$
by *auto*
then have *h1*: $u *_R\ (a + x) + v *_R (a + y) \in (\lambda x.\ a + x)\ `\ S$
using *xy assms* **unfolding** *affine-def* **by** *auto*
have $u *_R (a + x) + v *_R (a + y) = (u + v) *_R a + (u *_R x + v *_R y)$
by (*simp add*: *algebra-simps*)
also have $\ldots = a + (u *_R x + v *_R y)$
using ‹$u + v = 1$› **by** *auto*
ultimately have $a + (u *_R x + v *_R y) \in (\lambda x.\ a + x)\ `\ S$
using *h1* **by** *auto*
then have $u *_R x + v *_R y : S$ **by** *auto*
}
then show *?thesis* **unfolding** *affine-def* **by** *auto*
qed

lemma *affine-translation*:
fixes $a :: 'a$::*real-vector*
shows *affine S* $\longleftrightarrow$ *affine* $((\lambda x.\ a + x)\ `\ S)$
proof −
have *affine S* $\Longrightarrow$ *affine* $((\lambda x.\ a + x)\ `\ S)$
using *affine-translation-aux*[*of* $-a$ $((\lambda x.\ a + x)\ `\ S)$]
using *translation-assoc*[*of* $-a$ *a S*] **by** *auto*
then show *?thesis* **using** *affine-translation-aux* **by** *auto*
qed

lemma *parallel-is-affine*:
fixes *S T* :: $'a$::*real-vector set*
assumes *affine S affine-parallel S T*

```
  shows affine T
proof −
  from assms obtain a where T = (λx. a + x) ‘ S
    unfolding affine-parallel-def by auto
  then show ?thesis
    using affine-translation assms by auto
qed

lemma subspace-imp-affine: subspace s ⟹ affine s
  unfolding subspace-def affine-def by auto
```

19.1.5 Subspace parallel to an affine set

```
lemma subspace-affine: subspace S ⟷ affine S ∧ 0 ∈ S
proof −
  have h0: subspace S ⟹ affine S ∧ 0 ∈ S
    using subspace-imp-affine[of S] subspace-0 by auto
  {
    assume assm: affine S ∧ 0 ∈ S
    {
      fix c :: real
      fix x
      assume x: x ∈ S
      have c *R x = (1−c) *R 0 + c *R x by auto
      moreover
      have (1 − c) *R 0 + c *R x ∈ S
        using affine-alt[of S] assm x by auto
      ultimately have c *R x ∈ S by auto
    }
    then have h1: ∀ c. ∀ x ∈ S. c *R x ∈ S by auto

    {
      fix x y
      assume xy: x ∈ S y ∈ S
      def u == (1 :: real)/2
      have (1/2) *R (x+y) = (1/2) *R (x+y)
        by auto
      moreover
      have (1/2) *R (x+y)=(1/2) *R x + (1−(1/2)) *R y
        by (simp add: algebra-simps)
      moreover
      have (1 − u) *R x + u *R y ∈ S
        using affine-alt[of S] assm xy by auto
      ultimately
      have (1/2) *R (x+y) ∈ S
        using u-def by auto
      moreover
      have x + y = 2 *R ((1/2) *R (x+y))
        by auto
```

```
    ultimately
    have x + y ∈ S
      using h1[rule-format, of (1/2) *R (x+y) 2] by auto
  }
  then have ∀ x ∈ S. ∀ y ∈ S. x + y ∈ S
    by auto
  then have subspace S
    using h1 assm unfolding subspace-def by auto
 }
 then show ?thesis using h0 by metis
qed

lemma affine-diffs-subspace:
 assumes affine S a ∈ S
 shows subspace ((λx. (−a)+x) ' S)
proof −
 have [simp]: (λx. x − a) = plus (− a) by (simp add: fun-eq-iff)
 have affine ((λx. (−a)+x) ' S)
   using affine-translation assms by auto
 moreover have 0 : ((λx. (−a)+x) ' S)
   using assms exI[of (λx. x∈S ∧ −a+x = 0) a] by auto
 ultimately show ?thesis using subspace-affine by auto
qed

lemma parallel-subspace-explicit:
 assumes affine S
   and a ∈ S
 assumes L ≡ {y. ∃ x ∈ S. (−a) + x = y}
 shows subspace L ∧ affine-parallel S L
proof −
 from assms have L = plus (− a) ' S by auto
 then have par: affine-parallel S L
   unfolding affine-parallel-def ..
 then have affine L using assms parallel-is-affine by auto
 moreover have 0 ∈ L
   using assms by auto
 ultimately show ?thesis
   using subspace-affine par by auto
qed

lemma parallel-subspace-aux:
 assumes subspace A
   and subspace B
   and affine-parallel A B
 shows A ⊇ B
proof −
 from assms obtain a where a: ∀ x. x ∈ A ⟷ a + x ∈ B
   using affine-parallel-expl[of A B] by auto
 then have −a ∈ A
```

using *assms subspace-0*[*of B*] **by** *auto*
then have $a \in A$
using *assms subspace-neg*[*of A* $-a$] **by** *auto*
then show *?thesis*
using *assms a* **unfolding** *subspace-def* **by** *auto*
qed

lemma *parallel-subspace*:
assumes *subspace A*
and *subspace B*
and *affine-parallel A B*
shows $A = B$
proof
show $A \supseteq B$
using *assms parallel-subspace-aux* **by** *auto*
show $A \subseteq B$
using *assms parallel-subspace-aux*[*of B A*] *affine-parallel-commut* **by** *auto*
qed

lemma *affine-parallel-subspace*:
assumes *affine S* $S \neq \{\}$
shows $\exists!L.$ *subspace L* $\wedge$ *affine-parallel S L*
proof −
have *ex*: $\exists L.$ *subspace L* $\wedge$ *affine-parallel S L*
using *assms parallel-subspace-explicit* **by** *auto*
{
fix *L1 L2*
assume *ass*: *subspace L1* $\wedge$ *affine-parallel S L1 subspace L2* $\wedge$ *affine-parallel S L2*
then have *affine-parallel L1 L2*
using *affine-parallel-commut*[*of S L1*] *affine-parallel-assoc*[*of L1 S L2*] **by** *auto*
then have *L1* = *L2*
using *ass parallel-subspace* **by** *auto*
}
then show *?thesis* **using** *ex* **by** *auto*
qed

19.2 Cones

definition *cone* :: *'a::real-vector set* ⇒ *bool*
where *cone s* ⟷ $(\forall x{\in}s.\ \forall c{\geq}0.\ c *_R x \in s)$

lemma *cone-empty*[*intro*, *simp*]: *cone* {}
unfolding *cone-def* **by** *auto*

lemma *cone-univ*[*intro*, *simp*]: *cone UNIV*
unfolding *cone-def* **by** *auto*

lemma *cone-Inter*[*intro*]: $\forall s{\in}f.\ cone\ s \Longrightarrow cone\ (\bigcap f)$
unfolding *cone-def* **by** *auto*

19.2.1 Conic hull

lemma *cone-cone-hull*: *cone* (*cone hull s*)
unfolding *hull-def* **by** *auto*

lemma *cone-hull-eq*: *cone hull s* = *s* ⟷ *cone s*
apply (*rule hull-eq*)
using *cone-Inter*
unfolding *subset-eq*
apply *auto*
done

lemma *mem-cone*:
assumes *cone S x* ∈ *S c* ≥ *0*
shows *c* ∗*R x* : *S*
using *assms cone-def*[*of S*] **by** *auto*

lemma *cone-contains-0*:
assumes *cone S*
shows *S* ≠ {} ⟷ *0* ∈ *S*
proof −
{
assume *S* ≠ {}
then obtain *a* **where** *a* ∈ *S* **by** *auto*
then have *0* ∈ *S*
using *assms mem-cone*[*of S a 0*] **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *cone-0*: *cone* {*0*}
unfolding *cone-def* **by** *auto*

lemma *cone-Union*[*intro*]: (∀ *s*∈*f*. *cone s*) ⟶ *cone* (⋃ *f*)
unfolding *cone-def* **by** *blast*

lemma *cone-iff*:
assumes *S* ≠ {}
shows *cone S* ⟷ *0* ∈ *S* ∧ (∀ *c*. *c* > *0* ⟶ (*op* ∗*R c*) ‘ *S* = *S*)
proof −
{
assume *cone S*
{
fix *c* :: *real*
assume *c* > *0*
{

```
      fix x
      assume x ∈ S
      then have x ∈ (op *R c) ' S
        unfolding image-def
        using ⟨cone S⟩ ⟨c>0⟩ mem-cone[of S x 1/c]
          exI[of (λt. t ∈ S ∧ x = c *R t) (1 / c) *R x]
        by auto
    }
    moreover
    {
      fix x
      assume x ∈ (op *R c) ' S
      then have x ∈ S
        using ⟨cone S⟩ ⟨c > 0⟩
        unfolding cone-def image-def ⟨c > 0⟩ by auto
    }
    ultimately have (op *R c) ' S = S by auto
  }
  then have 0 ∈ S ∧ (∀ c. c > 0 ⟶ (op *R c) ' S = S)
    using ⟨cone S⟩ cone-contains-0[of S] assms by auto
 }
 moreover
 {
  assume a: 0 ∈ S ∧ (∀ c. c > 0 ⟶ (op *R c) ' S = S)
  {
    fix x
    assume x ∈ S
    fix c1 :: real
    assume c1 ≥ 0
    then have c1 = 0 ∨ c1 > 0 by auto
    then have c1 *R x ∈ S using a ⟨x ∈ S⟩ by auto
  }
  then have cone S unfolding cone-def by auto
 }
 ultimately show ?thesis by blast
qed

lemma cone-hull-empty: cone hull {} = {}
  by (metis cone-empty cone-hull-eq)

lemma cone-hull-empty-iff: S = {} ⟷ cone hull S = {}
  by (metis bot-least cone-hull-empty hull-subset xtrans(5))

lemma cone-hull-contains-0: S ≠ {} ⟷ 0 ∈ cone hull S
  using cone-cone-hull[of S] cone-contains-0[of cone hull S] cone-hull-empty-iff[of
S]
  by auto

lemma mem-cone-hull:
```

```
  assumes x ∈ S c ≥ 0
  shows c *R x ∈ cone hull S
  by (metis assms cone-cone-hull hull-inc mem-cone)

lemma cone-hull-expl: cone hull S = {c *R x | c x. c ≥ 0 ∧ x ∈ S}
  (is ?lhs = ?rhs)
proof −
  {
    fix x
    assume x ∈ ?rhs
    then obtain cx :: real and xx where x: x = cx *R xx cx ≥ 0 xx ∈ S
      by auto
    fix c :: real
    assume c: c ≥ 0
    then have c *R x = (c * cx) *R xx
      using x by (simp add: algebra-simps)
    moreover
    have c * cx ≥ 0 using c x by auto
    ultimately
    have c *R x ∈ ?rhs using x by auto
  }
  then have cone ?rhs
    unfolding cone-def by auto
  then have ?rhs ∈ Collect cone
    unfolding mem-Collect-eq by auto
  {
    fix x
    assume x ∈ S
    then have 1 *R x ∈ ?rhs
      apply auto
      apply (rule-tac x = 1 in exI)
      apply auto
      done
    then have x ∈ ?rhs by auto
  }
  then have S ⊆ ?rhs by auto
  then have ?lhs ⊆ ?rhs
    using ‹?rhs ∈ Collect cone› hull-minimal[of S ?rhs cone] by auto
  moreover
  {
    fix x
    assume x ∈ ?rhs
    then obtain cx :: real and xx where x: x = cx *R xx cx ≥ 0 xx ∈ S
      by auto
    then have xx ∈ cone hull S
      using hull-subset[of S] by auto
    then have x ∈ ?lhs
      using x cone-cone-hull[of S] cone-def[of cone hull S] by auto
  }
```

ultimately show *?thesis* **by** *auto*
qed

lemma *cone-closure*:
fixes S :: *'a*::*real-normed-vector set*
assumes *cone S*
shows *cone* (*closure S*)
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then have $0 \in S \wedge (\forall c.\ c > 0 \longrightarrow op *_R\ c\ `\ S = S)$
using *cone-iff*[*of S*] *assms* **by** *auto*
then have $0 \in closure\ S \wedge (\forall c.\ c > 0 \longrightarrow op *_R\ c\ `\ closure\ S = closure\ S)$
using *closure-subset* **by** (*auto simp add*: *closure-scaleR*)
then show *?thesis*
using *False cone-iff*[*of closure S*] **by** *auto*
qed

19.3 Affine dependence and consequential theorems (from Lars Schewe)

definition *affine-dependent* :: *'a*::*real-vector set* ⇒ *bool*
where *affine-dependent s* ⟷ $(\exists x{\in}s.\ x \in affine\ hull\ (s - \{x\}))$

lemma *affine-dependent-explicit*:
affine-dependent p ⟷
$(\exists s\ u.\ finite\ s \wedge s \subseteq p \wedge setsum\ u\ s = 0 \wedge$
$(\exists v{\in}s.\ u\ v \neq 0) \wedge setsum\ (\lambda v.\ u\ v *_R v)\ s = 0)$
unfolding *affine-dependent-def affine-hull-explicit mem-Collect-eq*
apply *rule*
apply (*erule bexE*, *erule exE*, *erule exE*)
apply (*erule conjE*)+
defer
apply (*erule exE*, *erule exE*)
apply (*erule conjE*)+
apply (*erule bexE*)
proof −
fix *x s u*
assume *as*: $x \in p$ *finite s* $s \neq \{\}$ $s \subseteq p - \{x\}$ *setsum u s* = *1* $(\sum v{\in}s.\ u\ v *_R v) = x$
have $x \notin s$ **using** *as*(*1*,*4*) **by** *auto*
show $\exists s\ u.\ finite\ s \wedge s \subseteq p \wedge setsum\ u\ s = 0 \wedge (\exists v{\in}s.\ u\ v \neq 0) \wedge (\sum v{\in}s.\ u\ v *_R v) = 0$
apply (*rule-tac x*=*insert x s* **in** *exI*, *rule-tac x*=$\lambda v.$ *if v* = *x then* − *1 else u v* **in** *exI*)
unfolding *if-smult* **and** *setsum-clauses*(*2*)[*OF as*(*2*)] **and** *setsum-delta-notmem*[*OF* ‹$x \notin s$›] **and** *as*

```
    using as
    apply auto
    done
next
  fix s u v
  assume as: finite s s ⊆ p setsum u s = 0 (∑ v∈s. u v *R v) = 0 v ∈ s u v ≠ 0
  have s ≠ {v}
    using as(3,6) by auto
  then show ∃ x∈p. ∃ s u. finite s ∧ s ≠ {} ∧ s ⊆ p − {x} ∧ setsum u s = 1 ∧
(∑ v∈s. u v *R v) = x
    apply (rule-tac x=v in bexI)
    apply (rule-tac x=s − {v} in exI)
    apply (rule-tac x=λx. − (1 / u v) * u x in exI)
    unfolding scaleR-scaleR[symmetric] and scaleR-right.setsum [symmetric]
    unfolding setsum-right-distrib[symmetric] and setsum-diff1[OF as(1)]
    using as
    apply auto
    done
qed

lemma affine-dependent-explicit-finite:
  fixes s :: 'a::real-vector set
  assumes finite s
  shows affine-dependent s ⟷
    (∃ u. setsum u s = 0 ∧ (∃ v∈s. u v ≠ 0) ∧ setsum (λv. u v *R v) s = 0)
  (is ?lhs = ?rhs)
proof
  have *: ⋀vt u v. (if vt then u v else 0) *R v = (if vt then (u v) *R v else 0::'a)
    by auto
  assume ?lhs
  then obtain t u v where
    finite t t ⊆ s setsum u t = 0 v∈t u v ≠ 0  (∑ v∈t. u v *R v) = 0
    unfolding affine-dependent-explicit by auto
  then show ?rhs
    apply (rule-tac x=λx. if x∈t then u x else 0 in exI)
    apply auto unfolding * and setsum.inter-restrict[OF assms, symmetric]
    unfolding Int-absorb1[OF ‹t⊆s›]
    apply auto
    done
next
  assume ?rhs
  then obtain u v where setsum u s = 0  v∈s u v ≠ 0 (∑ v∈s. u v *R v) = 0
    by auto
  then show ?lhs unfolding affine-dependent-explicit
    using assms by auto
qed
```

19.4 Connectedness of convex sets

lemma *connectedD*:
connected S $\Longrightarrow$ *open A* $\Longrightarrow$ *open B* $\Longrightarrow$ $S \subseteq A \cup B \Longrightarrow A \cap B \cap S = \{\} \Longrightarrow A \cap S = \{\} \vee B \cap S = \{\}$
by (*rule Topological-Spaces.topological-space-class.connectedD*)

lemma *convex-connected*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s*
shows *connected s*
proof (*rule connectedI*)
fix *A B*
assume *open A open B* $A \cap B \cap s = \{\}$ $s \subseteq A \cup B$
moreover
assume $A \cap s \neq \{\}$ $B \cap s \neq \{\}$
then obtain *a b* **where** *a*: $a \in A$ $a \in s$ **and** *b*: $b \in B$ $b \in s$ **by** *auto*
def $f \equiv \lambda u.\ u *_R a + (1 - u) *_R b$
then have *continuous-on* $\{0 .. 1\}$ *f*
by (*auto intro*!: *continuous-intros*)
then have *connected* (*f* ‘ $\{0 .. 1\}$)
by (*auto intro*!: *connected-continuous-image*)
note *connectedD*[*OF this*, *of A B*]
moreover have $a \in A \cap f$ ‘ $\{0 .. 1\}$
using *a* **by** (*auto intro*!: *image-eqI*[*of - - 1*] *simp*: *f-def*)
moreover have $b \in B \cap f$ ‘ $\{0 .. 1\}$
using *b* **by** (*auto intro*!: *image-eqI*[*of - - 0*] *simp*: *f-def*)
moreover have f ‘ $\{0 .. 1\} \subseteq s$
using ‹*convex s*› *a b* **unfolding** *convex-def f-def* **by** *auto*
ultimately show *False* **by** *auto*
qed

corollary *connected-UNIV*[*intro*]: *connected* (*UNIV* :: *'a*::*real-normed-vector set*)
by(*simp add*: *convex-connected*)

proposition *clopen*:
fixes *s* :: *'a* :: *real-normed-vector set*
shows *closed s* $\wedge$ *open s* $\longleftrightarrow$ $s = \{\} \vee s = UNIV$
apply (*rule iffI*)
apply (*rule connected-UNIV* [*unfolded connected-clopen*, *rule-format*])
apply (*force simp add*: *open-openin closed-closedin*, *force*)
done

corollary *compact-open*:
fixes *s* :: *'a* :: *euclidean-space set*
shows *compact s* $\wedge$ *open s* $\longleftrightarrow$ $s = \{\}$
by (*auto simp*: *compact-eq-bounded-closed clopen*)

Balls, being convex, are connected.

lemma *convex-prod*:

assumes $\bigwedge i.\ i \in Basis \Longrightarrow convex\ \{x.\ P\ i\ x\}$
shows $convex\ \{x.\ \forall i{\in}Basis.\ P\ i\ (x{\cdot}i)\}$
using *assms* **unfolding** *convex-def*
by (*auto simp*: *inner-add-left*)

lemma *convex-positive-orthant*: $convex\ \{x::'a::euclidean\text{-}space.\ (\forall i{\in}Basis.\ 0 \le x{\cdot}i)\}$
by (*rule convex-prod*) (*simp add*: *atLeast-def*[*symmetric*] *convex-real-interval*)

lemma *convex-local-global-minimum*:
fixes $s :: 'a::real\text{-}normed\text{-}vector\ set$
assumes $e > 0$
and *convex-on s f*
and $ball\ x\ e \subseteq s$
and $\forall y{\in}ball\ x\ e.\ f\ x \le f\ y$
shows $\forall y{\in}s.\ f\ x \le f\ y$
proof (*rule ccontr*)
have $x \in s$ **using** *assms*(*1*,*3*) **by** *auto*
assume $\neg$ *?thesis*
then obtain *y* **where** $y{\in}s$ **and** *y*: $f\ x > f\ y$ **by** *auto*
then have *xy*: $0 < dist\ x\ y$ **by** *auto*
then obtain *u* **where** $0 < u\ u \le 1$ **and** *u*: $u < e\ /\ dist\ x\ y$
using *real-lbound-gt-zero*[*of 1 e / dist x y*] *xy* ‹*e*>*0*› **by** *auto*
then have $f\ ((1{-}u) *_R x + u *_R y) \le (1{-}u) * f\ x + u * f\ y$
using ‹*x*∈*s*› ‹*y*∈*s*›
using *assms*(*2*)[*unfolded convex-on-def*,
THEN bspec[**where** *x*=*x*], *THEN bspec*[**where** *x*=*y*], *THEN spec*[**where** *x*=*1*−*u*]]
by *auto*
moreover
have ∗: $x - ((1 - u) *_R x + u *_R y) = u *_R (x - y)$
by (*simp add*: *algebra-simps*)
have $(1 - u) *_R x + u *_R y \in ball\ x\ e$
unfolding *mem-ball dist-norm*
unfolding ∗ **and** *norm-scaleR* **and** *abs-of-pos*[*OF* ‹*0*<*u*›]
unfolding *dist-norm*[*symmetric*]
using *u*
unfolding *pos-less-divide-eq*[*OF xy*]
by *auto*
then have $f\ x \le f\ ((1 - u) *_R x + u *_R y)$
using *assms*(*4*) **by** *auto*
ultimately show *False*
using *mult-strict-left-mono*[*OF y* ‹*u*>*0*›]
unfolding *left-diff-distrib*
by *auto*
qed

lemma *convex-ball* [*iff*]:
fixes $x :: 'a::real\text{-}normed\text{-}vector$

```
  shows convex (ball x e)
proof (auto simp add: convex-def)
  fix y z
  assume yz: dist x y < e dist x z < e
  fix u v :: real
  assume uv: 0 ≤ u 0 ≤ v u + v = 1
  have dist x (u *R y + v *R z) ≤ u * dist x y + v * dist x z
    using uv yz
    using convex-on-dist [of ball x e x, unfolded convex-on-def,
      THEN bspec[where x=y], THEN bspec[where x=z]]
    by auto
  then show dist x (u *R y + v *R z) < e
    using convex-bound-lt[OF yz uv] by auto
qed

lemma convex-cball [iff]:
  fixes x :: 'a::real-normed-vector
  shows convex (cball x e)
proof −
  {
    fix y z
    assume yz: dist x y ≤ e dist x z ≤ e
    fix u v :: real
    assume uv: 0 ≤ u 0 ≤ v u + v = 1
    have dist x (u *R y + v *R z) ≤ u * dist x y + v * dist x z
      using uv yz
      using convex-on-dist [of cball x e x, unfolded convex-on-def,
        THEN bspec[where x=y], THEN bspec[where x=z]]
      by auto
    then have dist x (u *R y + v *R z) ≤ e
      using convex-bound-le[OF yz uv] by auto
  }
  then show ?thesis by (auto simp add: convex-def Ball-def)
qed

lemma connected-ball [iff]:
  fixes x :: 'a::real-normed-vector
  shows connected (ball x e)
  using convex-connected convex-ball by auto

lemma connected-cball [iff]:
  fixes x :: 'a::real-normed-vector
  shows connected (cball x e)
  using convex-connected convex-cball by auto
```

19.5 Convex hull

```
lemma convex-convex-hull [iff]: convex (convex hull s)
  unfolding hull-def
```

using *convex-Inter*[*of* {*t*. *convex t* ∧ *s* ⊆ *t*}]
by *auto*

lemma *convex-hull-eq*: *convex hull s* = *s* ⟷ *convex s*
by (*metis convex-convex-hull hull-same*)

lemma *bounded-convex-hull*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *bounded s*
shows *bounded* (*convex hull s*)
proof −
from *assms* **obtain** *B* **where** *B*: ∀ *x*∈*s*. *norm x* ≤ *B*
unfolding *bounded-iff* **by** *auto*
show *?thesis*
apply (*rule bounded-subset*[*OF bounded-cball*, *of - 0 B*])
unfolding *subset-hull*[*of convex*, *OF convex-cball*]
unfolding *subset-eq mem-cball dist-norm* **using** *B*
apply *auto*
done
qed

lemma *finite-imp-bounded-convex-hull*:
fixes *s* :: *'a*::*real-normed-vector set*
shows *finite s* ⟹ *bounded* (*convex hull s*)
using *bounded-convex-hull finite-imp-bounded*
by *auto*

19.5.1 Convex hull is "preserved" by a linear function

lemma *convex-hull-linear-image*:
assumes *f*: *linear f*
shows *f* ‘ (*convex hull s*) = *convex hull* (*f* ‘ *s*)
proof
show *convex hull* (*f* ‘ *s*) ⊆ *f* ‘ (*convex hull s*)
by (*intro hull-minimal image-mono hull-subset convex-linear-image assms convex-convex-hull*)
show *f* ‘ (*convex hull s*) ⊆ *convex hull* (*f* ‘ *s*)
proof (*unfold image-subset-iff-subset-vimage*, *rule hull-minimal*)
show *s* ⊆ *f* −‘ (*convex hull* (*f* ‘ *s*))
by (*fast intro*: *hull-inc*)
show *convex* (*f* −‘ (*convex hull* (*f* ‘ *s*)))
by (*intro convex-linear-vimage* [*OF f*] *convex-convex-hull*)
qed
qed

lemma *in-convex-hull-linear-image*:
assumes *linear f*
and *x* ∈ *convex hull s*
shows *f x* ∈ *convex hull* (*f* ‘ *s*)
using *convex-hull-linear-image*[*OF assms*(*1*)] *assms*(*2*) **by** *auto*

lemma *convex-hull-Times*:
convex hull $(s \times t) = (\textit{convex hull } s) \times (\textit{convex hull } t)$
proof
show *convex hull* $(s \times t) \subseteq (\textit{convex hull } s) \times (\textit{convex hull } t)$
by (*intro hull-minimal Sigma-mono hull-subset convex-Times convex-convex-hull*)
have $\forall x{\in}\textit{convex hull } s.\ \forall y{\in}\textit{convex hull } t.\ (x,\ y) \in \textit{convex hull } (s \times t)$
proof (*intro hull-induct*)
fix $x\ y$ **assume** $x \in s$ **and** $y \in t$
then show $(x,\ y) \in \textit{convex hull } (s \times t)$
by (*simp add*: *hull-inc*)
next
fix x **let** $?S = ((\lambda y.\ (0,\ y)) -` (\lambda p.\ (-\ x,\ 0) + p)\ ` (\textit{convex hull } s \times t))$
have *convex ?S*
by (*intro convex-linear-vimage convex-translation convex-convex-hull*,
simp add: *linear-iff*)
also have $?S = \{y.\ (x,\ y) \in \textit{convex hull } (s \times t)\}$
by (*auto simp add*: *image-def Bex-def*)
finally show *convex* $\{y.\ (x,\ y) \in \textit{convex hull } (s \times t)\}$ **.**
next
show *convex* $\{x.\ \forall y{\in}\textit{convex hull } t.\ (x,\ y) \in \textit{convex hull } (s \times t)\}$
proof (*unfold Collect-ball-eq*, *rule convex-INT* [*rule-format*])
fix y **let** $?S = ((\lambda x.\ (x,\ 0)) -` (\lambda p.\ (0,\ -\ y) + p)\ ` (\textit{convex hull } s \times t))$
have *convex ?S*
by (*intro convex-linear-vimage convex-translation convex-convex-hull*,
simp add: *linear-iff*)
also have $?S = \{x.\ (x,\ y) \in \textit{convex hull } (s \times t)\}$
by (*auto simp add*: *image-def Bex-def*)
finally show *convex* $\{x.\ (x,\ y) \in \textit{convex hull } (s \times t)\}$ **.**
qed
qed
then show $(\textit{convex hull } s) \times (\textit{convex hull } t) \subseteq \textit{convex hull } (s \times t)$
unfolding *subset-eq split-paired-Ball-Sigma* **.**
qed

19.5.2 Stepping theorems for convex hulls of finite sets

lemma *convex-hull-empty*[*simp*]: *convex hull* $\{\} = \{\}$
by (*rule hull-unique*) *auto*

lemma *convex-hull-singleton*[*simp*]: *convex hull* $\{a\} = \{a\}$
by (*rule hull-unique*) *auto*

lemma *convex-hull-insert*:
fixes $s :: {'a}{::}\textit{real-vector set}$
assumes $s \neq \{\}$
shows *convex hull* $(\textit{insert } a\ s) =$
$\{x.\ \exists u{\geq}0.\ \exists v{\geq}0.\ \exists b.\ (u + v = 1) \wedge b \in (\textit{convex hull } s) \wedge (x = u *_R a + v *_R b)\}$

```
  (is - = ?hull)
  apply (rule, rule hull-minimal, rule)
  unfolding insert-iff
  prefer 3
  apply rule
proof −
  fix x
  assume x: x = a ∨ x ∈ s
  then show x ∈ ?hull
    apply rule
    unfolding mem-Collect-eq
    apply (rule-tac x=1 in exI)
    defer
    apply (rule-tac x=0 in exI)
    using assms hull-subset[of s convex]
    apply auto
    done
next
  fix x
  assume x ∈ ?hull
  then obtain u v b where obt: u≥0 v≥0 u + v = 1 b ∈ convex hull s x = u *R
a + v *R b
    by auto
  have a ∈ convex hull insert a s b ∈ convex hull insert a s
     using hull-mono[of s insert a s convex] hull-mono[of {a} insert a s convex]
and obt(4)
    by auto
  then show x ∈ convex hull insert a s
    unfolding obt(5) using obt(1−3)
    by (rule convexD [OF convex-convex-hull])
next
  show convex ?hull
  proof (rule convexI)
    fix x y u v
    assume as: (0::real) ≤ u 0 ≤ v u + v = 1 x∈?hull y∈?hull
    from as(4) obtain u1 v1 b1 where
      obt1: u1≥0 v1≥0 u1 + v1 = 1 b1 ∈ convex hull s x = u1 *R a + v1 *R b1
      by auto
    from as(5) obtain u2 v2 b2 where
      obt2: u2≥0 v2≥0 u2 + v2 = 1 b2 ∈ convex hull s y = u2 *R a + v2 *R b2
      by auto
    have *: ⋀(x::'a) s1 s2. x − s1 *R x − s2 *R x = ((1::real) − (s1 + s2)) *R x
      by (auto simp add: algebra-simps)
    have **: ∃ b ∈ convex hull s. u *R x + v *R y =
      (u * u1) *R a + (v * u2) *R a + (b − (u * u1) *R b − (v * u2) *R b)
    proof (cases u * v1 + v * v2 = 0)
      case True
       have *: ⋀(x::'a) s1 s2. x − s1 *R x − s2 *R x = ((1::real) − (s1 + s2))
*R x
```

```
      by (auto simp add: algebra-simps)
    from True have ***: u * v1 = 0 v * v2 = 0
      using mult-nonneg-nonneg[OF ‹u≥0› ‹v1≥0›] mult-nonneg-nonneg[OF
‹v≥0› ‹v2≥0›]
      by arith+
    then have u * u1 + v * u2 = 1
      using as(3) obt1(3) obt2(3) by auto
    then show ?thesis
      unfolding obt1(5) obt2(5) *
      using assms hull-subset[of s convex]
      by (auto simp add: *** scaleR-right-distrib)
  next
    case False
    have 1 - (u * u1 + v * u2) = (u + v) - (u * u1 + v * u2)
      using as(3) obt1(3) obt2(3) by (auto simp add: field-simps)
    also have ... = u * (v1 + u1 - u1) + v * (v2 + u2 - u2)
      using as(3) obt1(3) obt2(3) by (auto simp add: field-simps)
    also have ... = u * v1 + v * v2
      by simp
    finally have **:1 - (u * u1 + v * u2) = u * v1 + v * v2 by auto
    have 0 ≤ u * v1 + v * v2 0 ≤ u * v1 0 ≤ u * v1 + v * v2 0 ≤ v * v2
      using as(1,2) obt1(1,2) obt2(1,2) by auto
    then show ?thesis
      unfolding obt1(5) obt2(5)
      unfolding * and **
      using False
      apply (rule-tac
        x = ((u * v1) / (u * v1 + v * v2)) *_R b1 + ((v * v2) / (u * v1 + v *
v2)) *_R b2 in bexI)
      defer
      apply (rule convexD [OF convex-convex-hull])
      using obt1(4) obt2(4)
      unfolding add-divide-distrib[symmetric] and zero-le-divide-iff
      apply (auto simp add: scaleR-left-distrib scaleR-right-distrib)
      done
  qed
  have u1: u1 ≤ 1
    unfolding obt1(3)[symmetric] and not-le using obt1(2) by auto
  have u2: u2 ≤ 1
    unfolding obt2(3)[symmetric] and not-le using obt2(2) by auto
  have u1 * u + u2 * v ≤ max u1 u2 * u + max u1 u2 * v
    apply (rule add-mono)
    apply (rule-tac [!] mult-right-mono)
    using as(1,2) obt1(1,2) obt2(1,2)
    apply auto
    done
  also have ... ≤ 1
    unfolding distrib-left[symmetric] and as(3) using u1 u2 by auto
  finally show u *_R x + v *_R y ∈ ?hull
```

```
    unfolding mem-Collect-eq
    apply (rule-tac x=u * u1 + v * u2 in exI)
    apply (rule conjI)
    defer
    apply (rule-tac x=1 - u * u1 - v * u2 in exI)
    unfolding Bex-def
    using as(1,2) obt1(1,2) obt2(1,2) **
    apply (auto simp add: algebra-simps)
    done
  qed
qed
```

19.5.3 Explicit expression for convex hull

```
lemma convex-hull-indexed:
  fixes s :: 'a::real-vector set
  shows convex hull s =
    {y. ∃k u x.
      (∀i∈{1::nat .. k}. 0 ≤ u i ∧ x i ∈ s) ∧
      (setsum u {1..k} = 1) ∧ (setsum (λi. u i *R x i) {1..k} = y)}
  (is ?xyz = ?hull)
  apply (rule hull-unique)
  apply rule
  defer
  apply (rule convexI)
proof −
  fix x
  assume x∈s
  then show x ∈ ?hull
    unfolding mem-Collect-eq
    apply (rule-tac x=1 in exI, rule-tac x=λx. 1 in exI)
    apply auto
    done
next
  fix t
  assume as: s ⊆ t convex t
  show ?hull ⊆ t
    apply rule
    unfolding mem-Collect-eq
    apply (elim exE conjE)
  proof −
    fix x k u y
    assume assm:
      ∀i∈{1::nat..k}. 0 ≤ u i ∧ y i ∈ s
      setsum u {1..k} = 1 (∑i = 1..k. u i *R y i) = x
    show x∈t
      unfolding assm(3) [symmetric]
      apply (rule as(2)[unfolded convex, rule-format])
      using assm(1,2) as(1) apply auto
```

```
    done
  qed
next
  fix x y u v
  assume uv: 0 ≤ u 0 ≤ v u + v = (1::real)
  assume xy: x ∈ ?hull y ∈ ?hull
  from xy obtain k1 u1 x1 where
    x: ∀ i∈{1::nat..k1}. 0≤u1 i ∧ x1 i ∈ s setsum u1 {Suc 0..k1} = 1 (∑ i = Suc
0..k1. u1 i *R x1 i) = x
    by auto
  from xy obtain k2 u2 x2 where
    y: ∀ i∈{1::nat..k2}. 0≤u2 i ∧ x2 i ∈ s setsum u2 {Suc 0..k2} = 1 (∑ i = Suc
0..k2. u2 i *R x2 i) = y
    by auto
  have *: ⋀P (x1::'a) x2 s1 s2 i.
    (if P i then s1 else s2) *R (if P i then x1 else x2) = (if P i then s1 *R x1 else
s2 *R x2)
    {1..k1 + k2} ∩ {1..k1} = {1..k1} {1..k1 + k2} ∩ - {1..k1} = (λi. i + k1)
‘ {1..k2}
    prefer 3
    apply (rule, rule)
    unfolding image-iff
    apply (rule-tac x = x - k1 in bexI)
    apply (auto simp add: not-le)
    done
  have inj: inj-on (λi. i + k1) {1..k2}
    unfolding inj-on-def by auto
  show u *R x + v *R y ∈ ?hull
    apply rule
    apply (rule-tac x=k1 + k2 in exI)
   apply (rule-tac x=λi. if i ∈ {1..k1} then u * u1 i else v * u2 (i - k1) in exI)
    apply (rule-tac x=λi. if i ∈ {1..k1} then x1 i else x2 (i - k1) in exI)
    apply (rule, rule)
    defer
    apply rule
    unfolding * and setsum.If-cases[OF finite-atLeastAtMost[of 1 k1 + k2]] and
      setsum.reindex[OF inj] and o-def Collect-mem-eq
   unfolding scaleR-scaleR[symmetric] scaleR-right.setsum [symmetric] setsum-right-distrib[symmetric]
  proof -
    fix i
    assume i: i ∈ {1..k1+k2}
    show 0 ≤ (if i ∈ {1..k1} then u * u1 i else v * u2 (i - k1)) ∧
      (if i ∈ {1..k1} then x1 i else x2 (i - k1)) ∈ s
    proof (cases i∈{1..k1})
      case True
      then show ?thesis
        using uv(1) x(1)[THEN bspec[where x=i]] by auto
    next
      case False
```

```
      def j ≡ i − k1
      from i False have j ∈ {1..k2}
        unfolding j-def by auto
      then show ?thesis
        using False uv(2) y(1)[THEN bspec[where x=j]]
        by (auto simp: j-def[symmetric])
    qed
  qed (auto simp add: not-le x(2,3) y(2,3) uv(3))
qed

lemma convex-hull-finite:
  fixes s :: 'a::real-vector set
  assumes finite s
  shows convex hull s = {y. ∃ u. (∀ x∈s. 0 ≤ u x) ∧
    setsum u s = 1 ∧ setsum (λx. u x *R x) s = y}
  (is ?HULL = ?set)
proof (rule hull-unique, auto simp add: convex-def[of ?set])
  fix x
  assume x ∈ s
  then show ∃ u. (∀ x∈s. 0 ≤ u x) ∧ setsum u s = 1 ∧ (∑ x∈s. u x *R x) = x
    apply (rule-tac x=λy. if x=y then 1 else 0 in exI)
    apply auto
    unfolding setsum.delta'[OF assms] and setsum-delta''[OF assms]
    apply auto
    done
next
  fix u v :: real
  assume uv: 0 ≤ u 0 ≤ v u + v = 1
  fix ux assume ux: ∀ x∈s. 0 ≤ ux x setsum ux s = (1::real)
  fix uy assume uy: ∀ x∈s. 0 ≤ uy x setsum uy s = (1::real)
  {
    fix x
    assume x∈s
    then have 0 ≤ u * ux x + v * uy x
      using ux(1)[THEN bspec[where x=x]] uy(1)[THEN bspec[where x=x]] and
uv(1,2)
      by auto
  }
  moreover
  have (∑ x∈s. u * ux x + v * uy x) = 1
   unfolding setsum.distrib and setsum-right-distrib[symmetric] and ux(2) uy(2)
    using uv(3) by auto
  moreover
  have (∑ x∈s. (u * ux x + v * uy x) *R x) = u *R (∑ x∈s. ux x *R x) + v *R
(∑ x∈s. uy x *R x)
    unfolding scaleR-left-distrib and setsum.distrib and scaleR-scaleR[symmetric]
      and scaleR-right.setsum [symmetric]
    by auto
  ultimately
```

```
  show ∃ uc. (∀ x∈s. 0 ≤ uc x) ∧ setsum uc s = 1 ∧
      (∑ x∈s. uc x *R x) = u *R (∑ x∈s. ux x *R x) + v *R (∑ x∈s. uy x *R x)
    apply (rule-tac x=λx. u * ux x + v * uy x in exI)
    apply auto
    done
next
  fix t
  assume t: s ⊆ t convex t
  fix u
  assume u: ∀ x∈s. 0 ≤ u x setsum u s = (1::real)
  then show (∑ x∈s. u x *R x) ∈ t
   using t(2)[unfolded convex-explicit, THEN spec[where x=s], THEN spec[where
x=u]]
    using assms and t(1) by auto
qed
```

19.5.4 Another formulation from Lars Schewe

```
lemma convex-hull-explicit:
  fixes p :: 'a::real-vector set
  shows convex hull p =
    {y. ∃ s u. finite s ∧ s ⊆ p ∧ (∀ x∈s. 0 ≤ u x) ∧ setsum u s = 1 ∧ setsum (λv.
u v *R v) s = y}
  (is ?lhs = ?rhs)
proof −
  {
    fix x
    assume x∈?lhs
    then obtain k u y where
        obt: ∀ i∈{1::nat..k}. 0 ≤ u i ∧ y i ∈ p setsum u {1..k} = 1 (∑ i = 1..k.
u i *R y i) = x
      unfolding convex-hull-indexed by auto

    have fin: finite {1..k} by auto
    have fin': ⋀v. finite {i ∈ {1..k}. y i = v} by auto
    {
      fix j
      assume j∈{1..k}
      then have y j ∈ p 0 ≤ setsum u {i. Suc 0 ≤ i ∧ i ≤ k ∧ y i = y j}
        using obt(1)[THEN bspec[where x=j]] and obt(2)
        apply simp
        apply (rule setsum-nonneg)
        using obt(1)
        apply auto
        done
    }
    moreover
    have (∑ v∈y ' {1..k}. setsum u {i ∈ {1..k}. y i = v}) = 1
      unfolding setsum-image-gen[OF fin, symmetric] using obt(2) by auto
```

```
    moreover have (∑ v∈y ' {1..k}. setsum u {i ∈ {1..k}. y i = v} *R v) = x
      using setsum-image-gen[OF fin, of λi. u i *R y i y, symmetric]
      unfolding scaleR-left.setsum using obt(3) by auto
    ultimately
    have ∃ s u. finite s ∧ s ⊆ p ∧ (∀ x∈s. 0 ≤ u x) ∧ setsum u s = 1 ∧ (∑ v∈s.
u v *R v) = x
      apply (rule-tac x=y ' {1..k} in exI)
      apply (rule-tac x=λv. setsum u {i∈{1..k}. y i = v} in exI)
      apply auto
      done
    then have x∈?rhs by auto
  }
  moreover
  {
    fix y
    assume y∈?rhs
    then obtain s u where
      obt: finite s s ⊆ p ∀ x∈s. 0 ≤ u x setsum u s = 1 (∑ v∈s. u v *R v) = y
      by auto

    obtain f where f: inj-on f {1..card s} f ' {1..card s} = s
      using ex-bij-betw-nat-finite-1[OF obt(1)] unfolding bij-betw-def by auto

    {
      fix i :: nat
      assume i∈{1..card s}
      then have f i ∈ s
        apply (subst f(2)[symmetric])
        apply auto
        done
      then have 0 ≤ u (f i) f i ∈ p using obt(2,3) by auto
    }
    moreover have *: finite {1..card s} by auto
    {
      fix y
      assume y∈s
      then obtain i where i∈{1..card s} f i = y
        using f using image-iff[of y f {1..card s}]
        by auto
      then have {x. Suc 0 ≤ x ∧ x ≤ card s ∧ f x = y} = {i}
        apply auto
        using f(1)[unfolded inj-on-def]
        apply(erule-tac x=x in ballE)
        apply auto
        done
      then have card {x. Suc 0 ≤ x ∧ x ≤ card s ∧ f x = y} = 1 by auto
      then have (∑ x∈{x ∈ {1..card s}. f x = y}. u (f x)) = u y
          (∑ x∈{x ∈ {1..card s}. f x = y}. u (f x) *R f x) = u y *R y
        by (auto simp add: setsum-constant-scaleR)
```

```
    }
    then have (∑ x = 1..card s. u (f x)) = 1 (∑ i = 1..card s. u (f i) *R f i) = y
      unfolding setsum-image-gen[OF *(1), of λx. u (f x) *R f x f]
        and setsum-image-gen[OF *(1), of λx. u (f x) f]
      unfolding f
      using setsum.cong [of s s λy. (∑ x∈{x ∈ {1..card s}. f x = y}. u (f x) *R f
x) λv. u v *R v]
      using setsum.cong [of s s λy. (∑ x∈{x ∈ {1..card s}. f x = y}. u (f x)) u]
      unfolding obt(4,5)
      by auto
    ultimately
    have ∃ k u x. (∀ i∈{1..k}. 0 ≤ u i ∧ x i ∈ p) ∧ setsum u {1..k} = 1 ∧
        (∑ i::nat = 1..k. u i *R x i) = y
      apply (rule-tac x=card s in exI)
      apply (rule-tac x=u ∘ f in exI)
      apply (rule-tac x=f in exI)
      apply fastforce
      done
    then have y ∈ ?lhs
      unfolding convex-hull-indexed by auto
  }
  ultimately show ?thesis
    unfolding set-eq-iff by blast
qed
```

19.5.5 A stepping theorem for that expansion

```
lemma convex-hull-finite-step:
  fixes s :: 'a::real-vector set
  assumes finite s
  shows
    (∃ u. (∀ x∈insert a s. 0 ≤ u x) ∧ setsum u (insert a s) = w ∧ setsum (λx. u x
*R x) (insert a s) = y)
      ⟷ (∃ v≥0. ∃ u. (∀ x∈s. 0 ≤ u x) ∧ setsum u s = w − v ∧ setsum (λx. u x
*R x) s = y − v *R a)
  (is ?lhs = ?rhs)
proof (rule, case-tac[!] a∈s)
  assume a ∈ s
  then have *: insert a s = s by auto
  assume ?lhs
  then show ?rhs
    unfolding *
    apply (rule-tac x=0 in exI)
    apply auto
    done
next
  assume ?lhs
  then obtain u where
      u: ∀ x∈insert a s. 0 ≤ u x setsum u (insert a s) = w (∑ x∈insert a s. u x *R
```

$x) = y$
by *auto*
assume $a \notin s$
then show *?rhs*
apply (*rule-tac* $x{=}u\ a$ **in** *exI*)
using $u(1)$[*THEN bspec*[**where** $x{=}a$]]
apply *simp*
apply (*rule-tac* $x{=}u$ **in** *exI*)
using u[*unfolded setsum-clauses*(2)[*OF assms*]] **and** ‹$a \notin s$›
apply *auto*
done
next
assume $a \in s$
then have $*$: *insert* $a\ s = s$ **by** *auto*
have *fin*: *finite* (*insert* $a\ s$) **using** *assms* **by** *auto*
assume *?rhs*
then obtain $v\ u$ **where** *uv*: $v \geq 0\ \forall x{\in}s.\ 0 \leq u\ x$ *setsum* $u\ s = w - v\ (\sum x{\in}s.\ u\ x *_R x) = y - v *_R a$
by *auto*
show *?lhs*
apply (*rule-tac* $x = \lambda x.\ (\textit{if}\ a = x\ \textit{then}\ v\ \textit{else}\ 0) + u\ x$ **in** *exI*)
unfolding *scaleR-left-distrib* **and** *setsum.distrib* **and** *setsum-delta''*[*OF fin*] **and** *setsum.delta'*[*OF fin*]
unfolding *setsum-clauses*(2)[*OF assms*]
using *uv* **and** *uv*(2)[*THEN bspec*[**where** $x{=}a$]] **and** ‹$a \in s$›
apply *auto*
done
next
assume *?rhs*
then obtain $v\ u$ **where**
uv: $v \geq 0\ \forall x{\in}s.\ 0 \leq u\ x$ *setsum* $u\ s = w - v\ (\sum x{\in}s.\ u\ x *_R x) = y - v *_R a$
by *auto*
moreover
assume $a \notin s$
moreover
have $(\sum x{\in}s.\ \textit{if}\ a = x\ \textit{then}\ v\ \textit{else}\ u\ x) = \textit{setsum}\ u\ s$
and $(\sum x{\in}s.\ (\textit{if}\ a = x\ \textit{then}\ v\ \textit{else}\ u\ x) *_R x) = (\sum x{\in}s.\ u\ x *_R x)$
apply (*rule-tac setsum.cong*) **apply** *rule*
defer
apply (*rule-tac setsum.cong*) **apply** *rule*
using ‹$a \notin s$›
apply *auto*
done
ultimately show *?lhs*
apply (*rule-tac* $x{=}\lambda x.\ \textit{if}\ a = x\ \textit{then}\ v\ \textit{else}\ u\ x$ **in** *exI*)
unfolding *setsum-clauses*(2)[*OF assms*]
apply *auto*
done
qed

19.5.6 Hence some special cases

lemma *convex-hull-2*:
  convex hull $\{a,b\} = \{u *_R a + v *_R b \mid u\ v.\ 0 \leq u \wedge 0 \leq v \wedge u + v = 1\}$
proof −
  have ∗: $\bigwedge u.\ (\forall x{\in}\{a, b\}.\ 0 \leq u\ x) \longleftrightarrow 0 \leq u\ a \wedge 0 \leq u\ b$
    by *auto*
  have ∗∗: *finite* $\{b\}$ **by** *auto*
  show *?thesis*
    apply (*simp add*: *convex-hull-finite*)
    unfolding *convex-hull-finite-step*[*OF* ∗∗, *of a 1*, *unfolded* ∗ *conj-assoc*]
    apply *auto*
    apply (*rule-tac x*=*v* **in** *exI*)
    apply (*rule-tac x*=*1* − *v* **in** *exI*)
    apply *simp*
    apply (*rule-tac x*=*u* **in** *exI*)
    apply *simp*
    apply (*rule-tac x*=λ*x*. *v* **in** *exI*)
    apply *simp*
    done
qed

lemma *convex-hull-2-alt*: *convex hull* $\{a,b\} = \{a + u *_R (b - a) \mid u.\ \ 0 \leq u \wedge u \leq 1\}$
  unfolding *convex-hull-2*
proof (*rule Collect-cong*)
  have ∗: $\bigwedge x\ y$ ::*real*. $x + y = 1 \longleftrightarrow x = 1 - y$
    by *auto*
  fix *x*
  show $(\exists v\ u.\ x = v *_R a + u *_R b \wedge 0 \leq v \wedge 0 \leq u \wedge v + u = 1) \longleftrightarrow$
    $(\exists u.\ x = a + u *_R (b - a) \wedge 0 \leq u \wedge u \leq 1)$
    unfolding ∗
    apply *auto*
    apply (*rule-tac*[!] *x*=*u* **in** *exI*)
    apply (*auto simp add*: *algebra-simps*)
    done
qed

lemma *convex-hull-3*:
  convex hull $\{a,b,c\} = \{\ u *_R a + v *_R b + w *_R c \mid u\ v\ w.\ 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1\}$
proof −
  have *fin*: *finite* $\{a,b,c\}$ *finite* $\{b,c\}$ *finite* $\{c\}$
    by *auto*
  have ∗: $\bigwedge x\ y\ z$ ::*real*. $x + y + z = 1 \longleftrightarrow x = 1 - y - z$
    by (*auto simp add*: *field-simps*)
  show *?thesis*
    unfolding *convex-hull-finite*[*OF fin*(*1*)] **and** *convex-hull-finite-step*[*OF fin*(*2*)]
and ∗
    unfolding *convex-hull-finite-step*[*OF fin*(*3*)]

apply (*rule Collect-cong*)
apply *simp*
apply *auto*
apply (*rule-tac x=va* **in** *exI*)
apply (*rule-tac x=u c* **in** *exI*)
apply *simp*
apply (*rule-tac x=1 − v − w* **in** *exI*)
apply *simp*
apply (*rule-tac x=v* **in** *exI*)
apply *simp*
apply (*rule-tac x=λx. w* **in** *exI*)
apply *simp*
done
qed

lemma *convex-hull-3-alt*:
convex hull {*a*,*b*,*c*} = {*a* + *u* ∗*R* (*b* − *a*) + *v* ∗*R* (*c* − *a*) | *u v*. 0 ≤ *u* ∧ 0 ≤ *v* ∧ *u* + *v* ≤ 1}
proof −
have ∗: ⋀*x y z* ::*real*. *x* + *y* + *z* = 1 ⟷ *x* = 1 − *y* − *z*
by *auto*
show *?thesis*
unfolding *convex-hull-3*
apply (*auto simp add*: ∗)
apply (*rule-tac x=v* **in** *exI*)
apply (*rule-tac x=w* **in** *exI*)
apply (*simp add*: *algebra-simps*)
apply (*rule-tac x=u* **in** *exI*)
apply (*rule-tac x=v* **in** *exI*)
apply (*simp add*: *algebra-simps*)
done
qed

19.6 Relations among closure notions and corresponding hulls

lemma *affine-imp-convex*: *affine s* ⟹ *convex s*
unfolding *affine-def convex-def* **by** *auto*

lemma *subspace-imp-convex*: *subspace s* ⟹ *convex s*
using *subspace-imp-affine affine-imp-convex* **by** *auto*

lemma *affine-hull-subset-span*: (*affine hull s*) ⊆ (*span s*)
by (*metis hull-minimal span-inc subspace-imp-affine subspace-span*)

lemma *convex-hull-subset-span*: (*convex hull s*) ⊆ (*span s*)
by (*metis hull-minimal span-inc subspace-imp-convex subspace-span*)

lemma *convex-hull-subset-affine-hull*: (*convex hull s*) ⊆ (*affine hull s*)
by (*metis affine-affine-hull affine-imp-convex hull-minimal hull-subset*)

lemma *affine-dependent-imp-dependent*: *affine-dependent s* $\Longrightarrow$ *dependent s*
unfolding *affine-dependent-def dependent-def*
using *affine-hull-subset-span* **by** *auto*

lemma *dependent-imp-affine-dependent*:
assumes *dependent* $\{x - a| \ x \ . \ x \in s\}$
and $a \notin s$
shows *affine-dependent* (*insert a s*)
proof −
from *assms*(*1*)[*unfolded dependent-explicit*] **obtain** *S u v*
where *obt*: *finite S* $S \subseteq \{x - a \ |x. \ x \in s\}$ $v \in S$ *u v* $\neq 0$ $(\sum v \in S. \ u \ v *_R \ v) = 0$
by *auto*
def $t \equiv (\lambda x. \ x + a) \ ` \ S$

have *inj*: *inj-on* $(\lambda x. \ x + a)$ *S*
unfolding *inj-on-def* **by** *auto*
have $0 \notin S$
using *obt*(*2*) *assms*(*2*) **unfolding** *subset-eq* **by** *auto*
have *fin*: *finite t* **and** $t \subseteq s$
unfolding *t-def* **using** *obt*(*1*,*2*) **by** *auto*
then have *finite* (*insert a t*) **and** *insert a t* $\subseteq$ *insert a s*
by *auto*
moreover have $*$: $\bigwedge P \ Q. \ (\sum x \in t. \ (\textit{if } x = a \textit{ then } P \ x \textit{ else } Q \ x)) = (\sum x \in t. \ Q \ x)$
apply (*rule setsum.cong*)
using ‹*a*∉*s*› ‹*t*⊆*s*›
apply *auto*
done
have $(\sum x \in \textit{insert } a \ t. \ \textit{if } x = a \textit{ then } - (\sum x \in t. \ u \ (x - a)) \textit{ else } u \ (x - a)) = 0$
unfolding *setsum-clauses*(*2*)[*OF fin*]
using ‹*a*∉*s*› ‹*t*⊆*s*›
apply *auto*
unfolding $*$
apply *auto*
done
moreover have $\exists v \in \textit{insert } a \ t. \ (\textit{if } v = a \textit{ then } - (\sum x \in t. \ u \ (x - a)) \textit{ else } u \ (v - a)) \neq 0$
apply (*rule-tac x*=*v* + *a* **in** *bexI*)
using *obt*(*3*,*4*) **and** ‹*0*∉*S*›
unfolding *t-def*
apply *auto*
done
moreover have $*$: $\bigwedge P \ Q. \ (\sum x \in t. \ (\textit{if } x = a \textit{ then } P \ x \textit{ else } Q \ x) *_R \ x) = (\sum x \in t. \ Q \ x *_R \ x)$
apply (*rule setsum.cong*)
using ‹*a*∉*s*› ‹*t*⊆*s*›

```
    apply auto
    done
  have (∑x∈t. u (x − a)) *R a = (∑v∈t. u (v − a) *R v)
    unfolding scaleR-left.setsum
    unfolding t-def and setsum.reindex[OF inj] and o-def
    using obt(5)
    by (auto simp add: setsum.distrib scaleR-right-distrib)
  then have (∑v∈insert a t. (if v = a then − (∑x∈t. u (x − a)) else u (v −
a)) *R v) = 0
    unfolding setsum-clauses(2)[OF fin]
    using ‹a∉s› ‹t⊆s›
    by (auto simp add: *)
  ultimately show ?thesis
    unfolding affine-dependent-explicit
    apply (rule-tac x=insert a t in exI)
    apply auto
    done
qed

lemma convex-cone:
  convex s ∧ cone s ⟷ (∀x∈s. ∀y∈s. (x + y) ∈ s) ∧ (∀x∈s. ∀c≥0. (c *R x) ∈
s)
  (is ?lhs = ?rhs)
proof −
  {
    fix x y
    assume x∈s y∈s and ?lhs
    then have 2 *R x ∈s 2 *R y ∈ s
      unfolding cone-def by auto
    then have x + y ∈ s
      using ‹?lhs›[unfolded convex-def, THEN conjunct1]
      apply (erule-tac x=2*R x in ballE)
      apply (erule-tac x=2*R y in ballE)
      apply (erule-tac x=1/2 in allE)
      apply simp
      apply (erule-tac x=1/2 in allE)
      apply auto
      done
  }
  then show ?thesis
    unfolding convex-def cone-def by blast
qed

lemma affine-dependent-biggerset:
  fixes s :: 'a::euclidean-space set
  assumes finite s card s ≥ DIM('a) + 2
  shows affine-dependent s
proof −
  have s ≠ {} using assms by auto
```

```
  then obtain a where a∈s by auto
  have ∗: {x − a |x. x ∈ s − {a}} = (λx. x − a) ‘ (s − {a})
    by auto
  have card {x − a |x. x ∈ s − {a}} = card (s − {a})
    unfolding ∗
    apply (rule card-image)
    unfolding inj-on-def
    apply auto
    done
  also have ... > DIM('a) using assms(2)
    unfolding card-Diff-singleton[OF assms(1) ‹a∈s›] by auto
  finally show ?thesis
    apply (subst insert-Diff[OF ‹a∈s›, symmetric])
    apply (rule dependent-imp-affine-dependent)
    apply (rule dependent-biggerset)
    apply auto
    done
qed

lemma affine-dependent-biggerset-general:
  assumes finite (s :: 'a::euclidean-space set)
    and card s ≥ dim s + 2
  shows affine-dependent s
proof −
  from assms(2) have s ≠ {} by auto
  then obtain a where a∈s by auto
  have ∗: {x − a |x. x ∈ s − {a}} = (λx. x − a) ‘ (s − {a})
    by auto
  have ∗∗: card {x − a |x. x ∈ s − {a}} = card (s − {a})
    unfolding ∗
    apply (rule card-image)
    unfolding inj-on-def
    apply auto
    done
  have dim {x − a |x. x ∈ s − {a}} ≤ dim s
    apply (rule subset-le-dim)
    unfolding subset-eq
    using ‹a∈s›
    apply (auto simp add:span-superset span-sub)
    done
  also have ... < dim s + 1 by auto
  also have ... ≤ card (s − {a})
    using assms
    using card-Diff-singleton[OF assms(1) ‹a∈s›]
    by auto
  finally show ?thesis
    apply (subst insert-Diff[OF ‹a∈s›, symmetric])
    apply (rule dependent-imp-affine-dependent)
    apply (rule dependent-biggerset-general)
```

unfolding $**$
apply *auto*
done
qed

19.7 Some Properties of Affine Dependent Sets

lemma *affine-independent-empty*: $\neg$ *affine-dependent* $\{\}$
by (*simp add*: *affine-dependent-def*)

lemma *affine-independent-sing*: $\neg$ *affine-dependent* $\{a\}$
by (*simp add*: *affine-dependent-def*)

lemma *affine-hull-translation*: *affine hull* $((\lambda x.\ a + x)\ `\ S) = (\lambda x.\ a + x)\ `$ (*affine hull* S)
proof −
have *affine* $((\lambda x.\ a + x)\ `$ (*affine hull* S))
using *affine-translation affine-affine-hull* **by** *blast*
moreover have $(\lambda x.\ a + x)\ `\ S \subseteq (\lambda x.\ a + x)\ `$ (*affine hull* S)
using *hull-subset*[*of* S] **by** *auto*
ultimately have *h1*: *affine hull* $((\lambda x.\ a + x)\ `\ S) \subseteq (\lambda x.\ a + x)\ `$ (*affine hull* S)
by (*metis hull-minimal*)
have *affine*$((\lambda x.\ -a + x)\ `$ (*affine hull* $((\lambda x.\ a + x)\ `\ S)))$
using *affine-translation affine-affine-hull* **by** *blast*
moreover have $(\lambda x.\ -a + x)\ `\ (\lambda x.\ a + x)\ `\ S \subseteq (\lambda x.\ -a + x)\ `$ (*affine hull* $((\lambda x.\ a + x)\ `\ S)$)
using *hull-subset*[*of* $(\lambda x.\ a + x)\ `\ S$] **by** *auto*
moreover have $S = (\lambda x.\ -a + x)\ `\ (\lambda x.\ a + x)\ `\ S$
using *translation-assoc*[*of* $-a\ a$] **by** *auto*
ultimately have $(\lambda x.\ -a + x)\ `$ (*affine hull* $((\lambda x.\ a + x)\ `\ S)) >=$ (*affine hull* S)
by (*metis hull-minimal*)
then have *affine hull* $((\lambda x.\ a + x)\ `\ S) >= (\lambda x.\ a + x)\ `$ (*affine hull* S)
by *auto*
then show *?thesis* **using** *h1* **by** *auto*
qed

lemma *affine-dependent-translation*:
assumes *affine-dependent* S
shows *affine-dependent* $((\lambda x.\ a + x)\ `\ S)$
proof −
obtain x **where** x: $x \in S \wedge x \in$ *affine hull* $(S - \{x\})$
using *assms affine-dependent-def* **by** *auto*
have $op + a\ `\ (S - \{x\}) = op + a\ `\ S - \{a + x\}$
by *auto*
then have $a + x \in$ *affine hull* $((\lambda x.\ a + x)\ `\ S - \{a + x\})$
using *affine-hull-translation*[*of* $a\ S - \{x\}$] x **by** *auto*
moreover have $a + x \in (\lambda x.\ a + x)\ `\ S$

using *x* **by** *auto*
ultimately show *?thesis*
unfolding *affine-dependent-def* **by** *auto*
qed

lemma *affine-dependent-translation-eq*:
affine-dependent S $\longleftrightarrow$ *affine-dependent* $((\lambda x.\ a + x)\ `\ S)$
proof −
{
assume *affine-dependent* $((\lambda x.\ a + x)\ `\ S)$
then have *affine-dependent S*
using *affine-dependent-translation*[*of* $((\lambda x.\ a + x)\ `\ S)\ -a$] *translation-assoc*[*of* $-a\ a$]
by *auto*
}
then show *?thesis*
using *affine-dependent-translation* **by** *auto*
qed

lemma *affine-hull-0-dependent*:
assumes $0 \in$ *affine hull S*
shows *dependent S*
proof −
obtain *s u* **where** *s-u*: *finite s* $\wedge$ $s \neq \{\}$ $\wedge$ $s \subseteq S$ $\wedge$ *setsum u s* $= 1$ $\wedge$ $(\sum v{\in}s.\ u\ v *_R v) = 0$
using *assms affine-hull-explicit*[*of S*] **by** *auto*
then have $\exists\, v{\in}s.\ u\ v \neq 0$
using *setsum-not-0*[*of u s*] **by** *auto*
then have *finite s* $\wedge$ $s \subseteq S$ $\wedge$ $(\exists\, v{\in}s.\ u\ v \neq 0 \wedge (\sum v{\in}s.\ u\ v *_R v) = 0)$
using *s-u* **by** *auto*
then show *?thesis*
unfolding *dependent-explicit*[*of S*] **by** *auto*
qed

lemma *affine-dependent-imp-dependent2*:
assumes *affine-dependent* (*insert 0 S*)
shows *dependent S*
proof −
obtain *x* **where** *x*: $x \in$ *insert 0 S* $\wedge$ $x \in$ *affine hull* $(\text{insert } 0\ S - \{x\})$
using *affine-dependent-def*[*of* (*insert 0 S*)] *assms* **by** *blast*
then have $x \in$ *span* $(\text{insert } 0\ S - \{x\})$
using *affine-hull-subset-span* **by** *auto*
moreover have *span* $(\text{insert } 0\ S - \{x\}) =$ *span* $(S - \{x\})$
using *insert-Diff-if*[*of 0 S* $\{x\}$] *span-insert-0*[*of S*$-\{x\}$] **by** *auto*
ultimately have $x \in$ *span* $(S - \{x\})$ **by** *auto*
then have $x \neq 0 \Longrightarrow$ *dependent S*
using *x dependent-def* **by** *auto*
moreover
{

```
    assume x = 0
    then have 0 ∈ affine hull S
      using x hull-mono[of S − {0} S] by auto
    then have dependent S
      using affine-hull-0-dependent by auto
  }
  ultimately show ?thesis by auto
qed

lemma affine-dependent-iff-dependent:
  assumes a ∉ S
  shows affine-dependent (insert a S) ⟷ dependent ((λx. −a + x) ‘ S)
proof −
  have (op + (− a) ‘ S) = {x − a| x . x : S} by auto
  then show ?thesis
    using affine-dependent-translation-eq[of (insert a S) −a]
      affine-dependent-imp-dependent2 assms
      dependent-imp-affine-dependent[of a S]
    by (auto simp del: uminus-add-conv-diff)
qed

lemma affine-dependent-iff-dependent2:
  assumes a ∈ S
  shows affine-dependent S ⟷ dependent ((λx. −a + x) ‘ (S−{a}))
proof −
  have insert a (S − {a}) = S
    using assms by auto
  then show ?thesis
    using assms affine-dependent-iff-dependent[of a S−{a}] by auto
qed

lemma affine-hull-insert-span-gen:
  affine hull (insert a s) = (λx. a + x) ‘ span ((λx. − a + x) ‘ s)
proof −
  have h1: {x − a |x. x ∈ s} = ((λx. −a+x) ‘ s)
    by auto
  {
    assume a ∉ s
    then have ?thesis
      using affine-hull-insert-span[of a s] h1 by auto
  }
  moreover
  {
    assume a1: a ∈ s
    have ∃x. x ∈ s ∧ −a+x=0
      apply (rule exI[of - a])
      using a1
      apply auto
      done
```

then have *insert 0* $((\lambda x.\ -a+x)\ `\ (s - \{a\})) = (\lambda x.\ -a+x)\ `\ s$
by *auto*
then have *span* $((\lambda x.\ -a+x)\ `\ (s - \{a\}))$=*span* $((\lambda x.\ -a+x)\ `\ s)$
using *span-insert-0*[*of op* + $(-\ a)\ `\ (s - \{a\})$] **by** (*auto simp del*: *uminus-add-conv-diff*)
moreover have $\{x - a\ |x.\ x \in (s - \{a\})\} = ((\lambda x.\ -a+x)\ `\ (s - \{a\}))$
by *auto*
moreover have *insert a* $(s - \{a\}) =$ *insert a s*
using *assms* **by** *auto*
ultimately have *?thesis*
using *assms affine-hull-insert-span*[*of a s*−{*a*}] **by** *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *affine-hull-span2*:
assumes $a \in s$
shows *affine hull s* = $(\lambda x.\ a+x)\ `$ *span* $((\lambda x.\ -a+x)\ `\ (s-\{a\}))$
using *affine-hull-insert-span-gen*[*of a s* − {*a*}, *unfolded insert-Diff*[*OF assms*]]
by *auto*

lemma *affine-hull-span-gen*:
assumes $a \in$ *affine hull s*
shows *affine hull s* = $(\lambda x.\ a+x)\ `$ *span* $((\lambda x.\ -a+x)\ `\ s)$
proof −
have *affine hull* (*insert a s*) = *affine hull s*
using *hull-redundant*[*of a affine s*] *assms* **by** *auto*
then show *?thesis*
using *affine-hull-insert-span-gen*[*of a s*] **by** *auto*
qed

lemma *affine-hull-span-0*:
assumes $0 \in$ *affine hull S*
shows *affine hull S* = *span S*
using *affine-hull-span-gen*[*of 0 S*] *assms* **by** *auto*

lemma *extend-to-affine-basis*:
fixes *S V* :: *'n*::*euclidean-space set*
assumes ¬ *affine-dependent S S* ⊆ *V S* ≠ {}
shows ∃ *T*. ¬ *affine-dependent T* ∧ *S* ⊆ *T* ∧ *T* ⊆ *V* ∧ *affine hull T* = *affine hull V*
proof −
obtain *a* **where** *a*: $a \in S$
using *assms* **by** *auto*
then have *h0*: *independent* $((\lambda x.\ -a\ +\ x)\ `\ (S-\{a\}))$
using *affine-dependent-iff-dependent2 assms* **by** *auto*
then obtain *B* **where** *B*:
$(\lambda x.\ -a+x)\ `\ (S - \{a\}) \subseteq B \wedge B \subseteq (\lambda x.\ -a+x)\ `\ V \wedge$ *independent B* ∧ $(\lambda x.\ -a+x)\ `\ V \subseteq$ *span B*

using *maximal-independent-subset-extend*[*of* (λ*x*. −*a*+*x*) ‘ (*S*−{*a*}) (λ*x*. −*a* + *x*) ‘ *V*] *assms*
by *blast*
def *T* ≡ (λ*x*. *a*+*x*) ‘ *insert 0 B*
then have *T* = *insert a* ((λ*x*. *a*+*x*) ‘ *B*)
by *auto*
then have *affine hull T* = (λ*x*. *a*+*x*) ‘ *span B*
using *affine-hull-insert-span-gen*[*of a* ((λ*x*. *a*+*x*) ‘ *B*)] *translation-assoc*[*of* −*a a B*]
by *auto*
then have *V* ⊆ *affine hull T*
using *B assms translation-inverse-subset*[*of a V span B*]
by *auto*
moreover have *T* ⊆ *V*
using *T-def B a assms* **by** *auto*
ultimately have *affine hull T* = *affine hull V*
by (*metis Int-absorb1 Int-absorb2 hull-hull hull-mono*)
moreover have *S* ⊆ *T*
using *T-def B translation-inverse-subset*[*of a S*−{*a*} *B*]
by *auto*
moreover have ¬ *affine-dependent T*
using *T-def affine-dependent-translation-eq*[*of insert 0 B*]
affine-dependent-imp-dependent2 B
by *auto*
ultimately show *?thesis* **using** ‹*T* ⊆ *V*› **by** *auto*
qed

lemma *affine-basis-exists*:
fixes *V* :: *'n*::*euclidean-space set*
shows ∃ *B*. *B* ⊆ *V* ∧ ¬ *affine-dependent B* ∧ *affine hull V* = *affine hull B*
proof (*cases V* = {})
case *True*
then show *?thesis*
using *affine-independent-empty* **by** *auto*
next
case *False*
then obtain *x* **where** *x* ∈ *V* **by** *auto*
then show *?thesis*
using *affine-dependent-def*[*of* {*x*}] *extend-to-affine-basis*[*of* {*x*} *V*]
by *auto*
qed

19.8 Affine Dimension of a Set

definition *aff-dim* :: (*'a*::*euclidean-space*) *set* ⇒ *int*
where *aff-dim V* =
(*SOME d* :: *int*.
∃ *B*. *affine hull B* = *affine hull V* ∧ ¬ *affine-dependent B* ∧ *of-nat* (*card B*) = *d* + *1*)

lemma *aff-dim-basis-exists*:
fixes $V :: ('n::euclidean\text{-}space)\ set$
shows $\exists B.\ affine\ hull\ B = affine\ hull\ V \wedge \neg\ affine\text{-}dependent\ B \wedge of\text{-}nat\ (card\ B) = aff\text{-}dim\ V + 1$
proof −
obtain B **where** $\neg\ affine\text{-}dependent\ B \wedge affine\ hull\ B = affine\ hull\ V$
using *affine-basis-exists*[*of* V] **by** *auto*
then show *?thesis*
unfolding *aff-dim-def*
some-eq-ex[*of* $\lambda d.\ \exists B.\ affine\ hull\ B = affine\ hull\ V \wedge \neg\ affine\text{-}dependent\ B \wedge of\text{-}nat\ (card\ B) = d + 1$]
apply *auto*
apply (*rule exI*[*of* - *int* (*card* B) − (*1* :: *int*)])
apply (*rule exI*[*of* - B])
apply *auto*
done
qed

lemma *affine-hull-nonempty*: $S \neq \{\} \longleftrightarrow affine\ hull\ S \neq \{\}$
proof −
have $S = \{\} \Longrightarrow affine\ hull\ S = \{\}$
using *affine-hull-empty* **by** *auto*
moreover have $affine\ hull\ S = \{\} \Longrightarrow S = \{\}$
unfolding *hull-def* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *aff-dim-parallel-subspace-aux*:
fixes $B :: 'n::euclidean\text{-}space\ set$
assumes $\neg\ affine\text{-}dependent\ B\ a \in B$
shows $finite\ B \wedge ((card\ B) - 1 = dim\ (span\ ((\lambda x.\ -a+x)\ `\ (B-\{a\}))))$
proof −
have $independent\ ((\lambda x.\ -a + x)\ `\ (B-\{a\}))$
using *affine-dependent-iff-dependent2 assms* **by** *auto*
then have *fin*: $dim\ (span\ ((\lambda x.\ -a+x)\ `\ (B-\{a\}))) = card\ ((\lambda x.\ -a + x)\ `\ (B-\{a\}))$
$finite\ ((\lambda x.\ -a + x)\ `\ (B - \{a\}))$
using *indep-card-eq-dim-span*[*of* $(\lambda x.\ -a+x)\ `\ (B-\{a\})$] **by** *auto*
show *?thesis*
proof (*cases* $(\lambda x.\ -a + x)\ `\ (B - \{a\}) = \{\}$)
case *True*
have $B = insert\ a\ ((\lambda x.\ a + x)\ `\ (\lambda x.\ -a + x)\ `\ (B - \{a\}))$
using *translation-assoc*[*of* $a\ -a\ (B - \{a\})$] *assms* **by** *auto*
then have $B = \{a\}$ **using** *True* **by** *auto*
then show *?thesis* **using** *assms fin* **by** *auto*
next
case *False*
then have $card\ ((\lambda x.\ -a + x)\ `\ (B - \{a\})) > 0$

```
    using fin by auto
  moreover have h1: card ((λx. −a + x) ‘ (B−{a})) = card (B−{a})
    apply (rule card-image)
    using translate-inj-on
    apply (auto simp del: uminus-add-conv-diff)
    done
  ultimately have card (B−{a}) > 0 by auto
  then have ∗: finite (B − {a})
    using card-gt-0-iff[of (B − {a})] by auto
  then have card (B − {a}) = card B − 1
    using card-Diff-singleton assms by auto
  with ∗ show ?thesis using fin h1 by auto
 qed
qed

lemma aff-dim-parallel-subspace:
 fixes V L :: 'n::euclidean-space set
 assumes V ≠ {}
  and subspace L
  and affine-parallel (affine hull V) L
 shows aff-dim V = int (dim L)
proof −
 obtain B where
   B: affine hull B = affine hull V ∧ ¬ affine-dependent B ∧ int (card B) =
aff-dim V + 1
  using aff-dim-basis-exists by auto
 then have B ≠ {}
  using assms B affine-hull-nonempty[of V] affine-hull-nonempty[of B]
  by auto
 then obtain a where a: a ∈ B by auto
 def Lb ≡ span ((λx. −a+x) ‘ (B−{a}))
 moreover have affine-parallel (affine hull B) Lb
  using Lb-def B assms affine-hull-span2[of a B] a
   affine-parallel-commut[of Lb (affine hull B)]
  unfolding affine-parallel-def
  by auto
 moreover have subspace Lb
  using Lb-def subspace-span by auto
 moreover have affine hull B ≠ {}
  using assms B affine-hull-nonempty[of V] by auto
 ultimately have L = Lb
  using assms affine-parallel-subspace[of affine hull B] affine-affine-hull[of B] B
  by auto
 then have dim L = dim Lb
  by auto
 moreover have card B − 1 = dim Lb and finite B
  using Lb-def aff-dim-parallel-subspace-aux a B by auto
 ultimately show ?thesis
  using B ‹B ≠ {}› card-gt-0-iff[of B] by auto
```

qed

lemma *aff-independent-finite*:
fixes B :: *'n::euclidean-space set*
assumes ¬ *affine-dependent B*
shows *finite B*
proof −
{
assume $B \neq \{\}$
then obtain a **where** $a \in B$ **by** *auto*
then have *?thesis*
using *aff-dim-parallel-subspace-aux assms* **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *independent-finite*:
fixes B :: *'n::euclidean-space set*
assumes *independent B*
shows *finite B*
using *affine-dependent-imp-dependent*[*of B*] *aff-independent-finite*[*of B*] *assms*
by *auto*

lemma *subspace-dim-equal*:
assumes *subspace* (S :: (*'n::euclidean-space*) *set*)
and *subspace T*
and $S \subseteq T$
and *dim S* ≥ *dim T*
shows $S = T$
proof −
obtain B **where** B: $B \leq S$ *independent B* ∧ $S \subseteq$ *span B card B* = *dim S*
using *basis-exists*[*of S*] **by** *auto*
then have *span B* ⊆ *S*
using *span-mono*[*of B S*] *span-eq*[*of S*] *assms* **by** *metis*
then have *span B* = *S*
using *B* **by** *auto*
have *dim S* = *dim T*
using *assms dim-subset*[*of S T*] **by** *auto*
then have *T* ⊆ *span B*
using *card-eq-dim*[*of B T*] *B independent-finite assms* **by** *auto*
then show *?thesis*
using *assms* ‹*span B* = *S*› **by** *auto*
qed

lemma *span-substd-basis*:
assumes *d*: *d* ⊆ *Basis*
shows *span d* = {*x*. ∀ *i*∈*Basis*. *i* ∉ *d* ⟶ *x*•*i* = *0*}
(**is** - = *?B*)
proof −

have $d \subseteq ?B$
using d **by** (*auto simp*: *inner-Basis*)
moreover have s: *subspace ?B*
using *subspace-substandard*[*of* $\lambda i.\ i \notin d$] .
ultimately have *span* $d \subseteq ?B$
using *span-mono*[*of* d *?B*] *span-eq*[*of ?B*] **by** *blast*
moreover have $*$: *card* $d \leq$ *dim* (*span* d)
using *independent-card-le-dim*[*of* d *span* d] *independent-substdbasis*[*OF assms*] *span-inc*[*of* d]
by *auto*
moreover from $*$ **have** *dim ?B* $\leq$ *dim* (*span* d)
using *dim-substandard*[*OF assms*] **by** *auto*
ultimately show *?thesis*
using *s subspace-dim-equal*[*of span d ?B*] *subspace-span*[*of* d] **by** *auto*
qed

lemma *basis-to-substdbasis-subspace-isomorphism*:
fixes $B :: 'a$::*euclidean-space set*
assumes *independent B*
shows $\exists f\ d{::}'a$ *set*. *card* d = *card B* $\wedge$ *linear* f $\wedge$ f ‘ $B = d \wedge$
f ‘ *span B* = $\{x.\ \forall i{\in}$*Basis*. $i \notin d \longrightarrow x \cdot i = 0\}$ $\wedge$ *inj-on* f (*span B*) $\wedge$ $d \subseteq$ *Basis*
proof −
have *B*: *card B* = *dim B*
using *dim-unique*[*of B B card B*] *assms span-inc*[*of B*] **by** *auto*
have *dim B* $\leq$ *card* (*Basis* :: $'a$ *set*)
using *dim-subset-UNIV*[*of B*] **by** *simp*
from *ex-card*[*OF this*] **obtain** d :: $'a$ *set* **where** d: $d \subseteq$ *Basis* **and** t: *card* d = *dim B*
by *auto*
let *?t* = $\{x{::}'a{::}$*euclidean-space*. $\forall i{\in}$*Basis*. $i \notin d \longrightarrow x{\cdot}i = 0\}$
have $\exists f$. *linear* f $\wedge$ f ‘ *B* = $d \wedge$ f ‘ *span B* = *?t* $\wedge$ *inj-on* f (*span B*)
apply (*rule basis-to-basis-subspace-isomorphism*[*of span B ?t B d*])
apply (*rule subspace-span*)
apply (*rule subspace-substandard*)
defer
apply (*rule span-inc*)
apply (*rule assms*)
defer
unfolding *dim-span*[*of B*]
apply(*rule B*)
unfolding *span-substd-basis*[*OF d*, *symmetric*]
apply (*rule span-inc*)
apply (*rule independent-substdbasis*[*OF d*])
apply *rule*
apply *assumption*
unfolding *t*[*symmetric*] *span-substd-basis*[*OF d*] *dim-substandard*[*OF d*]
apply *auto*
done

```
  with t ‹card B = dim B› d show ?thesis by auto
qed

lemma aff-dim-empty:
  fixes S :: 'n::euclidean-space set
  shows S = {} ⟷ aff-dim S = -1
proof -
  obtain B where *: affine hull B = affine hull S
    and ¬ affine-dependent B
    and int (card B) = aff-dim S + 1
    using aff-dim-basis-exists by auto
  moreover
  from * have S = {} ⟷ B = {}
    using affine-hull-nonempty[of B] affine-hull-nonempty[of S] by auto
  ultimately show ?thesis
    using aff-independent-finite[of B] card-gt-0-iff[of B] by auto
qed

lemma aff-dim-empty-eq [simp]: aff-dim ({}::'a::euclidean-space set) = -1
  by (simp add: aff-dim-empty [symmetric])

lemma aff-dim-affine-hull: aff-dim (affine hull S) = aff-dim S
  unfolding aff-dim-def using hull-hull[of - S] by auto

lemma aff-dim-affine-hull2:
  assumes affine hull S = affine hull T
  shows aff-dim S = aff-dim T
  unfolding aff-dim-def using assms by auto

lemma aff-dim-unique:
  fixes B V :: 'n::euclidean-space set
  assumes affine hull B = affine hull V ∧ ¬ affine-dependent B
  shows of-nat (card B) = aff-dim V + 1
proof (cases B = {})
  case True
  then have V = {}
    using affine-hull-nonempty[of V] affine-hull-nonempty[of B] assms
    by auto
  then have aff-dim V = (-1::int)
    using aff-dim-empty by auto
  then show ?thesis
    using ‹B = {}› by auto
next
  case False
  then obtain a where a: a ∈ B by auto
  def Lb ≡ span ((λx. -a+x) ‘ (B-{a}))
  have affine-parallel (affine hull B) Lb
    using Lb-def affine-hull-span2[of a B] a
      affine-parallel-commut[of Lb (affine hull B)]
```

unfolding *affine-parallel-def* **by** *auto*
moreover have *subspace Lb*
using *Lb-def subspace-span* **by** *auto*
ultimately have *aff-dim B = int(dim Lb)*
using *aff-dim-parallel-subspace[of B Lb]* ‹*B ≠ {}*› **by** *auto*
moreover have *(card B) − 1 = dim Lb finite B*
using *Lb-def aff-dim-parallel-subspace-aux a assms* **by** *auto*
ultimately have *of-nat (card B) = aff-dim B + 1*
using ‹*B ≠ {}*› *card-gt-0-iff[of B]* **by** *auto*
then show *?thesis*
using *aff-dim-affine-hull2 assms* **by** *auto*
qed

lemma *aff-dim-affine-independent*:
fixes *B :: 'n::euclidean-space set*
assumes ¬ *affine-dependent B*
shows *of-nat (card B) = aff-dim B + 1*
using *aff-dim-unique[of B B] assms* **by** *auto*

lemma *affine-independent-iff-card*:
fixes *s :: 'a::euclidean-space set*
shows ~ *affine-dependent s ⟷ finite s ∧ aff-dim s = int(card s) − 1*
apply (*rule iffI*)
apply (*simp add*: *aff-dim-affine-independent aff-independent-finite*)
by (*metis affine-basis-exists [of s] aff-dim-unique card-subset-eq diff-add-cancel of-nat-eq-iff*)

lemma *aff-dim-sing*:
fixes *a :: 'n::euclidean-space*
shows *aff-dim {a} = 0*
using *aff-dim-affine-independent[of {a}] affine-independent-sing* **by** *auto*

lemma *aff-dim-inner-basis-exists*:
fixes *V :: ('n::euclidean-space) set*
shows ∃*B*. *B ⊆ V ∧ affine hull B = affine hull V ∧*
¬ *affine-dependent B ∧ of-nat (card B) = aff-dim V + 1*
proof −
obtain *B* **where** *B*: ¬ *affine-dependent B B ⊆ V affine hull B = affine hull V*
using *affine-basis-exists[of V]* **by** *auto*
then have *of-nat(card B) = aff-dim V+1* **using** *aff-dim-unique* **by** *auto*
with *B* **show** *?thesis* **by** *auto*
qed

lemma *aff-dim-le-card*:
fixes *V :: 'n::euclidean-space set*
assumes *finite V*
shows *aff-dim V ≤ of-nat (card V) − 1*
proof −
obtain *B* **where** *B*: *B ⊆ V of-nat (card B) = aff-dim V + 1*

```
    using aff-dim-inner-basis-exists[of V] by auto
  then have card B ≤ card V
    using assms card-mono by auto
  with B show ?thesis by auto
qed

lemma aff-dim-parallel-eq:
  fixes S T :: 'n::euclidean-space set
  assumes affine-parallel (affine hull S) (affine hull T)
  shows aff-dim S = aff-dim T
proof −
  {
    assume T ≠ {} S ≠ {}
    then obtain L where L: subspace L ∧ affine-parallel (affine hull T) L
      using affine-parallel-subspace[of affine hull T]
        affine-affine-hull[of T] affine-hull-nonempty
      by auto
    then have aff-dim T = int (dim L)
      using aff-dim-parallel-subspace ‹T ≠ {}› by auto
    moreover have ∗: subspace L ∧ affine-parallel (affine hull S) L
       using L affine-parallel-assoc[of affine hull S affine hull T L] assms by auto
    moreover from ∗ have aff-dim S = int (dim L)
      using aff-dim-parallel-subspace ‹S ≠ {}› by auto
    ultimately have ?thesis by auto
  }
  moreover
  {
    assume S = {}
    then have S = {} and T = {}
      using assms affine-hull-nonempty
      unfolding affine-parallel-def
      by auto
    then have ?thesis using aff-dim-empty by auto
  }
  moreover
  {
    assume T = {}
    then have S = {} and T = {}
      using assms affine-hull-nonempty
      unfolding affine-parallel-def
      by auto
    then have ?thesis
      using aff-dim-empty by auto
  }
  ultimately show ?thesis by blast
qed

lemma aff-dim-translation-eq:
  fixes a :: 'n::euclidean-space
```

shows *aff-dim* ((λ*x*. *a* + *x*) ' *S*) = *aff-dim S*
proof −
have *affine-parallel* (*affine hull S*) (*affine hull* ((λ*x*. *a* + *x*) ' *S*))
unfolding *affine-parallel-def*
apply (*rule exI*[*of* - *a*])
using *affine-hull-translation*[*of a S*]
apply *auto*
done
then show *?thesis*
using *aff-dim-parallel-eq*[*of S* (λ*x*. *a* + *x*) ' *S*] **by** *auto*
qed

lemma *aff-dim-affine*:
fixes *S L* :: '*n*::*euclidean-space set*
assumes *S* ≠ {}
and *affine S*
and *subspace L*
and *affine-parallel S L*
shows *aff-dim S* = *int* (*dim L*)
proof −
have ∗: *affine hull S* = *S*
using *assms affine-hull-eq*[*of S*] **by** *auto*
then have *affine-parallel* (*affine hull S*) *L*
using *assms* **by** (*simp add*: ∗)
then show *?thesis*
using *assms aff-dim-parallel-subspace*[*of S L*] **by** *blast*
qed

lemma *dim-affine-hull*:
fixes *S* :: '*n*::*euclidean-space set*
shows *dim* (*affine hull S*) = *dim S*
proof −
have *dim* (*affine hull S*) ≥ *dim S*
using *dim-subset* **by** *auto*
moreover have *dim* (*span S*) ≥ *dim* (*affine hull S*)
using *dim-subset affine-hull-subset-span* **by** *blast*
moreover have *dim* (*span S*) = *dim S*
using *dim-span* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *aff-dim-subspace*:
fixes *S* :: '*n*::*euclidean-space set*
assumes *S* ≠ {}
and *subspace S*
shows *aff-dim S* = *int* (*dim S*)
using *aff-dim-affine*[*of S S*] *assms subspace-imp-affine*[*of S*] *affine-parallel-reflex*[*of S*]
by *auto*

lemma *aff-dim-zero*:
fixes S :: *'n::euclidean-space set*
assumes $0 \in$ *affine hull* S
shows *aff-dim* S = *int* (*dim* S)
proof −
have *subspace* (*affine hull* S)
using *subspace-affine*[*of affine hull* S] *affine-affine-hull assms*
by *auto*
then have *aff-dim* (*affine hull* S) = *int* (*dim* (*affine hull* S))
using *assms aff-dim-subspace*[*of affine hull* S] **by** *auto*
then show *?thesis*
using *aff-dim-affine-hull*[*of* S] *dim-affine-hull*[*of* S]
by *auto*
qed

lemma *aff-dim-univ*: *aff-dim* (*UNIV* :: *'n::euclidean-space set*) = *int*(*DIM*(*'n*))
using *aff-dim-subspace*[*of* (*UNIV* :: *'n::euclidean-space set*)]
dim-UNIV[**where** *'a='n::euclidean-space*]
by *auto*

lemma *aff-dim-geq*:
fixes V :: *'n::euclidean-space set*
shows *aff-dim* $V \geq -1$
proof −
obtain B **where** *affine hull* B = *affine hull* V
and ¬ *affine-dependent* B
and *int* (*card* B) = *aff-dim* V + *1*
using *aff-dim-basis-exists* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *independent-card-le-aff-dim*:
fixes B :: *'n::euclidean-space set*
assumes $B \subseteq V$
assumes ¬ *affine-dependent* B
shows *int* (*card* B) ≤ *aff-dim* V + *1*
proof (*cases* $B = \{\}$)
case *True*
then have $-1 \leq$ *aff-dim* V
using *aff-dim-geq* **by** *auto*
with *True* **show** *?thesis* **by** *auto*
next
case *False*
then obtain T **where** T: ¬ *affine-dependent* $T \wedge B \subseteq T \wedge T \subseteq V \wedge$ *affine hull* T = *affine hull* V
using *assms extend-to-affine-basis*[*of* B V] **by** *auto*
then have *of-nat* (*card* T) = *aff-dim* V + *1*
using *aff-dim-unique* **by** *auto*

then show *?thesis*
using *T card-mono*[*of T B*] *aff-independent-finite*[*of T*] **by** *auto*
qed

lemma *aff-dim-subset*:
fixes *S T* :: *'n*::*euclidean-space set*
assumes $S \subseteq T$
shows *aff-dim S* ≤ *aff-dim T*
proof −
obtain *B* **where** *B*: ¬ *affine-dependent B B* ⊆ *S affine hull B* = *affine hull S*
of-nat (*card B*) = *aff-dim S* + *1*
using *aff-dim-inner-basis-exists*[*of S*] **by** *auto*
then have *int* (*card B*) ≤ *aff-dim T* + *1*
using *assms independent-card-le-aff-dim*[*of B T*] **by** *auto*
with *B* **show** *?thesis* **by** *auto*
qed

lemma *aff-dim-subset-univ*:
fixes *S* :: *'n*::*euclidean-space set*
shows *aff-dim S* ≤ *int* (*DIM*(*'n*))
proof −
have *aff-dim* (*UNIV* :: *'n*::*euclidean-space set*) = *int*(*DIM*(*'n*))
using *aff-dim-univ* **by** *auto*
then show *aff-dim* (*S*:: *'n*::*euclidean-space set*) ≤ *int*(*DIM*(*'n*))
using *assms aff-dim-subset*[*of S* (*UNIV* :: (*'n*::*euclidean-space*) *set*)] *subset-UNIV*
by *auto*
qed

lemma *affine-dim-equal*:
fixes *S* :: *'n*::*euclidean-space set*
assumes *affine S affine T S* ≠ {} *S* ⊆ *T aff-dim S* = *aff-dim T*
shows *S* = *T*
proof −
obtain *a* **where** *a* ∈ *S* **using** *assms* **by** *auto*
then have *a* ∈ *T* **using** *assms* **by** *auto*
def *LS* ≡ {*y*. ∃ *x* ∈ *S*. (−*a*) + *x* = *y*}
then have *ls*: *subspace LS affine-parallel S LS*
using *assms parallel-subspace-explicit*[*of S a LS*] ‹*a* ∈ *S*› **by** *auto*
then have *h1*: *int*(*dim LS*) = *aff-dim S*
using *assms aff-dim-affine*[*of S LS*] **by** *auto*
have *T* ≠ {} **using** *assms* **by** *auto*
def *LT* ≡ {*y*. ∃ *x* ∈ *T*. (−*a*) + *x* = *y*}
then have *lt*: *subspace LT* ∧ *affine-parallel T LT*
using *assms parallel-subspace-explicit*[*of T a LT*] ‹*a* ∈ *T*› **by** *auto*
then have *int*(*dim LT*) = *aff-dim T*
using *assms aff-dim-affine*[*of T LT*] ‹*T* ≠ {}› **by** *auto*
then have *dim LS* = *dim LT*
using *h1 assms* **by** *auto*
moreover have *LS* ≤ *LT*

using *LS-def LT-def assms* **by** *auto*
ultimately have $LS = LT$
using *subspace-dim-equal*[*of LS LT*] *ls lt* **by** *auto*
moreover have $S = \{x.\ \exists\, y \in LS.\ a+y=x\}$
using *LS-def* **by** *auto*
moreover have $T = \{x.\ \exists\, y \in LT.\ a+y=x\}$
using *LT-def* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *affine-hull-univ*:
fixes $S :: 'n::euclidean\text{-}space\ set$
assumes *aff-dim* $S = int(DIM('n))$
shows *affine hull* $S = (UNIV :: ('n::euclidean\text{-}space)\ set)$
proof −
have $S \neq \{\}$
using *assms aff-dim-empty*[*of S*] **by** *auto*
have *h0*: $S \subseteq$ *affine hull S*
using *hull-subset*[*of S* -] **by** *auto*
have *h1*: *aff-dim* $(UNIV :: ('n::euclidean\text{-}space)\ set) =$ *aff-dim S*
using *aff-dim-univ assms* **by** *auto*
then have *h2*: *aff-dim* (*affine hull S*) ≤ *aff-dim* $(UNIV :: ('n::euclidean\text{-}space)\ set)$
using *aff-dim-subset-univ*[*of affine hull S*] *assms h0* **by** *auto*
have *h3*: *aff-dim S* ≤ *aff-dim* (*affine hull S*)
using *h0 aff-dim-subset*[*of S affine hull S*] *assms* **by** *auto*
then have *h4*: *aff-dim* (*affine hull S*) = *aff-dim* $(UNIV :: ('n::euclidean\text{-}space)\ set)$
using *h0 h1 h2* **by** *auto*
then show *?thesis*
using *affine-dim-equal*[*of affine hull S* $(UNIV :: ('n::euclidean\text{-}space)\ set)$]
affine-affine-hull[*of S*] *affine-UNIV assms h4 h0* ‹$S \neq \{\}$›
by *auto*
qed

lemma *aff-dim-convex-hull*:
fixes $S :: 'n::euclidean\text{-}space\ set$
shows *aff-dim* (*convex hull S*) = *aff-dim S*
using *aff-dim-affine-hull*[*of S*] *convex-hull-subset-affine-hull*[*of S*]
hull-subset[*of S convex*] *aff-dim-subset*[*of S convex hull S*]
aff-dim-subset[*of convex hull S affine hull S*]
by *auto*

lemma *aff-dim-cball*:
fixes $a :: 'n::euclidean\text{-}space$
assumes $e > 0$
shows *aff-dim* (*cball a e*) = *int* $(DIM('n))$
proof −
have $(\lambda x.\ a + x)$ ‘ (*cball 0 e*) ⊆ *cball a e*

unfolding *cball-def dist-norm* **by** *auto*
then have *aff-dim* (*cball* (*0* :: *'n*::*euclidean-space*) *e*) ≤ *aff-dim* (*cball a e*)
using *aff-dim-translation-eq*[*of a cball 0 e*]
aff-dim-subset[*of op + a ' cball 0 e cball a e*]
by *auto*
moreover have *aff-dim* (*cball* (*0* :: *'n*::*euclidean-space*) *e*) = *int* (*DIM*(*'n*))
using *hull-inc*[*of* (*0* :: *'n*::*euclidean-space*) *cball 0 e*]
centre-in-cball[*of* (*0* :: *'n*::*euclidean-space*)] *assms*
by (*simp add*: *dim-cball*[*of e*] *aff-dim-zero*[*of cball 0 e*])
ultimately show *?thesis*
using *aff-dim-subset-univ*[*of cball a e*] **by** *auto*
qed

lemma *aff-dim-open*:
fixes *S* :: *'n*::*euclidean-space set*
assumes *open S*
and *S* ≠ {}
shows *aff-dim S* = *int* (*DIM*(*'n*))
proof −
obtain *x* **where** *x* ∈ *S*
using *assms* **by** *auto*
then obtain *e* **where** *e*: *e* > *0 cball x e* ⊆ *S*
using *open-contains-cball*[*of S*] *assms* **by** *auto*
then have *aff-dim* (*cball x e*) ≤ *aff-dim S*
using *aff-dim-subset* **by** *auto*
with *e* **show** *?thesis*
using *aff-dim-cball*[*of e x*] *aff-dim-subset-univ*[*of S*] **by** *auto*
qed

lemma *low-dim-interior*:
fixes *S* :: *'n*::*euclidean-space set*
assumes ¬ *aff-dim S* = *int* (*DIM*(*'n*))
shows *interior S* = {}
proof −
have *aff-dim*(*interior S*) ≤ *aff-dim S*
using *interior-subset aff-dim-subset*[*of interior S S*] **by** *auto*
then show *?thesis*
using *aff-dim-open*[*of interior S*] *aff-dim-subset-univ*[*of S*] *assms* **by** *auto*
qed

corollary *empty-interior-lowdim*:
fixes *S* :: *'n*::*euclidean-space set*
shows *dim S* < *DIM* (*'n*) ⟹ *interior S* = {}
by (*metis low-dim-interior affine-hull-univ dim-affine-hull less-not-refl dim-UNIV*)

19.9 Caratheodory's theorem.

lemma *convex-hull-caratheodory-aff-dim*:
fixes *p* :: (*'a*::*euclidean-space*) *set*

```
  shows convex hull p =
    {y. ∃ s u. finite s ∧ s ⊆ p ∧ card s ≤ aff-dim p + 1 ∧
      (∀ x∈s. 0 ≤ u x) ∧ setsum u s = 1 ∧ setsum (λv. u v *R v) s = y}
  unfolding convex-hull-explicit set-eq-iff mem-Collect-eq
proof (intro allI iffI)
  fix y
  let ?P = λn. ∃ s u. finite s ∧ card s = n ∧ s ⊆ p ∧ (∀ x∈s. 0 ≤ u x) ∧
    setsum u s = 1 ∧ (∑ v∈s. u v *R v) = y
  assume ∃ s u. finite s ∧ s ⊆ p ∧ (∀ x∈s. 0 ≤ u x) ∧ setsum u s = 1 ∧ (∑ v∈s.
u v *R v) = y
  then obtain N where ?P N by auto
  then have ∃ n≤N. (∀ k<n. ¬ ?P k) ∧ ?P n
    apply (rule-tac ex-least-nat-le)
    apply auto
    done
  then obtain n where ?P n and smallest: ∀ k<n. ¬ ?P k
    by blast
  then obtain s u where obt: finite s card s = n s⊆p ∀ x∈s. 0 ≤ u x
    setsum u s = 1  (∑ v∈s. u v *R v) = y by auto

  have card s ≤ aff-dim p + 1
  proof (rule ccontr, simp only: not-le)
    assume aff-dim p + 1 < card s
    then have affine-dependent s
    using affine-dependent-biggerset[OF obt(1)] independent-card-le-aff-dim not-less
obt(3)
      by blast
    then obtain w v where wv: setsum w s = 0 v∈s w v ≠ 0 (∑ v∈s. w v *R v)
= 0
      using affine-dependent-explicit-finite[OF obt(1)] by auto
    def i ≡ (λv. (u v) / (− w v)) ` {v∈s. w v < 0}
    def t ≡ Min i
    have ∃ x∈s. w x < 0
    proof (rule ccontr, simp add: not-less)
      assume as:∀ x∈s. 0 ≤ w x
      then have setsum w (s − {v}) ≥ 0
        apply (rule-tac setsum-nonneg)
        apply auto
        done
      then have setsum w s > 0
        unfolding setsum.remove[OF obt(1) ‹v∈s›]
        using as[THEN bspec[where x=v]]  ‹v∈s›  ‹w v ≠ 0› by auto
      then show False using wv(1) by auto
    qed
    then have i ≠ {} unfolding i-def by auto
    then have t ≥ 0
      using Min-ge-iff[of i 0 ] and obt(1)
      unfolding t-def i-def
      using obt(4)[unfolded le-less]
```

```
    by (auto simp: divide-le-0-iff)
  have t: ∀ v∈s. u v + t * w v ≥ 0
  proof
    fix v
    assume v ∈ s
    then have v: 0 ≤ u v
      using obt(4)[THEN bspec[where x=v]] by auto
    show 0 ≤ u v + t * w v
    proof (cases w v < 0)
      case False
      thus ?thesis using v ‹t≥0› by auto
    next
      case True
      then have t ≤ u v / (- w v)
        using ‹v∈s› unfolding t-def i-def
        apply (rule-tac Min-le)
        using obt(1) apply auto
        done
      then show ?thesis
        unfolding real-0-le-add-iff
        using pos-le-divide-eq[OF True[unfolded neg-0-less-iff-less[symmetric]]]
        by auto
    qed
  qed
  obtain a where a ∈ s and t = (λv. (u v) / (- w v)) a and w a < 0
    using Min-in[OF - ‹i≠{}›] and obt(1) unfolding i-def t-def by auto
  then have a: a ∈ s u a + t * w a = 0 by auto
  have *: ⋀f. setsum f (s - {a}) = setsum f s - ((f a)::'b::ab-group-add)
    unfolding setsum.remove[OF obt(1) ‹a∈s›] by auto
  have (∑ v∈s. u v + t * w v) = 1
     unfolding setsum.distrib wv(1) setsum-right-distrib[symmetric] obt(5) by
auto
  moreover have (∑ v∈s. u v *R v + (t * w v) *R v) - (u a *R a + (t * w a)
*R a) = y
   unfolding setsum.distrib obt(6) scaleR-scaleR[symmetric] scaleR-right.setsum
[symmetric] wv(4)
    using a(2) [THEN eq-neg-iff-add-eq-0 [THEN iffD2]] by simp
  ultimately have ?P (n - 1)
    apply (rule-tac x=(s - {a}) in exI)
    apply (rule-tac x=λv. u v + t * w v in exI)
    using obt(1-3) and t and a
    apply (auto simp add: * scaleR-left-distrib)
    done
  then show False
    using smallest[THEN spec[where x=n - 1]] by auto
qed
then show ∃ s u. finite s ∧ s ⊆ p ∧ card s ≤ aff-dim p + 1 ∧
    (∀ x∈s. 0 ≤ u x) ∧ setsum u s = 1 ∧ (∑ v∈s. u v *R v) = y
  using obt by auto
```

qed *auto*

lemma *caratheodory-aff-dim*:
fixes p :: ('*a*::*euclidean-space*) *set*
shows *convex hull* $p = \{x.\ \exists s.$ *finite* $s \wedge s \subseteq p \wedge$ *card* $s \leq$ *aff-dim* $p + 1 \wedge x \in$ *convex hull* $s\}$
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
apply (*subst convex-hull-caratheodory-aff-dim*)
apply *clarify*
apply (*rule-tac x*=*s* **in** *exI*)
apply (*simp add*: *hull-subset convex-explicit* [*THEN iffD1*, *OF convex-convex-hull*])
done
next
show *?rhs* ⊆ *?lhs*
using *hull-mono* **by** *blast*
qed

lemma *convex-hull-caratheodory*:
fixes p :: ('*a*::*euclidean-space*) *set*
shows *convex hull* $p =$
$\{y.\ \exists s\ u.$ *finite* $s \wedge s \subseteq p \wedge$ *card* $s \leq DIM('a) + 1 \wedge$
$(\forall x{\in}s.\ 0 \leq u\ x) \wedge$ *setsum* $u\ s = 1 \wedge$ *setsum* $(\lambda v.\ u\ v *_R v)\ s = y\}$
(**is** *?lhs* = *?rhs*)
proof (*intro set-eqI iffI*)
fix x
assume $x \in$ *?lhs* **then show** $x \in$ *?rhs*
apply (*simp only*: *convex-hull-caratheodory-aff-dim Set.mem-Collect-eq*)
apply (*erule ex-forward*)+
using *aff-dim-subset-univ* [*of p*]
apply *simp*
done
next
fix x
assume $x \in$ *?rhs* **then show** $x \in$ *?lhs*
by (*auto simp add*: *convex-hull-explicit*)
qed

theorem *caratheodory*:
convex hull $p =$
$\{x::'a::$*euclidean-space*$.\ \exists s.$ *finite* $s \wedge s \subseteq p \wedge$
card $s \leq DIM('a) + 1 \wedge x \in$ *convex hull* $s\}$
proof *safe*
fix x
assume $x \in$ *convex hull* p
then obtain $s\ u$ **where** *finite* $s\ s \subseteq p$ *card* $s \leq DIM('a) + 1$
$\forall x{\in}s.\ 0 \leq u\ x$ *setsum* $u\ s = 1\ (\sum v{\in}s.\ u\ v *_R v) = x$
unfolding *convex-hull-caratheodory* **by** *auto*

then show $\exists s.$ *finite* $s \wedge s \subseteq p \wedge$ *card* $s \leq DIM('a) + 1 \wedge x \in$ *convex hull* s
apply (*rule-tac* $x{=}s$ **in** *exI*)
using *hull-subset*[*of* s *convex*]
using *convex-convex-hull*[*unfolded convex-explicit*, *of* s,
THEN spec[**where** $x{=}s$], *THEN spec*[**where** $x{=}u$]]
apply *auto*
done
next
fix x s
assume *finite* s $s \subseteq p$ *card* $s \leq DIM('a) + 1$ $x \in$ *convex hull* s
then show $x \in$ *convex hull* p
using *hull-mono*[*OF* ‹$s \subseteq p$›] **by** *auto*
qed

19.10 Relative interior of a set

definition *rel-interior* $S =$
$\{x.\ \exists T.$ *openin* (*subtopology euclidean* (*affine hull* S)) $T \wedge x \in T \wedge T \subseteq S\}$

lemma *rel-interior*:
rel-interior $S = \{x \in S.\ \exists T.$ *open* $T \wedge x \in T \wedge T \cap$ *affine hull* $S \subseteq S\}$
unfolding *rel-interior-def*[*of* S] *openin-open*[*of affine hull* S]
apply *auto*
proof −
fix x T
assume $*$: $x \in S$ *open* T $x \in T$ $T \cap$ *affine hull* $S \subseteq S$
then have $**$: $x \in T \cap$ *affine hull* S
using *hull-inc* **by** *auto*
show $\exists Tb.\ (\exists Ta.$ *open* $Ta \wedge Tb =$ *affine hull* $S \cap Ta) \wedge x \in Tb \wedge Tb \subseteq S$
apply (*rule-tac* $x = T \cap$ (*affine hull* S) **in** *exI*)
using $*$ $**$
apply *auto*
done
qed

lemma *mem-rel-interior*: $x \in$ *rel-interior* $S \longleftrightarrow (\exists T.$ *open* $T \wedge x \in T \cap S \wedge T \cap$ *affine hull* $S \subseteq S$)
by (*auto simp add*: *rel-interior*)

lemma *mem-rel-interior-ball*:
$x \in$ *rel-interior* $S \longleftrightarrow x \in S \wedge (\exists e.\ e > 0 \wedge$ *ball* x $e \cap$ *affine hull* $S \subseteq S$)
apply (*simp add*: *rel-interior*, *safe*)
apply (*force simp add*: *open-contains-ball*)
apply (*rule-tac* $x =$ *ball* x e **in** *exI*)
apply *simp*
done

lemma *rel-interior-ball*:
rel-interior $S = \{x \in S.\ \exists e.\ e > 0 \wedge$ *ball* x $e \cap$ *affine hull* $S \subseteq S\}$

using *mem-rel-interior-ball* [*of* - *S*] **by** *auto*

lemma *mem-rel-interior-cball*:
x ∈ *rel-interior S* ⟷ *x* ∈ *S* ∧ (∃ *e*. *e* > *0* ∧ *cball x e* ∩ *affine hull S* ⊆ *S*)
apply (*simp add*: *rel-interior*, *safe*)
apply (*force simp add*: *open-contains-cball*)
apply (*rule-tac x* = *ball x e* **in** *exI*)
apply (*simp add*: *subset-trans* [*OF ball-subset-cball*])
apply *auto*
done

lemma *rel-interior-cball*:
rel-interior S = {*x* ∈ *S*. ∃ *e*. *e* > *0* ∧ *cball x e* ∩ *affine hull S* ⊆ *S*}
using *mem-rel-interior-cball* [*of* - *S*] **by** *auto*

lemma *rel-interior-empty* [*simp*]: *rel-interior* {} = {}
by (*auto simp add*: *rel-interior-def*)

lemma *affine-hull-sing* [*simp*]: *affine hull* {*a* :: *'n*::*euclidean-space*} = {*a*}
by (*metis affine-hull-eq affine-sing*)

lemma *rel-interior-sing* [*simp*]: *rel-interior* {*a* :: *'n*::*euclidean-space*} = {*a*}
unfolding *rel-interior-ball affine-hull-sing*
apply *auto*
apply (*rule-tac x* = *1* :: *real* **in** *exI*)
apply *simp*
done

lemma *subset-rel-interior*:
fixes *S T* :: *'n*::*euclidean-space set*
assumes *S* ⊆ *T*
and *affine hull S* = *affine hull T*
shows *rel-interior S* ⊆ *rel-interior T*
using *assms* **by** (*auto simp add*: *rel-interior-def*)

lemma *rel-interior-subset*: *rel-interior S* ⊆ *S*
by (*auto simp add*: *rel-interior-def*)

lemma *rel-interior-subset-closure*: *rel-interior S* ⊆ *closure S*
using *rel-interior-subset* **by** (*auto simp add*: *closure-def*)

lemma *interior-subset-rel-interior*: *interior S* ⊆ *rel-interior S*
by (*auto simp add*: *rel-interior interior-def*)

lemma *interior-rel-interior*:
fixes *S* :: *'n*::*euclidean-space set*
assumes *aff-dim S* = *int*(*DIM*(*'n*))
shows *rel-interior S* = *interior S*
proof −

```
  have affine hull S = UNIV
    using assms affine-hull-univ[of S] by auto
  then show ?thesis
    unfolding rel-interior interior-def by auto
qed

lemma rel-interior-interior:
  fixes S :: 'n::euclidean-space set
  assumes affine hull S = UNIV
  shows rel-interior S = interior S
  using assms unfolding rel-interior interior-def by auto

lemma rel-interior-open:
  fixes S :: 'n::euclidean-space set
  assumes open S
  shows rel-interior S = S
 by (metis assms interior-eq interior-subset-rel-interior rel-interior-subset set-eq-subset)

lemma interior-ball [simp]: interior (ball x e) = ball x e
  by (simp add: interior-open)

lemma interior-rel-interior-gen:
  fixes S :: 'n::euclidean-space set
  shows interior S = (if aff-dim S = int(DIM('n)) then rel-interior S else {})
  by (metis interior-rel-interior low-dim-interior)

lemma rel-interior-univ:
  fixes S :: 'n::euclidean-space set
  shows rel-interior (affine hull S) = affine hull S
proof −
  have ∗: rel-interior (affine hull S) ⊆ affine hull S
    using rel-interior-subset by auto
  {
    fix x
    assume x: x ∈ affine hull S
    def e ≡ 1::real
    then have e > 0 ball x e ∩ affine hull (affine hull S) ⊆ affine hull S
      using hull-hull[of - S] by auto
    then have x ∈ rel-interior (affine hull S)
      using x rel-interior-ball[of affine hull S] by auto
  }
  then show ?thesis using ∗ by auto
qed

lemma rel-interior-univ2: rel-interior (UNIV :: ('n::euclidean-space) set) = UNIV
  by (metis open-UNIV rel-interior-open)

lemma rel-interior-convex-shrink:
  fixes S :: 'a::euclidean-space set
```

```
  assumes convex S
    and c ∈ rel-interior S
    and x ∈ S
    and 0 < e
    and e ≤ 1
  shows x − e *R (x − c) ∈ rel-interior S
proof −
  obtain d where d > 0 and d: ball c d ∩ affine hull S ⊆ S
    using assms(2) unfolding  mem-rel-interior-ball by auto
  {
    fix y
    assume as: dist (x − e *R (x − c)) y < e * d y ∈ affine hull S
    have *: y = (1 − (1 − e)) *R ((1 / e) *R y − ((1 − e) / e) *R x) + (1 −
e) *R x
      using ‹e > 0› by (auto simp add: scaleR-left-diff-distrib scaleR-right-diff-distrib)
    have x ∈ affine hull S
      using assms hull-subset[of S] by auto
    moreover have 1 / e + − ((1 − e) / e) = 1
      using ‹e > 0› left-diff-distrib[of 1 (1−e) 1/e] by auto
    ultimately have **: (1 / e) *R y − ((1 − e) / e) *R x ∈ affine hull S
      using as affine-affine-hull[of S] mem-affine[of affine hull S y x (1 / e) −((1
− e) / e)]
      by (simp add: algebra-simps)
    have dist c ((1 / e) *R y − ((1 − e) / e) *R x) = |1/e| * norm (e *R c − y
+ (1 − e) *R x)
      unfolding dist-norm norm-scaleR[symmetric]
      apply (rule arg-cong[where f=norm])
      using ‹e > 0›
     apply (auto simp add: euclidean-eq-iff[where 'a='a] field-simps inner-simps)
      done
    also have ... = |1/e| * norm (x − e *R (x − c) − y)
      by (auto intro!:arg-cong[where f=norm] simp add: algebra-simps)
    also have ... < d
      using as[unfolded dist-norm] and ‹e > 0›
      by (auto simp add:pos-divide-less-eq[OF ‹e > 0›] mult.commute)
    finally have y ∈ S
      apply (subst *)
      apply (rule assms(1)[unfolded convex-alt,rule-format])
      apply (rule d[unfolded subset-eq,rule-format])
      unfolding mem-ball
      using assms(3−5) **
      apply auto
      done
  }
  then have ball (x − e *R (x − c)) (e*d) ∩ affine hull S ⊆ S
    by auto
  moreover have e * d > 0
    using ‹e > 0› ‹d > 0› by simp
  moreover have c: c ∈ S
```

```
    using assms rel-interior-subset by auto
  moreover from c have x - e *R (x - c) ∈ S
    using convexD-alt[of S x c e]
    apply (simp add: algebra-simps)
    using assms
    apply auto
    done
  ultimately show ?thesis
    using mem-rel-interior-ball[of x - e *R (x - c) S] ‹e > 0› by auto
qed

lemma interior-real-semiline:
  fixes a :: real
  shows interior {a..} = {a<..}
proof -
  {
    fix y
    assume a < y
    then have y ∈ interior {a..}
      apply (simp add: mem-interior)
      apply (rule-tac x=(y-a) in exI)
      apply (auto simp add: dist-norm)
      done
  }
  moreover
  {
    fix y
    assume y ∈ interior {a..}
    then obtain e where e: e > 0 cball y e ⊆ {a..}
      using mem-interior-cball[of y {a..}] by auto
    moreover from e have y - e ∈ cball y e
      by (auto simp add: cball-def dist-norm)
    ultimately have a ≤ y - e by blast
    then have a < y using e by auto
  }
  ultimately show ?thesis by auto
qed

lemma continuous-ge-on-Ioo:
   assumes continuous-on {c..d} g ⋀x. x ∈ {c<..<d} ⟹ g x ≥ a c < d x ∈
{c..d}
  shows g (x::real) ≥ (a::real)
proof-
 from assms(3) have {c..d} = closure {c<..<d} by (rule closure-greaterThanLessThan[symmetric])
  also from assms(2) have {c<..<d} ⊆ (g -‘ {a..} ∩ {c..d}) by auto
 hence closure {c<..<d} ⊆ closure (g -‘ {a..} ∩ {c..d}) by (rule closure-mono)
 also from assms(1) have closed (g -‘ {a..} ∩ {c..d})
   by (auto simp: continuous-on-closed-vimage)
 hence closure (g -‘ {a..} ∩ {c..d}) = g -‘ {a..} ∩ {c..d} by simp
```

```
  finally show ?thesis using ‹x ∈ {c..d}› by auto
qed

lemma interior-real-semiline':
  fixes a :: real
  shows interior {..a} = {..<a}
proof −
  {
    fix y
    assume a > y
    then have y ∈ interior {..a}
      apply (simp add: mem-interior)
      apply (rule-tac x=(a−y) in exI)
      apply (auto simp add: dist-norm)
      done
  }
  moreover
  {
    fix y
    assume y ∈ interior {..a}
    then obtain e where e: e > 0 cball y e ⊆ {..a}
      using mem-interior-cball[of y {..a}] by auto
    moreover from e have y + e ∈ cball y e
      by (auto simp add: cball-def dist-norm)
    ultimately have a ≥ y + e by auto
    then have a > y using e by auto
  }
  ultimately show ?thesis by auto
qed

lemma interior-atLeastAtMost-real: interior {a..b} = {a<..<b :: real}
proof−
  have {a..b} = {a..} ∩ {..b} by auto
  also have interior ... = {a<..} ∩ {..<b}
    by (simp add: interior-real-semiline interior-real-semiline')
  also have ... = {a<..<b} by auto
  finally show ?thesis .
qed

lemma frontier-real-Iic:
  fixes a :: real
  shows frontier {..a} = {a}
  unfolding frontier-def by (auto simp add: interior-real-semiline')

lemma rel-interior-real-box:
  fixes a b :: real
  assumes a < b
  shows rel-interior {a .. b} = {a <..< b}
proof −
```

have *box a b* ≠ {}
using *assms*
unfolding *set-eq-iff*
by (*auto intro*!: *exI*[*of* - (*a* + *b*) / *2*] *simp*: *box-def*)
then show *?thesis*
using *interior-rel-interior-gen*[*of cbox a b*, *symmetric*]
by (*simp split*: *split-if-asm del*: *box-real add*: *box-real*[*symmetric*] *interior-cbox*)
qed

lemma *rel-interior-real-semiline*:
fixes *a* :: *real*
shows *rel-interior* {*a*..} = {*a*<..}
proof −
have ∗: {*a*<..} ≠ {}
unfolding *set-eq-iff* **by** (*auto intro*!: *exI*[*of* - *a* + *1*])
then show *?thesis* **using** *interior-real-semiline interior-rel-interior-gen*[*of* {*a*..}]
by (*auto split*: *split-if-asm*)
qed

19.10.1 Relative open sets

definition *rel-open S* ⟷ *rel-interior S* = *S*

lemma *rel-open*: *rel-open S* ⟷ *openin* (*subtopology euclidean* (*affine hull S*)) *S*
unfolding *rel-open-def rel-interior-def*
apply *auto*
using *openin-subopen*[*of subtopology euclidean* (*affine hull S*) *S*]
apply *auto*
done

lemma *opein-rel-interior*: *openin* (*subtopology euclidean* (*affine hull S*)) (*rel-interior S*)
apply (*simp add*: *rel-interior-def*)
apply (*subst openin-subopen*)
apply *blast*
done

lemma *affine-rel-open*:
fixes *S* :: *'n*::*euclidean-space set*
assumes *affine S*
shows *rel-open S*
unfolding *rel-open-def*
using *assms rel-interior-univ*[*of S*] *affine-hull-eq*[*of S*]
by *metis*

lemma *affine-closed*:
fixes *S* :: *'n*::*euclidean-space set*
assumes *affine S*
shows *closed S*

proof −
{
assume $S \neq \{\}$
then obtain L **where** L: *subspace* L *affine-parallel* S L
using *assms affine-parallel-subspace*[*of* S] **by** *auto*
then obtain a **where** a: $S = (op + a\ `\ L)$
using *affine-parallel-def*[*of* L S] *affine-parallel-commut* **by** *auto*
from L **have** *closed* L **using** *closed-subspace* **by** *auto*
then have *closed* S
using *closed-translation* a **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *closure-affine-hull*:
fixes S :: $'n$::*euclidean-space set*
shows *closure* $S \subseteq$ *affine hull* S
by (*intro closure-minimal hull-subset affine-closed affine-affine-hull*)

lemma *closure-same-affine-hull*:
fixes S :: $'n$::*euclidean-space set*
shows *affine hull* (*closure* S) = *affine hull* S
proof −
have *affine hull* (*closure* S) $\subseteq$ *affine hull* S
using *hull-mono*[*of closure S affine hull S affine*]
closure-affine-hull[*of* S] *hull-hull*[*of affine* S]
by *auto*
moreover have *affine hull* (*closure* S) $\supseteq$ *affine hull* S
using *hull-mono*[*of* S *closure* S *affine*] *closure-subset* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *closure-aff-dim*:
fixes S :: $'n$::*euclidean-space set*
shows *aff-dim* (*closure* S) = *aff-dim* S
proof −
have *aff-dim* $S \leq$ *aff-dim* (*closure* S)
using *aff-dim-subset closure-subset* **by** *auto*
moreover have *aff-dim* (*closure* S) $\leq$ *aff-dim* (*affine hull* S)
using *aff-dim-subset closure-affine-hull* **by** *auto*
moreover have *aff-dim* (*affine hull* S) = *aff-dim* S
using *aff-dim-affine-hull* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *rel-interior-closure-convex-shrink*:
fixes S :: -::*euclidean-space set*
assumes *convex* S
and $c \in$ *rel-interior* S

```
    and x ∈ closure S
    and e > 0
    and e ≤ 1
  shows x − e *R (x − c) ∈ rel-interior S
proof −
  obtain d where d > 0 and d: ball c d ∩ affine hull S ⊆ S
    using assms(2) unfolding mem-rel-interior-ball by auto
  have ∃y ∈ S. norm (y − x) * (1 − e) < e * d
  proof (cases x ∈ S)
    case True
    then show ?thesis using ‹e > 0› ‹d > 0›
      apply (rule-tac bexI[where x=x])
      apply (auto)
      done
  next
    case False
    then have x: x islimpt S
      using assms(3)[unfolded closure-def] by auto
    show ?thesis
    proof (cases e = 1)
      case True
      obtain y where y ∈ S y ≠ x dist y x < 1
        using x[unfolded islimpt-approachable,THEN spec[where x=1]] by auto
      then show ?thesis
        apply (rule-tac x=y in bexI)
        unfolding True
        using ‹d > 0›
        apply auto
        done
    next
      case False
      then have 0 < e * d / (1 − e) and *: 1 − e > 0
        using ‹e ≤ 1› ‹e > 0› ‹d > 0› by (auto)
      then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 − e)
       using x[unfolded islimpt-approachable,THEN spec[where x=e*d / (1 − e)]]
by auto
      then show ?thesis
        apply (rule-tac x=y in bexI)
        unfolding dist-norm
        using pos-less-divide-eq[OF *]
        apply auto
        done
    qed
  qed
  then obtain y where y ∈ S and y: norm (y − x) * (1 − e) < e * d
    by auto
  def z ≡ c + ((1 − e) / e) *R (x − y)
  have *: x − e *R (x − c) = y − e *R (y − z)
    unfolding z-def using ‹e > 0›
```

by (*auto simp add*: *scaleR-right-diff-distrib scaleR-right-distrib scaleR-left-diff-distrib*)
have *zball*: $z \in$ *ball c d*
using *mem-ball z-def dist-norm*[*of c*]
using y **and** *assms*(*4*,*5*)
by (*auto simp add*:*field-simps norm-minus-commute*)
have $x \in$ *affine hull S*
using *closure-affine-hull assms* **by** *auto*
moreover have $y \in$ *affine hull S*
using ‹$y \in S$› *hull-subset*[*of S*] **by** *auto*
moreover have $c \in$ *affine hull S*
using *assms rel-interior-subset hull-subset*[*of S*] **by** *auto*
ultimately have $z \in$ *affine hull S*
using *z-def affine-affine-hull*[*of S*]
mem-affine-3-minus [*of affine hull S c x y* $(1 - e) / e$]
assms
by (*auto simp add*: *field-simps*)
then have $z \in S$ **using** *d zball* **by** *auto*
obtain *d1* **where** $d1 > 0$ **and** *d1*: *ball z d1* $\leq$ *ball c d*
using *zball open-ball*[*of c d*] *openE*[*of ball c d z*] **by** *auto*
then have *ball z d1* $\cap$ *affine hull S* $\subseteq$ *ball c d* $\cap$ *affine hull S*
by *auto*
then have *ball z d1* $\cap$ *affine hull S* $\subseteq S$
using *d* **by** *auto*
then have $z \in$ *rel-interior S*
using *mem-rel-interior-ball* **using** ‹$d1 > 0$› ‹$z \in S$› **by** *auto*
then have $y - e *_R (y - z) \in$ *rel-interior S*
using *rel-interior-convex-shrink*[*of S z y e*] *assms* ‹$y \in S$› **by** *auto*
then show *?thesis* **using** $*$ **by** *auto*
qed

19.10.2 Relative interior preserves under linear transformations

lemma *rel-interior-translation-aux*:
fixes a :: *'n*::*euclidean-space*
shows $((\lambda x.\ a + x)$ ‘ *rel-interior S*$) \subseteq$ *rel-interior* $((\lambda x.\ a + x)$ ‘ $S)$
proof −
{
fix x
assume x: $x \in$ *rel-interior S*
then obtain T **where** *open* T $x \in T \cap S$ $T \cap$ *affine hull S* $\subseteq S$
using *mem-rel-interior*[*of x S*] **by** *auto*
then have *open* $((\lambda x.\ a + x)$ ‘ $T)$
and $a + x \in ((\lambda x.\ a + x)$ ‘ $T) \cap ((\lambda x.\ a + x)$ ‘ $S)$
and $((\lambda x.\ a + x)$ ‘ $T) \cap$ *affine hull* $((\lambda x.\ a + x)$ ‘ $S) \subseteq (\lambda x.\ a + x)$ ‘ S
using *affine-hull-translation*[*of a S*] *open-translation*[*of T a*] x **by** *auto*
then have $a + x \in$ *rel-interior* $((\lambda x.\ a + x)$ ‘ $S)$
using *mem-rel-interior*[*of a+x* $((\lambda x.\ a + x)$ ‘ $S)$] **by** *auto*
}
then show *?thesis* **by** *auto*

qed

lemma *rel-interior-translation*:
fixes $a :: 'n::euclidean\text{-}space$
shows $rel\text{-}interior\ ((\lambda x.\ a + x)\ `\ S) = (\lambda x.\ a + x)\ `\ rel\text{-}interior\ S$
proof −
have $(\lambda x.\ (-a) + x)\ `\ rel\text{-}interior\ ((\lambda x.\ a + x)\ `\ S) \subseteq rel\text{-}interior\ S$
using *rel-interior-translation-aux*[*of* $-a$ $(\lambda x.\ a + x)\ `\ S$]
translation-assoc[*of* $-a$ *a*]
by *auto*
then have $((\lambda x.\ a + x)\ `\ rel\text{-}interior\ S) \supseteq rel\text{-}interior\ ((\lambda x.\ a + x)\ `\ S)$
using *translation-inverse-subset*[*of a rel-interior* $(op + a\ `\ S)$ *rel-interior S*]
by *auto*
then show *?thesis*
using *rel-interior-translation-aux*[*of a S*] **by** *auto*
qed

lemma *affine-hull-linear-image*:
assumes *bounded-linear f*
shows $f\ `\ (affine\ hull\ s) = affine\ hull\ f\ `\ s$
apply *rule*
unfolding *subset-eq ball-simps*
apply (*rule-tac*[!] *hull-induct*, *rule hull-inc*)
prefer *3*
apply (*erule imageE*)
apply (*rule-tac x*=*xa* **in** *image-eqI*)
apply *assumption*
apply (*rule hull-subset*[*unfolded subset-eq*, *rule-format*])
apply *assumption*
proof −
interpret *f*: *bounded-linear f* **by** *fact*
show *affine* $\{x.\ f\ x \in affine\ hull\ f\ `\ s\}$
unfolding *affine-def*
by (*auto simp add*: *f.scaleR f.add affine-affine-hull*[*unfolded affine-def*, *rule-format*])
show *affine* $\{x.\ x \in f\ `\ (affine\ hull\ s)\}$
using *affine-affine-hull*[*unfolded affine-def*, *of s*]
unfolding *affine-def* **by** (*auto simp add*: *f.scaleR* [*symmetric*] *f.add* [*symmetric*])
qed *auto*

lemma *rel-interior-injective-on-span-linear-image*:
fixes $f :: 'm::euclidean\text{-}space \Rightarrow 'n::euclidean\text{-}space$
and $S :: 'm::euclidean\text{-}space\ set$
assumes *bounded-linear f*
and *inj-on f* (*span S*)
shows $rel\text{-}interior\ (f\ `\ S) = f\ `\ (rel\text{-}interior\ S)$
proof −
{

```
  fix z
  assume z: z ∈ rel-interior (f ‘ S)
  then have z ∈ f ‘ S
    using rel-interior-subset[of f ‘ S] by auto
  then obtain x where x: x ∈ S f x = z by auto
  obtain e2 where e2: e2 > 0 cball z e2 ∩ affine hull (f ‘ S) ⊆ (f ‘ S)
    using z rel-interior-cball[of f ‘ S] by auto
  obtain K where K: K > 0 ⋀x. norm (f x) ≤ norm x * K
   using assms Real-Vector-Spaces.bounded-linear.pos-bounded[of f] by auto
  def e1 ≡ 1 / K
  then have e1: e1 > 0 ⋀x. e1 * norm (f x) ≤ norm x
    using K pos-le-divide-eq[of e1] by auto
  def e ≡ e1 * e2
  then have e > 0 using e1 e2 by auto
  {
    fix y
    assume y: y ∈ cball x e ∩ affine hull S
    then have h1: f y ∈ affine hull (f ‘ S)
      using affine-hull-linear-image[of f S] assms by auto
    from y have norm (x−y) ≤ e1 * e2
      using cball-def[of x e] dist-norm[of x y] e-def by auto
    moreover have f x − f y = f (x − y)
      using assms linear-sub[of f x y] linear-conv-bounded-linear[of f] by auto
    moreover have e1 * norm (f (x−y)) ≤ norm (x − y)
      using e1 by auto
    ultimately have e1 * norm ((f x)−(f y)) ≤ e1 * e2
      by auto
    then have f y ∈ cball z e2
      using cball-def[of f x e2] dist-norm[of f x f y] e1 x by auto
    then have f y ∈ f ‘ S
      using y e2 h1 by auto
    then have y ∈ S
      using assms y hull-subset[of S] affine-hull-subset-span
        inj-on-image-mem-iff [OF ‹inj-on f (span S)›]
      by (metis Int-iff span-inc subsetCE)
  }
  then have z ∈ f ‘ (rel-interior S)
    using mem-rel-interior-cball[of x S] ‹e > 0› x by auto
}
moreover
{
  fix x
  assume x: x ∈ rel-interior S
  then obtain e2 where e2: e2 > 0 cball x e2 ∩ affine hull S ⊆ S
    using rel-interior-cball[of S] by auto
  have x ∈ S using x rel-interior-subset by auto
  then have *: f x ∈ f ‘ S by auto
  have ∀ x∈span S. f x = 0 ⟶ x = 0
    using assms subspace-span linear-conv-bounded-linear[of f]
```

```
      linear-injective-on-subspace-0[of f span S]
    by auto
  then obtain e1 where e1: e1 > 0 ∀x∈span S. e1 * norm x ≤ norm (f x)
    using assms injective-imp-isometric[of span S f]
      subspace-span[of S] closed-subspace[of span S]
    by auto
  def e ≡ e1 * e2
  hence e > 0 using e1 e2 by auto
  {
    fix y
    assume y: y ∈ cball (f x) e ∩ affine hull (f ` S)
    then have y ∈ f ` (affine hull S)
      using affine-hull-linear-image[of f S] assms by auto
    then obtain xy where xy: xy ∈ affine hull S f xy = y by auto
    with y have norm (f x − f xy) ≤ e1 * e2
      using cball-def[of f x e] dist-norm[of f x y] e-def by auto
    moreover have f x − f xy = f (x − xy)
      using assms linear-sub[of f x xy] linear-conv-bounded-linear[of f] by auto
    moreover have *: x − xy ∈ span S
      using subspace-sub[of span S x xy] subspace-span ‹x ∈ S› xy
        affine-hull-subset-span[of S] span-inc
      by auto
    moreover from * have e1 * norm (x − xy) ≤ norm (f (x − xy))
      using e1 by auto
    ultimately have e1 * norm (x − xy) ≤ e1 * e2
      by auto
    then have xy ∈ cball x e2
      using cball-def[of x e2] dist-norm[of x xy] e1 by auto
    then have y ∈ f ` S
      using xy e2 by auto
  }
  then have f x ∈ rel-interior (f ` S)
    using mem-rel-interior-cball[of (f x) (f ` S)] * ‹e > 0› by auto
 }
 ultimately show ?thesis by auto
qed

lemma rel-interior-injective-linear-image:
 fixes f :: 'm::euclidean-space ⇒ 'n::euclidean-space
 assumes bounded-linear f
   and inj f
 shows rel-interior (f ` S) = f ` (rel-interior S)
 using assms rel-interior-injective-on-span-linear-image[of f S]
   subset-inj-on[of f UNIV span S]
 by auto
```

19.11 Some Properties of subset of standard basis

lemma *affine-hull-substd-basis*:

assumes $d \subseteq Basis$
shows *affine hull* (*insert 0 d*) = $\{x::'a::euclidean\text{-}space.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x{\cdot}i = 0\}$
(**is** *affine hull* (*insert 0 ?A*) = *?B*)
proof −
have ∗: $\bigwedge A.$ *op* + (*0*::'*a*) ' *A* = *A* $\bigwedge A.$ *op* + (− (*0*::'*a*)) ' *A* = *A*
by *auto*
show *?thesis*
unfolding *affine-hull-insert-span-gen span-substd-basis*[*OF assms*,*symmetric*] ∗
..
qed

lemma *affine-hull-convex-hull* [*simp*]: *affine hull* (*convex hull S*) = *affine hull S*
by (*metis Int-absorb1 Int-absorb2 convex-hull-subset-affine-hull hull-hull hull-mono hull-subset*)

19.12 Openness and compactness are preserved by convex hull operation.

lemma *open-convex-hull*[*intro*]:
fixes *s* :: '*a*::*real-normed-vector set*
assumes *open s*
shows *open* (*convex hull s*)
unfolding *open-contains-cball convex-hull-explicit*
unfolding *mem-Collect-eq ball-simps*(*8*)
proof (*rule*, *rule*)
fix *a*
assume $\exists sa\ u.$ *finite sa* ∧ *sa* ⊆ *s* ∧ $(\forall x{\in}sa.\ 0 \le u\ x)$ ∧ *setsum u sa* = *1* ∧ $(\sum v{\in}sa.\ u\ v *_R v)$ = *a*
then obtain *t u* **where** *obt*: *finite t t*⊆*s* $\forall x{\in}t.\ 0 \le u\ x$ *setsum u t* = *1* $(\sum v{\in}t.\ u\ v *_R v)$ = *a*
by *auto*

from *assms*[*unfolded open-contains-cball*] **obtain** *b*
where *b*: $\forall x{\in}s.\ 0 < b\ x \wedge$ *cball x* (*b x*) ⊆ *s*
using *bchoice*[*of s* $\lambda x\ e.\ e > 0 \wedge$ *cball x e* ⊆ *s*] **by** *auto*
have *b* ' *t* ≠ {}
using *obt* **by** *auto*
def *i* ≡ *b* ' *t*

show ∃ *e* > *0*.
cball a e ⊆ $\{y.\ \exists sa\ u.$ *finite sa* ∧ *sa* ⊆ *s* ∧ $(\forall x{\in}sa.\ 0 \le u\ x)$ ∧ *setsum u sa* = *1* ∧ $(\sum v{\in}sa.\ u\ v *_R v) = y\}$
apply (*rule-tac x* = *Min i* **in** *exI*)
unfolding *subset-eq*
apply *rule*
defer
apply *rule*
unfolding *mem-Collect-eq*

```
proof −
  show 0 < Min i
    unfolding i-def and Min-gr-iff[OF finite-imageI[OF obt(1)] ‹b ‘ t≠{}›]
    using b
    apply simp
    apply rule
    apply (erule-tac x=x in ballE)
    using ‹t⊆s›
    apply auto
    done
next
  fix y
  assume y ∈ cball a (Min i)
  then have y: norm (a − y) ≤ Min i
    unfolding dist-norm[symmetric] by auto
  {
    fix x
    assume x ∈ t
    then have Min i ≤ b x
      unfolding i-def
      apply (rule-tac Min-le)
      using obt(1)
      apply auto
      done
    then have x + (y − a) ∈ cball x (b x)
      using y unfolding mem-cball dist-norm by auto
    moreover from ‹x∈t› have x ∈ s
      using obt(2) by auto
    ultimately have x + (y − a) ∈ s
      using y and b[THEN bspec[where x=x]] unfolding subset-eq by fast
  }
  moreover
  have ∗: inj-on (λv. v + (y − a)) t
    unfolding inj-on-def by auto
  have (∑ v∈(λv. v + (y − a)) ‘ t. u (v − (y − a))) = 1
    unfolding setsum.reindex[OF ∗] o-def using obt(4) by auto
  moreover have (∑ v∈(λv. v + (y − a)) ‘ t. u (v − (y − a)) ∗R v) = y
    unfolding setsum.reindex[OF ∗] o-def using obt(4,5)
    by (simp add: setsum.distrib setsum-subtractf scaleR-left.setsum[symmetric]
scaleR-right-distrib)
  ultimately
  show ∃ sa u. finite sa ∧ (∀ x∈sa. x ∈ s) ∧ (∀ x∈sa. 0 ≤ u x) ∧ setsum u sa =
1 ∧ (∑ v∈sa. u v ∗R v) = y
    apply (rule-tac x=(λv. v + (y − a)) ‘ t in exI)
    apply (rule-tac x=λv. u (v − (y − a)) in exI)
    using obt(1, 3)
    apply auto
    done
qed
```

qed

lemma *compact-convex-combinations*:
fixes *s t* :: *'a*::*real-normed-vector set*
assumes *compact s compact t*
shows *compact* { $(1 - u) *_R x + u *_R y \mid x\ y\ u.\ 0 \le u \wedge u \le 1 \wedge x \in s \wedge y \in t$}
proof −
let *?X* = {*0..1*} × *s* × *t*
let *?h* = (λ*z*. (*1* − *fst z*) *∗R fst* (*snd z*) + *fst z ∗R snd* (*snd z*))
have *∗*: { $(1 - u) *_R x + u *_R y \mid x\ y\ u.\ 0 \le u \wedge u \le 1 \wedge x \in s \wedge y \in t$} = *?h ' ?X*
apply (*rule set-eqI*)
unfolding *image-iff mem-Collect-eq*
apply *rule*
apply *auto*
apply (*rule-tac x*=*u* **in** *rev-bexI*)
apply *simp*
apply (*erule rev-bexI*)
apply (*erule rev-bexI*)
apply *simp*
apply *auto*
done
have *continuous-on ?X* (λ*z*. (*1* − *fst z*) *∗R fst* (*snd z*) + *fst z ∗R snd* (*snd z*))
unfolding *continuous-on* **by** (*rule ballI*) (*intro tendsto-intros*)
then show *?thesis*
unfolding *∗*
apply (*rule compact-continuous-image*)
apply (*intro compact-Times compact-Icc assms*)
done
qed

lemma *finite-imp-compact-convex-hull*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *finite s*
shows *compact* (*convex hull s*)
proof (*cases s* = {})
case *True*
then show *?thesis* **by** *simp*
next
case *False*
with *assms* **show** *?thesis*
proof (*induct rule*: *finite-ne-induct*)
case (*singleton x*)
show *?case* **by** *simp*
next
case (*insert x A*)
let *?f* = λ(*u*, *y*::*'a*). *u ∗R x* + (*1* − *u*) *∗R y*
let *?T* = {*0..1*::*real*} × (*convex hull A*)

```
    have continuous-on ?T ?f
      unfolding split-def continuous-on by (intro ballI tendsto-intros)
    moreover have compact ?T
      by (intro compact-Times compact-Icc insert)
    ultimately have compact (?f ` ?T)
      by (rule compact-continuous-image)
    also have ?f ` ?T = convex hull (insert x A)
      unfolding convex-hull-insert [OF ‹A ≠ {}›]
      apply safe
      apply (rule-tac x=a in exI, simp)
      apply (rule-tac x=1 − a in exI, simp)
      apply fast
      apply (rule-tac x=(u, b) in image-eqI, simp-all)
      done
    finally show compact (convex hull (insert x A)) .
  qed
qed

lemma compact-convex-hull:
  fixes s :: 'a::euclidean-space set
  assumes compact s
  shows compact (convex hull s)
proof (cases s = {})
  case True
  then show ?thesis using compact-empty by simp
next
  case False
  then obtain w where w ∈ s by auto
  show ?thesis
    unfolding caratheodory[of s]
  proof (induct (DIM('a) + 1))
    case 0
    have ∗: {x.∃ sa. finite sa ∧ sa ⊆ s ∧ card sa ≤ 0 ∧ x ∈ convex hull sa} = {}
      using compact-empty by auto
    from 0 show ?case unfolding ∗ by simp
  next
    case (Suc n)
    show ?case
    proof (cases n = 0)
      case True
      have {x. ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull t} = s
        unfolding set-eq-iff and mem-Collect-eq
      proof (rule, rule)
        fix x
        assume ∃ t. finite t ∧ t ⊆ s ∧ card t ≤ Suc n ∧ x ∈ convex hull t
        then obtain t where t: finite t t ⊆ s card t ≤ Suc n x ∈ convex hull t
          by auto
        show x ∈ s
        proof (cases card t = 0)
```

case *True*
then show *?thesis*
using *t(4)* **unfolding** *card-0-eq[OF t(1)]* **by** *simp*
next
case *False*
then have *card t = Suc 0* **using** *t(3)* ‹*n=0*› **by** *auto*
then obtain *a* **where** $t = \{a\}$ **unfolding** *card-Suc-eq* **by** *auto*
then show *?thesis* **using** *t(2,4)* **by** *simp*
qed
next
fix *x* **assume** $x \in s$
then show $\exists t.$ *finite* $t \wedge t \subseteq s \wedge$ *card* $t \leq$ *Suc* $n \wedge x \in$ *convex hull* t
apply (*rule-tac x=*$\{x\}$ **in** *exI*)
unfolding *convex-hull-singleton*
apply *auto*
done
qed
then show *?thesis* **using** *assms* **by** *simp*
next
case *False*
have $\{x.\ \exists t.$ *finite* $t \wedge t \subseteq s \wedge$ *card* $t \leq$ *Suc* $n \wedge x \in$ *convex hull* $t\} =$
$\{(1 - u) *_R x + u *_R y \mid x\ y\ u.$
$0 \leq u \wedge u \leq 1 \wedge x \in s \wedge y \in \{x.\ \exists t.$ *finite* $t \wedge t \subseteq s \wedge$ *card* $t \leq n \wedge x \in$ *convex hull* $t\}\}$
unfolding *set-eq-iff* **and** *mem-Collect-eq*
proof (*rule*, *rule*)
fix *x*
assume $\exists u\ v\ c.\ x = (1 - c) *_R u + c *_R v \wedge$
$0 \leq c \wedge c \leq 1 \wedge u \in s \wedge (\exists t.$ *finite* $t \wedge t \subseteq s \wedge$ *card* $t \leq n \wedge v \in$ *convex hull* t)
then obtain *u v c t* **where** *obt*: $x = (1 - c) *_R u + c *_R v$
$0 \leq c \wedge c \leq 1\ u \in s$ *finite* $t\ t \subseteq s$ *card* $t \leq n\ \ v \in$ *convex hull* t
by *auto*
moreover have $(1 - c) *_R u + c *_R v \in$ *convex hull insert u t*
apply (*rule convexD-alt*)
using *obt(2)* **and** *convex-convex-hull* **and** *hull-subset*[*of insert u t convex*]
using *obt(7)* **and** *hull-mono*[*of t insert u t*]
apply *auto*
done
ultimately show $\exists t.$ *finite* $t \wedge t \subseteq s \wedge$ *card* $t \leq$ *Suc* $n \wedge x \in$ *convex hull* t
apply (*rule-tac x=insert u t* **in** *exI*)
apply (*auto simp add*: *card-insert-if*)
done
next
fix *x*
assume $\exists t.$ *finite* $t \wedge t \subseteq s \wedge$ *card* $t \leq$ *Suc* $n \wedge x \in$ *convex hull* t
then obtain *t* **where** *t*: *finite* $t\ t \subseteq s$ *card* $t \leq$ *Suc* $n\ x \in$ *convex hull* t
by *auto*
show $\exists u\ v\ c.\ x = (1 - c) *_R u + c *_R v \wedge$

```
      0 ≤ c ∧ c ≤ 1 ∧ u ∈ s ∧ (∃ t. finite t ∧ t ⊆ s ∧ card t ≤ n ∧ v ∈ convex
hull t)
     proof (cases card t = Suc n)
      case False
      then have card t ≤ n using t(3) by auto
      then show ?thesis
       apply (rule-tac x=w in exI, rule-tac x=x in exI, rule-tac x=1 in exI)
       using ‹w∈s› and t
       apply (auto intro!: exI[where x=t])
       done
     next
      case True
      then obtain a u where au: t = insert a u a∉u
       apply (drule-tac card-eq-SucD)
       apply auto
       done
      show ?thesis
      proof (cases u = {})
       case True
       then have x = a using t(4)[unfolded au] by auto
       show ?thesis unfolding ‹x = a›
        apply (rule-tac x=a in exI)
        apply (rule-tac x=a in exI)
        apply (rule-tac x=1 in exI)
        using t and ‹n ≠ 0›
        unfolding au
        apply (auto intro!: exI[where x={a}])
        done
       next
       case False
       obtain ux vx b where obt: ux≥0 vx≥0 ux + vx = 1
         b ∈ convex hull u x = ux *R a + vx *R b
         using t(4)[unfolded au convex-hull-insert[OF False]]
         by auto
       have *: 1 - vx = ux using obt(3) by auto
       show ?thesis
        apply (rule-tac x=a in exI)
        apply (rule-tac x=b in exI)
        apply (rule-tac x=vx in exI)
        using obt and t(1-3)
        unfolding au and * using card-insert-disjoint[OF - au(2)]
        apply (auto intro!: exI[where x=u])
        done
       qed
     qed
    qed
    then show ?thesis
     using compact-convex-combinations[OF assms Suc] by simp
   qed
```

qed
qed

19.13 Extremal points of a simplex are some vertices.

lemma *dist-increases-online*:
fixes *a b d* :: *'a*::*real-inner*
assumes $d \neq 0$
shows $dist\ a\ (b + d) > dist\ a\ b \vee dist\ a\ (b - d) > dist\ a\ b$
proof (*cases inner a d − inner b d > 0*)
case *True*
then have $0 < inner\ d\ d + (inner\ a\ d * 2 - inner\ b\ d * 2)$
apply (*rule-tac add-pos-pos*)
using *assms*
apply *auto*
done
then show *?thesis*
apply (*rule-tac disjI2*)
unfolding *dist-norm* **and** *norm-eq-sqrt-inner* **and** *real-sqrt-less-iff*
apply (*simp add*: *algebra-simps inner-commute*)
done
next
case *False*
then have $0 < inner\ d\ d + (inner\ b\ d * 2 - inner\ a\ d * 2)$
apply (*rule-tac add-pos-nonneg*)
using *assms*
apply *auto*
done
then show *?thesis*
apply (*rule-tac disjI1*)
unfolding *dist-norm* **and** *norm-eq-sqrt-inner* **and** *real-sqrt-less-iff*
apply (*simp add*: *algebra-simps inner-commute*)
done
qed

lemma *norm-increases-online*:
fixes *d* :: *'a*::*real-inner*
shows $d \neq 0 \Longrightarrow norm\ (a + d) > norm\ a \vee norm(a - d) > norm\ a$
using *dist-increases-online*[*of d a 0*] **unfolding** *dist-norm* **by** *auto*

lemma *simplex-furthest-lt*:
fixes *s* :: *'a*::*real-inner set*
assumes *finite s*
shows $\forall x \in convex\ hull\ s.\ \ x \notin s \longrightarrow (\exists y \in convex\ hull\ s.\ norm\ (x - a) < norm(y - a))$
using *assms*
proof *induct*
fix *x s*
assume *as*: *finite s* $x \notin s$ $\forall x \in convex\ hull\ s.\ x \notin s \longrightarrow (\exists y \in convex\ hull\ s.\ norm$

```
(x − a) < norm (y − a))
  show ∀ xa∈convex hull insert x s. xa ∉ insert x s ⟶
    (∃ y∈convex hull insert x s. norm (xa − a) < norm (y − a))
  proof (rule, rule, cases s = {})
    case False
    fix y
    assume y: y ∈ convex hull insert x s y ∉ insert x s
    obtain u v b where obt: u≥0 v≥0 u + v = 1 b ∈ convex hull s y = u *R x
+ v *R b
      using y(1)[unfolded convex-hull-insert[OF False]] by auto
    show ∃ z∈convex hull insert x s. norm (y − a) < norm (z − a)
    proof (cases y ∈ convex hull s)
      case True
      then obtain z where z ∈ convex hull s norm (y − a) < norm (z − a)
        using as(3)[THEN bspec[where x=y]] and y(2) by auto
      then show ?thesis
        apply (rule-tac x=z in bexI)
        unfolding convex-hull-insert[OF False]
        apply auto
        done
    next
      case False
      show ?thesis
        using obt(3)
      proof (cases u = 0, case-tac[!] v = 0)
        assume u = 0 v ≠ 0
        then have y = b using obt by auto
        then show ?thesis using False and obt(4) by auto
      next
        assume u ≠ 0 v = 0
        then have y = x using obt by auto
        then show ?thesis using y(2) by auto
      next
        assume u ≠ 0 v ≠ 0
        then obtain w where w: w>0 w<u w<v
          using real-lbound-gt-zero[of u v] and obt(1,2) by auto
        have x ≠ b
        proof
          assume x = b
          then have y = b unfolding obt(5)
            using obt(3) by (auto simp add: scaleR-left-distrib[symmetric])
          then show False using obt(4) and False by simp
        qed
        then have *: w *R (x − b) ≠ 0 using w(1) by auto
        show ?thesis
          using dist-increases-online[OF *, of a y]
        proof (elim disjE)
          assume dist a y < dist a (y + w *R (x − b))
          then have norm (y − a) < norm ((u + w) *R x + (v − w) *R b − a)
```

```
        unfolding dist-commute[of a]
        unfolding dist-norm obt(5)
        by (simp add: algebra-simps)
      moreover have (u + w) *R x + (v - w) *R b ∈ convex hull insert x s
        unfolding convex-hull-insert[OF ‹s≠{}›] and mem-Collect-eq
        apply (rule-tac x=u + w in exI)
        apply rule
        defer
        apply (rule-tac x=v - w in exI)
        using ‹u ≥ 0› and w and obt(3,4)
        apply auto
        done
      ultimately show ?thesis by auto
    next
      assume dist a y < dist a (y - w *R (x - b))
      then have norm (y - a) < norm ((u - w) *R x + (v + w) *R b - a)
        unfolding dist-commute[of a]
        unfolding dist-norm obt(5)
        by (simp add: algebra-simps)
      moreover have (u - w) *R x + (v + w) *R b ∈ convex hull insert x s
        unfolding convex-hull-insert[OF ‹s≠{}›] and mem-Collect-eq
        apply (rule-tac x=u - w in exI)
        apply rule
        defer
        apply (rule-tac x=v + w in exI)
        using ‹u ≥ 0› and w and obt(3,4)
        apply auto
        done
      ultimately show ?thesis by auto
    qed
  qed auto
 qed
qed (auto simp add: assms)

lemma simplex-furthest-le:
 fixes s :: 'a::real-inner set
 assumes finite s
   and s ≠ {}
 shows ∃y∈s. ∀x∈ convex hull s. norm (x - a) ≤ norm (y - a)
proof -
 have convex hull s ≠ {}
   using hull-subset[of s convex] and assms(2) by auto
 then obtain x where x: x ∈ convex hull s ∀y∈convex hull s. norm (y - a) ≤
norm (x - a)
   using distance-attains-sup[OF finite-imp-compact-convex-hull[OF assms(1)], of
a]
   unfolding dist-commute[of a]
   unfolding dist-norm
```

by *auto*
show *?thesis*
proof (*cases* $x \in s$)
case *False*
then obtain y **where** $y \in$ *convex hull* s *norm* $(x - a) <$ *norm* $(y - a)$
using *simplex-furthest-lt*[*OF assms*(*1*), *THEN bspec*[**where** $x{=}x$]] **and** $x(1)$
by *auto*
then show *?thesis*
using $x(2)$[*THEN bspec*[**where** $x{=}y$]] **by** *auto*
next
case *True*
with x **show** *?thesis* **by** *auto*
qed
qed

lemma *simplex-furthest-le-exists*:
fixes s :: ('*a*::*real-inner*) *set*
shows *finite* $s \Longrightarrow \forall x{\in}$(*convex hull* s). $\exists y{\in}s$. *norm* $(x - a) \leq$ *norm* $(y - a)$
using *simplex-furthest-le*[*of* s] **by** (*cases* $s = \{\}$) *auto*

lemma *simplex-extremal-le*:
fixes s :: '*a*::*real-inner set*
assumes *finite* s
and $s \neq \{\}$
shows $\exists u{\in}s$. $\exists v{\in}s$. $\forall x{\in}$*convex hull* s. $\forall y \in$ *convex hull* s. *norm* $(x - y) \leq$ *norm* $(u - v)$
proof −
have *convex hull* $s \neq \{\}$
using *hull-subset*[*of* s *convex*] **and** *assms*(*2*) **by** *auto*
then obtain u v **where** *obt*: $u \in$ *convex hull* s $v \in$ *convex hull* s
$\forall x{\in}$*convex hull* s. $\forall y{\in}$*convex hull* s. *norm* $(x - y) \leq$ *norm* $(u - v)$
using *compact-sup-maxdistance*[*OF finite-imp-compact-convex-hull*[*OF assms*(*1*)]]
by (*auto simp*: *dist-norm*)
then show *?thesis*
proof (*cases* $u{\notin}s \vee v{\notin}s$, *elim disjE*)
assume $u \notin s$
then obtain y **where** $y \in$ *convex hull* s *norm* $(u - v) <$ *norm* $(y - v)$
using *simplex-furthest-lt*[*OF assms*(*1*), *THEN bspec*[**where** $x{=}u$]] **and** *obt*(*1*)
by *auto*
then show *?thesis*
using *obt*(*3*)[*THEN bspec*[**where** $x{=}y$], *THEN bspec*[**where** $x{=}v$]] **and** *obt*(*2*)
by *auto*
next
assume $v \notin s$
then obtain y **where** $y \in$ *convex hull* s *norm* $(v - u) <$ *norm* $(y - u)$
using *simplex-furthest-lt*[*OF assms*(*1*), *THEN bspec*[**where** $x{=}v$]] **and** *obt*(*2*)
by *auto*
then show *?thesis*
using *obt*(*3*)[*THEN bspec*[**where** $x{=}u$], *THEN bspec*[**where** $x{=}y$]] **and** *obt*(*1*)

by (*auto simp add*: *norm-minus-commute*)
qed *auto*
qed

lemma *simplex-extremal-le-exists*:
fixes *s* :: *'a*::*real-inner set*
shows *finite s* ⟹ *x* ∈ *convex hull s* ⟹ *y* ∈ *convex hull s* ⟹
∃ *u*∈*s*. ∃ *v*∈*s*. *norm* (*x* − *y*) ≤ *norm* (*u* − *v*)
using *convex-hull-empty simplex-extremal-le*[*of s*]
by(*cases s* = {}) *auto*

19.14 Closest point of a convex set is unique, with a continuous projection.

definition *closest-point* :: *'a*::{*real-inner*,*heine-borel*} *set* ⇒ *'a* ⇒ *'a*
where *closest-point s a* = (*SOME x*. *x* ∈ *s* ∧ (∀ *y*∈*s*. *dist a x* ≤ *dist a y*))

lemma *closest-point-exists*:
assumes *closed s*
and *s* ≠ {}
shows *closest-point s a* ∈ *s*
and ∀ *y*∈*s*. *dist a* (*closest-point s a*) ≤ *dist a y*
unfolding *closest-point-def*
apply(*rule-tac*[!] *someI2-ex*)
using *distance-attains-inf*[*OF assms*(*1*,*2*), *of a*]
apply *auto*
done

lemma *closest-point-in-set*: *closed s* ⟹ *s* ≠ {} ⟹ *closest-point s a* ∈ *s*
by (*meson closest-point-exists*)

lemma *closest-point-le*: *closed s* ⟹ *x* ∈ *s* ⟹ *dist a* (*closest-point s a*) ≤ *dist a x*
using *closest-point-exists*[*of s*] **by** *auto*

lemma *closest-point-self*:
assumes *x* ∈ *s*
shows *closest-point s x* = *x*
unfolding *closest-point-def*
apply (*rule some1-equality*, *rule ex1I*[*of* - *x*])
using *assms*
apply *auto*
done

lemma *closest-point-refl*: *closed s* ⟹ *s* ≠ {} ⟹ *closest-point s x* = *x* ⟷ *x* ∈ *s*
using *closest-point-in-set*[*of s x*] *closest-point-self*[*of x s*]
by *auto*

lemma *closer-points-lemma*:
assumes *inner y z > 0*
shows $\exists u{>}0.\ \forall v{>}0.\ v \leq u \longrightarrow norm(v *_R z - y) < norm\ y$
proof −
have *z*: *inner z z > 0*
unfolding *inner-gt-zero-iff* **using** *assms* **by** *auto*
then show *?thesis*
using *assms*
apply (*rule-tac x = inner y z / inner z z* **in** *exI*)
apply *rule*
defer
proof *rule+*
fix *v*
assume *0 < v* **and** *v ≤ inner y z / inner z z*
then show *norm (v *R z − y) < norm y*
unfolding *norm-lt* **using** *z* **and** *assms*
by (*simp add*: *field-simps inner-diff inner-commute mult-strict-left-mono*[*OF* - ‹*0<v*›])
qed *auto*
qed

lemma *closer-point-lemma*:
assumes *inner (y − x) (z − x) > 0*
shows $\exists u{>}0.\ u \leq 1 \wedge dist\ (x + u *_R (z - x))\ y < dist\ x\ y$
proof −
obtain *u* **where** *u > 0*
and *u*: $\forall v{>}0.\ v \leq u \longrightarrow norm\ (v *_R (z - x) - (y - x)) < norm\ (y - x)$
using *closer-points-lemma*[*OF assms*] **by** *auto*
show *?thesis*
apply (*rule-tac x=min u 1* **in** *exI*)
using *u*[*THEN spec*[**where** *x=min u 1*]] **and** ‹*u > 0*›
unfolding *dist-norm* **by** (*auto simp add*: *norm-minus-commute field-simps*)
qed

lemma *any-closest-point-dot*:
assumes *convex s closed s x ∈ s y ∈ s* $\forall z{\in}s.$ *dist a x ≤ dist a z*
shows *inner (a − x) (y − x) ≤ 0*
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *u* **where** *u*: *u>0 u≤1 dist (x + u *R (y − x)) a < dist x a*
using *closer-point-lemma*[*of a x y*] **by** *auto*
let *?z = (1 − u) *R x + u *R y*
have *?z ∈ s*
using *convexD-alt*[*OF assms*(*1*,*3*,*4*), *of u*] **using** *u* **by** *auto*
then show *False*
using *assms*(*5*)[*THEN bspec*[**where** *x=?z*]] **and** *u*(*3*)
by (*auto simp add*: *dist-commute algebra-simps*)
qed

lemma *any-closest-point-unique*:
fixes x :: *'a::real-inner*
assumes *convex s closed s* $x \in s$ $y \in s$
$\forall z{\in}s.$ *dist a x* $\leq$ *dist a z* $\forall z{\in}s.$ *dist a y* $\leq$ *dist a z*
shows $x = y$
using *any-closest-point-dot*[*OF assms*(*1*−*4*,*5*)] **and** *any-closest-point-dot*[*OF assms*(*1*−*2*,*4*,*3*,*6*)]
unfolding *norm-pths*(*1*) **and** *norm-le-square*
by (*auto simp add*: *algebra-simps*)

lemma *closest-point-unique*:
assumes *convex s closed s* $x \in s$ $\forall z{\in}s.$ *dist a x* $\leq$ *dist a z*
shows $x =$ *closest-point s a*
using *any-closest-point-unique*[*OF assms*(*1*−*3*) - *assms*(*4*), *of closest-point s a*]
using *closest-point-exists*[*OF assms*(*2*)] **and** *assms*(*3*) **by** *auto*

lemma *closest-point-dot*:
assumes *convex s closed s* $x \in s$
shows *inner* (*a* − *closest-point s a*) (*x* − *closest-point s a*) $\leq$ *0*
apply (*rule any-closest-point-dot*[*OF assms*(*1*,*2*) - *assms*(*3*)])
using *closest-point-exists*[*OF assms*(*2*)] **and** *assms*(*3*)
apply *auto*
done

lemma *closest-point-lt*:
assumes *convex s closed s* $x \in s$ $x \neq$ *closest-point s a*
shows *dist a* (*closest-point s a*) < *dist a x*
apply (*rule ccontr*)
apply (*rule-tac notE*[*OF assms*(*4*)])
apply (*rule closest-point-unique*[*OF assms*(*1*−*3*), *of a*])
using *closest-point-le*[*OF assms*(*2*), *of* - *a*]
apply *fastforce*
done

lemma *closest-point-lipschitz*:
assumes *convex s*
and *closed s s* $\neq \{\}$
shows *dist* (*closest-point s x*) (*closest-point s y*) $\leq$ *dist x y*
proof −
have *inner* (*x* − *closest-point s x*) (*closest-point s y* − *closest-point s x*) $\leq$ *0*
and *inner* (*y* − *closest-point s y*) (*closest-point s x* − *closest-point s y*) $\leq$ *0*
apply (*rule-tac*[!] *any-closest-point-dot*[*OF assms*(*1*−*2*)])
using *closest-point-exists*[*OF assms*(*2*−*3*)]
apply *auto*
done
then show *?thesis* **unfolding** *dist-norm* **and** *norm-le*
using *inner-ge-zero*[*of* (*x* − *closest-point s x*) − (*y* − *closest-point s y*)]
by (*simp add*: *inner-add inner-diff inner-commute*)
qed

lemma *continuous-at-closest-point*:
assumes *convex s*
and *closed s*
and $s \neq \{\}$
shows *continuous* (*at x*) (*closest-point s*)
unfolding *continuous-at-eps-delta*
using *le-less-trans*[*OF closest-point-lipschitz*[*OF assms*]] **by** *auto*

lemma *continuous-on-closest-point*:
assumes *convex s*
and *closed s*
and $s \neq \{\}$
shows *continuous-on t* (*closest-point s*)
by (*metis continuous-at-imp-continuous-on continuous-at-closest-point*[*OF assms*])

19.14.1 Various point-to-set separating/supporting hyperplane theorems.

lemma *supporting-hyperplane-closed-point*:
fixes *z* :: *'a*::{*real-inner*,*heine-borel*}
assumes *convex s*
and *closed s*
and $s \neq \{\}$
and $z \notin s$
shows $\exists\, a\ b.\ \exists\, y{\in}s.\ \mathit{inner}\ a\ z < b \wedge \mathit{inner}\ a\ y = b \wedge (\forall\, x{\in}s.\ \mathit{inner}\ a\ x \geq b)$
proof −
from *distance-attains-inf*[*OF assms*(*2*−*3*)]
obtain *y* **where** $y \in s$ **and** *y*: $\forall\, x{\in}s.\ \mathit{dist}\ z\ y \leq \mathit{dist}\ z\ x$
by *auto*
show *?thesis*
apply (*rule-tac x*=*y* − *z* **in** *exI*)
apply (*rule-tac x*=*inner* (*y* − *z*) *y* **in** *exI*)
apply (*rule-tac x*=*y* **in** *bexI*)
apply *rule*
defer
apply *rule*
defer
apply *rule*
apply (*rule ccontr*)
using ‹*y* ∈ *s*›
proof −
show *inner* (*y* − *z*) *z* < *inner* (*y* − *z*) *y*
apply (*subst diff-gt-0-iff-gt* [*symmetric*])
unfolding *inner-diff-right*[*symmetric*] **and** *inner-gt-zero-iff*
using ‹*y*∈*s*› ‹*z*∉*s*›
apply *auto*
done
next

```
    fix x
    assume x ∈ s
    have *: ∀u. 0 ≤ u ∧ u ≤ 1 ⟶ dist z y ≤ dist z ((1 - u) *R y + u *R x)
      using assms(1)[unfolded convex-alt] and y and ‹x∈s› and ‹y∈s› by auto
    assume ¬ inner (y - z) y ≤ inner (y - z) x
    then obtain v where v > 0 v ≤ 1 dist (y + v *R (x - y)) z < dist y z
      using closer-point-lemma[of z y x] by (auto simp add: inner-diff)
    then show False
    using *[THEN spec[where x=v]] by (auto simp add: dist-commute algebra-simps)
  qed auto
qed

lemma separating-hyperplane-closed-point:
  fixes z :: 'a::{real-inner,heine-borel}
  assumes convex s
    and closed s
    and z ∉ s
  shows ∃a b. inner a z < b ∧ (∀x∈s. inner a x > b)
proof (cases s = {})
  case True
  then show ?thesis
    apply (rule-tac x=-z in exI)
    apply (rule-tac x=1 in exI)
    using less-le-trans[OF - inner-ge-zero[of z]]
    apply auto
    done
next
  case False
  obtain y where y ∈ s and y: ∀x∈s. dist z y ≤ dist z x
    using distance-attains-inf[OF assms(2) False] by auto
  show ?thesis
    apply (rule-tac x=y - z in exI)
    apply (rule-tac x=inner (y - z) z + (norm (y - z))^2 / 2 in exI)
    apply rule
    defer
    apply rule
  proof -
    fix x
    assume x ∈ s
    have ¬ 0 < inner (z - y) (x - y)
      apply (rule notI)
      apply (drule closer-point-lemma)
    proof -
      assume ∃u>0. u ≤ 1 ∧ dist (y + u *R (x - y)) z < dist y z
      then obtain u where u > 0 u ≤ 1 dist (y + u *R (x - y)) z < dist y z
        by auto
      then show False using y[THEN bspec[where x=y + u *R (x - y)]]
        using assms(1)[unfolded convex-alt, THEN bspec[where x=y]]
        using ‹x∈s› ‹y∈s› by (auto simp add: dist-commute algebra-simps)
```

```
    qed
    moreover have 0 < (norm (y − z))²
      using ‹y∈s› ‹z∉s› by auto
    then have 0 < inner (y − z) (y − z)
      unfolding power2-norm-eq-inner by simp
    ultimately show inner (y − z) z + (norm (y − z))² / 2 < inner (y − z) x
      unfolding power2-norm-eq-inner and not-less
      by (auto simp add: field-simps inner-commute inner-diff)
  qed (insert ‹y∈s› ‹z∉s›, auto)
qed

lemma separating-hyperplane-closed-0:
  assumes convex (s::('a::euclidean-space) set)
    and closed s
    and 0 ∉ s
  shows ∃ a b. a ≠ 0 ∧ 0 < b ∧ (∀ x∈s. inner a x > b)
proof (cases s = {})
  case True
  have norm ((SOME i. i∈Basis)::'a) = 1 (SOME i. i∈Basis) ≠ (0::'a)
    defer
    apply (subst norm-le-zero-iff[symmetric])
    apply (auto simp: SOME-Basis)
    done
  then show ?thesis
    apply (rule-tac x=SOME i. i∈Basis in exI)
    apply (rule-tac x=1 in exI)
    using True using DIM-positive[where 'a='a]
    apply auto
    done
next
  case False
  then show ?thesis
    using False using separating-hyperplane-closed-point[OF assms]
    apply (elim exE)
    unfolding inner-zero-right
    apply (rule-tac x=a in exI)
    apply (rule-tac x=b in exI)
    apply auto
    done
qed
```

19.14.2 Now set-to-set for closed/compact sets

```
lemma separating-hyperplane-closed-compact:
  fixes s :: 'a::euclidean-space set
  assumes convex s
    and closed s
    and convex t
    and compact t
```

```
    and t ≠ {}
    and s ∩ t = {}
  shows ∃a b. (∀x∈s. inner a x < b) ∧ (∀x∈t. inner a x > b)
proof (cases s = {})
  case True
  obtain b where b: b > 0 ∀x∈t. norm x ≤ b
    using compact-imp-bounded[OF assms(4)] unfolding bounded-pos by auto
  obtain z :: 'a where z: norm z = b + 1
    using vector-choose-size[of b + 1] and b(1) by auto
  then have z ∉ t using b(2)[THEN bspec[where x=z]] by auto
  then obtain a b where ab: inner a z < b ∀x∈t. b < inner a x
    using separating-hyperplane-closed-point[OF assms(3) compact-imp-closed[OF
assms(4)], of z]
    by auto
  then show ?thesis
    using True by auto
next
  case False
  then obtain y where y ∈ s by auto
  obtain a b where 0 < b ∀x∈{x - y |x y. x ∈ s ∧ y ∈ t}. b < inner a x
   using separating-hyperplane-closed-point[OF convex-differences[OF assms(1,3)],
of 0]
    using closed-compact-differences[OF assms(2,4)]
    using assms(6) by auto blast
  then have ab: ∀x∈s. ∀y∈t. b + inner a y < inner a x
    apply -
    apply rule
    apply rule
    apply (erule-tac x=x - y in ballE)
    apply (auto simp add: inner-diff)
    done
  def k ≡ SUP x:t. a • x
  show ?thesis
    apply (rule-tac x=-a in exI)
    apply (rule-tac x=-(k + b / 2) in exI)
    apply (intro conjI ballI)
    unfolding inner-minus-left and neg-less-iff-less
  proof -
    fix x assume x ∈ t
    then have inner a x - b / 2 < k
      unfolding k-def
    proof (subst less-cSUP-iff)
      show t ≠ {} by fact
      show bdd-above (op • a ' t)
        using ab[rule-format, of y] ‹y ∈ s›
        by (intro bdd-aboveI2[where M=inner a y - b]) (auto simp: field-simps
intro: less-imp-le)
    qed (auto intro!: bexI[of - x] ‹0<b›)
    then show inner a x < k + b / 2
```

```
    by auto
  next
    fix x
    assume x ∈ s
    then have k ≤ inner a x − b
      unfolding k-def
      apply (rule-tac cSUP-least)
      using assms(5)
      using ab[THEN bspec[where x=x]]
      apply auto
      done
    then show k + b / 2 < inner a x
      using ‹0 < b› by auto
  qed
qed

lemma separating-hyperplane-compact-closed:
  fixes s :: 'a::euclidean-space set
  assumes convex s
    and compact s
    and s ≠ {}
    and convex t
    and closed t
    and s ∩ t = {}
  shows ∃ a b. (∀ x∈s. inner a x < b) ∧ (∀ x∈t. inner a x > b)
proof −
  obtain a b where (∀ x∈t. inner a x < b) ∧ (∀ x∈s. b < inner a x)
   using separating-hyperplane-closed-compact[OF assms(4−5,1−2,3)] and assms(6)
    by auto
  then show ?thesis
    apply (rule-tac x=−a in exI)
    apply (rule-tac x=−b in exI)
    apply auto
    done
qed
```

19.14.3 General case without assuming closure and getting non-strict separation

```
lemma separating-hyperplane-set-0:
  assumes convex s (0::'a::euclidean-space) ∉ s
  shows ∃ a. a ≠ 0 ∧ (∀ x∈s. 0 ≤ inner a x)
proof −
  let ?k = λc. {x::'a. 0 ≤ inner c x}
  have frontier (cball 0 1) ∩ (⋂ (?k ` s)) ≠ {}
    apply (rule compact-imp-fip)
    apply (rule compact-frontier[OF compact-cball])
    defer
    apply rule
```

```
    apply rule
    apply (erule conjE)
  proof −
    fix f
    assume as: f ⊆ ?k ‘ s finite f
    obtain c where c: f = ?k ‘ c c ⊆ s finite c
      using finite-subset-image[OF as(2,1)] by auto
    then obtain a b where ab: a ≠ 0 0 < b ∀ x∈convex hull c. b < inner a x
      using separating-hyperplane-closed-0[OF convex-convex-hull, of c]
       using finite-imp-compact-convex-hull[OF c(3), THEN compact-imp-closed]
and assms(2)
      using subset-hull[of convex, OF assms(1), symmetric, of c]
      by force
    then have ∃ x. norm x = 1 ∧ (∀ y∈c. 0 ≤ inner y x)
      apply (rule-tac x = inverse(norm a) *R a in exI)
      using hull-subset[of c convex]
      unfolding subset-eq and inner-scaleR
      by (auto simp add: inner-commute del: ballE elim!: ballE)
    then show frontier (cball 0 1) ∩ ⋂f ≠ {}
      unfolding c(1) frontier-cball dist-norm by auto
  qed (insert closed-halfspace-ge, auto)
  then obtain x where norm x = 1 ∀ y∈s. x∈?k y
    unfolding frontier-cball dist-norm by auto
  then show ?thesis
    apply (rule-tac x=x in exI)
    apply (auto simp add: inner-commute)
    done
qed

lemma separating-hyperplane-sets:
  fixes s t :: 'a::euclidean-space set
  assumes convex s
    and convex t
    and s ≠ {}
    and t ≠ {}
    and s ∩ t = {}
  shows ∃ a b. a ≠ 0 ∧ (∀ x∈s. inner a x ≤ b) ∧ (∀ x∈t. inner a x ≥ b)
proof −
  from separating-hyperplane-set-0[OF convex-differences[OF assms(2,1)]]
  obtain a where a ≠ 0 ∀ x∈{x − y |x y. x ∈ t ∧ y ∈ s}. 0 ≤ inner a x
    using assms(3−5) by fastforce
  then have *: ⋀x y. x ∈ t ⟹ y ∈ s ⟹ inner a y ≤ inner a x
    by (force simp add: inner-diff)
  then have bdd: bdd-above ((op · a)‘s)
    using ‹t ≠ {}› by (auto intro: bdd-aboveI2[OF *])
  show ?thesis
    using ‹a≠0›
    by (intro exI[of - a] exI[of - SUP x:s. a · x])
       (auto intro!: cSUP-upper bdd cSUP-least ‹a ≠ 0› ‹s ≠ {}› *)
```

qed

19.15 More convexity generalities

lemma *convex-closure*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s*
shows *convex* (*closure s*)
apply (*rule convexI*)
apply (*unfold closure-sequential*, *elim exE*)
apply (*rule-tac x*=λ*n*. *u* ∗$_R$ *xa n* + *v* ∗$_R$ *xb n* **in** *exI*)
apply (*rule*,*rule*)
apply (*rule convexD* [*OF assms*])
apply (*auto del*: *tendsto-const intro*!: *tendsto-intros*)
done

lemma *convex-interior*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s*
shows *convex* (*interior s*)
unfolding *convex-alt Ball-def mem-interior*
apply (*rule*,*rule*,*rule*,*rule*,*rule*,*rule*)
apply (*elim exE conjE*)
proof −
fix *x y u*
assume *u*: *0* ≤ *u u* ≤ (*1*::*real*)
fix *e d*
assume *ed*: *ball x e* ⊆ *s ball y d* ⊆ *s 0*<*d 0*<*e*
show ∃ *e*>*0*. *ball* ((*1* − *u*) ∗$_R$ *x* + *u* ∗$_R$ *y*) *e* ⊆ *s*
apply (*rule-tac x*=*min d e* **in** *exI*)
apply *rule*
unfolding *subset-eq*
defer
apply *rule*
proof −
fix *z*
assume *z* ∈ *ball* ((*1* − *u*) ∗$_R$ *x* + *u* ∗$_R$ *y*) (*min d e*)
then have (*1*− *u*) ∗$_R$ (*z* − *u* ∗$_R$ (*y* − *x*)) + *u* ∗$_R$ (*z* + (*1* − *u*) ∗$_R$ (*y* − *x*))
∈ *s*
apply (*rule-tac assms*[*unfolded convex-alt*, *rule-format*])
using *ed*(*1*,*2*) **and** *u*
unfolding *subset-eq mem-ball Ball-def dist-norm*
apply (*auto simp add*: *algebra-simps*)
done
then show *z* ∈ *s*
using *u* **by** (*auto simp add*: *algebra-simps*)
qed(*insert u ed*(*3*−*4*), *auto*)
qed

lemma *convex-hull-eq-empty*[*simp*]: *convex hull s* = {} ⟷ *s* = {}
using *hull-subset*[*of s convex*] *convex-hull-empty* **by** *auto*

19.16 Moving and scaling convex hulls.

lemma *convex-hull-set-plus*:
convex hull (*s* + *t*) = *convex hull s* + *convex hull t*
unfolding *set-plus-image*
apply (*subst convex-hull-linear-image* [*symmetric*])
apply (*simp add*: *linear-iff scaleR-right-distrib*)
apply (*simp add*: *convex-hull-Times*)
done

lemma *translation-eq-singleton-plus*: (λ*x*. *a* + *x*) ‘ *t* = {*a*} + *t*
unfolding *set-plus-def* **by** *auto*

lemma *convex-hull-translation*:
convex hull ((λ*x*. *a* + *x*) ‘ *s*) = (λ*x*. *a* + *x*) ‘ (*convex hull s*)
unfolding *translation-eq-singleton-plus*
by (*simp only*: *convex-hull-set-plus convex-hull-singleton*)

lemma *convex-hull-scaling*:
convex hull ((λ*x*. *c* ∗*R* *x*) ‘ *s*) = (λ*x*. *c* ∗*R* *x*) ‘ (*convex hull s*)
using *linear-scaleR* **by** (*rule convex-hull-linear-image* [*symmetric*])

lemma *convex-hull-affinity*:
convex hull ((λ*x*. *a* + *c* ∗*R* *x*) ‘ *s*) = (λ*x*. *a* + *c* ∗*R* *x*) ‘ (*convex hull s*)
by(*simp only*: *image-image*[*symmetric*] *convex-hull-scaling convex-hull-translation*)

19.17 Convexity of cone hulls

lemma *convex-cone-hull*:
assumes *convex S*
shows *convex* (*cone hull S*)
proof (*rule convexI*)
fix *x y*
assume *xy*: *x* ∈ *cone hull S y* ∈ *cone hull S*
then have *S* ≠ {}
using *cone-hull-empty-iff*[*of S*] **by** *auto*
fix *u v* :: *real*
assume *uv*: *u* ≥ *0 v* ≥ *0 u* + *v* = *1*
then have ∗: *u* ∗*R* *x* ∈ *cone hull S v* ∗*R* *y* ∈ *cone hull S*
using *cone-cone-hull*[*of S*] *xy cone-def*[*of cone hull S*] **by** *auto*
from ∗ **obtain** *cx* :: *real* **and** *xx* **where** *x*: *u* ∗*R* *x* = *cx* ∗*R* *xx cx* ≥ *0 xx* ∈ *S*
using *cone-hull-expl*[*of S*] **by** *auto*
from ∗ **obtain** *cy* :: *real* **and** *yy* **where** *y*: *v* ∗*R* *y* = *cy* ∗*R* *yy cy* ≥ *0 yy* ∈ *S*
using *cone-hull-expl*[*of S*] **by** *auto*
{
assume *cx* + *cy* ≤ *0*
then have *u* ∗*R* *x* = *0* **and** *v* ∗*R* *y* = *0*

```
    using x y by auto
  then have u *R x + v *R y = 0
    by auto
  then have u *R x + v *R y ∈ cone hull S
    using cone-hull-contains-0[of S] ‹S ≠ {}› by auto
 }
 moreover
 {
  assume cx + cy > 0
  then have (cx / (cx + cy)) *R xx + (cy / (cx + cy)) *R yy ∈ S
    using assms mem-convex-alt[of S xx yy cx cy] x y by auto
  then have cx *R xx + cy *R yy ∈ cone hull S
   using mem-cone-hull[of (cx/(cx+cy)) *R xx + (cy/(cx+cy)) *R yy S cx+cy]
‹cx+cy>0›
    by (auto simp add: scaleR-right-distrib)
  then have u *R x + v *R y ∈ cone hull S
    using x y by auto
 }
 moreover have cx + cy ≤ 0 ∨ cx + cy > 0 by auto
 ultimately show u *R x + v *R y ∈ cone hull S by blast
qed

lemma cone-convex-hull:
 assumes cone S
 shows cone (convex hull S)
proof (cases S = {})
 case True
 then show ?thesis by auto
next
 case False
 then have *: 0 ∈ S ∧ (∀ c. c > 0 ⟶ op *R c ‘ S = S)
   using cone-iff[of S] assms by auto
 {
  fix c :: real
  assume c > 0
  then have op *R c ‘ (convex hull S) = convex hull (op *R c ‘ S)
    using convex-hull-scaling[of - S] by auto
  also have ... = convex hull S
    using * ‹c > 0› by auto
  finally have op *R c ‘ (convex hull S) = convex hull S
    by auto
 }
 then have 0 ∈ convex hull S ⋀c. c > 0 ⟹ (op *R c ‘ (convex hull S)) =
(convex hull S)
   using * hull-subset[of S convex] by auto
 then show ?thesis
   using ‹S ≠ {}› cone-iff[of convex hull S] by auto
qed
```

19.18 Convex set as intersection of halfspaces

lemma *convex-halfspace-intersection*:
fixes $s :: ('a::euclidean\text{-}space)\ set$
assumes *closed s convex s*
shows $s = \bigcap \{h.\ s \subseteq h \wedge (\exists\, a\ b.\ h = \{x.\ inner\ a\ x \leq b\})\}$
apply (*rule set-eqI*)
apply *rule*
unfolding *Inter-iff Ball-def mem-Collect-eq*
apply (*rule*,*rule*,*erule conjE*)
proof −
fix *x*
assume $\forall\, xa.\ s \subseteq xa \wedge (\exists\, a\ b.\ xa = \{x.\ inner\ a\ x \leq b\}) \longrightarrow x \in xa$
then have $\forall\, a\ b.\ s \subseteq \{x.\ inner\ a\ x \leq b\} \longrightarrow x \in \{x.\ inner\ a\ x \leq b\}$
by *blast*
then show $x \in s$
apply (*rule-tac ccontr*)
apply (*drule separating-hyperplane-closed-point*[*OF assms*(*2*,*1*)])
apply (*erule exE*)+
apply (*erule-tac x*=−*a* **in** *allE*)
apply (*erule-tac x*=−*b* **in** *allE*)
apply *auto*
done
qed *auto*

19.19 Radon's theorem (from Lars Schewe)

lemma *radon-ex-lemma*:
assumes *finite c affine-dependent c*
shows $\exists\, u.\ setsum\ u\ c = 0 \wedge (\exists\, v{\in}c.\ u\ v \neq 0) \wedge setsum\ (\lambda v.\ u\ v *_R v)\ c = 0$
proof −
from *assms*(*2*)[*unfolded affine-dependent-explicit*]
obtain *s u* **where**
finite s s ⊆ *c setsum u s* = *0* ∃*v*∈*s*. *u v* ≠ *0* (∑*v*∈*s*. *u v* ∗*R v*) = *0*
by *blast*
then show *?thesis*
apply (*rule-tac x*=λ*v*. *if v*∈*s then u v else 0* **in** *exI*)
unfolding *if-smult scaleR-zero-left* **and** *setsum.inter-restrict*[*OF assms*(*1*), *symmetric*]
apply (*auto simp add*: *Int-absorb1*)
done
qed

lemma *radon-s-lemma*:
assumes *finite s*
and *setsum f s* = (*0*::*real*)
shows *setsum f* {*x*∈*s*. *0* < *f x*} = − *setsum f* {*x*∈*s*. *f x* < *0*}
proof −
have ∗: ⋀*x*. (*if f x* < *0 then f x else 0*) + (*if 0* < *f x then f x else 0*) = *f x*
by *auto*

```
  show ?thesis
    unfolding add-eq-0-iff[symmetric] and setsum.inter-filter[OF assms(1)]
      and setsum.distrib[symmetric] and *
    using assms(2)
    by assumption
qed

lemma radon-v-lemma:
  assumes finite s
    and setsum f s = 0
    and ∀x. g x = (0::real) ⟶ f x = (0::'a::euclidean-space)
  shows (setsum f {x∈s. 0 < g x}) = - setsum f {x∈s. g x < 0}
proof -
  have *: ⋀x. (if 0 < g x then f x else 0) + (if g x < 0 then f x else 0) = f x
    using assms(3) by auto
  show ?thesis
    unfolding eq-neg-iff-add-eq-0 and setsum.inter-filter[OF assms(1)]
      and setsum.distrib[symmetric] and *
    using assms(2)
    apply assumption
    done
qed

lemma radon-partition:
  assumes finite c affine-dependent c
  shows ∃m p. m ∩ p = {} ∧ m ∪ p = c ∧ (convex hull m) ∩ (convex hull p) ≠
{}
proof -
  obtain u v where uv: setsum u c = 0 v∈c u v ≠ 0  (∑v∈c. u v *R v) = 0
    using radon-ex-lemma[OF assms] by auto
  have fin: finite {x ∈ c. 0 < u x} finite {x ∈ c. 0 > u x}
    using assms(1) by auto
  def z ≡ inverse (setsum u {x∈c. u x > 0}) *R setsum (λx. u x *R x) {x∈c. u
x > 0}
  have setsum u {x ∈ c. 0 < u x} ≠ 0
  proof (cases u v ≥ 0)
    case False
    then have u v < 0 by auto
    then show ?thesis
    proof (cases ∃w∈{x ∈ c. 0 < u x}. u w > 0)
      case True
      then show ?thesis
        using setsum-nonneg-eq-0-iff[of - u, OF fin(1)] by auto
    next
      case False
      then have setsum u c ≤ setsum (λx. if x=v then u v else 0) c
        apply (rule-tac setsum-mono)
        apply auto
        done
```

then show *?thesis*
unfolding *setsum.delta*[*OF assms*(*1*)] **using** *uv*(*2*) **and** ‹*u v* < *0*› **and** *uv*(*1*) **by** *auto*
qed
qed (*insert setsum-nonneg-eq-0-iff*[*of - u, OF fin*(*1*)] *uv*(*2*−*3*), *auto*)

then have ∗: *setsum u* {*x*∈*c*. *u x* > *0*} > *0*
unfolding *less-le*
apply (*rule-tac conjI*)
apply (*rule-tac setsum-nonneg*)
apply *auto*
done
moreover have *setsum u* ({*x* ∈ *c*. *0* < *u x*} ∪ {*x* ∈ *c*. *u x* < *0*}) = *setsum u c*
(∑ *x*∈{*x* ∈ *c*. *0* < *u x*} ∪ {*x* ∈ *c*. *u x* < *0*}. *u x* ∗$_R$ *x*) = (∑ *x*∈*c*. *u x* ∗$_R$ *x*)
using *assms*(*1*)
apply (*rule-tac*[!] *setsum.mono-neutral-left*)
apply *auto*
done
then have *setsum u* {*x* ∈ *c*. *0* < *u x*} = − *setsum u* {*x* ∈ *c*. *0* > *u x*}
(∑ *x*∈{*x* ∈ *c*. *0* < *u x*}. *u x* ∗$_R$ *x*) = − (∑ *x*∈{*x* ∈ *c*. *0* > *u x*}. *u x* ∗$_R$ *x*)
unfolding *eq-neg-iff-add-eq-0*
using *uv*(*1*,*4*)
by (*auto simp add*: *setsum.union-inter-neutral*[*OF fin*, *symmetric*])
moreover have ∀ *x*∈{*v* ∈ *c*. *u v* < *0*}. *0* ≤ *inverse* (*setsum u* {*x* ∈ *c*. *0* < *u x*}) ∗ − *u x*
apply *rule*
apply (*rule mult-nonneg-nonneg*)
using ∗
apply *auto*
done
ultimately have *z* ∈ *convex hull* {*v* ∈ *c*. *u v* ≤ *0*}
unfolding *convex-hull-explicit mem-Collect-eq*
apply (*rule-tac x*={*v* ∈ *c*. *u v* < *0*} **in** *exI*)
apply (*rule-tac x*=λ*y*. *inverse* (*setsum u* {*x*∈*c*. *u x* > *0*}) ∗ − *u y* **in** *exI*)
using *assms*(*1*) **unfolding** *scaleR-scaleR*[*symmetric*] *scaleR-right.setsum* [*symmetric*] **and** *z-def*
apply (*auto simp add*: *setsum-negf setsum-right-distrib*[*symmetric*])
done
moreover have ∀ *x*∈{*v* ∈ *c*. *0* < *u v*}. *0* ≤ *inverse* (*setsum u* {*x* ∈ *c*. *0* < *u x*}) ∗ *u x*
apply *rule*
apply (*rule mult-nonneg-nonneg*)
using ∗
apply *auto*
done
then have *z* ∈ *convex hull* {*v* ∈ *c*. *u v* > *0*}
unfolding *convex-hull-explicit mem-Collect-eq*
apply (*rule-tac x*={*v* ∈ *c*. *0* < *u v*} **in** *exI*)
apply (*rule-tac x*=λ*y*. *inverse* (*setsum u* {*x*∈*c*. *u x* > *0*}) ∗ *u y* **in** *exI*)

```
    using assms(1)
   unfolding scaleR-scaleR[symmetric] scaleR-right.setsum [symmetric] and z-def
    using ∗
    apply (auto simp add: setsum-negf setsum-right-distrib[symmetric])
    done
  ultimately show ?thesis
    apply (rule-tac x={v∈c. u v ≤ 0} in exI)
    apply (rule-tac x={v∈c. u v > 0} in exI)
    apply auto
    done
qed

lemma radon:
  assumes affine-dependent c
  obtains m p where m ⊆ c p ⊆ c m ∩ p = {} (convex hull m) ∩ (convex hull
p) ≠ {}
proof −
  from assms[unfolded affine-dependent-explicit]
  obtain s u where
      finite s s ⊆ c setsum u s = 0 ∃ v∈s. u v ≠ 0 (∑ v∈s. u v ∗R v) = 0
    by blast
  then have ∗: finite s affine-dependent s and s: s ⊆ c
    unfolding affine-dependent-explicit by auto
  from radon-partition[OF ∗]
  obtain m p where m ∩ p = {} m ∪ p = s convex hull m ∩ convex hull p ≠ {}
    by blast
  then show ?thesis
    apply (rule-tac that[of p m])
    using s
    apply auto
    done
qed
```

19.20 Helly's theorem

```
lemma helly-induct:
  fixes f :: 'a::euclidean-space set set
  assumes card f = n
    and n ≥ DIM('a) + 1
    and ∀ s∈f. convex s ∀ t⊆f. card t = DIM('a) + 1 ⟶ ⋂ t ≠ {}
  shows ⋂ f ≠ {}
  using assms
proof (induct n arbitrary: f)
  case 0
  then show ?case by auto
next
  case (Suc n)
  have finite f
    using ‹card f = Suc n› by (auto intro: card-ge-0-finite)
```

```
  show ⋂f ≠ {}
    apply (cases n = DIM('a))
    apply (rule Suc(5)[rule-format])
    unfolding ‹card f = Suc n›
  proof −
    assume ng: n ≠ DIM('a)
    then have ∃X. ∀s∈f. X s ∈ ⋂(f − {s})
      apply (rule-tac bchoice)
      unfolding ex-in-conv
      apply (rule, rule Suc(1)[rule-format])
      unfolding card-Diff-singleton-if[OF ‹finite f›] ‹card f = Suc n›
      defer
      defer
      apply (rule Suc(4)[rule-format])
      defer
      apply (rule Suc(5)[rule-format])
      using Suc(3) ‹finite f›
      apply auto
      done
    then obtain X where X: ∀s∈f. X s ∈ ⋂(f − {s}) by auto
    show ?thesis
    proof (cases inj-on X f)
      case False
      then obtain s t where st: s≠t s∈f t∈f X s = X t
        unfolding inj-on-def by auto
      then have ∗: ⋂f = ⋂(f − {s}) ∩ ⋂(f − {t}) by auto
      show ?thesis
        unfolding ∗
        unfolding ex-in-conv[symmetric]
        apply (rule-tac x=X s in exI)
        apply rule
        apply (rule X[rule-format])
        using X st
        apply auto
        done
    next
      case True
      then obtain m p where mp: m ∩ p = {} m ∪ p = X ‘ f convex hull m ∩
convex hull p ≠ {}
        using radon-partition[of X ‘ f] and affine-dependent-biggerset[of X ‘ f]
        unfolding card-image[OF True] and ‹card f = Suc n›
        using Suc(3) ‹finite f› and ng
        by auto
      have m ⊆ X ‘ f p ⊆ X ‘ f
        using mp(2) by auto
      then obtain g h where gh:m = X ‘ g p = X ‘ h g ⊆ f h ⊆ f
        unfolding subset-image-iff by auto
      then have f ∪ (g ∪ h) = f by auto
      then have f: f = g ∪ h
```

```
      using inj-on-Un-image-eq-iff[of X f g ∪ h] and True
      unfolding mp(2)[unfolded image-Un[symmetric] gh]
      by auto
    have ∗: g ∩ h = {}
      using mp(1)
      unfolding gh
      using inj-on-image-Int[OF True gh(3,4)]
      by auto
    have convex hull (X ‘ h) ⊆ ⋂ g convex hull (X ‘ g) ⊆ ⋂ h
      apply (rule-tac [!] hull-minimal)
      using Suc gh(3−4)
      unfolding subset-eq
      apply (rule-tac [2] convex-Inter, rule-tac [4] convex-Inter)
      apply rule
      prefer 3
      apply rule
    proof −
      fix x
      assume x ∈ X ‘ g
      then obtain y where y ∈ g x = X y
        unfolding image-iff ..
      then show x ∈ ⋂ h
        using X[THEN bspec[where x=y]] using ∗ f by auto
    next
      fix x
      assume x ∈ X ‘ h
      then obtain y where y ∈ h x = X y
        unfolding image-iff ..
      then show x ∈ ⋂ g
        using X[THEN bspec[where x=y]] using ∗ f by auto
    qed auto
    then show ?thesis
      unfolding f using mp(3)[unfolded gh] by blast
  qed
 qed auto
qed

lemma helly:
 fixes f :: 'a::euclidean-space set set
 assumes card f ≥ DIM('a) + 1 ∀ s∈f. convex s
  and ∀ t⊆f. card t = DIM('a) + 1 ⟶ ⋂ t ≠ {}
 shows ⋂ f ≠ {}
 apply (rule helly-induct)
 using assms
 apply auto
 done
```

19.21 Homeomorphism of all convex compact sets with nonempty interior

lemma *compact-frontier-line-lemma*:
fixes $s :: 'a::euclidean\text{-}space\ set$
assumes *compact s*
and $0 \in s$
and $x \neq 0$
obtains u **where** $0 \leq u$ **and** $(u *_R x) \in frontier\ s\ \forall v{>}u.\ (v *_R x) \notin s$
proof −
obtain b **where** $b{:}\ b > 0\ \forall x{\in}s.\ norm\ x \leq b$
using *compact-imp-bounded*[*OF assms*(1), *unfolded bounded-pos*] **by** *auto*
let $?A = \{y.\ \exists u.\ 0 \leq u \wedge u \leq b\ /\ norm(x) \wedge (y = u *_R x)\}$
have $A{:}\ ?A = (\lambda u.\ u *_R x)\ `\ \{0\ ..\ b\ /\ norm\ x\}$
by *auto*
have $*{:}\ \bigwedge x\ A\ B.\ x{\in}A \Longrightarrow x{\in}B \Longrightarrow A{\cap}B \neq \{\}$ **by** *blast*
have *compact ?A*
unfolding *A*
apply (*rule compact-continuous-image*)
apply (*rule continuous-at-imp-continuous-on*)
apply *rule*
apply (*intro continuous-intros*)
apply (*rule compact-Icc*)
done
moreover have $\{y.\ \exists u{\geq}0.\ u \leq b\ /\ norm\ x \wedge y = u *_R x\} \cap s \neq \{\}$
apply(*rule* $*$[*OF - assms*(2)])
unfolding *mem-Collect-eq*
using ‹$b > 0$› *assms*(3)
apply *auto*
done
ultimately obtain $u\ y$ **where** *obt*: $u{\geq}0\ u \leq b\ /\ norm\ x\ y = u *_R x$
$y \in ?A\ y \in s\ \forall z{\in}?A \cap s.\ dist\ 0\ z \leq dist\ 0\ y$
using *distance-attains-sup*[*OF compact-inter*[*OF - assms*(1), *of ?A*], *of 0*]
by *auto*

have $norm\ x > 0$
using *assms*(3)[*unfolded zero-less-norm-iff*[*symmetric*]] **by** *auto*
{
fix v
assume *as*: $v > u\ v *_R x \in s$
then have $v \leq b\ /\ norm\ x$
using *b*(2)[*rule-format*, *OF as*(2)]
using ‹$u{\geq}0$›
unfolding *pos-le-divide-eq*[*OF* ‹$norm\ x > 0$›]
by *auto*
then have $norm\ (v *_R x) \leq norm\ y$
apply (*rule-tac obt*(6)[*rule-format*, *unfolded dist-0-norm*])
apply (*rule IntI*)
defer
apply (*rule as*(2))

```
      unfolding mem-Collect-eq
      apply (rule-tac x=v in exI)
      using as(1) ‹u≥0›
      apply (auto simp add: field-simps)
      done
    then have False
      unfolding obt(3) using ‹u≥0› ‹norm x > 0› ‹v > u›
      by (auto simp add:field-simps)
  } note u-max = this

  have u *R x ∈ frontier s
    unfolding frontier-straddle
    apply (rule,rule,rule)
    apply (rule-tac x=u *R x in bexI)
    unfolding obt(3)[symmetric]
    prefer 3
    apply (rule-tac x=(u + (e / 2) / norm x) *R x in exI)
    apply (rule, rule)
  proof −
    fix e
    assume e > 0 and as: (u + e / 2 / norm x) *R x ∈ s
    then have u + e / 2 / norm x > u
      using ‹norm x > 0› by (auto simp del:zero-less-norm-iff)
    then show False using u-max[OF - as] by auto
  qed (insert ‹y∈s›, auto simp add: dist-norm scaleR-left-distrib obt(3))
  then show ?thesis by(metis that[of u] u-max obt(1))
qed

lemma starlike-compact-projective:
  assumes compact s
    and cball (0::'a::euclidean-space) 1 ⊆ s
    and ∀x∈s. ∀u. 0 ≤ u ∧ u < 1 ⟶ u *R x ∈ s − frontier s
  shows s homeomorphic (cball (0::'a::euclidean-space) 1)
proof −
  have fs: frontier s ⊆ s
    apply (rule frontier-subset-closed)
    using compact-imp-closed[OF assms(1)]
    apply simp
    done
  def pi ≡ λx::'a. inverse (norm x) *R x
  have 0 ∉ frontier s
    unfolding frontier-straddle
    apply (rule notI)
    apply (erule-tac x=1 in allE)
    using assms(2)[unfolded subset-eq Ball-def mem-cball]
    apply auto
    done
  have injpi: ⋀x y. pi x = pi y ∧ norm x = norm y ⟷ x = y
    unfolding pi-def by auto
```

have *contpi*: *continuous-on* (*UNIV* − {*0*}) *pi*
apply (*rule continuous-at-imp-continuous-on*)
apply *rule* **unfolding** *pi-def*
apply (*intro continuous-intros*)
apply *simp*
done
def *sphere* ≡ {*x*::′*a*. *norm x* = *1*}
have *pi*: ⋀*x*. *x* ≠ *0* ⟹ *pi x* ∈ *sphere* ⋀*x u*. *u*>*0* ⟹ *pi* (*u* ∗*R x*) = *pi x*
unfolding *pi-def sphere-def* **by** *auto*

have *0* ∈ *s*
using *assms*(*2*) **and** *centre-in-cball*[*of 0 1*] **by** *auto*
have *front-smul*: ∀ *x*∈*frontier s*. ∀ *u*≥*0*. *u* ∗*R x* ∈ *s* ⟷ *u* ≤ *1*
proof (*rule*,*rule*,*rule*)
fix *x* **and** *u* :: *real*
assume *x*: *x* ∈ *frontier s* **and** *0* ≤ *u*
then have *x* ≠ *0*
using ‹*0* ∉ *frontier s*› **by** *auto*
obtain *v* **where** *v*: *0* ≤ *v v* ∗*R x* ∈ *frontier s* ∀ *w*>*v*. *w* ∗*R x* ∉ *s*
using *compact-frontier-line-lemma*[*OF assms*(*1*) ‹*0*∈*s*› ‹*x*≠*0*›] **by** *auto*
have *v* = *1*
apply (*rule ccontr*)
unfolding *neq-iff*
apply (*erule disjE*)
proof −
assume *v* < *1*
then show *False*
using *v*(*3*)[*THEN spec*[**where** *x*=*1*]] **using** *x* **and** *fs* **by** *auto*
next
assume *v* > *1*
then show *False*
using *assms*(*3*)[*THEN bspec*[**where** *x*=*v* ∗*R x*], *THEN spec*[**where** *x*=*inverse v*]]
using *v* **and** *x* **and** *fs*
unfolding *inverse-less-1-iff* **by** *auto*
qed
show *u* ∗*R x* ∈ *s* ⟷ *u* ≤ *1*
apply *rule*
using *v*(*3*)[*unfolded* ‹*v*=*1*›, *THEN spec*[**where** *x*=*u*]]
proof −
assume *u* ≤ *1*
then show *u* ∗*R x* ∈ *s*
apply (*cases u* = *1*)
using *assms*(*3*)[*THEN bspec*[**where** *x*=*x*], *THEN spec*[**where** *x*=*u*]]
using ‹*0*≤*u*› **and** *x* **and** *fs*
apply *auto*
done
qed *auto*

qed

have $\exists$ *surf*. *homeomorphism* (*frontier s*) *sphere pi surf*
apply (*rule homeomorphism-compact*)
apply (*rule compact-frontier*[*OF assms*(*1*)])
apply (*rule continuous-on-subset*[*OF contpi*])
defer
apply (*rule set-eqI*)
apply *rule*
unfolding *inj-on-def*
prefer *3*
apply(*rule*,*rule*,*rule*)
proof −
fix *x*
assume *x* ∈ *pi* ' *frontier s*
then obtain *y* **where** *y* ∈ *frontier s x* = *pi y* **by** *auto*
then show *x* ∈ *sphere*
using *pi*(*1*)[*of y*] **and** ‹*0* ∉ *frontier s*› **by** *auto*
next
fix *x*
assume *x* ∈ *sphere*
then have *norm x* = *1 x* ≠ *0*
unfolding *sphere-def* **by** *auto*
then obtain *u* **where** *0* ≤ *u u* ∗$_R$ *x* ∈ *frontier s* ∀ *v*>*u*. *v* ∗$_R$ *x* ∉ *s*
using *compact-frontier-line-lemma*[*OF assms*(*1*) ‹*0*∈*s*›, *of x*] **by** *auto*
then show *x* ∈ *pi* ' *frontier s*
unfolding *image-iff le-less pi-def*
apply (*rule-tac x*=*u* ∗$_R$ *x* **in** *bexI*)
using ‹*norm x* = *1*› ‹*0* ∉ *frontier s*›
apply *auto*
done
next
fix *x y*
assume *as*: *x* ∈ *frontier s y* ∈ *frontier s pi x* = *pi y*
then have *xys*: *x* ∈ *s y* ∈ *s*
using *fs* **by** *auto*
from *as*(*1*,*2*) **have** *nor*: *norm x* ≠ *0 norm y* ≠ *0*
using ‹*0*∉*frontier s*› **by** *auto*
from *nor* **have** *x*: *x* = *norm x* ∗$_R$ ((*inverse* (*norm y*)) ∗$_R$ *y*)
unfolding *as*(*3*)[*unfolded pi-def*, *symmetric*] **by** *auto*
from *nor* **have** *y*: *y* = *norm y* ∗$_R$ ((*inverse* (*norm x*)) ∗$_R$ *x*)
unfolding *as*(*3*)[*unfolded pi-def*] **by** *auto*
have *0* ≤ *norm y* ∗ *inverse* (*norm x*) **and** *0* ≤ *norm x* ∗ *inverse* (*norm y*)
using *nor*
apply *auto*
done
then have *norm x* = *norm y*
apply −
apply (*rule ccontr*)

```
    unfolding neq-iff
   using x y and front-smul[THEN bspec, OF as(1), THEN spec[where x=norm
y * (inverse (norm x))]]
    using front-smul[THEN bspec, OF as(2), THEN spec[where x=norm x *
(inverse (norm y))]]
    using xys nor
    apply (auto simp add: field-simps)
    done
  then show x = y
    apply (subst injpi[symmetric])
    using as(3)
    apply auto
    done
 qed (insert ‹0 ∉ frontier s›, auto)
 then obtain surf where
   surf: ∀x∈frontier s. surf (pi x) = x  pi ' frontier s = sphere continuous-on
(frontier s) pi
  ∀y∈sphere. pi (surf y) = y surf ' sphere = frontier s continuous-on sphere surf
  unfolding homeomorphism-def by auto

 have cont-surfpi: continuous-on (UNIV - {0}) (surf ∘ pi)
  apply (rule continuous-on-compose)
  apply (rule contpi)
  apply (rule continuous-on-subset[of sphere])
  apply (rule surf(6))
  using pi(1)
  apply auto
  done

 {
  fix x
  assume as: x ∈ cball (0::'a) 1
  have norm x *R surf (pi x) ∈ s
  proof (cases x=0 ∨ norm x = 1)
    case False
    then have pi x ∈ sphere norm x < 1
      using pi(1)[of x] as by(auto simp add: dist-norm)
    then show ?thesis
      apply (rule-tac assms(3)[rule-format, THEN DiffD1])
      apply (rule-tac fs[unfolded subset-eq, rule-format])
      unfolding surf(5)[symmetric]
      apply auto
      done
  next
    case True
    then show ?thesis
      apply rule
      defer
      unfolding pi-def
```

```
      apply (rule fs[unfolded subset-eq, rule-format])
      unfolding surf(5)[unfolded sphere-def, symmetric]
      using ‹0∈s›
      apply auto
      done
  qed
 } note hom = this

 {
  fix x
  assume x ∈ s
  then have x ∈ (λx. norm x *R surf (pi x)) ‘ cball 0 1
  proof (cases x = 0)
    case True
    show ?thesis
      unfolding image-iff True
      apply (rule-tac x=0 in bexI)
      apply auto
      done
  next
    let ?a = inverse (norm (surf (pi x)))
    case False
    then have invn: inverse (norm x) ≠ 0 by auto
    from False have pix: pi x∈sphere using pi(1) by auto
    then have pi (surf (pi x)) = pi x
      apply (rule-tac surf(4)[rule-format])
      apply assumption
      done
    then have **: norm x *R (?a *R surf (pi x)) = x
      apply (rule-tac scaleR-left-imp-eq[OF invn])
      unfolding pi-def
      using invn
      apply auto
      done
    then have *: ?a * norm x > 0 and ?a > 0 ?a ≠ 0
      using surf(5) ‹0∉frontier s›
      apply −
      apply (rule mult-pos-pos)
      using False[unfolded zero-less-norm-iff[symmetric]]
      apply auto
      done
    have norm (surf (pi x)) ≠ 0
      using ** False by auto
    then have norm x = norm ((?a * norm x) *R surf (pi x))
      unfolding norm-scaleR abs-mult abs-norm-cancel abs-of-pos[OF ‹?a > 0›]
by auto
    moreover have pi x = pi ((inverse (norm (surf (pi x))) * norm x) *R surf
(pi x))
      unfolding pi(2)[OF *] surf(4)[rule-format, OF pix] ..
```

```
    moreover have surf (pi x) ∈ frontier s
      using surf(5) pix by auto
    then have dist 0 (inverse (norm (surf (pi x))) *R x) ≤ 1
      unfolding dist-norm
      using ** and *
       using front-smul[THEN bspec[where x=surf (pi x)], THEN spec[where
x=norm x * ?a]]
      using False ‹x∈s›
      by (auto simp add: field-simps)
    ultimately show ?thesis
      unfolding image-iff
      apply (rule-tac x=inverse (norm (surf(pi x))) *R x in bexI)
      apply (subst injpi[symmetric])
      unfolding abs-mult abs-norm-cancel abs-of-pos[OF ‹?a > 0›]
      unfolding pi(2)[OF ‹?a > 0›]
      apply auto
      done
  qed
 } note hom2 = this

 show ?thesis
  apply (subst homeomorphic-sym)
  apply (rule homeomorphic-compact[where f=λx. norm x *R surf (pi x)])
  apply (rule compact-cball)
  defer
  apply (rule set-eqI)
  apply rule
  apply (erule imageE)
  apply (drule hom)
  prefer 4
  apply (rule continuous-at-imp-continuous-on)
  apply rule
  apply (rule-tac [3] hom2)
 proof −
  fix x :: 'a
  assume as: x ∈ cball 0 1
  then show continuous (at x) (λx. norm x *R surf (pi x))
  proof (cases x = 0)
    case False
    then show ?thesis
      apply (intro continuous-intros)
      using cont-surfpi
     unfolding continuous-on-eq-continuous-at[OF open-delete[OF open-UNIV]]
o-def
      apply auto
      done
  next
    case True
    obtain B where B: ∀ x∈s. norm x ≤ B
```

```
      using compact-imp-bounded[OF assms(1)] unfolding bounded-iff by auto
    then have B > 0
      using assms(2)
      unfolding subset-eq
      apply (erule-tac x=SOME i. i∈Basis in ballE)
      defer
      apply (erule-tac x=SOME i. i∈Basis in ballE)
      unfolding Ball-def mem-cball dist-norm
      using DIM-positive[where 'a='a]
      apply (auto simp: SOME-Basis)
      done
    show ?thesis
      unfolding True continuous-at Lim-at
      apply(rule,rule)
      apply(rule-tac x=e / B in exI)
      apply rule
      apply (rule divide-pos-pos)
      prefer 3
      apply(rule,rule,erule conjE)
        unfolding norm-zero scaleR-zero-left dist-norm diff-0-right norm-scaleR
abs-norm-cancel
    proof −
      fix e and x :: 'a
      assume as: norm x < e / B 0 < norm x e > 0
      then have surf (pi x) ∈ frontier s
        using pi(1)[of x] unfolding surf(5)[symmetric] by auto
      then have norm (surf (pi x)) ≤ B
        using B fs by auto
      then have norm x * norm (surf (pi x)) ≤ norm x * B
        using as(2) by auto
      also have ... < e / B * B
        apply (rule mult-strict-right-mono)
        using as(1) ‹B>0›
        apply auto
        done
      also have ... = e using ‹B > 0› by auto
      finally show norm x * norm (surf (pi x)) < e .
    qed (insert ‹B>0›, auto)
  qed
 next
  {
    fix x
    assume as: surf (pi x) = 0
    have x = 0
    proof (rule ccontr)
      assume x ≠ 0
      then have pi x ∈ sphere
        using pi(1) by auto
      then have surf (pi x) ∈ frontier s
```

```
        using surf(5) by auto
      then show False
        using ‹0∉frontier s› unfolding as by simp
    qed
  } note surf-0 = this
  show inj-on (λx. norm x *R surf (pi x)) (cball 0 1)
    unfolding inj-on-def
  proof (rule,rule,rule)
    fix x y
    assume as: x ∈ cball 0 1 y ∈ cball 0 1 norm x *R surf (pi x) = norm y *R
surf (pi y)
    then show x = y
    proof (cases x=0 ∨ y=0)
      case True
      then show ?thesis
        using as by (auto elim: surf-0)
    next
      case False
      then have pi (surf (pi x)) = pi (surf (pi y))
        using as(3)
        using pi(2)[of norm x surf (pi x)] pi(2)[of norm y surf (pi y)]
        by auto
      moreover have pi x ∈ sphere pi y ∈ sphere
        using pi(1) False by auto
      ultimately have *: pi x = pi y
        using surf(4)[THEN bspec[where x=pi x]] surf(4)[THEN bspec[where
x=pi y]]
        by auto
      moreover have norm x = norm y
        using as(3)[unfolded *] using False
        by (auto dest:surf-0)
      ultimately show ?thesis
        using injpi by auto
    qed
  qed
 qed auto
qed

lemma homeomorphic-convex-compact-lemma:
 fixes s :: 'a::euclidean-space set
 assumes convex s
   and compact s
   and cball 0 1 ⊆ s
 shows s homeomorphic (cball (0::'a) 1)
proof (rule starlike-compact-projective[OF assms(2−3)], clarify)
 fix x u
 assume x ∈ s and 0 ≤ u and u < (1::real)
 have open (ball (u *R x) (1 − u))
   by (rule open-ball)
```

```
  moreover have u *R x ∈ ball (u *R x) (1 − u)
    unfolding centre-in-ball using ‹u < 1› by simp
  moreover have ball (u *R x) (1 − u) ⊆ s
  proof
    fix y
    assume y ∈ ball (u *R x) (1 − u)
    then have dist (u *R x) y < 1 − u
      unfolding mem-ball .
    with ‹u < 1› have inverse (1 − u) *R (y − u *R x) ∈ cball 0 1
      by (simp add: dist-norm inverse-eq-divide norm-minus-commute)
    with assms(3) have inverse (1 − u) *R (y − u *R x) ∈ s ..
    with assms(1) have (1 − u) *R ((y − u *R x) /R (1 − u)) + u *R x ∈ s
      using ‹x ∈ s› ‹0 ≤ u› ‹u < 1› [THEN less-imp-le] by (rule convexD-alt)
    then show y ∈ s using ‹u < 1›
      by simp
  qed
  ultimately have u *R x ∈ interior s ..
  then show u *R x ∈ s − frontier s
    using frontier-def and interior-subset by auto
qed

lemma homeomorphic-convex-compact-cball:
  fixes e :: real
    and s :: 'a::euclidean-space set
  assumes convex s
    and compact s
    and interior s ≠ {}
    and e > 0
  shows s homeomorphic (cball (b::'a) e)
proof −
  obtain a where a ∈ interior s
    using assms(3) by auto
  then obtain d where d > 0 and d: cball a d ⊆ s
    unfolding mem-interior-cball by auto
  let ?d = inverse d and ?n = 0::'a
  have cball ?n 1 ⊆ (λx. inverse d *R (x − a)) ‘ s
    apply rule
    apply (rule-tac x=d *R x + a in image-eqI)
    defer
    apply (rule d[unfolded subset-eq, rule-format])
    using ‹d > 0›
    unfolding mem-cball dist-norm
    apply (auto simp add: mult-right-le-one-le)
    done
  then have (λx. inverse d *R (x − a)) ‘ s homeomorphic cball ?n 1
    using homeomorphic-convex-compact-lemma[of (λx. ?d *R −a + ?d *R x) ‘ s,
      OF convex-affinity compact-affinity]
    using assms(1,2)
    by (auto simp add: scaleR-right-diff-distrib)
```

then show *?thesis*
apply (*rule-tac homeomorphic-trans*[*OF - homeomorphic-balls*(*2*)[*of 1 - ?n*]])
apply (*rule homeomorphic-trans*[*OF homeomorphic-affinity*[*of ?d s ?d* $*_R$ $-a$]])
using ‹*d*>*0*› ‹*e*>*0*›
apply (*auto simp add*: *scaleR-right-diff-distrib*)
done
qed

lemma *homeomorphic-convex-compact*:
fixes *s* :: *'a*::*euclidean-space set*
and *t* :: *'a set*
assumes *convex s compact s interior s* $\neq$ {}
and *convex t compact t interior t* $\neq$ {}
shows *s homeomorphic t*
using *assms*
by (*meson zero-less-one homeomorphic-trans homeomorphic-convex-compact-cball homeomorphic-sym*)

19.22 Epigraphs of convex functions

definition *epigraph s* (*f* :: - $\Rightarrow$ *real*) = {*xy*. *fst xy* $\in$ *s* $\wedge$ *f* (*fst xy*) $\leq$ *snd xy*}

lemma *mem-epigraph*: (*x*, *y*) $\in$ *epigraph s f* $\longleftrightarrow$ *x* $\in$ *s* $\wedge$ *f x* $\leq$ *y*
unfolding *epigraph-def* **by** *auto*

lemma *convex-epigraph*: *convex* (*epigraph s f*) $\longleftrightarrow$ *convex-on s f* $\wedge$ *convex s*
unfolding *convex-def convex-on-def*
unfolding *Ball-def split-paired-All epigraph-def*
unfolding *mem-Collect-eq fst-conv snd-conv fst-add snd-add fst-scaleR snd-scaleR Ball-def*[*symmetric*]
apply *safe*
defer
apply (*erule-tac x*=*x* **in** *allE*)
apply (*erule-tac x*=*f x* **in** *allE*)
apply *safe*
apply (*erule-tac x*=*xa* **in** *allE*)
apply (*erule-tac x*=*f xa* **in** *allE*)
prefer *3*
apply (*rule-tac y*=*u* $*$ *f a* + *v* $*$ *f aa* **in** *order-trans*)
defer
apply (*auto intro*!:*mult-left-mono add-mono*)
done

lemma *convex-epigraphI*: *convex-on s f* $\Longrightarrow$ *convex s* $\Longrightarrow$ *convex* (*epigraph s f*)
unfolding *convex-epigraph* **by** *auto*

lemma *convex-epigraph-convex*: *convex s* $\Longrightarrow$ *convex-on s f* $\longleftrightarrow$ *convex*(*epigraph s f*)
by (*simp add*: *convex-epigraph*)

19.22.1 Use this to derive general bound property of convex function

lemma *convex-on*:
assumes *convex s*
shows *convex-on s f* $\longleftrightarrow$
$(\forall k\ u\ x.\ (\forall i{\in}\{1..k{::}nat\}.\ 0 \le u\ i \wedge x\ i \in s) \wedge setsum\ u\ \{1..k\} = 1 \longrightarrow$
$f\ (setsum\ (\lambda i.\ u\ i *_R x\ i)\ \{1..k\}\ ) \le setsum\ (\lambda i.\ u\ i * f(x\ i))\ \{1..k\})$
unfolding *convex-epigraph-convex*[*OF assms*] *convex epigraph-def Ball-def mem-Collect-eq*
unfolding *fst-setsum snd-setsum fst-scaleR snd-scaleR*
apply *safe*
apply (*drule-tac x*=*k* **in** *spec*)
apply (*drule-tac x*=*u* **in** *spec*)
apply (*drule-tac x*=$\lambda i.\ (x\ i, f\ (x\ i))$ **in** *spec*)
apply *simp*
using *assms*[*unfolded convex*]
apply *simp*
apply (*rule-tac y*=$\sum i = 1..k.\ u\ i * f\ (fst\ (x\ i))$ **in** *order-trans*)
defer
apply (*rule setsum-mono*)
apply (*erule-tac x*=*i* **in** *allE*)
unfolding *real-scaleR-def*
apply (*rule mult-left-mono*)
using *assms*[*unfolded convex*]
apply *auto*
done

19.23 Convexity of general and special intervals

lemma *is-interval-convex*:
fixes $s :: {'a}{::}euclidean\text{-}space\ set$
assumes *is-interval s*
shows *convex s*
proof (*rule convexI*)
fix *x y* **and** *u v* :: *real*
assume *as*: $x \in s\ y \in s\ 0 \le u\ 0 \le v\ u + v = 1$
then have $*$: $u = 1 - v\ 1 - v \ge 0$ **and** $**$: $v = 1 - u\ 1 - u \ge 0$
by *auto*
{
fix *a b*
assume $\neg\ b \le u * a + v * b$
then have $u * a < (1 - v) * b$
unfolding *not-le* **using** *as*(*4*) **by** (*auto simp add*: *field-simps*)
then have $a < b$
unfolding $*$ **using** *as*(*4*) $*$(*2*)
apply (*rule-tac mult-left-less-imp-less*[*of 1 − v*])
apply (*auto simp add*: *field-simps*)
done
then have $a \le u * a + v * b$
unfolding $*$ **using** *as*(*4*)

```
    by (auto simp add: field-simps intro!:mult-right-mono)
  }
  moreover
  {
   fix a b
   assume ¬ u * a + v * b ≤ a
   then have v * b > (1 - u) * a
    unfolding not-le using as(4) by (auto simp add: field-simps)
   then have a < b
    unfolding * using as(4)
    apply (rule-tac mult-left-less-imp-less)
    apply (auto simp add: field-simps)
    done
   then have u * a + v * b ≤ b
    unfolding **
    using **(2) as(3)
    by (auto simp add: field-simps intro!:mult-right-mono)
  }
  ultimately show u *R x + v *R y ∈ s
   apply -
   apply (rule assms[unfolded is-interval-def, rule-format, OF as(1,2)])
   using as(3-) DIM-positive[where 'a='a]
   apply (auto simp: inner-simps)
   done
qed

lemma is-interval-connected:
  fixes s :: 'a::euclidean-space set
  shows is-interval s ⟹ connected s
  using is-interval-convex convex-connected by auto

lemma convex-box: convex (cbox a b) convex (box a (b::'a::euclidean-space))
  apply (rule-tac[!] is-interval-convex)+
  using is-interval-box is-interval-cbox
  apply auto
  done
```

19.24 On *real*, *is-interval*, *convex* and *connected* are all equivalent.

```
lemma is-interval-1:
  is-interval (s::real set) ⟷ (∀ a∈s. ∀ b∈s. ∀ x. a ≤ x ∧ x ≤ b ⟶ x ∈ s)
  unfolding is-interval-def by auto

lemma is-interval-connected-1:
  fixes s :: real set
  shows is-interval s ⟷ connected s
  apply rule
  apply (rule is-interval-connected, assumption)
```

unfolding *is-interval-1*
apply *rule*
apply *rule*
apply *rule*
apply *rule*
apply (*erule conjE*)
apply (*rule ccontr*)
proof −
fix *a b x*
assume *as*: *connected s a* ∈ *s b* ∈ *s a* ≤ *x x* ≤ *b x* ∉ *s*
then have ∗: *a* < *x x* < *b*
unfolding *not-le* [*symmetric*] **by** *auto*
let *?halfl* = {..<*x*}
let *?halfr* = {*x*<..}
{
fix *y*
assume *y* ∈ *s*
with ‹*x* ∉ *s*› **have** *x* ≠ *y* **by** *auto*
then have *y* ∈ *?halfr* ∪ *?halfl* **by** *auto*
}
moreover have *a* ∈ *?halfl b* ∈ *?halfr* **using** ∗ **by** *auto*
then have *?halfl* ∩ *s* ≠ {} *?halfr* ∩ *s* ≠ {}
using *as*(*2*−*3*) **by** *auto*
ultimately show *False*
apply (*rule-tac notE*[*OF as*(*1*)[*unfolded connected-def*]])
apply (*rule-tac x* = *?halfl* **in** *exI*)
apply (*rule-tac x* = *?halfr* **in** *exI*)
apply *rule*
apply (*rule open-lessThan*)
apply *rule*
apply (*rule open-greaterThan*)
apply *auto*
done
qed

lemma *is-interval-convex-1*:
fixes *s* :: *real set*
shows *is-interval s* ⟷ *convex s*
by (*metis is-interval-convex convex-connected is-interval-connected-1*)

lemma *connected-convex-1*:
fixes *s* :: *real set*
shows *connected s* ⟷ *convex s*
by (*metis is-interval-convex convex-connected is-interval-connected-1*)

lemma *connected-convex-1-gen*:
fixes *s* :: *'a* :: *euclidean-space set*
assumes *DIM*(*'a*) = *1*
shows *connected s* ⟷ *convex s*

proof −
obtain *f*:: *'a* ⇒ *real* **where** *linf*: *linear f* **and** *inj f*
using *subspace-isomorphism* [**where** *'a* = *'a* **and** *'b* = *real*]
by (*metis DIM-real dim-UNIV subspace-UNIV assms*)
then have *f* −*‘* (*f ‘ s*) = *s*
by (*simp add*: *inj-vimage-image-eq*)
then show *?thesis*
by (*metis connected-convex-1 convex-linear-vimage linf convex-connected connected-linear-image*)
qed

19.25 Another intermediate value theorem formulation

lemma *ivt-increasing-component-on-1*:
fixes *f* :: *real* ⇒ *'a*::*euclidean-space*
assumes *a* ≤ *b*
and *continuous-on* {*a*..*b*} *f*
and (*f a*)•*k* ≤ *y y* ≤ (*f b*)•*k*
shows ∃ *x*∈{*a*..*b*}. (*f x*)•*k* = *y*
proof −
have *f a* ∈ *f ‘ cbox a b f b* ∈ *f ‘ cbox a b*
apply (*rule-tac*[!] *imageI*)
using *assms*(*1*)
apply *auto*
done
then show *?thesis*
using *connected-ivt-component*[*of f ‘ cbox a b f a f b k y*]
by (*simp add*: *Topology-Euclidean-Space.connected-continuous-image assms*)
qed

lemma *ivt-increasing-component-1*:
fixes *f* :: *real* ⇒ *'a*::*euclidean-space*
shows *a* ≤ *b* ⟹ ∀ *x*∈{*a*..*b*}. *continuous* (*at x*) *f* ⟹
f a•*k* ≤ *y* ⟹ *y* ≤ *f b*•*k* ⟹ ∃ *x*∈{*a*..*b*}. (*f x*)•*k* = *y*
by (*rule ivt-increasing-component-on-1*) (*auto simp add*: *continuous-at-imp-continuous-on*)

lemma *ivt-decreasing-component-on-1*:
fixes *f* :: *real* ⇒ *'a*::*euclidean-space*
assumes *a* ≤ *b*
and *continuous-on* {*a*..*b*} *f*
and (*f b*)•*k* ≤ *y*
and *y* ≤ (*f a*)•*k*
shows ∃ *x*∈{*a*..*b*}. (*f x*)•*k* = *y*
apply (*subst neg-equal-iff-equal*[*symmetric*])
using *ivt-increasing-component-on-1*[*of a b* λ*x*. − *f x k* − *y*]
using *assms* **using** *continuous-on-minus*
apply *auto*
done

lemma *ivt-decreasing-component-1*:

fixes $f :: real \Rightarrow 'a::euclidean\text{-}space$
shows $a \leq b \Longrightarrow \forall x{\in}\{a..b\}.\ continuous\ (at\ x)\ f \Longrightarrow$
$f\ b{\cdot}k \leq y \Longrightarrow y \leq f\ a{\cdot}k \Longrightarrow \exists x{\in}\{a..b\}.\ (f\ x){\cdot}k = y$
by (*rule ivt-decreasing-component-on-1*) (*auto simp*: *continuous-at-imp-continuous-on*)

19.26 A bound within a convex hull, and so an interval

lemma *convex-on-convex-hull-bound*:
assumes *convex-on* (*convex hull s*) *f*
and $\forall x{\in}s.\ f\ x \leq b$
shows $\forall x{\in}\ convex\ hull\ s.\ f\ x \leq b$
proof
fix *x*
assume $x \in convex\ hull\ s$
then obtain *k u v* **where**
obt: $\forall i{\in}\{1..k{::}nat\}.\ 0 \leq u\ i \wedge v\ i \in s\ setsum\ u\ \{1..k\} = 1\ (\sum i = 1..k.\ u\ i *_R v\ i) = x$
unfolding *convex-hull-indexed mem-Collect-eq* **by** *auto*
have $(\sum i = 1..k.\ u\ i * f\ (v\ i)) \leq b$
using *setsum-mono*[*of* $\{1..k\}$ $\lambda i.\ u\ i * f\ (v\ i)$ $\lambda i.\ u\ i * b$]
unfolding *setsum-left-distrib*[*symmetric*] *obt*(*2*) *mult-1*
apply (*drule-tac meta-mp*)
apply (*rule mult-left-mono*)
using *assms*(*2*) *obt*(*1*)
apply *auto*
done
then show $f\ x \leq b$
using *assms*(*1*)[*unfolded convex-on*[*OF convex-convex-hull*], *rule-format*, *of k u v*]
unfolding *obt*(*2*−*3*)
using *obt*(*1*) **and** *hull-subset*[*unfolded subset-eq*, *rule-format*, *of - s*]
by *auto*
qed

lemma *inner-setsum-Basis*[*simp*]: $i \in Basis \Longrightarrow (\sum Basis) \cdot i = 1$
by (*simp add*: *inner-setsum-left setsum.If-cases inner-Basis*)

lemma *convex-set-plus*:
assumes *convex s* **and** *convex t* **shows** *convex* $(s + t)$
proof −
have *convex* $\{x + y \mid x\ y.\ x \in s \wedge y \in t\}$
using *assms* **by** (*rule convex-sums*)
moreover have $\{x + y \mid x\ y.\ x \in s \wedge y \in t\} = s + t$
unfolding *set-plus-def* **by** *auto*
finally show *convex* $(s + t)$ **.**
qed

lemma *convex-set-setsum*:
assumes $\bigwedge i.\ i \in A \Longrightarrow convex\ (B\ i)$

shows *convex* $(\sum i{\in}A.\ B\ i)$
proof (*cases finite A*)
case *True* **then show** *?thesis* **using** *assms*
by *induct* (*auto simp*: *convex-set-plus*)
qed *auto*

lemma *finite-set-setsum*:
assumes *finite A* **and** $\forall\, i{\in}A.$ *finite* $(B\ i)$ **shows** *finite* $(\sum i{\in}A.\ B\ i)$
using *assms* **by** (*induct set*: *finite*, *simp*, *simp add*: *finite-set-plus*)

lemma *set-setsum-eq*:
finite A $\Longrightarrow$ $(\sum i{\in}A.\ B\ i) = \{\sum i{\in}A.\ f\ i\ |f.\ \forall\, i{\in}A.\ f\ i \in B\ i\}$
apply (*induct set*: *finite*)
apply *simp*
apply *simp*
apply (*safe elim*!: *set-plus-elim*)
apply (*rule-tac x*=*fun-upd f x a* **in** *exI*)
apply *simp*
apply (*rule-tac f*=$\lambda x.\ a + x$ **in** *arg-cong*)
apply (*rule setsum.cong* [*OF refl*])
apply *clarsimp*
apply *fast*
done

lemma *box-eq-set-setsum-Basis*:
shows $\{x.\ \forall\, i{\in}Basis.\ x{\cdot}i \in B\ i\} = (\sum i{\in}Basis.\ image\ (\lambda x.\ x *_R i)\ (B\ i))$
apply (*subst set-setsum-eq* [*OF finite-Basis*])
apply *safe*
apply (*fast intro*: *euclidean-representation* [*symmetric*])
apply (*subst inner-setsum-left*)
apply (*subgoal-tac* $(\sum x{\in}Basis.\ f\ x \cdot i) = f\ i \cdot i$)
apply (*drule* (*1*) *bspec*)
apply *clarsimp*
apply (*frule setsum.remove* [*OF finite-Basis*])
apply (*erule trans*)
apply *simp*
apply (*rule setsum.neutral*)
apply *clarsimp*
apply (*frule-tac x*=*i* **in** *bspec*, *assumption*)
apply (*drule-tac x*=*x* **in** *bspec*, *assumption*)
apply *clarsimp*
apply (*cut-tac u*=*x* **and** *v*=*i* **in** *inner-Basis*, *assumption*+)
apply (*rule ccontr*)
apply *simp*
done

lemma *convex-hull-set-setsum*:
convex hull $(\sum i{\in}A.\ B\ i) = (\sum i{\in}A.$ *convex hull* $(B\ i))$
proof (*cases finite A*)

assume *finite A* **then show** *?thesis*
by (*induct set*: *finite*, *simp*, *simp add*: *convex-hull-set-plus*)
qed *simp*

lemma *convex-hull-eq-real-cbox*:
fixes *x y* :: *real* **assumes** $x \leq y$
shows *convex hull* $\{x, y\} = cbox\ x\ y$
proof (*rule hull-unique*)
show $\{x, y\} \subseteq cbox\ x\ y$ **using** ‹$x \leq y$› **by** *auto*
show *convex* (*cbox x y*)
by (*rule convex-box*)
next
fix *s* **assume** $\{x, y\} \subseteq s$ **and** *convex s*
then show *cbox x y* $\subseteq s$
unfolding *is-interval-convex-1* [*symmetric*] *is-interval-def Basis-real-def*
by − (*clarify*, *simp* (*no-asm-use*), *fast*)
qed

lemma *unit-interval-convex-hull*:
cbox (*0*::*'a*::*euclidean-space*) *One* = *convex hull* $\{x. \forall i{\in}Basis.\ (x{\cdot}i = 0) \vee (x{\cdot}i = 1)\}$
(**is** *?int* = *convex hull ?points*)
proof −
have *One*[*simp*]: $\bigwedge i.\ i \in Basis \Longrightarrow One \cdot i = 1$
by (*simp add*: *inner-setsum-left setsum.If-cases inner-Basis*)
have *?int* = $\{x. \forall i{\in}Basis.\ x \cdot i \in cbox\ 0\ 1\}$
by (*auto simp*: *cbox-def*)
also have ... = $(\sum i{\in}Basis.\ (\lambda x.\ x *_R i)\ `\ cbox\ 0\ 1)$
by (*simp only*: *box-eq-set-setsum-Basis*)
also have ... = $(\sum i{\in}Basis.\ (\lambda x.\ x *_R i)\ `\ (convex\ hull\ \{0, 1\}))$
by (*simp only*: *convex-hull-eq-real-cbox zero-le-one*)
also have ... = $(\sum i{\in}Basis.\ convex\ hull\ ((\lambda x.\ x *_R i)\ `\ \{0, 1\}))$
by (*simp only*: *convex-hull-linear-image linear-scaleR-left*)
also have ... = *convex hull* $(\sum i{\in}Basis.\ (\lambda x.\ x *_R i)\ `\ \{0, 1\})$
by (*simp only*: *convex-hull-set-setsum*)
also have ... = *convex hull* $\{x. \forall i{\in}Basis.\ x{\cdot}i \in \{0, 1\}\}$
by (*simp only*: *box-eq-set-setsum-Basis*)
also have *convex hull* $\{x. \forall i{\in}Basis.\ x{\cdot}i \in \{0, 1\}\}$ = *convex hull ?points*
by *simp*
finally show *?thesis* **.**
qed

And this is a finite set of vertices.

lemma *unit-cube-convex-hull*:
obtains *s* :: *'a*::*euclidean-space set*
where *finite s* **and** *cbox 0* $(\sum Basis)$ = *convex hull s*
apply (*rule that*[*of* $\{x::'a.\ \forall i{\in}Basis.\ x{\cdot}i{=}0 \vee x{\cdot}i{=}1\}$])
apply (*rule finite-subset*[*of* - $(\lambda s.\ (\sum i{\in}Basis.\ (if\ i{\in}s\ then\ 1\ else\ 0) *_R i){::}'a)\ `$ *Pow Basis*])

```
  prefer 3
  apply (rule unit-interval-convex-hull)
  apply rule
  unfolding mem-Collect-eq
proof −
  fix x :: 'a
  assume as: ∀ i∈Basis. x · i = 0 ∨ x · i = 1
  show x ∈ (λs. ∑ i∈Basis. (if i∈s then 1 else 0) *R i) ' Pow Basis
    apply (rule image-eqI[where x={i. i∈Basis ∧ x·i = 1}])
    using as
    apply (subst euclidean-eq-iff)
    apply auto
    done
qed auto
```

Hence any cube (could do any nonempty interval).

```
lemma cube-convex-hull:
  assumes d > 0
  obtains s :: 'a::euclidean-space set where
    finite s and cbox (x − (∑ i∈Basis. d*Ri)) (x + (∑ i∈Basis. d*Ri)) = convex
hull s
proof −
  let ?d = (∑ i∈Basis. d*Ri)::'a
  have *: cbox (x − ?d) (x + ?d) = (λy. x − ?d + (2 * d) *R y) ' cbox 0
(∑ Basis)
    apply (rule set-eqI, rule)
    unfolding image-iff
    defer
    apply (erule bexE)
  proof −
    fix y
    assume as: y∈cbox (x − ?d) (x + ?d)
    then have inverse (2 * d) *R (y − (x − ?d)) ∈ cbox 0 (∑ Basis)
      using assms by (simp add: mem-box field-simps inner-simps)
    with ‹0 < d› show ∃ z∈cbox 0 (∑ Basis). y = x − ?d + (2 * d) *R z
      by (intro bexI[of - inverse (2 * d) *R (y − (x − ?d))]) auto
  next
    fix y z
    assume as: z∈cbox 0 (∑ Basis) y = x − ?d + (2*d) *R z
    have ⋀i. i∈Basis ⟹ 0 ≤ d * (z · i) ∧ d * (z · i) ≤ d
      using assms as(1)[unfolded mem-box]
      apply (erule-tac x=i in ballE)
      apply rule
      prefer 2
      apply (rule mult-right-le-one-le)
      using assms
      apply auto
      done
    then show y ∈ cbox (x − ?d) (x + ?d)
```

```
    unfolding as(2) mem-box
    apply −
    apply rule
    using as(1)[unfolded mem-box]
    apply (erule-tac x=i in ballE)
    using assms
    apply (auto simp: inner-simps)
    done
  qed
  obtain s where finite s cbox 0 (∑ Basis::'a) = convex hull s
    using unit-cube-convex-hull by auto
  then show ?thesis
    apply (rule-tac that[of (λy. x − ?d + (2 ∗ d) ∗R y)‘ s])
    unfolding ∗ and convex-hull-affinity
    apply auto
    done
qed
```

19.27 Bounded convex function on open set is continuous

```
lemma convex-on-bounded-continuous:
  fixes s :: ('a::real-normed-vector) set
  assumes open s
    and convex-on s f
    and ∀ x∈s. |f x| ≤ b
  shows continuous-on s f
  apply (rule continuous-at-imp-continuous-on)
  unfolding continuous-at-real-range
proof (rule,rule,rule)
  fix x and e :: real
  assume x ∈ s e > 0
  def B ≡ |b| + 1
  have B: 0 < B ⋀x. x∈s ⟹ |f x| ≤ B
    unfolding B-def
    defer
    apply (drule assms(3)[rule-format])
    apply auto
    done
  obtain k where k > 0 and k: cball x k ⊆ s
    using assms(1)[unfolded open-contains-cball, THEN bspec[where x=x]]
    using ‹x∈s› by auto
  show ∃ d>0. ∀ x'. norm (x' − x) < d ⟶ |f x' − f x| < e
    apply (rule-tac x=min (k / 2) (e / (2 ∗ B) ∗ k) in exI)
    apply rule
    defer
  proof (rule, rule)
    fix y
    assume as: norm (y − x) < min (k / 2) (e / (2 ∗ B) ∗ k)
    show |f y − f x| < e
```

```
  proof (cases y = x)
    case False
    def t ≡ k / norm (y - x)
    have 2 < t 0<t
      unfolding t-def using as False and ‹k>0›
      by (auto simp add:field-simps)
    have y ∈ s
      apply (rule k[unfolded subset-eq,rule-format])
      unfolding mem-cball dist-norm
      apply (rule order-trans[of - 2 * norm (x - y)])
      using as
      by (auto simp add: field-simps norm-minus-commute)
    {
      def w ≡ x + t *R (y - x)
      have w ∈ s
        unfolding w-def
        apply (rule k[unfolded subset-eq,rule-format])
        unfolding mem-cball dist-norm
        unfolding t-def
        using ‹k>0›
        apply auto
        done
      have (1 / t) *R x + - x + ((t - 1) / t) *R x = (1 / t - 1 + (t - 1) /
t) *R x
        by (auto simp add: algebra-simps)
      also have ... = 0
        using ‹t > 0› by (auto simp add:field-simps)
      finally have w: (1 / t) *R w + ((t - 1) / t) *R x = y
        unfolding w-def using False and ‹t > 0›
        by (auto simp add: algebra-simps)
      have  2 * B < e * t
        unfolding t-def using ‹0 < e› ‹0 < k› ‹B > 0› and as and False
        by (auto simp add:field-simps)
      then have (f w - f x) / t < e
        using B(2)[OF ‹w∈s›] and B(2)[OF ‹x∈s›]
        using ‹t > 0› by (auto simp add:field-simps)
      then have th1: f y - f x < e
        apply -
        apply (rule le-less-trans)
        defer
        apply assumption
        using assms(2)[unfolded convex-on-def,rule-format,of w x 1/t (t - 1)/t,
unfolded w]
        using ‹0 < t› ‹2 < t› and ‹x ∈ s› ‹w ∈ s›
        by (auto simp add:field-simps)
    }
    moreover
    {
      def w ≡ x - t *R (y - x)
```

```
      have w ∈ s
        unfolding w-def
        apply (rule k[unfolded subset-eq,rule-format])
        unfolding mem-cball dist-norm
        unfolding t-def
        using ‹k > 0›
        apply auto
        done
      have (1 / (1 + t)) *R x + (t / (1 + t)) *R x = (1 / (1 + t) + t / (1 +
t)) *R x
        by (auto simp add: algebra-simps)
      also have ... = x
        using ‹t > 0› by (auto simp add:field-simps)
      finally have w: (1 / (1+t)) *R w + (t / (1 + t)) *R y = x
        unfolding w-def using False and ‹t > 0›
        by (auto simp add: algebra-simps)
      have 2 * B < e * t
        unfolding t-def
        using ‹0 < e› ‹0 < k› ‹B > 0› and as and False
        by (auto simp add:field-simps)
      then have *: (f w − f y) / t < e
        using B(2)[OF ‹w∈s›] and B(2)[OF ‹y∈s›]
        using ‹t > 0›
        by (auto simp add:field-simps)
      have f x ≤ 1 / (1 + t) * f w + (t / (1 + t)) * f y
        using assms(2)[unfolded convex-on-def,rule-format,of w y 1/(1+t) t /
(1+t),unfolded w]
        using ‹0 < t› ‹2 < t› and ‹y ∈ s› ‹w ∈ s›
        by (auto simp add:field-simps)
      also have ... = (f w + t * f y) / (1 + t)
        using ‹t > 0› by (auto simp add: divide-simps)
      also have ... < e + f y
        using ‹t > 0› * ‹e > 0› by (auto simp add: field-simps)
      finally have f x − f y < e by auto
    }
    ultimately show ?thesis by auto
  qed (insert ‹0<e›, auto)
 qed (insert ‹0<e› ‹0<k› ‹0<B›, auto simp: field-simps)
qed
```

19.28 Upper bound on a ball implies upper and lower bounds

```
lemma convex-bounds-lemma:
 fixes x :: 'a::real-normed-vector
 assumes convex-on (cball x e) f
   and ∀ y ∈ cball x e. f y ≤ b
 shows ∀ y ∈ cball x e. |f y| ≤ b + 2 * |f x|
 apply rule
proof (cases 0 ≤ e)
```

case *True*
fix y
assume y: $y \in cball\ x\ e$
def $z \equiv 2 *_R x - y$
have $*$: $x - (2 *_R x - y) = y - x$
by (*simp add*: *scaleR-2*)
have z: $z \in cball\ x\ e$
using y **unfolding** *z-def mem-cball dist-norm* $*$ **by** (*auto simp add*: *norm-minus-commute*)
have $(1 / 2) *_R y + (1 / 2) *_R z = x$
unfolding *z-def* **by** (*auto simp add*: *algebra-simps*)
then show $|f\ y| \leq b + 2 * |f\ x|$
using *assms*(*1*)[*unfolded convex-on-def*,*rule-format*, *OF y z*, *of 1/2 1/2*]
using *assms*(*2*)[*rule-format*,*OF y*] *assms*(*2*)[*rule-format*,*OF z*]
by (*auto simp add*:*field-simps*)
next
case *False*
fix y
assume $y \in cball\ x\ e$
then have *dist x y* < 0
using *False* **unfolding** *mem-cball not-le* **by** (*auto simp del*: *dist-not-less-zero*)
then show $|f\ y| \leq b + 2 * |f\ x|$
using *zero-le-dist*[*of x y*] **by** *auto*
qed

19.28.1 Hence a convex function on an open set is continuous

lemma *real-of-nat-ge-one-iff*: $1 \leq real\ (n::nat) \longleftrightarrow 1 \leq n$
by *auto*

lemma *convex-on-continuous*:
assumes *open* $(s::('a::euclidean\text{-}space)\ set)$ *convex-on s f*
shows *continuous-on s f*
unfolding *continuous-on-eq-continuous-at*[*OF assms*(*1*)]
proof
note *dimge1* = *DIM-positive*[**where** $'a='a$]
fix x
assume $x \in s$
then obtain e **where** e: *cball x e* $\subseteq s$ $e > 0$
using *assms*(*1*) **unfolding** *open-contains-cball* **by** *auto*
def $d \equiv e$ / *real DIM*($'a$)
have $0 < d$
unfolding *d-def* **using** ‹$e > 0$› *dimge1* **by** *auto*
let *?d* = $(\sum i{\in}Basis.\ d *_R i)::'a$
obtain c
where c: *finite c*
and *c1*: *convex hull c* $\subseteq$ *cball x e*
and *c2*: *cball x d* $\subseteq$ *convex hull c*
proof
def $c \equiv \sum i{\in}Basis.\ (\lambda a.\ a *_R i)$ ` $\{x{\cdot}i - d, x{\cdot}i + d\}$

```
  show finite c
    unfolding c-def by (simp add: finite-set-setsum)
  have 1: convex hull c = {a. ∀ i∈Basis. a • i ∈ cbox (x • i - d) (x • i + d)}
    unfolding box-eq-set-setsum-Basis
    unfolding c-def convex-hull-set-setsum
    apply (subst convex-hull-linear-image [symmetric])
    apply (simp add: linear-iff scaleR-add-left)
    apply (rule setsum.cong [OF refl])
    apply (rule image-cong [OF - refl])
    apply (rule convex-hull-eq-real-cbox)
    apply (cut-tac ‹0 < d›, simp)
    done
  then have 2: convex hull c = {a. ∀ i∈Basis. a • i ∈ cball (x • i) d}
    by (simp add: dist-norm abs-le-iff algebra-simps)
  show cball x d ⊆ convex hull c
    unfolding 2
    apply clarsimp
    apply (simp only: dist-norm)
    apply (subst inner-diff-left [symmetric])
    apply simp
    apply (erule (1) order-trans [OF Basis-le-norm])
    done
  have e': e = (∑(i::'a)∈Basis. d)
    by (simp add: d-def DIM-positive)
  show convex hull c ⊆ cball x e
    unfolding 2
    apply clarsimp
    apply (subst euclidean-dist-l2)
    apply (rule order-trans [OF setL2-le-setsum])
    apply (rule zero-le-dist)
    unfolding e'
    apply (rule setsum-mono)
    apply simp
    done
qed
def k ≡ Max (f ‘ c)
have convex-on (convex hull c) f
  apply(rule convex-on-subset[OF assms(2)])
  apply(rule subset-trans[OF - e(1)])
  apply(rule c1)
  done
then have k: ∀ y∈convex hull c. f y ≤ k
  apply (rule-tac convex-on-convex-hull-bound)
  apply assumption
  unfolding k-def
  apply (rule, rule Max-ge)
  using c(1)
  apply auto
  done
```

have $d \leq e$
unfolding *d-def*
apply (*rule mult-imp-div-pos-le*)
using ‹$e > 0$›
unfolding *mult-le-cancel-left1*
apply (*auto simp*: *real-of-nat-ge-one-iff Suc-le-eq DIM-positive*)
done
then have *dsube*: *cball x d* $\subseteq$ *cball x e*
by (*rule subset-cball*)
have *conv*: *convex-on* (*cball x d*) *f*
apply (*rule convex-on-subset*)
apply (*rule convex-on-subset*[*OF assms*(*2*)])
apply (*rule e*(*1*))
apply (*rule dsube*)
done
then have $\forall\, y{\in}$*cball x d*. $|f\ y| \leq k + 2 * |f\ x|$
apply (*rule convex-bounds-lemma*)
apply (*rule ballI*)
apply (*rule k* [*rule-format*])
apply (*erule rev-subsetD*)
apply (*rule c2*)
done
then have *continuous-on* (*ball x d*) *f*
apply (*rule-tac convex-on-bounded-continuous*)
apply (*rule open-ball*, *rule convex-on-subset*[*OF conv*])
apply (*rule ball-subset-cball*)
apply *force*
done
then show *continuous* (*at x*) *f*
unfolding *continuous-on-eq-continuous-at*[*OF open-ball*]
using ‹$d > 0$› **by** *auto*
qed

19.29 Line segments, Starlike Sets, etc.

definition *midpoint* :: *'a*::*real-vector* $\Rightarrow$ *'a* $\Rightarrow$ *'a*
where *midpoint a b* = (*inverse* (*2*::*real*)) $*_R$ (*a* + *b*)

definition *closed-segment* :: *'a*::*real-vector* $\Rightarrow$ *'a* $\Rightarrow$ *'a set*
where *closed-segment a b* = $\{(1 - u) *_R a + u *_R b \mid u$::*real*. $0 \leq u \wedge u \leq 1\}$

definition *open-segment* :: *'a*::*real-vector* $\Rightarrow$ *'a* $\Rightarrow$ *'a set* **where**
open-segment a b $\equiv$ *closed-segment a b* $-$ $\{a,b\}$

definition *between* = ($\lambda(a,b)$ *x*. *x* $\in$ *closed-segment a b*)

definition *starlike s* $\longleftrightarrow$ ($\exists\, a{\in}s.\ \forall\, x{\in}s.$ *closed-segment a x* $\subseteq$ *s*)

lemmas *segment* = *open-segment-def closed-segment-def*

lemma *midpoint-refl*: *midpoint x x = x*
unfolding *midpoint-def*
unfolding *scaleR-right-distrib*
unfolding *scaleR-left-distrib*[*symmetric*]
by *auto*

lemma *midpoint-sym*: *midpoint a b = midpoint b a*
unfolding *midpoint-def* **by** (*auto simp add*: *scaleR-right-distrib*)

lemma *midpoint-eq-iff*: *midpoint a b = c* ⟷ *a + b = c + c*
proof −
have *midpoint a b = c* ⟷ *scaleR 2* (*midpoint a b*) = *scaleR 2 c*
by *simp*
then show *?thesis*
unfolding *midpoint-def scaleR-2* [*symmetric*] **by** *simp*
qed

lemma *dist-midpoint*:
fixes *a b* :: *'a*::*real-normed-vector* **shows**
dist a (*midpoint a b*) = (*dist a b*) / *2* (**is** *?t1*)
dist b (*midpoint a b*) = (*dist a b*) / *2* (**is** *?t2*)
dist (*midpoint a b*) *a* = (*dist a b*) / *2* (**is** *?t3*)
dist (*midpoint a b*) *b* = (*dist a b*) / *2* (**is** *?t4*)
proof −
have ∗: ⋀*x y*::*'a*. *2* ∗*R x = − y* ⟹ *norm x* = (*norm y*) / *2*
unfolding *equation-minus-iff* **by** *auto*
have ∗∗: ⋀*x y*::*'a*. *2* ∗*R x = y* ⟹ *norm x* = (*norm y*) / *2*
by *auto*
note *scaleR-right-distrib* [*simp*]
show *?t1*
unfolding *midpoint-def dist-norm*
apply (*rule* ∗∗)
apply (*simp add*: *scaleR-right-diff-distrib*)
apply (*simp add*: *scaleR-2*)
done
show *?t2*
unfolding *midpoint-def dist-norm*
apply (*rule* ∗)
apply (*simp add*: *scaleR-right-diff-distrib*)
apply (*simp add*: *scaleR-2*)
done
show *?t3*
unfolding *midpoint-def dist-norm*
apply (*rule* ∗)
apply (*simp add*: *scaleR-right-diff-distrib*)
apply (*simp add*: *scaleR-2*)
done
show *?t4*

unfolding *midpoint-def dist-norm*
apply (*rule* ∗∗)
apply (*simp add*: *scaleR-right-diff-distrib*)
apply (*simp add*: *scaleR-2*)
done
qed

lemma *midpoint-eq-endpoint*:
midpoint a b = *a* ⟷ *a* = *b*
midpoint a b = *b* ⟷ *a* = *b*
unfolding *midpoint-eq-iff* **by** *auto*

lemma *convex-contains-segment*:
convex s ⟷ (∀ *a*∈*s*. ∀ *b*∈*s*. *closed-segment a b* ⊆ *s*)
unfolding *convex-alt closed-segment-def* **by** *auto*

lemma *closed-segment-subset*: ⟦*x* ∈ *s*; *y* ∈ *s*; *convex s*⟧ ⟹ *closed-segment x y* ⊆ *s*
by (*simp add*: *convex-contains-segment*)

lemma *closed-segment-subset-convex-hull*:
⟦*x* ∈ *convex hull s*; *y* ∈ *convex hull s*⟧ ⟹ *closed-segment x y* ⊆ *convex hull s*
using *convex-contains-segment* **by** *blast*

lemma *convex-imp-starlike*:
convex s ⟹ *s* ≠ {} ⟹ *starlike s*
unfolding *convex-contains-segment starlike-def* **by** *auto*

lemma *segment-convex-hull*:
closed-segment a b = *convex hull* {*a*,*b*}
proof −
have ∗: ⋀*x*. {*x*} ≠ {} **by** *auto*
show *?thesis*
unfolding *segment convex-hull-insert*[*OF* ∗] *convex-hull-singleton*
by (*safe*; *rule-tac x*=*1* − *u* **in** *exI*; *force*)
qed

lemma *open-closed-segment*: *u* ∈ *open-segment w z* ⟹ *u* ∈ *closed-segment w z*
by (*auto simp add*: *closed-segment-def open-segment-def*)

lemma *segment-open-subset-closed*:
open-segment a b ⊆ *closed-segment a b*
by (*auto simp*: *closed-segment-def open-segment-def*)

lemma *bounded-closed-segment*:
fixes *a* :: ′*a*::*euclidean-space* **shows** *bounded* (*closed-segment a b*)
by (*simp add*: *segment-convex-hull compact-convex-hull compact-imp-bounded*)

lemma *bounded-open-segment*:

fixes $a :: 'a$::*euclidean-space* **shows** *bounded* (*open-segment a b*)
by (*rule bounded-subset* [*OF bounded-closed-segment segment-open-subset-closed*])

lemmas *bounded-segment* = *bounded-closed-segment open-closed-segment*

lemma *ends-in-segment* [*iff*]: $a \in$ *closed-segment a b* $b \in$ *closed-segment a b*
unfolding *segment-convex-hull*
by (*auto intro*!: *hull-subset*[*unfolded subset-eq*, *rule-format*])

lemma *segment-furthest-le*:
fixes $a\ b\ x\ y :: 'a$::*euclidean-space*
assumes $x \in$ *closed-segment a b*
shows *norm* $(y - x) \leq$ *norm* $(y - a) \vee$ *norm* $(y - x) \leq$ *norm* $(y - b)$
proof −
obtain z **where** $z \in \{a, b\}$ *norm* $(x - y) \leq$ *norm* $(z - y)$
using *simplex-furthest-le*[*of* $\{a, b\}$ y]
using *assms*[*unfolded segment-convex-hull*]
by *auto*
then show *?thesis*
by (*auto simp add*:*norm-minus-commute*)
qed

lemma *closed-segment-commute*: *closed-segment a b* = *closed-segment b a*
proof −
have $\{a, b\} = \{b, a\}$ **by** *auto*
thus *?thesis*
by (*simp add*: *segment-convex-hull*)
qed

lemma *segment-bound1*:
assumes $x \in$ *closed-segment a b*
shows *norm* $(x - a) \leq$ *norm* $(b - a)$
proof −
obtain u **where** $x = (1 - u) *_R a + u *_R b\ 0 \leq u\ u \leq 1$
using *assms* **by** (*auto simp add*: *closed-segment-def*)
then show *norm* $(x - a) \leq$ *norm* $(b - a)$
apply *clarify*
apply (*auto simp*: *algebra-simps*)
apply (*simp add*: *scaleR-diff-right* [*symmetric*] *mult-left-le-one-le*)
done
qed

lemma *segment-bound*:
assumes $x \in$ *closed-segment a b*
shows *norm* $(x - a) \leq$ *norm* $(b - a)$ *norm* $(x - b) \leq$ *norm* $(b - a)$
apply (*simp add*: *assms segment-bound1*)
by (*metis assms closed-segment-commute dist-commute dist-norm segment-bound1*)

lemma *open-segment-commute*: *open-segment a b* = *open-segment b a*

proof −
have $\{a, b\} = \{b, a\}$ **by** *auto*
thus *?thesis*
by (*simp add*: *closed-segment-commute open-segment-def*)
qed

lemma *closed-segment-idem* [*simp*]: *closed-segment a a* = $\{a\}$
unfolding *segment* **by** (*auto simp add*: *algebra-simps*)

lemma *open-segment-idem* [*simp*]: *open-segment a a* = $\{\}$
by (*simp add*: *open-segment-def*)

lemma *closed-segment-eq-real-ivl*:
fixes *a b*::*real*
shows *closed-segment a b* = (*if a* ≤ *b then* $\{a$ .. $b\}$ *else* $\{b$ .. $a\}$)
proof −
have *b* ≤ *a* ⟹ *closed-segment b a* = $\{b$ .. $a\}$
and *a* ≤ *b* ⟹ *closed-segment a b* = $\{a$ .. $b\}$
by (*auto simp*: *convex-hull-eq-real-cbox segment-convex-hull*)
thus *?thesis*
by (*auto simp*: *closed-segment-commute*)
qed

lemma *closed-segment-real-eq*:
fixes *u*::*real* **shows** *closed-segment u v* = (λ*x*. (*v* − *u*) ∗ *x* + *u*) ' $\{0..1\}$
by (*simp add*: *add.commute* [*of u*] *image-affinity-atLeastAtMost* [**where** *c*=*u*] *closed-segment-eq-real-ivl*)

19.29.1 More lemmas, especially for working with the underlying formula

lemma *segment-eq-compose*:
fixes *a* :: *'a* :: *real-vector*
shows (λ*u*. (*1* − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b*) = (λ*x*. *a* + *x*) *o* (λ*u*. *u* ∗$_R$ (*b* − *a*))
by (*simp add*: *o-def algebra-simps*)

lemma *segment-degen-1*:
fixes *a* :: *'a* :: *real-vector*
shows (*1* − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b* = *b* ⟷ *a*=*b* ∨ *u*=*1*
proof −
{ assume (*1* − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b* = *b*
then have (*1* − *u*) ∗$_R$ *a* = (*1* − *u*) ∗$_R$ *b*
by (*simp add*: *algebra-simps*)
then have *a*=*b* ∨ *u*=*1*
by *simp*
} then show *?thesis*
by (*auto simp*: *algebra-simps*)
qed

lemma *segment-degen-0*:
  fixes *a* :: *'a* :: *real-vector*
  shows $(1 - u) *_R a + u *_R b = a \longleftrightarrow a{=}b \vee u{=}0$
 using *segment-degen-1* [*of 1−u b a*]
 by (*auto simp*: *algebra-simps*)

lemma *closed-segment-image-interval*:
  closed-segment a b = $(\lambda u.\ (1 - u) *_R a + u *_R b)$ ‘ $\{0..1\}$
 by (*auto simp*: *set-eq-iff image-iff closed-segment-def*)

lemma *open-segment-image-interval*:
  open-segment a b = (*if a=b then* {} *else* $(\lambda u.\ (1 - u) *_R a + u *_R b)$ ‘ $\{0<..<1\}$)
 by (*auto simp*: *open-segment-def closed-segment-def segment-degen-0 segment-degen-1*)

lemmas *segment-image-interval* = *closed-segment-image-interval open-segment-image-interval*

lemma *open-segment-eq*: *open-segment a b* = (*if a=b then* {} *else* $\{(1 - u) *_R a + u *_R b \mid u{::}real.\ 0 < u \wedge u < 1\}$)
 by (*auto simp*: *open-segment-def closed-segment-def segment-degen-0 segment-degen-1*)

lemma *open-segment-bound1*:
 assumes $x \in$ *open-segment a b*
 shows *norm* $(x - a) <$ *norm* $(b - a)$
proof −
 obtain *u* **where** $x = (1 - u) *_R a + u *_R b\ 0 < u\ u < 1\ a \neq b$
  using *assms* **by** (*auto simp add*: *open-segment-eq split*: *split-if-asm*)
 then show *norm* $(x - a) <$ *norm* $(b - a)$
  apply *clarify*
  apply (*auto simp*: *algebra-simps*)
  apply (*simp add*: *scaleR-diff-right* [*symmetric*])
  done
qed

lemma *open-segment-bound*:
 assumes $x \in$ *open-segment a b*
 shows *norm* $(x - a) <$ *norm* $(b - a)$ *norm* $(x - b) <$ *norm* $(b - a)$
apply (*simp add*: *assms open-segment-bound1*)
by (*metis assms norm-minus-commute open-segment-bound1 open-segment-commute*)

lemma *closure-closed-segment* [*simp*]:
  fixes *a* :: *'a*::*euclidean-space*
  shows *closure*(*closed-segment a b*) = *closed-segment a b*
 by (*simp add*: *closure-eq compact-imp-closed segment-convex-hull compact-convex-hull*)

lemma *closure-open-segment* [*simp*]:
  fixes *a* :: *'a*::*euclidean-space*
  shows *closure*(*open-segment a b*) = (*if a = b then* {} *else closed-segment a b*)
proof −

have *closure* $((\lambda u.\ u *_R (b - a))$ ‘ $\{0<..<1\}) = (\lambda u.\ u *_R (b - a))$ ‘ *closure* $\{0<..<1\}$ **if** $a \neq b$
apply (*rule closure-injective-linear-image* [*symmetric*])
apply (*simp add*:)
using *that* **by** (*simp add*: *inj-on-def*)
then show *?thesis*
by (*simp add*: *segment-image-interval segment-eq-compose closure-greaterThanLessThan* [*symmetric*]
closure-translation image-comp [*symmetric*] *del*: *closure-greaterThanLessThan*)
qed

lemma *closed-segment* [*simp*]:
fixes *a* :: *'a*::*euclidean-space* **shows** *closed* (*closed-segment a b*)
using *closure-subset-eq* **by** *fastforce*

lemma *closed-open-segment-iff* [*simp*]:
fixes *a* :: *'a*::*euclidean-space* **shows** *closed*(*open-segment a b*) $\longleftrightarrow$ $a = b$
by (*metis open-segment-def DiffE closure-eq closure-open-segment ends-in-segment*(*1*) *insert-iff segment-image-interval*(*2*))

lemma *compact-segment* [*simp*]:
fixes *a* :: *'a*::*euclidean-space* **shows** *compact* (*closed-segment a b*)
by (*simp add*: *compact-convex-hull segment-convex-hull*)

lemma *compact-open-segment-iff* [*simp*]:
fixes *a* :: *'a*::*euclidean-space* **shows** *compact*(*open-segment a b*) $\longleftrightarrow$ $a = b$
by (*simp add*: *bounded-open-segment compact-eq-bounded-closed*)

lemma *convex-closed-segment* [*iff*]: *convex* (*closed-segment a b*)
unfolding *segment-convex-hull* **by**(*rule convex-convex-hull*)

lemma *convex-open-segment* [*iff*]: *convex*(*open-segment a b*)
proof −
have *convex* $((\lambda u.\ u *_R (b-a))$ ‘ $\{0<..<1\})$
by (*rule convex-linear-image*) *auto*
then show *?thesis*
apply (*simp add*: *open-segment-image-interval segment-eq-compose*)
by (*metis image-comp convex-translation*)
qed

lemmas *convex-segment* = *convex-closed-segment convex-open-segment*

lemma *connected-segment* [*iff*]:
fixes *x* :: *'a* :: *real-normed-vector*
shows *connected* (*closed-segment x y*)
by (*simp add*: *convex-connected*)

lemma *affine-hull-closed-segment* [*simp*]:
affine hull (*closed-segment a b*) = *affine hull* {*a*,*b*}

by (*simp add*: *segment-convex-hull*)

lemma *affine-hull-open-segment* [*simp*]:
fixes *a* :: *'a*::*euclidean-space*
shows *affine hull* (*open-segment a b*) = (*if a* = *b then* {} *else affine hull* {*a*,*b*})
by (*metis affine-hull-convex-hull affine-hull-empty closure-open-segment closure-same-affine-hull segment-convex-hull*)

19.29.2 Betweenness

lemma *between-mem-segment*: *between* (*a*,*b*) *x* ⟷ *x* ∈ *closed-segment a b*
unfolding *between-def* **by** *auto*

lemma *between*: *between* (*a*, *b*) (*x*::*'a*::*euclidean-space*) ⟷ *dist a b* = (*dist a x*) + (*dist x b*)
proof (*cases a* = *b*)
case *True*
then show *?thesis*
unfolding *between-def split-conv*
by (*auto simp add*: *dist-commute*)
next
case *False*
then have *Fal*: *norm* (*a* − *b*) ≠ *0* **and** *Fal2*: *norm* (*a* − *b*) > *0*
by *auto*
have ∗: ⋀*u*. *a* − ((*1* − *u*) ∗*R a* + *u* ∗*R b*) = *u* ∗*R* (*a* − *b*)
by (*auto simp add*: *algebra-simps*)
show *?thesis*
unfolding *between-def split-conv closed-segment-def mem-Collect-eq*
apply *rule*
apply (*elim exE conjE*)
apply (*subst dist-triangle-eq*)
proof −
fix *u*
assume *as*: *x* = (*1* − *u*) ∗*R a* + *u* ∗*R b 0* ≤ *u u* ≤ *1*
then have ∗: *a* − *x* = *u* ∗*R* (*a* − *b*) *x* − *b* = (*1* − *u*) ∗*R* (*a* − *b*)
unfolding *as*(*1*) **by** (*auto simp add*:*algebra-simps*)
show *norm* (*a* − *x*) ∗*R* (*x* − *b*) = *norm* (*x* − *b*) ∗*R* (*a* − *x*)
unfolding *norm-minus-commute*[*of x a*] ∗ **using** *as*(*2*,*3*)
by (*auto simp add*: *field-simps*)
next
assume *as*: *dist a b* = *dist a x* + *dist x b*
have *norm* (*a* − *x*) / *norm* (*a* − *b*) ≤ *1*
using *Fal2* **unfolding** *as*[*unfolded dist-norm*] *norm-ge-zero* **by** *auto*
then show ∃ *u*. *x* = (*1* − *u*) ∗*R a* + *u* ∗*R b* ∧ *0* ≤ *u* ∧ *u* ≤ *1*
apply (*rule-tac x*=*dist a x* / *dist a b* **in** *exI*)
unfolding *dist-norm*
apply (*subst euclidean-eq-iff*)
apply *rule*
defer

```
      apply rule
      prefer 3
      apply rule
    proof −
      fix i :: 'a
      assume i: i ∈ Basis
      have ((1 − norm (a − x) / norm (a − b)) *R a + (norm (a − x) / norm
(a − b)) *R b) · i =
        ((norm (a − b) − norm (a − x)) * (a · i) + norm (a − x) * (b · i)) /
norm (a − b)
        using Fal by (auto simp add: field-simps inner-simps)
      also have ... = x·i
        apply (rule divide-eq-imp[OF Fal])
        unfolding as[unfolded dist-norm]
        using as[unfolded dist-triangle-eq]
        apply −
        apply (subst (asm) euclidean-eq-iff)
        using i
        apply (erule-tac x=i in ballE)
        apply (auto simp add: field-simps inner-simps)
        done
      finally show x · i =
        ((1 − norm (a − x) / norm (a − b)) *R a + (norm (a − x) / norm (a −
b)) *R b) · i
        by auto
    qed (insert Fal2, auto)
  qed
qed

lemma between-midpoint:
  fixes a :: 'a::euclidean-space
  shows between (a,b) (midpoint a b) (is ?t1)
    and between (b,a) (midpoint a b) (is ?t2)
proof −
  have *: ⋀x y z. x = (1/2::real) *R z ⟹ y = (1/2) *R z ⟹ norm z = norm
x + norm y
    by auto
  show ?t1 ?t2
    unfolding between midpoint-def dist-norm
    apply(rule-tac[!] *)
    unfolding euclidean-eq-iff[where 'a='a]
    apply (auto simp add: field-simps inner-simps)
    done
qed

lemma between-mem-convex-hull:
  between (a,b) x ⟷ x ∈ convex hull {a,b}
  unfolding between-mem-segment segment-convex-hull ..
```

19.30 Shrinking towards the interior of a convex set

lemma *mem-interior-convex-shrink*:
fixes s :: *'a*::*euclidean-space set*
assumes *convex s*
and $c \in$ *interior s*
and $x \in s$
and $0 < e$
and $e \leq 1$
shows $x - e *_R (x - c) \in$ *interior s*
proof −
obtain d **where** $d > 0$ **and** d: *ball c d* $\subseteq s$
using *assms*(2) **unfolding** *mem-interior* **by** *auto*
show *?thesis*
unfolding *mem-interior*
apply (*rule-tac x*=*e*∗*d* **in** *exI*)
apply *rule*
defer
unfolding *subset-eq Ball-def mem-ball*
proof (*rule*, *rule*)
fix y
assume *as*: *dist* $(x - e *_R (x - c))\ y < e * d$
have ∗: $y = (1 - (1 - e)) *_R ((1 / e) *_R y - ((1 - e) / e) *_R x) + (1 - e) *_R x$
using ‹$e > 0$› **by** (*auto simp add*: *scaleR-left-diff-distrib scaleR-right-diff-distrib*)
have *dist c* $((1 / e) *_R y - ((1 - e) / e) *_R x) = |1/e| * norm\ (e *_R c - y + (1 - e) *_R x)$
unfolding *dist-norm*
unfolding *norm-scaleR*[*symmetric*]
apply (*rule arg-cong*[**where** *f*=*norm*])
using ‹$e > 0$›
by (*auto simp add*: *euclidean-eq-iff*[**where** *'a*=*'a*] *field-simps inner-simps*)
also have $\ldots = |1/e| * norm\ (x - e *_R (x - c) - y)$
by (*auto intro*!:*arg-cong*[**where** *f*=*norm*] *simp add*: *algebra-simps*)
also have $\ldots < d$
using *as*[*unfolded dist-norm*] **and** ‹$e > 0$›
by (*auto simp add*:*pos-divide-less-eq*[*OF* ‹$e > 0$›] *mult.commute*)
finally show $y \in s$
apply (*subst* ∗)
apply (*rule assms*(1)[*unfolded convex-alt*,*rule-format*])
apply (*rule d*[*unfolded subset-eq*,*rule-format*])
unfolding *mem-ball*
using *assms*(3−5)
apply *auto*
done
qed (*insert* ‹*e*>0› ‹*d*>0›, *auto*)
qed

lemma *mem-interior-closure-convex-shrink*:
fixes s :: *'a*::*euclidean-space set*

```
  assumes convex s
    and c ∈ interior s
    and x ∈ closure s
    and 0 < e
    and e ≤ 1
  shows x − e *R (x − c) ∈ interior s
proof −
  obtain d where d > 0 and d: ball c d ⊆ s
    using assms(2) unfolding mem-interior by auto
  have ∃y∈s. norm (y − x) * (1 − e) < e * d
  proof (cases x ∈ s)
    case True
    then show ?thesis
      using ‹e > 0› ‹d > 0›
      apply (rule-tac bexI[where x=x])
      apply (auto)
      done
  next
    case False
    then have x: x islimpt s
      using assms(3)[unfolded closure-def] by auto
    show ?thesis
    proof (cases e = 1)
      case True
      obtain y where y ∈ s y ≠ x dist y x < 1
        using x[unfolded islimpt-approachable,THEN spec[where x=1]] by auto
      then show ?thesis
        apply (rule-tac x=y in bexI)
        unfolding True
        using ‹d > 0›
        apply auto
        done
    next
      case False
      then have 0 < e * d / (1 − e) and ∗: 1 − e > 0
        using ‹e ≤ 1› ‹e > 0› ‹d > 0› by auto
      then obtain y where y ∈ s y ≠ x dist y x < e * d / (1 − e)
       using x[unfolded islimpt-approachable,THEN spec[where x=e*d / (1 − e)]]
by auto
      then show ?thesis
        apply (rule-tac x=y in bexI)
        unfolding dist-norm
        using pos-less-divide-eq[OF ∗]
        apply auto
        done
    qed
  qed
  then obtain y where y ∈ s and y: norm (y − x) * (1 − e) < e * d
    by auto
```

def $z \equiv c + ((1 - e) \,/\, e) *_R (x - y)$
have $*$: $x - e *_R (x - c) = y - e *_R (y - z)$
unfolding *z-def* **using** ‹$e > 0$›
by (*auto simp add*: *scaleR-right-diff-distrib scaleR-right-distrib scaleR-left-diff-distrib*)
have $z \in$ *interior s*
apply (*rule interior-mono*[*OF d*,*unfolded subset-eq*,*rule-format*])
unfolding *interior-open*[*OF open-ball*] *mem-ball z-def dist-norm* **using** *y* **and** *assms*(*4*,*5*)
apply (*auto simp add*:*field-simps norm-minus-commute*)
done
then show *?thesis*
unfolding $*$
apply −
apply (*rule mem-interior-convex-shrink*)
using *assms*(*1*,*4*−*5*) ‹*y*∈*s*›
apply *auto*
done
qed

19.31 Some obvious but surprisingly hard simplex lemmas

lemma *simplex*:
assumes *finite s*
and $0 \notin s$
shows *convex hull* (*insert 0 s*) =
$\{y.\ (\exists u.\ (\forall x{\in}s.\ 0 \le u\ x) \wedge \mathit{setsum}\ u\ s \le 1 \wedge \mathit{setsum}\ (\lambda x.\ u\ x *_R x)\ s = y)\}$
unfolding *convex-hull-finite*[*OF finite.insertI*[*OF assms*(*1*)]]
apply (*rule set-eqI*, *rule*)
unfolding *mem-Collect-eq*
apply (*erule-tac*[!] *exE*)
apply (*erule-tac*[!] *conjE*)+
unfolding *setsum-clauses*(*2*)[*OF assms*(*1*)]
apply (*rule-tac x*=*u* **in** *exI*)
defer
apply (*rule-tac x*=$\lambda x.$ *if x = 0 then 1 − setsum u s else u x* **in** *exI*)
using *assms*(*2*)
unfolding *if-smult* **and** *setsum-delta-notmem*[*OF assms*(*2*)]
apply *auto*
done

lemma *substd-simplex*:
assumes *d*: $d \subseteq$ *Basis*
shows *convex hull* (*insert 0 d*) =
$\{x.\ (\forall i{\in}\mathit{Basis}.\ 0 \le x{\cdot}i) \wedge (\sum i{\in}d.\ x{\cdot}i) \le 1 \wedge (\forall i{\in}\mathit{Basis}.\ i \notin d \longrightarrow x{\cdot}i = 0)\}$
(**is** *convex hull* (*insert 0 ?p*) = *?s*)
proof −
let *?D* = *d*
have $0 \notin$ *?p*

```
    using assms by (auto simp: image-def)
  from d have finite d
    by (blast intro: finite-subset finite-Basis)
  show ?thesis
    unfolding simplex[OF ‹finite d› ‹0 ∉ ?p›]
    apply (rule set-eqI)
    unfolding mem-Collect-eq
    apply rule
    apply (elim exE conjE)
    apply (erule-tac[2] conjE)+
  proof −
    fix x :: 'a::euclidean-space
    fix u
    assume as: ∀ x∈?D. 0 ≤ u x setsum u ?D ≤ 1 (∑ x∈?D. u x *R x) = x
    have *: ∀ i∈Basis. i:d ⟶ u i = x•i
      and (∀ i∈Basis. i ∉ d ⟶ x • i = 0)
      using as(3)
      unfolding substdbasis-expansion-unique[OF assms]
      by auto
    then have **: setsum u ?D = setsum (op • x) ?D
      apply −
      apply (rule setsum.cong)
      using assms
      apply auto
      done
    have (∀ i∈Basis. 0 ≤ x•i) ∧ setsum (op • x) ?D ≤ 1
    proof (rule,rule)
      fix i :: 'a
      assume i: i ∈ Basis
      have i ∈ d ⟹ 0 ≤ x•i
        unfolding *[rule-format,OF i,symmetric]
        apply (rule-tac as(1)[rule-format])
        apply auto
        done
      moreover have i ∉ d ⟹ 0 ≤ x•i
        using ‹(∀ i∈Basis. i ∉ d ⟶ x • i = 0)›[rule-format, OF i] by auto
      ultimately show 0 ≤ x•i by auto
    qed (insert as(2)[unfolded **], auto)
    then show (∀ i∈Basis. 0 ≤ x•i) ∧ setsum (op • x) ?D ≤ 1 ∧ (∀ i∈Basis. i ∉
d ⟶ x • i = 0)
      using ‹(∀ i∈Basis. i ∉ d ⟶ x • i = 0)› by auto
  next
    fix x :: 'a::euclidean-space
    assume as: ∀ i∈Basis. 0 ≤ x • i setsum (op • x) ?D ≤ 1 (∀ i∈Basis. i ∉ d
⟶ x • i = 0)
    show ∃ u. (∀ x∈?D. 0 ≤ u x) ∧ setsum u ?D ≤ 1 ∧ (∑ x∈?D. u x *R x) = x
      using as d
      unfolding substdbasis-expansion-unique[OF assms]
      apply (rule-tac x=inner x in exI)
```

```
    apply auto
    done
  qed
qed

lemma std-simplex:
  convex hull (insert 0 Basis) =
    {x::'a::euclidean-space. (∀ i∈Basis. 0 ≤ x•i) ∧ setsum (λi. x•i) Basis ≤ 1}
  using substd-simplex[of Basis] by auto

lemma interior-std-simplex:
  interior (convex hull (insert 0 Basis)) =
    {x::'a::euclidean-space. (∀ i∈Basis. 0 < x•i) ∧ setsum (λi. x•i) Basis < 1}
  apply (rule set-eqI)
  unfolding mem-interior std-simplex
  unfolding subset-eq mem-Collect-eq Ball-def mem-ball
  unfolding Ball-def[symmetric]
  apply rule
  apply (elim exE conjE)
  defer
  apply (erule conjE)
proof −
  fix x :: 'a
  fix e
  assume e > 0 and as: ∀ xa. dist x xa < e ⟶ (∀ x∈Basis. 0 ≤ xa · x) ∧ setsum
(op · xa) Basis ≤ 1
  show (∀ xa∈Basis. 0 < x · xa) ∧ setsum (op · x) Basis < 1
    apply safe
  proof −
    fix i :: 'a
    assume i: i ∈ Basis
    then show 0 < x · i
      using as[THEN spec[where x=x − (e / 2) *R i]] and ‹e > 0›
      unfolding dist-norm
      by (auto elim!: ballE[where x=i] simp: inner-simps)
  next
    have **: dist x (x + (e / 2) *R (SOME i. i∈Basis)) < e using ‹e > 0›
      unfolding dist-norm
      by (auto intro!: mult-strict-left-mono simp: SOME-Basis)
    have ⋀i. i ∈ Basis ⟹ (x + (e / 2) *R (SOME i. i∈Basis)) · i =
      x•i + (if i = (SOME i. i∈Basis) then e/2 else 0)
      by (auto simp: SOME-Basis inner-Basis inner-simps)
    then have *: setsum (op · (x + (e / 2) *R (SOME i. i∈Basis))) Basis =
      setsum (λi. x•i + (if (SOME i. i∈Basis) = i then e/2 else 0)) Basis
      apply (rule-tac setsum.cong)
      apply auto
      done
  have setsum (op · x) Basis < setsum (op · (x + (e / 2) *R (SOME i. i∈Basis)))
Basis
```

```
    unfolding * setsum.distrib
    using ‹e > 0› DIM-positive[where 'a='a]
    apply (subst setsum.delta')
    apply (auto simp: SOME-Basis)
    done
  also have ... ≤ 1
    using **
    apply (drule-tac as[rule-format])
    apply auto
    done
  finally show setsum (op · x) Basis < 1 by auto
 qed
next
 fix x :: 'a
 assume as: ∀ i∈Basis. 0 < x · i setsum (op · x) Basis < 1
 obtain a :: 'b where a ∈ UNIV using UNIV-witness ..
 let ?d = (1 - setsum (op · x) Basis) / real (DIM('a))
 have Min ((op · x) ` Basis) > 0
  apply (rule Min-grI)
  using as(1)
  apply auto
  done
 moreover have ?d > 0
  using as(2) by (auto simp: Suc-le-eq DIM-positive)
 ultimately show ∃ e>0. ∀ y. dist x y < e ⟶ (∀ i∈Basis. 0 ≤ y · i) ∧ setsum
(op · y) Basis ≤ 1
  apply (rule-tac x=min (Min ((op · x) ` Basis)) D for D in exI)
  apply rule
  defer
  apply (rule, rule)
 proof -
  fix y
  assume y: dist x y < min (Min (op · x ` Basis)) ?d
  have setsum (op · y) Basis ≤ setsum (λi. x·i + ?d) Basis
  proof (rule setsum-mono)
   fix i :: 'a
   assume i: i ∈ Basis
   then have |y·i - x·i| < ?d
    apply -
    apply (rule le-less-trans)
    using Basis-le-norm[OF i, of y - x]
    using y[unfolded min-less-iff-conj dist-norm, THEN conjunct2]
    apply (auto simp add: norm-minus-commute inner-diff-left)
    done
   then show y · i ≤ x · i + ?d by auto
  qed
  also have ... ≤ 1
   unfolding setsum.distrib setsum-constant
   by (auto simp add: Suc-le-eq)
```

```
    finally show (∀ i∈Basis. 0 ≤ y • i) ∧ setsum (op • y) Basis ≤ 1
    proof safe
      fix i :: 'a
      assume i: i ∈ Basis
      have norm (x − y) < x•i
        apply (rule less-le-trans)
        apply (rule y[unfolded min-less-iff-conj dist-norm, THEN conjunct1])
        using i
        apply auto
        done
      then show 0 ≤ y•i
        using Basis-le-norm[OF i, of x − y] and as(1)[rule-format, OF i]
        by (auto simp: inner-simps)
    qed
  qed auto
qed

lemma interior-std-simplex-nonempty:
  obtains a :: 'a::euclidean-space where
    a ∈ interior(convex hull (insert 0 Basis))
proof −
  let ?D = Basis :: 'a set
  let ?a = setsum (λb::'a. inverse (2 * real DIM('a)) *R b) Basis
  {
    fix i :: 'a
    assume i: i ∈ Basis
    have ?a • i = inverse (2 * real DIM('a))
      by (rule trans[of - setsum (λj. if i = j then inverse (2 * real DIM('a)) else
0) ?D])
        (simp-all add: setsum.If-cases i) }
  note ** = this
  show ?thesis
    apply (rule that[of ?a])
    unfolding interior-std-simplex mem-Collect-eq
  proof safe
    fix i :: 'a
    assume i: i ∈ Basis
    show 0 < ?a • i
      unfolding **[OF i] by (auto simp add: Suc-le-eq DIM-positive)
  next
    have setsum (op • ?a) ?D = setsum (λi. inverse (2 * real DIM('a))) ?D
      apply (rule setsum.cong)
      apply rule
      apply auto
      done
    also have ... < 1
      unfolding setsum-constant divide-inverse[symmetric]
      by (auto simp add: field-simps)
    finally show setsum (op • ?a) ?D < 1 by auto
```

```
  qed
qed

lemma rel-interior-substd-simplex:
  assumes d: d ⊆ Basis
  shows rel-interior (convex hull (insert 0 d)) =
    {x::'a::euclidean-space. (∀ i∈d. 0 < x•i) ∧ (∑ i∈d. x•i) < 1 ∧ (∀ i∈Basis. i ∉
d ⟶ x•i = 0)}
  (is rel-interior (convex hull (insert 0 ?p)) = ?s)
proof −
  have finite d
    apply (rule finite-subset)
    using assms
    apply auto
    done
  show ?thesis
  proof (cases d = {})
    case True
    then show ?thesis
      using rel-interior-sing using euclidean-eq-iff[of - 0] by auto
  next
    case False
    have h0: affine hull (convex hull (insert 0 ?p)) =
      {x::'a::euclidean-space. (∀ i∈Basis. i ∉ d ⟶ x•i = 0)}
      using affine-hull-convex-hull affine-hull-substd-basis assms by auto
    have aux: ⋀x::'a. ∀ i∈Basis. (∀ i∈d. 0 ≤ x•i) ∧ (∀ i∈Basis. i ∉ d ⟶ x•i =
0) ⟶ 0 ≤ x•i
      by auto
    {
      fix x :: 'a::euclidean-space
      assume x: x ∈ rel-interior (convex hull (insert 0 ?p))
      then obtain e where e0: e > 0 and
        ball x e ∩ {xa. (∀ i∈Basis. i ∉ d ⟶ xa•i = 0)} ⊆ convex hull (insert 0 ?p)
        using mem-rel-interior-ball[of x convex hull (insert 0 ?p)] h0 by auto
      then have as: ∀ xa. dist x xa < e ∧ (∀ i∈Basis. i ∉ d ⟶ xa•i = 0) ⟶
        (∀ i∈d. 0 ≤ xa • i) ∧ setsum (op • xa) d ≤ 1
        unfolding ball-def unfolding substd-simplex[OF assms] using assms by
auto
      have x0: (∀ i∈Basis. i ∉ d ⟶ x•i = 0)
        using x rel-interior-subset substd-simplex[OF assms] by auto
      have (∀ i∈d. 0 < x • i) ∧ setsum (op • x) d < 1 ∧ (∀ i∈Basis. i ∉ d ⟶
x•i = 0)
        apply rule
        apply rule
      proof −
        fix i :: 'a
        assume i ∈ d
        then have ∀ ia∈d. 0 ≤ (x − (e / 2) *R i) • ia
          apply −
```

```
      apply (rule as[rule-format,THEN conjunct1])
      unfolding dist-norm
      using d ‹e > 0› x0
      apply (auto simp: inner-simps inner-Basis)
      done
    then show 0 < x • i
      apply (erule-tac x=i in ballE)
      using ‹e > 0› ‹i ∈ d› d
      apply (auto simp: inner-simps inner-Basis)
      done
  next
    obtain a where a: a ∈ d
      using ‹d ≠ {}› by auto
    then have **: dist x (x + (e / 2) *R a) < e
      using ‹e > 0› norm-Basis[of a] d
      unfolding dist-norm
      by auto
    have ⋀i. i ∈ Basis ⟹ (x + (e / 2) *R a) • i = x•i + (if i = a then e/2
else 0)
      using a d by (auto simp: inner-simps inner-Basis)
    then have *: setsum (op • (x + (e / 2) *R a)) d =
      setsum (λi. x•i + (if a = i then e/2 else 0)) d
      using d by (intro setsum.cong) auto
    have a ∈ Basis
      using ‹a ∈ d› d by auto
    then have h1: (∀ i∈Basis. i ∉ d ⟶ (x + (e / 2) *R a) • i = 0)
      using x0 d ‹a∈d› by (auto simp add: inner-add-left inner-Basis)
    have setsum (op • x) d < setsum (op • (x + (e / 2) *R a)) d
      unfolding * setsum.distrib
      using ‹e > 0› ‹a ∈ d›
      using ‹finite d›
      by (auto simp add: setsum.delta')
    also have ... ≤ 1
      using ** h1 as[rule-format, of x + (e / 2) *R a]
      by auto
    finally show setsum (op • x) d < 1 ∧ (∀ i∈Basis. i ∉ d ⟶ x•i = 0)
      using x0 by auto
  qed
}
moreover
{
  fix x :: 'a::euclidean-space
  assume as: x ∈ ?s
  have ∀ i. 0 < x•i ∨ 0 = x•i ⟶ 0 ≤ x•i
    by auto
  moreover have ∀ i. i ∈ d ∨ i ∉ d by auto
  ultimately
  have ∀ i. (∀ i∈d. 0 < x•i) ∧ (∀ i. i ∉ d ⟶ x•i = 0) ⟶ 0 ≤ x•i
    by metis
```

then have *h2*: $x \in$ *convex hull* (*insert 0 ?p*)
using *as assms*
unfolding *substd-simplex*[*OF assms*] **by** *fastforce*
obtain *a* **where** *a*: $a \in d$
using ‹$d \neq \{\}$› **by** *auto*
let *?d* = (*1 − setsum* (*op ·* *x*) *d*) */ real* (*card d*)
have *0 < card d* **using** ‹$d \neq \{\}$› ‹*finite d*›
by (*simp add*: *card-gt-0-iff*)
have *Min* ((*op ·* *x*) ‘ *d*) *> 0*
using *as* ‹$d \neq \{\}$› ‹*finite d*› **by** (*simp add*: *Min-grI*)
moreover have *?d > 0* **using** *as* **using** ‹*0 < card d*› **by** *auto*
ultimately have *h3*: *min* (*Min* ((*op ·* *x*) ‘ *d*)) *?d > 0*
by *auto*

have $x \in$ *rel-interior* (*convex hull* (*insert 0 ?p*))
unfolding *rel-interior-ball mem-Collect-eq h0*
apply (*rule*,*rule h2*)
unfolding *substd-simplex*[*OF assms*]
apply (*rule-tac x=min* (*Min* ((*op ·* *x*) ‘ *d*)) *?d* **in** *exI*)
apply (*rule*, *rule h3*)
apply *safe*
unfolding *mem-ball*
proof −
fix *y* :: *'a*
assume *y*: *dist x y < min* (*Min* (*op ·* *x* ‘ *d*)) *?d*
assume *y2*: $\forall i \in Basis.\ i \notin d \longrightarrow y{\cdot}i = 0$
have *setsum* (*op ·* *y*) *d* ≤ *setsum* (λ*i*. *x·i* + *?d*) *d*
proof (*rule setsum-mono*)
fix *i*
assume $i \in d$
with *d* **have** *i*: $i \in Basis$
by *auto*
have $|y{\cdot}i - x{\cdot}i| <$ *?d*
apply (*rule le-less-trans*)
using *Basis-le-norm*[*OF i*, *of y − x*]
using *y*[*unfolded min-less-iff-conj dist-norm*, *THEN conjunct2*]
apply (*auto simp add*: *norm-minus-commute inner-simps*)
done
then show *y · i* ≤ *x · i* + *?d* **by** *auto*
qed
also have ... ≤ *1*
unfolding *setsum.distrib setsum-constant* **using** ‹*0 < card d*›
by *auto*
finally show *setsum* (*op ·* *y*) *d* ≤ *1* **.**

fix *i* :: *'a*
assume *i*: $i \in Basis$
then show *0* ≤ *y·i*
proof (*cases i∈d*)

```
        case True
        have norm (x − y) < x•i
          using y[unfolded min-less-iff-conj dist-norm, THEN conjunct1]
          using Min-gr-iff[of op • x ‘ d norm (x − y)] ⟨0 < card d⟩ ⟨i:d⟩
          by (simp add: card-gt-0-iff)
        then show 0 ≤ y•i
          using Basis-le-norm[OF i, of x − y] and as(1)[rule-format]
          by (auto simp: inner-simps)
      qed (insert y2, auto)
    qed
   }
   ultimately have
    ⋀x. x ∈ rel-interior (convex hull insert 0 d) ⟷
      x ∈ {x. (∀i∈d. 0 < x • i) ∧ setsum (op • x) d < 1 ∧ (∀i∈Basis. i ∉ d
⟶ x • i = 0)}
    by blast
   then show ?thesis by (rule set-eqI)
  qed
qed

lemma rel-interior-substd-simplex-nonempty:
  assumes d ≠ {}
    and d ⊆ Basis
  obtains a :: 'a::euclidean-space
    where a ∈ rel-interior (convex hull (insert 0 d))
proof −
  let ?D = d
  let ?a = setsum (λb::'a::euclidean-space. inverse (2 ∗ real (card d)) ∗R b) ?D
  have finite d
    apply (rule finite-subset)
    using assms(2)
    apply auto
    done
  then have d1: 0 < real (card d)
    using ⟨d ≠ {}⟩ by auto
  {
    fix i
    assume i ∈ d
    have ?a • i = inverse (2 ∗ real (card d))
      apply (rule trans[of - setsum (λj. if i = j then inverse (2 ∗ real (card d))
else 0) ?D])
     unfolding inner-setsum-left
     apply (rule setsum.cong)
     using ⟨i ∈ d⟩ ⟨finite d⟩ setsum.delta'[of d i (λk. inverse (2 ∗ real (card d)))]
      d1 assms(2)
     by (auto simp: inner-Basis set-rev-mp[OF - assms(2)])
  }
  note ∗∗ = this
  show ?thesis
```

```
    apply (rule that[of ?a])
    unfolding rel-interior-substd-simplex[OF assms(2)] mem-Collect-eq
  proof safe
    fix i
    assume i ∈ d
    have 0 < inverse (2 * real (card d))
      using d1 by auto
    also have ... = ?a · i using **[of i] ‹i ∈ d›
      by auto
    finally show 0 < ?a · i by auto
  next
    have setsum (op · ?a) ?D = setsum (λi. inverse (2 * real (card d))) ?D
      by (rule setsum.cong) (rule refl, rule **)
    also have ... < 1
      unfolding setsum-constant divide-real-def[symmetric]
      by (auto simp add: field-simps)
    finally show setsum (op · ?a) ?D < 1 by auto
  next
    fix i
    assume i ∈ Basis and i ∉ d
    have ?a ∈ span d
    proof (rule span-setsum[of d (λb. b /R (2 * real (card d))) d])
      {
        fix x :: 'a::euclidean-space
        assume x ∈ d
        then have x ∈ span d
          using span-superset[of - d] by auto
        then have x /R (2 * real (card d)) ∈ span d
          using span-mul[of x d (inverse (real (card d)) / 2)] by auto
      }
      then show ∀x∈d. x /R (2 * real (card d)) ∈ span d
        by auto
    qed
    then show ?a · i = 0
      using ‹i ∉ d› unfolding span-substd-basis[OF assms(2)] using ‹i ∈ Basis›
by auto
  qed
qed
```

19.32 Relative interior of convex set

```
lemma rel-interior-convex-nonempty-aux:
  fixes S :: 'n::euclidean-space set
  assumes convex S
    and 0 ∈ S
  shows rel-interior S ≠ {}
proof (cases S = {0})
  case True
  then show ?thesis using rel-interior-sing by auto
```

next
case *False*
obtain B **where** B: *independent* $B \wedge B \leq S \wedge S \leq$ *span* $B \wedge$ *card* $B =$ *dim* S
using *basis-exists*[*of* S] **by** *auto*
then have $B \neq \{\}$
using B *assms* ‹$S \neq \{0\}$› *span-empty* **by** *auto*
have *insert 0* $B \leq$ *span* B
using *subspace-span*[*of* B] *subspace-0*[*of span* B] *span-inc* **by** *auto*
then have *span* (*insert 0* B) $\leq$ *span* B
using *span-span*[*of* B] *span-mono*[*of insert 0* B *span* B] **by** *blast*
then have *convex hull insert 0* $B \leq$ *span* B
using *convex-hull-subset-span*[*of insert 0* B] **by** *auto*
then have *span* (*convex hull insert 0* B) $\leq$ *span* B
using *span-span*[*of* B] *span-mono*[*of convex hull insert 0* B *span* B] **by** *blast*
then have $*$: *span* (*convex hull insert 0* B) = *span* B
using *span-mono*[*of* B *convex hull insert 0* B] *hull-subset*[*of insert 0* B] **by** *auto*
then have *span* (*convex hull insert 0* B) = *span* S
using B *span-mono*[*of* B S] *span-mono*[*of* S *span* B] *span-span*[*of* B] **by** *auto*
moreover have $0 \in$ *affine hull* (*convex hull insert 0* B)
using *hull-subset*[*of convex hull insert 0* B] *hull-subset*[*of insert 0* B] **by** *auto*
ultimately have $**$: *affine hull* (*convex hull insert 0* B) = *affine hull* S
using *affine-hull-span-0*[*of convex hull insert 0* B] *affine-hull-span-0*[*of* S]
assms hull-subset[*of* S]
by *auto*
obtain d **and** $f :: 'n \Rightarrow 'n$ **where**
fd: *card* $d =$ *card* B *linear* f f ‘ $B = d$
f ‘ *span* $B = \{x.\ \forall i \in Basis.\ i \notin d \longrightarrow x \cdot i = (0::real)\} \wedge$ *inj-on* f (*span* B)
and d: $d \subseteq Basis$
using *basis-to-substdbasis-subspace-isomorphism*[*of* B,*OF* -] B **by** *auto*
then have *bounded-linear* f
using *linear-conv-bounded-linear* **by** *auto*
have $d \neq \{\}$
using fd B ‹$B \neq \{\}$› **by** *auto*
have *insert 0* $d = f$ ‘ (*insert 0* B)
using fd *linear-0* **by** *auto*
then have (*convex hull* (*insert 0* d)) = f ‘ (*convex hull* (*insert 0* B))
using *convex-hull-linear-image*[*of* f (*insert 0* d)]
convex-hull-linear-image[*of* f (*insert 0* B)] ‹*linear* f›
by *auto*
moreover have *rel-interior* (f ‘ (*convex hull insert 0* B)) =
f ‘ *rel-interior* (*convex hull insert 0* B)
apply (*rule rel-interior-injective-on-span-linear-image*[*of* f (*convex hull insert 0* B)])
using ‹*bounded-linear* f› fd $*$
apply *auto*
done
ultimately have *rel-interior* (*convex hull insert 0* B) $\neq \{\}$
using *rel-interior-substd-simplex-nonempty*[*OF* ‹$d \neq \{\}$› d]
apply *auto*

```
    apply blast
    done
  moreover have convex hull (insert 0 B) ⊆ S
    using B assms hull-mono[of insert 0 B S convex] convex-hull-eq
    by auto
  ultimately show ?thesis
    using subset-rel-interior[of convex hull insert 0 B S] ** by auto
qed

lemma rel-interior-eq-empty:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows rel-interior S = {} ⟷ S = {}
proof −
  {
    assume S ≠ {}
    then obtain a where a ∈ S by auto
    then have 0 ∈ op + (−a) ` S
      using assms exI[of (λx. x ∈ S ∧ − a + x = 0) a] by auto
    then have rel-interior (op + (−a) ` S) ≠ {}
      using rel-interior-convex-nonempty-aux[of op + (−a) ` S]
        convex-translation[of S −a] assms
      by auto
    then have rel-interior S ≠ {}
      using rel-interior-translation by auto
  }
  then show ?thesis
    using rel-interior-empty by auto
qed

lemma convex-rel-interior:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows convex (rel-interior S)
proof −
  {
    fix x y and u :: real
    assume assm: x ∈ rel-interior S y ∈ rel-interior S 0 ≤ u u ≤ 1
    then have x ∈ S
      using rel-interior-subset by auto
    have x − u *R (x−y) ∈ rel-interior S
    proof (cases 0 = u)
      case False
      then have 0 < u using assm by auto
      then show ?thesis
        using assm rel-interior-convex-shrink[of S y x u] assms ‹x ∈ S› by auto
    next
      case True
      then show ?thesis using assm by auto
```

```
    qed
    then have (1 − u) *R x + u *R y ∈ rel-interior S
      by (simp add: algebra-simps)
  }
  then show ?thesis
    unfolding convex-alt by auto
qed

lemma convex-closure-rel-interior:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows closure (rel-interior S) = closure S
proof −
  have h1: closure (rel-interior S) ≤ closure S
    using closure-mono[of rel-interior S S] rel-interior-subset[of S] by auto
  show ?thesis
  proof (cases S = {})
    case False
    then obtain a where a: a ∈ rel-interior S
      using rel-interior-eq-empty assms by auto
    { fix x
      assume x: x ∈ closure S
      {
        assume x = a
        then have x ∈ closure (rel-interior S)
          using a unfolding closure-def by auto
      }
      moreover
      {
        assume x ≠ a
        {
          fix e :: real
          assume e > 0
          def e1 ≡ min 1 (e/norm (x − a))
          then have e1: e1 > 0 e1 ≤ 1 e1 * norm (x − a) ≤ e
            using ‹x ≠ a› ‹e > 0› le-divide-eq[of e1 e norm (x − a)]
            by simp-all
          then have *: x − e1 *R (x − a) : rel-interior S
            using rel-interior-closure-convex-shrink[of S a x e1] assms x a e1-def
            by auto
          have ∃ y. y ∈ rel-interior S ∧ y ≠ x ∧ dist y x ≤ e
            apply (rule-tac x=x − e1 *R (x − a) in exI)
            using * e1 dist-norm[of x − e1 *R (x − a) x] ‹x ≠ a›
            apply simp
            done
        }
        then have x islimpt rel-interior S
          unfolding islimpt-approachable-le by auto
        then have x ∈ closure(rel-interior S)
```

```
        unfolding closure-def by auto
      }
      ultimately have x ∈ closure(rel-interior S) by auto
    }
    then show ?thesis using h1 by auto
  next
    case True
    then have rel-interior S = {}
      using rel-interior-empty by auto
    then have closure (rel-interior S) = {}
      using closure-empty by auto
    with True show ?thesis by auto
  qed
qed

lemma rel-interior-same-affine-hull:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows affine hull (rel-interior S) = affine hull S
  by (metis assms closure-same-affine-hull convex-closure-rel-interior)

lemma rel-interior-aff-dim:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows aff-dim (rel-interior S) = aff-dim S
  by (metis aff-dim-affine-hull2 assms rel-interior-same-affine-hull)

lemma rel-interior-rel-interior:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows rel-interior (rel-interior S) = rel-interior S
proof −
  have openin (subtopology euclidean (affine hull (rel-interior S))) (rel-interior S)
    using opein-rel-interior[of S] rel-interior-same-affine-hull[of S] assms by auto
  then show ?thesis
    using rel-interior-def by auto
qed

lemma rel-interior-rel-open:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows rel-open (rel-interior S)
  unfolding rel-open-def using rel-interior-rel-interior assms by auto

lemma convex-rel-interior-closure-aux:
  fixes x y z :: 'n::euclidean-space
  assumes 0 < a 0 < b (a + b) *R z = a *R x + b *R y
  obtains e where 0 < e e ≤ 1 z = y − e *R (y − x)
proof −
```

```
  def e ≡ a / (a + b)
  have z = (1 / (a + b)) *R ((a + b) *R z)
    apply auto
    using assms
    apply simp
    done
  also have ... = (1 / (a + b)) *R (a *R x + b *R y)
    using assms scaleR-cancel-left[of 1/(a+b) (a + b) *R z a *R x + b *R y]
    by auto
  also have ... = y − e *R (y−x)
    using e-def
    apply (simp add: algebra-simps)
    using scaleR-left-distrib[of a/(a+b) b/(a+b) y] assms add-divide-distrib[of a b
a+b]
    apply auto
    done
  finally have z = y − e *R (y−x)
    by auto
  moreover have e > 0 using e-def assms by auto
  moreover have e ≤ 1 using e-def assms by auto
  ultimately show ?thesis using that[of e] by auto
qed

lemma convex-rel-interior-closure:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows rel-interior (closure S) = rel-interior S
proof (cases S = {})
  case True
  then show ?thesis
    using assms rel-interior-eq-empty by auto
next
  case False
  have rel-interior (closure S) ⊇ rel-interior S
    using subset-rel-interior[of S closure S] closure-same-affine-hull closure-subset
    by auto
  moreover
  {
    fix z
    assume z: z ∈ rel-interior (closure S)
    obtain x where x: x ∈ rel-interior S
      using ‹S ≠ {}› assms rel-interior-eq-empty by auto
    have z ∈ rel-interior S
    proof (cases x = z)
      case True
      then show ?thesis using x by auto
    next
      case False
      obtain e where e: e > 0 cball z e ∩ affine hull closure S ≤ closure S
```

```
      using z rel-interior-cball[of closure S] by auto
    hence *: 0 < e/norm(z−x) using e False by auto
    def y ≡ z + (e/norm(z−x)) *R (z−x)
    have yball: y ∈ cball z e
      using mem-cball y-def dist-norm[of z y] e by auto
    have x ∈ affine hull closure S
      using x rel-interior-subset-closure hull-inc[of x closure S] by auto
    moreover have z ∈ affine hull closure S
      using z rel-interior-subset hull-subset[of closure S] by auto
    ultimately have y ∈ affine hull closure S
      using y-def affine-affine-hull[of closure S]
        mem-affine-3-minus [of affine hull closure S z z x e/norm(z−x)] by auto
    then have y ∈ closure S using e yball by auto
    have (1 + (e/norm(z−x))) *R z = (e/norm(z−x)) *R x + y
      using y-def by (simp add: algebra-simps)
    then obtain e1 where 0 < e1 e1 ≤ 1 z = y − e1 *R (y − x)
      using * convex-rel-interior-closure-aux[of e / norm (z − x) 1 z x y]
      by (auto simp add: algebra-simps)
    then show ?thesis
      using rel-interior-closure-convex-shrink assms x ‹y ∈ closure S›
      by auto
  qed
 }
 ultimately show ?thesis by auto
qed

lemma convex-interior-closure:
 fixes S :: 'n::euclidean-space set
 assumes convex S
 shows interior (closure S) = interior S
 using closure-aff-dim[of S] interior-rel-interior-gen[of S]
  interior-rel-interior-gen[of closure S]
  convex-rel-interior-closure[of S] assms
 by auto

lemma closure-eq-rel-interior-eq:
 fixes S1 S2 :: 'n::euclidean-space set
 assumes convex S1
  and convex S2
 shows closure S1 = closure S2 ⟷ rel-interior S1 = rel-interior S2
 by (metis convex-rel-interior-closure convex-closure-rel-interior assms)

lemma closure-eq-between:
 fixes S1 S2 :: 'n::euclidean-space set
 assumes convex S1
  and convex S2
 shows closure S1 = closure S2 ⟷ rel-interior S1 ≤ S2 ∧ S2 ⊆ closure S1
 (is ?A ⟷ ?B)
proof
```

assume *?A*
then show *?B*
by (*metis assms closure-subset convex-rel-interior-closure rel-interior-subset*)
next
assume *?B*
then have *closure S1 ⊆ closure S2*
by (*metis assms*(*1*) *convex-closure-rel-interior closure-mono*)
moreover from ‹*?B*› **have** *closure S1 ⊇ closure S2*
by (*metis closed-closure closure-minimal*)
ultimately show *?A* **..**
qed

lemma *open-inter-closure-rel-interior*:
fixes *S A* :: *'n*::*euclidean-space set*
assumes *convex S*
and *open A*
shows *A ∩ closure S = {} ⟷ A ∩ rel-interior S = {}*
by (*metis assms convex-closure-rel-interior open-inter-closure-eq-empty*)

definition *rel-frontier S = closure S − rel-interior S*

lemma *closed-affine-hull*:
fixes *S* :: *'n*::*euclidean-space set*
shows *closed* (*affine hull S*)
by (*metis affine-affine-hull affine-closed*)

lemma *closed-rel-frontier*:
fixes *S* :: *'n*::*euclidean-space set*
shows *closed* (*rel-frontier S*)
proof −
have ∗: *closedin* (*subtopology euclidean* (*affine hull S*)) (*closure S − rel-interior S*)
apply (*rule closedin-diff*[*of subtopology euclidean* (*affine hull S*)*closure S rel-interior S*])
using *closed-closedin-trans*[*of affine hull S closure S*] *closed-affine-hull*[*of S*] *closure-affine-hull*[*of S*] *opein-rel-interior*[*of S*]
apply *auto*
done
show *?thesis*
apply (*rule closedin-closed-trans*[*of affine hull S rel-frontier S*])
unfolding *rel-frontier-def*
using ∗ *closed-affine-hull*
apply *auto*
done
qed

lemma *convex-rel-frontier-aff-dim*:
fixes *S1 S2* :: *'n*::*euclidean-space set*

```
  assumes convex S1
    and convex S2
    and S2 ≠ {}
    and S1 ≤ rel-frontier S2
  shows aff-dim S1 < aff-dim S2
proof −
  have S1 ⊆ closure S2
    using assms unfolding rel-frontier-def by auto
  then have ∗: affine hull S1 ⊆ affine hull S2
    using hull-mono[of S1 closure S2] closure-same-affine-hull[of S2]
    by auto
  then have aff-dim S1 ≤ aff-dim S2
    using ∗ aff-dim-affine-hull[of S1] aff-dim-affine-hull[of S2]
      aff-dim-subset[of affine hull S1 affine hull S2]
    by auto
  moreover
  {
    assume eq: aff-dim S1 = aff-dim S2
    then have S1 ≠ {}
      using aff-dim-empty[of S1] aff-dim-empty[of S2] ‹S2 ≠ {}› by auto
    have ∗∗: affine hull S1 = affine hull S2
      apply (rule affine-dim-equal)
      using ∗ affine-affine-hull
      apply auto
      using ‹S1 ≠ {}› hull-subset[of S1]
      apply auto
      using eq aff-dim-affine-hull[of S1] aff-dim-affine-hull[of S2]
      apply auto
      done
    obtain a where a: a ∈ rel-interior S1
      using ‹S1 ≠ {}› rel-interior-eq-empty assms by auto
    obtain T where T: open T a ∈ T ∩ S1 T ∩ affine hull S1 ⊆ S1
      using mem-rel-interior[of a S1] a by auto
    then have a ∈ T ∩ closure S2
      using a assms unfolding rel-frontier-def by auto
    then obtain b where b: b ∈ T ∩ rel-interior S2
      using open-inter-closure-rel-interior[of S2 T] assms T by auto
    then have b ∈ affine hull S1
      using rel-interior-subset hull-subset[of S2] ∗∗ by auto
    then have b ∈ S1
      using T b by auto
    then have False
      using b assms unfolding rel-frontier-def by auto
  }
  ultimately show ?thesis
    using less-le by auto
qed
```

lemma *convex-rel-interior-if*:
fixes S :: *'n::euclidean-space set*
assumes *convex S*
and $z \in$ *rel-interior S*
shows $\forall x \in$*affine hull S*. $\exists m.\ m > 1 \wedge (\forall e.\ e > 1 \wedge e \leq m \longrightarrow (1 - e) *_R x + e *_R z \in S)$
proof −
obtain *e1* **where** *e1*: $e1 > 0 \wedge$ *cball z e1* $\cap$ *affine hull S* $\subseteq S$
using *mem-rel-interior-cball*[*of z S*] *assms* **by** *auto*
{
fix x
assume x: $x \in$ *affine hull S*
{
assume $x \neq z$
def $m \equiv 1 + e1/norm(x-z)$
hence $m > 1$ **using** *e1* ‹$x \neq z$› **by** *auto*
{
fix e
assume e: $e > 1 \wedge e \leq m$
have $z \in$ *affine hull S*
using *assms rel-interior-subset hull-subset*[*of S*] **by** *auto*
then have $*$: $(1 - e)*_R\ x + e *_R z \in$ *affine hull S*
using *mem-affine*[*of affine hull S x z* $(1-e)$ *e*] *affine-affine-hull*[*of S*] *x*
by *auto*
have $norm\ (z + e *_R x - (x + e *_R z)) = norm\ ((e - 1) *_R (x - z))$
by (*simp add*: *algebra-simps*)
also have $\ldots = (e - 1) * norm\ (x-z)$
using *norm-scaleR e* **by** *auto*
also have $\ldots \leq (m - 1) * norm\ (x - z)$
using *e mult-right-mono*[*of - - norm*$(x-z)$] **by** *auto*
also have $\ldots = (e1\ /\ norm\ (x - z)) * norm\ (x - z)$
using *m-def* **by** *auto*
also have $\ldots = e1$
using ‹$x \neq z$› *e1* **by** *simp*
finally have $**$: $norm\ (z + e *_R x - (x + e *_R z)) \leq e1$
by *auto*
have $(1 - e)*_R\ x + e *_R z \in$ *cball z e1*
using *m-def* $**$
unfolding *cball-def dist-norm*
by (*auto simp add*: *algebra-simps*)
then have $(1 - e) *_R x + e *_R z \in S$
using $e * e1$ **by** *auto*
}
then have $\exists m.\ m > 1 \wedge (\forall e.\ e > 1 \wedge e \leq m \longrightarrow (1 - e) *_R x + e *_R z \in S\ )$
using ‹$m > 1$ › **by** *auto*
}
moreover
{

```
    assume x = z
    def m ≡ 1 + e1
    then have m > 1
      using e1 by auto
    {
      fix e
      assume e: e > 1 ∧ e ≤ m
      then have (1 − e) *R x + e *R z ∈ S
        using e1 x ‹x = z› by (auto simp add: algebra-simps)
      then have (1 − e) *R x + e *R z ∈ S
        using e by auto
    }
    then have ∃ m. m > 1 ∧ (∀ e. e > 1 ∧ e ≤ m ⟶ (1 − e) *R x + e *R z
∈ S)
      using ‹m > 1› by auto
   }
   ultimately have ∃ m. m > 1 ∧ (∀ e. e > 1 ∧ e ≤ m ⟶ (1 − e) *R x + e
*R z ∈ S )
    by auto
  }
  then show ?thesis by auto
qed

lemma convex-rel-interior-if2:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  assumes z ∈ rel-interior S
  shows ∀ x∈affine hull S. ∃ e. e > 1 ∧ (1 − e)*R x + e *R z ∈ S
  using convex-rel-interior-if[of S z] assms by auto

lemma convex-rel-interior-only-if:
  fixes S :: 'n::euclidean-space set
  assumes convex S
    and S ≠ {}
  assumes ∀ x∈S. ∃ e. e > 1 ∧ (1 − e) *R x + e *R z ∈ S
  shows z ∈ rel-interior S
proof −
  obtain x where x: x ∈ rel-interior S
    using rel-interior-eq-empty assms by auto
  then have x ∈ S
    using rel-interior-subset by auto
  then obtain e where e: e > 1 ∧ (1 − e) *R x + e *R z ∈ S
    using assms by auto
  def y ≡ (1 − e) *R x + e *R z
  then have y ∈ S using e by auto
  def e1 ≡ 1/e
  then have 0 < e1 ∧ e1 < 1 using e by auto
  then have z =y − (1 − e1) *R (y − x)
    using e1-def y-def by (auto simp add: algebra-simps)
```

```
  then show ?thesis
    using rel-interior-convex-shrink[of S x y 1−e1] ‹0 < e1 ∧ e1 < 1› ‹y ∈ S› x
assms
    by auto
qed

lemma convex-rel-interior-iff:
  fixes S :: 'n::euclidean-space set
  assumes convex S
    and S ≠ {}
  shows z ∈ rel-interior S ⟷ (∀ x∈S. ∃ e. e > 1 ∧ (1 − e) *R x + e *R z ∈ S)
  using assms hull-subset[of S affine]
    convex-rel-interior-if[of S z] convex-rel-interior-only-if[of S z]
  by auto

lemma convex-rel-interior-iff2:
  fixes S :: 'n::euclidean-space set
  assumes convex S
    and S ≠ {}
  shows z ∈ rel-interior S ⟷ (∀ x∈affine hull S. ∃ e. e > 1 ∧ (1 − e) *R x +
e *R z ∈ S)
 using assms hull-subset[of S] convex-rel-interior-if2[of S z] convex-rel-interior-only-if[of
S z]
  by auto

lemma convex-interior-iff:
  fixes S :: 'n::euclidean-space set
  assumes convex S
  shows z ∈ interior S ⟷ (∀ x. ∃ e. e > 0 ∧ z + e *R x ∈ S)
proof (cases aff-dim S = int DIM('n))
  case False
  {
    assume z ∈ interior S
    then have False
      using False interior-rel-interior-gen[of S] by auto
  }
  moreover
  {
    assume r: ∀ x. ∃ e. e > 0 ∧ z + e *R x ∈ S
    {
      fix x
      obtain e1 where e1: e1 > 0 ∧ z + e1 *R (x − z) ∈ S
        using r by auto
      obtain e2 where e2: e2 > 0 ∧ z + e2 *R (z − x) ∈ S
        using r by auto
      def x1 ≡ z + e1 *R (x − z)
      then have x1: x1 ∈ affine hull S
        using e1 hull-subset[of S] by auto
      def x2 ≡ z + e2 *R (z − x)
```

```
    then have x2: x2 ∈ affine hull S
      using e2 hull-subset[of S] by auto
    have ∗: e1/(e1+e2) + e2/(e1+e2) = 1
      using add-divide-distrib[of e1 e2 e1+e2] e1 e2 by simp
    then have z = (e2/(e1+e2)) ∗R x1 + (e1/(e1+e2)) ∗R x2
      using x1-def x2-def
      apply (auto simp add: algebra-simps)
      using scaleR-left-distrib[of e1/(e1+e2) e2/(e1+e2) z]
      apply auto
      done
    then have z: z ∈ affine hull S
      using mem-affine[of affine hull S x1 x2 e2/(e1+e2) e1/(e1+e2)]
        x1 x2 affine-affine-hull[of S] ∗
      by auto
    have x1 − x2 = (e1 + e2) ∗R (x − z)
      using x1-def x2-def by (auto simp add: algebra-simps)
    then have x = z+(1/(e1+e2)) ∗R (x1−x2)
      using e1 e2 by simp
    then have x ∈ affine hull S
      using mem-affine-3-minus[of affine hull S z x1 x2 1/(e1+e2)]
        x1 x2 z affine-affine-hull[of S]
      by auto
  }
  then have affine hull S = UNIV
    by auto
  then have aff-dim S = int DIM('n)
    using aff-dim-affine-hull[of S] by (simp add: aff-dim-univ)
  then have False
    using False by auto
 }
 ultimately show ?thesis by auto
next
 case True
 then have S ≠ {}
  using aff-dim-empty[of S] by auto
 have ∗: affine hull S = UNIV
  using True affine-hull-univ by auto
 {
  assume z ∈ interior S
  then have z ∈ rel-interior S
    using True interior-rel-interior-gen[of S] by auto
  then have ∗∗: ∀ x. ∃ e. e > 1 ∧ (1 − e) ∗R x + e ∗R z ∈ S
    using convex-rel-interior-iff2[of S z] assms ‹S ≠ {}› ∗ by auto
  fix x
  obtain e1 where e1: e1 > 1 (1 − e1) ∗R (z − x) + e1 ∗R z ∈ S
    using ∗∗[rule-format, of z−x] by auto
  def e ≡ e1 − 1
  then have (1 − e1) ∗R (z − x) + e1 ∗R z = z + e ∗R x
    by (simp add: algebra-simps)
```

```
    then have e > 0 z + e *R x ∈ S
      using e1 e-def by auto
    then have ∃ e. e > 0 ∧ z + e *R x ∈ S
      by auto
  }
  moreover
  {
    assume r: ∀ x. ∃ e. e > 0 ∧ z + e *R x ∈ S
    {
      fix x
      obtain e1 where e1: e1 > 0 z + e1 *R (z − x) ∈ S
        using r[rule-format, of z−x] by auto
      def e ≡ e1 + 1
      then have z + e1 *R (z − x) = (1 − e) *R x + e *R z
        by (simp add: algebra-simps)
      then have e > 1 (1 − e)*R x + e *R z ∈ S
        using e1 e-def by auto
      then have ∃ e. e > 1 ∧ (1 − e) *R x + e *R z ∈ S by auto
    }
    then have z ∈ rel-interior S
      using convex-rel-interior-iff2[of S z] assms ‹S ≠ {}› by auto
    then have z ∈ interior S
      using True interior-rel-interior-gen[of S] by auto
  }
  ultimately show ?thesis by auto
qed
```

19.32.1 Relative interior and closure under common operations

```
lemma rel-interior-inter-aux: ⋂{rel-interior S |S. S : I} ⊆ ⋂I
proof −
  {
    fix y
    assume y ∈ ⋂{rel-interior S |S. S : I}
    then have y: ∀ S ∈ I. y ∈ rel-interior S
      by auto
    {
      fix S
      assume S ∈ I
      then have y ∈ S
        using rel-interior-subset y by auto
    }
    then have y ∈ ⋂I by auto
  }
  then show ?thesis by auto
qed

lemma closure-inter: closure (⋂I) ≤ ⋂{closure S |S. S ∈ I}
proof −
```

```
  {
    fix y
    assume y ∈ ⋂ I
    then have y: ∀ S ∈ I. y ∈ S by auto
    {
      fix S
      assume S ∈ I
      then have y ∈ closure S
        using closure-subset y by auto
    }
    then have y ∈ ⋂ {closure S |S. S ∈ I}
      by auto
  }
  then have ⋂ I ⊆ ⋂ {closure S |S. S ∈ I}
    by auto
  moreover have closed (⋂ {closure S |S. S ∈ I})
    unfolding closed-Inter closed-closure by auto
  ultimately show ?thesis using closure-hull[of ⋂ I]
    hull-minimal[of ⋂ I ⋂ {closure S |S. S ∈ I} closed] by auto
qed

lemma convex-closure-rel-interior-inter:
  assumes ∀ S∈I. convex (S :: 'n::euclidean-space set)
    and ⋂ {rel-interior S |S. S ∈ I} ≠ {}
  shows ⋂ {closure S |S. S ∈ I} ≤ closure (⋂ {rel-interior S |S. S ∈ I})
proof −
  obtain x where x: ∀ S∈I. x ∈ rel-interior S
    using assms by auto
  {
    fix y
    assume y ∈ ⋂ {closure S |S. S ∈ I}
    then have y: ∀ S ∈ I. y ∈ closure S
      by auto
    {
      assume y = x
      then have y ∈ closure (⋂ {rel-interior S |S. S ∈ I})
        using x closure-subset[of ⋂ {rel-interior S |S. S ∈ I}] by auto
    }
    moreover
    {
      assume y ≠ x
      { fix e :: real
        assume e: e > 0
        def e1 ≡ min 1 (e/norm (y − x))
        then have e1: e1 > 0 e1 ≤ 1 e1 * norm (y − x) ≤ e
          using ‹y ≠ x› ‹e > 0› le-divide-eq[of e1 e norm (y − x)]
          by simp-all
        def z ≡ y − e1 *R (y − x)
        {
```

```
      fix S
      assume S ∈ I
      then have z ∈ rel-interior S
        using rel-interior-closure-convex-shrink[of S x y e1] assms x y e1 z-def
        by auto
    }
    then have ∗: z ∈ ⋂{rel-interior S |S. S ∈ I}
      by auto
    have ∃z. z ∈ ⋂{rel-interior S |S. S ∈ I} ∧ z ≠ y ∧ dist z y ≤ e
      apply (rule-tac x=z in exI)
      using ‹y ≠ x› z-def ∗ e1 e dist-norm[of z y]
      apply simp
      done
  }
  then have y islimpt ⋂{rel-interior S |S. S ∈ I}
    unfolding islimpt-approachable-le by blast
  then have y ∈ closure (⋂{rel-interior S |S. S ∈ I})
    unfolding closure-def by auto
 }
 ultimately have y ∈ closure (⋂{rel-interior S |S. S ∈ I})
  by auto
}
then show ?thesis by auto
qed


lemma convex-closure-inter:
 assumes ∀S∈I. convex (S :: 'n::euclidean-space set)
  and ⋂{rel-interior S |S. S ∈ I} ≠ {}
 shows closure (⋂I) = ⋂{closure S |S. S ∈ I}
proof −
 have ⋂{closure S |S. S ∈ I} ≤ closure (⋂{rel-interior S |S. S ∈ I})
  using convex-closure-rel-interior-inter assms by auto
 moreover
 have closure (⋂{rel-interior S |S. S ∈ I}) ≤ closure (⋂I)
  using rel-interior-inter-aux closure-mono[of ⋂{rel-interior S |S. S ∈ I} ⋂I]
  by auto
 ultimately show ?thesis
  using closure-inter[of I] by auto
qed

lemma convex-inter-rel-interior-same-closure:
 assumes ∀S∈I. convex (S :: 'n::euclidean-space set)
  and ⋂{rel-interior S |S. S ∈ I} ≠ {}
 shows closure (⋂{rel-interior S |S. S ∈ I}) = closure (⋂I)
proof −
 have ⋂{closure S |S. S ∈ I} ≤ closure (⋂{rel-interior S |S. S ∈ I})
  using convex-closure-rel-interior-inter assms by auto
 moreover
```

```
  have closure (⋂{rel-interior S |S. S ∈ I}) ≤ closure (⋂ I)
    using rel-interior-inter-aux closure-mono[of ⋂{rel-interior S |S. S ∈ I} ⋂ I]
    by auto
  ultimately show ?thesis
    using closure-inter[of I] by auto
qed

lemma convex-rel-interior-inter:
  assumes ∀ S∈I. convex (S :: 'n::euclidean-space set)
    and ⋂{rel-interior S |S. S ∈ I} ≠ {}
  shows rel-interior (⋂ I) ⊆ ⋂{rel-interior S |S. S ∈ I}
proof −
  have convex (⋂ I)
    using assms convex-Inter by auto
  moreover
  have convex (⋂{rel-interior S |S. S ∈ I})
    apply (rule convex-Inter)
    using assms convex-rel-interior
    apply auto
    done
  ultimately
  have rel-interior (⋂{rel-interior S |S. S ∈ I}) = rel-interior (⋂ I)
    using convex-inter-rel-interior-same-closure assms
      closure-eq-rel-interior-eq[of ⋂{rel-interior S |S. S ∈ I} ⋂ I]
    by blast
  then show ?thesis
    using rel-interior-subset[of ⋂{rel-interior S |S. S ∈ I}] by auto
qed

lemma convex-rel-interior-finite-inter:
  assumes ∀ S∈I. convex (S :: 'n::euclidean-space set)
    and ⋂{rel-interior S |S. S ∈ I} ≠ {}
    and finite I
  shows rel-interior (⋂ I) = ⋂{rel-interior S |S. S ∈ I}
proof −
  have ⋂ I ≠ {}
    using assms rel-interior-inter-aux[of I] by auto
  have convex (⋂ I)
    using convex-Inter assms by auto
  show ?thesis
  proof (cases I = {})
    case True
    then show ?thesis
      using Inter-empty rel-interior-univ2 by auto
  next
    case False
    {
      fix z
      assume z: z ∈ ⋂{rel-interior S |S. S ∈ I}
```

```
    {
      fix x
      assume x: x ∈ ⋂I
      {
        fix S
        assume S: S ∈ I
        then have z ∈ rel-interior S x ∈ S
          using z x by auto
        then have ∃ m. m > 1 ∧ (∀ e. e > 1 ∧ e ≤ m ⟶ (1 - e)*R x + e *R
z ∈ S)
          using convex-rel-interior-if[of S z] S assms hull-subset[of S] by auto
      }
      then obtain mS where
        mS: ∀ S∈I. mS S > 1 ∧ (∀ e. e > 1 ∧ e ≤ mS S ⟶ (1 - e) *R x + e
*R z ∈ S) by metis
      def e ≡ Min (mS ` I)
      then have e ∈ mS ` I using assms ‹I ≠ {}› by simp
      then have e > 1 using mS by auto
      moreover have ∀ S∈I. e ≤ mS S
        using e-def assms by auto
      ultimately have ∃ e > 1. (1 - e) *R x + e *R z ∈ ⋂I
        using mS by auto
    }
    then have z ∈ rel-interior (⋂I)
      using convex-rel-interior-iff[of ⋂I z] ‹⋂I ≠ {}› ‹convex (⋂I)› by auto
  }
  then show ?thesis
    using convex-rel-interior-inter[of I] assms by auto
 qed
qed

lemma convex-closure-inter-two:
 fixes S T :: 'n::euclidean-space set
 assumes convex S
   and convex T
 assumes rel-interior S ∩ rel-interior T ≠ {}
 shows closure (S ∩ T) = closure S ∩ closure T
 using convex-closure-inter[of {S,T}] assms by auto

lemma convex-rel-interior-inter-two:
 fixes S T :: 'n::euclidean-space set
 assumes convex S
   and convex T
   and rel-interior S ∩ rel-interior T ≠ {}
 shows rel-interior (S ∩ T) = rel-interior S ∩ rel-interior T
 using convex-rel-interior-finite-inter[of {S,T}] assms by auto

lemma convex-affine-closure-inter:
 fixes S T :: 'n::euclidean-space set
```

assumes *convex S*
and *affine T*
and *rel-interior S ∩ T ≠ {}*
shows *closure (S ∩ T) = closure S ∩ T*
proof −
have *affine hull T = T*
using *assms* **by** *auto*
then have *rel-interior T = T*
using *rel-interior-univ*[*of T*] **by** *metis*
moreover have *closure T = T*
using *assms affine-closed*[*of T*] **by** *auto*
ultimately show *?thesis*
using *convex-closure-inter-two*[*of S T*] *assms affine-imp-convex* **by** *auto*
qed

lemma *convex-affine-rel-interior-inter*:
fixes *S T* :: *'n*::*euclidean-space set*
assumes *convex S*
and *affine T*
and *rel-interior S ∩ T ≠ {}*
shows *rel-interior (S ∩ T) = rel-interior S ∩ T*
proof −
have *affine hull T = T*
using *assms* **by** *auto*
then have *rel-interior T = T*
using *rel-interior-univ*[*of T*] **by** *metis*
moreover have *closure T = T*
using *assms affine-closed*[*of T*] **by** *auto*
ultimately show *?thesis*
using *convex-rel-interior-inter-two*[*of S T*] *assms affine-imp-convex* **by** *auto*
qed

lemma *subset-rel-interior-convex*:
fixes *S T* :: *'n*::*euclidean-space set*
assumes *convex S*
and *convex T*
and *S ≤ closure T*
and *¬ S ⊆ rel-frontier T*
shows *rel-interior S ⊆ rel-interior T*
proof −
have ∗: *S ∩ closure T = S*
using *assms* **by** *auto*
have *¬ rel-interior S ⊆ rel-frontier T*
using *closure-mono*[*of rel-interior S rel-frontier T*] *closed-rel-frontier*[*of T*]
closure-closed[*of S*] *convex-closure-rel-interior*[*of S*] *closure-subset*[*of S*] *assms*
by *auto*
then have *rel-interior S ∩ rel-interior (closure T) ≠ {}*
using *assms rel-frontier-def*[*of T*] *rel-interior-subset convex-rel-interior-closure*[*of T*]

```
    by auto
  then have rel-interior S ∩ rel-interior T = rel-interior (S ∩ closure T)
    using assms convex-closure convex-rel-interior-inter-two[of S closure T]
      convex-rel-interior-closure[of T]
    by auto
  also have ... = rel-interior S
    using ∗ by auto
  finally show ?thesis
    by auto
qed

lemma rel-interior-convex-linear-image:
  fixes f :: 'm::euclidean-space ⇒ 'n::euclidean-space
  assumes linear f
    and convex S
  shows f ‘ (rel-interior S) = rel-interior (f ‘ S)
proof (cases S = {})
  case True
  then show ?thesis
    using assms rel-interior-empty rel-interior-eq-empty by auto
next
  case False
  have ∗: f ‘ (rel-interior S) ⊆ f ‘ S
    unfolding image-mono using rel-interior-subset by auto
  have f ‘ S ⊆ f ‘ (closure S)
    unfolding image-mono using closure-subset by auto
  also have ... = f ‘ (closure (rel-interior S))
    using convex-closure-rel-interior assms by auto
  also have ... ⊆ closure (f ‘ (rel-interior S))
    using closure-linear-image-subset assms by auto
  finally have closure (f ‘ S) = closure (f ‘ rel-interior S)
    using closure-mono[of f ‘ S closure (f ‘ rel-interior S)] closure-closure
      closure-mono[of f ‘ rel-interior S f ‘ S] ∗
    by auto
  then have rel-interior (f ‘ S) = rel-interior (f ‘ rel-interior S)
    using assms convex-rel-interior
      linear-conv-bounded-linear[of f] convex-linear-image[of - S]
      convex-linear-image[of - rel-interior S]
      closure-eq-rel-interior-eq[of f ‘ S f ‘ rel-interior S]
    by auto
  then have rel-interior (f ‘ S) ⊆ f ‘ rel-interior S
    using rel-interior-subset by auto
  moreover
  {
    fix z
    assume z ∈ f ‘ rel-interior S
    then obtain z1 where z1: z1 ∈ rel-interior S f z1 = z by auto
    {
      fix x
```

```
    assume x ∈ f ‘ S
    then obtain x1 where x1: x1 ∈ S f x1 = x by auto
    then obtain e where e: e > 1 (1 − e) *R x1 + e *R z1 : S
      using convex-rel-interior-iff[of S z1] ‹convex S› x1 z1 by auto
    moreover have f ((1 − e) *R x1 + e *R z1) = (1 − e) *R x + e *R z
      using x1 z1 ‹linear f› by (simp add: linear-add-cmul)
    ultimately have (1 − e) *R x + e *R z : f ‘ S
      using imageI[of (1 − e) *R x1 + e *R z1 S f] by auto
    then have ∃ e. e > 1 ∧ (1 − e) *R x + e *R z : f ‘ S
      using e by auto
  }
  then have z ∈ rel-interior (f ‘ S)
    using convex-rel-interior-iff[of f ‘ S z] ‹convex S›
     ‹linear f› ‹S ≠ {}› convex-linear-image[of f S]  linear-conv-bounded-linear[of
f]
    by auto
 }
 ultimately show ?thesis by auto
qed

lemma rel-interior-convex-linear-preimage:
 fixes f :: 'm::euclidean-space ⇒ 'n::euclidean-space
 assumes linear f
   and convex S
   and f −‘ (rel-interior S) ≠ {}
 shows rel-interior (f −‘ S) = f −‘ (rel-interior S)
proof −
 have S ≠ {}
   using assms rel-interior-empty by auto
 have nonemp: f −‘ S ≠ {}
   by (metis assms(3) rel-interior-subset subset-empty vimage-mono)
 then have S ∩ (range f) ≠ {}
   by auto
 have conv: convex (f −‘ S)
   using convex-linear-vimage assms by auto
 then have convex (S ∩ range f)
   by (metis assms(1) assms(2) convex-Int subspace-UNIV subspace-imp-convex
subspace-linear-image)
 {
   fix z
   assume z ∈ f −‘ (rel-interior S)
   then have z: f z : rel-interior S
     by auto
   {
     fix x
     assume x ∈ f −‘ S
     then have f x ∈ S by auto
     then obtain e where e: e > 1 (1 − e) *R f x + e *R f z ∈ S
       using convex-rel-interior-iff[of S f z] z assms ‹S ≠ {}› by auto
```

```
    moreover have (1 - e) *R f x + e *R f z = f ((1 - e) *R x + e *R z)
      using ‹linear f› by (simp add: linear-iff)
    ultimately have ∃e. e > 1 ∧ (1 - e) *R x + e *R z ∈ f -‘ S
      using e by auto
  }
  then have z ∈ rel-interior (f -‘ S)
    using convex-rel-interior-iff[of f -‘ S z] conv nonemp by auto
}
moreover
{
  fix z
  assume z: z ∈ rel-interior (f -‘ S)
  {
    fix x
    assume x ∈ S ∩ range f
    then obtain y where y: f y = x y ∈ f -‘ S by auto
    then obtain e where e: e > 1 (1 - e) *R y + e *R z ∈ f -‘ S
      using convex-rel-interior-iff[of f -‘ S z] z conv by auto
    moreover have (1 - e) *R x + e *R f z = f ((1 - e) *R y + e *R z)
      using ‹linear f› y by (simp add: linear-iff)
    ultimately have ∃e. e > 1 ∧ (1 - e) *R x + e *R f z ∈ S ∩ range f
      using e by auto
  }
  then have f z ∈ rel-interior (S ∩ range f)
    using ‹convex (S ∩ (range f))› ‹S ∩ range f ≠ {}›
      convex-rel-interior-iff[of S ∩ (range f) f z]
    by auto
  moreover have affine (range f)
   by (metis assms(1) subspace-UNIV subspace-imp-affine subspace-linear-image)
  ultimately have f z ∈ rel-interior S
    using convex-affine-rel-interior-inter[of S range f] assms by auto
  then have z ∈ f -‘ (rel-interior S)
    by auto
}
ultimately show ?thesis by auto
qed

lemma rel-interior-direct-sum:
  fixes S :: 'n::euclidean-space set
    and T :: 'm::euclidean-space set
  assumes convex S
    and convex T
  shows rel-interior (S × T) = rel-interior S × rel-interior T
proof -
  { assume S = {}
    then have ?thesis
      by auto
  }
  moreover
```

```
  { assume T = {}
    then have ?thesis
      by auto
  }
  moreover
  {
    assume S ≠ {} T ≠ {}
    then have ri: rel-interior S ≠ {} rel-interior T ≠ {}
      using rel-interior-eq-empty assms by auto
    then have fst -` rel-interior S ≠ {}
      using fst-vimage-eq-Times[of rel-interior S] by auto
    then have rel-interior ((fst :: 'n * 'm ⇒ 'n) -` S) = fst -` rel-interior S
       using fst-linear ‹convex S› rel-interior-convex-linear-preimage[of fst S] by
auto
    then have s: rel-interior (S × (UNIV :: 'm set)) = rel-interior S × UNIV
      by (simp add: fst-vimage-eq-Times)
    from ri have snd -` rel-interior T ≠ {}
      using snd-vimage-eq-Times[of rel-interior T] by auto
    then have rel-interior ((snd :: 'n * 'm ⇒ 'm) -` T) = snd -` rel-interior T
      using snd-linear ‹convex T› rel-interior-convex-linear-preimage[of snd T] by
auto
    then have t: rel-interior ((UNIV :: 'n set) × T) = UNIV × rel-interior T
      by (simp add: snd-vimage-eq-Times)
    from s t have *: rel-interior (S × (UNIV :: 'm set)) ∩ rel-interior ((UNIV ::
'n set) × T) =
      rel-interior S × rel-interior T by auto
    have S × T = S × (UNIV :: 'm set) ∩ (UNIV :: 'n set) × T
      by auto
     then have rel-interior (S × T) = rel-interior ((S × (UNIV :: 'm set)) ∩
((UNIV :: 'n set) × T))
      by auto
    also have ... = rel-interior (S × (UNIV :: 'm set)) ∩ rel-interior ((UNIV ::
'n set) × T)
       apply (subst convex-rel-interior-inter-two[of S × (UNIV :: 'm set) (UNIV
:: 'n set) × T])
       using * ri assms convex-Times
       apply auto
       done
    finally have ?thesis using * by auto
  }
  ultimately show ?thesis by blast
qed

lemma rel-interior-scaleR:
  fixes S :: 'n::euclidean-space set
  assumes c ≠ 0
  shows (op *R c) ` (rel-interior S) = rel-interior ((op *R c) ` S)
  using rel-interior-injective-linear-image[of (op *R c) S]
    linear-conv-bounded-linear[of op *R c] linear-scaleR injective-scaleR[of c] assms
```

by *auto*

lemma *rel-interior-convex-scaleR*:
fixes S :: *'n::euclidean-space set*
assumes *convex* S
shows $(op\ *_R\ c)$ ` $(rel\text{-}interior\ S) = rel\text{-}interior\ ((op\ *_R\ c)$ ` $S)$
by (*metis assms linear-scaleR rel-interior-convex-linear-image*)

lemma *convex-rel-open-scaleR*:
fixes S :: *'n::euclidean-space set*
assumes *convex* S
and *rel-open* S
shows *convex* $((op\ *_R\ c)$ ` $S) \wedge$ *rel-open* $((op\ *_R\ c)$ ` $S)$
by (*metis assms convex-scaling rel-interior-convex-scaleR rel-open-def*)

lemma *convex-rel-open-finite-inter*:
assumes $\forall S{\in}I.$ *convex* $(S$ :: *'n::euclidean-space set*$) \wedge$ *rel-open* S
and *finite* I
shows *convex* $(\bigcap I) \wedge$ *rel-open* $(\bigcap I)$
proof (*cases* $\bigcap \{rel\text{-}interior\ S\ |S.\ S \in I\} = \{\}$)
case *True*
then have $\bigcap I = \{\}$
using *assms* **unfolding** *rel-open-def* **by** *auto*
then show *?thesis*
unfolding *rel-open-def* **using** *rel-interior-empty* **by** *auto*
next
case *False*
then have *rel-open* $(\bigcap I)$
using *assms* **unfolding** *rel-open-def*
using *convex-rel-interior-finite-inter*[*of* I]
by *auto*
then show *?thesis*
using *convex-Inter assms* **by** *auto*
qed

lemma *convex-rel-open-linear-image*:
fixes f :: *'m::euclidean-space* $\Rightarrow$ *'n::euclidean-space*
assumes *linear* f
and *convex* S
and *rel-open* S
shows *convex* $(f$ ` $S) \wedge$ *rel-open* $(f$ ` $S)$
by (*metis assms convex-linear-image rel-interior-convex-linear-image rel-open-def*)

lemma *convex-rel-open-linear-preimage*:
fixes f :: *'m::euclidean-space* $\Rightarrow$ *'n::euclidean-space*
assumes *linear* f
and *convex* S
and *rel-open* S
shows *convex* $(f -` S) \wedge$ *rel-open* $(f -` S)$

proof (*cases* $f -` (\mathit{rel\text{-}interior}\ S) = \{\}$)
case *True*
then have $f -` S = \{\}$
using *assms* **unfolding** *rel-open-def* **by** *auto*
then show *?thesis*
unfolding *rel-open-def* **using** *rel-interior-empty* **by** *auto*
next
case *False*
then have *rel-open* $(f -` S)$
using *assms* **unfolding** *rel-open-def*
using *rel-interior-convex-linear-preimage*[*of f S*]
by *auto*
then show *?thesis*
using *convex-linear-vimage assms*
by *auto*
qed

lemma *rel-interior-projection*:
fixes S :: ('*m*::*euclidean-space* × '*n*::*euclidean-space*) *set*
and f :: '*m*::*euclidean-space* ⇒ '*n*::*euclidean-space set*
assumes *convex S*
and $f = (\lambda y.\ \{z.\ (y, z) \in S\})$
shows $(y, z) \in \mathit{rel\text{-}interior}\ S \longleftrightarrow (y \in \mathit{rel\text{-}interior}\ \{y.\ (f\ y \neq \{\})\} \wedge z \in \mathit{rel\text{-}interior}\ (f\ y))$
proof −
{
fix y
assume $y \in \{y.\ f\ y \neq \{\}\}$
then obtain z **where** $(y, z) \in S$
using *assms* **by** *auto*
then have $\exists x.\ x \in S \wedge y = \mathit{fst}\ x$
apply (*rule-tac* $x=(y, z)$ **in** *exI*)
apply *auto*
done
then obtain x **where** $x \in S\ y = \mathit{fst}\ x$
by *blast*
then have $y \in \mathit{fst} ` S$
unfolding *image-def* **by** *auto*
}
then have $\mathit{fst} ` S = \{y.\ f\ y \neq \{\}\}$
unfolding *fst-def* **using** *assms* **by** *auto*
then have *h1*: $\mathit{fst} ` \mathit{rel\text{-}interior}\ S = \mathit{rel\text{-}interior}\ \{y.\ f\ y \neq \{\}\}$
using *rel-interior-convex-linear-image*[*of fst S*] *assms fst-linear* **by** *auto*
{
fix y
assume $y \in \mathit{rel\text{-}interior}\ \{y.\ f\ y \neq \{\}\}$
then have $y \in \mathit{fst} ` \mathit{rel\text{-}interior}\ S$
using *h1* **by** *auto*
then have $*$: $\mathit{rel\text{-}interior}\ S \cap \mathit{fst} -` \{y\} \neq \{\}$

```
    by auto
  moreover have aff: affine (fst -` {y})
    unfolding affine-alt by (simp add: algebra-simps)
  ultimately have **: rel-interior (S ∩ fst -` {y}) = rel-interior S ∩ fst -`
{y}
    using convex-affine-rel-interior-inter[of S fst -` {y}] assms by auto
  have conv: convex (S ∩ fst -` {y})
    using convex-Int assms aff affine-imp-convex by auto
  {
    fix x
    assume x ∈ f y
    then have (y, x) ∈ S ∩ (fst -` {y})
      using assms by auto
    moreover have x = snd (y, x) by auto
    ultimately have x ∈ snd ` (S ∩ fst -` {y})
      by blast
  }
  then have snd ` (S ∩ fst -` {y}) = f y
    using assms by auto
  then have ***: rel-interior (f y) = snd ` rel-interior (S ∩ fst -` {y})
    using rel-interior-convex-linear-image[of snd S ∩ fst -` {y}] snd-linear conv
    by auto
  {
    fix z
    assume z ∈ rel-interior (f y)
    then have z ∈ snd ` rel-interior (S ∩ fst -` {y})
      using *** by auto
    moreover have {y} = fst ` rel-interior (S ∩ fst -` {y})
      using * ** rel-interior-subset by auto
    ultimately have (y, z) ∈ rel-interior (S ∩ fst -` {y})
      by force
    then have (y,z) ∈ rel-interior S
      using ** by auto
  }
  moreover
  {
    fix z
    assume (y, z) ∈ rel-interior S
    then have (y, z) ∈ rel-interior (S ∩ fst -` {y})
      using ** by auto
    then have z ∈ snd ` rel-interior (S ∩ fst -` {y})
      by (metis Range-iff snd-eq-Range)
    then have z ∈ rel-interior (f y)
      using *** by auto
  }
  ultimately have ⋀z. (y, z) ∈ rel-interior S ⟷ z ∈ rel-interior (f y)
    by auto
}
then have h2: ⋀y z. y ∈ rel-interior {t. f t ≠ {}} ⟹
```

```
    (y, z) ∈ rel-interior S ⟷ z ∈ rel-interior (f y)
    by auto
  {
    fix y z
    assume asm: (y, z) ∈ rel-interior S
    then have y ∈ fst ` rel-interior S
      by (metis Domain-iff fst-eq-Domain)
    then have y ∈ rel-interior {t. f t ≠ {}}
      using h1 by auto
    then have y ∈ rel-interior {t. f t ≠ {}} and (z : rel-interior (f y))
      using h2 asm by auto
  }
  then show ?thesis using h2 by blast
qed
```

19.32.2 Relative interior of convex cone

```
lemma cone-rel-interior:
  fixes S :: 'm::euclidean-space set
  assumes cone S
  shows cone ({0} ∪ rel-interior S)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: rel-interior-empty cone-0)
next
  case False
  then have ∗: 0 ∈ S ∧ (∀ c. c > 0 ⟶ op ∗R c ` S = S)
    using cone-iff[of S] assms by auto
  then have ∗: 0 ∈ ({0} ∪ rel-interior S)
    and ∀ c. c > 0 ⟶ op ∗R c ` ({0} ∪ rel-interior S) = ({0} ∪ rel-interior S)
    by (auto simp add: rel-interior-scaleR)
  then show ?thesis
    using cone-iff[of {0} ∪ rel-interior S] by auto
qed

lemma rel-interior-convex-cone-aux:
  fixes S :: 'm::euclidean-space set
  assumes convex S
  shows (c, x) ∈ rel-interior (cone hull ({(1 :: real)} × S)) ⟷
    c > 0 ∧ x ∈ ((op ∗R c) ` (rel-interior S))
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: rel-interior-empty cone-hull-empty)
next
  case False
  then obtain s where s ∈ S by auto
  have conv: convex ({(1 :: real)} × S)
```

```
    using convex-Times[of {(1 :: real)} S] assms convex-singleton[of 1 :: real]
    by auto
  def f ≡ λy. {z. (y, z) ∈ cone hull ({1 :: real} × S)}
  then have ∗: (c, x) ∈ rel-interior (cone hull ({(1 :: real)} × S)) =
    (c ∈ rel-interior {y. f y ≠ {}} ∧ x ∈ rel-interior (f c))
    apply (subst rel-interior-projection[of cone hull ({(1 :: real)} × S) f c x])
    using convex-cone-hull[of {(1 :: real)} × S] conv
    apply auto
    done
  {
    fix y :: real
    assume y ≥ 0
    then have y *R (1,s) ∈ cone hull ({1 :: real} × S)
      using cone-hull-expl[of {(1 :: real)} × S] ‹s ∈ S› by auto
    then have f y ≠ {}
      using f-def by auto
  }
  then have {y. f y ≠ {}} = {0..}
    using f-def cone-hull-expl[of {1 :: real} × S] by auto
  then have ∗∗: rel-interior {y. f y ≠ {}} = {0<..}
    using rel-interior-real-semiline by auto
  {
    fix c :: real
    assume c > 0
    then have f c = (op *R c ' S)
      using f-def cone-hull-expl[of {1 :: real} × S] by auto
    then have rel-interior (f c) = op *R c ' rel-interior S
      using rel-interior-convex-scaleR[of S c] assms by auto
  }
  then show ?thesis using ∗ ∗∗ by auto
qed

lemma rel-interior-convex-cone:
  fixes S :: 'm::euclidean-space set
  assumes convex S
  shows rel-interior (cone hull ({1 :: real} × S)) =
    {(c, c *R x) | c x. c > 0 ∧ x ∈ rel-interior S}
  (is ?lhs = ?rhs)
proof −
  {
    fix z
    assume z ∈ ?lhs
    have ∗: z = (fst z, snd z)
      by auto
    have z ∈ ?rhs
      using rel-interior-convex-cone-aux[of S fst z snd z] assms ‹z ∈ ?lhs›
      apply auto
      apply (rule-tac x = fst z in exI)
      apply (rule-tac x = x in exI)
```

```
    using ∗
    apply auto
    done
  }
  moreover
  {
    fix z
    assume z ∈ ?rhs
    then have z ∈ ?lhs
      using rel-interior-convex-cone-aux[of S fst z snd z] assms
      by auto
  }
  ultimately show ?thesis by blast
qed

lemma convex-hull-finite-union:
  assumes finite I
  assumes ∀ i∈I. convex (S i) ∧ (S i) ≠ {}
  shows convex hull (⋃(S ‘ I)) =
    {setsum (λi. c i ∗R s i) I | c s. (∀ i∈I. c i ≥ 0) ∧ setsum c I = 1 ∧ (∀ i∈I. s
i ∈ S i)}
  (is ?lhs = ?rhs)
proof −
  have ?lhs ⊇ ?rhs
  proof
    fix x
    assume x : ?rhs
    then obtain c s where ∗: setsum (λi. c i ∗R s i) I = x setsum c I = 1
      (∀ i∈I. c i ≥ 0) ∧ (∀ i∈I. s i ∈ S i) by auto
    then have ∀ i∈I. s i ∈ convex hull (⋃(S ‘ I))
      using hull-subset[of ⋃(S ‘ I) convex] by auto
    then show x ∈ ?lhs
      unfolding ∗(1)[symmetric]
      apply (subst convex-setsum[of I convex hull ⋃(S ‘ I) c s])
      using ∗ assms convex-convex-hull
      apply auto
      done
  qed

  {
    fix i
    assume i ∈ I
    with assms have ∃ p. p ∈ S i by auto
  }
  then obtain p where p: ∀ i∈I. p i ∈ S i by metis

  {
    fix i
    assume i ∈ I
```

```
  {
    fix x
    assume x ∈ S i
    def c ≡ λj. if j = i then 1::real else 0
    then have ∗: setsum c I = 1
      using ⟨finite I⟩ ⟨i ∈ I⟩ setsum.delta[of I i λj::'a. 1::real]
      by auto
    def s ≡ λj. if j = i then x else p j
    then have ∀ j. c j ∗R s j = (if j = i then x else 0)
      using c-def by (auto simp add: algebra-simps)
    then have x = setsum (λi. c i ∗R s i) I
      using s-def c-def ⟨finite I⟩ ⟨i ∈ I⟩ setsum.delta[of I i λj::'a. x]
      by auto
    then have x ∈ ?rhs
      apply auto
      apply (rule-tac x = c in exI)
      apply (rule-tac x = s in exI)
      using ∗ c-def s-def p ⟨x ∈ S i⟩
      apply auto
      done
  }
  then have ?rhs ⊇ S i by auto
 }
 then have ∗: ?rhs ⊇ ⋃(S ' I) by auto

 {
  fix u v :: real
  assume uv: u ≥ 0 ∧ v ≥ 0 ∧ u + v = 1
  fix x y
  assume xy: x ∈ ?rhs ∧ y ∈ ?rhs
  from xy obtain c s where
    xc: x = setsum (λi. c i ∗R s i) I ∧ (∀ i∈I. c i ≥ 0) ∧ setsum c I = 1 ∧
(∀ i∈I. s i ∈ S i)
    by auto
  from xy obtain d t where
    yc: y = setsum (λi. d i ∗R t i) I ∧ (∀ i∈I. d i ≥ 0) ∧ setsum d I = 1 ∧
(∀ i∈I. t i ∈ S i)
    by auto
  def e ≡ λi. u ∗ c i + v ∗ d i
  have ge0: ∀ i∈I. e i ≥ 0
    using e-def xc yc uv by simp
  have setsum (λi. u ∗ c i) I = u ∗ setsum c I
    by (simp add: setsum-right-distrib)
  moreover have setsum (λi. v ∗ d i) I = v ∗ setsum d I
    by (simp add: setsum-right-distrib)
  ultimately have sum1: setsum e I = 1
    using e-def xc yc uv by (simp add: setsum.distrib)
  def q ≡ λi. if e i = 0 then p i else (u ∗ c i / e i) ∗R s i + (v ∗ d i / e i) ∗R t i
  {
```

```
    fix i
    assume i: i ∈ I
    have q i ∈ S i
    proof (cases e i = 0)
      case True
      then show ?thesis using i p q-def by auto
    next
      case False
      then show ?thesis
        using mem-convex-alt[of S i s i t i u * (c i) v * (d i)]
          mult-nonneg-nonneg[of u c i] mult-nonneg-nonneg[of v d i]
          assms q-def e-def i False xc yc uv
        by (auto simp del: mult-nonneg-nonneg)
    qed
  }
  then have qs: ∀ i∈I. q i ∈ S i by auto
  {
    fix i
    assume i: i ∈ I
    have (u * c i) *R s i + (v * d i) *R t i = e i *R q i
    proof (cases e i = 0)
      case True
      have ge: u * (c i) ≥ 0 ∧ v * d i ≥ 0
        using xc yc uv i by simp
      moreover from ge have u * c i ≤ 0 ∧ v * d i ≤ 0
        using True e-def i by simp
      ultimately have u * c i = 0 ∧ v * d i = 0 by auto
      with True show ?thesis by auto
    next
      case False
      then have (u * (c i)/(e i))*R (s i)+(v * (d i)/(e i))*R (t i) = q i
        using q-def by auto
      then have e i *R ((u * (c i)/(e i))*R (s i)+(v * (d i)/(e i))*R (t i))
            = (e i) *R (q i) by auto
      with False show ?thesis by (simp add: algebra-simps)
    qed
  }
  then have *: ∀ i∈I. (u * c i) *R s i + (v * d i) *R t i = e i *R q i
    by auto
  have u *R x + v *R y = setsum (λi. (u * c i) *R s i + (v * d i) *R t i) I
    using xc yc by (simp add: algebra-simps scaleR-right.setsum setsum.distrib)
  also have ... = setsum (λi. e i *R q i) I
    using * by auto
  finally have u *R x + v *R y = setsum (λi. (e i) *R (q i)) I
    by auto
  then have u *R x + v *R y ∈ ?rhs
    using ge0 sum1 qs by auto
}
then have convex ?rhs unfolding convex-def by auto
```

then show *?thesis*
using ‹*?lhs* ⊇ *?rhs*› ∗ *hull-minimal*[*of* ⋃(*S* ‘ *I*) *?rhs convex*]
by *blast*
qed

lemma *convex-hull-union-two*:
fixes *S T* :: *'m*::*euclidean-space set*
assumes *convex S*
and *S* ≠ {}
and *convex T*
and *T* ≠ {}
shows *convex hull* (*S* ∪ *T*) =
{*u* ∗*R s* + *v* ∗*R t* | *u v s t*. *u* ≥ *0* ∧ *v* ≥ *0* ∧ *u* + *v* = *1* ∧ *s* ∈ *S* ∧ *t* ∈ *T*}
(**is** *?lhs* = *?rhs*)
proof
def *I* ≡ {*1*::*nat*, *2*}
def *s* ≡ λ*i*. *if i* = (*1*::*nat*) *then S else T*
have ⋃(*s* ‘ *I*) = *S* ∪ *T*
using *s-def I-def* **by** *auto*
then have *convex hull* (⋃(*s* ‘ *I*)) = *convex hull* (*S* ∪ *T*)
by *auto*
moreover have *convex hull* ⋃(*s* ‘ *I*) =
{∑ *i*∈*I*. *c i* ∗*R sa i* | *c sa*. (∀ *i*∈*I*. *0* ≤ *c i*) ∧ *setsum c I* = *1* ∧ (∀ *i*∈*I*. *sa i* ∈ *s i*)}
apply (*subst convex-hull-finite-union*[*of I s*])
using *assms s-def I-def*
apply *auto*
done
moreover have
{∑ *i*∈*I*. *c i* ∗*R sa i* | *c sa*. (∀ *i*∈*I*. *0* ≤ *c i*) ∧ *setsum c I* = *1* ∧ (∀ *i*∈*I*. *sa i* ∈ *s i*)} ≤ *?rhs*
using *s-def I-def* **by** *auto*
ultimately show *?lhs* ⊆ *?rhs* **by** *auto*
{
fix *x*
assume *x* ∈ *?rhs*
then obtain *u v s t* **where** ∗: *x* = *u* ∗*R s* + *v* ∗*R t* ∧ *u* ≥ *0* ∧ *v* ≥ *0* ∧ *u* + *v* = *1* ∧ *s* ∈ *S* ∧ *t* ∈ *T*
by *auto*
then have *x* ∈ *convex hull* {*s*, *t*}
using *convex-hull-2*[*of s t*] **by** *auto*
then have *x* ∈ *convex hull* (*S* ∪ *T*)
using ∗ *hull-mono*[*of* {*s*, *t*} *S* ∪ *T*] **by** *auto*
}
then show *?lhs* ⊇ *?rhs* **by** *blast*
qed

19.33 Convexity on direct sums

lemma *closure-sum*:
fixes S T :: *'a*::*real-normed-vector set*
shows *closure* S + *closure* $T \subseteq$ *closure* $(S + T)$
unfolding *set-plus-image closure-Times* [*symmetric*] *split-def*
by (*intro closure-bounded-linear-image-subset bounded-linear-add bounded-linear-fst bounded-linear-snd*)

lemma *rel-interior-sum*:
fixes S T :: *'n*::*euclidean-space set*
assumes *convex* S
and *convex* T
shows *rel-interior* $(S + T)$ = *rel-interior* S + *rel-interior* T
proof −
have *rel-interior* S + *rel-interior* T = $(\lambda(x,y).\ x + y)$ ‘ (*rel-interior* S × *rel-interior* T)
by (*simp add*: *set-plus-image*)
also have ... = $(\lambda(x,y).\ x + y)$ ‘ *rel-interior* $(S \times T)$
using *rel-interior-direct-sum assms* **by** *auto*
also have ... = *rel-interior* $(S + T)$
using *fst-snd-linear convex-Times assms*
rel-interior-convex-linear-image[*of* $(\lambda(x,y).\ x + y)$ $S \times T$]
by (*auto simp add*: *set-plus-image*)
finally show *?thesis* **..**
qed

lemma *rel-interior-sum-gen*:
fixes S :: *'a* ⇒ *'n*::*euclidean-space set*
assumes ∀ i∈I. *convex* $(S\ i)$
shows *rel-interior* $(setsum\ S\ I)$ = $setsum\ (\lambda i.\ rel\text{-}interior\ (S\ i))\ I$
apply (*subst setsum-set-cond-linear*[*of convex*])
using *rel-interior-sum rel-interior-sing*[*of 0*] *assms*
apply (*auto simp add*: *convex-set-plus*)
done

lemma *convex-rel-open-direct-sum*:
fixes S T :: *'n*::*euclidean-space set*
assumes *convex* S
and *rel-open* S
and *convex* T
and *rel-open* T
shows *convex* $(S \times T) \wedge$ *rel-open* $(S \times T)$
by (*metis assms convex-Times rel-interior-direct-sum rel-open-def*)

lemma *convex-rel-open-sum*:
fixes S T :: *'n*::*euclidean-space set*
assumes *convex* S
and *rel-open* S
and *convex* T

```
    and rel-open T
  shows convex (S + T) ∧ rel-open (S + T)
  by (metis assms convex-set-plus rel-interior-sum rel-open-def)

lemma convex-hull-finite-union-cones:
  assumes finite I
    and I ≠ {}
  assumes ∀ i∈I. convex (S i) ∧ cone (S i) ∧ S i ≠ {}
  shows convex hull (⋃(S ' I)) = setsum S I
  (is ?lhs = ?rhs)
proof −
  {
    fix x
    assume x ∈ ?lhs
    then obtain c xs where
      x: x = setsum (λi. c i *R xs i) I ∧ (∀ i∈I. c i ≥ 0) ∧ setsum c I = 1 ∧
(∀ i∈I. xs i ∈ S i)
      using convex-hull-finite-union[of I S] assms by auto
    def s ≡ λi. c i *R xs i
    {
      fix i
      assume i ∈ I
      then have s i ∈ S i
        using s-def x assms mem-cone[of S i xs i c i] by auto
    }
    then have ∀ i∈I. s i ∈ S i by auto
    moreover have x = setsum s I using x s-def by auto
    ultimately have x ∈ ?rhs
      using set-setsum-alt[of I S] assms by auto
  }
  moreover
  {
    fix x
    assume x ∈ ?rhs
    then obtain s where x: x = setsum s I ∧ (∀ i∈I. s i ∈ S i)
      using set-setsum-alt[of I S] assms by auto
    def xs ≡ λi. of-nat(card I) *R s i
    then have x = setsum (λi. ((1 :: real) / of-nat(card I)) *R xs i) I
      using x assms by auto
    moreover have ∀ i∈I. xs i ∈ S i
      using x xs-def assms by (simp add: cone-def)
    moreover have ∀ i∈I. (1 :: real) / of-nat (card I) ≥ 0
      by auto
    moreover have setsum (λi. (1 :: real) / of-nat (card I)) I = 1
      using assms by auto
    ultimately have x ∈ ?lhs
      apply (subst convex-hull-finite-union[of I S])
      using assms
      apply blast
```

```
    using assms
    apply blast
    apply rule
    apply (rule-tac x = (λi. (1 :: real) / of-nat (card I)) in exI)
    apply auto
    done
  }
  ultimately show ?thesis by auto
qed

lemma convex-hull-union-cones-two:
  fixes S T :: 'm::euclidean-space set
  assumes convex S
    and cone S
    and S ≠ {}
  assumes convex T
    and cone T
    and T ≠ {}
  shows convex hull (S ∪ T) = S + T
proof −
  def I ≡ {1::nat, 2}
  def A ≡ (λi. if i = (1::nat) then S else T)
  have ⋃(A ‘ I) = S ∪ T
    using A-def I-def by auto
  then have convex hull (⋃(A ‘ I)) = convex hull (S ∪ T)
    by auto
  moreover have convex hull ⋃(A ‘ I) = setsum A I
    apply (subst convex-hull-finite-union-cones[of I A])
    using assms A-def I-def
    apply auto
    done
  moreover have setsum A I = S + T
    using A-def I-def
    unfolding set-plus-def
    apply auto
    unfolding set-plus-def
    apply auto
    done
  ultimately show ?thesis by auto
qed

lemma rel-interior-convex-hull-union:
  fixes S :: 'a ⇒ 'n::euclidean-space set
  assumes finite I
    and ∀ i∈I. convex (S i) ∧ S i ≠ {}
  shows rel-interior (convex hull (⋃(S ‘ I))) =
    {setsum (λi. c i *R s i) I | c s. (∀ i∈I. c i > 0) ∧ setsum c I = 1 ∧
      (∀ i∈I. s i ∈ rel-interior(S i))}
  (is ?lhs = ?rhs)
```

```
proof (cases I = {})
  case True
  then show ?thesis
    using convex-hull-empty rel-interior-empty by auto
next
  case False
  def C0 ≡ convex hull (⋃(S ‘ I))
  have ∀ i∈I. C0 ≥ S i
    unfolding C0-def using hull-subset[of ⋃(S ‘ I)] by auto
  def K0 ≡ cone hull ({1 :: real} × C0)
  def K ≡ λi. cone hull ({1 :: real} × S i)
  have ∀ i∈I. K i ≠ {}
    unfolding K-def using assms
    by (simp add: cone-hull-empty-iff[symmetric])
  {
    fix i
    assume i ∈ I
    then have convex (K i)
      unfolding K-def
      apply (subst convex-cone-hull)
      apply (subst convex-Times)
      using assms
      apply auto
      done
  }
  then have convK: ∀ i∈I. convex (K i)
    by auto
  {
    fix i
    assume i ∈ I
    then have K0 ⊇ K i
      unfolding K0-def K-def
      apply (subst hull-mono)
      using ‹∀ i∈I. C0 ≥ S i›
      apply auto
      done
  }
  then have K0 ⊇ ⋃(K ‘ I) by auto
  moreover have convex K0
    unfolding K0-def
    apply (subst convex-cone-hull)
    apply (subst convex-Times)
    unfolding C0-def
    using convex-convex-hull
    apply auto
    done
  ultimately have geq: K0 ⊇ convex hull (⋃(K ‘ I))
    using hull-minimal[of - K0 convex] by blast
  have ∀ i∈I. K i ⊇ {1 :: real} × S i
```

using *K-def* **by** (*simp add*: *hull-subset*)
then have $\bigcup(K\ `\ I) \supseteq \{1 :: real\} \times \bigcup(S\ `\ I)$
by *auto*
then have *convex hull* $\bigcup(K\ `\ I) \supseteq$ *convex hull* $(\{1 :: real\} \times \bigcup(S\ `\ I))$
by (*simp add*: *hull-mono*)
then have *convex hull* $\bigcup(K\ `\ I) \supseteq \{1 :: real\} \times C0$
unfolding *C0-def*
using *convex-hull-Times*[*of* $\{(1 :: real)\}$ $\bigcup(S\ `\ I)$] *convex-hull-singleton*
by *auto*
moreover have *cone* (*convex hull* $(\bigcup(K\ `\ I)))$
apply (*subst cone-convex-hull*)
using *cone-Union*[*of K ` I*]
apply *auto*
unfolding *K-def*
using *cone-cone-hull*
apply *auto*
done
ultimately have *convex hull* $(\bigcup(K\ `\ I)) \supseteq K0$
unfolding *K0-def*
using *hull-minimal*[*of - convex hull* $(\bigcup(K\ `\ I))$ *cone*]
by *blast*
then have *K0* = *convex hull* $(\bigcup(K\ `\ I))$
using *geq* **by** *auto*
also have ... = *setsum K I*
apply (*subst convex-hull-finite-union-cones*[*of I K*])
using *assms*
apply *blast*
using *False*
apply *blast*
unfolding *K-def*
apply *rule*
apply (*subst convex-cone-hull*)
apply (*subst convex-Times*)
using *assms cone-cone-hull* ⟨$\forall i{\in}I.\ K\ i \neq \{\}$⟩ *K-def*
apply *auto*
done
finally have *K0* = *setsum K I* **by** *auto*
then have ∗: *rel-interior K0* = *setsum* $(\lambda i.\ (rel\text{-}interior\ (K\ i)))\ I$
using *rel-interior-sum-gen*[*of I K*] *convK* **by** *auto*
{
fix *x*
assume *x* ∈ *?lhs*
then have (*1*::*real*, *x*) ∈ *rel-interior K0*
using *K0-def C0-def rel-interior-convex-cone-aux*[*of C0 1*::*real x*] *convex-convex-hull*
by *auto*
then obtain *k* **where** *k*: (*1*::*real*, *x*) = *setsum k I* ∧ ($\forall i{\in}I.\ k\ i \in rel\text{-}interior$ (*K i*))
using ⟨*finite I*⟩ ∗ *set-setsum-alt*[*of I* $\lambda i.$ *rel-interior* (*K i*)] **by** *auto*
{

```
    fix i
    assume i ∈ I
    then have convex (S i) ∧ k i ∈ rel-interior (cone hull {1} × S i)
      using k K-def assms by auto
    then have ∃ ci si. k i = (ci, ci *R si) ∧ 0 < ci ∧ si ∈ rel-interior (S i)
      using rel-interior-convex-cone[of S i] by auto
  }
  then obtain c s where
    cs: ∀ i∈I. k i = (c i, c i *R s i) ∧ 0 < c i ∧ s i ∈ rel-interior (S i)
    by metis
  then have x = (∑ i∈I. c i *R s i) ∧ setsum c I = 1
    using k by (simp add: setsum-prod)
  then have x ∈ ?rhs
    using k
    apply auto
    apply (rule-tac x = c in exI)
    apply (rule-tac x = s in exI)
    using cs
    apply auto
    done
 }
 moreover
 {
  fix x
  assume x ∈ ?rhs
  then obtain c s where cs: x = setsum (λi. c i *R s i) I ∧
      (∀ i∈I. c i > 0) ∧ setsum c I = 1 ∧ (∀ i∈I. s i ∈ rel-interior (S i))
    by auto
  def k ≡ λi. (c i, c i *R s i)
  {
    fix i assume i:I
    then have k i ∈ rel-interior (K i)
      using k-def K-def assms cs rel-interior-convex-cone[of S i]
      by auto
  }
  then have (1::real, x) ∈ rel-interior K0
    using K0-def * set-setsum-alt[of I (λi. rel-interior (K i))] assms k-def cs
    apply auto
    apply (rule-tac x = k in exI)
    apply (simp add: setsum-prod)
    done
  then have x ∈ ?lhs
    using K0-def C0-def rel-interior-convex-cone-aux[of C0 1 x]
    by (auto simp add: convex-convex-hull)
 }
 ultimately show ?thesis by blast
qed
```

lemma *convex-le-Inf-differential*:
fixes $f :: real \Rightarrow real$
assumes *convex-on I f*
and $x \in interior\ I$
and $y \in I$
shows $f\ y \geq f\ x + Inf\ ((\lambda t.\ (f\ x - f\ t)\ /\ (x - t))\ `\ (\{x<..\} \cap I)) * (y - x)$
(**is** - $\geq$ - + *Inf* (*?F x*) $*$ $(y - x)$)
proof (*cases rule*: *linorder-cases*)
assume $x < y$
moreover
have *open* (*interior I*) **by** *auto*
from *openE*[*OF this* ‹$x \in interior\ I$›]
obtain *e* **where** *e*: $0 < e$ *ball x e* $\subseteq$ *interior I* **.**
moreover def $t \equiv min\ (x + e\ /\ 2)\ ((x + y)\ /\ 2)$
ultimately have $x < t\ t < y\ t \in ball\ x\ e$
by (*auto simp*: *dist-real-def field-simps split*: *split-min*)
with ‹$x \in interior\ I$› *e interior-subset*[*of I*] **have** $t \in I\ x \in I$ **by** *auto*

have *open* (*interior I*) **by** *auto*
from *openE*[*OF this* ‹$x \in interior\ I$›]
obtain *e* **where** $0 < e$ *ball x e* $\subseteq$ *interior I* **.**
moreover def $K \equiv x - e\ /\ 2$
with ‹$0 < e$› **have** $K \in ball\ x\ e\ K < x$
by (*auto simp*: *dist-real-def*)
ultimately have $K \in I\ K < x\ x \in I$
using *interior-subset*[*of I*] ‹$x \in interior\ I$› **by** *auto*

have *Inf* (*?F x*) $\leq (f\ x - f\ y)\ /\ (x - y)$
proof (*intro bdd-belowI cInf-lower2*)
show $(f\ x - f\ t)\ /\ (x - t) \in$ *?F x*
using ‹$t \in I$› ‹$x < t$› **by** *auto*
show $(f\ x - f\ t)\ /\ (x - t) \leq (f\ x - f\ y)\ /\ (x - y)$
using ‹*convex-on I f*› ‹$x \in I$› ‹$y \in I$› ‹$x < t$› ‹$t < y$›
by (*rule convex-on-diff*)
next
fix *y*
assume $y \in$ *?F x*
with *order-trans*[*OF convex-on-diff*[*OF* ‹*convex-on I f*› ‹$K \in I$› - ‹$K < x$› -]]
show $(f\ K - f\ x)\ /\ (K - x) \leq y$ **by** *auto*
qed
then show *?thesis*
using ‹$x < y$› **by** (*simp add*: *field-simps*)
next
assume $y < x$
moreover
have *open* (*interior I*) **by** *auto*
from *openE*[*OF this* ‹$x \in interior\ I$›]
obtain *e* **where** *e*: $0 < e$ *ball x e* $\subseteq$ *interior I* **.**
moreover def $t \equiv x + e\ /\ 2$

ultimately have $x < t$ $t \in$ *ball* x e
by (*auto simp*: *dist-real-def field-simps*)
with ‹$x \in$ *interior* I› e *interior-subset*[*of* I] **have** $t \in I$ $x \in I$ **by** *auto*

have $(f\ x - f\ y) / (x - y) \leq$ *Inf* (*?F* x)
proof (*rule cInf-greatest*)
have $(f\ x - f\ y) / (x - y) = (f\ y - f\ x) / (y - x)$
using ‹$y < x$› **by** (*auto simp*: *field-simps*)
also
fix z
assume $z \in$ *?F* x
with *order-trans*[*OF convex-on-diff*[*OF* ‹*convex-on* I f› ‹$y \in I$› - ‹$y < x$›]]
have $(f\ y - f\ x) / (y - x) \leq z$
by *auto*
finally show $(f\ x - f\ y) / (x - y) \leq z$ **.**
next
have *open* (*interior* I) **by** *auto*
from *openE*[*OF this* ‹$x \in$ *interior* I›]
obtain e **where** e: $0 < e$ *ball* x $e \subseteq$ *interior* I **.**
then have $x + e / 2 \in$ *ball* x e
by (*auto simp*: *dist-real-def*)
with e *interior-subset*[*of* I] **have** $x + e / 2 \in \{x<..\} \cap I$
by *auto*
then show *?F* $x \neq \{\}$
by *blast*
qed
then show *?thesis*
using ‹$y < x$› **by** (*simp add*: *field-simps*)
qed *simp*

19.34 Explicit formulas for interior and relative interior of convex hull

lemma *interior-atLeastAtMost* [*simp*]:
fixes a::*real* **shows** *interior* $\{a..b\} = \{a<..<b\}$
using *interior-cbox* [*of* a b] **by** *auto*

lemma *interior-atLeastLessThan* [*simp*]:
fixes a::*real* **shows** *interior* $\{a..<b\} = \{a<..<b\}$
by (*metis atLeastLessThan-def greaterThanLessThan-def interior-atLeastAtMost interior-Int interior-interior interior-real-semiline*)

lemma *interior-lessThanAtMost* [*simp*]:
fixes a::*real* **shows** *interior* $\{a<..b\} = \{a<..<b\}$
by (*metis atLeastAtMost-def greaterThanAtMost-def interior-atLeastAtMost interior-Int interior-interior interior-real-semiline*)

lemma *at-within-closed-interval*:
fixes x::*real*

shows $a < x \Longrightarrow x < b \Longrightarrow (at\ x\ within\ \{a..b\}) = at\ x$
by (*metis at-within-interior greaterThanLessThan-iff interior-atLeastAtMost*)

lemma *affine-independent-convex-affine-hull*:
fixes $s :: 'a::euclidean\text{-}space\ set$
assumes $\neg affine\text{-}dependent\ s\ t \subseteq s$
shows $convex\ hull\ t = affine\ hull\ t \cap convex\ hull\ s$
proof −
have *fin*: *finite s finite t* **using** *assms aff-independent-finite finite-subset* **by** *auto*
{ fix *u v x*
assume *uv*: $setsum\ u\ t = 1\ \forall x{\in}s.\ 0 \leq v\ x\ setsum\ v\ s = 1$
$(\sum x{\in}s.\ v\ x *_R x) = (\sum v{\in}t.\ u\ v *_R v)\ x \in t$
then have *s*: $s = (s - t) \cup t$ — split into separate cases
using *assms* **by** *auto*
have [*simp*]: $(\sum x{\in}t.\ v\ x *_R x) + (\sum x{\in}s - t.\ v\ x *_R x) = (\sum x{\in}t.\ u\ x *_R x)$
$setsum\ v\ t + setsum\ v\ (s - t) = 1$
using *uv fin s*
by (*auto simp*: *setsum.union-disjoint* [*symmetric*] *Un-commute*)
have $(\sum x{\in}s.\ if\ x \in t\ then\ v\ x - u\ x\ else\ v\ x) = 0$
$(\sum x{\in}s.\ (if\ x \in t\ then\ v\ x - u\ x\ else\ v\ x) *_R x) = 0$
using *uv fin*
by (*subst s*, *subst setsum.union-disjoint*, *auto simp*: *algebra-simps setsum-subtractf*)+
} note [*simp*] = *this*
have $convex\ hull\ t \subseteq affine\ hull\ t$
using *convex-hull-subset-affine-hull* **by** *blast*
moreover have $convex\ hull\ t \subseteq convex\ hull\ s$
using *assms hull-mono* **by** *blast*
moreover have $affine\ hull\ t \cap convex\ hull\ s \subseteq convex\ hull\ t$
using *assms*
apply (*simp add*: *convex-hull-finite affine-hull-finite fin affine-dependent-explicit*)
apply (*drule-tac x*=*s* **in** *spec*)
apply (*auto simp*: *fin*)
apply (*rule-tac x*=*u* **in** *exI*)
apply (*rename-tac v*)
apply (*drule-tac x*=$\lambda x.\ if\ x \in t\ then\ v\ x - u\ x\ else\ v\ x$ **in** *spec*)
apply (*force*)+
done
ultimately show *?thesis*
by *blast*
qed

lemma *affine-independent-span-eq*:
fixes $s :: 'a::euclidean\text{-}space\ set$
assumes $\neg affine\text{-}dependent\ s\ card\ s = Suc\ (DIM\ ('a))$
shows $affine\ hull\ s = UNIV$
proof (*cases* $s = \{\}$)
case *True* **then show** *?thesis*
using *assms* **by** *simp*

next
case *False*
then obtain *a t* **where** *t*: $a \notin t$ $s = insert\ a\ t$
by *blast*
then have *fin*: *finite t* **using** *assms*
by (*metis finite-insert aff-independent-finite*)
show *?thesis*
using *assms t fin*
apply (*simp add*: *affine-dependent-iff-dependent affine-hull-insert-span-gen*)
apply (*rule subset-antisym*)
apply *force*
apply (*rule Fun.vimage-subsetD*)
apply (*metis add.commute diff-add-cancel surj-def*)
apply (*rule card-ge-dim-independent*)
apply (*auto simp*: *card-image inj-on-def dim-subset-UNIV*)
done
qed

lemma *affine-independent-span-gt*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *ind*: ~ *affine-dependent s* **and** *dim*: $DIM\ ('a) < card\ s$
shows *affine hull s* = *UNIV*
apply (*rule affine-independent-span-eq* [*OF ind*])
apply (*rule antisym*)
using *assms*
apply *auto*
apply (*metis add-2-eq-Suc' not-less-eq-eq affine-dependent-biggerset aff-independent-finite*)
done

lemma *empty-interior-affine-hull*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *finite s* **and** *dim*: $card\ s \leq DIM\ ('a)$
shows $interior(affine\ hull\ s) = \{\}$
using *assms*
apply (*induct s rule*: *finite-induct*)
apply (*simp-all add*: *affine-dependent-iff-dependent affine-hull-insert-span-gen interior-translation*)
apply (*rule empty-interior-lowdim*)
apply (*simp add*: *affine-dependent-iff-dependent affine-hull-insert-span-gen*)
apply (*metis Suc-le-lessD not-less order-trans card-image-le finite-imageI dim-le-card*)
done

lemma *empty-interior-convex-hull*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *finite s* **and** *dim*: $card\ s \leq DIM\ ('a)$
shows $interior(convex\ hull\ s) = \{\}$
by (*metis Diff-empty Diff-eq-empty-iff convex-hull-subset-affine-hull interior-mono empty-interior-affine-hull* [*OF assms*])

lemma *explicit-subset-rel-interior-convex-hull*:
fixes s :: *'a*::*euclidean-space set*
shows *finite s*
$\Longrightarrow \{y.\ \exists u.\ (\forall x \in s.\ 0 < u\ x \wedge u\ x < 1) \wedge setsum\ u\ s = 1 \wedge setsum\ (\lambda x.\ u\ x *_R x)\ s = y\}$
$\subseteq$ *rel-interior* (*convex hull s*)
by (*force simp add*: *rel-interior-convex-hull-union* [**where** $S{=}\lambda x.\ \{x\}$ **and** $I{=}s$, *simplified*])

lemma *explicit-subset-rel-interior-convex-hull-minimal*:
fixes s :: *'a*::*euclidean-space set*
shows *finite s*
$\Longrightarrow \{y.\ \exists u.\ (\forall x \in s.\ 0 < u\ x) \wedge setsum\ u\ s = 1 \wedge setsum\ (\lambda x.\ u\ x *_R x)\ s = y\}$
$\subseteq$ *rel-interior* (*convex hull s*)
by (*force simp add*: *rel-interior-convex-hull-union* [**where** $S{=}\lambda x.\ \{x\}$ **and** $I{=}s$, *simplified*])

lemma *rel-interior-convex-hull-explicit*:
fixes s :: *'a*::*euclidean-space set*
assumes ~ *affine-dependent s*
shows *rel-interior*(*convex hull s*) =
$\{y.\ \exists u.\ (\forall x \in s.\ 0 < u\ x) \wedge setsum\ u\ s = 1 \wedge setsum\ (\lambda x.\ u\ x *_R x)\ s = y\}$
(**is** *?lhs* = *?rhs*)
proof
show *?rhs* $\leq$ *?lhs*
by (*simp add*: *aff-independent-finite explicit-subset-rel-interior-convex-hull-minimal assms*)
next
show *?lhs* $\leq$ *?rhs*
proof (*cases* $\exists a.\ s = \{a\}$)
case *True* **then show** *?lhs* $\leq$ *?rhs*
by *force*
next
case *False*
have *fs*: *finite s*
using *assms* **by** (*simp add*: *aff-independent-finite*)
{ fix *a b* **and** *d*::*real*
assume *ab*: $a \in s\ b \in s\ a \neq b$
then have *s*: $s = (s - \{a,b\}) \cup \{a,b\}$ — split into separate cases
by *auto*
have $(\sum x{\in}s.\ if\ x = a\ then\ -\ d\ else\ if\ x = b\ then\ d\ else\ 0) = 0$
$(\sum x{\in}s.\ (if\ x = a\ then\ -\ d\ else\ if\ x = b\ then\ d\ else\ 0) *_R x) = d *_R b - d *_R a$
using *ab fs*
by (*subst s*, *subst setsum.union-disjoint*, *auto*)+
} note [*simp*] = *this*
{ fix *y*

```
    assume y: y ∈ convex hull s y ∉ ?rhs
    { fix u T a
      assume ua: ∀x∈s. 0 ≤ u x setsum u s = 1 ¬ 0 < u a a ∈ s
        and yT: y = (∑x∈s. u x *R x) y ∈ T open T
        and sb: T ∩ affine hull s ⊆ {w. ∃u. (∀x∈s. 0 ≤ u x) ∧ setsum u s = 1
∧ (∑x∈s. u x *R x) = w}
      have ua0: u a = 0
        using ua by auto
      obtain b where b: b∈s a ≠ b
        using ua False by auto
      obtain e where e: 0 < e ball (∑x∈s. u x *R x) e ⊆ T
        using yT by (auto elim: openE)
      with b obtain d where d: 0 < d norm(d *R (a−b)) < e
        by (auto intro: that [of e / 2 / norm(a−b)])
      have (∑x∈s. u x *R x) ∈ affine hull s
        using yT y by (metis affine-hull-convex-hull hull-redundant-eq)
      then have (∑x∈s. u x *R x) − d *R (a − b) ∈ affine hull s
        using ua b by (auto simp: hull-inc intro: mem-affine-3-minus2)
      then have y − d *R (a − b) ∈ T ∩ affine hull s
        using d e yT by auto
      then obtain v where ∀x∈s. 0 ≤ v x
                          setsum v s = 1
                          (∑x∈s. v x *R x) = (∑x∈s. u x *R x) − d *R (a − b)
        using subsetD [OF sb] yT
        by auto
      then have False
        using assms
        apply (simp add: affine-dependent-explicit-finite fs)
         apply (drule-tac x=λx. (v x − u x) − (if x = a then −d else if x = b
then d else 0) in spec)
        using ua b d
        apply (auto simp: algebra-simps setsum-subtractf setsum.distrib)
        done
    } note ∗ = this
    have y ∉ rel-interior (convex hull s)
      using y
      apply (simp add: mem-rel-interior affine-hull-convex-hull)
      apply (auto simp: convex-hull-finite [OF fs])
      apply (drule-tac x=u in spec)
      apply (auto intro: ∗)
      done
  } with rel-interior-subset show ?lhs ≤ ?rhs
    by blast
 qed
qed

lemma interior-convex-hull-explicit-minimal:
 fixes s :: 'a::euclidean-space set
 shows
```

```
  ~ affine-dependent s
      ==> interior(convex hull s) =
          (if card(s) ≤ DIM('a) then {}
           else {y. ∃u. (∀x ∈ s. 0 < u x) ∧ setsum u s = 1 ∧ (∑x∈s. u x *R
x) = y})
  apply (simp add: aff-independent-finite empty-interior-convex-hull, clarify)
  apply (rule trans [of - rel-interior(convex hull s)])
 apply (simp add: affine-hull-convex-hull affine-independent-span-gt rel-interior-interior)
  by (simp add: rel-interior-convex-hull-explicit)

lemma interior-convex-hull-explicit:
  fixes s :: 'a::euclidean-space set
  assumes ~ affine-dependent s
  shows
   interior(convex hull s) =
           (if card(s) ≤ DIM('a) then {}
           else {y. ∃u. (∀x ∈ s. 0 < u x ∧ u x < 1) ∧ setsum u s = 1 ∧ (∑x∈s.
u x *R x) = y})
proof −
  { fix u :: 'a ⇒ real and a
    assume card Basis < card s and u: ⋀x. x∈s ⟹ 0 < u x setsum u s = 1 and
a: a ∈ s
    then have cs: Suc 0 < card s
      by (metis DIM-positive less-trans-Suc)
    obtain b where b: b ∈ s a ≠ b
    proof (cases s ≤ {a})
      case True
      then show thesis
        using cs subset-singletonD by fastforce
    next
      case False
      then show thesis
      by (blast intro: that)
    qed
    have u a + u b ≤ setsum u {a,b}
      using a b by simp
    also have ... ≤ setsum u s
      apply (rule Groups-Big.setsum-mono2)
      using a b u
      apply (auto simp: less-imp-le aff-independent-finite assms)
      done
    finally have u a < 1
      using ‹b ∈ s› u by fastforce
  } note [simp] = this
  show ?thesis
    using assms
    apply (auto simp: interior-convex-hull-explicit-minimal)
    apply (rule-tac x=u in exI)
    apply (auto simp: not-le)
```

done
qed

19.35 Similar results for closure and (relative or absolute) frontier.

lemma *closure-convex-hull* [*simp*]:
fixes *s* :: *'a*::*euclidean-space set*
shows *compact s* ==> *closure*(*convex hull s*) = *convex hull s*
by (*simp add*: *compact-imp-closed compact-convex-hull*)

lemma *rel-frontier-convex-hull-explicit*:
fixes *s* :: *'a*::*euclidean-space set*
assumes ~ *affine-dependent s*
shows *rel-frontier*(*convex hull s*) =
{*y*. ∃ *u*. (∀ *x* ∈ *s*. *0* ≤ *u x*) ∧ (∃ *x* ∈ *s*. *u x* = *0*) ∧ *setsum u s* = *1* ∧ *setsum* (λ*x*. *u x* ∗*R x*) *s* = *y*}
proof −
have *fs*: *finite s*
using *assms* **by** (*simp add*: *aff-independent-finite*)
show *?thesis*
apply (*simp add*: *rel-frontier-def finite-imp-compact rel-interior-convex-hull-explicit assms fs*)
apply (*auto simp*: *convex-hull-finite fs*)
apply (*drule-tac x*=*u* **in** *spec*)
apply (*rule-tac x*=*u* **in** *exI*)
apply *force*
apply (*rename-tac v*)
apply (*rule notE* [*OF assms*])
apply (*simp add*: *affine-dependent-explicit*)
apply (*rule-tac x*=*s* **in** *exI*)
apply (*auto simp*: *fs*)
apply (*rule-tac x* = λ*x*. *u x* − *v x* **in** *exI*)
apply (*force simp*: *setsum-subtractf scaleR-diff-left*)
done
qed

lemma *frontier-convex-hull-explicit*:
fixes *s* :: *'a*::*euclidean-space set*
assumes ~ *affine-dependent s*
shows *frontier*(*convex hull s*) =
{*y*. ∃ *u*. (∀ *x* ∈ *s*. *0* ≤ *u x*) ∧ (*DIM* (*'a*) < *card s* ⟶ (∃ *x* ∈ *s*. *u x* = *0*)) ∧
setsum u s = *1* ∧ *setsum* (λ*x*. *u x* ∗*R x*) *s* = *y*}
proof −
have *fs*: *finite s*
using *assms* **by** (*simp add*: *aff-independent-finite*)
show *?thesis*
proof (*cases DIM* (*'a*) < *card s*)

```
    case True
    with assms fs show ?thesis
      by (simp add: rel-frontier-def frontier-def rel-frontier-convex-hull-explicit
[symmetric]
          interior-convex-hull-explicit-minimal rel-interior-convex-hull-explicit)
  next
    case False
    then have card s ≤ DIM ('a)
      by linarith
    then show ?thesis
      using assms fs
     apply (simp add: frontier-def interior-convex-hull-explicit finite-imp-compact)
      apply (simp add: convex-hull-finite)
      done
  qed
qed

lemma rel-frontier-convex-hull-cases:
  fixes s :: 'a::euclidean-space set
  assumes ~ affine-dependent s
  shows rel-frontier(convex hull s) = ⋃{convex hull (s - {x}) |x. x ∈ s}
proof -
  have fs: finite s
    using assms by (simp add: aff-independent-finite)
  { fix u a
  have ∀x∈s. 0 ≤ u x ⟹ a ∈ s ⟹ u a = 0 ⟹ setsum u s = 1 ⟹
          ∃x v. x ∈ s ∧
               (∀x∈s - {x}. 0 ≤ v x) ∧
                  setsum v (s - {x}) = 1 ∧ (∑x∈s - {x}. v x *R x) = (∑x∈s.
u x *R x)
    apply (rule-tac x=a in exI)
    apply (rule-tac x=u in exI)
    apply (simp add: Groups-Big.setsum-diff1 fs)
    done }
  moreover
  { fix a u
    have a ∈ s ⟹ ∀x∈s - {a}. 0 ≤ u x ⟹ setsum u (s - {a}) = 1 ⟹
          ∃v. (∀x∈s. 0 ≤ v x) ∧
              (∃x∈s. v x = 0) ∧ setsum v s = 1 ∧ (∑x∈s. v x *R x) = (∑x∈s
- {a}. u x *R x)
    apply (rule-tac x=λx. if x = a then 0 else u x in exI)
    apply (auto simp: setsum.If-cases Diff-eq if-smult fs)
    done }
  ultimately show ?thesis
    using assms
    apply (simp add: rel-frontier-convex-hull-explicit)
    apply (simp add: convex-hull-finite fs Union-SetCompr-eq, auto)
    done
qed
```

lemma *frontier-convex-hull-eq-rel-frontier*:
fixes *s* :: *'a*::*euclidean-space set*
assumes ~ *affine-dependent s*
shows *frontier*(*convex hull s*) =
(*if card s* ≤ *DIM* (*'a*) *then convex hull s else rel-frontier*(*convex hull s*))
using *assms*
unfolding *rel-frontier-def frontier-def*
by (*simp add*: *affine-independent-span-gt rel-interior-interior*
finite-imp-compact empty-interior-convex-hull aff-independent-finite)

lemma *frontier-convex-hull-cases*:
fixes *s* :: *'a*::*euclidean-space set*
assumes ~ *affine-dependent s*
shows *frontier*(*convex hull s*) =
(*if card s* ≤ *DIM* (*'a*) *then convex hull s else* ⋃{*convex hull* (*s* − {*x*}) |*x*. *x* ∈ *s*})
by (*simp add*: *assms frontier-convex-hull-eq-rel-frontier rel-frontier-convex-hull-cases*)

lemma *in-frontier-convex-hull*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *finite s card s* ≤ *Suc* (*DIM* (*'a*)) *x* ∈ *s*
shows *x* ∈ *frontier*(*convex hull s*)
proof (*cases affine-dependent s*)
case *True*
with *assms* **show** *?thesis*
apply (*auto simp*: *affine-dependent-def frontier-def finite-imp-compact hull-inc*)
by (*metis card.insert-remove convex-hull-subset-affine-hull empty-interior-affine-hull finite-Diff hull-redundant insert-Diff insert-Diff-single insert-not-empty interior-mono not-less-eq-eq subset-empty*)
next
case *False*
{ assume *card s* = *Suc* (*card Basis*)
then have *cs*: *Suc 0* < *card s*
by (*simp add*: *DIM-positive*)
with *subset-singletonD* **have** ∃ *y* ∈ *s*. *y* ≠ *x*
by (*cases s* ≤ {*x*}) *fastforce*+
} note [*dest*!] = *this*
show *?thesis* **using** *assms*
unfolding *frontier-convex-hull-cases* [*OF False*] *Union-SetCompr-eq*
by (*auto simp*: *le-Suc-eq hull-inc*)
qed

lemma *not-in-interior-convex-hull*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *finite s card s* ≤ *Suc* (*DIM* (*'a*)) *x* ∈ *s*
shows *x* ∉ *interior*(*convex hull s*)
using *in-frontier-convex-hull* [*OF assms*]
by (*metis Diff-iff frontier-def*)

lemma *interior-convex-hull-eq-empty*:
fixes $s :: 'a::euclidean\text{-}space\ set$
assumes $card\ s = Suc\ (DIM\ ('a))$
shows $interior(convex\ hull\ s) = \{\} \longleftrightarrow affine\text{-}dependent\ s$
proof −
{ fix *a b*
assume *ab*: $a \in interior\ (convex\ hull\ s)\ b \in s\ b \in affine\ hull\ (s - \{b\})$
then have $interior(affine\ hull\ s) = \{\}$ **using** *assms*
by (*metis DIM-positive One-nat-def Suc-mono card.remove card-infinite empty-interior-affine-hull eq-iff hull-redundant insert-Diff not-less zero-le-one*)
then have *False* **using** *ab*
by (*metis convex-hull-subset-affine-hull equals0D interior-mono subset-eq*)
} then
show *?thesis*
using *assms*
apply *auto*
apply (*metis UNIV-I affine-hull-convex-hull affine-hull-empty affine-independent-span-eq convex-convex-hull empty-iff rel-interior-interior rel-interior-same-affine-hull*)
apply (*auto simp*: *affine-dependent-def*)
done
qed

19.36 Coplanarity, and collinearity in terms of affine hull

definition *coplanar* **where**
$coplanar\ s \equiv \exists u\ v\ w.\ s \subseteq affine\ hull\ \{u,v,w\}$

lemma *collinear-affine-hull*:
$collinear\ s \longleftrightarrow (\exists u\ v.\ s \subseteq affine\ hull\ \{u,v\})$
proof (*cases* $s=\{\}$)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
then obtain *x* **where** x: $x \in s$ **by** *auto*
{ fix *u*
assume $*$: $\bigwedge x\ y.\ [\![x \in s;\ y \in s]\!] \Longrightarrow \exists c.\ x - y = c *_R u$
have $\exists u\ v.\ s \subseteq \{a *_R u + b *_R v \mid a\ b.\ a + b = 1\}$
apply (*rule-tac* $x=x$ **in** *exI*)
apply (*rule-tac* $x=x+u$ **in** *exI*, *clarify*)
apply (*erule exE* [*OF* $*$ [*OF* x]])
apply (*rename-tac c*)
apply (*rule-tac* $x=1+c$ **in** *exI*)
apply (*rule-tac* $x=-c$ **in** *exI*)
apply (*simp add*: *algebra-simps*)
done
} moreover
{ fix *u v x y*

```
    assume *: s ⊆ {a *R u + b *R v |a b. a + b = 1}
    have x∈s ⟹ y∈s ⟹ ∃ c. x − y = c *R (v−u)
      apply (drule subsetD [OF *])+
      apply simp
      apply clarify
      apply (rename-tac r1 r2)
      apply (rule-tac x=r1−r2 in exI)
      apply (simp add: algebra-simps)
      apply (metis scaleR-left.add)
      done
  } ultimately
  show ?thesis
  unfolding collinear-def affine-hull-2
    by blast
qed

lemma collinear-imp-coplanar:
  collinear s ==> coplanar s
by (metis collinear-affine-hull coplanar-def insert-absorb2)

lemma collinear-small:
  assumes finite s card s ≤ 2
    shows collinear s
proof −
  have card s = 0 ∨ card s = 1 ∨ card s = 2
    using assms by linarith
  then show ?thesis using assms
    using card-eq-SucD
    by auto (metis collinear-2 numeral-2-eq-2)
qed

lemma coplanar-small:
  assumes finite s card s ≤ 3
    shows coplanar s
proof −
  have card s ≤ 2 ∨ card s = Suc (Suc (Suc 0))
    using assms by linarith
  then show ?thesis using assms
    apply safe
    apply (simp add: collinear-small collinear-imp-coplanar)
    apply (safe dest!: card-eq-SucD)
    apply (auto simp: coplanar-def)
    apply (metis hull-subset insert-subset)
    done
qed

lemma coplanar-empty: coplanar {}
  by (simp add: coplanar-small)
```

lemma *coplanar-sing*: *coplanar* {*a*}
by (*simp add*: *coplanar-small*)

lemma *coplanar-2*: *coplanar* {*a*,*b*}
by (*auto simp*: *card-insert-if coplanar-small*)

lemma *coplanar-3*: *coplanar* {*a*,*b*,*c*}
by (*auto simp*: *card-insert-if coplanar-small*)

lemma *collinear-affine-hull-collinear*: *collinear*(*affine hull s*) ⟷ *collinear s*
unfolding *collinear-affine-hull*
by (*metis affine-affine-hull subset-hull hull-hull hull-mono*)

lemma *coplanar-affine-hull-coplanar*: *coplanar*(*affine hull s*) ⟷ *coplanar s*
unfolding *coplanar-def*
by (*metis affine-affine-hull subset-hull hull-hull hull-mono*)

lemma *coplanar-linear-image*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*real-normed-vector*
assumes *coplanar s linear f* **shows** *coplanar*(*f* ‘ *s*)
proof −
{ fix *u v w*
assume *s* ⊆ *affine hull* {*u*, *v*, *w*}
then have *f* ‘ *s* ⊆ *f* ‘ (*affine hull* {*u*, *v*, *w*})
by (*simp add*: *image-mono*)
then have *f* ‘ *s* ⊆ *affine hull* (*f* ‘ {*u*, *v*, *w*})
by (*metis assms*(*2*) *linear-conv-bounded-linear affine-hull-linear-image*)
} then
show *?thesis*
by *auto* (*meson assms*(*1*) *coplanar-def*)
qed

lemma *coplanar-translation-imp*: *coplanar s* ⟹ *coplanar* ((λ*x*. *a* + *x*) ‘ *s*)
unfolding *coplanar-def*
apply *clarify*
apply (*rule-tac x*=*u*+*a* **in** *exI*)
apply (*rule-tac x*=*v*+*a* **in** *exI*)
apply (*rule-tac x*=*w*+*a* **in** *exI*)
using *affine-hull-translation* [*of a* {*u*,*v*,*w*} **for** *u v w*]
apply (*force simp*: *add.commute*)
done

lemma *coplanar-translation-eq*: *coplanar*((λ*x*. *a* + *x*) ‘ *s*) ⟷ *coplanar s*
by (*metis* (*no-types*) *coplanar-translation-imp translation-galois*)

lemma *coplanar-linear-image-eq*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *linear f inj f* **shows** *coplanar*(*f* ‘ *s*) = *coplanar s*
proof

assume *coplanar s*
then show *coplanar* (*f ' s*)
unfolding *coplanar-def*
using *affine-hull-linear-image* [*of f* {*u*,*v*,*w*} **for** *u v w*] *assms*
by (*meson coplanar-def coplanar-linear-image*)
next
obtain *g* **where** *g*: *linear g g* ∘ *f* = *id*
using *linear-injective-left-inverse* [*OF assms*]
by *blast*
assume *coplanar* (*f ' s*)
then obtain *u v w* **where** *f ' s* ⊆ *affine hull* {*u*, *v*, *w*}
by (*auto simp*: *coplanar-def*)
then have *g ' f ' s* ⊆ *g '* (*affine hull* {*u*, *v*, *w*})
by *blast*
then have *s* ⊆ *g '* (*affine hull* {*u*, *v*, *w*})
using *g* **by** (*simp add*: *Fun.image-comp*)
then show *coplanar s*
unfolding *coplanar-def*
using *affine-hull-linear-image* [*of g* {*u*,*v*,*w*} **for** *u v w*] ‹*linear g*› *linear-conv-bounded-linear*
by *fastforce*
qed

lemma *coplanar-subset*: ⟦*coplanar t*; *s* ⊆ *t*⟧ ⟹ *coplanar s*
by (*meson coplanar-def order-trans*)

lemma *affine-hull-3-imp-collinear*: *c* ∈ *affine hull* {*a*,*b*} ⟹ *collinear* {*a*,*b*,*c*}
by (*metis collinear-2 collinear-affine-hull-collinear hull-redundant insert-commute*)

lemma *collinear-3-imp-in-affine-hull*: ⟦*collinear* {*a*,*b*,*c*}; *a* ≠ *b*⟧ ⟹ *c* ∈ *affine hull* {*a*,*b*}
unfolding *collinear-def*
apply *clarify*
apply (*frule-tac x*=*b* **in** *bspec*, *blast*, *drule-tac x*=*a* **in** *bspec*, *blast*, *erule exE*)
apply (*drule-tac x*=*c* **in** *bspec*, *blast*, *drule-tac x*=*a* **in** *bspec*, *blast*, *erule exE*)
apply (*rename-tac y x*)
apply (*simp add*: *affine-hull-2*)
apply (*rule-tac x*=*1* − *x*/*y* **in** *exI*)
apply (*simp add*: *algebra-simps*)
done

lemma *collinear-3-affine-hull*:
assumes *a* ≠ *b*
shows *collinear* {*a*,*b*,*c*} ⟷ *c* ∈ *affine hull* {*a*,*b*}
using *affine-hull-3-imp-collinear assms collinear-3-imp-in-affine-hull* **by** *blast*

lemma *collinear-3-eq-affine-dependent*:
collinear{*a*,*b*,*c*} ⟷ *a* = *b* ∨ *a* = *c* ∨ *b* = *c* ∨ *affine-dependent* {*a*,*b*,*c*}
apply (*case-tac a*=*b*, *simp*)

apply (*case-tac a=c*)
apply (*simp add*: *insert-commute*)
apply (*case-tac b=c*)
apply (*simp add*: *insert-commute*)
apply (*auto simp*: *affine-dependent-def collinear-3-affine-hull insert-Diff-if*)
apply (*metis collinear-3-affine-hull insert-commute*)+
done

lemma *affine-dependent-imp-collinear-3*:
affine-dependent {*a*,*b*,*c*} ⟹ *collinear*{*a*,*b*,*c*}
by (*simp add*: *collinear-3-eq-affine-dependent*)

lemma *collinear-3*: *NO-MATCH 0 x* ⟹ *collinear* {*x*,*y*,*z*} ⟷ *collinear* {*0*, *x*−*y*, *z*−*y*}
by (*auto simp add*: *collinear-def*)

thm *affine-hull-nonempty*
corollary *affine-hull-eq-empty* [*simp*]: *affine hull S* = {} ⟷ *S* = {}
using *affine-hull-nonempty* **by** *blast*

lemma *affine-hull-2-alt*:
fixes *a b* :: *'a*::*real-vector*
shows *affine hull* {*a*,*b*} = *range* (λ*u*. *a* + *u* ∗*R* (*b* − *a*))
apply (*simp add*: *affine-hull-2*, *safe*)
apply (*rule-tac x=v* **in** *image-eqI*)
apply (*simp add*: *algebra-simps*)
apply (*metis scaleR-add-left scaleR-one*, *simp*)
apply (*rule-tac x=1−u* **in** *exI*)
apply (*simp add*: *algebra-simps*)
done

lemma *interior-convex-hull-3-minimal*:
fixes *a* :: *'a*::*euclidean-space*
shows ⟦~ *collinear*{*a*,*b*,*c*}; *DIM*(*'a*) = *2*⟧
⟹ *interior*(*convex hull* {*a*,*b*,*c*}) =
{*v*. ∃*x y z*. *0* < *x* ∧ *0* < *y* ∧ *0* < *z* ∧ *x* + *y* + *z* = *1* ∧
x ∗*R* *a* + *y* ∗*R* *b* + *z* ∗*R* *c* = *v*}
apply (*simp add*: *collinear-3-eq-affine-dependent interior-convex-hull-explicit-minimal*, *safe*)
apply (*rule-tac x=u a* **in** *exI*, *simp*)
apply (*rule-tac x=u b* **in** *exI*, *simp*)
apply (*rule-tac x=u c* **in** *exI*, *simp*)
apply (*rename-tac uu x y z*)
apply (*rule-tac x=*λ*r*. (*if r=a then x else if r=b then y else if r=c then z else 0*)
in *exI*)
apply *simp*
done

19.37 The infimum of the distance between two sets

definition *setdist* :: *'a::metric-space set* ⇒ *'a set* ⇒ *real* **where**
setdist s t ≡
(*if s* = {} ∨ *t* = {} *then 0*
else Inf {*dist x y*| *x y*. *x* ∈ *s* ∧ *y* ∈ *t*})

lemma *setdist-empty1* [*simp*]: *setdist* {} *t* = *0*
by (*simp add*: *setdist-def*)

lemma *setdist-empty2* [*simp*]: *setdist t* {} = *0*
by (*simp add*: *setdist-def*)

lemma *setdist-pos-le*: *0* ≤ *setdist s t*
by (*auto simp*: *setdist-def ex-in-conv* [*symmetric*] *intro*: *cInf-greatest*)

lemma *le-setdistI*:
assumes *s* ≠ {} *t* ≠ {} ⋀*x y*. ⟦*x* ∈ *s*; *y* ∈ *t*⟧ ⟹ *d* ≤ *dist x y*
shows *d* ≤ *setdist s t*
using *assms*
by (*auto simp*: *setdist-def Set.ex-in-conv* [*symmetric*] *intro*: *cInf-greatest*)

lemma *setdist-le-dist*: ⟦*x* ∈ *s*; *y* ∈ *t*⟧ ⟹ *setdist s t* ≤ *dist x y*
unfolding *setdist-def*
by (*auto intro*!: *bdd-belowI* [**where** *m=0*] *cInf-lower*)

lemma *le-setdist-iff*:
d ≤ *setdist s t* ⟷
(∀ *x* ∈ *s*. ∀ *y* ∈ *t*. *d* ≤ *dist x y*) ∧ (*s* = {} ∨ *t* = {} ⟶ *d* ≤ *0*)
apply (*cases s* = {} ∨ *t* = {})
apply (*force simp add*: *setdist-def*)
apply (*intro iffI conjI*)
using *setdist-le-dist* **apply** *fastforce*
apply (*auto simp*: *intro*: *le-setdistI*)
done

lemma *setdist-ltE*:
assumes *setdist s t* < *b s* ≠ {} *t* ≠ {}
obtains *x y* **where** *x* ∈ *s y* ∈ *t dist x y* < *b*
using *assms*
by (*auto simp*: *not-le* [*symmetric*] *le-setdist-iff*)

lemma *setdist-refl*: *setdist s s* = *0*
apply (*cases s* = {})
apply (*force simp add*: *setdist-def*)
apply (*rule antisym* [*OF - setdist-pos-le*])
apply (*metis all-not-in-conv dist-self setdist-le-dist*)
done

lemma *setdist-sym*: *setdist s t* = *setdist t s*

by (*force simp*: *setdist-def dist-commute intro*!: *arg-cong* [**where** *f=Inf*])

lemma *setdist-triangle*: *setdist s t* ≤ *setdist s* {*a*} + *setdist* {*a*} *t*
proof (*cases s* = {} ∨ *t* = {})
case *True* **then show** *?thesis*
using *setdist-pos-le* **by** *fastforce*
next
case *False*
have ⋀*x*. *x* ∈ *s* ⟹ *setdist s t* − *dist x a* ≤ *setdist* {*a*} *t*
apply (*rule le-setdistI*, *blast*)
using *False* **apply** (*fastforce intro*: *le-setdistI*)
apply (*simp add*: *algebra-simps*)
apply (*metis dist-commute dist-triangle-alt order-trans* [*OF setdist-le-dist*])
done
then have *setdist s t* − *setdist* {*a*} *t* ≤ *setdist s* {*a*}
using *False* **by** (*fastforce intro*: *le-setdistI*)
then show *?thesis*
by (*simp add*: *algebra-simps*)
qed

lemma *setdist-singletons* [*simp*]: *setdist* {*x*} {*y*} = *dist x y*
by (*simp add*: *setdist-def*)

lemma *setdist-Lipschitz*: |*setdist* {*x*} *s* − *setdist* {*y*} *s*| ≤ *dist x y*
apply (*subst setdist-singletons* [*symmetric*])
by (*metis abs-diff-le-iff diff-le-eq setdist-triangle setdist-sym*)

lemma *continuous-at-setdist*: *continuous* (*at x*) (λ*y*. (*setdist* {*y*} *s*))
by (*force simp*: *continuous-at-eps-delta dist-real-def intro*: *le-less-trans* [*OF setdist-Lipschitz*])

lemma *continuous-on-setdist*: *continuous-on t* (λ*y*. (*setdist* {*y*} *s*))
by (*metis continuous-at-setdist continuous-at-imp-continuous-on*)

lemma *uniformly-continuous-on-setdist*: *uniformly-continuous-on t* (λ*y*. (*setdist* {*y*} *s*))
by (*force simp*: *uniformly-continuous-on-def dist-real-def intro*: *le-less-trans* [*OF setdist-Lipschitz*])

lemma *setdist-subset-right*: ⟦*t* ≠ {}; *t* ⊆ *u*⟧ ⟹ *setdist s u* ≤ *setdist s t*
apply (*cases s* = {} ∨ *u* = {}, *force*)
apply (*auto simp*: *setdist-def intro*!: *bdd-belowI* [**where** *m=0*] *cInf-superset-mono*)
done

lemma *setdist-subset-left*: ⟦*s* ≠ {}; *s* ⊆ *t*⟧ ⟹ *setdist t u* ≤ *setdist s u*
by (*metis setdist-subset-right setdist-sym*)

lemma *setdist-closure-1* [*simp*]: *setdist* (*closure s*) *t* = *setdist s t*
proof (*cases s* = {} ∨ *t* = {})
case *True* **then show** *?thesis* **by** *force*

next
case *False*
{ fix *y*
assume $y \in t$
have *continuous-on* (*closure s*) ($\lambda a.$ *dist a y*)
by (*auto simp*: *continuous-intros dist-norm*)
then have $*$: $\bigwedge x.\ x \in$ *closure s* $\Longrightarrow$ *setdist s t* $\leq$ *dist x y*
apply (*rule continuous-ge-on-closure*)
apply *assumption*
apply (*blast intro*: *setdist-le-dist* ‹$y \in t$›)
done
} note $* =$ *this*
show *?thesis*
apply (*rule antisym*)
using *False closure-subset* **apply** (*blast intro*: *setdist-subset-left*)
using *False* $*$
apply (*force simp add*: *closure-eq-empty intro*!: *le-setdistI*)
done
qed

lemma *setdist-closure-2* [*simp*]: *setdist t* (*closure s*) = *setdist t s*
by (*metis setdist-closure-1 setdist-sym*)

lemma *setdist-compact-closed*:
fixes *s* :: 'a::*euclidean-space set*
assumes *s*: *compact s* **and** *t*: *closed t*
and $s \neq \{\}$ $t \neq \{\}$
shows $\exists x \in s.\ \exists y \in t.$ *dist x y* = *setdist s t*
proof −
have $\{x - y \mid x\ y.\ x \in s \wedge y \in t\} \neq \{\}$
using *assms* **by** *blast*
then
have $\exists x \in s.\ \exists y \in t.$ *dist x y* $\leq$ *setdist s t*
using *distance-attains-inf* [**where** *a=0*, *OF compact-closed-differences* [*OF s t*]] *assms*
apply (*clarsimp simp*: *dist-norm le-setdist-iff*, *blast*)
done
then show *?thesis*
by (*blast intro*!: *antisym* [*OF - setdist-le-dist*])
qed

lemma *setdist-closed-compact*:
fixes *s* :: 'a::*euclidean-space set*
assumes *s*: *closed s* **and** *t*: *compact t*
and $s \neq \{\}$ $t \neq \{\}$
shows $\exists x \in s.\ \exists y \in t.$ *dist x y* = *setdist s t*
using *setdist-compact-closed* [*OF t s* ‹$t \neq \{\}$› ‹$s \neq \{\}$›]
by (*metis dist-commute setdist-sym*)

lemma *setdist-eq-0I*: $[\![x \in s;\ x \in t]\!] \Longrightarrow$ *setdist s t* $= 0$
by (*metis antisym dist-self setdist-le-dist setdist-pos-le*)

lemma *setdist-eq-0-compact-closed*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *s*: *compact s* **and** *t*: *closed t*
shows *setdist s t* $= 0 \longleftrightarrow s = \{\} \vee t = \{\} \vee s \cap t \neq \{\}$
apply (*cases* $s = \{\} \vee t = \{\}$, *force*)
using *setdist-compact-closed* [*OF s t*]
apply (*force intro*: *setdist-eq-0I*)
done

corollary *setdist-gt-0-compact-closed*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *s*: *compact s* **and** *t*: *closed t*
shows *setdist s t* $> 0 \longleftrightarrow (s \neq \{\} \wedge t \neq \{\} \wedge s \cap t = \{\})$
using *setdist-pos-le* [*of s t*] *setdist-eq-0-compact-closed* [*OF assms*]
by *linarith*

lemma *setdist-eq-0-closed-compact*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *s*: *closed s* **and** *t*: *compact t*
shows *setdist s t* $= 0 \longleftrightarrow s = \{\} \vee t = \{\} \vee s \cap t \neq \{\}$
using *setdist-eq-0-compact-closed* [*OF t s*]
by (*metis Int-commute setdist-sym*)

lemma *setdist-eq-0-bounded*:
fixes *s* :: *'a*::*euclidean-space set*
assumes *bounded s* $\vee$ *bounded t*
shows *setdist s t* $= 0 \longleftrightarrow s = \{\} \vee t = \{\} \vee$ *closure s* $\cap$ *closure t* $\neq \{\}$
apply (*cases* $s = \{\} \vee t = \{\}$, *force*)
using *setdist-eq-0-compact-closed* [*of closure s closure t*]
setdist-eq-0-closed-compact [*of closure s closure t*] *assms*
apply (*force simp add*: *bounded-closure compact-eq-bounded-closed*)
done

lemma *setdist-unique*:
$[\![a \in s;\ b \in t;\ \bigwedge x\ y.\ x \in s \wedge y \in t \Longrightarrow$ *dist a b* $\leq$ *dist x y*$]\!]$
$\Longrightarrow$ *setdist s t* $=$ *dist a b*
by (*force simp add*: *setdist-le-dist le-setdist-iff intro*: *antisym*)

lemma *setdist-closest-point*:
$[\![$*closed s*; $s \neq \{\}]\!] \Longrightarrow$ *setdist* $\{a\}$ *s* $=$ *dist a* (*closest-point s a*)
apply (*rule setdist-unique*)
using *closest-point-le*
apply (*auto simp*: *closest-point-in-set*)
done

lemma *setdist-eq-0-sing-1* [*simp*]:

fixes s :: *'a::euclidean-space set*
shows *setdist* $\{x\}$ $s = 0 \longleftrightarrow s = \{\} \vee x \in$ *closure* s
by (*auto simp*: *setdist-eq-0-bounded*)

lemma *setdist-eq-0-sing-2* [*simp*]:
fixes s :: *'a::euclidean-space set*
shows *setdist* s $\{x\} = 0 \longleftrightarrow s = \{\} \vee x \in$ *closure* s
by (*auto simp*: *setdist-eq-0-bounded*)

lemma *setdist-sing-in-set*:
fixes s :: *'a::euclidean-space set*
shows $x \in s \Longrightarrow$ *setdist* $\{x\}$ $s = 0$
using *closure-subset* **by** *force*

lemma *setdist-le-sing*: $x \in s$ ==> *setdist* s $t \leq$ *setdist* $\{x\}$ t
using *setdist-subset-left* **by** *auto*

end

20 Results connected with topological dimension.

theory *Brouwer-Fixpoint*
imports *Convex-Euclidean-Space*
begin

lemma *bij-betw-singleton-eq*:
assumes f: *bij-betw* f A B **and** g: *bij-betw* g A B **and** a: $a \in A$
assumes *eq*: $(\bigwedge x.\ x \in A \Longrightarrow x \neq a \Longrightarrow f\ x = g\ x)$
shows $f\ a = g\ a$
proof −
have f ‘ $(A - \{a\}) = g$ ‘ $(A - \{a\})$
by (*intro image-cong*) (*simp-all add*: *eq*)
then have $B - \{f\ a\} = B - \{g\ a\}$
using *f g a* **by** (*auto simp*: *bij-betw-def inj-on-image-set-diff set-eq-iff Diff-subset*)
moreover have $f\ a \in B$ $g\ a \in B$
using *f g a* **by** (*auto simp*: *bij-betw-def*)
ultimately show *?thesis*
by *auto*
qed

lemma *swap-image*:
Fun.swap i j f ‘ $A =$ (*if* $i \in A$ *then* (*if* $j \in A$ *then* f ‘ A *else* f ‘ $((A - \{i\}) \cup \{j\})$)
else (*if* $j \in A$ *then* f ‘ $((A - \{j\}) \cup \{i\})$ *else* f ‘ A))
apply (*auto simp*: *Fun.swap-def image-iff*)
apply *metis*
apply (*metis member-remove remove-def*)
apply (*metis member-remove remove-def*)

done

lemma *swap-apply1*: *Fun.swap x y f x = f y*
by (*simp add*: *Fun.swap-def*)

lemma *swap-apply2*: *Fun.swap x y f y = f x*
by (*simp add*: *Fun.swap-def*)

lemma *lessThan-empty-iff*: $\{..< n::nat\} = \{\} \longleftrightarrow n = 0$
by *auto*

lemma *Zero-notin-Suc*: $0 \notin Suc\ `\ A$
by *auto*

lemma *atMost-Suc-eq-insert-0*: $\{..\ Suc\ n\} = insert\ 0\ (Suc\ `\ \{..\ n\})$
apply *auto*
apply (*case-tac x*)
apply *auto*
done

lemma *setsum-union-disjoint'*:
assumes *finite A*
and *finite B*
and $A \cap B = \{\}$
and $A \cup B = C$
shows *setsum g C = setsum g A + setsum g B*
using *setsum.union-disjoint*[*OF assms*(*1*−*3*)] **and** *assms*(*4*) **by** *auto*

lemma *pointwise-minimal-pointwise-maximal*:
fixes *s* :: (*nat* ⇒ *nat*) *set*
assumes *finite s*
and $s \neq \{\}$
and $\forall x{\in}s.\ \forall y{\in}s.\ x \leq y \vee y \leq x$
shows $\exists a{\in}s.\ \forall x{\in}s.\ a \leq x$
and $\exists a{\in}s.\ \forall x{\in}s.\ x \leq a$
using *assms*
proof (*induct s rule*: *finite-ne-induct*)
case (*insert b s*)
assume ∗: $\forall x{\in}insert\ b\ s.\ \forall y{\in}insert\ b\ s.\ x \leq y \vee y \leq x$
moreover then obtain *u l* **where** $l \in s\ \forall b{\in}s.\ l \leq b\ u \in s\ \forall b{\in}s.\ b \leq u$
using *insert* **by** *auto*
ultimately show $\exists a{\in}insert\ b\ s.\ \forall x{\in}insert\ b\ s.\ a \leq x\ \exists a{\in}insert\ b\ s.\ \forall x{\in}insert\ b\ s.\ x \leq a$
using ∗[*rule-format*, *of b u*] ∗[*rule-format*, *of b l*] **by** (*metis insert-iff order.trans*)+
qed *auto*

lemma *brouwer-compactness-lemma*:
fixes *f* :: *'a*::*metric-space* ⇒ *'b*::*real-normed-vector*

assumes *compact s*
and *continuous-on s f*
and $\neg$ ($\exists$ *x*$\in$*s*. *f x* = *0*)
obtains *d* **where** *0* < *d* **and** $\forall$ *x*$\in$*s*. *d* $\leq$ *norm* (*f x*)
proof (*cases s* = {})
case *True*
show *thesis*
by (*rule that* [*of 1*]) (*auto simp*: *True*)
next
case *False*
have *continuous-on s* (*norm* $\circ$ *f*)
by (*rule continuous-intros continuous-on-norm assms*(*2*))+
with *False* **obtain** *x* **where** *x*: *x* $\in$ *s* $\forall$ *y*$\in$*s*. (*norm* $\circ$ *f*) *x* $\leq$ (*norm* $\circ$ *f*) *y*
using *continuous-attains-inf*[*OF assms*(*1*), *of norm* $\circ$ *f*]
unfolding *o-def*
by *auto*
have (*norm* $\circ$ *f*) *x* > *0*
using *assms*(*3*) **and** *x*(*1*)
by *auto*
then show *?thesis*
by (*rule that*) (*insert x*(*2*), *auto simp*: *o-def*)
qed

lemma *kuhn-labelling-lemma*:
fixes *P Q* :: *'a*::*euclidean-space* $\Rightarrow$ *bool*
assumes $\forall$ *x*. *P x* $\longrightarrow$ *P* (*f x*)
and $\forall$ *x*. *P x* $\longrightarrow$ ($\forall$ *i*$\in$*Basis*. *Q i* $\longrightarrow$ *0* $\leq$ *x*$\cdot$*i* $\wedge$ *x*$\cdot$*i* $\leq$ *1*)
shows $\exists$ *l*. ($\forall$ *x*.$\forall$ *i*$\in$*Basis*. *l x i* $\leq$ (*1*::*nat*)) $\wedge$
($\forall$ *x*.$\forall$ *i*$\in$*Basis*. *P x* $\wedge$ *Q i* $\wedge$ (*x*$\cdot$*i* = *0*) $\longrightarrow$ (*l x i* = *0*)) $\wedge$
($\forall$ *x*.$\forall$ *i*$\in$*Basis*. *P x* $\wedge$ *Q i* $\wedge$ (*x*$\cdot$*i* = *1*) $\longrightarrow$ (*l x i* = *1*)) $\wedge$
($\forall$ *x*.$\forall$ *i*$\in$*Basis*. *P x* $\wedge$ *Q i* $\wedge$ (*l x i* = *0*) $\longrightarrow$ *x*$\cdot$*i* $\leq$ *f x*$\cdot$*i*) $\wedge$
($\forall$ *x*.$\forall$ *i*$\in$*Basis*. *P x* $\wedge$ *Q i* $\wedge$ (*l x i* = *1*) $\longrightarrow$ *f x*$\cdot$*i* $\leq$ *x*$\cdot$*i*)
proof −
{ fix *x i*
let *?R* = λ*y*. (*P x* $\wedge$ *Q i* $\wedge$ *x* $\cdot$ *i* = *0* $\longrightarrow$ *y* = (*0*::*nat*)) $\wedge$
(*P x* $\wedge$ *Q i* $\wedge$ *x* $\cdot$ *i* = *1* $\longrightarrow$ *y* = *1*) $\wedge$
(*P x* $\wedge$ *Q i* $\wedge$ *y* = *0* $\longrightarrow$ *x* $\cdot$ *i* $\leq$ *f x* $\cdot$ *i*) $\wedge$
(*P x* $\wedge$ *Q i* $\wedge$ *y* = *1* $\longrightarrow$ *f x* $\cdot$ *i* $\leq$ *x* $\cdot$ *i*)
{ assume *P x Q i i* $\in$ *Basis* **with** *assms* **have** *0* $\leq$ *f x* $\cdot$ *i* $\wedge$ *f x* $\cdot$ *i* $\leq$ *1* **by** *auto* **}**
then have *i* $\in$ *Basis* $\Longrightarrow$ *?R 0* $\vee$ *?R 1* **by** *auto* **}**
then show *?thesis*
unfolding *all-conj-distrib*[*symmetric*] *Ball-def*
by (*subst choice-iff*[*symmetric*])+ *blast*
qed

20.1 The key "counting" observation, somewhat abstracted.

lemma *kuhn-counting-lemma*:

fixes *bnd compo compo' face S F*
defines *nF s == card {f∈F. face f s ∧ compo' f}*
assumes [*simp*, *intro*]: *finite F* — faces **and** [*simp*, *intro*]: *finite S* — simplices
and $\bigwedge f.\ f \in F \Longrightarrow bnd\ f \Longrightarrow card\ \{s{\in}S.\ face\ f\ s\} = 1$
and $\bigwedge f.\ f \in F \Longrightarrow \neg\ bnd\ f \Longrightarrow card\ \{s{\in}S.\ face\ f\ s\} = 2$
and $\bigwedge s.\ s \in S \Longrightarrow compo\ s \Longrightarrow nF\ s = 1$
and $\bigwedge s.\ s \in S \Longrightarrow \neg\ compo\ s \Longrightarrow nF\ s = 0 \vee nF\ s = 2$
and *odd* (*card* {*f*∈*F*. *compo' f* ∧ *bnd f*})
shows *odd* (*card* {*s*∈*S*. *compo s*})
proof −
have $(\sum s \mid s \in S \wedge \neg\ compo\ s.\ nF\ s) + (\sum s \mid s \in S \wedge compo\ s.\ nF\ s) = (\sum s{\in}S.\ nF\ s)$
by (*subst setsum.union-disjoint*[*symmetric*]) (*auto intro*!: *setsum.cong*)
also have ... = $(\sum s{\in}S.\ card\ \{f \in \{f{\in}F.\ compo'\ f \wedge bnd\ f\}.\ face\ f\ s\}) +$
$(\sum s{\in}S.\ card\ \{f \in \{f{\in}F.\ compo'\ f \wedge \neg\ bnd\ f\}.\ face\ f\ s\})$
unfolding *setsum.distrib*[*symmetric*]
by (*subst card-Un-disjoint*[*symmetric*])
(*auto simp*: *nF-def intro*!: *setsum.cong arg-cong*[**where** *f*=*card*])
also have ... = *1* ∗ *card* {*f*∈*F*. *compo' f* ∧ *bnd f*} + *2* ∗ *card* {*f*∈*F*. *compo' f* ∧ ¬ *bnd f*}
using *assms*(*4*,*5*) **by** (*fastforce intro*!: *arg-cong2*[**where** *f*=*op +*] *setsum-multicount*)
finally have *odd* $((\sum s \mid s \in S \wedge \neg\ compo\ s.\ nF\ s) + card\ \{s{\in}S.\ compo\ s\})$
using *assms*(*6*,*8*) **by** *simp*
moreover have $(\sum s \mid s \in S \wedge \neg\ compo\ s.\ nF\ s) =$
$(\sum s \mid s \in S \wedge \neg\ compo\ s \wedge nF\ s = 0.\ nF\ s) + (\sum s \mid s \in S \wedge \neg\ compo\ s \wedge nF\ s = 2.\ nF\ s)$
using *assms*(*7*) **by** (*subst setsum.union-disjoint*[*symmetric*]) (*fastforce intro*!: *setsum.cong*)+
ultimately show *?thesis*
by *auto*
qed

20.2 The odd/even result for faces of complete vertices, generalized.

lemma *kuhn-complete-lemma*:
assumes [*simp*]: *finite simplices*
and *face*: $\bigwedge f\ s.\ face\ f\ s \longleftrightarrow (\exists\, a{\in}s.\ f = s - \{a\})$
and *card-s*[*simp*]: $\bigwedge s.\ s \in simplices \Longrightarrow card\ s = n + 2$
and *rl-bd*: $\bigwedge s.\ s \in simplices \Longrightarrow rl\ `\ s \subseteq \{..Suc\ n\}$
and *bnd*: $\bigwedge f\ s.\ s \in simplices \Longrightarrow face\ f\ s \Longrightarrow bnd\ f \Longrightarrow card\ \{s{\in}simplices.\ face\ f\ s\} = 1$
and *nbnd*: $\bigwedge f\ s.\ s \in simplices \Longrightarrow face\ f\ s \Longrightarrow \neg\ bnd\ f \Longrightarrow card\ \{s{\in}simplices.\ face\ f\ s\} = 2$
and *odd-card*: *odd* (*card* {*f*. (∃ *s*∈*simplices*. *face f s*) ∧ *rl* ` *f* = {..*n*} ∧ *bnd f*})
shows *odd* (*card* {*s*∈*simplices*. (*rl* ` *s* = {..*Suc n*})})
proof (*rule kuhn-counting-lemma*)
have *finite-s*[*simp*]: $\bigwedge s.\ s \in simplices \Longrightarrow finite\ s$
by (*metis add-is-0 zero-neq-numeral card-infinite assms*(*3*))

let *?F* = {*f*. ∃ *s*∈*simplices*. *face f s*}
have *F-eq*: *?F* = (⋃ *s*∈*simplices*. ⋃ *a*∈*s*. {*s* − {*a*}})
by (*auto simp*: *face*)
show *finite ?F*
using ‹*finite simplices*› **unfolding** *F-eq* **by** *auto*

show *card* {*s* ∈ *simplices*. *face f s*} = *1* **if** *f* ∈ *?F bnd f* **for** *f*
using *bnd that* **by** *auto*

show *card* {*s* ∈ *simplices*. *face f s*} = *2* **if** *f* ∈ *?F* ¬ *bnd f* **for** *f*
using *nbnd that* **by** *auto*

show *odd* (*card* {*f* ∈ {*f*. ∃ *s*∈*simplices*. *face f s*}. *rl* ‘ *f* = {..*n*} ∧ *bnd f*})
using *odd-card* **by** *simp*

fix *s* **assume** *s*[*simp*]: *s* ∈ *simplices*
let *?S* = {*f* ∈ {*f*. ∃ *s*∈*simplices*. *face f s*}. *face f s* ∧ *rl* ‘ *f* = {..*n*}}
have *?S* = (λ*a*. *s* − {*a*}) ‘ {*a*∈*s*. *rl* ‘ (*s* − {*a*}) = {..*n*}}
using *s* **by** (*fastforce simp*: *face*)
then have *card-S*: *card ?S* = *card* {*a*∈*s*. *rl* ‘ (*s* − {*a*}) = {..*n*}}
by (*auto intro*!: *card-image inj-onI*)

{ assume *rl*: *rl* ‘ *s* = {..*Suc n*}
then have *inj-rl*: *inj-on rl s*
by (*intro eq-card-imp-inj-on*) *auto*
moreover obtain *a* **where** *rl a* = *Suc n a* ∈ *s*
by (*metis atMost-iff image-iff le-Suc-eq rl*)
ultimately have *n*: {..*n*} = *rl* ‘ (*s* − {*a*})
by (*auto simp add*: *inj-on-image-set-diff Diff-subset rl*)
have {*a*∈*s*. *rl* ‘ (*s* − {*a*}) = {..*n*}} = {*a*}
using *inj-rl* ‹*a* ∈ *s*› **by** (*auto simp add*: *n inj-on-image-eq-iff*[*OF inj-rl*] *Diff-subset*)
then show *card ?S* = *1*
unfolding *card-S* **by** *simp* **}**

{ assume *rl*: *rl* ‘ *s* ≠ {..*Suc n*}
show *card ?S* = *0* ∨ *card ?S* = *2*
proof *cases*
assume ∗: {..*n*} ⊆ *rl* ‘ *s*
with *rl rl-bd*[*OF s*] **have** *rl-s*: *rl* ‘ *s* = {..*n*}
by (*auto simp add*: *atMost-Suc subset-insert-iff split*: *split-if-asm*)
then have ¬ *inj-on rl s*
by (*intro pigeonhole*) *simp*
then obtain *a b* **where** *ab*: *a* ∈ *s b* ∈ *s rl a* = *rl b a* ≠ *b*
by (*auto simp*: *inj-on-def*)
then have *eq*: *rl* ‘ (*s* − {*a*}) = *rl* ‘ *s*
by *auto*
with *ab* **have** *inj*: *inj-on rl* (*s* − {*a*})

```
      by (intro eq-card-imp-inj-on) (auto simp add: rl-s card-Diff-singleton-if)

    { fix x assume x ∈ s x ∉ {a, b}
      then have rl ‘ s − {rl x} = rl ‘ ((s − {a}) − {x})
        by (auto simp: eq Diff-subset inj-on-image-set-diff[OF inj])
      also have ... = rl ‘ (s − {x})
        using ab ‹x ∉ {a, b}› by auto
      also assume ... = rl ‘ s
      finally have False
        using ‹x∈s› by auto }
    moreover
    { fix x assume x ∈ {a, b} with ab have x ∈ s ∧ rl ‘ (s − {x}) = rl ‘ s
        by (simp add: set-eq-iff image-iff Bex-def) metis }
    ultimately have {a∈s. rl ‘ (s − {a}) = {..n}} = {a, b}
      unfolding rl-s[symmetric] by fastforce
    with ‹a ≠ b› show card ?S = 0 ∨ card ?S = 2
      unfolding card-S by simp
  next
    assume ¬ {..n} ⊆ rl ‘ s
    then have ⋀x. rl ‘ (s − {x}) ≠ {..n}
      by auto
    then show card ?S = 0 ∨ card ?S = 2
      unfolding card-S by simp
  qed }
qed fact

locale kuhn-simplex =
  fixes p n and base upd and s :: (nat ⇒ nat) set
  assumes base: base ∈ {..< n} → {..< p}
  assumes base-out: ⋀i. n ≤ i ⟹ base i = p
  assumes upd: bij-betw upd {..< n} {..< n}
  assumes s-pre: s = (λi j. if j ∈ upd‘{..< i} then Suc (base j) else base j) ‘ {..
n}
begin

definition enum i j = (if j ∈ upd‘{..< i} then Suc (base j) else base j)

lemma s-eq: s = enum ‘ {.. n}
  unfolding s-pre enum-def[abs-def] ..

lemma upd-space: i < n ⟹ upd i < n
  using upd by (auto dest!: bij-betwE)

lemma s-space: s ⊆ {..< n} → {.. p}
proof −
  { fix i assume i ≤ n then have enum i ∈ {..< n} → {.. p}
    proof (induct i)
      case 0 then show ?case
        using base by (auto simp: Pi-iff less-imp-le enum-def)
```

```
    next
      case (Suc i) with base show ?case
        by (auto simp: Pi-iff Suc-le-eq less-imp-le enum-def intro: upd-space)
    qed }
  then show ?thesis
    by (auto simp: s-eq)
qed

lemma inj-upd: inj-on upd {..< n}
  using upd by (simp add: bij-betw-def)

lemma inj-enum: inj-on enum {.. n}
proof −
  { fix x y :: nat assume x ≠ y x ≤ n y ≤ n
    with upd have upd ‘ {..< x} ≠ upd ‘ {..< y}
      by (subst inj-on-image-eq-iff[where C={..< n}]) (auto simp: bij-betw-def)
    then have enum x ≠ enum y
      by (auto simp add: enum-def fun-eq-iff) }
  then show ?thesis
    by (auto simp: inj-on-def)
qed

lemma enum-0: enum 0 = base
  by (simp add: enum-def[abs-def])

lemma base-in-s: base ∈ s
  unfolding s-eq by (subst enum-0[symmetric]) auto

lemma enum-in: i ≤ n ⟹ enum i ∈ s
  unfolding s-eq by auto

lemma one-step:
  assumes a: a ∈ s j < n
  assumes ∗: ⋀a'. a' ∈ s ⟹ a' ≠ a ⟹ a' j = p'
  shows a j ≠ p'
proof
  assume a j = p'
  with ∗ a have ⋀a'. a' ∈ s ⟹ a' j = p'
    by auto
  then have ⋀i. i ≤ n ⟹ enum i j = p'
    unfolding s-eq by auto
  from this[of 0] this[of n] have j ∉ upd ‘ {..< n}
    by (auto simp: enum-def fun-eq-iff split: split-if-asm)
  with upd ‹j < n› show False
    by (auto simp: bij-betw-def)
qed

lemma upd-inj: i < n ⟹ j < n ⟹ upd i = upd j ⟷ i = j
  using upd by (auto simp: bij-betw-def inj-on-eq-iff)
```

lemma *upd-surj*: *upd* ‘ $\{..< n\} = \{..< n\}$
using *upd* **by** (*auto simp*: *bij-betw-def*)

lemma *in-upd-image*: $A \subseteq \{..< n\} \Longrightarrow i < n \Longrightarrow upd\ i \in upd\ ‘\ A \longleftrightarrow i \in A$
using *inj-on-image-mem-iff*[*of upd* $\{..< n\}$] *upd*
by (*auto simp*: *bij-betw-def*)

lemma *enum-inj*: $i \leq n \Longrightarrow j \leq n \Longrightarrow enum\ i = enum\ j \longleftrightarrow i = j$
using *inj-enum* **by** (*auto simp*: *inj-on-eq-iff*)

lemma *in-enum-image*: $A \subseteq \{..\ n\} \Longrightarrow i \leq n \Longrightarrow enum\ i \in enum\ ‘\ A \longleftrightarrow i \in A$
using *inj-on-image-mem-iff*[*OF inj-enum*] **by** *auto*

lemma *enum-mono*: $i \leq n \Longrightarrow j \leq n \Longrightarrow enum\ i \leq enum\ j \longleftrightarrow i \leq j$
by (*auto simp*: *enum-def le-fun-def in-upd-image Ball-def*[*symmetric*])

lemma *enum-strict-mono*: $i \leq n \Longrightarrow j \leq n \Longrightarrow enum\ i < enum\ j \longleftrightarrow i < j$
using *enum-mono*[*of i j*] *enum-inj*[*of i j*] **by** (*auto simp add*: *le-less*)

lemma *chain*: $a \in s \Longrightarrow b \in s \Longrightarrow a \leq b \vee b \leq a$
by (*auto simp*: *s-eq enum-mono*)

lemma *less*: $a \in s \Longrightarrow b \in s \Longrightarrow a\ i < b\ i \Longrightarrow a < b$
using *chain*[*of a b*] **by** (*auto simp*: *less-fun-def le-fun-def not-le*[*symmetric*])

lemma *enum-0-bot*: $a \in s \Longrightarrow a = enum\ 0 \longleftrightarrow (\forall\, a' \in s.\ a \leq a')$
unfolding *s-eq* **by** (*auto simp*: *enum-mono Ball-def*)

lemma *enum-n-top*: $a \in s \Longrightarrow a = enum\ n \longleftrightarrow (\forall\, a' \in s.\ a' \leq a)$
unfolding *s-eq* **by** (*auto simp*: *enum-mono Ball-def*)

lemma *enum-Suc*: $i < n \Longrightarrow enum\ (Suc\ i) = (enum\ i)(upd\ i := Suc\ (enum\ i\ (upd\ i)))$
by (*auto simp*: *fun-eq-iff enum-def upd-inj*)

lemma *enum-eq-p*: $i \leq n \Longrightarrow n \leq j \Longrightarrow enum\ i\ j = p$
by (*induct i*) (*auto simp*: *enum-Suc enum-0 base-out upd-space not-less*[*symmetric*])

lemma *out-eq-p*: $a \in s \Longrightarrow n \leq j \Longrightarrow a\ j = p$
unfolding *s-eq* **by** (*auto simp add*: *enum-eq-p*)

lemma *s-le-p*: $a \in s \Longrightarrow a\ j \leq p$
using *out-eq-p*[*of a j*] *s-space* **by** (*cases* $j < n$) *auto*

lemma *le-Suc-base*: $a \in s \Longrightarrow a\ j \leq Suc\ (base\ j)$
unfolding *s-eq* **by** (*auto simp*: *enum-def*)

lemma *base-le*: $a \in s \Longrightarrow base\ j \leq a\ j$

unfolding *s-eq* **by** (*auto simp*: *enum-def*)

lemma *enum-le-p*: $i \leq n \Longrightarrow j < n \Longrightarrow enum\ i\ j \leq p$
using *enum-in*[*of i*] *s-space* **by** *auto*

lemma *enum-less*: $a \in s \Longrightarrow i < n \Longrightarrow enum\ i < a \longleftrightarrow enum\ (Suc\ i) \leq a$
unfolding *s-eq* **by** (*auto simp*: *enum-strict-mono enum-mono*)

lemma *ksimplex-0*:
$n = 0 \Longrightarrow s = \{(\lambda x.\ p)\}$
using *s-eq enum-def base-out* **by** *auto*

lemma *replace-0*:
assumes $j < n\ a \in s$ **and** p: $\forall x{\in}s - \{a\}.\ x\ j = 0$ **and** $x \in s$
shows $x \leq a$
proof *cases*
assume $x \neq a$
have $a\ j \neq 0$
using *assms* **by** (*intro one-step*[**where** $a{=}a$]) *auto*
with *less*[*OF* ‹$x{\in}s$› ‹$a{\in}s$›, *of j*] *p*[*rule-format*, *of x*] ‹$x \in s$› ‹$x \neq a$›
show *?thesis*
by *auto*
qed *simp*

lemma *replace-1*:
assumes $j < n\ a \in s$ **and** p: $\forall x{\in}s - \{a\}.\ x\ j = p$ **and** $x \in s$
shows $a \leq x$
proof *cases*
assume $x \neq a$
have $a\ j \neq p$
using *assms* **by** (*intro one-step*[**where** $a{=}a$]) *auto*
with *enum-le-p*[*of - j*] ‹$j < n$› ‹$a{\in}s$›
have $a\ j < p$
by (*auto simp*: *less-le s-eq*)
with *less*[*OF* ‹$a{\in}s$› ‹$x{\in}s$›, *of j*] *p*[*rule-format*, *of x*] ‹$x \in s$› ‹$x \neq a$›
show *?thesis*
by *auto*
qed *simp*

end

locale *kuhn-simplex-pair* = *s*: *kuhn-simplex p n b-s u-s s* + *t*: *kuhn-simplex p n b-t u-t t*
for *p n b-s u-s s b-t u-t t*
begin

lemma *enum-eq*:
assumes *l*: $i \leq l\ l \leq j$ **and** $j + d \leq n$
assumes *eq*: *s.enum* ‘ $\{i\ ..\ j\}$ = *t.enum* ‘ $\{i + d\ ..\ j + d\}$

```
  shows s.enum l = t.enum (l + d)
using l proof (induct l rule: dec-induct)
  case base
  then have s: s.enum i ∈ t.enum ‘ {i + d .. j + d} and t: t.enum (i + d) ∈
s.enum ‘ {i .. j}
    using eq by auto
  from t ‹i ≤ j› ‹j + d ≤ n› have s.enum i ≤ t.enum (i + d)
    by (auto simp: s.enum-mono)
  moreover from s ‹i ≤ j› ‹j + d ≤ n› have t.enum (i + d) ≤ s.enum i
    by (auto simp: t.enum-mono)
  ultimately show ?case
    by auto
next
  case (step l)
  moreover from step.prems ‹j + d ≤ n› have
      s.enum l < s.enum (Suc l)
      t.enum (l + d) < t.enum (Suc l + d)
    by (simp-all add: s.enum-strict-mono t.enum-strict-mono)
  moreover have
      s.enum (Suc l) ∈ t.enum ‘ {i + d .. j + d}
      t.enum (Suc l + d) ∈ s.enum ‘ {i .. j}
    using step ‹j + d ≤ n› eq by (auto simp: s.enum-inj t.enum-inj)
  ultimately have s.enum (Suc l) = t.enum (Suc (l + d))
    using ‹j + d ≤ n›
    by (intro antisym s.enum-less[THEN iffD1] t.enum-less[THEN iffD1])
       (auto intro!: s.enum-in t.enum-in)
  then show ?case by simp
qed

lemma ksimplex-eq-bot:
  assumes a: a ∈ s ⋀a'. a' ∈ s ⟹ a ≤ a'
  assumes b: b ∈ t ⋀b'. b' ∈ t ⟹ b ≤ b'
  assumes eq: s − {a} = t − {b}
  shows s = t
proof cases
  assume n = 0 with s.ksimplex-0 t.ksimplex-0 show ?thesis by simp
next
  assume n ≠ 0
  have s.enum 0 = (s.enum (Suc 0)) (u-s 0 := s.enum (Suc 0) (u-s 0) − 1)
       t.enum 0 = (t.enum (Suc 0)) (u-t 0 := t.enum (Suc 0) (u-t 0) − 1)
    using ‹n ≠ 0› by (simp-all add: s.enum-Suc t.enum-Suc)
  moreover have e0: a = s.enum 0 b = t.enum 0
    using a b by (simp-all add: s.enum-0-bot t.enum-0-bot)
  moreover
  { fix j assume 0 < j j ≤ n
    moreover have s − {a} = s.enum ‘ {Suc 0 .. n} t − {b} = t.enum ‘ {Suc 0
.. n}
      unfolding s.s-eq t.s-eq e0 by (auto simp: s.enum-inj t.enum-inj)
    ultimately have s.enum j = t.enum j
```

```
    using enum-eq[of 1 j n 0] eq by auto }
  note enum-eq = this
  then have s.enum (Suc 0) = t.enum (Suc 0)
    using ‹n ≠ 0› by auto
  moreover
  { fix j assume Suc j < n
    with enum-eq[of Suc j] enum-eq[of Suc (Suc j)]
    have u-s (Suc j) = u-t (Suc j)
      using s.enum-Suc[of Suc j] t.enum-Suc[of Suc j]
      by (auto simp: fun-eq-iff split: split-if-asm) }
  then have ⋀j. 0 < j ⟹ j < n ⟹ u-s j = u-t j
    by (auto simp: gr0-conv-Suc)
  with ‹n ≠ 0› have u-t 0 = u-s 0
    by (intro bij-betw-singleton-eq[OF t.upd s.upd, of 0]) auto
  ultimately have a = b
    by simp
  with assms show s = t
    by auto
qed

lemma ksimplex-eq-top:
  assumes a: a ∈ s ⋀a'. a' ∈ s ⟹ a' ≤ a
  assumes b: b ∈ t ⋀b'. b' ∈ t ⟹ b' ≤ b
  assumes eq: s - {a} = t - {b}
  shows s = t
proof (cases n)
  assume n = 0 with s.ksimplex-0 t.ksimplex-0 show ?thesis by simp
next
  case (Suc n')
  have s.enum n = (s.enum n') (u-s n' := Suc (s.enum n' (u-s n')))
      t.enum n = (t.enum n') (u-t n' := Suc (t.enum n' (u-t n')))
    using Suc by (simp-all add: s.enum-Suc t.enum-Suc)
  moreover have en: a = s.enum n b = t.enum n
    using a b by (simp-all add: s.enum-n-top t.enum-n-top)
  moreover
  { fix j assume j < n
    moreover have s - {a} = s.enum ‘ {0 .. n'} t - {b} = t.enum ‘ {0 .. n'}
      unfolding s.s-eq t.s-eq en by (auto simp: s.enum-inj t.enum-inj Suc)
    ultimately have s.enum j = t.enum j
      using enum-eq[of 0 j n' 0] eq Suc by auto }
  note enum-eq = this
  then have s.enum n' = t.enum n'
    using Suc by auto
  moreover
  { fix j assume j < n'
    with enum-eq[of j] enum-eq[of Suc j]
    have u-s j = u-t j
      using s.enum-Suc[of j] t.enum-Suc[of j]
      by (auto simp: Suc fun-eq-iff split: split-if-asm) }
```

```
  then have ⋀j. j < n' ⟹ u-s j = u-t j
    by (auto simp: gr0-conv-Suc)
  then have u-t n' = u-s n'
    by (intro bij-betw-singleton-eq[OF t.upd s.upd, of n']) (auto simp: Suc)
  ultimately have a = b
    by simp
  with assms show s = t
    by auto
qed

end

inductive ksimplex for p n :: nat where
  ksimplex: kuhn-simplex p n base upd s ⟹ ksimplex p n s

lemma finite-ksimplexes: finite {s. ksimplex p n s}
proof (rule finite-subset)
  { fix a s assume ksimplex p n s a ∈ s
    then obtain b u where kuhn-simplex p n b u s by (auto elim: ksimplex.cases)
    then interpret kuhn-simplex p n b u s .
    from s-space ‹a ∈ s› out-eq-p[OF ‹a ∈ s›]
    have a ∈ (λf x. if n ≤ x then p else f x) ` ({..< n} →E {.. p})
      by (auto simp: image-iff subset-eq Pi-iff split: split-if-asm
            intro!: bexI[of - restrict a {..< n}]) }
  then show {s. ksimplex p n s} ⊆ Pow ((λf x. if n ≤ x then p else f x) ` ({..<
n} →E {.. p}))
    by auto
qed (simp add: finite-PiE)

lemma ksimplex-card:
  assumes ksimplex p n s shows card s = Suc n
using assms proof cases
  case (ksimplex u b)
  then interpret kuhn-simplex p n u b s .
  show ?thesis
    by (simp add: card-image s-eq inj-enum)
qed

lemma simplex-top-face:
  assumes 0 < p ∀x∈s'. x n = p
  shows ksimplex p n s' ⟷ (∃s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ s' = s - {a})
  using assms
proof safe
  fix s a assume ksimplex p (Suc n) s and a: a ∈ s and na: ∀x∈s - {a}. x n =
p
  then show ksimplex p n (s - {a})
  proof cases
    case (ksimplex base upd)
    then interpret kuhn-simplex p Suc n base upd s .
```

```
  have a n < p
    using one-step[of a n p] na ‹a∈s› s-space by (auto simp: less-le)
  then have a = enum 0
    using ‹a ∈ s› na by (subst enum-0-bot) (auto simp: le-less intro!: less[of a -
n])
  then have s-eq: s − {a} = enum ‘ Suc ‘ {.. n}
    using s-eq by (simp add: atMost-Suc-eq-insert-0 insert-ident Zero-notin-Suc
in-enum-image subset-eq)
  then have enum 1 ∈ s − {a}
    by auto
  then have upd 0 = n
    using ‹a n < p› ‹a = enum 0› na[rule-format, of enum 1]
    by (auto simp: fun-eq-iff enum-Suc split: split-if-asm)
  then have bij-betw upd (Suc ‘ {..< n}) {..< n}
    using upd
    by (subst notIn-Un-bij-betw3[where b=0])
      (auto simp: lessThan-Suc[symmetric] lessThan-Suc-eq-insert-0)
  then have bij-betw (upd∘Suc) {..<n} {..<n}
    by (rule bij-betw-trans[rotated]) (auto simp: bij-betw-def)

  have a n = p − 1
    using enum-Suc[of 0] na[rule-format, OF ‹enum 1 ∈ s − {a}›] ‹a = enum
0› by (auto simp: ‹upd 0 = n›)

  show ?thesis
  proof (rule ksimplex.intros, standard)
    show bij-betw (upd∘Suc) {..< n} {..< n} by fact
    show base(n := p) ∈ {..<n} → {..<p} ⋀i. n≤i ⟹ (base(n := p)) i = p
      using base base-out by (auto simp: Pi-iff)

    have ⋀i. Suc ‘ {..< i} = {..< Suc i} − {0}
      by (auto simp: image-iff Ball-def) arith
    then have upd-Suc: ⋀i. i ≤ n ⟹ (upd∘Suc) ‘ {..< i} = upd ‘ {..< Suc i}
− {n}
      using ‹upd 0 = n› upd-inj
    by (auto simp add: image-comp[symmetric] inj-on-image-set-diff[OF inj-upd])
    have n-in-upd: ⋀i. n ∈ upd ‘ {..< Suc i}
      using ‹upd 0 = n› by auto

     def f' ≡ λi j. if j ∈ (upd∘Suc)‘{..< i} then Suc ((base(n := p)) j) else
(base(n := p)) j
    { fix x i assume i[arith]: i ≤ n then have enum (Suc i) x = f' i x
      unfolding f'-def enum-def using ‹a n < p› ‹a = enum 0› ‹upd 0 = n› ‹a
n = p − 1›
      by (simp add: upd-Suc enum-0 n-in-upd) }
    then show s − {a} = f' ‘ {.. n}
      unfolding s-eq image-comp by (intro image-cong) auto
  qed
```

```
  qed
next
  assume ksimplex p n s' and ∗: ∀x∈s'. x n = p
  then show ∃s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ s' = s − {a}
  proof cases
    case (ksimplex base upd)
    then interpret kuhn-simplex p n base upd s' .
    def b ≡ base (n := p − 1)
    def u ≡ λi. case i of 0 ⇒ n | Suc i ⇒ upd i

    have ksimplex p (Suc n) (s' ∪ {b})
    proof (rule ksimplex.intros, standard)
      show b ∈ {..<Suc n} → {..<p}
        using base ‹0 < p› unfolding lessThan-Suc b-def by (auto simp: PiE-iff)
      show ⋀i. Suc n ≤ i ⟹ b i = p
        using base-out by (auto simp: b-def)

      have bij-betw u (Suc ` {..< n} ∪ {0}) ({..<n} ∪ {u 0})
        using upd
          by (intro notIn-Un-bij-betw) (auto simp: u-def bij-betw-def image-comp
comp-def inj-on-def)
      then show bij-betw u {..<Suc n} {..<Suc n}
        by (simp add: u-def lessThan-Suc[symmetric] lessThan-Suc-eq-insert-0)

      def f' ≡ λi j. if j ∈ u`{..< i} then Suc (b j) else b j

      have u-eq: ⋀i. i ≤ n ⟹ u ` {..< Suc i} = upd ` {..< i} ∪ { n }
        by (auto simp: u-def image-iff upd-inj Ball-def split: nat.split) arith

      { fix x have x ≤ n ⟹ n ∉ upd ` {..<x}
          using upd-space by (simp add: image-iff neq-iff) }
      note n-not-upd = this

      have ∗: f' ` {.. Suc n} = f' ` (Suc ` {.. n} ∪ {0})
        unfolding atMost-Suc-eq-insert-0 by simp
      also have ... = (f' ∘ Suc) ` {.. n} ∪ {b}
        by (auto simp: f'-def)
      also have (f' ∘ Suc) ` {.. n} = s'
        using ‹0 < p› base-out[of n]
        unfolding s-eq enum-def[abs-def] f'-def[abs-def] upd-space
        by (intro image-cong) (simp-all add: u-eq b-def fun-eq-iff n-not-upd)
      finally show s' ∪ {b} = f' ` {.. Suc n} ..
    qed
    moreover have b ∉ s'
      using ∗ ‹0 < p› by (auto simp: b-def)
    ultimately show ?thesis by auto
  qed
qed
```

lemma *ksimplex-replace-0*:
assumes *s*: *ksimplex p n s* **and** *a*: $a \in s$
assumes *j*: $j < n$ **and** *p*: $\forall x{\in}s - \{a\}.\ x\ j = 0$
shows *card* $\{s'.\ ksimplex\ p\ n\ s' \wedge (\exists b{\in}s'.\ s' - \{b\} = s - \{a\})\} = 1$
using *s*
proof *cases*
case (*ksimplex b-s u-s*)

{ fix *t b* **assume** *ksimplex p n t*
then obtain *b-t u-t* **where** *kuhn-simplex p n b-t u-t t*
by (*auto elim*: *ksimplex.cases*)
interpret *kuhn-simplex-pair p n b-s u-s s b-t u-t t*
by *intro-locales fact+*

assume *b*: $b \in t$ $t - \{b\} = s - \{a\}$
with *a j p s.replace-0*[*of - a*] *t.replace-0*[*of - b*] **have** $s = t$
by (*intro ksimplex-eq-top*[*of a b*]) *auto* **}**
then have $\{s'.\ ksimplex\ p\ n\ s' \wedge (\exists b{\in}s'.\ s' - \{b\} = s - \{a\})\} = \{s\}$
using *s* ‹$a \in s$› **by** *auto*
then show *?thesis*
by *simp*
qed

lemma *ksimplex-replace-1*:
assumes *s*: *ksimplex p n s* **and** *a*: $a \in s$
assumes *j*: $j < n$ **and** *p*: $\forall x{\in}s - \{a\}.\ x\ j = p$
shows *card* $\{s'.\ ksimplex\ p\ n\ s' \wedge (\exists b{\in}s'.\ s' - \{b\} = s - \{a\})\} = 1$
using *s*
proof *cases*
case (*ksimplex b-s u-s*)

{ fix *t b* **assume** *ksimplex p n t*
then obtain *b-t u-t* **where** *kuhn-simplex p n b-t u-t t*
by (*auto elim*: *ksimplex.cases*)
interpret *kuhn-simplex-pair p n b-s u-s s b-t u-t t*
by *intro-locales fact+*

assume *b*: $b \in t$ $t - \{b\} = s - \{a\}$
with *a j p s.replace-1*[*of - a*] *t.replace-1*[*of - b*] **have** $s = t$
by (*intro ksimplex-eq-bot*[*of a b*]) *auto* **}**
then have $\{s'.\ ksimplex\ p\ n\ s' \wedge (\exists b{\in}s'.\ s' - \{b\} = s - \{a\})\} = \{s\}$
using *s* ‹$a \in s$› **by** *auto*
then show *?thesis*
by *simp*
qed

lemma *card-2-exists*: $card\ s = 2 \longleftrightarrow (\exists x{\in}s.\ \exists y{\in}s.\ x \neq y \wedge (\forall z{\in}s.\ z = x \vee z = y))$
by (*auto simp add*: *card-Suc-eq eval-nat-numeral*)

lemma *ksimplex-replace-2*:
assumes *s*: *ksimplex p n s* **and** $a \in s$ **and** $n \neq 0$
and *lb*: $\forall j{<}n.\ \exists x{\in}s - \{a\}.\ x\ j \neq 0$
and *ub*: $\forall j{<}n.\ \exists x{\in}s - \{a\}.\ x\ j \neq p$
shows *card* $\{s'.\ ksimplex\ p\ n\ s' \wedge (\exists b{\in}s'.\ s' - \{b\} = s - \{a\})\} = 2$
using *s*
proof *cases*
case (*ksimplex base upd*)
then interpret *kuhn-simplex p n base upd s* **.**

from ‹$a \in s$› **obtain** *i* **where** $i \leq n\ a = enum\ i$
unfolding *s-eq* **by** *auto*

from ‹$i \leq n$› **have** $i = 0 \vee i = n \vee (0 < i \wedge i < n)$
by *linarith*
then have $\exists!s'.\ s' \neq s \wedge ksimplex\ p\ n\ s' \wedge (\exists b{\in}s'.\ s - \{a\} = s' - \{b\})$
proof (*elim disjE conjE*)
assume $i = 0$
def *rot* $\equiv \lambda i.\ if\ i + 1 = n\ then\ 0\ else\ i + 1$
let *?upd* $= upd \circ rot$

have *rot*: *bij-betw rot* $\{..< n\}$ $\{..< n\}$
by (*auto simp*: *bij-betw-def inj-on-def image-iff Ball-def rot-def*)
arith+
from *rot upd* **have** *bij-betw ?upd* $\{..<n\}$ $\{..<n\}$
by (*rule bij-betw-trans*)

def $f' \equiv \lambda i\ j.\ if\ j \in$ *?upd*‘$\{..< i\}$ *then Suc* (*enum* (*Suc 0*) *j*) *else enum* (*Suc 0*) *j*

interpret *b*: *kuhn-simplex p n enum* (*Suc 0*) *upd* $\circ$ *rot f'* ‘ $\{..\ n\}$
proof
from ‹$a = enum\ i$› *ub* ‹$n \neq 0$› ‹$i = 0$›
obtain i' **where** $i' \leq n\ enum\ i' \neq enum\ 0\ enum\ i'\ (upd\ 0) \neq p$
unfolding *s-eq* **by** (*auto intro*: *upd-space simp*: *enum-inj*)
then have $enum\ 1 \leq enum\ i'\ enum\ i'\ (upd\ 0) < p$
using *enum-le-p*[*of i' upd 0*] **by** (*auto simp add*: *enum-inj enum-mono upd-space*)
then have $enum\ 1\ (upd\ 0) < p$
by (*auto simp add*: *le-fun-def intro*: *le-less-trans*)
then show $enum\ (Suc\ 0) \in \{..<n\} \to \{..<p\}$
using *base* ‹$n \neq 0$› **by** (*auto simp add*: *enum-0 enum-Suc PiE-iff extensional-def upd-space*)

{ fix *i* **assume** $n \leq i$ **then show** $enum\ (Suc\ 0)\ i = p$
using ‹$n \neq 0$› **by** (*auto simp*: *enum-eq-p*) **}**
show *bij-betw ?upd* $\{..<n\}$ $\{..<n\}$ **by** *fact*
qed (*simp add*: *f'-def*)

have *ks-f'*: *ksimplex p n* (*f'* ‘ {.. *n*})
by *rule unfold-locales*

have *b-enum*: *b.enum* = *f'* **unfolding** *f'-def b.enum-def*[*abs-def*] **..**
with *b.inj-enum* **have** *inj-f'*: *inj-on f'* {.. *n*} **by** *simp*

have [*simp*]: ⋀*j*. *j* < *n* ⟹ *rot* ‘ {..< *j*} = {*0* <..< *Suc j*}
by (*auto simp*: *rot-def image-iff Ball-def*)
arith

{ fix *j* **assume** *j*: *j* < *n*
from *j* ‹*n* ≠ *0*› **have** *f' j* = *enum* (*Suc j*)
by (*auto simp add*: *f'-def enum-def upd-inj in-upd-image image-comp*[*symmetric*] *fun-eq-iff*) **}**
note *f'-eq-enum* = *this*
then have *enum* ‘ *Suc* ‘ {..< *n*} = *f'* ‘ {..< *n*}
by (*force simp*: *enum-inj*)
also have *Suc* ‘ {..< *n*} = {.. *n*} − {*0*}
by (*auto simp*: *image-iff Ball-def*) *arith*
also have {..< *n*} = {.. *n*} − {*n*}
by *auto*
finally have *eq*: *s* − {*a*} = *f'* ‘ {.. *n*} − {*f' n*}
unfolding *s-eq* ‹*a* = *enum i*› ‹*i* = *0*›
by (*simp add*: *Diff-subset inj-on-image-set-diff*[*OF inj-enum*] *inj-on-image-set-diff*[*OF inj-f'*])

have *enum 0* < *f' 0*
using ‹*n* ≠ *0*› **by** (*simp add*: *enum-strict-mono f'-eq-enum*)
also have ... < *f' n*
using ‹*n* ≠ *0*› *b.enum-strict-mono*[*of 0 n*] **unfolding** *b-enum* **by** *simp*
finally have *a* ≠ *f' n*
using ‹*a* = *enum i*› ‹*i* = *0*› **by** *auto*

{ fix *t c* **assume** *ksimplex p n t c* ∈ *t* **and** *eq-sma*: *s* − {*a*} = *t* − {*c*}
obtain *b u* **where** *kuhn-simplex p n b u t*
using ‹*ksimplex p n t*› **by** (*auto elim*: *ksimplex.cases*)
then interpret *t*: *kuhn-simplex p n b u t* **.**

{ fix *x* **assume** *x* ∈ *s x* ≠ *a*
then have *x* (*upd 0*) = *enum* (*Suc 0*) (*upd 0*)
by (*auto simp*: ‹*a* = *enum i*› ‹*i* = *0*› *s-eq enum-def enum-inj*) **}**
then have *eq-upd0*: ∀ *x*∈*t*−{*c*}. *x* (*upd 0*) = *enum* (*Suc 0*) (*upd 0*)
unfolding *eq-sma*[*symmetric*] **by** *auto*
then have *c* (*upd 0*) ≠ *enum* (*Suc 0*) (*upd 0*)
using ‹*n* ≠ *0*› **by** (*intro t.one-step*[*OF* ‹*c*∈*t*›]) (*auto simp*: *upd-space*)
then have *c* (*upd 0*) < *enum* (*Suc 0*) (*upd 0*) ∨ *c* (*upd 0*) > *enum* (*Suc 0*) (*upd 0*)
by *auto*
then have *t* = *s* ∨ *t* = *f'* ‘ {..*n*}

```
    proof (elim disjE conjE)
      assume *: c (upd 0) < enum (Suc 0) (upd 0)
      interpret st: kuhn-simplex-pair p n base upd s b u t ..
      { fix x assume x ∈ t with * ‹c∈t› eq-upd0[rule-format, of x] have c ≤ x
          by (auto simp: le-less intro!: t.less[of - - upd 0]) }
      note top = this
      have s = t
        using ‹a = enum i› ‹i = 0› ‹c ∈ t›
        by (intro st.ksimplex-eq-bot[OF - - - - eq-sma])
           (auto simp: s-eq enum-mono t.s-eq t.enum-mono top)
      then show ?thesis by simp
    next
      assume *: c (upd 0) > enum (Suc 0) (upd 0)
      interpret st: kuhn-simplex-pair p n enum (Suc 0) upd ∘ rot f' ‘ {.. n} b u
t ..
      have eq: f' ‘ {..n} - {f' n} = t - {c}
        using eq-sma eq by simp
      { fix x assume x ∈ t with * ‹c∈t› eq-upd0[rule-format, of x] have x ≤ c
          by (auto simp: le-less intro!: t.less[of - - upd 0]) }
      note top = this
      have f' ‘ {..n} = t
        using ‹a = enum i› ‹i = 0› ‹c ∈ t›
        by (intro st.ksimplex-eq-top[OF - - - - eq])
           (auto simp: b.s-eq b.enum-mono t.s-eq t.enum-mono b-enum[symmetric]
top)
      then show ?thesis by simp
    qed }
  with ks-f' eq ‹a ≠ f' n› ‹n ≠ 0› show ?thesis
    apply (intro ex1I[of - f' ‘ {.. n}])
    apply auto []
    apply metis
    done
next
  assume i = n
  from ‹n ≠ 0› obtain n' where n': n = Suc n'
    by (cases n) auto

  def rot ≡ λi. case i of 0 ⇒ n' | Suc i ⇒ i
  let ?upd = upd ∘ rot

  have rot: bij-betw rot {..< n} {..< n}
      by (auto simp: bij-betw-def inj-on-def image-iff Bex-def rot-def n' split:
nat.splits)
       arith
  from rot upd have bij-betw ?upd {..<n} {..<n}
    by (rule bij-betw-trans)

  def b ≡ base (upd n' := base (upd n') - 1)
  def f' ≡ λi j. if j ∈ ?upd‘{..< i} then Suc (b j) else b j
```

interpret *b*: *kuhn-simplex p n b upd* ∘ *rot f'* ‘ {.. *n*}
proof
{ fix *i* **assume** *n* ≤ *i* **then show** *b i* = *p*
using *base-out*[*of i*] *upd-space*[*of n'*] **by** (*auto simp*: *b-def n'*) **}**
show *b* ∈ {..<*n*} → {..<*p*}
using *base* ‹*n* ≠ *0*› *upd-space*[*of n'*]
by (*auto simp*: *b-def PiE-def Pi-iff Ball-def upd-space extensional-def n'*)

show *bij-betw ?upd* {..<*n*} {..<*n*} **by** *fact*
qed (*simp add*: *f'-def*)
have *f'*: *b.enum* = *f'* **unfolding** *f'-def b.enum-def*[*abs-def*] **..**
have *ks-f'*: *ksimplex p n* (*b.enum* ‘ {.. *n*})
unfolding *f'* **by** *rule unfold-locales*

have *0* < *n*
using ‹*n* ≠ *0*› **by** *auto*

{ from ‹*a* = *enum i*› ‹*n* ≠ *0*› ‹*i* = *n*› *lb upd-space*[*of n'*]
obtain *i'* **where** *i'* ≤ *n enum i'* ≠ *enum n 0* < *enum i'* (*upd n'*)
unfolding *s-eq* **by** (*auto simp*: *enum-inj n'*)
moreover have *enum i'* (*upd n'*) = *base* (*upd n'*)
unfolding *enum-def* **using** ‹*i'* ≤ *n*› ‹*enum i'* ≠ *enum n*› **by** (*auto simp*: *n' upd-inj enum-inj*)
ultimately have *0* < *base* (*upd n'*)
by *auto* **}**
then have *benum1*: *b.enum* (*Suc 0*) = *base*
unfolding *b.enum-Suc*[*OF* ‹*0*<*n*›] *b.enum-0* **by** (*auto simp*: *b-def rot-def*)

have [*simp*]: ⋀*j*. *Suc j* < *n* ⟹ *rot* ‘ {..< *Suc j*} = {*n'*} ∪ {..< *j*}
by (*auto simp*: *rot-def image-iff Ball-def split*: *nat.splits*)
have *rot-simps*: ⋀*j*. *rot* (*Suc j*) = *j rot 0* = *n'*
by (*simp-all add*: *rot-def*)

{ fix *j* **assume** *j*: *Suc j* ≤ *n* **then have** *b.enum* (*Suc j*) = *enum j*
by (*induct j*) (*auto simp add*: *benum1 enum-0 b.enum-Suc enum-Suc rot-simps*) **}**
note *b-enum-eq-enum* = *this*
then have *enum* ‘ {..< *n*} = *b.enum* ‘ *Suc* ‘ {..< *n*}
by (*auto simp add*: *image-comp intro*!: *image-cong*)
also have *Suc* ‘ {..< *n*} = {.. *n*} − {*0*}
by (*auto simp*: *image-iff Ball-def*) *arith*
also have {..< *n*} = {.. *n*} − {*n*}
by *auto*
finally have *eq*: *s* − {*a*} = *b.enum* ‘ {.. *n*} − {*b.enum 0*}
unfolding *s-eq* ‹*a* = *enum i*› ‹*i* = *n*›
using *inj-on-image-set-diff*[*OF inj-enum Diff-subset*, *of* {*n*}]
inj-on-image-set-diff[*OF b.inj-enum Diff-subset*, *of* {*0*}]
by (*simp add*: *comp-def*)

```
  have b.enum 0 ≤ b.enum n
    by (simp add: b.enum-mono)
  also have b.enum n < enum n
    using ‹n ≠ 0› by (simp add: enum-strict-mono b-enum-eq-enum n')
  finally have a ≠ b.enum 0
    using ‹a = enum i› ‹i = n› by auto

  { fix t c assume ksimplex p n t c ∈ t and eq-sma: s − {a} = t − {c}
    obtain b' u where kuhn-simplex p n b' u t
      using ‹ksimplex p n t› by (auto elim: ksimplex.cases)
    then interpret t: kuhn-simplex p n b' u t .

    { fix x assume x ∈ s x ≠ a
      then have x (upd n') = enum n' (upd n')
          by (auto simp: ‹a = enum i› n' ‹i = n› s-eq enum-def enum-inj
in-upd-image) }
    then have eq-upd0: ∀ x∈t−{c}. x (upd n') = enum n' (upd n')
      unfolding eq-sma[symmetric] by auto
    then have c (upd n') ≠ enum n' (upd n')
    using ‹n ≠ 0› by (intro t.one-step[OF ‹c∈t›]) (auto simp: n' upd-space[unfolded
n'])
    then have c (upd n') < enum n' (upd n') ∨ c (upd n') > enum n' (upd n')
      by auto
    then have t = s ∨ t = b.enum ' {..n}
    proof (elim disjE conjE)
      assume *: c (upd n') > enum n' (upd n')
      interpret st: kuhn-simplex-pair p n base upd s b' u t ..
      { fix x assume x ∈ t with * ‹c∈t› eq-upd0[rule-format, of x] have x ≤ c
        by (auto simp: le-less intro!: t.less[of - - upd n']) }
      note top = this
      have s = t
        using ‹a = enum i› ‹i = n› ‹c ∈ t›
        by (intro st.ksimplex-eq-top[OF - - - - eq-sma])
          (auto simp: s-eq enum-mono t.s-eq t.enum-mono top)
      then show ?thesis by simp
    next
      assume *: c (upd n') < enum n' (upd n')
      interpret st: kuhn-simplex-pair p n b upd ∘ rot f' ' {.. n} b' u t ..
      have eq: f' ' {..n} − {b.enum 0} = t − {c}
        using eq-sma eq f' by simp
      { fix x assume x ∈ t with * ‹c∈t› eq-upd0[rule-format, of x] have c ≤ x
        by (auto simp: le-less intro!: t.less[of - - upd n']) }
      note bot = this
      have f' ' {..n} = t
        using ‹a = enum i› ‹i = n› ‹c ∈ t›
        by (intro st.ksimplex-eq-bot[OF - - - - eq])
          (auto simp: b.s-eq b.enum-mono t.s-eq t.enum-mono bot)
      with f' show ?thesis by simp
```

```
    qed }
  with ks-f' eq ‹a ≠ b.enum 0› ‹n ≠ 0› show ?thesis
    apply (intro ex1I[of - b.enum ` {.. n}])
    apply auto []
    apply metis
    done
next
  assume i: 0 < i i < n
  def i' ≡ i − 1
  with i have Suc i' < n
    by simp
  with i have Suc-i': Suc i' = i
    by (simp add: i'-def)

  let ?upd = Fun.swap i' i upd
  from i upd have bij-betw ?upd {..< n} {..< n}
    by (subst bij-betw-swap-iff) (auto simp: i'-def)

  def f' ≡ λi j. if j ∈ ?upd`{..< i} then Suc (base j) else base j
  interpret b: kuhn-simplex p n base ?upd f' ` {.. n}
  proof
    show base ∈ {..<n} → {..<p} by fact
    { fix i assume n ≤ i then show base i = p by fact }
    show bij-betw ?upd {..<n} {..<n} by fact
  qed (simp add: f'-def)
  have f': b.enum = f' unfolding f'-def b.enum-def[abs-def] ..
  have ks-f': ksimplex p n (b.enum ` {.. n})
    unfolding f' by rule unfold-locales

  have {i} ⊆ {..n}
    using i by auto
  { fix j assume j ≤ n
    moreover have j < i ∨ i = j ∨ i < j by arith
    moreover note i
    ultimately have enum j = b.enum j ⟷ j ≠ i
      unfolding enum-def[abs-def] b.enum-def[abs-def]
      by (auto simp add: fun-eq-iff swap-image i'-def
                         in-upd-image inj-on-image-set-diff[OF inj-upd]) }
  note enum-eq-benum = this
  then have enum ` ({.. n} − {i}) = b.enum ` ({.. n} − {i})
    by (intro image-cong) auto
  then have eq: s − {a} = b.enum ` {.. n} − {b.enum i}
    unfolding s-eq ‹a = enum i›
    using inj-on-image-set-diff[OF inj-enum Diff-subset ‹{i} ⊆ {..n}›]
          inj-on-image-set-diff[OF b.inj-enum Diff-subset ‹{i} ⊆ {..n}›]
    by (simp add: comp-def)

  have a ≠ b.enum i
    using ‹a = enum i› enum-eq-benum i by auto
```

{ fix t c **assume** *ksimplex* p n t $c \in t$ **and** *eq-sma*: $s - \{a\} = t - \{c\}$
obtain b' u **where** *kuhn-simplex* p n b' u t
using ‹*ksimplex* p n t› **by** (*auto elim*: *ksimplex.cases*)
then interpret t: *kuhn-simplex* p n b' u t **.**
have *enum* $i' \in s - \{a\}$ *enum* $(i + 1) \in s - \{a\}$
using ‹a = *enum* i› i *enum-in* **by** (*auto simp*: *enum-inj* i'*-def*)
then obtain l k **where**
l: $t.enum\ l$ = *enum* i' $l \leq n$ $t.enum\ l \neq c$ **and**
k: $t.enum\ k$ = *enum* $(i + 1)$ $k \leq n$ $t.enum\ k \neq c$
unfolding *eq-sma* **by** (*auto simp*: *t.s-eq*)
with i **have** $t.enum\ l < t.enum\ k$
by (*simp add*: *enum-strict-mono* i'*-def*)
with ‹$l \leq n$› ‹$k \leq n$› **have** $l < k$
by (*simp add*: *t.enum-strict-mono*)
{ assume *Suc* $l = k$
have *enum* (*Suc* (*Suc* i')) = $t.enum$ (*Suc* l)
using i **by** (*simp add*: k ‹*Suc* $l = k$› i'*-def*)
then have *False*
using ‹$l < k$› ‹$k \leq n$› ‹*Suc* $i' < n$›
by (*auto simp*: *t.enum-Suc enum-Suc l upd-inj fun-eq-iff split*: *split-if-asm*)
(*metis Suc-lessD n-not-Suc-n upd-inj*) **}**
with ‹$l < k$› **have** *Suc* $l < k$
by *arith*
have *c-eq*: $c = t.enum$ (*Suc* l)
proof (*rule ccontr*)
assume $c \neq t.enum$ (*Suc* l)
then have $t.enum$ (*Suc* l) $\in s - \{a\}$
using ‹$l < k$› ‹$k \leq n$› **by** (*simp add*: *t.s-eq eq-sma*)
then obtain j **where** $t.enum$ (*Suc* l) = *enum* j $j \leq n$ *enum* $j \neq$ *enum* i
unfolding *s-eq* ‹a = *enum* i› **by** *auto*
with i **have** $t.enum$ (*Suc* l) $\leq t.enum\ l \vee t.enum\ k \leq t.enum$ (*Suc* l)
by (*auto simp add*: i'*-def enum-mono enum-inj l k*)
with ‹*Suc* $l < k$› ‹$k \leq n$› **show** *False*
by (*simp add*: *t.enum-mono*)
qed

{ have $t.enum$ (*Suc* (*Suc* l)) $\in s - \{a\}$
unfolding *eq-sma c-eq t.s-eq* **using** ‹*Suc* $l < k$› ‹$k \leq n$› **by** (*auto simp*: *t.enum-inj*)
then obtain j **where** *eq*: $t.enum$ (*Suc* (*Suc* l)) = *enum* j **and** $j \leq n$ $j \neq i$
by (*auto simp*: *s-eq* ‹a = *enum* i›)
moreover have *enum* $i' < t.enum$ (*Suc* (*Suc* l))
unfolding $l(1)$[*symmetric*] **using** ‹*Suc* $l < k$› ‹$k \leq n$› **by** (*auto simp*: *t.enum-strict-mono*)
ultimately have $i' < j$
using i **by** (*simp add*: *enum-strict-mono* i'*-def*)
with ‹$j \neq i$› ‹$j \leq n$› **have** $t.enum\ k \leq t.enum$ (*Suc* (*Suc* l))
unfolding i'*-def* **by** (*simp add*: *enum-mono k eq*)

```
      then have k ≤ Suc (Suc l)
        using ‹k ≤ n› ‹Suc l < k› by (simp add: t.enum-mono) }
      with ‹Suc l < k› have Suc (Suc l) = k by simp
      then have enum (Suc (Suc i')) = t.enum (Suc (Suc l))
        using i by (simp add: k i'-def)
       also have ... = (enum i') (u l := Suc (enum i' (u l)), u (Suc l) := Suc
(enum i' (u (Suc l))))
        using ‹Suc l < k› ‹k ≤ n› by (simp add: t.enum-Suc l t.upd-inj)
      finally have (u l = upd i' ∧ u (Suc l) = upd (Suc i')) ∨
        (u l = upd (Suc i') ∧ u (Suc l) = upd i')
        using ‹Suc i' < n› by (auto simp: enum-Suc fun-eq-iff split: split-if-asm)

      then have t = s ∨ t = b.enum ' {..n}
      proof (elim disjE conjE)
        assume u: u l = upd i'
        have c = t.enum (Suc l) unfolding c-eq ..
        also have t.enum (Suc l) = enum (Suc i')
          using u ‹l < k› ‹k ≤ n› ‹Suc i' < n› by (simp add: enum-Suc t.enum-Suc
l)
        also have ... = a
          using ‹a = enum i› i by (simp add: i'-def)
        finally show ?thesis
          using eq-sma ‹a ∈ s› ‹c ∈ t› by auto
      next
        assume u: u l = upd (Suc i')
        def B ≡ b.enum ' {..n}
        have b.enum i' = enum i'
          using enum-eq-benum[of i'] i by (auto simp add: i'-def gr0-conv-Suc)
        have c = t.enum (Suc l) unfolding c-eq ..
        also have t.enum (Suc l) = b.enum (Suc i')
          using u ‹l < k› ‹k ≤ n› ‹Suc i' < n›
          by (simp-all add: enum-Suc t.enum-Suc l b.enum-Suc ‹b.enum i' = enum
i'› swap-apply1)
             (simp add: Suc-i')
        also have ... = b.enum i
          using i by (simp add: i'-def)
        finally have c = b.enum i .
        then have t - {c} = B - {c} c ∈ B
          unfolding eq-sma[symmetric] eq B-def using i by auto
        with ‹c ∈ t› have t = B
          by auto
        then show ?thesis
          by (simp add: B-def)
      qed }
    with ks-f' eq ‹a ≠ b.enum i› ‹n ≠ 0› ‹i ≤ n› show ?thesis
      apply (intro ex1I[of - b.enum ' {.. n}])
      apply auto []
      apply metis
      done
```

qed
then show *?thesis*
using *s* ‹*a* ∈ *s*› **by** (*simp add*: *card-2-exists Ex1-def*) *metis*
qed

Hence another step towards concreteness.

lemma *kuhn-simplex-lemma*:
assumes ∀ *s*. *ksimplex p* (*Suc n*) *s* ⟶ *rl* ‘ *s* ⊆ {.. *Suc n*}
and *odd* (*card* {*f*. ∃ *s a*. *ksimplex p* (*Suc n*) *s* ∧ *a* ∈ *s* ∧ (*f* = *s* − {*a*}) ∧
rl ‘ *f* = {..*n*} ∧ ((∃ *j*≤*n*. ∀ *x*∈*f*. *x j* = *0*) ∨ (∃ *j*≤*n*. ∀ *x*∈*f*. *x j* = *p*))})
shows *odd* (*card* {*s*. *ksimplex p* (*Suc n*) *s* ∧ *rl* ‘ *s* = {..*Suc n*}})
proof (*rule kuhn-complete-lemma*[*OF finite-ksimplexes refl*, *unfolded mem-Collect-eq*,
where *bnd*=λ*f*. (∃ *j*∈{..*n*}. ∀ *x*∈*f*. *x j* = *0*) ∨ (∃ *j*∈{..*n*}. ∀ *x*∈*f*. *x j* = *p*)],
safe del: *notI*)

have ∗: ⋀*x y*. *x* = *y* ⟹ *odd* (*card x*) ⟹ *odd* (*card y*)
by *auto*
show *odd* (*card* {*f*. (∃ *s*∈{*s*. *ksimplex p* (*Suc n*) *s*}. ∃ *a*∈*s*. *f* = *s* − {*a*}) ∧
rl ‘ *f* = {..*n*} ∧ ((∃ *j*∈{..*n*}. ∀ *x*∈*f*. *x j* = *0*) ∨ (∃ *j*∈{..*n*}. ∀ *x*∈*f*. *x j* = *p*))})
apply (*rule* ∗[*OF* - *assms*(*2*)])
apply (*auto simp*: *atLeast0AtMost*)
done

next

fix *s* **assume** *s*: *ksimplex p* (*Suc n*) *s*
then show *card s* = *n* + *2*
by (*simp add*: *ksimplex-card*)

fix *a* **assume** *a*: *a* ∈ *s* **then show** *rl a* ≤ *Suc n*
using *assms*(*1*) *s* **by** (*auto simp*: *subset-eq*)

let *?S* = {*t*. *ksimplex p* (*Suc n*) *t* ∧ (∃ *b*∈*t*. *s* − {*a*} = *t* − {*b*})}
{ fix *j* **assume** *j*: *j* ≤ *n* ∀ *x*∈*s* − {*a*}. *x j* = *0*
with *s a* **show** *card ?S* = *1*
using *ksimplex-replace-0*[*of p n* + *1 s a j*]
by (*subst eq-commute*) *simp* **}**

{ fix *j* **assume** *j*: *j* ≤ *n* ∀ *x*∈*s* − {*a*}. *x j* = *p*
with *s a* **show** *card ?S* = *1*
using *ksimplex-replace-1*[*of p n* + *1 s a j*]
by (*subst eq-commute*) *simp* **}**

{ assume *card ?S* ≠ *2* ¬ (∃ *j*∈{..*n*}. ∀ *x*∈*s* − {*a*}. *x j* = *p*)
with *s a* **show** ∃ *j*∈{..*n*}. ∀ *x*∈*s* − {*a*}. *x j* = *0*
using *ksimplex-replace-2*[*of p n* + *1 s a*]
by (*subst* (*asm*) *eq-commute*) *auto* **}**
qed

20.3 Reduced labelling

definition *reduced* :: *nat* ⇒ (*nat* ⇒ *nat*) ⇒ *nat* **where** *reduced n x* = (*LEAST k*. *k* = *n* ∨ *x k* ≠ *0*)

lemma *reduced-labelling*:
shows *reduced n x* ≤ *n*
and ∀ *i*<*reduced n x*. *x i* = *0*
and *reduced n x* = *n* ∨ *x* (*reduced n x*) ≠ *0*
proof −
show *reduced n x* ≤ *n*
unfolding *reduced-def* **by** (*rule LeastI2-wellorder*[**where** *a*=*n*]) *auto*
show ∀ *i*<*reduced n x*. *x i* = *0*
unfolding *reduced-def* **by** (*rule LeastI2-wellorder*[**where** *a*=*n*]) *fastforce*+
show *reduced n x* = *n* ∨ *x* (*reduced n x*) ≠ *0*
unfolding *reduced-def* **by** (*rule LeastI2-wellorder*[**where** *a*=*n*]) *fastforce*+
qed

lemma *reduced-labelling-unique*:
r ≤ *n* ⟹ ∀ *i*<*r*. *x i* = *0* ⟹ *r* = *n* ∨ *x r* ≠ *0* ⟹ *reduced n x* = *r*
unfolding *reduced-def* **by** (*rule LeastI2-wellorder*[**where** *a*=*n*]) (*metis le-less not-le*)+

lemma *reduced-labelling-zero*: *j* < *n* ⟹ *x j* = *0* ⟹ *reduced n x* ≠ *j*
using *reduced-labelling*[*of n x*] **by** *auto*

lemma *reduce-labelling-zero*[*simp*]: *reduced 0 x* = *0*
by (*rule reduced-labelling-unique*) *auto*

lemma *reduced-labelling-nonzero*: *j* < *n* ⟹ *x j* ≠ *0* ⟹ *reduced n x* ≤ *j*
using *reduced-labelling*[*of n x*] **by** (*elim allE*[**where** *x*=*j*]) *auto*

lemma *reduced-labelling-Suc*: *reduced* (*Suc n*) *x* ≠ *Suc n* ⟹ *reduced* (*Suc n*) *x* = *reduced n x*
using *reduced-labelling*[*of Suc n x*]
by (*intro reduced-labelling-unique*[*symmetric*]) *auto*

lemma *complete-face-top*:
assumes ∀ *x*∈*f*. ∀ *j*≤*n*. *x j* = *0* ⟶ *lab x j* = *0*
and ∀ *x*∈*f*. ∀ *j*≤*n*. *x j* = *p* ⟶ *lab x j* = *1*
and *eq*: (*reduced* (*Suc n*) ∘ *lab*) ‘ *f* = {..*n*}
shows ((∃ *j*≤*n*. ∀ *x*∈*f*. *x j* = *0*) ∨ (∃ *j*≤*n*. ∀ *x*∈*f*. *x j* = *p*)) ⟷ (∀ *x*∈*f*. *x n* = *p*)
proof (*safe del*: *disjCI*)
fix *x j* **assume** *j*: *j* ≤ *n* ∀ *x*∈*f*. *x j* = *0*
{ fix *x* **assume** *x* ∈ *f* **with** *assms j* **have** *reduced* (*Suc n*) (*lab x*) ≠ *j*
by (*intro reduced-labelling-zero*) *auto* **}**
moreover have *j* ∈ (*reduced* (*Suc n*) ∘ *lab*) ‘ *f*
using *j eq* **by** *auto*
ultimately show *x n* = *p*

```
    by force
next
  fix x j assume j: j ≤ n ∀x∈f. x j = p and x: x ∈ f
  have j = n
  proof (rule ccontr)
    assume ¬ ?thesis
    { fix x assume x ∈ f
      with assms j have reduced (Suc n) (lab x) ≤ j
        by (intro reduced-labelling-nonzero) auto
      then have reduced (Suc n) (lab x) ≠ n
        using ‹j ≠ n› ‹j ≤ n› by simp }
    moreover
    have n ∈ (reduced (Suc n) ∘ lab) ‘ f
      using eq by auto
    ultimately show False
      by force
  qed
  moreover have j ∈ (reduced (Suc n) ∘ lab) ‘ f
    using j eq by auto
  ultimately show x n = p
    using j x by auto
qed auto
```

Hence we get just about the nice induction.

```
lemma kuhn-induction:
  assumes 0 < p
    and lab-0: ∀x. ∀j≤n. (∀j. x j ≤ p) ∧ x j = 0 ⟶ lab x j = 0
    and lab-1: ∀x. ∀j≤n. (∀j. x j ≤ p) ∧ x j = p ⟶ lab x j = 1
    and odd: odd (card {s. ksimplex p n s ∧ (reduced n∘lab) ‘ s = {..n}})
  shows odd (card {s. ksimplex p (Suc n) s ∧ (reduced (Suc n)∘lab) ‘ s = {..Suc
n}})
proof −
  let ?rl = reduced (Suc n) ∘ lab and ?ext = λf v. ∃j≤n. ∀x∈f. x j = v
  let ?ext = λs. (∃j≤n. ∀x∈s. x j = 0) ∨ (∃j≤n. ∀x∈s. x j = p)
  have ∀s. ksimplex p (Suc n) s ⟶ ?rl ‘ s ⊆ {..Suc n}
    by (simp add: reduced-labelling subset-eq)
  moreover
  have {s. ksimplex p n s ∧ (reduced n ∘ lab) ‘ s = {..n}} =
        {f. ∃s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ f = s − {a} ∧ ?rl ‘ f = {..n} ∧
?ext f}
  proof (intro set-eqI, safe del: disjCI equalityI disjE)
    fix s assume s: ksimplex p n s and rl: (reduced n ∘ lab) ‘ s = {..n}
   from s obtain u b where kuhn-simplex p n u b s by (auto elim: ksimplex.cases)
    then interpret kuhn-simplex p n u b s .
    have all-eq-p: ∀x∈s. x n = p
      by (auto simp: out-eq-p)
    moreover
    { fix x assume x ∈ s
      with lab-1[rule-format, of n x] all-eq-p s-le-p[of x]
```

```
    have ?rl x ≤ n
      by (auto intro!: reduced-labelling-nonzero)
    then have ?rl x = reduced n (lab x)
      by (auto intro!: reduced-labelling-Suc) }
  then have ?rl ‘ s = {..n}
    using rl by (simp cong: image-cong)
  moreover
  obtain t a where ksimplex p (Suc n) t a ∈ t s = t − {a}
    using s unfolding simplex-top-face[OF ‹0 < p› all-eq-p] by auto
  ultimately
  show ∃t a. ksimplex p (Suc n) t ∧ a ∈ t ∧ s = t − {a} ∧ ?rl ‘ s = {..n} ∧
?ext s
    by auto
 next
  fix x s a assume s: ksimplex p (Suc n) s and rl: ?rl ‘ (s − {a}) = {.. n}
    and a: a ∈ s and ?ext (s − {a})
   from s obtain u b where kuhn-simplex p (Suc n) u b s by (auto elim:
ksimplex.cases)
  then interpret kuhn-simplex p Suc n u b s .
  have all-eq-p: ∀x∈s. x (Suc n) = p
    by (auto simp: out-eq-p)

  { fix x assume x ∈ s − {a}
    then have ?rl x ∈ ?rl ‘ (s − {a})
      by auto
    then have ?rl x ≤ n
      unfolding rl by auto
    then have ?rl x = reduced n (lab x)
      by (auto intro!: reduced-labelling-Suc) }
  then show rl': (reduced n∘lab) ‘ (s − {a}) = {..n}
    unfolding rl[symmetric] by (intro image-cong) auto

  from ‹?ext (s − {a})›
  have all-eq-p: ∀x∈s − {a}. x n = p
  proof (elim disjE exE conjE)
    fix j assume j ≤ n ∀x∈s − {a}. x j = 0
    with lab-0[rule-format, of j] all-eq-p s-le-p
    have ⋀x. x ∈ s − {a} ⟹ reduced (Suc n) (lab x) ≠ j
      by (intro reduced-labelling-zero) auto
    moreover have j ∈ ?rl ‘ (s − {a})
      using ‹j ≤ n› unfolding rl by auto
    ultimately show ?thesis
      by force
  next
    fix j assume j ≤ n and eq-p: ∀x∈s − {a}. x j = p
    show ?thesis
    proof cases
      assume j = n with eq-p show ?thesis by simp
    next
```

```
      assume j ≠ n
      { fix x assume x: x ∈ s − {a}
        have reduced n (lab x) ≤ j
        proof (rule reduced-labelling-nonzero)
          show lab x j ≠ 0
            using lab-1[rule-format, of j x] x s-le-p[of x] eq-p ‹j ≤ n› by auto
          show j < n
            using ‹j ≤ n› ‹j ≠ n› by simp
        qed
        then have reduced n (lab x) ≠ n
          using ‹j ≤ n› ‹j ≠ n› by simp }
      moreover have n ∈ (reduced n∘lab) ` (s − {a})
        unfolding rl' by auto
      ultimately show ?thesis
        by force
    qed
  qed
  show ksimplex p n (s − {a})
    unfolding simplex-top-face[OF ‹0 < p› all-eq-p] using s a by auto
 qed
 ultimately show ?thesis
   using assms by (intro kuhn-simplex-lemma) auto
qed
```

And so we get the final combinatorial result.

```
lemma ksimplex-0: ksimplex p 0 s ⟷ s = {(λx. p)}
proof
  assume ksimplex p 0 s then show s = {(λx. p)}
    by (blast dest: kuhn-simplex.ksimplex-0 elim: ksimplex.cases)
next
  assume s: s = {(λx. p)}
  show ksimplex p 0 s
  proof (intro ksimplex, unfold-locales)
    show (λ-. p) ∈ {..<0::nat} → {..<p} by auto
    show bij-betw id {..<0} {..<0}
      by simp
  qed (auto simp: s)
qed

lemma kuhn-combinatorial:
  assumes 0 < p
    and ∀x j. (∀j. x j ≤ p) ∧ j < n ∧ x j = 0 ⟶ lab x j = 0
    and ∀x j. (∀j. x j ≤ p) ∧ j < n  ∧ x j = p ⟶ lab x j = 1
  shows odd (card {s. ksimplex p n s ∧ (reduced n∘lab) ` s = {..n}})
    (is odd (card (?M n)))
  using assms
proof (induct n)
  case 0 then show ?case
    by (simp add: ksimplex-0 cong: conj-cong)
```

next
case (*Suc n*)
then have *odd* (*card* (*?M n*))
by *force*
with *Suc* **show** *?case*
using *kuhn-induction*[*of p n*] **by** (*auto simp*: *comp-def*)
qed

lemma *kuhn-lemma*:
fixes *n p* :: *nat*
assumes *0* < *p*
and ∀ *x*. (∀ *i*<*n*. *x i* ≤ *p*) ⟶ (∀ *i*<*n*. *label x i* = (*0*::*nat*) ∨ *label x i* = *1*)
and ∀ *x*. (∀ *i*<*n*. *x i* ≤ *p*) ⟶ (∀ *i*<*n*. *x i* = *0* ⟶ *label x i* = *0*)
and ∀ *x*. (∀ *i*<*n*. *x i* ≤ *p*) ⟶ (∀ *i*<*n*. *x i* = *p* ⟶ *label x i* = *1*)
obtains *q* **where** ∀ *i*<*n*. *q i* < *p*
and ∀ *i*<*n*. ∃ *r s*. (∀ *j*<*n*. *q j* ≤ *r j* ∧ *r j* ≤ *q j* + *1*) ∧ (∀ *j*<*n*. *q j* ≤ *s j* ∧ *s j* ≤ *q j* + *1*) ∧ *label r i* ≠ *label s i*
proof −
let *?rl* = *reduced n* ∘ *label*
let *?A* = {*s*. *ksimplex p n s* ∧ *?rl* ' *s* = {..*n*}}
have *odd* (*card ?A*)
using *assms* **by** (*intro kuhn-combinatorial*[*of p n label*]) *auto*
then have *?A* ≠ {}
by *fastforce*
then obtain *s b u* **where** *kuhn-simplex p n b u s* **and** *rl*: *?rl* ' *s* = {..*n*}
by (*auto elim*: *ksimplex.cases*)
interpret *kuhn-simplex p n b u s* **by** *fact*

show *?thesis*
proof (*intro that*[*of b*] *allI impI*)
fix *i*
assume *i* < *n*
then show *b i* < *p*
using *base* **by** *auto*
next
fix *i*
assume *i* < *n*
then have *i* ∈ {.. *n*} *Suc i* ∈ {.. *n*}
by *auto*
then obtain *u v* **where** *u*: *u* ∈ *s Suc i* = *?rl u* **and** *v*: *v* ∈ *s i* = *?rl v*
unfolding *rl*[*symmetric*] **by** *blast*

have *label u i* ≠ *label v i*
using *reduced-labelling* [*of n label u*] *reduced-labelling* [*of n label v*]
u(*2*)[*symmetric*] *v*(*2*)[*symmetric*] ‹*i* < *n*›
by *auto*
moreover
have *b j* ≤ *u j u j* ≤ *b j* + *1 b j* ≤ *v j v j* ≤ *b j* + *1* **if** *j* < *n* **for** *j*
using *that base-le*[*OF* ‹*u*∈*s*›] *le-Suc-base*[*OF* ‹*u*∈*s*›] *base-le*[*OF* ‹*v*∈*s*›] *le-Suc-base*[*OF*

$\langle v \in s \rangle$]
by *auto*
ultimately show $\exists r\ s.\ (\forall j{<}n.\ b\ j \le r\ j \wedge r\ j \le b\ j + 1) \wedge$
$(\forall j{<}n.\ b\ j \le s\ j \wedge s\ j \le b\ j + 1) \wedge$ *label r i* $\neq$ *label s i*
by *blast*
qed
qed

20.4 The main result for the unit cube

lemma *kuhn-labelling-lemma'*:
assumes $(\forall x{::}nat \Rightarrow real.\ P\ x \longrightarrow P\ (f\ x))$
and $\forall x.\ P\ x \longrightarrow (\forall i{::}nat.\ Q\ i \longrightarrow 0 \le x\ i \wedge x\ i \le 1)$
shows $\exists l.\ (\forall x\ i.\ l\ x\ i \le (1{::}nat)) \wedge$
$(\forall x\ i.\ P\ x \wedge Q\ i \wedge x\ i = 0 \longrightarrow l\ x\ i = 0) \wedge$
$(\forall x\ i.\ P\ x \wedge Q\ i \wedge x\ i = 1 \longrightarrow l\ x\ i = 1) \wedge$
$(\forall x\ i.\ P\ x \wedge Q\ i \wedge l\ x\ i = 0 \longrightarrow x\ i \le f\ x\ i) \wedge$
$(\forall x\ i.\ P\ x \wedge Q\ i \wedge l\ x\ i = 1 \longrightarrow f\ x\ i \le x\ i)$
proof −
have *and-forall-thm*: $\bigwedge P\ Q.\ (\forall x.\ P\ x) \wedge (\forall x.\ Q\ x) \longleftrightarrow (\forall x.\ P\ x \wedge Q\ x)$
by *auto*
have $*$: $\forall x\ y{::}real.\ 0 \le x \wedge x \le 1 \wedge 0 \le y \wedge y \le 1 \longrightarrow x \neq 1 \wedge x \le y \vee x \neq 0 \wedge y \le x$
by *auto*
show *?thesis*
unfolding *and-forall-thm*
apply (*subst choice-iff*[*symmetric*])+
apply *rule*
apply *rule*
proof −
fix *x x'*
let *?R* $= \lambda y{::}nat.$
$(P\ x \wedge Q\ x' \wedge x\ x' = 0 \longrightarrow y = 0) \wedge$
$(P\ x \wedge Q\ x' \wedge x\ x' = 1 \longrightarrow y = 1) \wedge$
$(P\ x \wedge Q\ x' \wedge y = 0 \longrightarrow x\ x' \le (f\ x)\ x') \wedge$
$(P\ x \wedge Q\ x' \wedge y = 1 \longrightarrow (f\ x)\ x' \le x\ x')$
have $0 \le f\ x\ x' \wedge f\ x\ x' \le 1$ **if** $P\ x\ Q\ x'$
using *assms*(*2*)[*rule-format*,*of f x x'*] *that*
apply (*drule-tac assms*(*1*)[*rule-format*])
apply *auto*
done
then have *?R 0* $\vee$ *?R 1*
by *auto*
then show $\exists y{\le}1.$ *?R y*
by *auto*
qed
qed

definition *unit-cube* :: *'a*::*euclidean-space set*

where *unit-cube* = $\{x.\ \forall i{\in}Basis.\ 0 \le x \cdot i \wedge x \cdot i \le 1\}$

lemma *mem-unit-cube*: $x \in$ *unit-cube* $\longleftrightarrow$ $(\forall i{\in}Basis.\ 0 \le x \cdot i \wedge x \cdot i \le 1)$
unfolding *unit-cube-def* **by** *simp*

lemma *bounded-unit-cube*: *bounded unit-cube*
unfolding *bounded-def*
proof (*intro exI ballI*)
fix $y :: 'a$ **assume** *y*: $y \in$ *unit-cube*
have *dist 0 y* = *norm y* **by** (*rule dist-0-norm*)
also have $\ldots$ = *norm* $(\sum i{\in}Basis.\ (y \cdot i) *_R i)$ **unfolding** *euclidean-representation*
..
also have $\ldots \le (\sum i{\in}Basis.\ norm\ ((y \cdot i) *_R i))$ **by** (*rule norm-setsum*)
also have $\ldots \le (\sum i{::}'a{\in}Basis.\ 1)$
by (*rule setsum-mono*, *simp add*: *y* [*unfolded mem-unit-cube*])
finally show *dist 0 y* $\le (\sum i{::}'a{\in}Basis.\ 1)$ **.**
qed

lemma *closed-unit-cube*: *closed unit-cube*
unfolding *unit-cube-def Collect-ball-eq Collect-conj-eq*
by (*rule closed-INT*, *auto intro*!: *closed-Collect-le*)

lemma *compact-unit-cube*: *compact unit-cube* (**is** *compact ?C*)
unfolding *compact-eq-seq-compact-metric*
using *bounded-unit-cube closed-unit-cube*
by (*rule bounded-closed-imp-seq-compact*)

lemma *brouwer-cube*:
fixes $f :: 'a{::}euclidean\text{-}space \Rightarrow 'a$
assumes *continuous-on unit-cube f*
and *f* ‘ *unit-cube* $\subseteq$ *unit-cube*
shows $\exists x{\in}unit\text{-}cube.\ f\ x = x$
proof (*rule ccontr*)
def $n \equiv DIM('a)$
have *n*: $1 \le n\ 0 < n\ n \neq 0$
unfolding *n-def* **by** (*auto simp add*: *Suc-le-eq DIM-positive*)
assume $\neg$ *?thesis*
then have $*$: $\neg\ (\exists x{\in}unit\text{-}cube.\ f\ x - x = 0)$
by *auto*
obtain *d* **where**
d: $d > 0\ \bigwedge x.\ x \in unit\text{-}cube \Longrightarrow d \le norm\ (f\ x - x)$
apply (*rule brouwer-compactness-lemma*[*OF compact-unit-cube - **])
apply (*rule continuous-intros assms*)+
apply *blast*
done
have $*$: $\forall x.\ x \in unit\text{-}cube \longrightarrow f\ x \in unit\text{-}cube$
$\forall x.\ x \in (unit\text{-}cube{::}'a\ set) \longrightarrow (\forall i{\in}Basis.\ True \longrightarrow 0 \le x \cdot i \wedge x \cdot i \le 1)$
using *assms*(*2*)[*unfolded image-subset-iff Ball-def*]
unfolding *mem-unit-cube*

by *auto*
obtain *label* :: $'a \Rightarrow 'a \Rightarrow nat$ **where**
$\forall x.\ \forall i{\in}Basis.\ label\ x\ i \leq 1$
$\forall x.\ \forall i{\in}Basis.\ x \in unit\text{-}cube \wedge True \wedge x \cdot i = 0 \longrightarrow label\ x\ i = 0$
$\forall x.\ \forall i{\in}Basis.\ x \in unit\text{-}cube \wedge True \wedge x \cdot i = 1 \longrightarrow label\ x\ i = 1$
$\forall x.\ \forall i{\in}Basis.\ x \in unit\text{-}cube \wedge True \wedge label\ x\ i = 0 \longrightarrow x \cdot i \leq f\ x \cdot i$
$\forall x.\ \forall i{\in}Basis.\ x \in unit\text{-}cube \wedge True \wedge label\ x\ i = 1 \longrightarrow f\ x \cdot i \leq x \cdot i$
using *kuhn-labelling-lemma*[*OF* $*$] **by** *blast*
note *label* = *this* [*rule-format*]
have *lem1*: $\forall x{\in}unit\text{-}cube.\ \forall y{\in}unit\text{-}cube.\ \forall i{\in}Basis.\ label\ x\ i \neq label\ y\ i \longrightarrow$
$|f\ x \cdot i - x \cdot i| \leq norm\ (f\ y - f\ x) + norm\ (y - x)$
proof *safe*
fix $x\ y :: 'a$
assume x: $x \in unit\text{-}cube$
assume y: $y \in unit\text{-}cube$
fix i
assume i: $label\ x\ i \neq label\ y\ i\ i \in Basis$
have $*$: $\bigwedge x\ y\ fx\ fy :: real.\ x \leq fx \wedge fy \leq y \vee fx \leq x \wedge y \leq fy \Longrightarrow$
$|fx - x| \leq |fy - fx| + |y - x|$ **by** *auto*
have $|(f\ x - x) \cdot i| \leq |(f\ y - f\ x){\cdot}i| + |(y - x){\cdot}i|$
unfolding *inner-simps*
apply (*rule* $*$)
apply (*cases* $label\ x\ i = 0$)
apply (*rule disjI1*)
apply *rule*
prefer *3*
apply (*rule disjI2*)
apply *rule*
proof −
assume *lx*: $label\ x\ i = 0$
then have *ly*: $label\ y\ i = 1$
using i *label*(*1*)[*of* i y]
by *auto*
show $x \cdot i \leq f\ x \cdot i$
apply (*rule label*(*4*)[*rule-format*])
using x y *lx* i(*2*)
apply *auto*
done
show $f\ y \cdot i \leq y \cdot i$
apply (*rule label*(*5*)[*rule-format*])
using x y *ly* i(*2*)
apply *auto*
done
next
assume $label\ x\ i \neq 0$
then have *l*: $label\ x\ i = 1\ label\ y\ i = 0$
using i *label*(*1*)[*of* i x] *label*(*1*)[*of* i y]
by *auto*
show $f\ x \cdot i \leq x \cdot i$

```
      apply (rule label(5)[rule-format])
      using x y l i(2)
      apply auto
      done
    show y · i ≤ f y · i
      apply (rule label(4)[rule-format])
      using x y l i(2)
      apply auto
      done
  qed
  also have ... ≤ norm (f y − f x) + norm (y − x)
    apply (rule add-mono)
    apply (rule Basis-le-norm[OF i(2)])+
    done
  finally show |f x · i − x · i| ≤ norm (f y − f x) + norm (y − x)
    unfolding inner-simps .
qed
have ∃ e>0. ∀ x∈unit-cube. ∀ y∈unit-cube. ∀ z∈unit-cube. ∀ i∈Basis.
  norm (x − z) < e ∧ norm (y − z) < e ∧ label x i ≠ label y i ⟶
    |(f(z) − z)·i| < d / (real n)
proof −
  have d': d / real n / 8 > 0
    using d(1) by (simp add: n-def DIM-positive)
  have ∗: uniformly-continuous-on unit-cube f
    by (rule compact-uniformly-continuous[OF assms(1) compact-unit-cube])
  obtain e where e:
      e > 0
      ⋀x x'. x ∈ unit-cube ⟹
        x' ∈ unit-cube ⟹
        norm (x' − x) < e ⟹
        norm (f x' − f x) < d / real n / 8
    using ∗[unfolded uniformly-continuous-on-def,rule-format,OF d']
    unfolding dist-norm
    by blast
  show ?thesis
    apply (rule-tac x=min (e/2) (d/real n/8) in exI)
    apply safe
  proof −
    show 0 < min (e / 2) (d / real n / 8)
      using d' e by auto
    fix x y z i
    assume as:
      x ∈ unit-cube y ∈ unit-cube z ∈ unit-cube
      norm (x − z) < min (e / 2) (d / real n / 8)
      norm (y − z) < min (e / 2) (d / real n / 8)
      label x i ≠ label y i
    assume i: i ∈ Basis
    have ∗: ⋀z fz x fx n1 n2 n3 n4 d4 d :: real. |fx − x| ≤ n1 + n2 ⟹
      |fx − fz| ≤ n3 ⟹ |x − z| ≤ n4 ⟹
```

```
      n1 < d4 ⟹ n2 < 2 * d4 ⟹ n3 < d4 ⟹ n4 < d4 ⟹
      (8 * d4 = d) ⟹ |fz - z| < d
      by auto
    show |(f z - z) · i| < d / real n
      unfolding inner-simps
    proof (rule *)
      show |f x · i - x · i| ≤ norm (f y -f x) + norm (y - x)
        apply (rule lem1[rule-format])
        using as i
        apply auto
        done
      show |f x · i - f z · i| ≤ norm (f x - f z) |x · i - z · i| ≤ norm (x - z)
        unfolding inner-diff-left[symmetric]
        by (rule Basis-le-norm[OF i])+
      have tria: norm (y - x) ≤ norm (y - z) + norm (x - z)
        using dist-triangle[of y x z, unfolded dist-norm]
        unfolding norm-minus-commute
        by auto
      also have ... < e / 2 + e / 2
        apply (rule add-strict-mono)
        using as(4,5)
        apply auto
        done
      finally show norm (f y - f x) < d / real n / 8
        apply -
        apply (rule e(2))
        using as
        apply auto
        done
      have norm (y - z) + norm (x - z) < d / real n / 8 + d / real n / 8
        apply (rule add-strict-mono)
        using as
        apply auto
        done
      then show norm (y - x) < 2 * (d / real n / 8)
        using tria
        by auto
      show norm (f x - f z) < d / real n / 8
        apply (rule e(2))
        using as e(1)
        apply auto
        done
    qed (insert as, auto)
  qed
qed
then
obtain e where e:
  e > 0
  ⋀x y z i. x ∈ unit-cube ⟹
```

```
    y ∈ unit-cube ⟹
    z ∈ unit-cube ⟹
    i ∈ Basis ⟹
    norm (x − z) < e ∧ norm (y − z) < e ∧ label x i ≠ label y i ⟹
    |(f z − z) • i| < d / real n
  by blast
 obtain p :: nat where p: 1 + real n / e ≤ real p
  using real-arch-simple ..
 have 1 + real n / e > 0
  using e(1) n by (simp add: add-pos-pos)
 then have p > 0
  using p by auto

 obtain b :: nat ⇒ 'a where b: bij-betw b {..< n} Basis
  by atomize-elim (auto simp: n-def intro!: finite-same-card-bij)
 def b' ≡ inv-into {..< n} b
 then have b': bij-betw b' Basis {..< n}
  using bij-betw-inv-into[OF b] by auto
 then have b'-Basis: ⋀i. i ∈ Basis ⟹ b' i ∈ {..< n}
  unfolding bij-betw-def by (auto simp: set-eq-iff)
 have bb'[simp]:⋀i. i ∈ Basis ⟹ b (b' i) = i
  unfolding b'-def
  using b
  by (auto simp: f-inv-into-f bij-betw-def)
 have b'b[simp]:⋀i. i < n ⟹ b' (b i) = i
  unfolding b'-def
  using b
  by (auto simp: inv-into-f-eq bij-betw-def)
 have ∗: ⋀x :: nat. x = 0 ∨ x = 1 ⟷ x ≤ 1
  by auto
 have b'': ⋀j. j < n ⟹ b j ∈ Basis
  using b unfolding bij-betw-def by auto
 have q1: 0 < p ∀x. (∀i<n. x i ≤ p) ⟶
  (∀i<n. (label (∑i∈Basis. (real (x (b' i)) / real p) ∗R i) ∘ b) i = 0 ∨
        (label (∑i∈Basis. (real (x (b' i)) / real p) ∗R i) ∘ b) i = 1)
  unfolding ∗
  using ‹p > 0› ‹n > 0›
  using label(1)[OF b'']
  by auto
 { fix x :: nat ⇒ nat and i assume ∀i<n. x i ≤ p i < n x i = p ∨ x i = 0
  then have (∑i∈Basis. (real (x (b' i)) / real p) ∗R i) ∈ (unit-cube::'a set)
   using b'-Basis
   by (auto simp add: mem-unit-cube inner-simps bij-betw-def zero-le-divide-iff
divide-le-eq-1) }
 note cube = this
 have q2: ∀x. (∀i<n. x i ≤ p) ⟶ (∀i<n. x i = 0 ⟶
   (label (∑i∈Basis. (real (x (b' i)) / real p) ∗R i) ∘ b) i = 0)
  unfolding o-def using cube ‹p > 0› by (intro allI impI label(2)) (auto simp
add: b'')
```

have *q3*: $\forall x.\ (\forall i{<}n.\ x\ i \le p) \longrightarrow (\forall i{<}n.\ x\ i = p \longrightarrow$
$(label\ (\sum i{\in}Basis.\ (real\ (x\ (b'\ i))\ /\ real\ p) *_R i) \circ b)\ i = 1)$
using *cube* ‹$p > 0$› **unfolding** *o-def* **by** (*intro allI impI label*(*3*)) (*auto simp add*: *b''*)
obtain *q* **where** *q*:
$\forall i{<}n.\ q\ i < p$
$\forall i{<}n.$
$\exists r\ s.\ (\forall j{<}n.\ q\ j \le r\ j \wedge r\ j \le q\ j + 1) \wedge$
$(\forall j{<}n.\ q\ j \le s\ j \wedge s\ j \le q\ j + 1) \wedge$
$(label\ (\sum i{\in}Basis.\ (real\ (r\ (b'\ i))\ /\ real\ p) *_R i) \circ b)\ i \neq$
$(label\ (\sum i{\in}Basis.\ (real\ (s\ (b'\ i))\ /\ real\ p) *_R i) \circ b)\ i$
by (*rule kuhn-lemma*[*OF q1 q2 q3*])
def $z \equiv (\sum i{\in}Basis.\ (real\ (q\ (b'\ i))\ /\ real\ p) *_R i)::'a$
have $\exists i{\in}Basis.\ d\ /\ real\ n \le |(f\ z - z){\cdot}i|$
proof (*rule ccontr*)
have $\forall i{\in}Basis.\ q\ (b'\ i) \in \{0..p\}$
using *q*(*1*) *b'*
by (*auto intro*: *less-imp-le simp*: *bij-betw-def*)
then have $z \in$ *unit-cube*
unfolding *z-def mem-unit-cube*
using *b'-Basis*
by (*auto simp add*: *bij-betw-def zero-le-divide-iff divide-le-eq-1*)
then have *d-fz-z*: $d \le norm\ (f\ z - z)$
by (*rule d*)
assume ¬ *?thesis*
then have *as*: $\forall i{\in}Basis.\ |f\ z \cdot i - z \cdot i| < d\ /\ real\ n$
using ‹$n > 0$›
by (*auto simp add*: *not-le inner-diff*)
have $norm\ (f\ z - z) \le (\sum i{\in}Basis.\ |f\ z \cdot i - z \cdot i|)$
unfolding *inner-diff-left*[*symmetric*]
by (*rule norm-le-l1*)
also have $\ldots < (\sum (i::'a) \in Basis.\ d\ /\ real\ n)$
apply (*rule setsum-strict-mono*)
using *as*
apply *auto*
done
also have $\ldots = d$
using *DIM-positive*[**where** *'a='a*]
by (*auto simp*: *n-def*)
finally show *False*
using *d-fz-z* **by** *auto*
qed
then obtain *i* **where** *i*: $i \in Basis\ d\ /\ real\ n \le |(f\ z - z) \cdot i|$ **..**
have $*$: $b'\ i < n$
using *i* **and** *b'*[*unfolded bij-betw-def*]
by *auto*
obtain *r s* **where** *rs*:
$\bigwedge j.\ j < n \Longrightarrow q\ j \le r\ j \wedge r\ j \le q\ j + 1$
$\bigwedge j.\ j < n \Longrightarrow q\ j \le s\ j \wedge s\ j \le q\ j + 1$

```
  (label (∑ i∈Basis. (real (r (b' i)) / real p) *R i) ∘ b) (b' i) ≠
    (label (∑ i∈Basis. (real (s (b' i)) / real p) *R i) ∘ b) (b' i)
  using q(2)[rule-format,OF *] by blast
have b'-im: ⋀i. i ∈ Basis ⟹ b' i < n
  using b' unfolding bij-betw-def by auto
def r' ≡ (∑ i∈Basis. (real (r (b' i)) / real p) *R i)::'a
have ⋀i. i ∈ Basis ⟹ r (b' i) ≤ p
  apply (rule order-trans)
  apply (rule rs(1)[OF b'-im,THEN conjunct2])
  using q(1)[rule-format,OF b'-im]
  apply (auto simp add: Suc-le-eq)
  done
then have r' ∈ unit-cube
  unfolding r'-def mem-unit-cube
  using b'-Basis
  by (auto simp add: bij-betw-def zero-le-divide-iff divide-le-eq-1)
def s' ≡ (∑ i∈Basis. (real (s (b' i)) / real p) *R i)::'a
have ⋀i. i ∈ Basis ⟹ s (b' i) ≤ p
  apply (rule order-trans)
  apply (rule rs(2)[OF b'-im, THEN conjunct2])
  using q(1)[rule-format,OF b'-im]
  apply (auto simp add: Suc-le-eq)
  done
then have s' ∈ unit-cube
  unfolding s'-def mem-unit-cube
  using b'-Basis
  by (auto simp add: bij-betw-def zero-le-divide-iff divide-le-eq-1)
have z ∈ unit-cube
  unfolding z-def mem-unit-cube
  using b'-Basis q(1)[rule-format,OF b'-im] ‹p > 0›
  by (auto simp add: bij-betw-def zero-le-divide-iff divide-le-eq-1 less-imp-le)
have *: ⋀x. 1 + real x = real (Suc x)
  by auto
{
  have (∑ i∈Basis. |real (r (b' i)) − real (q (b' i))|) ≤ (∑ (i::'a)∈Basis. 1)
    apply (rule setsum-mono)
    using rs(1)[OF b'-im]
    apply (auto simp add:* field-simps simp del: of-nat-Suc)
    done
  also have ... < e * real p
    using p ‹e > 0› ‹p > 0›
    by (auto simp add: field-simps n-def)
  finally have (∑ i∈Basis. |real (r (b' i)) − real (q (b' i))|) < e * real p .
}
moreover
{
  have (∑ i∈Basis. |real (s (b' i)) − real (q (b' i))|) ≤ (∑ (i::'a)∈Basis. 1)
    apply (rule setsum-mono)
    using rs(2)[OF b'-im]
```

apply (*auto simp add*:∗ *field-simps simp del*: *of-nat-Suc*)
done
also have ... $<$ e ∗ *real p*
using *p* ‹$e > 0$› ‹$p > 0$›
by (*auto simp add*: *field-simps n-def*)
finally have ($\sum i{\in}Basis.$ $|real\ (s\ (b'\ i)) - real\ (q\ (b'\ i))|$) $<$ e ∗ *real p* **.**
}
ultimately
have *norm* $(r' - z) < e$ **and** *norm* $(s' - z) < e$
unfolding *r′-def s′-def z-def*
using ‹$p > 0$›
apply (*rule-tac*[!] *le-less-trans*[*OF norm-le-l1*])
apply (*auto simp add*: *field-simps setsum-divide-distrib*[*symmetric*] *inner-diff-left*)
done
then have $|(f\ z - z) \cdot i| < d\ /$ *real n*
using *rs*(*3*) *i*
unfolding *r′-def*[*symmetric*] *s′-def*[*symmetric*] *o-def bb′*
by (*intro e*(*2*)[*OF* ‹$r'{\in}unit\text{-}cube$› ‹$s'{\in}unit\text{-}cube$› ‹$z{\in}unit\text{-}cube$›]) *auto*
then show *False*
using *i* **by** *auto*
qed

20.5 Retractions

definition *retraction s t r* $\longleftrightarrow$ $t \subseteq s \wedge$ *continuous-on s r* $\wedge$ r ‘ $s \subseteq t \wedge (\forall x{\in}t.\ r\ x = x)$

definition *retract-of* (**infixl** *retract′-of 12*)
where (*t retract-of s*) $\longleftrightarrow$ ($\exists r.$ *retraction s t r*)

lemma *retraction-idempotent*: *retraction s t r* $\Longrightarrow$ $x \in s \Longrightarrow$ $r\ (r\ x) = r\ x$
unfolding *retraction-def* **by** *auto*

20.6 Preservation of fixpoints under (more general notion of) retraction

lemma *invertible-fixpoint-property*:
fixes s :: *′a*::*euclidean-space set*
and t :: *′b*::*euclidean-space set*
assumes *continuous-on t i*
and i ‘ $t \subseteq s$
and *continuous-on s r*
and r ‘ $s \subseteq t$
and $\forall y{\in}t.\ r\ (i\ y) = y$
and $\forall f.$ *continuous-on s f* $\wedge$ f ‘ $s \subseteq s \longrightarrow (\exists x{\in}s.\ f\ x = x)$
and *continuous-on t g*
and g ‘ $t \subseteq t$
obtains y **where** $y \in t$ **and** $g\ y = y$
proof −

```
  have ∃x∈s. (i ∘ g ∘ r) x = x
    apply (rule assms(6)[rule-format])
    apply rule
    apply (rule continuous-on-compose assms)+
    apply ((rule continuous-on-subset)?, rule assms)+
    using assms(2,4,8)
    apply auto
    apply blast
    done
  then obtain x where x: x ∈ s (i ∘ g ∘ r) x = x ..
  then have ∗: g (r x) ∈ t
    using assms(4,8) by auto
  have r ((i ∘ g ∘ r) x) = r x
    using x by auto
  then show ?thesis
    apply (rule-tac that[of r x])
    using x
    unfolding o-def
    unfolding assms(5)[rule-format,OF ∗]
    using assms(4)
    apply auto
    done
qed

lemma homeomorphic-fixpoint-property:
  fixes s :: 'a::euclidean-space set
    and t :: 'b::euclidean-space set
  assumes s homeomorphic t
  shows (∀f. continuous-on s f ∧ f ‘ s ⊆ s ⟶ (∃x∈s. f x = x)) ⟷
    (∀g. continuous-on t g ∧ g ‘ t ⊆ t ⟶ (∃y∈t. g y = y))
proof −
  obtain r i where
      ∀x∈s. i (r x) = x
      r ‘ s = t
      continuous-on s r
      ∀y∈t. r (i y) = y
      i ‘ t = s
      continuous-on t i
    using assms
    unfolding homeomorphic-def homeomorphism-def
    by blast
  then show ?thesis
    apply −
    apply rule
    apply (rule-tac[!] allI impI)+
    apply (rule-tac g=g in invertible-fixpoint-property[of t i s r])
    prefer 10
    apply (rule-tac g=f in invertible-fixpoint-property[of s r t i])
    apply auto
```

done
qed

lemma *retract-fixpoint-property*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
and *s* :: *'a set*
assumes *t retract-of s*
and ∀*f*. *continuous-on s f* ∧ *f* ' *s* ⊆ *s* ⟶ (∃ *x*∈*s*. *f x* = *x*)
and *continuous-on t g*
and *g* ' *t* ⊆ *t*
obtains *y* **where** *y* ∈ *t* **and** *g y* = *y*
proof −
obtain *h* **where** *retraction s t h*
using *assms*(*1*) **unfolding** *retract-of-def* **..**
then show *?thesis*
unfolding *retraction-def*
apply −
apply (*rule invertible-fixpoint-property*[*OF continuous-on-id - - - - assms*(*2*), *of t h g*])
prefer *7*
apply (*rule-tac y* = *y* **in** *that*)
using *assms*
apply *auto*
done
qed

20.7 The Brouwer theorem for any set with nonempty interior

lemma *convex-unit-cube*: *convex unit-cube*
apply (*rule is-interval-convex*)
apply (*clarsimp simp add*: *is-interval-def mem-unit-cube*)
apply (*drule* (*1*) *bspec*)+
apply *auto*
done

lemma *brouwer-weak*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'a*
assumes *compact s*
and *convex s*
and *interior s* ≠ {}
and *continuous-on s f*
and *f* ' *s* ⊆ *s*
obtains *x* **where** *x* ∈ *s* **and** *f x* = *x*
proof −
let *?U* = *unit-cube* :: *'a set*
have ∑ *Basis* /$_R$ *2* ∈ *interior ?U*
proof (*rule interiorI*)
let *?I* = (⋂ *i*∈*Basis*. {*x*::*'a*. *0* < *x* ∙ *i*} ∩ {*x*. *x* ∙ *i* < *1*})

```
    show open ?I
      by (intro open-INT finite-Basis ballI open-Int, auto intro: open-Collect-less)
    show ∑ Basis /R 2 ∈ ?I
      by simp
    show ?I ⊆ unit-cube
      unfolding unit-cube-def by force
  qed
  then have ∗: interior ?U ≠ {} by fast
  have ∗: ?U homeomorphic s
    using homeomorphic-convex-compact[OF convex-unit-cube compact-unit-cube ∗
assms(2,1,3)] .
  have ∀f. continuous-on ?U f ∧ f ‘ ?U ⊆ ?U ⟶
    (∃x∈?U. f x = x)
    using brouwer-cube by auto
  then show ?thesis
    unfolding homeomorphic-fixpoint-property[OF ∗]
    using assms
    by (auto simp: intro: that)
qed
```

And in particular for a closed ball.

```
lemma brouwer-ball:
  fixes f :: 'a::euclidean-space ⇒ 'a
  assumes e > 0
    and continuous-on (cball a e) f
    and f ‘ cball a e ⊆ cball a e
  obtains x where x ∈ cball a e and f x = x
  using brouwer-weak[OF compact-cball convex-cball, of a e f]
  unfolding interior-cball ball-eq-empty
  using assms by auto
```

Still more general form; could derive this directly without using the rather involved *HOMEOMORPHIC-CONVEX-COMPACT* theorem, just using a scaling and translation to put the set inside the unit cube.

```
lemma brouwer:
  fixes f :: 'a::euclidean-space ⇒ 'a
  assumes compact s
    and convex s
    and s ≠ {}
    and continuous-on s f
    and f ‘ s ⊆ s
  obtains x where x ∈ s and f x = x
proof −
  have ∃e>0. s ⊆ cball 0 e
    using compact-imp-bounded[OF assms(1)]
    unfolding bounded-pos
    apply (erule-tac exE)
    apply (rule-tac x=b in exI)
    apply (auto simp add: dist-norm)
```

```
    done
  then obtain e where e: e > 0 s ⊆ cball 0 e
    by blast
  have ∃x∈ cball 0 e. (f ∘ closest-point s) x = x
    apply (rule-tac brouwer-ball[OF e(1), of 0 f ∘ closest-point s])
    apply (rule continuous-on-compose )
     apply (rule continuous-on-closest-point[OF assms(2) compact-imp-closed[OF
assms(1)] assms(3)])
    apply (rule continuous-on-subset[OF assms(4)])
   apply (insert closest-point-in-set[OF compact-imp-closed[OF assms(1)] assms(3)])
    using assms(5)[unfolded subset-eq]
    using e(2)[unfolded subset-eq mem-cball]
    apply (auto simp add: dist-norm)
    done
  then obtain x where x: x ∈ cball 0 e (f ∘ closest-point s) x = x ..
  have ∗: closest-point s x = x
    apply (rule closest-point-self)
     apply (rule assms(5)[unfolded subset-eq,THEN bspec[where x=x], unfolded
image-iff])
    apply (rule-tac x=closest-point s x in bexI)
    using x
    unfolding o-def
     using closest-point-in-set[OF compact-imp-closed[OF assms(1)] assms(3), of
x]
    apply auto
    done
  show thesis
    apply (rule-tac x=closest-point s x in that)
    unfolding x(2)[unfolded o-def]
   apply (rule closest-point-in-set[OF compact-imp-closed[OF assms(1)] assms(3)])
    using ∗
    apply auto
    done
qed
```

So we get the no-retraction theorem.

```
lemma no-retraction-cball:
  fixes a :: 'a::euclidean-space
  assumes e > 0
  shows ¬ (frontier (cball a e) retract-of (cball a e))
proof
  assume ∗: frontier (cball a e) retract-of (cball a e)
  have ∗∗: ⋀xa. a − (2 ∗R a − xa) = − (a − xa)
    using scaleR-left-distrib[of 1 1 a] by auto
  obtain x where x:
      x ∈ {x. norm (a − x) = e}
      2 ∗R a − x = x
    apply (rule retract-fixpoint-property[OF ∗, of λx. scaleR 2 a − x])
    apply (blast intro: brouwer-ball[OF assms])
```

apply (*intro continuous-intros*)
unfolding *frontier-cball subset-eq Ball-def image-iff dist-norm*
apply (*auto simp add*: ∗∗ *norm-minus-commute*)
done
then have *scaleR 2 a = scaleR 1 x + scaleR 1 x*
by (*auto simp add*: *algebra-simps*)
then have *a = x*
unfolding *scaleR-left-distrib*[*symmetric*]
by *auto*
then show *False*
using *x assms* **by** *auto*
qed

end

21 Continuous paths and path-connected sets

theory *Path-Connected*
imports *Convex-Euclidean-Space*
begin

21.1 Paths and Arcs

definition *path* :: (*real* ⇒ *'a*::*topological-space*) ⇒ *bool*
where *path g* ⟷ *continuous-on* {*0..1*} *g*

definition *pathstart* :: (*real* ⇒ *'a*::*topological-space*) ⇒ *'a*
where *pathstart g* = *g 0*

definition *pathfinish* :: (*real* ⇒ *'a*::*topological-space*) ⇒ *'a*
where *pathfinish g* = *g 1*

definition *path-image* :: (*real* ⇒ *'a*::*topological-space*) ⇒ *'a set*
where *path-image g* = *g* ‘ {*0* .. *1*}

definition *reversepath* :: (*real* ⇒ *'a*::*topological-space*) ⇒ *real* ⇒ *'a*
where *reversepath g* = (λ*x*. *g*(*1* − *x*))

definition *joinpaths* :: (*real* ⇒ *'a*::*topological-space*) ⇒ (*real* ⇒ *'a*) ⇒ *real* ⇒ *'a*
(**infixr** +++ *75*)
where *g1* +++ *g2* = (λ*x*. *if x* ≤ *1/2 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*))

definition *simple-path* :: (*real* ⇒ *'a*::*topological-space*) ⇒ *bool*
where *simple-path g* ⟷
path g ∧ (∀ *x*∈{*0..1*}. ∀ *y*∈{*0..1*}. *g x* = *g y* ⟶ *x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0*)

definition *arc* :: (*real* ⇒ *'a* :: *topological-space*) ⇒ *bool*
where *arc g* ⟷ *path g* ∧ *inj-on g* {*0..1*}

21.2 Invariance theorems

lemma *path-eq*: *path p* $\Longrightarrow$ $(\bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ t = q\ t) \Longrightarrow$ *path q*
using *continuous-on-eq path-def* **by** *blast*

lemma *path-continuous-image*: *path g* $\Longrightarrow$ *continuous-on* (*path-image g*) *f* $\Longrightarrow$ *path*(*f o g*)
unfolding *path-def path-image-def*
using *continuous-on-compose* **by** *blast*

lemma *path-translation-eq*:
fixes *g* :: *real* $\Rightarrow$ *'a* :: *real-normed-vector*
shows *path*$((\lambda x.\ a + x)\ o\ g) =$ *path g*
proof −
have *g*: $g = (\lambda x.\ -a + x)\ o\ ((\lambda x.\ a + x)\ o\ g)$
by (*rule ext*) *simp*
show *?thesis*
unfolding *path-def*
apply *safe*
apply (*subst g*)
apply (*rule continuous-on-compose*)
apply (*auto intro*: *continuous-intros*)
done
qed

lemma *path-linear-image-eq*:
fixes *f* :: *'a*::*euclidean-space* $\Rightarrow$ *'b*::*euclidean-space*
assumes *linear f inj f*
shows *path*(*f o g*) = *path g*
proof −
from *linear-injective-left-inverse* [*OF assms*]
obtain *h* **where** *h*: *linear h h* $\circ$ *f* = *id*
by *blast*
then have *g*: $g = h\ o\ (f\ o\ g)$
by (*metis comp-assoc id-comp*)
show *?thesis*
unfolding *path-def*
using *h assms*
by (*metis g continuous-on-compose linear-continuous-on linear-conv-bounded-linear*)
qed

lemma *pathstart-translation*: *pathstart*$((\lambda x.\ a + x)\ o\ g) = a +$ *pathstart g*
by (*simp add*: *pathstart-def*)

lemma *pathstart-linear-image-eq*: *linear f* $\Longrightarrow$ *pathstart*(*f o g*) = *f*(*pathstart g*)
by (*simp add*: *pathstart-def*)

lemma *pathfinish-translation*: *pathfinish*$((\lambda x.\ a + x)\ o\ g) = a +$ *pathfinish g*
by (*simp add*: *pathfinish-def*)

lemma *pathfinish-linear-image*: *linear f* $\Longrightarrow$ *pathfinish*(*f o g*) = *f*(*pathfinish g*)
by (*simp add*: *pathfinish-def*)

lemma *path-image-translation*: *path-image*((λ*x*. *a* + *x*) *o g*) = (λ*x*. *a* + *x*) ‘ (*path-image g*)
by (*simp add*: *image-comp path-image-def*)

lemma *path-image-linear-image*: *linear f* $\Longrightarrow$ *path-image*(*f o g*) = *f* ‘ (*path-image g*)
by (*simp add*: *image-comp path-image-def*)

lemma *reversepath-translation*: *reversepath*((λ*x*. *a* + *x*) *o g*) = (λ*x*. *a* + *x*) *o reversepath g*
by (*rule ext*) (*simp add*: *reversepath-def*)

lemma *reversepath-linear-image*: *linear f* $\Longrightarrow$ *reversepath*(*f o g*) = *f o reversepath g*
by (*rule ext*) (*simp add*: *reversepath-def*)

lemma *joinpaths-translation*:
((λ*x*. *a* + *x*) *o g1*) +++ ((λ*x*. *a* + *x*) *o g2*) = (λ*x*. *a* + *x*) *o* (*g1* +++ *g2*)
by (*rule ext*) (*simp add*: *joinpaths-def*)

lemma *joinpaths-linear-image*: *linear f* $\Longrightarrow$ (*f o g1*) +++ (*f o g2*) = *f o* (*g1* +++ *g2*)
by (*rule ext*) (*simp add*: *joinpaths-def*)

lemma *simple-path-translation-eq*:
fixes *g* :: *real* ⇒ *'a*::*euclidean-space*
shows *simple-path*((λ*x*. *a* + *x*) *o g*) = *simple-path g*
by (*simp add*: *simple-path-def path-translation-eq*)

lemma *simple-path-linear-image-eq*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *linear f inj f*
shows *simple-path*(*f o g*) = *simple-path g*
using *assms inj-on-eq-iff* [*of f*]
by (*auto simp*: *path-linear-image-eq simple-path-def path-translation-eq*)

lemma *arc-translation-eq*:
fixes *g* :: *real* ⇒ *'a*::*euclidean-space*
shows *arc*((λ*x*. *a* + *x*) *o g*) = *arc g*
by (*auto simp*: *arc-def inj-on-def path-translation-eq*)

lemma *arc-linear-image-eq*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *linear f inj f*
shows *arc*(*f o g*) = *arc g*
using *assms inj-on-eq-iff* [*of f*]

by (*auto simp*: *arc-def inj-on-def path-linear-image-eq*)

21.3 Basic lemmas about paths

lemma *arc-imp-simple-path*: *arc g* $\Longrightarrow$ *simple-path g*
by (*simp add*: *arc-def inj-on-def simple-path-def*)

lemma *arc-imp-path*: *arc g* $\Longrightarrow$ *path g*
using *arc-def* **by** *blast*

lemma *simple-path-imp-path*: *simple-path g* $\Longrightarrow$ *path g*
using *simple-path-def* **by** *blast*

lemma *simple-path-cases*: *simple-path g* $\Longrightarrow$ *arc g* $\vee$ *pathfinish g* = *pathstart g*
unfolding *simple-path-def arc-def inj-on-def pathfinish-def pathstart-def*
by (*force*)

lemma *simple-path-imp-arc*: *simple-path g* $\Longrightarrow$ *pathfinish g* $\neq$ *pathstart g* $\Longrightarrow$ *arc g*
using *simple-path-cases* **by** *auto*

lemma *arc-distinct-ends*: *arc g* $\Longrightarrow$ *pathfinish g* $\neq$ *pathstart g*
unfolding *arc-def inj-on-def pathfinish-def pathstart-def*
by *fastforce*

lemma *arc-simple-path*: *arc g* $\longleftrightarrow$ *simple-path g* $\wedge$ *pathfinish g* $\neq$ *pathstart g*
using *arc-distinct-ends arc-imp-simple-path simple-path-cases* **by** *blast*

lemma *simple-path-eq-arc*: *pathfinish g* $\neq$ *pathstart g* $\Longrightarrow$ (*simple-path g* = *arc g*)
by (*simp add*: *arc-simple-path*)

lemma *path-image-nonempty* [*simp*]: *path-image g* $\neq$ {}
unfolding *path-image-def image-is-empty box-eq-empty*
by *auto*

lemma *pathstart-in-path-image*[*intro*]: *pathstart g* $\in$ *path-image g*
unfolding *pathstart-def path-image-def*
by *auto*

lemma *pathfinish-in-path-image*[*intro*]: *pathfinish g* $\in$ *path-image g*
unfolding *pathfinish-def path-image-def*
by *auto*

lemma *connected-path-image*[*intro*]: *path g* $\Longrightarrow$ *connected* (*path-image g*)
unfolding *path-def path-image-def*
using *connected-continuous-image connected-Icc* **by** *blast*

lemma *compact-path-image*[*intro*]: *path g* $\Longrightarrow$ *compact* (*path-image g*)
unfolding *path-def path-image-def*

using *compact-continuous-image connected-Icc* **by** *blast*

lemma *reversepath-reversepath*[*simp*]: *reversepath* (*reversepath g*) = *g*
unfolding *reversepath-def*
by *auto*

lemma *pathstart-reversepath*[*simp*]: *pathstart* (*reversepath g*) = *pathfinish g*
unfolding *pathstart-def reversepath-def pathfinish-def*
by *auto*

lemma *pathfinish-reversepath*[*simp*]: *pathfinish* (*reversepath g*) = *pathstart g*
unfolding *pathstart-def reversepath-def pathfinish-def*
by *auto*

lemma *pathstart-join*[*simp*]: *pathstart* (*g1* +++ *g2*) = *pathstart g1*
unfolding *pathstart-def joinpaths-def pathfinish-def*
by *auto*

lemma *pathfinish-join*[*simp*]: *pathfinish* (*g1* +++ *g2*) = *pathfinish g2*
unfolding *pathstart-def joinpaths-def pathfinish-def*
by *auto*

lemma *path-image-reversepath*[*simp*]: *path-image* (*reversepath g*) = *path-image g*
proof −
have ∗: ⋀*g*. *path-image* (*reversepath g*) ⊆ *path-image g*
unfolding *path-image-def subset-eq reversepath-def Ball-def image-iff*
by *force*
show *?thesis*
using ∗[*of g*] ∗[*of reversepath g*]
unfolding *reversepath-reversepath*
by *auto*
qed

lemma *path-reversepath* [*simp*]: *path* (*reversepath g*) ⟷ *path g*
proof −
have ∗: ⋀*g*. *path g* ⟹ *path* (*reversepath g*)
unfolding *path-def reversepath-def*
apply (*rule continuous-on-compose*[*unfolded o-def*, *of* - λ*x*. *1* − *x*])
apply (*intro continuous-intros*)
apply (*rule continuous-on-subset*[*of* {*0..1*}])
apply *assumption*
apply *auto*
done
show *?thesis*
using ∗[*of reversepath g*] ∗[*of g*]
unfolding *reversepath-reversepath*
by (*rule iffI*)
qed

lemma *arc-reversepath*:
assumes *arc g* **shows** *arc*(*reversepath g*)
proof −
have *injg*: *inj-on g* {*0..1*}
using *assms*
by (*simp add*: *arc-def*)
have ∗∗: ⋀*x y*::*real*. *1*−*x* = *1*−*y* ⟹ *x* = *y*
by *simp*
show *?thesis*
apply (*auto simp*: *arc-def inj-on-def path-reversepath*)
apply (*simp add*: *arc-imp-path assms*)
apply (*rule* ∗∗)
apply (*rule inj-onD* [*OF injg*])
apply (*auto simp*: *reversepath-def*)
done
qed

lemma *simple-path-reversepath*: *simple-path g* ⟹ *simple-path* (*reversepath g*)
apply (*simp add*: *simple-path-def*)
apply (*force simp*: *reversepath-def*)
done

lemmas *reversepath-simps* =
path-reversepath path-image-reversepath pathstart-reversepath pathfinish-reversepath

lemma *path-join*[*simp*]:
assumes *pathfinish g1* = *pathstart g2*
shows *path* (*g1* +++ *g2*) ⟷ *path g1* ∧ *path g2*
unfolding *path-def pathfinish-def pathstart-def*
proof *safe*
assume *cont*: *continuous-on* {*0..1*} (*g1* +++ *g2*)
have *g1*: *continuous-on* {*0..1*} *g1* ⟷ *continuous-on* {*0..1*} ((*g1* +++ *g2*) ∘ (λ*x*. *x* / *2*))
by (*intro continuous-on-cong refl*) (*auto simp*: *joinpaths-def*)
have *g2*: *continuous-on* {*0..1*} *g2* ⟷ *continuous-on* {*0..1*} ((*g1* +++ *g2*) ∘ (λ*x*. *x* / *2* + *1*/*2*))
using *assms*
by (*intro continuous-on-cong refl*) (*auto simp*: *joinpaths-def pathfinish-def pathstart-def*)
show *continuous-on* {*0..1*} *g1* **and** *continuous-on* {*0..1*} *g2*
unfolding *g1 g2*
by (*auto intro*!: *continuous-intros continuous-on-subset*[*OF cont*] *simp del*: *o-apply*)
next
assume *g1g2*: *continuous-on* {*0..1*} *g1 continuous-on* {*0..1*} *g2*
have *01*: {*0* .. *1*} = {*0..1*/*2*} ∪ {*1*/*2* .. *1*::*real*}
by *auto*
{
fix *x* :: *real*

```
    assume 0 ≤ x and x ≤ 1
    then have x ∈ (λx. x * 2) ` {0..1 / 2}
      by (intro image-eqI[where x=x/2]) auto
  }
  note 1 = this
  {
    fix x :: real
    assume 0 ≤ x and x ≤ 1
    then have x ∈ (λx. x * 2 − 1) ` {1 / 2..1}
      by (intro image-eqI[where x=x/2 + 1/2]) auto
  }
  note 2 = this
  show continuous-on {0..1} (g1 +++ g2)
    using assms
    unfolding joinpaths-def 01
   apply (intro continuous-on-cases closed-atLeastAtMost g1g2[THEN continuous-on-compose2]
continuous-intros)
    apply (auto simp: field-simps pathfinish-def pathstart-def intro!: 1 2)
    done
qed
```

22 Path Images

```
lemma bounded-path-image: path g ⟹ bounded(path-image g)
  by (simp add: compact-imp-bounded compact-path-image)

lemma closed-path-image:
  fixes g :: real ⇒ 'a::t2-space
  shows path g ⟹ closed(path-image g)
  by (metis compact-path-image compact-imp-closed)

lemma connected-simple-path-image: simple-path g ⟹ connected(path-image g)
  by (metis connected-path-image simple-path-imp-path)

lemma compact-simple-path-image: simple-path g ⟹ compact(path-image g)
  by (metis compact-path-image simple-path-imp-path)

lemma bounded-simple-path-image: simple-path g ⟹ bounded(path-image g)
  by (metis bounded-path-image simple-path-imp-path)

lemma closed-simple-path-image:
  fixes g :: real ⇒ 'a::t2-space
  shows simple-path g ⟹ closed(path-image g)
  by (metis closed-path-image simple-path-imp-path)

lemma connected-arc-image: arc g ⟹ connected(path-image g)
  by (metis connected-path-image arc-imp-path)

lemma compact-arc-image: arc g ⟹ compact(path-image g)
```

by (*metis compact-path-image arc-imp-path*)

lemma *bounded-arc-image*: *arc g* $\Longrightarrow$ *bounded*(*path-image g*)
by (*metis bounded-path-image arc-imp-path*)

lemma *closed-arc-image*:
fixes *g* :: *real* $\Rightarrow$ *'a*::*t2-space*
shows *arc g* $\Longrightarrow$ *closed*(*path-image g*)
by (*metis closed-path-image arc-imp-path*)

lemma *path-image-join-subset*: *path-image* (*g1* +++ *g2*) $\subseteq$ *path-image g1* $\cup$ *path-image g2*
unfolding *path-image-def joinpaths-def*
by *auto*

lemma *subset-path-image-join*:
assumes *path-image g1* $\subseteq$ *s*
and *path-image g2* $\subseteq$ *s*
shows *path-image* (*g1* +++ *g2*) $\subseteq$ *s*
using *path-image-join-subset*[*of g1 g2*] **and** *assms*
by *auto*

lemma *path-image-join*:
pathfinish g1 = *pathstart g2* $\Longrightarrow$ *path-image*(*g1* +++ *g2*) = *path-image g1* $\cup$ *path-image g2*
apply (*rule subset-antisym* [*OF path-image-join-subset*])
apply (*auto simp*: *pathfinish-def pathstart-def path-image-def joinpaths-def image-def*)
apply (*drule sym*)
apply (*rule-tac x=xa/2* **in** *bexI*, *auto*)
apply (*rule ccontr*)
apply (*drule-tac x=(xa+1)/2* **in** *bspec*)
apply (*auto simp*: *field-simps*)
apply (*drule-tac x=1/2* **in** *bspec*, *auto*)
done

lemma *not-in-path-image-join*:
assumes *x* $\notin$ *path-image g1*
and *x* $\notin$ *path-image g2*
shows *x* $\notin$ *path-image* (*g1* +++ *g2*)
using *assms* **and** *path-image-join-subset*[*of g1 g2*]
by *auto*

lemma *pathstart-compose*: *pathstart*(*f o p*) = *f*(*pathstart p*)
by (*simp add*: *pathstart-def*)

lemma *pathfinish-compose*: *pathfinish*(*f o p*) = *f*(*pathfinish p*)
by (*simp add*: *pathfinish-def*)

lemma *path-image-compose*: *path-image* (*f o p*) = *f* ' (*path-image p*)

by (*simp add*: *image-comp path-image-def*)

lemma *path-compose-join*: *f o* (*p* +++ *q*) = (*f o p*) +++ (*f o q*)
by (*rule ext*) (*simp add*: *joinpaths-def*)

lemma *path-compose-reversepath*: *f o reversepath p* = *reversepath*(*f o p*)
by (*rule ext*) (*simp add*: *reversepath-def*)

lemma *joinpaths-eq*:
(⋀*t*. *t* ∈ {*0..1*} ⟹ *p t* = *p*′ *t*) ⟹
(⋀*t*. *t* ∈ {*0..1*} ⟹ *q t* = *q*′ *t*)
⟹ *t* ∈ {*0..1*} ⟹ (*p* +++ *q*) *t* = (*p*′ +++ *q*′) *t*
by (*auto simp*: *joinpaths-def*)

lemma *simple-path-inj-on*: *simple-path g* ⟹ *inj-on g* {*0*<*..*<*1*}
by (*auto simp*: *simple-path-def path-image-def inj-on-def less-eq-real-def Ball-def*)

22.1 Simple paths with the endpoints removed

lemma *simple-path-endless*:
simple-path c ⟹ *path-image c* − {*pathstart c*,*pathfinish c*} = *c* ‘ {*0*<*..*<*1*}
apply (*auto simp*: *simple-path-def path-image-def pathstart-def pathfinish-def Ball-def Bex-def image-def*)
apply (*metis eq-iff le-less-linear*)
apply (*metis leD linear*)
using *less-eq-real-def zero-le-one* **apply** *blast*
using *less-eq-real-def zero-le-one* **apply** *blast*
done

lemma *connected-simple-path-endless*:
simple-path c ⟹ *connected*(*path-image c* − {*pathstart c*,*pathfinish c*})
apply (*simp add*: *simple-path-endless*)
apply (*rule connected-continuous-image*)
apply (*meson continuous-on-subset greaterThanLessThan-subseteq-atLeastAtMost-iff le-numeral-extra*(*3*) *le-numeral-extra*(*4*) *path-def simple-path-imp-path*)
by *auto*

lemma *nonempty-simple-path-endless*:
simple-path c ⟹ *path-image c* − {*pathstart c*,*pathfinish c*} ≠ {}
by (*simp add*: *simple-path-endless*)

22.2 The operations on paths

lemma *path-image-subset-reversepath*: *path-image*(*reversepath g*) ≤ *path-image g*
by (*auto simp*: *path-image-def reversepath-def*)

lemma *path-imp-reversepath*: *path g* ⟹ *path*(*reversepath g*)
apply (*auto simp*: *path-def reversepath-def*)
using *continuous-on-compose* [*of* {*0..1*} λ*x*. *1* − *x g*]
apply (*auto simp*: *continuous-on-op-minus*)

done

lemma *half-bounded-equal*: $1 \leq x * 2 \Longrightarrow x * 2 \leq 1 \longleftrightarrow x = (1/2::real)$
by *simp*

lemma *continuous-on-joinpaths*:
assumes *continuous-on* $\{0..1\}$ *g1 continuous-on* $\{0..1\}$ *g2 pathfinish g1 = pathstart g2*
shows *continuous-on* $\{0..1\}$ $(g1 +++ g2)$
proof −
have $*$: $\{0..1::real\} = \{0..1/2\} \cup \{1/2..1\}$
by *auto*
have *gg*: *g2 0 = g1 1*
by (*metis assms*(*3*) *pathfinish-def pathstart-def*)
have *1*: *continuous-on* $\{0..1/2\}$ $(g1 +++ g2)$
apply (*rule continuous-on-eq* [*of - g1 o* $(\lambda x.\ 2*x)$])
apply (*rule continuous-intros* | *simp add*: *joinpaths-def assms*)+
done
have *continuous-on* $\{1/2..1\}$ $(g2\ o\ (\lambda x.\ 2*x-1))$
apply (*rule continuous-on-subset* [*of* $\{1/2..1\}$])
apply (*rule continuous-intros* | *simp add*: *image-affinity-atLeastAtMost-diff assms*)+
done
then have *2*: *continuous-on* $\{1/2..1\}$ $(g1 +++ g2)$
apply (*rule continuous-on-eq* [*of* $\{1/2..1\}$ *g2 o* $(\lambda x.\ 2*x-1)$])
apply (*rule assms continuous-intros* | *simp add*: *joinpaths-def mult.commute half-bounded-equal gg*)+
done
show *?thesis*
apply (*subst* $*$)
apply (*rule continuous-on-union*)
using *1 2*
apply *auto*
done
qed

lemma *path-join-imp*: $[\![path\ g1; path\ g2; pathfinish\ g1 = pathstart\ g2]\!] \Longrightarrow path(g1 +++ g2)$
by (*simp add*: *path-join*)

lemma *simple-path-join-loop*:
assumes *arc g1 arc g2*
pathfinish g1 = pathstart g2 pathfinish g2 = pathstart g1
path-image g1 $\cap$ *path-image g2* $\subseteq$ $\{pathstart\ g1, pathstart\ g2\}$
shows *simple-path*$(g1 +++ g2)$
proof −
have *injg1*: *inj-on g1* $\{0..1\}$
using *assms*
by (*simp add*: *arc-def*)

```
 have injg2: inj-on g2 {0..1}
  using assms
  by (simp add: arc-def)
 have g12: g1 1 = g2 0
  and g21: g2 1 = g1 0
  and sb:  g1 ` {0..1} ∩ g2 ` {0..1} ⊆ {g1 0, g2 0}
  using assms
  by (simp-all add: arc-def pathfinish-def pathstart-def path-image-def)
 { fix x and y::real
  assume xyI: x = 1 ⟶ y ≠ 0
    and xy: x ≤ 1 0 ≤ y  y * 2 ≤ 1 ¬ x * 2 ≤ 1 g2 (2 * x − 1) = g1 (2 * y)
  have g1im: g1 (2 * y) ∈ g1 ` {0..1} ∩ g2 ` {0..1}
   using xy
   apply simp
   apply (rule-tac x=2 * x − 1 in image-eqI, auto)
   done
  have False
   using subsetD [OF sb g1im] xy
   apply auto
   apply (drule inj-onD [OF injg1])
   using g21 [symmetric] xyI
   apply (auto dest: inj-onD [OF injg2])
   done
 } note ∗ = this
 { fix x and y::real
  assume xy: y ≤ 1 0 ≤ x ¬ y * 2 ≤ 1 x * 2 ≤ 1 g1 (2 * x) = g2 (2 * y − 1)
  have g1im: g1 (2 * x) ∈ g1 ` {0..1} ∩ g2 ` {0..1}
   using xy
   apply simp
   apply (rule-tac x=2 * x in image-eqI, auto)
   done
  have x = 0 ∧ y = 1
   using subsetD [OF sb g1im] xy
   apply auto
   apply (force dest: inj-onD [OF injg1])
   using  g21 [symmetric]
   apply (auto dest: inj-onD [OF injg2])
   done
 } note ∗∗ = this
 show ?thesis
  using assms
  apply (simp add: arc-def simple-path-def path-join, clarify)
  apply (simp add: joinpaths-def split: split-if-asm)
  apply (force dest: inj-onD [OF injg1])
  apply (metis ∗)
  apply (metis ∗∗)
  apply (force dest: inj-onD [OF injg2])
  done
qed
```

lemma *arc-join*:
assumes *arc g1 arc g2*
pathfinish g1 = pathstart g2
path-image g1 ∩ path-image g2 ⊆ {pathstart g2}
shows *arc(g1 +++ g2)*
proof −
have *injg1*: *inj-on g1 {0..1}*
using *assms*
by (*simp add*: *arc-def*)
have *injg2*: *inj-on g2 {0..1}*
using *assms*
by (*simp add*: *arc-def*)
have *g11*: *g1 1 = g2 0*
and *sb*: *g1 ‘ {0..1} ∩ g2 ‘ {0..1} ⊆ {g2 0}*
using *assms*
by (*simp-all add*: *arc-def pathfinish-def pathstart-def path-image-def*)
{ fix *x* **and** *y*::*real*
assume *xy*: *x ≤ 1 0 ≤ y y ∗ 2 ≤ 1 ¬ x ∗ 2 ≤ 1 g2 (2 ∗ x − 1) = g1 (2 ∗ y)*
have *g1im*: *g1 (2 ∗ y) ∈ g1 ‘ {0..1} ∩ g2 ‘ {0..1}*
using *xy*
apply *simp*
apply (*rule-tac x=2 ∗ x − 1* **in** *image-eqI*, *auto*)
done
have *False*
using *subsetD* [*OF sb g1im*] *xy*
by (*auto dest*: *inj-onD* [*OF injg2*])
} note *∗ = this*
show *?thesis*
apply (*simp add*: *arc-def inj-on-def*)
apply (*clarsimp simp add*: *arc-imp-path assms path-join*)
apply (*simp add*: *joinpaths-def split*: *split-if-asm*)
apply (*force dest*: *inj-onD* [*OF injg1*])
apply (*metis ∗*)
apply (*metis ∗*)
apply (*force dest*: *inj-onD* [*OF injg2*])
done
qed

lemma *reversepath-joinpaths*:
pathfinish g1 = pathstart g2 ⟹ reversepath(g1 +++ g2) = reversepath g2 +++ reversepath g1
unfolding *reversepath-def pathfinish-def pathstart-def joinpaths-def*
by (*rule ext*) (*auto simp*: *mult.commute*)

23 Choosing a subpath of an existing path

definition *subpath* :: *real ⇒ real ⇒ (real ⇒ 'a) ⇒ real ⇒ 'a*::*real-normed-vector*
where *subpath a b g ≡ λx. g((b − a) ∗ x + a)*

lemma *path-image-subpath-gen*:
fixes *g* :: - ⇒ *'a*::*real-normed-vector*
shows *path-image*(*subpath u v g*) = *g* ‘ (*closed-segment u v*)
apply (*simp add*: *closed-segment-real-eq path-image-def subpath-def*)
apply (*subst o-def* [*of g*, *symmetric*])
apply (*simp add*: *image-comp* [*symmetric*])
done

lemma *path-image-subpath*:
fixes *g* :: *real* ⇒ *'a*::*real-normed-vector*
shows *path-image*(*subpath u v g*) = (*if u* ≤ *v then g* ‘ {*u*..*v*} *else g* ‘ {*v*..*u*})
by (*simp add*: *path-image-subpath-gen closed-segment-eq-real-ivl*)

lemma *path-subpath* [*simp*]:
fixes *g* :: *real* ⇒ *'a*::*real-normed-vector*
assumes *path g u* ∈ {*0*..*1*} *v* ∈ {*0*..*1*}
shows *path*(*subpath u v g*)
proof −
have *continuous-on* {*0*..*1*} (*g o* (λ*x*. ((*v*−*u*) ∗ *x*+ *u*)))
apply (*rule continuous-intros* | *simp*)+
apply (*simp add*: *image-affinity-atLeastAtMost* [**where** *c*=*u*])
using *assms*
apply (*auto simp*: *path-def continuous-on-subset*)
done
then show *?thesis*
by (*simp add*: *path-def subpath-def*)
qed

lemma *pathstart-subpath* [*simp*]: *pathstart*(*subpath u v g*) = *g*(*u*)
by (*simp add*: *pathstart-def subpath-def*)

lemma *pathfinish-subpath* [*simp*]: *pathfinish*(*subpath u v g*) = *g*(*v*)
by (*simp add*: *pathfinish-def subpath-def*)

lemma *subpath-trivial* [*simp*]: *subpath 0 1 g* = *g*
by (*simp add*: *subpath-def*)

lemma *subpath-reversepath*: *subpath 1 0 g* = *reversepath g*
by (*simp add*: *reversepath-def subpath-def*)

lemma *reversepath-subpath*: *reversepath*(*subpath u v g*) = *subpath v u g*
by (*simp add*: *reversepath-def subpath-def algebra-simps*)

lemma *subpath-translation*: *subpath u v* ((λ*x*. *a* + *x*) *o g*) = (λ*x*. *a* + *x*) *o subpath u v g*
by (*rule ext*) (*simp add*: *subpath-def*)

lemma *subpath-linear-image*: *linear f* ⟹ *subpath u v* (*f o g*) = *f o subpath u v g*

by (*rule ext*) (*simp add*: *subpath-def*)

lemma *affine-ineq*:
fixes x :: *'a*::*linordered-idom*
assumes $x \leq 1$ $v \leq u$
shows $v + x * u \leq u + x * v$
proof −
have $(1-x)*(u-v) \geq 0$
using *assms* **by** *auto*
then show *?thesis*
by (*simp add*: *algebra-simps*)
qed

lemma *sum-le-prod1*:
fixes *a*::*real* **shows** $[\![a \leq 1; b \leq 1]\!] \Longrightarrow a + b \leq 1 + a * b$
by (*metis add.commute affine-ineq less-eq-real-def mult.right-neutral*)

lemma *simple-path-subpath-eq*:
simple-path(*subpath u v g*) $\longleftrightarrow$
path(*subpath u v g*) $\wedge$ $u \neq v$ $\wedge$
$(\forall x\ y.\ x \in$ *closed-segment u v* $\wedge$ $y \in$ *closed-segment u v* $\wedge$ $g\ x = g\ y$
$\longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u)$
(**is** *?lhs* = *?rhs*)
proof (*rule iffI*)
assume *?lhs*
then have *p*: *path* $(\lambda x.\ g\ ((v - u) * x + u))$
and *sim*: $(\bigwedge x\ y.\ [\![x \in \{0..1\}; y \in \{0..1\}; g\ ((v - u) * x + u) = g\ ((v - u) * y + u)]\!]$
$\Longrightarrow x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0)$
by (*auto simp*: *simple-path-def subpath-def*)
{ fix *x y*
assume $x \in$ *closed-segment u v* $y \in$ *closed-segment u v g* $x = g\ y$
then have $x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$
using *sim* [*of* $(x-u)/(v-u)$ $(y-u)/(v-u)$] *p*
by (*auto simp*: *closed-segment-real-eq image-affinity-atLeastAtMost divide-simps split*: *split-if-asm*)
} moreover
have *path*(*subpath u v g*) $\wedge$ $u \neq v$
using *sim* [*of 1/3 2/3*] *p*
by (*auto simp*: *subpath-def*)
ultimately show *?rhs*
by *metis*
next
assume *?rhs*
then
have *d1*: $\bigwedge x\ y.\ [\![g\ x = g\ y; u \leq x; x \leq v; u \leq y; y \leq v]\!] \Longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$
and *d2*: $\bigwedge x\ y.\ [\![g\ x = g\ y; v \leq x; x \leq u; v \leq y; y \leq u]\!] \Longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$

and *ne*: $u < v \vee v < u$
and *psp*: *path* (*subpath u v g*)
by (*auto simp*: *closed-segment-real-eq image-affinity-atLeastAtMost*)
have [*simp*]: $\bigwedge x.\ u + x * v = v + x * u \longleftrightarrow u{=}v \vee x{=}1$
by *algebra*
show *?lhs* **using** *psp ne*
unfolding *simple-path-def subpath-def*
by (*fastforce simp add*: *algebra-simps affine-ineq mult-left-mono crossproduct-eq dest*: *d1 d2*)
qed

lemma *arc-subpath-eq*:
arc(*subpath u v g*) $\longleftrightarrow$ *path*(*subpath u v g*) $\wedge$ $u{\neq}v$ $\wedge$ *inj-on g* (*closed-segment u v*)
(**is** *?lhs* = *?rhs*)
proof (*rule iffI*)
assume *?lhs*
then have *p*: *path* ($\lambda x.$ *g* ((*v* − *u*) * *x* + *u*))
and *sim*: ($\bigwedge x\ y.$ $[\![x{\in}\{0..1\};\ y{\in}\{0..1\};\ g\ ((v - u) * x + u) = g\ ((v - u) * y + u)]\!]$
$\Longrightarrow x = y$)
by (*auto simp*: *arc-def inj-on-def subpath-def*)
{ fix *x y*
assume $x \in$ *closed-segment u v* $y \in$ *closed-segment u v* $g\ x = g\ y$
then have $x = y$
using *sim* [*of* $(x{-}u)/(v{-}u)$ $(y{-}u)/(v{-}u)$] *p*
by (*force simp add*: *inj-on-def closed-segment-real-eq image-affinity-atLeastAtMost divide-simps*
split: *split-if-asm*)
} moreover
have *path*(*subpath u v g*) $\wedge$ $u{\neq}v$
using *sim* [*of 1/3 2/3*] *p*
by (*auto simp*: *subpath-def*)
ultimately show *?rhs*
unfolding *inj-on-def*
by *metis*
next
assume *?rhs*
then
have *d1*: $\bigwedge x\ y.\ [\![g\ x = g\ y;\ u \le x;\ x \le v;\ u \le y;\ y \le v]\!] \Longrightarrow x = y$
and *d2*: $\bigwedge x\ y.\ [\![g\ x = g\ y;\ v \le x;\ x \le u;\ v \le y;\ y \le u]\!] \Longrightarrow x = y$
and *ne*: $u < v \vee v < u$
and *psp*: *path* (*subpath u v g*)
by (*auto simp*: *inj-on-def closed-segment-real-eq image-affinity-atLeastAtMost*)
show *?lhs* **using** *psp ne*
unfolding *arc-def subpath-def inj-on-def*
by (*auto simp*: *algebra-simps affine-ineq mult-left-mono crossproduct-eq dest*: *d1 d2*)
qed

lemma *simple-path-subpath*:
assumes *simple-path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≠ *v*
shows *simple-path*(*subpath u v g*)
using *assms*
apply (*simp add*: *simple-path-subpath-eq simple-path-imp-path*)
apply (*simp add*: *simple-path-def closed-segment-real-eq image-affinity-atLeastAtMost*, *fastforce*)
done

lemma *arc-simple-path-subpath*:
⟦*simple-path g*; *u* ∈ {*0..1*}; *v* ∈ {*0..1*}; *g u* ≠ *g v*⟧ ⟹ *arc*(*subpath u v g*)
by (*force intro*: *simple-path-subpath simple-path-imp-arc*)

lemma *arc-subpath-arc*:
⟦*arc g*; *u* ∈ {*0..1*}; *v* ∈ {*0..1*}; *u* ≠ *v*⟧ ⟹ *arc*(*subpath u v g*)
by (*meson arc-def arc-imp-simple-path arc-simple-path-subpath inj-onD*)

lemma *arc-simple-path-subpath-interior*:
⟦*simple-path g*; *u* ∈ {*0..1*}; *v* ∈ {*0..1*}; *u* ≠ *v*; |*u*−*v*| < *1*⟧ ⟹ *arc*(*subpath u v g*)
apply (*rule arc-simple-path-subpath*)
apply (*force simp*: *simple-path-def*)+
done

lemma *path-image-subpath-subset*:
⟦*path g*; *u* ∈ {*0..1*}; *v* ∈ {*0..1*}⟧ ⟹ *path-image*(*subpath u v g*) ⊆ *path-image g*
apply (*simp add*: *closed-segment-real-eq image-affinity-atLeastAtMost path-image-subpath*)
apply (*auto simp*: *path-image-def*)
done

lemma *join-subpaths-middle*: *subpath* (*0*) ((*1* / *2*)) *p* +++ *subpath* ((*1* / *2*)) *1 p* = *p*
by (*rule ext*) (*simp add*: *joinpaths-def subpath-def divide-simps*)

23.1 There is a subpath to the frontier

lemma *subpath-to-frontier-explicit*:
fixes *S* :: *'a*::*metric-space set*
assumes *g*: *path g* **and** *pathfinish g* ∉ *S*
obtains *u* **where** *0* ≤ *u u* ≤ *1*
⋀*x*. *0* ≤ *x* ∧ *x* < *u* ⟹ *g x* ∈ *interior S*
(*g u* ∉ *interior S*) (*u* = *0* ∨ *g u* ∈ *closure S*)
proof −
have *gcon*: *continuous-on* {*0..1*} *g* **using** *g* **by** (*simp add*: *path-def*)
then have *com*: *compact* ({*0..1*} ∩ {*u*. *g u* ∈ *closure* (− *S*)})
apply (*simp add*: *Int-commute* [*of* {*0..1*}] *compact-eq-bounded-closed closed-vimage-Int* [*unfolded vimage-def*])

```
    using compact-eq-bounded-closed apply fastforce
    done
  have 1 ∈ {u. g u ∈ closure (− S)}
    using assms by (simp add: pathfinish-def closure-def)
  then have dis: {0..1} ∩ {u. g u ∈ closure (− S)} ≠ {}
    using atLeastAtMost-iff zero-le-one by blast
  then obtain u where 0 ≤ u u ≤ 1 and gu: g u ∈ closure (− S)
                and umin: ⋀t. ⟦0 ≤ t; t ≤ 1; g t ∈ closure (− S)⟧ ⟹ u ≤ t
    using compact-attains-inf [OF com dis] by fastforce
  then have umin': ⋀t. ⟦0 ≤ t; t ≤ 1; t < u⟧ ⟹  g t ∈ S
    using closure-def by fastforce
  { assume u ≠ 0
    then have u > 0 using ‹0 ≤ u› by auto
    { fix e::real assume e > 0
      obtain d where d>0 and d: ⋀x'. ⟦x' ∈ {0..1}; dist x' u < d⟧ ⟹ dist (g
x') (g u) < e
       using continuous-onD [OF gcon - ‹e > 0›] ‹0 ≤ -› ‹- ≤ 1› atLeastAtMost-iff
by auto
      have ∗: dist (max 0 (u − d / 2)) u < d
        using ‹0 ≤ u› ‹u ≤ 1› ‹d > 0› by (simp add: dist-real-def)
      have ∃ y∈S. dist y (g u) < e
        using ‹0 < u› ‹u ≤ 1› ‹d > 0›
        by (force intro: d [OF - ∗] umin')
    }
    then have g u ∈ closure S
      by (simp add: frontier-def closure-approachable)
  }
  then show ?thesis
    apply (rule-tac u=u in that)
    apply (auto simp: ‹0 ≤ u› ‹u ≤ 1› gu interior-closure umin)
    using ‹- ≤ 1› interior-closure umin apply fastforce
    done
qed

lemma subpath-to-frontier-strong:
    assumes g: path g and pathfinish g ∉ S
    obtains u where 0 ≤ u u ≤ 1 g u ∉ interior S
                    u = 0 ∨ (∀x. 0 ≤ x ∧ x < 1 ⟶ subpath 0 u g x ∈ interior S)
∧  g u ∈ closure S
proof −
  obtain u where 0 ≤ u u ≤ 1
            and gxin: ⋀x. 0 ≤ x ∧ x < u ⟹ g x ∈ interior S
            and gunot: (g u ∉ interior S) and u0: (u = 0 ∨ g u ∈ closure S)
    using subpath-to-frontier-explicit [OF assms] by blast
  show ?thesis
    apply (rule that [OF ‹0 ≤ u› ‹u ≤ 1›])
    apply (simp add: gunot)
    using ‹0 ≤ u› u0 by (force simp: subpath-def gxin)
qed
```

lemma *subpath-to-frontier*:
assumes *g*: *path g* **and** *g0*: *pathstart g* ∈ *closure S* **and** *g1*: *pathfinish g* ∉ *S*
obtains *u* **where** *0* ≤ *u u* ≤ *1 g u* ∈ *frontier S* (*path-image*(*subpath 0 u g*) − {*g u*}) ⊆ *interior S*
proof −
obtain *u* **where** *0* ≤ *u u* ≤ *1*
and *notin*: *g u* ∉ *interior S*
and *disj*: *u* = *0* ∨ (∀ *x*. *0* ≤ *x* ∧ *x* < *1* ⟶ *subpath 0 u g x* ∈ *interior S*) ∧ *g u* ∈ *closure S*
using *subpath-to-frontier-strong* [*OF g g1*] **by** *blast*
show *?thesis*
apply (*rule that* [*OF* ‹*0* ≤ *u*› ‹*u* ≤ *1*›])
apply (*metis DiffI disj frontier-def g0 notin pathstart-def*)
using ‹*0* ≤ *u*› *g0 disj*
apply (*simp add*: *path-image-subpath-gen*)
apply (*auto simp*: *closed-segment-eq-real-ivl pathstart-def pathfinish-def subpath-def*)
apply (*rename-tac y*)
apply (*drule-tac x*=*y*/*u* **in** *spec*)
apply (*auto split*: *split-if-asm*)
done
qed

lemma *exists-path-subpath-to-frontier*:
fixes *S* :: *'a*::*real-normed-vector set*
assumes *path g pathstart g* ∈ *closure S pathfinish g* ∉ *S*
obtains *h* **where** *path h pathstart h* = *pathstart g path-image h* ⊆ *path-image g*
path-image h − {*pathfinish h*} ⊆ *interior S*
pathfinish h ∈ *frontier S*
proof −
obtain *u* **where** *u*: *0* ≤ *u u* ≤ *1 g u* ∈ *frontier S* (*path-image*(*subpath 0 u g*) − {*g u*}) ⊆ *interior S*
using *subpath-to-frontier* [*OF assms*] **by** *blast*
show *?thesis*
apply (*rule that* [*of subpath 0 u g*])
using *assms u*
apply (*simp-all add*: *path-image-subpath*)
apply (*simp add*: *pathstart-def*)
apply (*force simp*: *closed-segment-eq-real-ivl path-image-def*)
done
qed

lemma *exists-path-subpath-to-frontier-closed*:
fixes *S* :: *'a*::*real-normed-vector set*
assumes *S*: *closed S* **and** *g*: *path g* **and** *g0*: *pathstart g* ∈ *S* **and** *g1*: *pathfinish g* ∉ *S*
obtains *h* **where** *path h pathstart h* = *pathstart g path-image h* ⊆ *path-image*

g ∩ *S*
pathfinish h ∈ *frontier S*
proof −
obtain *h* **where** *h*: *path h pathstart h* = *pathstart g path-image h* ⊆ *path-image g*
path-image h − {*pathfinish h*} ⊆ *interior S*
pathfinish h ∈ *frontier S*
using *exists-path-subpath-to-frontier* [*OF g - g1*] *closure-closed* [*OF S*] *g0* **by** *auto*
show *?thesis*
apply (*rule that* [*OF* ‹*path h*›])
using *assms h*
apply *auto*
apply (*metis Diff-single-insert frontier-subset-eq insert-iff interior-subset subset-iff*)
done
qed

23.2 Reparametrizing a closed curve to start at some chosen point

definition *shiftpath* :: *real* ⇒ (*real* ⇒ *'a*::*topological-space*) ⇒ *real* ⇒ *'a*
where *shiftpath a f* = (λ*x*. *if* (*a* + *x*) ≤ *1 then f* (*a* + *x*) *else f* (*a* + *x* − *1*))

lemma *pathstart-shiftpath*: *a* ≤ *1* ⟹ *pathstart* (*shiftpath a g*) = *g a*
unfolding *pathstart-def shiftpath-def* **by** *auto*

lemma *pathfinish-shiftpath*:
assumes *0* ≤ *a*
and *pathfinish g* = *pathstart g*
shows *pathfinish* (*shiftpath a g*) = *g a*
using *assms*
unfolding *pathstart-def pathfinish-def shiftpath-def*
by *auto*

lemma *endpoints-shiftpath*:
assumes *pathfinish g* = *pathstart g*
and *a* ∈ {*0* .. *1*}
shows *pathfinish* (*shiftpath a g*) = *g a*
and *pathstart* (*shiftpath a g*) = *g a*
using *assms*
by (*auto intro*!: *pathfinish-shiftpath pathstart-shiftpath*)

lemma *closed-shiftpath*:
assumes *pathfinish g* = *pathstart g*
and *a* ∈ {*0..1*}
shows *pathfinish* (*shiftpath a g*) = *pathstart* (*shiftpath a g*)
using *endpoints-shiftpath*[*OF assms*]
by *auto*

lemma *path-shiftpath*:
assumes *path g*
and *pathfinish g = pathstart g*
and $a \in \{0..1\}$
shows *path (shiftpath a g)*
proof −
have ∗: $\{0 .. 1\} = \{0 .. 1-a\} \cup \{1-a .. 1\}$
using *assms(3)* **by** *auto*
have ∗∗: $\bigwedge x.\ x + a = 1 \Longrightarrow g\ (x + a - 1) = g\ (x + a)$
using *assms(2)[unfolded pathfinish-def pathstart-def]*
by *auto*
show *?thesis*
unfolding *path-def shiftpath-def* ∗
apply (*rule continuous-on-union*)
apply (*rule closed-real-atLeastAtMost*)+
apply (*rule continuous-on-eq[of - g ∘ (λx. a + x)]*)
prefer *3*
apply (*rule continuous-on-eq[of - g ∘ (λx. a − 1 + x)]*)
prefer *3*
apply (*rule continuous-intros*)+
prefer *2*
apply (*rule continuous-intros*)+
apply (*rule-tac[1−2] continuous-on-subset[OF assms(1)[unfolded path-def]]*)
using *assms(3)* **and** ∗∗
apply *auto*
apply (*auto simp add*: *field-simps*)
done
qed

lemma *shiftpath-shiftpath*:
assumes *pathfinish g = pathstart g*
and $a \in \{0..1\}$
and $x \in \{0..1\}$
shows *shiftpath (1 − a) (shiftpath a g) x = g x*
using *assms*
unfolding *pathfinish-def pathstart-def shiftpath-def*
by *auto*

lemma *path-image-shiftpath*:
assumes $a \in \{0..1\}$
and *pathfinish g = pathstart g*
shows *path-image (shiftpath a g) = path-image g*
proof −
{ fix *x*
assume *as*: *g 1 = g 0* $x \in \{0..1::real\}$ $\forall\, y \in \{0..1\} \cap \{x.\ \neg\ a + x \leq 1\}.\ g\ x \neq g\ (a + y - 1)$
then have $\exists\, y \in \{0..1\} \cap \{x.\ a + x \leq 1\}.\ g\ x = g\ (a + y)$
proof (*cases* $a \leq x$)
case *False*

```
    then show ?thesis
      apply (rule-tac x=1 + x − a in bexI)
     using as(1,2) and as(3)[THEN bspec[where x=1 + x − a]] and assms(1)
      apply (auto simp add: field-simps atomize-not)
      done
  next
    case True
    then show ?thesis
      using as(1−2) and assms(1)
      apply (rule-tac x=x − a in bexI)
      apply (auto simp add: field-simps)
      done
  qed
 }
 then show ?thesis
  using assms
  unfolding shiftpath-def path-image-def pathfinish-def pathstart-def
  by (auto simp add: image-iff)
qed
```

23.3 Special case of straight-line paths

```
definition linepath :: 'a::real-normed-vector ⇒ 'a ⇒ real ⇒ 'a
 where linepath a b = (λx. (1 − x) *R a + x *R b)

lemma pathstart-linepath[simp]: pathstart (linepath a b) = a
 unfolding pathstart-def linepath-def
 by auto

lemma pathfinish-linepath[simp]: pathfinish (linepath a b) = b
 unfolding pathfinish-def linepath-def
 by auto

lemma continuous-linepath-at[intro]: continuous (at x) (linepath a b)
 unfolding linepath-def
 by (intro continuous-intros)

lemma continuous-on-linepath [intro,continuous-intros]: continuous-on s (linepath
a b)
 using continuous-linepath-at
 by (auto intro!: continuous-at-imp-continuous-on)

lemma path-linepath[intro]: path (linepath a b)
 unfolding path-def
 by (rule continuous-on-linepath)

lemma path-image-linepath[simp]: path-image (linepath a b) = closed-segment a b
 unfolding path-image-def segment linepath-def
 by auto
```

lemma *reversepath-linepath*[*simp*]: *reversepath* (*linepath a b*) = *linepath b a*
unfolding *reversepath-def linepath-def*
by *auto*

lemma *linepath-0* [*simp*]: *linepath 0 b x* = *x* $*_R$ *b*
by (*simp add*: *linepath-def*)

lemma *arc-linepath*:
assumes *a* ≠ *b*
shows *arc* (*linepath a b*)
proof −
{
fix *x y* :: *real*
assume *x* $*_R$ *b* + *y* $*_R$ *a* = *x* $*_R$ *a* + *y* $*_R$ *b*
then have (*x* − *y*) $*_R$ *a* = (*x* − *y*) $*_R$ *b*
by (*simp add*: *algebra-simps*)
with *assms* **have** *x* = *y*
by *simp*
}
then show *?thesis*
unfolding *arc-def inj-on-def*
by (*simp add*: *path-linepath*) (*force simp*: *algebra-simps linepath-def*)
qed

lemma *simple-path-linepath*[*intro*]: *a* ≠ *b* ⟹ *simple-path* (*linepath a b*)
by (*simp add*: *arc-imp-simple-path arc-linepath*)

lemma *linepath-trivial* [*simp*]: *linepath a a x* = *a*
by (*simp add*: *linepath-def real-vector.scale-left-diff-distrib*)

lemma *subpath-refl*: *subpath a a g* = *linepath* (*g a*) (*g a*)
by (*simp add*: *subpath-def linepath-def algebra-simps*)

23.4 Bounding a point away from a path

lemma *not-on-path-ball*:
fixes *g* :: *real* ⇒ *'a*::*heine-borel*
assumes *path g*
and *z* ∉ *path-image g*
shows ∃ *e* > *0*. *ball z e* ∩ *path-image g* = {}
proof −
obtain *a* **where** *a* ∈ *path-image g* ∀ *y* ∈ *path-image g*. *dist z a* ≤ *dist z y*
using *distance-attains-inf*[*OF - path-image-nonempty*, *of g z*]
using *compact-path-image*[*THEN compact-imp-closed*, *OF assms*(*1*)] **by** *auto*
then show *?thesis*
apply (*rule-tac x*=*dist z a* **in** *exI*)
using *assms*(*2*)
apply (*auto intro*!: *dist-pos-lt*)

done
qed

lemma *not-on-path-cball*:
fixes g :: *real* $\Rightarrow$ *'a*::*heine-borel*
assumes *path g*
and $z \notin$ *path-image g*
shows $\exists\, e{>}0.$ *cball z e* $\cap$ (*path-image g*) = {}
proof −
obtain *e* **where** *ball z e* $\cap$ *path-image g* = {} $e > 0$
using *not-on-path-ball*[*OF assms*] **by** *auto*
moreover have *cball z* (*e*/2) $\subseteq$ *ball z e*
using ‹$e > 0$› **by** *auto*
ultimately show *?thesis*
apply (*rule-tac x*=*e*/2 **in** *exI*)
apply *auto*
done
qed

24 Path component, considered as a "joinability" relation (from Tom Hales)

definition *path-component s x y* $\longleftrightarrow$
($\exists\, g.$ *path g* $\wedge$ *path-image g* $\subseteq$ *s* $\wedge$ *pathstart g* = *x* $\wedge$ *pathfinish g* = *y*)

abbreviation
path-component-set s x $\equiv$ *Collect* (*path-component s x*)

lemmas *path-defs* = *path-def pathstart-def pathfinish-def path-image-def path-component-def*

lemma *path-component-mem*:
assumes *path-component s x y*
shows $x \in s$ **and** $y \in s$
using *assms*
unfolding *path-defs*
by *auto*

lemma *path-component-refl*:
assumes $x \in s$
shows *path-component s x x*
unfolding *path-defs*
apply (*rule-tac x*=$\lambda u.$ *x* **in** *exI*)
using *assms*
apply (*auto intro*!: *continuous-intros*)
done

lemma *path-component-refl-eq*: *path-component s x x* $\longleftrightarrow$ $x \in s$
by (*auto intro*!: *path-component-mem path-component-refl*)

lemma *path-component-sym*: *path-component s x y* $\Longrightarrow$ *path-component s y x*
using *assms*
unfolding *path-component-def*
apply (*erule exE*)
apply (*rule-tac x=reversepath g* **in** *exI*)
apply *auto*
done

lemma *path-component-trans*:
assumes *path-component s x y* **and** *path-component s y z*
shows *path-component s x z*
using *assms*
unfolding *path-component-def*
apply (*elim exE*)
apply (*rule-tac x=g +++ ga* **in** *exI*)
apply (*auto simp add*: *path-image-join*)
done

lemma *path-component-of-subset*: $s \subseteq t \Longrightarrow$ *path-component s x y* $\Longrightarrow$ *path-component t x y*
unfolding *path-component-def* **by** *auto*

lemma *path-connected-linepath*:
fixes *s* :: *'a*::*real-normed-vector set*
shows *closed-segment a b* $\subseteq$ *s* $\Longrightarrow$ *path-component s a b*
apply (*simp add*: *path-component-def*)
apply (*rule-tac x=linepath a b* **in** *exI*, *auto*)
done

Can also consider it as a set, as the name suggests.

lemma *path-component-set*:
path-component-set s x =
{*y*. ($\exists$ *g*. *path g* $\wedge$ *path-image g* $\subseteq$ *s* $\wedge$ *pathstart g* = *x* $\wedge$ *pathfinish g* = *y*)}
by (*auto simp*: *path-component-def*)

lemma *path-component-subset*: *path-component-set s x* $\subseteq$ *s*
by (*auto simp add*: *path-component-mem*(*2*))

lemma *path-component-eq-empty*: *path-component-set s x* = {} $\longleftrightarrow$ *x* $\notin$ *s*
using *path-component-mem path-component-refl-eq*
by *fastforce*

lemma *path-component-mono*:
s $\subseteq$ *t* $\Longrightarrow$ (*path-component-set s x*) $\subseteq$ (*path-component-set t x*)
by (*simp add*: *Collect-mono path-component-of-subset*)

lemma *path-component-eq*:
y $\in$ *path-component-set s x* $\Longrightarrow$ *path-component-set s y* = *path-component-set s*

x
by (*metis* (*no-types*, *lifting*) *Collect-cong mem-Collect-eq path-component-sym path-component-trans*)

24.1 Path connectedness of a space

definition *path-connected s* ⟷
(∀ *x*∈*s*. ∀ *y*∈*s*. ∃ *g*. *path g* ∧ *path-image g* ⊆ *s* ∧ *pathstart g* = *x* ∧ *pathfinish g* = *y*)

lemma *path-connected-component*: *path-connected s* ⟷ (∀ *x*∈*s*. ∀ *y*∈*s*. *path-component s x y*)
unfolding *path-connected-def path-component-def* **by** *auto*

lemma *path-connected-component-set*: *path-connected s* ⟷ (∀ *x*∈*s*. *path-component-set s x* = *s*)
unfolding *path-connected-component path-component-subset*
using *path-component-mem* **by** *blast*

lemma *path-component-maximal*:
⟦*x* ∈ *t*; *path-connected t*; *t* ⊆ *s*⟧ ⟹ *t* ⊆ (*path-component-set s x*)
by (*metis path-component-mono path-connected-component-set*)

24.2 Some useful lemmas about path-connectedness

lemma *convex-imp-path-connected*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *convex s*
shows *path-connected s*
unfolding *path-connected-def*
apply *rule*
apply *rule*
apply (*rule-tac x* = *linepath x y* **in** *exI*)
unfolding *path-image-linepath*
using *assms* [*unfolded convex-contains-segment*]
apply *auto*
done

lemma *path-connected-imp-connected*:
assumes *path-connected s*
shows *connected s*
unfolding *connected-def not-ex*
apply *rule*
apply *rule*
apply (*rule ccontr*)
unfolding *not-not*
apply (*elim conjE*)
proof −
fix *e1 e2*
assume *as*: *open e1 open e2 s* ⊆ *e1* ∪ *e2 e1* ∩ *e2* ∩ *s* = {} *e1* ∩ *s* ≠ {} *e2* ∩ *s* ≠ {}

```
  then obtain x1 x2 where obt:x1 ∈ e1 ∩ s x2 ∈ e2 ∩ s
    by auto
  then obtain g where g: path g path-image g ⊆ s pathstart g = x1 pathfinish g
= x2
    using assms[unfolded path-connected-def,rule-format,of x1 x2] by auto
  have ∗: connected {0..1::real}
    by (auto intro!: convex-connected convex-real-interval)
  have {0..1} ⊆ {x ∈ {0..1}. g x ∈ e1} ∪ {x ∈ {0..1}. g x ∈ e2}
    using as(3) g(2)[unfolded path-defs] by blast
  moreover have {x ∈ {0..1}. g x ∈ e1} ∩ {x ∈ {0..1}. g x ∈ e2} = {}
    using as(4) g(2)[unfolded path-defs]
    unfolding subset-eq
    by auto
  moreover have {x ∈ {0..1}. g x ∈ e1} ≠ {} ∧ {x ∈ {0..1}. g x ∈ e2} ≠ {}
    using g(3,4)[unfolded path-defs]
    using obt
    by (simp add: ex-in-conv [symmetric], metis zero-le-one order-refl)
  ultimately show False
    using ∗[unfolded connected-local not-ex, rule-format,
      of {x∈{0..1}. g x ∈ e1} {x∈{0..1}. g x ∈ e2}]
    using continuous-openin-preimage[OF g(1)[unfolded path-def] as(1)]
    using continuous-openin-preimage[OF g(1)[unfolded path-def] as(2)]
    by auto
qed

lemma open-path-component:
  fixes s :: 'a::real-normed-vector set
  assumes open s
  shows open (path-component-set s x)
  unfolding open-contains-ball
proof
  fix y
  assume as: y ∈ path-component-set s x
  then have y ∈ s
    apply −
    apply (rule path-component-mem(2))
    unfolding mem-Collect-eq
    apply auto
    done
  then obtain e where e: e > 0 ball y e ⊆ s
    using assms[unfolded open-contains-ball]
    by auto
  show ∃ e > 0. ball y e ⊆ path-component-set s x
    apply (rule-tac x=e in exI)
    apply (rule,rule ‹e>0›)
    apply rule
    unfolding mem-ball mem-Collect-eq
  proof −
    fix z
```

assume *dist y z < e*
then show *path-component s x z*
apply (*rule-tac path-component-trans*[*of* - - *y*])
defer
apply (*rule path-component-of-subset*[*OF e*(*2*)])
apply (*rule convex-imp-path-connected*[*OF convex-ball*, *unfolded path-connected-component*, *rule-format*])
using ‹*e > 0*› *as*
apply *auto*
done
qed
qed

lemma *open-non-path-component*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *open s*
shows *open* (*s − path-component-set s x*)
unfolding *open-contains-ball*
proof
fix *y*
assume *as*: *y* ∈ *s − path-component-set s x*
then obtain *e* **where** *e*: *e > 0 ball y e* ⊆ *s*
using *assms* [*unfolded open-contains-ball*]
by *auto*
show ∃ *e*>*0*. *ball y e* ⊆ *s − path-component-set s x*
apply (*rule-tac x*=*e* **in** *exI*)
apply *rule*
apply (*rule* ‹*e*>*0*›)
apply *rule*
apply *rule*
defer
proof (*rule ccontr*)
fix *z*
assume *z* ∈ *ball y e* ¬ *z* ∉ *path-component-set s x*
then have *y* ∈ *path-component-set s x*
unfolding *not-not mem-Collect-eq* **using** ‹*e*>*0*›
apply −
apply (*rule path-component-trans*, *assumption*)
apply (*rule path-component-of-subset*[*OF e*(*2*)])
apply (*rule convex-imp-path-connected*[*OF convex-ball*, *unfolded path-connected-component*, *rule-format*])
apply *auto*
done
then show *False*
using *as* **by** *auto*
qed (*insert e*(*2*), *auto*)
qed

lemma *connected-open-path-connected*:

fixes $s :: 'a::real\text{-}normed\text{-}vector\ set$
assumes *open s*
and *connected s*
shows *path-connected s*
unfolding *path-connected-component-set*
proof (*rule*, *rule*, *rule path-component-subset*, *rule*)
fix *x y*
assume $x \in s$ **and** $y \in s$
show $y \in$ *path-component-set s x*
proof (*rule ccontr*)
assume $\neg$ *?thesis*
moreover have *path-component-set s x* $\cap$ *s* $\neq \{\}$
using ‹$x \in s$› *path-component-eq-empty path-component-subset*[*of s x*]
by *auto*
ultimately
show *False*
using ‹$y \in s$› *open-non-path-component*[*OF assms*(*1*)] *open-path-component*[*OF assms*(*1*)]
using *assms*(*2*)[*unfolded connected-def not-ex*, *rule-format*,
of path-component-set s x s − path-component-set s x]
by *auto*
qed
qed

lemma *path-connected-continuous-image*:
assumes *continuous-on s f*
and *path-connected s*
shows *path-connected* (*f ‘ s*)
unfolding *path-connected-def*
proof (*rule*, *rule*)
fix x' y'
assume $x' \in f$ ‘ *s* $y' \in f$ ‘ *s*
then obtain *x y* **where** *x*: $x \in s$ **and** *y*: $y \in s$ **and** x': $x' = f\ x$ **and** y': $y' = f\ y$
by *auto*
from *x y* **obtain** *g* **where** *path g* $\wedge$ *path-image g* $\subseteq$ *s* $\wedge$ *pathstart g* $=$ *x* $\wedge$ *pathfinish g* $=$ *y*
using *assms*(*2*)[*unfolded path-connected-def*] **by** *fast*
then show $\exists g.$ *path g* $\wedge$ *path-image g* $\subseteq f$ ‘ *s* $\wedge$ *pathstart g* $= x' \wedge$ *pathfinish g* $= y'$
unfolding x' y'
apply (*rule-tac x*=*f* $\circ$ *g* **in** *exI*)
unfolding *path-defs*
apply (*intro conjI continuous-on-compose continuous-on-subset*[*OF assms*(*1*)])
apply *auto*
done
qed

lemma *path-connected-segment*:

fixes *a* :: *'a*::*real-normed-vector*
shows *path-connected* (*closed-segment a b*)
by (*simp add*: *convex-imp-path-connected*)

lemma *path-connected-open-segment*:
fixes *a* :: *'a*::*real-normed-vector*
shows *path-connected* (*open-segment a b*)
by (*simp add*: *convex-imp-path-connected*)

lemma *homeomorphic-path-connectedness*:
s homeomorphic t $\Longrightarrow$ *path-connected s* $\longleftrightarrow$ *path-connected t*
unfolding *homeomorphic-def homeomorphism-def* **by** (*metis path-connected-continuous-image*)

lemma *path-connected-empty*: *path-connected* {}
unfolding *path-connected-def* **by** *auto*

lemma *path-connected-singleton*: *path-connected* {*a*}
unfolding *path-connected-def pathstart-def pathfinish-def path-image-def*
apply *clarify*
apply (*rule-tac x*=$\lambda x.$ *a* **in** *exI*)
apply (*simp add*: *image-constant-conv*)
apply (*simp add*: *path-def continuous-on-const*)
done

lemma *path-connected-Un*:
assumes *path-connected s*
and *path-connected t*
and $s \cap t \neq \{\}$
shows *path-connected* ($s \cup t$)
unfolding *path-connected-component*
proof (*rule*, *rule*)
fix *x y*
assume *as*: $x \in s \cup t$ $y \in s \cup t$
from *assms*(*3*) **obtain** *z* **where** $z \in s \cap t$
by *auto*
then show *path-component* ($s \cup t$) *x y*
using *as* **and** *assms*(*1*−*2*)[*unfolded path-connected-component*]
apply −
apply (*erule-tac*[!] *UnE*)+
apply (*rule-tac*[*2*−*3*] *path-component-trans*[*of* - - *z*])
apply (*auto simp add*:*path-component-of-subset* [*OF Un-upper1*] *path-component-of-subset*[*OF Un-upper2*])
done
qed

lemma *path-connected-UNION*:
assumes $\bigwedge i.\ i \in A \Longrightarrow$ *path-connected* (*S i*)
and $\bigwedge i.\ i \in A \Longrightarrow z \in S\ i$
shows *path-connected* ($\bigcup i{\in}A.\ S\ i$)

unfolding *path-connected-component*
proof *clarify*
fix $x\ i\ y\ j$
assume $*$: $i \in A\ x \in S\ i\ j \in A\ y \in S\ j$
then have *path-component* $(S\ i)\ x\ z$ **and** *path-component* $(S\ j)\ z\ y$
using *assms* **by** (*simp-all add*: *path-connected-component*)
then have *path-component* $(\bigcup i{\in}A.\ S\ i)\ x\ z$ **and** *path-component* $(\bigcup i{\in}A.\ S\ i)$ $z\ y$
using $*(1,3)$ **by** (*auto elim*!: *path-component-of-subset* [*rotated*])
then show *path-component* $(\bigcup i{\in}A.\ S\ i)\ x\ y$
by (*rule path-component-trans*)
qed

lemma *path-component-path-image-pathstart*:
assumes p: *path* p **and** x: $x \in$ *path-image* p
shows *path-component* (*path-image* p) (*pathstart* p) x
using x
proof (*clarsimp simp add*: *path-image-def*)
fix y
assume $x = p\ y$ **and** y: $0 \le y\ y \le 1$
show *path-component* $(p\ `\ \{0..1\})$ (*pathstart* p) $(p\ y)$
proof (*cases* $y{=}0$)
case *True* **then show** *?thesis*
by (*simp add*: *path-component-refl-eq pathstart-def*)
next
case *False* **have** *continuous-on* $\{0..1\}$ $(p\ o\ (op{*}y))$
apply (*rule continuous-intros*)+
using p [*unfolded path-def*] y
apply (*auto simp*: *mult-le-one intro*: *continuous-on-subset* [*of* - p])
done
then have *path* $(\lambda u.\ p\ (y * u))$
by (*simp add*: *path-def*)
then show *?thesis*
apply (*simp add*: *path-component-def*)
apply (*rule-tac* $x = \lambda u.\ p\ (y * u)$ **in** *exI*)
apply (*intro conjI*)
using y *False*
apply (*auto simp*: *mult-le-one pathstart-def pathfinish-def path-image-def*)
done
qed
qed

lemma *path-connected-path-image*: *path* $p \Longrightarrow$ *path-connected*(*path-image* p)
unfolding *path-connected-component*
by (*meson path-component-path-image-pathstart path-component-sym path-component-trans*)

lemma *path-connected-path-component*:
path-connected (*path-component-set* $s\ x$)
proof −

```
  { fix y z
    assume pa: path-component s x y path-component s x z
    then have pae: path-component-set s x = path-component-set s y
      using path-component-eq by auto
    have yz: path-component s y z
      using pa path-component-sym path-component-trans by blast
    then have ∃ g. path g ∧ path-image g ⊆ path-component-set s x ∧ pathstart g
= y ∧ pathfinish g = z
      apply (simp add: path-component-def, clarify)
      apply (rule-tac x=g in exI)
     by (simp add: pae path-component-maximal path-connected-path-image pathstart-in-path-image)
  }
  then show ?thesis
    by (simp add: path-connected-def)
qed

lemma path-component: path-component s x y ⟷ (∃ t. path-connected t ∧ t ⊆ s
∧ x ∈ t ∧ y ∈ t)
  apply (intro iffI)
   apply (metis path-connected-path-image path-defs(5) pathfinish-in-path-image
pathstart-in-path-image)
  using path-component-of-subset path-connected-component by blast

lemma path-component-path-component [simp]:
   path-component-set (path-component-set s x) x = path-component-set s x
proof (cases x ∈ s)
  case True show ?thesis
    apply (rule subset-antisym)
    apply (simp add: path-component-subset)
   by (simp add: True path-component-maximal path-component-refl path-connected-path-component)
next
  case False then show ?thesis
    by (metis False empty-iff path-component-eq-empty)
qed

lemma path-component-subset-connected-component:
   (path-component-set s x) ⊆ (connected-component-set s x)
proof (cases x ∈ s)
  case True show ?thesis
    apply (rule connected-component-maximal)
   apply (auto simp: True path-component-subset path-component-refl path-connected-imp-connected
path-connected-path-component)
    done
next
  case False then show ?thesis
    using path-component-eq-empty by auto
qed
```

24.3 Sphere is path-connected

lemma *path-connected-punctured-universe*:
assumes $2 \leq DIM('a::euclidean\text{-}space)$
shows *path-connected* $(- \{a::'a\})$
proof −
let *?A* $= \{x::'a.\ \exists i{\in}Basis.\ x \cdot i < a \cdot i\}$
let *?B* $= \{x::'a.\ \exists i{\in}Basis.\ a \cdot i < x \cdot i\}$

have *A*: *path-connected ?A*
unfolding *Collect-bex-eq*
proof (*rule path-connected-UNION*)
fix $i :: 'a$
assume $i \in Basis$
then show $(\sum i{\in}Basis.\ (a \cdot i - 1){*_R}\ i) \in \{x::'a.\ x \cdot i < a \cdot i\}$
by *simp*
show *path-connected* $\{x.\ x \cdot i < a \cdot i\}$
using *convex-imp-path-connected* [*OF convex-halfspace-lt*, *of i a* $\cdot$ *i*]
by (*simp add*: *inner-commute*)
qed
have *B*: *path-connected ?B*
unfolding *Collect-bex-eq*
proof (*rule path-connected-UNION*)
fix $i :: 'a$
assume $i \in Basis$
then show $(\sum i{\in}Basis.\ (a \cdot i + 1)\ {*_R}\ i) \in \{x::'a.\ a \cdot i < x \cdot i\}$
by *simp*
show *path-connected* $\{x.\ a \cdot i < x \cdot i\}$
using *convex-imp-path-connected* [*OF convex-halfspace-gt*, *of a* $\cdot$ *i i*]
by (*simp add*: *inner-commute*)
qed
obtain $S :: 'a\ set$ **where** $S \subseteq Basis$ **and** $card\ S = Suc\ (Suc\ 0)$
using *ex-card*[*OF assms*]
by *auto*
then obtain *b0 b1* $:: 'a$ **where** $b0 \in Basis$ **and** $b1 \in Basis$ **and** $b0 \neq b1$
unfolding *card-Suc-eq* **by** *auto*
then have $a + b0 - b1 \in ?A \cap ?B$
by (*auto simp*: *inner-simps inner-Basis*)
then have $?A \cap ?B \neq \{\}$
by *fast*
with *A B* **have** *path-connected* $(?A \cup ?B)$
by (*rule path-connected-Un*)
also have $?A \cup ?B = \{x.\ \exists i{\in}Basis.\ x \cdot i \neq a \cdot i\}$
unfolding *neq-iff bex-disj-distrib Collect-disj-eq* **..**
also have $\ldots = \{x.\ x \neq a\}$
unfolding *euclidean-eq-iff* [**where** $'a='a$]
by (*simp add*: *Bex-def*)
also have $\ldots = - \{a\}$
by *auto*
finally show *?thesis* **.**

qed

lemma *path-connected-sphere*:
assumes *2* ≤ *DIM*('a::*euclidean-space*)
shows *path-connected* {x::'a. *norm* (x − a) = r}
proof (*rule linorder-cases* [*of r 0*])
assume *r* < *0*
then have {x::'a. *norm*(x − a) = r} = {}
by *auto*
then show *?thesis*
using *path-connected-empty* **by** *simp*
next
assume *r* = *0*
then show *?thesis*
using *path-connected-singleton* **by** *simp*
next
assume *r*: *0* < *r*
have ∗: {x::'a. *norm*(x − a) = r} = (λx. a + r ∗$_R$ x) ‘ {x. *norm* x = *1*}
apply (*rule set-eqI*)
apply *rule*
unfolding *image-iff*
apply (*rule-tac* x=(*1*/r) ∗$_R$ (x − a) **in** *bexI*)
unfolding *mem-Collect-eq norm-scaleR*
using *r*
apply (*auto simp add*: *scaleR-right-diff-distrib*)
done
have ∗∗: {x::'a. *norm* x = *1*} = (λx. (*1*/*norm* x) ∗$_R$ x) ‘ (− {*0*})
apply (*rule set-eqI*)
apply *rule*
unfolding *image-iff*
apply (*rule-tac* x=x **in** *bexI*)
unfolding *mem-Collect-eq*
apply (*auto split*: *split-if-asm*)
done
have *continuous-on* (− {*0*}) (λx::'a. *1* / *norm* x)
by (*auto intro*!: *continuous-intros*)
then show *?thesis*
unfolding ∗ ∗∗
using *path-connected-punctured-universe*[*OF assms*]
by (*auto intro*!: *path-connected-continuous-image continuous-intros*)
qed

corollary *connected-sphere*: *2* ≤ *DIM*('a::*euclidean-space*) ⟹ *connected* {x::'a. *norm* (x − a) = r}
using *path-connected-sphere path-connected-imp-connected*
by *auto*

corollary *path-connected-complement-bounded-convex*:
fixes *s* :: 'a :: *euclidean-space set*

assumes *bounded s convex s* **and** *2*: *2* ≤ *DIM*('*a*)
shows *path-connected* (− *s*)
proof (*cases s*={})
case *True* **then show** *?thesis*
using *convex-imp-path-connected* **by** *auto*
next
case *False*
then obtain *a* **where** *a* ∈ *s* **by** *auto*
{ **fix** *x y* **assume** *x* ∉ *s y* ∉ *s*
then have *x* ≠ *a y* ≠ *a* **using** ‹*a* ∈ *s*› **by** *auto*
then have *bxy*: *bounded*(*insert x* (*insert y s*))
by (*simp add*: ‹*bounded s*›)
then obtain *B*::*real* **where** *B*: *0* < *B* **and** *Bx*: *norm* (*a* − *x*) < *B* **and** *By*: *norm* (*a* − *y*) < *B*
and *s* ⊆ *ball a B*
using *bounded-subset-ballD* [*OF bxy*, *of a*] **by** (*auto simp*: *dist-norm*)
def *C* == *B* / *norm*(*x* − *a*)
{ **fix** *u*
assume *u*: (*1* − *u*) *∗R x* + *u ∗R* (*a* + *C ∗R* (*x* − *a*)) ∈ *s* **and** *0* ≤ *u u* ≤ *1*
have *CC*: *1* ≤ *1* + (*C* − *1*) ∗ *u*
using ‹*x* ≠ *a*› ‹*0* ≤ *u*›
apply (*simp add*: *C-def divide-simps norm-minus-commute*)
using *Bx* **by** *auto*
have ∗: ⋀*v*. (*1* − *u*) *∗R x* + *u ∗R* (*a* + *v ∗R* (*x* − *a*)) = *a* + (*1* + (*v* − *1*) ∗ *u*) *∗R* (*x* − *a*)
by (*simp add*: *algebra-simps*)
have *a* + ((*1* / (*1* + *C* ∗ *u* − *u*)) *∗R x* + ((*u* / (*1* + *C* ∗ *u* − *u*)) *∗R a* + (*C* ∗ *u* / (*1* + *C* ∗ *u* − *u*)) *∗R x*)) =
(*1* + (*u* / (*1* + *C* ∗ *u* − *u*))) *∗R a* + ((*1* / (*1* + *C* ∗ *u* − *u*)) + (*C* ∗ *u* / (*1* + *C* ∗ *u* − *u*))) *∗R x*
by (*simp add*: *algebra-simps*)
also have ... = (*1* + (*u* / (*1* + *C* ∗ *u* − *u*))) *∗R a* + (*1* + (*u* / (*1* + *C* ∗ *u* − *u*))) *∗R x*
using *CC* **by** (*simp add*: *field-simps*)
also have ... = *x* + (*1* + (*u* / (*1* + *C* ∗ *u* − *u*))) *∗R a* + (*u* / (*1* + *C* ∗ *u* − *u*)) *∗R x*
by (*simp add*: *algebra-simps*)
also have ... = *x* + ((*1* / (*1* + *C* ∗ *u* − *u*)) *∗R a* +
((*u* / (*1* + *C* ∗ *u* − *u*)) *∗R x* + (*C* ∗ *u* / (*1* + *C* ∗ *u* − *u*)) *∗R a*))
using *CC* **by** (*simp add*: *field-simps*) (*simp add*: *add-divide-distrib scaleR-add-left*)
finally have *xeq*: (*1* − *1* / (*1* + (*C* − *1*) ∗ *u*)) *∗R a* + (*1* / (*1* + (*C* − *1*) ∗ *u*)) *∗R* (*a* + (*1* + (*C* − *1*) ∗ *u*) *∗R* (*x* − *a*)) = *x*
by (*simp add*: *algebra-simps*)
have *False*
using ‹*convex s*›
apply (*simp add*: *convex-alt*)
apply (*drule-tac x*=*a* **in** *bspec*)
apply (*rule* ‹*a* ∈ *s*›)
apply (*drule-tac x*=*a* + (*1* + (*C* − *1*) ∗ *u*) *∗R* (*x* − *a*) **in** *bspec*)

```
      using u apply (simp add: *)
     apply (drule-tac x=1 / (1 + (C - 1) * u) in spec)
     using ‹x ≠ a› ‹x ∉ s› ‹0 ≤ u› CC
     apply (auto simp: xeq)
     done
  }
  then have pcx: path-component (- s) x (a + C *R (x - a))
    by (force simp: closed-segment-def intro!: path-connected-linepath)
  def D == B / norm(y - a)  — massive duplication with the proof above
  { fix u
    assume u: (1 - u) *R y + u *R (a + D *R (y - a)) ∈ s and 0 ≤ u u ≤ 1
    have DD: 1 ≤ 1 + (D - 1) * u
      using ‹y ≠ a› ‹0 ≤ u›
      apply (simp add: D-def divide-simps norm-minus-commute)
      using By by auto
    have *: ⋀v. (1 - u) *R y + u *R (a + v *R (y - a)) = a + (1 + (v - 1)
* u) *R (y - a)
      by (simp add: algebra-simps)
    have a + ((1 / (1 + D * u - u)) *R y + ((u / (1 + D * u - u)) *R a +
(D * u / (1 + D * u - u)) *R y)) =
          (1 + (u / (1 + D * u - u))) *R a + ((1 / (1 + D * u - u)) + (D *
u / (1 + D * u - u))) *R y
      by (simp add: algebra-simps)
    also have ... = (1 + (u / (1 + D * u - u))) *R a + (1 + (u / (1 + D * u
- u))) *R y
      using DD by (simp add: field-simps)
    also have ... = y + (1 + (u / (1 + D * u - u))) *R a + (u / (1 + D * u
- u)) *R y
      by (simp add: algebra-simps)
    also have ... = y + ((1 / (1 + D * u - u)) *R a +
          ((u / (1 + D * u - u)) *R y + (D * u / (1 + D * u - u)) *R a))
    using DD by (simp add: field-simps) (simp add: add-divide-distrib scaleR-add-left)
    finally have xeq: (1 - 1 / (1 + (D - 1) * u)) *R a + (1 / (1 + (D - 1)
* u)) *R (a + (1 + (D - 1) * u) *R (y - a)) = y
      by (simp add: algebra-simps)
    have False
      using ‹convex s›
      apply (simp add: convex-alt)
      apply (drule-tac x=a in bspec)
       apply (rule ‹a ∈ s›)
      apply (drule-tac x=a + (1 + (D - 1) * u) *R (y - a) in bspec)
       using u apply (simp add: *)
      apply (drule-tac x=1 / (1 + (D - 1) * u) in spec)
      using ‹y ≠ a› ‹y ∉ s› ‹0 ≤ u› DD
      apply (auto simp: xeq)
      done
  }
  then have pdy: path-component (- s) y (a + D *R (y - a))
    by (force simp: closed-segment-def intro!: path-connected-linepath)
```

have *pyx*: *path-component* $(-\ s)$ $(a + D *_R (y - a))$ $(a + C *_R (x - a))$
apply (*rule path-component-of-subset* [*of* $\{x.\ norm(x - a) = B\}$])
using ‹$s \subseteq$ *ball a B*›
apply (*force simp*: *ball-def dist-norm norm-minus-commute*)
apply (*rule path-connected-sphere* [*OF 2*, *of a B*, *simplified path-connected-component*, *rule-format*])
using ‹$x \neq a$› **using** ‹$y \neq a$› *B* **apply** (*auto simp*: *C-def D-def*)
done
have *path-component* $(-\ s)$ *x y*
by (*metis path-component-trans path-component-sym pcx pdy pyx*)
}
then show *?thesis*
by (*auto simp*: *path-connected-component*)
qed

lemma *connected-complement-bounded-convex*:
fixes *s* :: *'a* :: *euclidean-space set*
assumes *bounded s convex s 2* $\leq$ *DIM*(*'a*)
shows *connected* $(-\ s)$
using *path-connected-complement-bounded-convex* [*OF assms*] *path-connected-imp-connected*
by *blast*

lemma *connected-diff-ball*:
fixes *s* :: *'a* :: *euclidean-space set*
assumes *connected s cball a r* $\subseteq$ *s 2* $\leq$ *DIM*(*'a*)
shows *connected* $(s - ball\ a\ r)$
apply (*rule connected-diff-open-from-closed* [*OF ball-subset-cball*])
using *assms connected-sphere*
apply (*auto simp*: *cball-diff-eq-sphere dist-norm*)
done

24.4 Relations between components and path components

lemma *open-connected-component*:
fixes *s* :: *'a*::*real-normed-vector set*
shows *open s* $\Longrightarrow$ *open* (*connected-component-set s x*)
apply (*simp add*: *open-contains-ball*, *clarify*)
apply (*rename-tac y*)
apply (*drule-tac x*=*y* **in** *bspec*)
apply (*simp add*: *connected-component-in*, *clarify*)
apply (*rule-tac x*=*e* **in** *exI*)
by (*metis mem-Collect-eq connected-component-eq connected-component-maximal centre-in-ball connected-ball*)

corollary *open-components*:
fixes *s* :: *'a*::*real-normed-vector set*
shows $[\![$*open u*; *s* $\in$ *components u*$]\!] \Longrightarrow$ *open s*
by (*simp add*: *components-iff*) (*metis open-connected-component*)

lemma *in-closure-connected-component*:
fixes s :: *'a::real-normed-vector set*
assumes x: $x \in s$ **and** s: *open s*
shows $x \in$ *closure* (*connected-component-set s y*) $\longleftrightarrow$ $x \in$ *connected-component-set s y*
proof −
{ assume $x \in$ *closure* (*connected-component-set s y*)
moreover have $x \in$ *connected-component-set s x*
using x **by** *simp*
ultimately have $x \in$ *connected-component-set s y*
using s **by** (*meson Compl-disjoint closure-iff-nhds-not-empty connected-component-disjoint disjoint-eq-subset-Compl open-connected-component*)
}
then show *?thesis*
by (*auto simp*: *closure-def*)
qed

24.5 Existence of unbounded components

lemma *cobounded-unbounded-component*:
fixes s :: *'a* :: *euclidean-space set*
assumes *bounded* (−s)
shows $\exists x.\ x \in s \wedge \sim$ *bounded* (*connected-component-set s x*)
proof −
obtain i::*'a* **where** i: $i \in$ *Basis*
using *nonempty-Basis* **by** *blast*
obtain B **where** B: $B>0$ $-s \subseteq$ *ball 0 B*
using *bounded-subset-ballD* [*OF assms*, *of 0*] **by** *auto*
then have $*$: $\bigwedge x.\ B \le$ *norm x* $\Longrightarrow$ $x \in s$
by (*force simp add*: *ball-def dist-norm*)
have *unbounded-inner*: $\sim$ *bounded* $\{x.$ *inner i x* $\ge B\}$
apply (*auto simp*: *bounded-def dist-norm*)
apply (*rule-tac x*=*x* + (*max B e* + *1* + $|i \cdot x|$) $*_R$ *i* **in** *exI*)
apply *simp*
using *i*
apply (*auto simp*: *algebra-simps*)
done
have $**$: $\{x.\ B \le i \cdot x\} \subseteq$ *connected-component-set s* (*B* $*_R$ *i*)
apply (*rule connected-component-maximal*)
apply (*auto simp*: *i intro*: *convex-connected convex-halfspace-ge* [*of B*])
apply (*rule* $*$)
apply (*rule order-trans* [*OF - Basis-le-norm* [*OF i*]])
by (*simp add*: *inner-commute*)
have *B* $*_R$ *i* $\in$ *s*
by (*rule* $*$) (*simp add*: *norm-Basis* [*OF i*])
then show *?thesis*
apply (*rule-tac x*=*B* $*_R$ *i* **in** *exI*, *clarify*)
apply (*frule bounded-subset* [*of* - $\{x.\ B \le i \cdot x\}$, *OF* - $**$])

using *unbounded-inner* **apply** *blast*
done
qed

lemma *cobounded-unique-unbounded-component*:
fixes $s :: 'a :: euclidean\text{-}space\ set$
assumes *bs*: *bounded* $(-s)$ **and** $2 \le DIM('a)$
and *bo*: $\sim$ *bounded*(*connected-component-set s x*)
$\sim$ *bounded*(*connected-component-set s y*)
shows *connected-component-set s x* = *connected-component-set s y*
proof −
obtain $i::'a$ **where** *i*: $i \in Basis$
using *nonempty-Basis* **by** *blast*
obtain *B* **where** *B*: $B>0$ $-s \subseteq ball\ 0\ B$
using *bounded-subset-ballD* [*OF bs*, *of 0*] **by** *auto*
then have $*$: $\bigwedge x.\ B \le norm\ x \Longrightarrow x \in s$
by (*force simp add*: *ball-def dist-norm*)
have *ccb*: *connected* $(-\ ball\ 0\ B :: 'a\ set)$
using *assms* **by** (*auto intro*: *connected-complement-bounded-convex*)
obtain x' **where** x': *connected-component s x x' norm* $x' > B$
using *bo* [*unfolded bounded-def dist-norm*, *simplified*, *rule-format*]
by (*metis diff-zero norm-minus-commute not-less*)
obtain y' **where** y': *connected-component s y y' norm* $y' > B$
using *bo* [*unfolded bounded-def dist-norm*, *simplified*, *rule-format*]
by (*metis diff-zero norm-minus-commute not-less*)
have $x'y'$: *connected-component s x' y'*
apply (*simp add*: *connected-component-def*)
apply (*rule-tac x*=− *ball 0 B* **in** *exI*)
using x' y'
apply (*auto simp*: *ccb dist-norm* $*$)
done
show *?thesis*
apply (*rule connected-component-eq*)
using x' y' $x'y'$
by (*metis* (*no-types*, *lifting*) *connected-component-eq-empty connected-component-eq-eq connected-component-idemp connected-component-in*)
qed

lemma *cobounded-unbounded-components*:
fixes $s :: 'a :: euclidean\text{-}space\ set$
shows *bounded* $(-s) \Longrightarrow \exists\, c.\ c \in components\ s \wedge \sim bounded\ c$
by (*metis cobounded-unbounded-component components-def imageI*)

lemma *cobounded-unique-unbounded-components*:
fixes $s :: 'a :: euclidean\text{-}space\ set$
shows $[\![bounded\ (-\ s);\ c \in components\ s;\ \neg\ bounded\ c;\ c' \in components\ s;\ \neg\ bounded\ c';\ 2 \le DIM('a)]\!] \Longrightarrow c' = c$
unfolding *components-iff*
by (*metis cobounded-unique-unbounded-component*)

lemma *cobounded-has-bounded-component*:
fixes $s :: 'a :: euclidean\text{-}space\ set$
shows $[\![bounded\ (-\ s);\ \sim connected\ s;\ 2 \le DIM('a)]\!] \Longrightarrow \exists c.\ c \in components\ s \wedge bounded\ c$
by (*meson cobounded-unique-unbounded-components connected-eq-connected-components-eq*)

25 The "inside" and "outside" of a set

The inside comprises the points in a bounded connected component of the set's complement. The outside comprises the points in unbounded connected component of the complement.

definition *inside* **where**
$inside\ s \equiv \{x.\ (x \notin s) \wedge bounded(connected\text{-}component\text{-}set\ (-\ s)\ x)\}$

definition *outside* **where**
$outside\ s \equiv -s \cap \{x.\ \sim bounded(connected\text{-}component\text{-}set\ (-\ s)\ x)\}$

lemma *outside*: $outside\ s = \{x.\ \sim bounded(connected\text{-}component\text{-}set\ (-\ s)\ x)\}$
by (*auto simp*: *outside-def*) (*metis Compl-iff bounded-empty connected-component-eq-empty*)

lemma *inside-no-overlap* [*simp*]: $inside\ s \cap s = \{\}$
by (*auto simp*: *inside-def*)

lemma *outside-no-overlap* [*simp*]:
$outside\ s \cap s = \{\}$
by (*auto simp*: *outside-def*)

lemma *inside-inter-outside* [*simp*]: $inside\ s \cap outside\ s = \{\}$
by (*auto simp*: *inside-def outside-def*)

lemma *inside-union-outside* [*simp*]: $inside\ s \cup outside\ s = (-\ s)$
by (*auto simp*: *inside-def outside-def*)

lemma *inside-eq-outside*:
$inside\ s = outside\ s \longleftrightarrow s = UNIV$
by (*auto simp*: *inside-def outside-def*)

lemma *inside-outside*: $inside\ s = (-\ (s \cup outside\ s))$
by (*force simp add*: *inside-def outside*)

lemma *outside-inside*: $outside\ s = (-\ (s \cup inside\ s))$
by (*auto simp*: *inside-outside*) (*metis IntI equals0D outside-no-overlap*)

lemma *union-with-inside*: $s \cup inside\ s = -\ outside\ s$
by (*auto simp*: *inside-outside*) (*simp add*: *outside-inside*)

lemma *union-with-outside*: $s \cup outside\ s = -\ inside\ s$

by (*simp add*: *inside-outside*)

lemma *outside-mono*: $s \subseteq t \Longrightarrow$ *outside* $t \subseteq$ *outside* s
by (*auto simp*: *outside bounded-subset connected-component-mono*)

lemma *inside-mono*: $s \subseteq t \Longrightarrow$ *inside* $s - t \subseteq$ *inside* t
by (*auto simp*: *inside-def bounded-subset connected-component-mono*)

lemma *segment-bound-lemma*:
fixes u::*real*
assumes $x \geq B$ $y \geq B$ $0 \leq u$ $u \leq 1$
shows $(1 - u) * x + u * y \geq B$
proof −
obtain *dx dy* **where** $dx \geq 0$ $dy \geq 0$ $x = B + dx$ $y = B + dy$
using *assms* **by** *auto* (*metis add.commute diff-add-cancel*)
with ‹$0 \leq u$› ‹$u \leq 1$› **show** *?thesis*
by (*simp add*: *add-increasing2 mult-left-le field-simps*)
qed

lemma *cobounded-outside*:
fixes s :: $'a$ :: *real-normed-vector set*
assumes *bounded* s **shows** *bounded* (− *outside* s)
proof −
obtain B **where** B: $B>0$ $s \subseteq$ *ball* 0 B
using *bounded-subset-ballD* [*OF assms*, *of 0*] **by** *auto*
{ fix x::$'a$ **and** C::*real*
assume *Bno*: $B \leq$ *norm* x **and** C: $0 < C$
have $\exists y$. *connected-component* (− s) x $y \wedge$ *norm* $y > C$
proof (*cases* $x = 0$)
case *True* **with** B *Bno* **show** *?thesis* **by** *force*
next
case *False* **with** B C **show** *?thesis*
apply (*rule-tac* x=((B+C)/*norm* x) $*_R$ x **in** *exI*)
apply (*simp add*: *connected-component-def*)
apply (*rule-tac* x=*closed-segment* x (((B+C)/*norm* x) $*_R$ x) **in** *exI*)
apply *simp*
apply (*rule-tac* y=− *ball* 0 B **in** *order-trans*)
prefer *2* **apply** *force*
apply (*simp add*: *closed-segment-def ball-def dist-norm*, *clarify*)
apply (*simp add*: *real-vector-class.scaleR-add-left* [*symmetric*] *divide-simps*)
using *segment-bound-lemma* [*of B norm x B+C*] *Bno*
by (*meson le-add-same-cancel1 less-eq-real-def not-le*)
qed
}
then show *?thesis*
apply (*simp add*: *outside-def assms*)
apply (*rule bounded-subset* [*OF bounded-ball* [*of 0 B*]])
apply (*force simp add*: *dist-norm not-less bounded-pos*)
done

qed

lemma *unbounded-outside*:
fixes s :: *'a*::{*real-normed-vector*, *perfect-space*} *set*
shows *bounded* $s \Longrightarrow \sim$ *bounded*(*outside* s)
using *cobounded-imp-unbounded cobounded-outside* **by** *blast*

lemma *bounded-inside*:
fixes s :: *'a*::{*real-normed-vector*, *perfect-space*} *set*
shows *bounded* $s \Longrightarrow$ *bounded*(*inside* s)
by (*simp add*: *bounded-Int cobounded-outside inside-outside*)

lemma *connected-outside*:
fixes s :: *'a*::*euclidean-space set*
assumes *bounded* s $2 \le$ *DIM*(*'a*)
shows *connected*(*outside* s)
apply (*simp add*: *connected-iff-connected-component*, *clarify*)
apply (*simp add*: *outside*)
apply (*rule-tac s=connected-component-set* $(-s)$ x **in** *connected-component-of-subset*)
apply (*metis* (*no-types*) *assms cobounded-unbounded-component cobounded-unique-unbounded-component connected-component-eq-eq connected-component-idemp double-complement mem-Collect-eq*)
apply *clarify*
apply (*metis connected-component-eq-eq connected-component-in*)
done

lemma *outside-connected-component-lt*:
outside $s = \{x.\ \forall B.\ \exists y.\ B < norm(y) \wedge$ *connected-component* $(-s)\ x\ y\}$
apply (*auto simp*: *outside bounded-def dist-norm*)
apply (*metis diff-0 norm-minus-cancel not-less*)
by (*metis less-diff-eq norm-minus-commute norm-triangle-ineq2 order.trans pinf*(*6*))

lemma *outside-connected-component-le*:
outside $s =$
$\{x.\ \forall B.\ \exists y.\ B \le norm(y) \wedge$
connected-component $(-s)\ x\ y\}$
apply (*simp add*: *outside-connected-component-lt*)
apply (*simp add*: *Set.set-eq-iff*)
by (*meson gt-ex leD le-less-linear less-imp-le order.trans*)

lemma *not-outside-connected-component-lt*:
fixes s :: *'a*::*euclidean-space set*
assumes s: *bounded* s **and** $2 \le$ *DIM*(*'a*)
shows $-$ (*outside* s) = $\{x.\ \forall B.\ \exists y.\ B < norm(y) \wedge \sim$ (*connected-component* $(-s)\ x\ y$)$\}$
proof $-$
obtain B::*real* **where** B: $0 < B$ **and** *Bno*: $\bigwedge x.\ x \in s \Longrightarrow norm\ x \le B$
using s [*simplified bounded-pos*] **by** *auto*
{ fix *y*::*'a* **and** *z*::*'a*
assume *yz*: $B < norm\ z\ B < norm\ y$

```
    have connected-component (− cball 0 B) y z
      apply (rule connected-componentI [OF - subset-refl])
      apply (rule connected-complement-bounded-convex)
      using assms yz
      by (auto simp: dist-norm)
    then have connected-component (− s) y z
      apply (rule connected-component-of-subset)
      apply (metis Bno Compl-anti-mono mem-cball-0 subset-iff)
      done
  } note cyz = this
  show ?thesis
    apply (auto simp: outside)
     apply (metis Compl-iff bounded-iff cobounded-imp-unbounded mem-Collect-eq
not-le)
    apply (simp add: bounded-pos)
    by (metis B connected-component-trans cyz not-le)
qed

lemma not-outside-connected-component-le:
    fixes s :: 'a::euclidean-space set
    assumes s: bounded s  2 ≤ DIM('a)
     shows − (outside s) = {x. ∀ B. ∃ y. B ≤ norm(y) ∧ ~ (connected-component
(− s) x y)}
apply (auto intro: less-imp-le simp: not-outside-connected-component-lt [OF assms])
by (meson gt-ex less-le-trans)

lemma inside-connected-component-lt:
    fixes s :: 'a::euclidean-space set
    assumes s: bounded s  2 ≤ DIM('a)
    shows inside s = {x. (x ∉ s) ∧ (∀ B. ∃ y. B < norm(y) ∧ ~(connected-component
(− s) x y))}
  by (auto simp: inside-outside not-outside-connected-component-lt [OF assms])

lemma inside-connected-component-le:
    fixes s :: 'a::euclidean-space set
    assumes s: bounded s  2 ≤ DIM('a)
    shows inside s = {x. (x ∉ s) ∧ (∀ B. ∃ y. B ≤ norm(y) ∧ ~(connected-component
(− s) x y))}
  by (auto simp: inside-outside not-outside-connected-component-le [OF assms])

lemma inside-subset:
  assumes connected u and ~bounded u and t ∪ u = − s
  shows inside s ⊆ t
apply (auto simp: inside-def)
by (metis bounded-subset [of connected-component-set (− s) -] connected-component-maximal
      Compl-iff Un-iff assms subsetI)

lemma frontier-interiors: frontier s = − interior(s) − interior(−s)
  by (simp add: Int-commute frontier-def interior-closure)
```

lemma *connected-inter-frontier*:
⟦*connected s*; *s* ∩ *t* ≠ {}; *s* − *t* ≠ {}⟧ ⟹ (*s* ∩ *frontier t* ≠ {})
apply (*simp add*: *frontier-interiors connected-open-in*, *safe*)
apply (*drule-tac x*=*s* ∩ *interior t* **in** *spec*, *safe*)
apply (*drule-tac* [*2*] *x*=*s* ∩ *interior* (−*t*) **in** *spec*)
apply (*auto simp*: *disjoint-eq-subset-Compl dest*: *interior-subset* [*THEN subsetD*])
done

lemma *connected-component-UNIV*:
fixes *x* :: *'a*::*real-normed-vector*
shows *connected-component-set UNIV x* = *UNIV*
using *connected-iff-eq-connected-component-set* [*of UNIV*::*'a set*] *connected-UNIV*
by *auto*

lemma *connected-component-eq-UNIV*:
fixes *x* :: *'a*::*real-normed-vector*
shows *connected-component-set s x* = *UNIV* ⟷ *s* = *UNIV*
using *connected-component-in connected-component-UNIV* **by** *blast*

lemma *components-univ* [*simp*]: *components UNIV* = {*UNIV* :: *'a*::*real-normed-vector set*}
by (*auto simp*: *components-eq-sing-iff*)

lemma *interior-inside-frontier*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *bounded s*
shows *interior s* ⊆ *inside* (*frontier s*)
proof −
{ fix *x y*
assume *x*: *x* ∈ *interior s* **and** *y*: *y* ∉ *s*
and *cc*: *connected-component* (− *frontier s*) *x y*
have *connected-component-set* (− *frontier s*) *x* ∩ *frontier s* ≠ {}
apply (*rule connected-inter-frontier*, *simp*)
apply (*metis IntI cc connected-component-in connected-component-refl empty-iff interiorE mem-Collect-eq set-rev-mp x*)
using *y cc*
by *blast*
then have *bounded* (*connected-component-set* (− *frontier s*) *x*)
using *connected-component-in* **by** *auto*
}
then show *?thesis*
apply (*auto simp*: *inside-def frontier-def*)
apply (*rule classical*)
apply (*rule bounded-subset* [*OF assms*], *blast*)
done
qed

lemma *inside-empty* [*simp*]: *inside* {} = ({} :: *'a* :: {*real-normed-vector*, *perfect-space*} *set*)
by (*simp add*: *inside-def connected-component-UNIV*)

lemma *outside-empty* [*simp*]: *outside* {} = (*UNIV* :: *'a* :: {*real-normed-vector*, *perfect-space*} *set*)
using *inside-empty inside-union-outside* **by** *blast*

lemma *inside-same-component*:
⟦*connected-component* (− *s*) *x y*; *x* ∈ *inside s*⟧ ⟹ *y* ∈ *inside s*
using *connected-component-eq connected-component-in*
by (*fastforce simp add*: *inside-def*)

lemma *outside-same-component*:
⟦*connected-component* (− *s*) *x y*; *x* ∈ *outside s*⟧ ⟹ *y* ∈ *outside s*
using *connected-component-eq connected-component-in*
by (*fastforce simp add*: *outside-def*)

lemma *convex-in-outside*:
fixes *s* :: *'a* :: {*real-normed-vector*, *perfect-space*} *set*
assumes *s*: *convex s* **and** *z*: *z* ∉ *s*
shows *z* ∈ *outside s*
proof (*cases s*={})
case *True* **then show** *?thesis* **by** *simp*
next
case *False* **then obtain** *a* **where** *a* ∈ *s* **by** *blast*
with *z* **have** *zna*: *z* ≠ *a* **by** *auto*
{ assume *bounded* (*connected-component-set* (− *s*) *z*)
with *bounded-pos-less* **obtain** *B* **where** *B*>*0* **and** *B*: ⋀*x*. *connected-component* (− *s*) *z x* ⟹ *norm x* < *B*
by (*metis mem-Collect-eq*)
def *C* ≡ ((*B* + *1* + *norm z*) / *norm* (*z*−*a*))
have *C* > *0*
using ‹*0* < *B*› *zna* **by** (*simp add*: *C-def divide-simps add-strict-increasing*)
have |*norm* (*z* + *C* ∗*R* (*z*−*a*)) − *norm* (*C* ∗*R* (*z*−*a*))| ≤ *norm z*
by (*metis add-diff-cancel norm-triangle-ineq3*)
moreover have *norm* (*C* ∗*R* (*z*−*a*)) > *norm z* + *B*
using *zna* ‹*B*>*0*› **by** (*simp add*: *C-def le-max-iff-disj field-simps*)
ultimately have *C*: *norm* (*z* + *C* ∗*R* (*z*−*a*)) > *B* **by** *linarith*
{ fix *u*::*real*
assume *u*: *0*≤*u u*≤*1* **and** *ins*: (*1* − *u*) ∗*R z* + *u* ∗*R* (*z* + *C* ∗*R* (*z* − *a*)) ∈ *s*
then have *Cpos*: *1* + *u* ∗ *C* > *0*
by (*meson* ‹*0* < *C*› *add-pos-nonneg less-eq-real-def zero-le-mult-iff zero-less-one*)
then have ∗: (*1* / (*1* + *u* ∗ *C*)) ∗*R z* + (*u* ∗ *C* / (*1* + *u* ∗ *C*)) ∗*R z* = *z*
by (*simp add*: *scaleR-add-left* [*symmetric*] *divide-simps*)
then have *False*
using *convexD-alt* [*OF s* ‹*a* ∈ *s*› *ins*, *of 1*/(*u*∗*C* + *1*)] ‹*C*>*0*› ‹*z* ∉ *s*› *Cpos u*
by (*simp add*: ∗ *divide-simps algebra-simps*)

```
    } note contra = this
    have connected-component (− s) z (z + C *R (z−a))
      apply (rule connected-componentI [OF connected-segment [of z z + C *R
(z−a)]])
      apply (simp add: closed-segment-def)
      using contra
      apply auto
      done
    then have False
      using zna B [of z + C *R (z−a)] C
      by (auto simp: divide-simps max-mult-distrib-right)
  }
  then show ?thesis
    by (auto simp: outside-def z)
qed

lemma outside-convex:
  fixes s :: 'a :: {real-normed-vector, perfect-space} set
  assumes convex s
    shows outside s = − s
  by (metis ComplD assms convex-in-outside equalityI inside-union-outside subsetI
sup.cobounded2)

lemma inside-convex:
  fixes s :: 'a :: {real-normed-vector, perfect-space} set
  shows convex s ⟹ inside s = {}
  by (simp add: inside-outside outside-convex)

lemma outside-subset-convex:
  fixes s :: 'a :: {real-normed-vector, perfect-space} set
  shows ⟦convex t; s ⊆ t⟧ ⟹ − t ⊆ outside s
  using outside-convex outside-mono by blast

lemma outside-frontier-misses-closure:
    fixes s :: 'a::real-normed-vector set
    assumes bounded s
    shows  outside(frontier s) ⊆ − closure s
  unfolding outside-inside Lattices.boolean-algebra-class.compl-le-compl-iff
proof −
  { assume interior s ⊆ inside (frontier s)
    hence interior s ∪ inside (frontier s) = inside (frontier s)
      by (simp add: subset-Un-eq)
    then have closure s ⊆ frontier s ∪ inside (frontier s)
      using frontier-def by auto
  }
  then show closure s ⊆ frontier s ∪ inside (frontier s)
    using interior-inside-frontier [OF assms] by blast
qed
```

lemma *outside-frontier-eq-complement-closure*:
fixes $s :: 'a :: \{real\text{-}normed\text{-}vector, perfect\text{-}space\}$ *set*
assumes *bounded s convex s*
shows *outside*(*frontier s*) = − *closure s*
by (*metis Diff-subset assms convex-closure frontier-def outside-frontier-misses-closure outside-subset-convex subset-antisym*)

lemma *inside-frontier-eq-interior*:
fixes $s :: 'a :: \{real\text{-}normed\text{-}vector, perfect\text{-}space\}$ *set*
shows ⟦*bounded s*; *convex s*⟧ ⟹ *inside*(*frontier s*) = *interior s*
apply (*simp add*: *inside-outside outside-frontier-eq-complement-closure*)
using *closure-subset interior-subset*
apply (*auto simp add*: *frontier-def*)
done

lemma *open-inside*:
fixes $s :: 'a$::*real-normed-vector set*
assumes *closed s*
shows *open* (*inside s*)
proof −
{ fix *x* **assume** *x*: *x* ∈ *inside s*
have *open* (*connected-component-set* (− *s*) *x*)
using *assms open-connected-component* **by** *blast*
then obtain *e* **where** *e*: *e*>*0* **and** *e*: ⋀*y*. *dist y x* < *e* ⟶ *connected-component* (− *s*) *x y*
using *dist-not-less-zero*
apply (*simp add*: *open-dist*)
by (*metis* (*no-types*, *lifting*) *Compl-iff connected-component-refl-eq inside-def mem-Collect-eq x*)
then have ∃ *e*>*0*. *ball x e* ⊆ *inside s*
by (*metis e dist-commute inside-same-component mem-ball subsetI x*)
}
then show *?thesis*
by (*simp add*: *open-contains-ball*)
qed

lemma *open-outside*:
fixes $s :: 'a$::*real-normed-vector set*
assumes *closed s*
shows *open* (*outside s*)
proof −
{ fix *x* **assume** *x*: *x* ∈ *outside s*
have *open* (*connected-component-set* (− *s*) *x*)
using *assms open-connected-component* **by** *blast*
then obtain *e* **where** *e*: *e*>*0* **and** *e*: ⋀*y*. *dist y x* < *e* ⟶ *connected-component* (− *s*) *x y*
using *dist-not-less-zero*
apply (*simp add*: *open-dist*)
by (*metis Int-iff outside-def connected-component-refl-eq x*)

```
    then have ∃ e>0. ball x e ⊆ outside s
      by (metis e dist-commute outside-same-component mem-ball subsetI x)
  }
  then show ?thesis
    by (simp add: open-contains-ball)
qed

lemma closure-inside-subset:
    fixes s :: 'a::real-normed-vector set
    assumes closed s
      shows closure(inside s) ⊆ s ∪ inside s
by (metis assms closure-minimal open-closed open-outside sup.cobounded2 union-with-inside)

lemma frontier-inside-subset:
    fixes s :: 'a::real-normed-vector set
    assumes closed s
      shows frontier(inside s) ⊆ s
proof −
  have closure (inside s) ∩ − inside s = closure (inside s) − interior (inside s)
   by (metis (no-types) Diff-Compl assms closure-closed interior-closure open-closed
open-inside)
  moreover have − inside s ∩ − outside s = s
    by (metis (no-types) compl-sup double-compl inside-union-outside)
  moreover have closure (inside s) ⊆ − outside s
    by (metis (no-types) assms closure-inside-subset union-with-inside)
  ultimately have closure (inside s) − interior (inside s) ⊆ s
    by blast
  then show ?thesis
    by (simp add: frontier-def open-inside interior-open)
qed

lemma closure-outside-subset:
    fixes s :: 'a::real-normed-vector set
    assumes closed s
      shows closure(outside s) ⊆ s ∪ outside s
  apply (rule closure-minimal, simp)
  by (metis assms closed-open inside-outside open-inside)

lemma frontier-outside-subset:
    fixes s :: 'a::real-normed-vector set
    assumes closed s
      shows frontier(outside s) ⊆ s
  apply (simp add: frontier-def open-outside interior-open)
   by (metis Diff-subset-conv assms closure-outside-subset interior-eq open-outside
sup.commute)

lemma inside-complement-unbounded-connected-empty:
    ⟦connected (− s); ¬ bounded (− s)⟧ ⟹ inside s = {}
  apply (simp add: inside-def)
```

by (*meson Compl-iff bounded-subset connected-component-maximal order-refl*)

lemma *inside-bounded-complement-connected-empty*:
fixes s :: *'a*::{*real-normed-vector*, *perfect-space*} *set*
shows ⟦*connected* (− s); *bounded* s⟧ ⟹ *inside* s = {}
by (*metis inside-complement-unbounded-connected-empty cobounded-imp-unbounded*)

lemma *inside-inside*:
assumes $s \subseteq$ *inside* t
shows *inside* $s - t \subseteq$ *inside* t
unfolding *inside-def*
proof *clarify*
fix x
assume x: $x \notin t$ $x \notin s$ **and** *bo*: *bounded* (*connected-component-set* (− s) x)
show *bounded* (*connected-component-set* (− t) x)
proof (*cases* $s \cap$ *connected-component-set* (− t) x = {})
case *True* **show** *?thesis*
apply (*rule bounded-subset* [*OF bo*])
apply (*rule connected-component-maximal*)
using x *True* **apply** *auto*
done
next
case *False* **then show** *?thesis*
using *assms* [*unfolded inside-def*] x
apply (*simp add*: *disjoint-iff-not-equal*, *clarify*)
apply (*drule subsetD*, *assumption*, *auto*)
by (*metis* (*no-types*, *hide-lams*) *ComplI connected-component-eq-eq*)
qed
qed

lemma *inside-inside-subset*: *inside*(*inside* s) $\subseteq$ s
using *inside-inside union-with-outside* **by** *fastforce*

lemma *inside-outside-intersect-connected*:
⟦*connected* t; *inside* $s \cap t \neq$ {}; *outside* $s \cap t \neq$ {}⟧ ⟹ $s \cap t \neq$ {}
apply (*simp add*: *inside-def outside-def ex-in-conv* [*symmetric*] *disjoint-eq-subset-Compl*, *clarify*)
by (*metis* (*no-types*, *hide-lams*) *Compl-anti-mono connected-component-eq connected-component-maximal contra-subsetD double-compl*)

lemma *outside-bounded-nonempty*:
fixes s :: *'a* :: {*real-normed-vector*, *perfect-space*} *set*
assumes *bounded* s **shows** *outside* $s \neq$ {}
by (*metis* (*no-types*, *lifting*) *Collect-empty-eq Collect-mem-eq Compl-eq-Diff-UNIV Diff-cancel Diff-disjoint UNIV-I assms ball-eq-empty bounded-diff cobounded-outside convex-ball double-complement order-refl outside-convex outside-def*)

```
lemma outside-compact-in-open:
   fixes s :: 'a :: {real-normed-vector,perfect-space} set
   assumes s: compact s and t: open t and s ⊆ t t ≠ {}
     shows outside s ∩ t ≠ {}
proof −
  have outside s ≠ {}
    by (simp add: compact-imp-bounded outside-bounded-nonempty s)
  with assms obtain a b where a: a ∈ outside s and b: b ∈ t by auto
  show ?thesis
  proof (cases a ∈ t)
    case True with a show ?thesis by blast
  next
    case False
      have front: frontier t ⊆ − s
        using ‹s ⊆ t› frontier-disjoint-eq t by auto
      { fix γ
       assume path γ and pimg-sbs: path-image γ − {pathfinish γ} ⊆ interior (−
t)
          and pf: pathfinish γ ∈ frontier t and ps: pathstart γ = a
       def c ≡ pathfinish γ
       have c ∈ −s unfolding c-def using front pf by blast
       moreover have open (−s) using s compact-imp-closed by blast
       ultimately obtain ε::real where ε > 0 and ε: cball c ε ⊆ −s
         using open-contains-cball[of −s] s by blast
       then obtain d where d ∈ t and d: dist d c < ε
         using closure-approachable [of c t] pf unfolding c-def
         by (metis Diff-iff frontier-def)
       then have d ∈ −s using ε
      using dist-commute by (metis contra-subsetD mem-cball not-le not-less-iff-gr-or-eq)
       have pimg-sbs-cos: path-image γ ⊆ −s
         using pimg-sbs apply (auto simp: path-image-def)
         apply (drule subsetD)
         using ‹c ∈ − s› ‹s ⊆ t› interior-subset apply (auto simp: c-def)
         done
       have closed-segment c d ≤ cball c ε
         apply (simp add: segment-convex-hull)
         apply (rule hull-minimal)
         using ‹ε > 0› d apply (auto simp: dist-commute)
         done
       with ε have closed-segment c d ⊆ −s by blast
       moreover have con-gcd: connected (path-image γ ∪ closed-segment c d)
         by (rule connected-Un) (auto simp: c-def ‹path γ› connected-path-image)
       ultimately have connected-component (− s) a d
         unfolding connected-component-def using pimg-sbs-cos ps by blast
       then have outside s ∩ t ≠ {}
        using outside-same-component [OF - a] by (metis IntI ‹d ∈ t› empty-iff)
      } note ∗ = this
      have pal: pathstart (linepath a b) ∈ closure (− t)
       by (auto simp: False closure-def)
```

```
      show ?thesis
       by (rule exists-path-subpath-to-frontier [OF path-linepath pal - *]) (auto simp:
b)
  qed
qed

lemma inside-inside-compact-connected:
    fixes s :: 'a :: euclidean-space set
    assumes s: closed s and t: compact t and connected t s ⊆ inside t
      shows inside s ⊆ inside t
proof (cases inside t = {})
  case True with assms show ?thesis by auto
next
  case False
  consider DIM('a) = 1 | DIM('a) ≥ 2
    using antisym not-less-eq-eq by fastforce
  then show ?thesis
  proof cases
    case 1 then show ?thesis
            using connected-convex-1-gen assms False inside-convex by blast
  next
    case 2
    have coms: compact s
      using assms apply (simp add: s compact-eq-bounded-closed)
       by (meson bounded-inside bounded-subset compact-imp-bounded)
    then have bst: bounded (s ∪ t)
      by (simp add: compact-imp-bounded t)
    then obtain r where 0 < r and r: s ∪ t ⊆ ball 0 r
      using bounded-subset-ballD by blast
    have outst: outside s ∩ outside t ≠ {}
    proof −
      have − ball 0 r ⊆ outside s
        apply (rule outside-subset-convex)
        using r by auto
      moreover have − ball 0 r ⊆ outside t
        apply (rule outside-subset-convex)
        using r by auto
      ultimately show ?thesis
          by (metis Compl-subset-Compl-iff Int-subset-iff bounded-ball inf.orderE
outside-bounded-nonempty outside-no-overlap)
    qed
    have s ∩ t = {} using assms
      by (metis disjoint-iff-not-equal inside-no-overlap subsetCE)
    moreover have outside s ∩ inside t ≠ {}
    by (meson False assms(4) compact-eq-bounded-closed coms open-inside outside-compact-in-open
t)
    ultimately have inside s ∩ t = {}
      using inside-outside-intersect-connected [OF ‹connected t›, of s]
       by (metis 2 compact-eq-bounded-closed coms connected-outside inf.commute
```

inside-outside-intersect-connected outst)
then show *?thesis*
using *inside-inside* [*OF* ‹$s \subseteq$ *inside* t›] **by** *blast*
qed
qed

lemma *connected-with-inside*:
fixes s :: $'a$ :: *real-normed-vector set*
assumes s: *closed* s **and** *cons*: *connected* s
shows *connected*($s \cup$ *inside* s)
proof (*cases* $s \cup$ *inside* $s = UNIV$)
case *True* **with** *assms* **show** *?thesis* **by** *auto*
next
case *False*
then obtain b **where** b: $b \notin s$ $b \notin$ *inside* s **by** *blast*
have $*$: $\exists y\ t.\ y \in s \wedge$ *connected* $t \wedge a \in t \wedge y \in t \wedge t \subseteq (s \cup$ *inside* s) **if** $a \in (s \cup$ *inside* s) **for** a
using *that* **proof**
assume $a \in s$ **then show** *?thesis*
apply (*rule-tac* x=a **in** *exI*)
apply (*rule-tac* x=$\{a\}$ **in** *exI*)
apply (*simp add*:)
done
next
assume a: $a \in$ *inside* s
show *?thesis*
apply (*rule exists-path-subpath-to-frontier* [*OF path-linepath* [*of a b*], *of inside* s])
using a **apply** (*simp add*: *closure-def*)
apply (*simp add*: b)
apply (*rule-tac* x=*pathfinish* h **in** *exI*)
apply (*rule-tac* x=*path-image* h **in** *exI*)
apply (*simp add*: *pathfinish-in-path-image connected-path-image*, *auto*)
using *frontier-inside-subset* s **apply** *fastforce*
by (*metis* (*no-types*, *lifting*) *frontier-inside-subset insertE insert-Diff interior-eq open-inside pathfinish-in-path-image* s *subsetCE*)
qed
show *?thesis*
apply (*simp add*: *connected-iff-connected-component*)
apply (*simp add*: *connected-component-def*)
apply (*clarify dest*!: $*$)
apply (*rename-tac* u u' t t')
apply (*rule-tac* x=$(s \cup t \cup t')$ **in** *exI*)
apply (*auto simp*: *intro*!: *connected-Un cons*)
done
qed

The proof is virtually the same as that above.

lemma *connected-with-outside*:

fixes $s :: 'a :: real\text{-}normed\text{-}vector\ set$
assumes s: $closed\ s$ **and** $cons$: $connected\ s$
shows $connected(s \cup outside\ s)$
proof $(cases\ s \cup outside\ s = UNIV)$
case $True$ **with** $assms$ **show** $?thesis$ **by** $auto$
next
case $False$
then obtain b **where** b: $b \notin s\ b \notin outside\ s$ **by** $blast$
have $*$: $\exists y\ t.\ y \in s \wedge connected\ t \wedge a \in t \wedge y \in t \wedge t \subseteq (s \cup outside\ s)$ **if** $a \in (s \cup outside\ s)$ **for** a
using $that$ **proof**
assume $a \in s$ **then show** $?thesis$
apply $(rule\text{-}tac\ x{=}a$ **in** $exI)$
apply $(rule\text{-}tac\ x{=}\{a\}$ **in** $exI)$
apply $(simp\ add{:})$
done
next
assume a: $a \in outside\ s$
show $?thesis$
apply $(rule\ exists\text{-}path\text{-}subpath\text{-}to\text{-}frontier\ [OF\ path\text{-}linepath\ [of\ a\ b],\ of\ outside\ s])$
using a **apply** $(simp\ add{:}\ closure\text{-}def)$
apply $(simp\ add{:}\ b)$
apply $(rule\text{-}tac\ x{=}pathfinish\ h$ **in** $exI)$
apply $(rule\text{-}tac\ x{=}path\text{-}image\ h$ **in** $exI)$
apply $(simp\ add{:}\ pathfinish\text{-}in\text{-}path\text{-}image\ connected\text{-}path\text{-}image,\ auto)$
using $frontier\text{-}outside\text{-}subset\ s$ **apply** $fastforce$
by $(metis\ (no\text{-}types, lifting)\ frontier\text{-}outside\text{-}subset\ insertE\ insert\text{-}Diff\ interior\text{-}eq\ open\text{-}outside\ pathfinish\text{-}in\text{-}path\text{-}image\ s\ subsetCE)$
qed
show $?thesis$
apply $(simp\ add{:}\ connected\text{-}iff\text{-}connected\text{-}component)$
apply $(simp\ add{:}\ connected\text{-}component\text{-}def)$
apply $(clarify\ dest!{:}\ *)$
apply $(rename\text{-}tac\ u\ u'\ t\ t')$
apply $(rule\text{-}tac\ x{=}(s \cup t \cup t')$ **in** $exI)$
apply $(auto\ simp{:}\ intro!{:}\ connected\text{-}Un\ cons)$
done
qed

lemma $inside\text{-}inside\text{-}eq\text{-}empty\ [simp]$:
fixes $s :: 'a :: \{real\text{-}normed\text{-}vector, perfect\text{-}space\}\ set$
assumes s: $closed\ s$ **and** $cons$: $connected\ s$
shows $inside\ (inside\ s) = \{\}$
by $(metis\ (no\text{-}types)\ unbounded\text{-}outside\ connected\text{-}with\text{-}outside\ [OF\ assms]\ bounded\text{-}Un\ inside\text{-}complement\text{-}unbounded\text{-}connected\text{-}empty\ unbounded\text{-}outside\ union\text{-}with\text{-}outside)$

lemma $inside\text{-}in\text{-}components$:
$inside\ s \in components\ (-\ s) \longleftrightarrow connected(inside\ s) \wedge inside\ s \neq \{\}$

apply (*simp add*: *in-components-maximal*)
apply (*auto intro*: *inside-same-component connected-componentI*)
apply (*metis IntI empty-iff inside-no-overlap*)
done

The proof is virtually the same as that above.

lemma *outside-in-components*:
outside s ∈ *components* (− *s*) ⟷ *connected*(*outside s*) ∧ *outside s* ≠ {}
apply (*simp add*: *in-components-maximal*)
apply (*auto intro*: *outside-same-component connected-componentI*)
apply (*metis IntI empty-iff outside-no-overlap*)
done

lemma *bounded-unique-outside*:
fixes *s* :: *'a* :: *euclidean-space set*
shows ⟦*bounded s*; *DIM*(*'a*) ≥ *2*⟧ ⟹ (*c* ∈ *components* (− *s*) ∧ ~*bounded c* ⟷ *c* = *outside s*)
apply (*rule iffI*)
apply (*metis cobounded-unique-unbounded-components connected-outside double-compl outside-bounded-nonempty outside-in-components unbounded-outside*)
by (*simp add*: *connected-outside outside-bounded-nonempty outside-in-components unbounded-outside*)

26 Homotopy of maps p,q : X=¿Y with property P of all intermediate maps.

We often just want to require that it fixes some subset, but to take in the case of a loop homotopy, it's convenient to have a general property P.

definition *homotopic-with* ::
[(*'a*::*topological-space* ⇒ *'b*::*topological-space*) ⇒ *bool*, *'a set*, *'b set*, *'a* ⇒ *'b*, *'a* ⇒ *'b*] ⇒ *bool*
where
homotopic-with P X Y p q ≡
(∃ *h*:: *real* × *'a* ⇒ *'b*.
continuous-on ({*0..1*} × *X*) *h* ∧
h ‘ ({*0..1*} × *X*) ⊆ *Y* ∧
(∀ *x*. *h*(*0*, *x*) = *p x*) ∧
(∀ *x*. *h*(*1*, *x*) = *q x*) ∧
(∀ *t* ∈ {*0..1*}. *P*(λ*x*. *h*(*t*, *x*))))

We often want to just localize the ending function equality or whatever.

proposition *homotopic-with*:
fixes *X* :: *'a*::*topological-space set* **and** *Y* :: *'b*::*topological-space set*
assumes ⋀*h k*. (⋀*x*. *x* ∈ *X* ⟹ *h x* = *k x*) ⟹ (*P h* ⟷ *P k*)
shows *homotopic-with P X Y p q* ⟷
(∃ *h* :: *real* × *'a* ⇒ *'b*.
continuous-on ({*0..1*} × *X*) *h* ∧

h ‘ $(\{0..1\} \times X) \subseteq Y \wedge$
$(\forall x \in X.\ h(0,x) = p\ x) \wedge$
$(\forall x \in X.\ h(1,x) = q\ x) \wedge$
$(\forall t \in \{0..1\}.\ P(\lambda x.\ h(t,\ x))))$
unfolding *homotopic-with-def*
apply (*rule iffI*, *blast*, *clarify*)
apply (*rule-tac* $x{=}\lambda(u,v).$ *if* $v \in X$ *then* $h(u,v)$ *else if* $u = 0$ *then* $p\ v$ *else* $q\ v$ **in** *exI*)
apply (*auto simp*:)
apply (*force elim*: *continuous-on-eq*)
apply (*drule-tac* $x{=}t$ **in** *bspec*, *force*)
apply (*subst assms*; *simp*)
done

proposition *homotopic-with-eq*:
assumes h: *homotopic-with* $P\ X\ Y\ f\ g$
and f': $\bigwedge x.\ x \in X \Longrightarrow f'\ x = f\ x$
and g': $\bigwedge x.\ x \in X \Longrightarrow g'\ x = g\ x$
and P: $(\bigwedge h\ k.\ (\bigwedge x.\ x \in X \Longrightarrow h\ x = k\ x) \Longrightarrow (P\ h \longleftrightarrow P\ k))$
shows *homotopic-with* $P\ X\ Y\ f'\ g'$
using h **unfolding** *homotopic-with-def*
apply *safe*
apply (*rule-tac* $x{=}\lambda(u,v).$ *if* $v \in X$ *then* $h(u,v)$ *else if* $u = 0$ *then* $f'\ v$ *else* $g'\ v$ **in** *exI*)
apply (*simp add*: $f'\ g'$, *safe*)
apply (*fastforce intro*: *continuous-on-eq*)
apply *fastforce*
apply (*subst* P; *fastforce*)
done

proposition *homotopic-with-equal*:
assumes *contf*: *continuous-on* $X\ f$ **and** fXY: f ‘ $X \subseteq Y$
and gf: $\bigwedge x.\ x \in X \Longrightarrow g\ x = f\ x$
and P: $P\ f\ P\ g$
shows *homotopic-with* $P\ X\ Y\ f\ g$
unfolding *homotopic-with-def*
apply (*rule-tac* $x{=}\lambda(u,v).$ *if* $u = 1$ *then* $g\ v$ *else* $f\ v$ **in** *exI*)
using *assms*
apply (*intro conjI*)
apply (*rule continuous-on-eq* [**where** $f = f\ o\ snd$])
apply (*rule continuous-intros* | *force*)+
apply *clarify*
apply (*case-tac* $t{=}1$; *force*)
done

lemma *image-Pair-const*: $(\lambda x.\ (x,\ c))$ ‘ $A = A \times \{c\}$
by (*auto simp*:)

lemma *homotopic-constant-maps*:
homotopic-with (λ*x*. *True*) *s t* (λ*x*. *a*) (λ*x*. *b*) ⟷ *s* = {} ∨ *path-component t a b*
proof (*cases s* = {} ∨ *t* = {})
case *True* **with** *continuous-on-const* **show** *?thesis*
by (*auto simp*: *homotopic-with path-component-def*)
next
case *False*
then obtain *c* **where** *c* ∈ *s* **by** *blast*
show *?thesis*
proof
assume *homotopic-with* (λ*x*. *True*) *s t* (λ*x*. *a*) (λ*x*. *b*)
then obtain *h* :: *real* × *'a* ⇒ *'b*
where *conth*: *continuous-on* ({*0..1*} × *s*) *h*
and *h*: *h* ' ({*0..1*} × *s*) ⊆ *t* (∀ *x*∈*s*. *h* (*0*, *x*) = *a*) (∀ *x*∈*s*. *h* (*1*, *x*) = *b*)
by (*auto simp*: *homotopic-with*)
have *continuous-on* {*0..1*} (*h* ∘ (λ*t*. (*t*, *c*)))
apply (*rule continuous-intros conth* | *simp add*: *image-Pair-const*)+
apply (*blast intro*: ‹*c* ∈ *s*› *continuous-on-subset* [*OF conth*])
done
with ‹*c* ∈ *s*› *h* **show** *s* = {} ∨ *path-component t a b*
apply (*simp-all add*: *homotopic-with path-component-def*)
apply (*auto simp*:)
apply (*drule-tac x*=*h o* (λ*t*. (*t*, *c*)) **in** *spec*)
apply (*auto simp*: *pathstart-def pathfinish-def path-image-def path-def*)
done
next
assume *s* = {} ∨ *path-component t a b*
with *False* **show** *homotopic-with* (λ*x*. *True*) *s t* (λ*x*. *a*) (λ*x*. *b*)
apply (*clarsimp simp*: *homotopic-with path-component-def pathstart-def pathfinish-def path-image-def path-def*)
apply (*rule-tac x*=*g o fst* **in** *exI*)
apply (*rule conjI continuous-intros* | *force*)+
done
qed
qed

26.1 Trivial properties.

lemma *homotopic-with-imp-property*: *homotopic-with P X Y f g* ⟹ *P f* ∧ *P g*
unfolding *homotopic-with-def Ball-def*
apply *clarify*
apply (*frule-tac x*=*0* **in** *spec*)
apply (*drule-tac x*=*1* **in** *spec*)
apply (*auto simp*:)
done

lemma *continuous-on-o-Pair*: ⟦*continuous-on* (*T* × *X*) *h*; *t* ∈ *T*⟧ ⟹ *continuous-on X* (*h o Pair t*)

by (*fast intro*: *continuous-intros elim*!: *continuous-on-subset*)

lemma *homotopic-with-imp-continuous*:
assumes *homotopic-with P X Y f g*
shows *continuous-on X f* ∧ *continuous-on X g*
proof −
obtain *h* :: *real* × *'a* ⇒ *'b*
where *conth*: *continuous-on* ({*0..1*} × *X*) *h*
and *h*: ∀ *x*. *h* (*0*, *x*) = *f x* ∀ *x*. *h* (*1*, *x*) = *g x*
using *assms* **by** (*auto simp*: *homotopic-with-def*)
have ∗: *t* ∈ {*0..1*} ⟹ *continuous-on X* (*h o* (λ*x*. (*t*,*x*))) **for** *t*
by (*rule continuous-intros continuous-on-subset* [*OF conth*] | *force*)+
show *?thesis*
using *h* ∗[*of 0*] ∗[*of 1*] **by** *auto*
qed

proposition *homotopic-with-imp-subset1*:
homotopic-with P X Y f g ⟹ *f* ' *X* ⊆ *Y*
by (*simp add*: *homotopic-with-def image-subset-iff*) (*metis atLeastAtMost-iff order-refl zero-le-one*)

proposition *homotopic-with-imp-subset2*:
homotopic-with P X Y f g ⟹ *g* ' *X* ⊆ *Y*
by (*simp add*: *homotopic-with-def image-subset-iff*) (*metis atLeastAtMost-iff order-refl zero-le-one*)

proposition *homotopic-with-mono*:
assumes *hom*: *homotopic-with P X Y f g*
and *Q*: ⋀*h*. ⟦*continuous-on X h*; *image h X* ⊆ *Y* ∧ *P h*⟧ ⟹ *Q h*
shows *homotopic-with Q X Y f g*
using *hom*
apply (*simp add*: *homotopic-with-def*)
apply (*erule ex-forward*)
apply (*force simp*: *intro*!: *Q dest*: *continuous-on-o-Pair*)
done

proposition *homotopic-with-subset-left*:
⟦*homotopic-with P X Y f g*; *Z* ⊆ *X*⟧ ⟹ *homotopic-with P Z Y f g*
apply (*simp add*: *homotopic-with-def*)
apply (*fast elim*!: *continuous-on-subset ex-forward*)
done

proposition *homotopic-with-subset-right*:
⟦*homotopic-with P X Y f g*; *Y* ⊆ *Z*⟧ ⟹ *homotopic-with P X Z f g*
apply (*simp add*: *homotopic-with-def*)
apply (*fast elim*!: *continuous-on-subset ex-forward*)
done

proposition *homotopic-with-compose-continuous-right*:

⟦*homotopic-with* (λ*f*. *p* (*f* ∘ *h*)) *X Y f g*; *continuous-on W h*; *h* ‘ *W* ⊆ *X*⟧
⟹ *homotopic-with p W Y* (*f o h*) (*g o h*)
apply (*clarsimp simp add*: *homotopic-with-def*)
apply (*rename-tac k*)
apply (*rule-tac x*=*k o* (λ*y*. (*fst y*, *h* (*snd y*))) **in** *exI*)
apply (*rule conjI continuous-intros continuous-on-compose* [**where** *f*=*snd* **and** *g*=*h*, *unfolded o-def*] | *simp*)+
apply (*erule continuous-on-subset*)
apply (*fastforce simp*: *o-def*)+
done

proposition *homotopic-compose-continuous-right*:
⟦*homotopic-with* (λ*f*. *True*) *X Y f g*; *continuous-on W h*; *h* ‘ *W* ⊆ *X*⟧
⟹ *homotopic-with* (λ*f*. *True*) *W Y* (*f o h*) (*g o h*)
using *homotopic-with-compose-continuous-right* **by** *fastforce*

proposition *homotopic-with-compose-continuous-left*:
⟦*homotopic-with* (λ*f*. *p* (*h* ∘ *f*)) *X Y f g*; *continuous-on Y h*; *h* ‘ *Y* ⊆ *Z*⟧
⟹ *homotopic-with p X Z* (*h o f*) (*h o g*)
apply (*clarsimp simp add*: *homotopic-with-def*)
apply (*rename-tac k*)
apply (*rule-tac x*=*h o k* **in** *exI*)
apply (*rule conjI continuous-intros continuous-on-compose* [**where** *f*=*snd* **and** *g*=*h*, *unfolded o-def*] | *simp*)+
apply (*erule continuous-on-subset*)
apply (*fastforce simp*: *o-def*)+
done

proposition *homotopic-compose-continuous-left*:
homotopic-with (λ*f*. *True*) *X Y f g* ∧
continuous-on Y h ∧ *image h Y* ⊆ *Z*
⟹ *homotopic-with* (λ*f*. *True*) *X Z* (*h o f*) (*h o g*)
using *homotopic-with-compose-continuous-left* **by** *fastforce*

proposition *homotopic-with-Pair*:
assumes *hom*: *homotopic-with p s t f g homotopic-with p′ s′ t′ f′ g′*
and *q*: ⋀*f g*. ⟦*p f*; *p′ g*⟧ ⟹ *q*(λ(*x*,*y*). (*f x*, *g y*))
shows *homotopic-with q* (*s* × *s′*) (*t* × *t′*)
(λ(*x*,*y*). (*f x*, *f′ y*)) (λ(*x*,*y*). (*g x*, *g′ y*))
using *hom*
apply (*clarsimp simp add*: *homotopic-with-def*)
apply (*rename-tac k k′*)
apply (*rule-tac x*=λ*z*. ((*k o* (λ*x*. (*fst x*, *fst* (*snd x*)))) *z*, (*k′ o* (λ*x*. (*fst x*, *snd* (*snd x*)))) *z*) **in** *exI*)
apply (*rule conjI continuous-intros* | *erule continuous-on-subset* | *clarsimp*)+
apply (*auto intro*!: *q* [*unfolded case-prod-unfold*])
done

lemma *homotopic-on-empty*: *homotopic-with* (λ*x*. *True*) {} *t f g*

by (*metis continuous-on-def empty-iff homotopic-with-equal image-subset-iff*)

Homotopy with P is an equivalence relation (on continuous functions mapping X into Y that satisfy P, though this only affects reflexivity.

proposition *homotopic-with-refl*:
homotopic-with P X Y f f ⟷ *continuous-on X f* ∧ *image f X* ⊆ *Y* ∧ *P f*
apply (*rule iffI*)
using *homotopic-with-imp-continuous homotopic-with-imp-property homotopic-with-imp-subset2*
apply *blast*
apply (*simp add*: *homotopic-with-def*)
apply (*rule-tac x=f o snd* **in** *exI*)
apply (*rule conjI continuous-intros* | *force*)+
done

lemma *homotopic-with-symD*:
fixes *X* :: *'a*::*real-normed-vector set*
assumes *homotopic-with P X Y f g*
shows *homotopic-with P X Y g f*
using *assms*
apply (*clarsimp simp add*: *homotopic-with-def*)
apply (*rename-tac h*)
apply (*rule-tac x=h o* (λ*y*. (*1* − *fst y*, *snd y*)) **in** *exI*)
apply (*rule conjI continuous-intros* | *erule continuous-on-subset* | *force simp add*: *image-subset-iff*)+
done

proposition *homotopic-with-sym*:
fixes *X* :: *'a*::*real-normed-vector set*
shows *homotopic-with P X Y f g* ⟷ *homotopic-with P X Y g f*
using *homotopic-with-symD* **by** *blast*

lemma *split-01*: {*0..1*::*real*} = {*0..1/2*} ∪ {*1/2..1*}
by *force*

lemma *split-01-prod*: {*0..1*::*real*} × *X* = ({*0..1/2*} × *X*) ∪ ({*1/2..1*} × *X*)
by *force*

proposition *homotopic-with-trans*:
fixes *X* :: *'a*::*real-normed-vector set*
assumes *homotopic-with P X Y f g* **and** *homotopic-with P X Y g h*
shows *homotopic-with P X Y f h*
proof −
have *clo1*: *closedin* (*subtopology euclidean* ({*0..1/2*} × *X* ∪ {*1/2..1*} × *X*)) ({*0..1/2*::*real*} × *X*)
apply (*simp add*: *closedin-closed split-01-prod* [*symmetric*])
apply (*rule-tac x*={*0..1/2*} × *UNIV* **in** *exI*)
apply (*force simp add*: *closed-Times*)
done
have *clo2*: *closedin* (*subtopology euclidean* ({*0..1/2*} × *X* ∪ {*1/2..1*} × *X*))

```
(\{1/2..1::real\} \times X)
    apply (simp add: closedin-closed split-01-prod [symmetric])
    apply (rule-tac x=\{1/2..1\} \times UNIV in exI)
    apply (force simp add: closed-Times)
    done
  { fix k1 k2:: real \times 'a \Rightarrow 'b
    assume cont: continuous-on (\{0..1\} \times X) k1 continuous-on (\{0..1\} \times X) k2
       and Y: k1 ` (\{0..1\} \times X) \subseteq Y k2 ` (\{0..1\} \times X) \subseteq Y
       and geq: \forall x. k1 (1, x) = g x \forall x. k2 (0, x) = g x
       and k12: \forall x. k1 (0, x) = f x \forall x. k2 (1, x) = h x
       and P:  \forall t\in\{0..1\}. P (\lambdax. k1 (t, x)) \forall t\in\{0..1\}. P (\lambdax. k2 (t, x))
    def k \equiv \lambday. if fst y \leq 1 / 2 then (k1 o (\lambdax. (2 *_R fst x, snd x))) y
                                  else (k2 o (\lambdax. (2 *_R fst x -1, snd x))) y
    have keq: k1 (2 * u, v) = k2 (2 * u - 1, v) if u = 1/2 for u v
      by (simp add: geq that)
    have continuous-on (\{0..1\} \times X) k
      using cont
      apply (simp add: split-01-prod k-def)
        apply (rule clo1 clo2 continuous-on-cases-local continuous-intros | erule
continuous-on-subset | simp add: linear image-subset-iff)+
      apply (force simp add: keq)
      done
    moreover have k ` (\{0..1\} \times X) \subseteq Y
      using Y by (force simp add: k-def)
    moreover have \forall x. k (0, x) = f x
      by (simp add: k-def k12)
    moreover have (\forall x. k (1, x) = h x)
      by (simp add: k-def k12)
    moreover have \forall t\in\{0..1\}. P (\lambdax. k (t, x))
      using P
      apply (clarsimp simp add: k-def)
      apply (case-tac t \leq 1/2)
      apply (auto simp:)
      done
    ultimately have *: \exists k :: real \times 'a \Rightarrow 'b.
                        continuous-on (\{0..1\} \times X) k \wedge k ` (\{0..1\} \times X) \subseteq Y \wedge
                        (\forall x. k (0, x) = f x) \wedge (\forall x. k (1, x) = h x) \wedge (\forall t\in\{0..1\}. P
(\lambdax. k (t, x)))
      by blast
  } note * = this
  show ?thesis
    using assms by (auto intro: * simp add: homotopic-with-def)
qed

proposition homotopic-compose:
     fixes s :: 'a::real-normed-vector set
     shows [\![homotopic-with (\lambdax. True) s t f f'; homotopic-with (\lambdax. True) t u g
g']\!]
           \Longrightarrow homotopic-with (\lambdax. True) s u (g o f) (g' o f')
```

apply (*rule homotopic-with-trans* [**where** $g = g\ o\ f^{-1}$])
apply (*metis homotopic-compose-continuous-left homotopic-with-imp-continuous homotopic-with-imp-subset1*)
by (*metis homotopic-compose-continuous-right homotopic-with-imp-continuous homotopic-with-imp-subset2*)

26.2 Homotopy of paths, maintaining the same endpoints.

definition *homotopic-paths* :: [*'a set*, *real* ⇒ *'a*, *real* ⇒ *'a*::*topological-space*] ⇒ *bool*
where
homotopic-paths s p q ≡
homotopic-with (λ*r*. *pathstart r* = *pathstart p* ∧ *pathfinish r* = *pathfinish p*) {*0..1*} *s p q*

lemma *homotopic-paths*:
homotopic-paths s p q ⟷
(∃ *h*. *continuous-on* ({*0..1*} × {*0..1*}) *h* ∧
h ‘ ({*0..1*} × {*0..1*}) ⊆ *s* ∧
(∀ *x* ∈ {*0..1*}. *h*(*0*,*x*) = *p x*) ∧
(∀ *x* ∈ {*0..1*}. *h*(*1*,*x*) = *q x*) ∧
(∀ *t* ∈ {*0..1*::*real*}. *pathstart*(*h o Pair t*) = *pathstart p* ∧
pathfinish(*h o Pair t*) = *pathfinish p*))
by (*auto simp*: *homotopic-paths-def homotopic-with pathstart-def pathfinish-def*)

proposition *homotopic-paths-imp-pathstart*:
homotopic-paths s p q ⟹ *pathstart p* = *pathstart q*
by (*metis* (*mono-tags*, *lifting*) *homotopic-paths-def homotopic-with-imp-property*)

proposition *homotopic-paths-imp-pathfinish*:
homotopic-paths s p q ⟹ *pathfinish p* = *pathfinish q*
by (*metis* (*mono-tags*, *lifting*) *homotopic-paths-def homotopic-with-imp-property*)

lemma *homotopic-paths-imp-path*:
homotopic-paths s p q ⟹ *path p* ∧ *path q*
using *homotopic-paths-def homotopic-with-imp-continuous path-def* **by** *blast*

lemma *homotopic-paths-imp-subset*:
homotopic-paths s p q ⟹ *path-image p* ⊆ *s* ∧ *path-image q* ⊆ *s*
by (*simp add*: *homotopic-paths-def homotopic-with-imp-subset1 homotopic-with-imp-subset2 path-image-def*)

proposition *homotopic-paths-refl* [*simp*]: *homotopic-paths s p p* ⟷ *path p* ∧ *path-image p* ⊆ *s*
by (*simp add*: *homotopic-paths-def homotopic-with-refl path-def path-image-def*)

proposition *homotopic-paths-sym*: *homotopic-paths s p q* ⟹ *homotopic-paths s q p*
by (*metis* (*mono-tags*) *homotopic-paths-def homotopic-paths-imp-pathfinish homotopic-paths-imp-pathstart*

homotopic-with-symD)

proposition *homotopic-paths-sym-eq*: *homotopic-paths s p q* ⟷ *homotopic-paths s q p*
by (*metis homotopic-paths-sym*)

proposition *homotopic-paths-trans* [*trans*]:
⟦*homotopic-paths s p q*; *homotopic-paths s q r*⟧ ⟹ *homotopic-paths s p r*
apply (*simp add*: *homotopic-paths-def*)
apply (*rule homotopic-with-trans*, *assumption*)
by (*metis* (*mono-tags*, *lifting*) *homotopic-with-imp-property homotopic-with-mono*)

proposition *homotopic-paths-eq*:
⟦*path p*; *path-image p* ⊆ *s*; ⋀*t*. *t* ∈ {*0..1*} ⟹ *p t* = *q t*⟧ ⟹ *homotopic-paths s p q*
apply (*simp add*: *homotopic-paths-def*)
apply (*rule homotopic-with-eq*)
apply (*auto simp*: *path-def homotopic-with-refl pathstart-def pathfinish-def path-image-def elim*: *continuous-on-eq*)
done

proposition *homotopic-paths-reparametrize*:
assumes *path p*
and *pips*: *path-image p* ⊆ *s*
and *contf*: *continuous-on* {*0..1*} *f*
and *f01*:*f* ‘ {*0..1*} ⊆ {*0..1*}
and [*simp*]: *f*(*0*) = *0 f*(*1*) = *1*
and *q*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *q*(*t*) = *p*(*f t*)
shows *homotopic-paths s p q*
proof −
have *contp*: *continuous-on* {*0..1*} *p*
by (*metis* ‹*path p*› *path-def*)
then have *continuous-on* {*0..1*} (*p o f*)
using *contf continuous-on-compose continuous-on-subset f01* **by** *blast*
then have *path q*
by (*simp add*: *path-def*) (*metis q continuous-on-cong*)
have *piqs*: *path-image q* ⊆ *s*
by (*metis* (*no-types*, *hide-lams*) *pips f01 image-subset-iff path-image-def q*)
have *fb0*: ⋀*a b*. ⟦*0* ≤ *a*; *a* ≤ *1*; *0* ≤ *b*; *b* ≤ *1*⟧ ⟹ *0* ≤ (*1* − *a*) ∗ *f b* + *a* ∗ *b*
using *f01* **by** *force*
have *fb1*: ⟦*0* ≤ *a*; *a* ≤ *1*; *0* ≤ *b*; *b* ≤ *1*⟧ ⟹ (*1* − *a*) ∗ *f b* + *a* ∗ *b* ≤ *1* **for** *a b*
using *f01* [*THEN subsetD*, *of f b*] **by** (*simp add*: *convex-bound-le*)
have *homotopic-paths s q p*
proof (*rule homotopic-paths-trans*)
show *homotopic-paths s q* (*p* ∘ *f*)
using *q* **by** (*force intro*: *homotopic-paths-eq* [*OF* ‹*path q*› *piqs*])
next
show *homotopic-paths s* (*p* ∘ *f*) *p*
apply (*simp add*: *homotopic-paths-def homotopic-with-def*)

apply (*rule-tac x=p o* (λ*y*. (*1* − (*fst y*)) ∗$_R$ ((*f o snd*) *y*) + (*fst y*) ∗$_R$ *snd y*) **in** *exI*)
apply (*rule conjI contf continuous-intros continuous-on-subset* [*OF contp*] | *simp*)+
using *pips* [*unfolded path-image-def*]
apply (*auto simp*: *fb0 fb1 pathstart-def pathfinish-def*)
done
qed
then show *?thesis*
by (*simp add*: *homotopic-paths-sym*)
qed

lemma *homotopic-paths-subset*: ⟦*homotopic-paths s p q*; *s* ⊆ *t*⟧ ⟹ *homotopic-paths t p q*
using *homotopic-paths-def homotopic-with-subset-right* **by** *blast*

A slightly ad-hoc but useful lemma in constructing homotopies.

lemma *homotopic-join-lemma*:
fixes *q* :: [*real*,*real*] ⇒ *'a*::*topological-space*
assumes *p*: *continuous-on* ({*0..1*} × {*0..1*}) (λ*y*. *p* (*fst y*) (*snd y*))
and *q*: *continuous-on* ({*0..1*} × {*0..1*}) (λ*y*. *q* (*fst y*) (*snd y*))
and *pf*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *pathfinish*(*p t*) = *pathstart*(*q t*)
shows *continuous-on* ({*0..1*} × {*0..1*}) (λ*y*. (*p*(*fst y*) +++ *q*(*fst y*)) (*snd y*))
proof −
have *1*: (λ*y*. *p* (*fst y*) (*2* ∗ *snd y*)) = (λ*y*. *p* (*fst y*) (*snd y*)) *o* (λ*y*. (*fst y*, *2* ∗ *snd y*))
by (*rule ext*) (*simp*)
have *2*: (λ*y*. *q* (*fst y*) (*2* ∗ *snd y* − *1*)) = (λ*y*. *q* (*fst y*) (*snd y*)) *o* (λ*y*. (*fst y*, *2* ∗ *snd y* − *1*))
by (*rule ext*) (*simp*)
show *?thesis*
apply (*simp add*: *joinpaths-def*)
apply (*rule continuous-on-cases-le*)
apply (*simp-all only*: *1 2*)
apply (*rule continuous-intros continuous-on-subset* [*OF p*] *continuous-on-subset* [*OF q*] | *force*)+
using *pf*
apply (*auto simp*: *mult.commute pathstart-def pathfinish-def*)
done
qed

Congruence properties of homotopy w.r.t. path-combining operations.

lemma *homotopic-paths-reversepath-D*:
assumes *homotopic-paths s p q*
shows *homotopic-paths s* (*reversepath p*) (*reversepath q*)
using *assms*
apply (*simp add*: *homotopic-paths-def homotopic-with-def*, *clarify*)
apply (*rule-tac x=h o* (λ*x*. (*fst x*, *1* − *snd x*)) **in** *exI*)
apply (*rule conjI continuous-intros*)+

apply (*auto simp*: *reversepath-def pathstart-def pathfinish-def elim*!: *continuous-on-subset*)
done

proposition *homotopic-paths-reversepath*:
homotopic-paths s (*reversepath p*) (*reversepath q*) ⟷ *homotopic-paths s p q*
using *homotopic-paths-reversepath-D* **by** *force*

proposition *homotopic-paths-join*:
⟦*homotopic-paths s p p'*; *homotopic-paths s q q'*; *pathfinish p* = *pathstart q*⟧ ⟹ *homotopic-paths s* (*p* +++ *q*) (*p'* +++ *q'*)
apply (*simp add*: *homotopic-paths-def homotopic-with-def*, *clarify*)
apply (*rename-tac k1 k2*)
apply (*rule-tac x*=(λ*y*. ((*k1 o Pair* (*fst y*)) +++ (*k2 o Pair* (*fst y*))) (*snd y*)) **in** *exI*)
apply (*rule conjI continuous-intros homotopic-join-lemma*)+
apply (*auto simp*: *joinpaths-def pathstart-def pathfinish-def path-image-def*)
done

proposition *homotopic-paths-continuous-image*:
⟦*homotopic-paths s f g*; *continuous-on s h*; *h* ' *s* ⊆ *t*⟧ ⟹ *homotopic-paths t* (*h o f*) (*h o g*)
unfolding *homotopic-paths-def*
apply (*rule homotopic-with-compose-continuous-left* [*of* - - - *s*])
apply (*auto simp*: *pathstart-def pathfinish-def elim*!: *homotopic-with-mono*)
done

26.3 Group properties for homotopy of paths

So taking equivalence classes under homotopy would give the fundamental group

proposition *homotopic-paths-rid*:
⟦*path p*; *path-image p* ⊆ *s*⟧ ⟹ *homotopic-paths s* (*p* +++ *linepath* (*pathfinish p*) (*pathfinish p*)) *p*
apply (*subst homotopic-paths-sym*)
apply (*rule homotopic-paths-reparametrize* [**where** *f* = λ*t*. *if t* ≤ *1* / *2 then 2* *$_R$ t else 1*])
apply (*simp-all del*: *le-divide-eq-numeral1*)
apply (*subst split-01*)
apply (*rule continuous-on-cases continuous-intros* | *force simp*: *pathfinish-def joinpaths-def*)+
done

proposition *homotopic-paths-lid*:
⟦*path p*; *path-image p* ⊆ *s*⟧ ⟹ *homotopic-paths s* (*linepath* (*pathstart p*) (*pathstart p*) +++ *p*) *p*
using *homotopic-paths-rid* [*of reversepath p s*]
by (*metis homotopic-paths-reversepath path-image-reversepath path-reversepath pathfinish-linepath*

pathfinish-reversepath reversepath-joinpaths reversepath-linepath)

proposition *homotopic-paths-assoc*:
⟦*path p*; *path-image p* ⊆ *s*; *path q*; *path-image q* ⊆ *s*; *path r*; *path-image r* ⊆ *s*; *pathfinish p* = *pathstart q*;
pathfinish q = *pathstart r*⟧
⟹ *homotopic-paths s* (*p* +++ (*q* +++ *r*)) ((*p* +++ *q*) +++ *r*)
apply (*subst homotopic-paths-sym*)
apply (*rule homotopic-paths-reparametrize*
[**where** *f* = λ*t*. *if* *t* ≤ *1* / *2* *then inverse 2* ∗*R* *t*
else if *t* ≤ *3* / *4* *then t* − (*1* / *4*)
else 2 ∗*R t* − *1*])
apply (*simp-all del*: *le-divide-eq-numeral1*)
apply (*simp add*: *subset-path-image-join*)
apply (*rule continuous-on-cases-1 continuous-intros*)+
apply (*auto simp*: *joinpaths-def*)
done

proposition *homotopic-paths-rinv*:
assumes *path p path-image p* ⊆ *s*
shows *homotopic-paths s* (*p* +++ *reversepath p*) (*linepath* (*pathstart p*) (*pathstart p*))
proof −
have *continuous-on* ({*0..1*} × {*0..1*}) (λ*x*. (*subpath 0* (*fst x*) *p* +++ *reversepath* (*subpath 0* (*fst x*) *p*)) (*snd x*))
using *assms*
apply (*simp add*: *joinpaths-def subpath-def reversepath-def path-def del*: *le-divide-eq-numeral1*)
apply (*rule continuous-on-cases-le*)
apply (*rule-tac* [*2*] *continuous-on-compose* [*of* - - *p*, *unfolded o-def*])
apply (*rule continuous-on-compose* [*of* - - *p*, *unfolded o-def*])
apply (*auto intro*!: *continuous-intros simp del*: *eq-divide-eq-numeral1*)
apply (*force elim*!: *continuous-on-subset simp add*: *mult-le-one*)+
done
then show *?thesis*
using *assms*
apply (*subst homotopic-paths-sym-eq*)
unfolding *homotopic-paths-def homotopic-with-def*
apply (*rule-tac x*=(λ*y*. (*subpath 0* (*fst y*) *p* +++ *reversepath*(*subpath 0* (*fst y*) *p*)) (*snd y*)) **in** *exI*)
apply (*simp add*: *path-defs joinpaths-def subpath-def reversepath-def*)
apply (*force simp*: *mult-le-one*)
done
qed

proposition *homotopic-paths-linv*:
assumes *path p path-image p* ⊆ *s*
shows *homotopic-paths s* (*reversepath p* +++ *p*) (*linepath* (*pathfinish p*) (*pathfinish p*))
using *homotopic-paths-rinv* [*of reversepath p s*] *assms* **by** *simp*

26.4 Homotopy of loops without requiring preservation of endpoints.

definition *homotopic-loops* :: *'a::topological-space set* ⇒ (*real* ⇒ *'a*) ⇒ (*real* ⇒ *'a*) ⇒ *bool* **where**
homotopic-loops s p q ≡
homotopic-with (λ*r*. *pathfinish r* = *pathstart r*) {*0..1*} *s p q*

lemma *homotopic-loops*:
homotopic-loops s p q ⟷
(∃ *h*. *continuous-on* ({*0..1*::*real*} × {*0..1*}) *h* ∧
image h ({*0..1*} × {*0..1*}) ⊆ *s* ∧
(∀ *x* ∈ {*0..1*}. *h*(*0*,*x*) = *p x*) ∧
(∀ *x* ∈ {*0..1*}. *h*(*1*,*x*) = *q x*) ∧
(∀ *t* ∈ {*0..1*}. *pathfinish*(*h o Pair t*) = *pathstart*(*h o Pair t*)))
by (*simp add*: *homotopic-loops-def pathstart-def pathfinish-def homotopic-with*)

proposition *homotopic-loops-imp-loop*:
homotopic-loops s p q ⟹ *pathfinish p* = *pathstart p* ∧ *pathfinish q* = *pathstart q*
using *homotopic-with-imp-property homotopic-loops-def* **by** *blast*

proposition *homotopic-loops-imp-path*:
homotopic-loops s p q ⟹ *path p* ∧ *path q*
unfolding *homotopic-loops-def path-def*
using *homotopic-with-imp-continuous* **by** *blast*

proposition *homotopic-loops-imp-subset*:
homotopic-loops s p q ⟹ *path-image p* ⊆ *s* ∧ *path-image q* ⊆ *s*
unfolding *homotopic-loops-def path-image-def*
by (*metis homotopic-with-imp-subset1 homotopic-with-imp-subset2*)

proposition *homotopic-loops-refl*:
homotopic-loops s p p ⟷
path p ∧ *path-image p* ⊆ *s* ∧ *pathfinish p* = *pathstart p*
by (*simp add*: *homotopic-loops-def homotopic-with-refl path-image-def path-def*)

proposition *homotopic-loops-sym*: *homotopic-loops s p q* ⟹ *homotopic-loops s q p*
by (*simp add*: *homotopic-loops-def homotopic-with-sym*)

proposition *homotopic-loops-sym-eq*: *homotopic-loops s p q* ⟷ *homotopic-loops s q p*
by (*metis homotopic-loops-sym*)

proposition *homotopic-loops-trans*:
⟦*homotopic-loops s p q*; *homotopic-loops s q r*⟧ ⟹ *homotopic-loops s p r*
unfolding *homotopic-loops-def* **by** (*blast intro*: *homotopic-with-trans*)

proposition *homotopic-loops-subset*:

⟦*homotopic-loops s p q*; *s* ⊆ *t*⟧ ⟹ *homotopic-loops t p q*
by (*simp add*: *homotopic-loops-def homotopic-with-subset-right*)

proposition *homotopic-loops-eq*:
⟦*path p*; *path-image p* ⊆ *s*; *pathfinish p* = *pathstart p*; ⋀*t*. *t* ∈ {*0..1*} ⟹ *p*(*t*) = *q*(*t*)⟧
⟹ *homotopic-loops s p q*
unfolding *homotopic-loops-def*
apply (*rule homotopic-with-eq*)
apply (*rule homotopic-with-refl* [**where** *f* = *p*, *THEN iffD2*])
apply (*simp-all add*: *path-image-def path-def pathstart-def pathfinish-def*)
done

proposition *homotopic-loops-continuous-image*:
⟦*homotopic-loops s f g*; *continuous-on s h*; *h* ‘ *s* ⊆ *t*⟧ ⟹ *homotopic-loops t* (*h* ∘ *f*) (*h* ∘ *g*)
unfolding *homotopic-loops-def*
apply (*rule homotopic-with-compose-continuous-left*)
apply (*erule homotopic-with-mono*)
by (*simp add*: *pathfinish-def pathstart-def*)

26.5 Relations between the two variants of homotopy

proposition *homotopic-paths-imp-homotopic-loops*:
⟦*homotopic-paths s p q*; *pathfinish p* = *pathstart p*; *pathfinish q* = *pathstart p*⟧
⟹ *homotopic-loops s p q*
by (*auto simp*: *homotopic-paths-def homotopic-loops-def intro*: *homotopic-with-mono*)

proposition *homotopic-loops-imp-homotopic-paths-null*:
assumes *homotopic-loops s p* (*linepath a a*)
shows *homotopic-paths s p* (*linepath* (*pathstart p*) (*pathstart p*))
proof −
have *path p* **by** (*metis assms homotopic-loops-imp-path*)
have *ploop*: *pathfinish p* = *pathstart p* **by** (*metis assms homotopic-loops-imp-loop*)
have *pip*: *path-image p* ⊆ *s* **by** (*metis assms homotopic-loops-imp-subset*)
obtain *h* **where** *conth*: *continuous-on* ({*0..1*::*real*} × {*0..1*}) *h*
and *hs*: *h* ‘ ({*0..1*} × {*0..1*}) ⊆ *s*
and [*simp*]: ⋀*x*. *x* ∈ {*0..1*} ⟹ *h*(*0*,*x*) = *p x*
and [*simp*]: ⋀*x*. *x* ∈ {*0..1*} ⟹ *h*(*1*,*x*) = *a*
and *ends*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *pathfinish* (*h* ∘ *Pair t*) = *pathstart* (*h* ∘ *Pair t*)
using *assms* **by** (*auto simp*: *homotopic-loops homotopic-with*)
have *conth0*: *path* (λ*u*. *h* (*u*, *0*))
unfolding *path-def*
apply (*rule continuous-on-compose* [*of* - - *h*, *unfolded o-def*])
apply (*force intro*: *continuous-intros continuous-on-subset* [*OF conth*])+
done
have *pih0*: *path-image* (λ*u*. *h* (*u*, *0*)) ⊆ *s*
using *hs* **by** (*force simp*: *path-image-def*)

have *c1*: *continuous-on* ({*0..1*} × {*0..1*}) (λ*x*. *h* (*fst x* ∗ *snd x*, *0*))
apply (*rule continuous-on-compose* [*of - - h*, *unfolded o-def*])
apply (*force simp*: *mult-le-one intro*: *continuous-intros continuous-on-subset* [*OF conth*])+
done
have *c2*: *continuous-on* ({*0..1*} × {*0..1*}) (λ*x*. *h* (*fst x* − *fst x* ∗ *snd x*, *0*))
apply (*rule continuous-on-compose* [*of - - h*, *unfolded o-def*])
apply (*force simp*: *mult-left-le mult-le-one intro*: *continuous-intros continuous-on-subset* [*OF conth*])+
apply (*rule continuous-on-subset* [*OF conth*])
apply (*auto simp*: *algebra-simps add-increasing2 mult-left-le*)
done
have [*simp*]: ⋀*t*. ⟦*0* ≤ *t* ∧ *t* ≤ *1*⟧ ⟹ *h* (*t*, *1*) = *h* (*t*, *0*)
using *ends* **by** (*simp add*: *pathfinish-def pathstart-def*)
have *adhoc-le*: *c* ∗ *4* ≤ *1* + *c* ∗ (*d* ∗ *4*) **if** ¬ *d* ∗ *4* ≤ *3* *0* ≤ *c* *c* ≤ *1* **for** *c d*::*real*
proof −
have *c* ∗ *3* ≤ *c* ∗ (*d* ∗ *4*) **using** *that less-eq-real-def* **by** *auto*
with ‹*c* ≤ *1*› **show** *?thesis* **by** *fastforce*
qed
have ∗: ⋀*p* *x*. (*path p* ∧ *path*(*reversepath p*)) ∧
(*path-image p* ⊆ *s* ∧ *path-image*(*reversepath p*) ⊆ *s*) ∧
(*pathfinish p* = *pathstart*(*linepath a a* +++ *reversepath p*) ∧
pathstart(*reversepath p*) = *a*) ∧ *pathstart p* = *x*
⟹ *homotopic-paths s* (*p* +++ *linepath a a* +++ *reversepath p*) (*linepath x x*)
by (*metis homotopic-paths-lid homotopic-paths-join*
homotopic-paths-trans homotopic-paths-sym homotopic-paths-rinv)
have *1*: *homotopic-paths s p* (*p* +++ *linepath* (*pathfinish p*) (*pathfinish p*))
using ‹*path p*› *homotopic-paths-rid homotopic-paths-sym pip* **by** *blast*
moreover have *homotopic-paths s* (*p* +++ *linepath* (*pathfinish p*) (*pathfinish p*))
(*linepath* (*pathstart p*) (*pathstart p*) +++ *p* +++ *linepath* (*pathfinish p*) (*pathfinish p*))
apply (*rule homotopic-paths-sym*)
using *homotopic-paths-lid* [*of p* +++ *linepath* (*pathfinish p*) (*pathfinish p*) *s*]
by (*metis 1 homotopic-paths-imp-path homotopic-paths-imp-pathstart homotopic-paths-imp-subset*)
moreover have *homotopic-paths s* (*linepath* (*pathstart p*) (*pathstart p*) +++ *p* +++ *linepath* (*pathfinish p*) (*pathfinish p*))
((λ*u*. *h* (*u*, *0*)) +++ *linepath a a* +++ *reversepath* (λ*u*. *h* (*u*, *0*)))
apply (*simp add*: *homotopic-paths-def homotopic-with-def*)
apply (*rule-tac x*=λ*y*. (*subpath 0* (*fst y*) (λ*u*. *h* (*u*, *0*)) +++ (λ*u*. *h* (*Pair* (*fst y*) *u*)) +++ *subpath* (*fst y*) *0* (λ*u*. *h* (*u*, *0*))) (*snd y*) **in** *exI*)
apply (*simp add*: *subpath-reversepath*)
apply (*intro conjI homotopic-join-lemma*)
using *ploop*
apply (*simp-all add*: *path-defs joinpaths-def o-def subpath-def conth c1 c2*)
apply (*force simp*: *algebra-simps mult-le-one mult-left-le intro*: *hs* [*THEN subsetD*] *adhoc-le*)

done
moreover have *homotopic-paths s* ((λ*u*. *h* (*u*, *0*)) *+++ linepath a a +++ reversepath* (λ*u*. *h* (*u*, *0*)))
(*linepath* (*pathstart p*) (*pathstart p*))
apply (*rule* ∗)
apply (*simp add*: *pih0 pathstart-def pathfinish-def conth0*)
apply (*simp add*: *reversepath-def joinpaths-def*)
done
ultimately show *?thesis*
by (*blast intro*: *homotopic-paths-trans*)
qed

proposition *homotopic-loops-conjugate*:
fixes *s* :: *'a*::*real-normed-vector set*
assumes *path p path q* **and** *pip*: *path-image p* ⊆ *s* **and** *piq*: *path-image q* ⊆ *s*
and *papp*: *pathfinish p* = *pathstart q* **and** *qloop*: *pathfinish q* = *pathstart q*
shows *homotopic-loops s* (*p +++ q +++ reversepath p*) *q*
proof −
have *contp*: *continuous-on* {*0..1*} *p* **using** ‹*path p*› [*unfolded path-def*] **by** *blast*
have *contq*: *continuous-on* {*0..1*} *q* **using** ‹*path q*› [*unfolded path-def*] **by** *blast*
have *c1*: *continuous-on* ({*0..1*} × {*0..1*}) (λ*x*. *p* ((*1* − *fst x*) ∗ *snd x* + *fst x*))
apply (*rule continuous-on-compose* [*of - - p*, *unfolded o-def*])
apply (*force simp*: *mult-le-one intro*!: *continuous-intros*)
apply (*rule continuous-on-subset* [*OF contp*])
apply (*auto simp*: *algebra-simps add-increasing2 mult-right-le-one-le sum-le-prod1*)
done
have *c2*: *continuous-on* ({*0..1*} × {*0..1*}) (λ*x*. *p* ((*fst x* − *1*) ∗ *snd x* + *1*))
apply (*rule continuous-on-compose* [*of - - p*, *unfolded o-def*])
apply (*force simp*: *mult-le-one intro*!: *continuous-intros*)
apply (*rule continuous-on-subset* [*OF contp*])
apply (*auto simp*: *algebra-simps add-increasing2 mult-left-le-one-le*)
done
have *ps1*: ⋀*a b*. ⟦*b* ∗ *2* ≤ *1*; *0* ≤ *b*; *0* ≤ *a*; *a* ≤ *1*⟧ ⟹ *p* ((*1* − *a*) ∗ (*2* ∗ *b*) + *a*) ∈ *s*
using *sum-le-prod1*
by (*force simp*: *algebra-simps add-increasing2 mult-left-le intro*: *pip* [*unfolded path-image-def*, *THEN subsetD*])
have *ps2*: ⋀*a b*. ⟦¬ *4* ∗ *b* ≤ *3*; *b* ≤ *1*; *0* ≤ *a*; *a* ≤ *1*⟧ ⟹ *p* ((*a* − *1*) ∗ (*4* ∗ *b* − *3*) + *1*) ∈ *s*
apply (*rule pip* [*unfolded path-image-def*, *THEN subsetD*])
apply (*rule image-eqI*, *blast*)
apply (*simp add*: *algebra-simps*)
by (*metis add-mono-thms-linordered-semiring*(*1*) *affine-ineq linear mult.commute mult.left-neutral mult-right-mono not-le*
add.commute zero-le-numeral)
have *qs*: ⋀*a b*. ⟦*4* ∗ *b* ≤ *3*; ¬ *b* ∗ *2* ≤ *1*⟧ ⟹ *q* (*4* ∗ *b* − *2*) ∈ *s*
using *path-image-def piq* **by** *fastforce*
have *homotopic-loops s* (*p +++ q +++ reversepath p*)
(*linepath* (*pathstart q*) (*pathstart q*) *+++ q +++ linepath*

(*pathstart q*) (*pathstart q*))
apply (*simp add*: *homotopic-loops-def homotopic-with-def*)
apply (*rule-tac x*=(λ*y*. (*subpath* (*fst y*) *1 p* +++ *q* +++ *subpath 1* (*fst y*) *p*) (*snd y*)) **in** *exI*)
apply (*simp add*: *subpath-refl subpath-reversepath*)
apply (*intro conjI homotopic-join-lemma*)
using *papp qloop*
apply (*simp-all add*: *path-defs joinpaths-def o-def subpath-def c1 c2*)
apply (*force simp*: *contq intro*: *continuous-on-compose* [*of - - q*, *unfolded o-def*] *continuous-on-id continuous-on-snd*)
apply (*auto simp*: *ps1 ps2 qs*)
done
moreover have *homotopic-loops s* (*linepath* (*pathstart q*) (*pathstart q*) +++ *q* +++ *linepath* (*pathstart q*) (*pathstart q*)) *q*
proof −
have *homotopic-paths s* (*linepath* (*pathfinish q*) (*pathfinish q*) +++ *q*) *q*
using ‹*path q*› *homotopic-paths-lid qloop piq* **by** *auto*
hence *1*: ⋀*f*. *homotopic-paths s f q* ∨ ¬ *homotopic-paths s f* (*linepath* (*pathfinish q*) (*pathfinish q*) +++ *q*)
using *homotopic-paths-trans* **by** *blast*
hence *homotopic-paths s* (*linepath* (*pathfinish q*) (*pathfinish q*) +++ *q* +++ *linepath* (*pathfinish q*) (*pathfinish q*)) *q*
proof −
have *homotopic-paths s* (*q* +++ *linepath* (*pathfinish q*) (*pathfinish q*)) *q*
by (*simp add*: ‹*path q*› *homotopic-paths-rid piq*)
thus *?thesis*
by (*metis* (*no-types*) *1* ‹*path q*› *homotopic-paths-join homotopic-paths-rinv homotopic-paths-sym*
homotopic-paths-trans qloop pathfinish-linepath piq)
qed
thus *?thesis*
by (*metis* (*no-types*) *qloop homotopic-loops-sym homotopic-paths-imp-homotopic-loops homotopic-paths-imp-pathfinish homotopic-paths-sym*)
qed
ultimately show *?thesis*
by (*blast intro*: *homotopic-loops-trans*)
qed

26.6 Homotopy of "nearby" function, paths and loops.

lemma *homotopic-with-linear*:
fixes *f g* :: - ⇒ *'b*::*real-normed-vector*
assumes *contf*: *continuous-on s f*
and *contg*:*continuous-on s g*
and *sub*: ⋀*x*. *x* ∈ *s* ⟹ *closed-segment* (*f x*) (*g x*) ⊆ *t*
shows *homotopic-with* (λ*z*. *True*) *s t f g*
apply (*simp add*: *homotopic-with-def*)
apply (*rule-tac x*=λ*y*. ((*1* − (*fst y*)) *R f*(*snd y*) + (*fst y*) *R g*(*snd y*)) **in** *exI*)
apply (*intro conjI*)

apply (*rule subset-refl continuous-intros continuous-on-subset* [*OF contf*] *continuous-on-compose2* [**where** *g=f*]
continuous-on-subset [*OF contg*] *continuous-on-compose2* [**where** *g=g*]| *simp*)+
using *sub closed-segment-def* **apply** *fastforce+*
done

lemma *homotopic-paths-linear*:
fixes *g h* :: *real* ⇒ *'a*::*real-normed-vector*
assumes *path g path h pathstart h* = *pathstart g pathfinish h* = *pathfinish g*
⋀*t x*. *t* ∈ {*0..1*} ⟹ *closed-segment* (*g t*) (*h t*) ⊆ *s*
shows *homotopic-paths s g h*
using *assms*
unfolding *path-def*
apply (*simp add*: *pathstart-def pathfinish-def homotopic-paths-def homotopic-with-def*)
apply (*rule-tac x=λy*. ((*1* − (*fst y*)) ∗*R g*(*snd y*) + (*fst y*) ∗*R h*(*snd y*)) **in** *exI*)
apply (*auto intro*!: *continuous-intros intro*: *continuous-on-compose2* [**where** *g=g*] *continuous-on-compose2* [**where** *g=h*])
apply (*force simp*: *closed-segment-def*)
apply (*simp-all add*: *algebra-simps*)
done

lemma *homotopic-loops-linear*:
fixes *g h* :: *real* ⇒ *'a*::*real-normed-vector*
assumes *path g path h pathfinish g* = *pathstart g pathfinish h* = *pathstart h*
⋀*t x*. *t* ∈ {*0..1*} ⟹ *closed-segment* (*g t*) (*h t*) ⊆ *s*
shows *homotopic-loops s g h*
using *assms*
unfolding *path-def*
apply (*simp add*: *pathstart-def pathfinish-def homotopic-loops-def homotopic-with-def*)
apply (*rule-tac x=λy*. ((*1* − (*fst y*)) ∗*R g*(*snd y*) + (*fst y*) ∗*R h*(*snd y*)) **in** *exI*)
apply (*auto intro*!: *continuous-intros intro*: *continuous-on-compose2* [**where** *g=g*] *continuous-on-compose2* [**where** *g=h*])
apply (*force simp*: *closed-segment-def*)
done

lemma *homotopic-paths-nearby-explicit*:
assumes *path g path h pathstart h* = *pathstart g pathfinish h* = *pathfinish g*
and *no*: ⋀*t x*. ⟦*t* ∈ {*0..1*}; *x* ∉ *s*⟧ ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *x*)
shows *homotopic-paths s g h*
apply (*rule homotopic-paths-linear* [*OF assms*(*1*−*4*)])
by (*metis no segment-bound*(*1*) *subsetI norm-minus-commute not-le*)

lemma *homotopic-loops-nearby-explicit*:
assumes *path g path h pathfinish g* = *pathstart g pathfinish h* = *pathstart h*
and *no*: ⋀*t x*. ⟦*t* ∈ {*0..1*}; *x* ∉ *s*⟧ ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *x*)
shows *homotopic-loops s g h*
apply (*rule homotopic-loops-linear* [*OF assms*(*1*−*4*)])
by (*metis no segment-bound*(*1*) *subsetI norm-minus-commute not-le*)

lemma *homotopic-nearby-paths*:
fixes *g h* :: *real* ⇒ *'a*::*euclidean-space*
assumes *path g open s path-image g* ⊆ *s*
shows ∃ *e*. *0* < *e* ∧
(∀ *h*. *path h* ∧
pathstart h = *pathstart g* ∧ *pathfinish h* = *pathfinish g* ∧
(∀ *t* ∈ {*0..1*}. *norm*(*h t* − *g t*) < *e*) ⟶ *homotopic-paths s g h*)
proof −
obtain *e* **where** *e* > *0* **and** *e*: ⋀*x y*. *x* ∈ *path-image g* ⟹ *y* ∈ − *s* ⟹ *e* ≤ *dist x y*
using *separate-compact-closed* [*of path-image g* − *s*] *assms* **by** *force*
show *?thesis*
apply (*intro exI conjI*)
using *e* [*unfolded dist-norm*]
apply (*auto simp*: *intro*!: *homotopic-paths-nearby-explicit assms* ‹*e* > *0*›)
by (*metis atLeastAtMost-iff imageI le-less-trans not-le path-image-def*)
qed

lemma *homotopic-nearby-loops*:
fixes *g h* :: *real* ⇒ *'a*::*euclidean-space*
assumes *path g open s path-image g* ⊆ *s pathfinish g* = *pathstart g*
shows ∃ *e*. *0* < *e* ∧
(∀ *h*. *path h* ∧ *pathfinish h* = *pathstart h* ∧
(∀ *t* ∈ {*0..1*}. *norm*(*h t* − *g t*) < *e*) ⟶ *homotopic-loops s g h*)
proof −
obtain *e* **where** *e* > *0* **and** *e*: ⋀*x y*. *x* ∈ *path-image g* ⟹ *y* ∈ − *s* ⟹ *e* ≤ *dist x y*
using *separate-compact-closed* [*of path-image g* − *s*] *assms* **by** *force*
show *?thesis*
apply (*intro exI conjI*)
using *e* [*unfolded dist-norm*]
apply (*auto simp*: *intro*!: *homotopic-loops-nearby-explicit assms* ‹*e* > *0*›)
by (*metis atLeastAtMost-iff imageI le-less-trans not-le path-image-def*)
qed

26.7 Homotopy and subpaths

lemma *homotopic-join-subpaths1*:
assumes *path g* **and** *pag*: *path-image g* ⊆ *s*
and *u*: *u* ∈ {*0..1*} **and** *v*: *v* ∈ {*0..1*} **and** *w*: *w* ∈ {*0..1*} *u* ≤ *v v* ≤ *w*
shows *homotopic-paths s* (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
proof −
have *1*: *t* ∗ *2* ≤ *1* ⟹ *u* + *t* ∗ (*v* ∗ *2*) ≤ *v* + *t* ∗ (*u* ∗ *2*) **for** *t*
using *affine-ineq* ‹*u* ≤ *v*› **by** *fastforce*
have *2*: *t* ∗ *2* > *1* ⟹ *u* + (*2*∗*t* − *1*) ∗ *v* ≤ *v* + (*2*∗*t* − *1*) ∗ *w* **for** *t*
by (*metis add-mono-thms-linordered-semiring*(*1*) *diff-gt-0-iff-gt less-eq-real-def mult.commute mult-right-mono* ‹*u* ≤ *v*› ‹*v* ≤ *w*›)
have *t2*: ⋀*t*::*real*. *t*∗*2* = *1* ⟹ *t* = *1*/*2* **by** *auto*

show *?thesis*
apply (*rule homotopic-paths-subset* [*OF - pag*])
using *assms*
apply (*cases w = u*)
using *homotopic-paths-rinv* [*of subpath u v g path-image g*]
apply (*force simp*: *closed-segment-eq-real-ivl image-mono path-image-def subpath-refl*)
apply (*rule homotopic-paths-sym*)
apply (*rule homotopic-paths-reparametrize*
[**where** *f* = λ*t*. *if t* ≤ *1 / 2*
then inverse((*w* − *u*)) ∗*R* (*2* ∗ (*v* − *u*)) ∗*R t*
else inverse((*w* − *u*)) ∗*R* ((*v* − *u*) + (*w* − *v*) ∗*R* (*2* ∗*R t* − *1*))])
using ‹*path g*› *path-subpath u w* **apply** *blast*
using ‹*path g*› *path-image-subpath-subset u w*(*1*) **apply** *blast*
apply *simp-all*
apply (*subst split-01*)
apply (*rule continuous-on-cases continuous-intros* | *force simp*: *pathfinish-def joinpaths-def*)+
apply (*simp-all add*: *field-simps not-le*)
apply (*force dest*!: *t2*)
apply (*force simp*: *algebra-simps mult-left-mono affine-ineq dest*!: *1 2*)
apply (*simp add*: *joinpaths-def subpath-def*)
apply (*force simp*: *algebra-simps*)
done
qed

lemma *homotopic-join-subpaths2*:
assumes *homotopic-paths s* (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
shows *homotopic-paths s* (*subpath w v g* +++ *subpath v u g*) (*subpath w u g*)
by (*metis assms homotopic-paths-reversepath-D pathfinish-subpath pathstart-subpath reversepath-joinpaths reversepath-subpath*)

lemma *homotopic-join-subpaths3*:
assumes *hom*: *homotopic-paths s* (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
and *path g* **and** *pag*: *path-image g* ⊆ *s*
and *u*: *u* ∈ {*0..1*} **and** *v*: *v* ∈ {*0..1*} **and** *w*: *w* ∈ {*0..1*}
shows *homotopic-paths s* (*subpath v w g* +++ *subpath w u g*) (*subpath v u g*)
proof −
have *homotopic-paths s* (*subpath u w g* +++ *subpath w v g*) ((*subpath u v g* +++ *subpath v w g*) +++ *subpath w v g*)
apply (*rule homotopic-paths-join*)
using *hom homotopic-paths-sym-eq* **apply** *blast*
apply (*metis* ‹*path g*› *homotopic-paths-eq pag path-image-subpath-subset path-subpath subset-trans v w*)
apply (*simp add*:)
done
also have *homotopic-paths s* ((*subpath u v g* +++ *subpath v w g*) +++ *subpath w v g*) (*subpath u v g* +++ *subpath v w g* +++ *subpath w v g*)

apply (*rule homotopic-paths-sym* [*OF homotopic-paths-assoc*])
using *assms* **by** (*simp-all add*: *path-image-subpath-subset* [*THEN order-trans*])
also have *homotopic-paths s* (*subpath u v g +++ subpath v w g +++ subpath w v g*)
(*subpath u v g +++ linepath* (*pathfinish* (*subpath u v g*)) (*pathfinish* (*subpath u v g*)))
apply (*rule homotopic-paths-join*)
apply (*metis* ‹*path g*› *homotopic-paths-eq order.trans pag path-image-subpath-subset path-subpath u v*)
apply (*metis* (*no-types*, *lifting*) ‹*path g*› *homotopic-paths-linv order-trans pag path-image-subpath-subset path-subpath pathfinish-subpath reversepath-subpath v w*)
apply (*simp add*:)
done
also have *homotopic-paths s* (*subpath u v g +++ linepath* (*pathfinish* (*subpath u v g*)) (*pathfinish* (*subpath u v g*))) (*subpath u v g*)
apply (*rule homotopic-paths-rid*)
using ‹*path g*› *path-subpath u v* **apply** *blast*
apply (*meson* ‹*path g*› *order.trans pag path-image-subpath-subset u v*)
done
finally have *homotopic-paths s* (*subpath u w g +++ subpath w v g*) (*subpath u v g*) **.**
then show *?thesis*
using *homotopic-join-subpaths2* **by** *blast*
qed

proposition *homotopic-join-subpaths*:
⟦*path g*; *path-image g* ⊆ *s*; *u* ∈ {*0..1*}; *v* ∈ {*0..1*}; *w* ∈ {*0..1*}⟧
⟹ *homotopic-paths s* (*subpath u v g +++ subpath v w g*) (*subpath u w g*)
apply (*rule le-cases3* [*of u v w*])
using *homotopic-join-subpaths1 homotopic-join-subpaths2 homotopic-join-subpaths3*
by *metis+*

end

27 A generic phantom type

theory *Phantom-Type*
imports *Main*
begin

datatype ('*a*, '*b*) *phantom* = *phantom* (*of-phantom*: '*b*)

lemma *type-definition-phantom'*: *type-definition of-phantom phantom UNIV*
by(*unfold-locales*) *simp-all*

lemma *phantom-comp-of-phantom* [*simp*]: *phantom* ∘ *of-phantom* = *id*
and *of-phantom-comp-phantom* [*simp*]: *of-phantom* ∘ *phantom* = *id*
by(*simp-all add*: *o-def id-def*)

syntax *-Phantom* :: *type* ⇒ *logic* ((*1Phantom*/(*1'*(-'))))
translations
Phantom('*t*) => *CONST phantom* :: - ⇒ ('*t*, -) *phantom*

typed-print-translation ‹
let
fun phantom-tr' ctxt (*Type* (@{*type-name fun*}, [-, *Type* (@{*type-name phantom*}, [*T*, -])])) *ts* =
list-comb
(*Syntax.const* @{*syntax-const -Phantom*} $ *Syntax-Phases.term-of-typ ctxt T*, *ts*)
| *phantom-tr' - - - = raise Match*;
in [(@{*const-syntax phantom*}, *phantom-tr'*)] *end*
›

lemma *of-phantom-inject* [*simp*]:
of-phantom x = *of-phantom y* ⟷ *x* = *y*
by(*cases x y rule*: *phantom.exhaust*[*case-product phantom.exhaust*]) *simp*

end

28 Cardinality of types

theory *Cardinality*
imports *Phantom-Type*
begin

28.1 Preliminary lemmas

lemma (**in** *type-definition*) *univ*:
UNIV = *Abs* ' *A*
proof
show *Abs* ' *A* ⊆ *UNIV* **by** (*rule subset-UNIV*)
show *UNIV* ⊆ *Abs* ' *A*
proof
fix *x* :: *'b*
have *x* = *Abs* (*Rep x*) **by** (*rule Rep-inverse* [*symmetric*])
moreover have *Rep x* ∈ *A* **by** (*rule Rep*)
ultimately show *x* ∈ *Abs* ' *A* **by** (*rule image-eqI*)
qed
qed

lemma (**in** *type-definition*) *card*: *card* (*UNIV* :: *'b set*) = *card A*
by (*simp add*: *univ card-image inj-on-def Abs-inject*)

lemma *finite-range-Some*: *finite* (*range* (*Some* :: *'a* ⇒ *'a option*)) = *finite* (*UNIV* :: *'a set*)
by(*auto dest*: *finite-imageD intro*: *inj-Some*)

lemma *infinite-literal*: ¬ *finite* (*UNIV* :: *String.literal set*)
proof −
have *inj STR* **by**(*auto intro*: *injI*)
thus *?thesis*
by(*auto simp add*: *type-definition.univ*[*OF type-definition-literal*] *infinite-UNIV-listI dest*: *finite-imageD*)
qed

28.2 Cardinalities of types

syntax -*type-card* :: *type* => *nat* ((*1CARD*/(*1* '(-'))))

translations *CARD*('*t*) => *CONST card* (*CONST UNIV* :: '*t set*)

print-translation ‹
let
fun card-univ-tr' *ctxt* [*Const* (@{*const-syntax UNIV*}, *Type* (-, [*T*]))] =
Syntax.const @{*syntax-const -type-card*} $ *Syntax-Phases.term-of-typ ctxt T*
in [(@{*const-syntax card*}, *card-univ-tr*')] *end*
›

lemma *card-prod* [*simp*]: *CARD*('*a* × '*b*) = *CARD*('*a*) ∗ *CARD*('*b*)
unfolding *UNIV-Times-UNIV* [*symmetric*] **by** (*simp only*: *card-cartesian-product*)

lemma *card-UNIV-sum*: *CARD*('*a* + '*b*) = (*if CARD*('*a*) ≠ *0* ∧ *CARD*('*b*) ≠ *0 then CARD*('*a*) + *CARD*('*b*) *else 0*)
unfolding *UNIV-Plus-UNIV*[*symmetric*]
by(*auto simp add*: *card-eq-0-iff card-Plus simp del*: *UNIV-Plus-UNIV*)

lemma *card-sum* [*simp*]: *CARD*('*a* + '*b*) = *CARD*('*a*::*finite*) + *CARD*('*b*::*finite*)
by(*simp add*: *card-UNIV-sum*)

lemma *card-UNIV-option*: *CARD*('*a option*) = (*if CARD*('*a*) = *0 then 0 else CARD*('*a*) + *1*)
proof −
have (*None* :: '*a option*) ∉ *range Some* **by** *clarsimp*
thus *?thesis*
by (*simp add*: *UNIV-option-conv card-eq-0-iff finite-range-Some card-image*)
qed

lemma *card-option* [*simp*]: *CARD*('*a option*) = *Suc CARD*('*a*::*finite*)
by(*simp add*: *card-UNIV-option*)

lemma *card-UNIV-set*: *CARD*('*a set*) = (*if CARD*('*a*) = *0 then 0 else 2* ^ *CARD*('*a*))
by(*simp add*: *Pow-UNIV*[*symmetric*] *card-eq-0-iff card-Pow del*: *Pow-UNIV*)

lemma *card-set* [*simp*]: *CARD*('*a set*) = *2* ^ *CARD*('*a*::*finite*)
by(*simp add*: *card-UNIV-set*)

lemma *card-nat* [*simp*]: *CARD*(*nat*) = *0*
by (*simp add*: *card-eq-0-iff*)

lemma *card-fun*: *CARD*('*a* ⇒ '*b*) = (*if CARD*('*a*) ≠ *0* ∧ *CARD*('*b*) ≠ *0* ∨ *CARD*('*b*) = *1 then CARD*('*b*) ^ *CARD*('*a*) *else 0*)
proof −
{ assume *0* < *CARD*('*a*) **and** *0* < *CARD*('*b*)
hence *fina*: *finite* (*UNIV* :: '*a set*) **and** *finb*: *finite* (*UNIV* :: '*b set*)
by(*simp-all only*: *card-ge-0-finite*)
from *finite-distinct-list*[*OF finb*] **obtain** *bs*
where *bs*: *set bs* = (*UNIV* :: '*b set*) **and** *distb*: *distinct bs* **by** *blast*
from *finite-distinct-list*[*OF fina*] **obtain** *as*
where *as*: *set as* = (*UNIV* :: '*a set*) **and** *dista*: *distinct as* **by** *blast*
have *cb*: *CARD*('*b*) = *length bs*
unfolding *bs*[*symmetric*] *distinct-card*[*OF distb*] **..**
have *ca*: *CARD*('*a*) = *length as*
unfolding *as*[*symmetric*] *distinct-card*[*OF dista*] **..**
let *?xs* = *map* (λ*ys*. *the o map-of* (*zip as ys*)) (*List.n-lists* (*length as*) *bs*)
have *UNIV* = *set ?xs*
proof(*rule UNIV-eq-I*)
fix *f* :: '*a* ⇒ '*b*
from *as* **have** *f* = *the* ∘ *map-of* (*zip as* (*map f as*))
by(*auto simp add*: *map-of-zip-map*)
thus *f* ∈ *set ?xs* **using** *bs* **by**(*auto simp add*: *set-n-lists*)
qed
moreover have *distinct ?xs* **unfolding** *distinct-map*
proof(*intro conjI distinct-n-lists distb inj-onI*)
fix *xs ys* :: '*b list*
assume *xs*: *xs* ∈ *set* (*List.n-lists* (*length as*) *bs*)
and *ys*: *ys* ∈ *set* (*List.n-lists* (*length as*) *bs*)
and *eq*: *the* ∘ *map-of* (*zip as xs*) = *the* ∘ *map-of* (*zip as ys*)
from *xs ys* **have** [*simp*]: *length xs* = *length as length ys* = *length as*
by(*simp-all add*: *length-n-lists-elem*)
have *map-of* (*zip as xs*) = *map-of* (*zip as ys*)
proof
fix *x*
from *as bs* **have** ∃ *y*. *map-of* (*zip as xs*) *x* = *Some y* ∃ *y*. *map-of* (*zip as ys*) *x* = *Some y*
by(*simp-all add*: *map-of-zip-is-Some*[*symmetric*])
with *eq* **show** *map-of* (*zip as xs*) *x* = *map-of* (*zip as ys*) *x*
by(*auto dest*: *fun-cong*[**where** *x*=*x*])
qed
with *dista* **show** *xs* = *ys* **by**(*simp add*: *map-of-zip-inject*)
qed
hence *card* (*set ?xs*) = *length ?xs* **by**(*simp only*: *distinct-card*)
moreover have *length ?xs* = *length bs* ^ *length as* **by**(*simp add*: *length-n-lists*)
ultimately have *CARD*('*a* ⇒ '*b*) = *CARD*('*b*) ^ *CARD*('*a*) **using** *cb ca* **by** *simp* **}**
moreover {

assume *cb*: *CARD*('*b*) = *1*
then obtain *b* **where** *b*: *UNIV* = {*b* :: '*b*} **by**(*auto simp add*: *card-Suc-eq*)
have *eq*: *UNIV* = {λ*x* :: '*a*. *b* ::'*b*}
proof(*rule UNIV-eq-I*)
fix *x* :: '*a* ⇒ '*b*
{ fix *y*
have *x y* ∈ *UNIV* **..**
hence *x y* = *b* **unfolding** *b* **by** *simp* **}**
thus *x* ∈ {λ*x*. *b*} **by**(*auto*)
qed
have *CARD*('*a* ⇒ '*b*) = *1* **unfolding** *eq* **by** *simp* **}**
ultimately show *?thesis*
by(*auto simp del*: *One-nat-def*)(*auto simp add*: *card-eq-0-iff dest*: *finite-fun-UNIVD2 finite-fun-UNIVD1*)
qed

corollary *finite-UNIV-fun*:
finite (*UNIV* :: ('*a* ⇒ '*b*) *set*) ⟷
finite (*UNIV* :: '*a set*) ∧ *finite* (*UNIV* :: '*b set*) ∨ *CARD*('*b*) = *1*
(**is** *?lhs* ⟷ *?rhs*)
proof −
have *?lhs* ⟷ *CARD*('*a* ⇒ '*b*) > *0* **by**(*simp add*: *card-gt-0-iff*)
also have ... ⟷ *CARD*('*a*) > *0* ∧ *CARD*('*b*) > *0* ∨ *CARD*('*b*) = *1*
by(*simp add*: *card-fun*)
also have ... = *?rhs* **by**(*simp add*: *card-gt-0-iff*)
finally show *?thesis* **.**
qed

lemma *card-nibble*: *CARD*(*nibble*) = *16*
unfolding *UNIV-nibble* **by** *simp*

lemma *card-UNIV-char*: *CARD*(*char*) = *256*
proof −
have *inj* (λ(*x*, *y*). *Char x y*) **by**(*auto intro*: *injI*)
thus *?thesis* **unfolding** *UNIV-char* **by**(*simp add*: *card-image card-nibble*)
qed

lemma *card-literal*: *CARD*(*String.literal*) = *0*
by(*simp add*: *card-eq-0-iff infinite-literal*)

28.3 Classes with at least 1 and 2

Class finite already captures "at least 1"

lemma *zero-less-card-finite* [*simp*]: *0* < *CARD*('*a*::*finite*)
unfolding *neq0-conv* [*symmetric*] **by** *simp*

lemma *one-le-card-finite* [*simp*]: *Suc 0* ≤ *CARD*('*a*::*finite*)
by (*simp add*: *less-Suc-eq-le* [*symmetric*])

Class for cardinality "at least 2"

class *card2* = *finite* +
  assumes *two-le-card*: *2* ≤ *CARD*('*a*)

lemma *one-less-card*: *Suc 0* < *CARD*('*a*::*card2*)
  using *two-le-card* [**where** '*a*='*a*] **by** *simp*

lemma *one-less-int-card*: *1* < *int CARD*('*a*::*card2*)
  using *one-less-card* [**where** '*a*='*a*] **by** *simp*

28.4 A type class for deciding finiteness of types

type-synonym '*a finite-UNIV* = ('*a*, *bool*) *phantom*

class *finite-UNIV* =
  fixes *finite-UNIV* :: ('*a*, *bool*) *phantom*
  assumes *finite-UNIV*: *finite-UNIV* = *Phantom*('*a*) (*finite* (*UNIV* :: '*a set*))

lemma *finite-UNIV-code* [*code-unfold*]:
  finite (*UNIV* :: '*a* :: *finite-UNIV set*)
  ⟷ *of-phantom* (*finite-UNIV* :: '*a finite-UNIV*)
by(*simp add*: *finite-UNIV*)

28.5 A type class for computing the cardinality of types

definition *is-list-UNIV* :: '*a list* ⇒ *bool*
where *is-list-UNIV xs* = (*let c* = *CARD*('*a*) *in if c* = *0 then False else size* (*remdups xs*) = *c*)

lemma *is-list-UNIV-iff*: *is-list-UNIV xs* ⟷ *set xs* = *UNIV*
by(*auto simp add*: *is-list-UNIV-def Let-def card-eq-0-iff List.card-set*[*symmetric*]
   dest: *subst*[**where** *P*=*finite*, *OF - finite-set*] *card-eq-UNIV-imp-eq-UNIV*)

type-synonym '*a card-UNIV* = ('*a*, *nat*) *phantom*

class *card-UNIV* = *finite-UNIV* +
  fixes *card-UNIV* :: '*a card-UNIV*
  assumes *card-UNIV*: *card-UNIV* = *Phantom*('*a*) *CARD*('*a*)

28.6 Instantiations for *card-UNIV*

instantiation *nat* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*nat*) *False*
definition *card-UNIV* = *Phantom*(*nat*) *0*
instance by *intro-classes* (*simp-all add*: *finite-UNIV-nat-def card-UNIV-nat-def*)
end

instantiation *int* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*int*) *False*

definition *card-UNIV* = *Phantom*(*int*) *0*
instance by *intro-classes* (*simp-all add*: *card-UNIV-int-def finite-UNIV-int-def infinite-UNIV-int*)
end

instantiation *natural* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*natural*) *False*
definition *card-UNIV* = *Phantom*(*natural*) *0*
instance
by *standard*
(*auto simp add*: *finite-UNIV-natural-def card-UNIV-natural-def card-eq-0-iff*
type-definition.univ [*OF type-definition-natural*] *natural-eq-iff*
dest!: *finite-imageD intro*: *inj-onI*)
end

instantiation *integer* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*integer*) *False*
definition *card-UNIV* = *Phantom*(*integer*) *0*
instance
by *standard*
(*auto simp add*: *finite-UNIV-integer-def card-UNIV-integer-def card-eq-0-iff*
type-definition.univ [*OF type-definition-integer*] *infinite-UNIV-int*
dest!: *finite-imageD intro*: *inj-onI*)
end

instantiation *list* :: (*type*) *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*('*a list*) *False*
definition *card-UNIV* = *Phantom*('*a list*) *0*
instance by *intro-classes* (*simp-all add*: *card-UNIV-list-def finite-UNIV-list-def infinite-UNIV-listI*)
end

instantiation *unit* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*unit*) *True*
definition *card-UNIV* = *Phantom*(*unit*) *1*
instance by *intro-classes* (*simp-all add*: *card-UNIV-unit-def finite-UNIV-unit-def*)
end

instantiation *bool* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*bool*) *True*
definition *card-UNIV* = *Phantom*(*bool*) *2*
instance by(*intro-classes*)(*simp-all add*: *card-UNIV-bool-def finite-UNIV-bool-def*)
end

instantiation *nibble* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*nibble*) *True*
definition *card-UNIV* = *Phantom*(*nibble*) *16*
instance by(*intro-classes*)(*simp-all add*: *card-UNIV-nibble-def card-nibble finite-UNIV-nibble-def*)
end

instantiation *char* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*char*) *True*
definition *card-UNIV* = *Phantom*(*char*) *256*
instance by *intro-classes* (*simp-all add*: *card-UNIV-char-def card-UNIV-char finite-UNIV-char-def*)
end

instantiation *prod* :: (*finite-UNIV*, *finite-UNIV*) *finite-UNIV* **begin**
definition *finite-UNIV* = *Phantom*('*a* × '*b*)
(*of-phantom* (*finite-UNIV* :: '*a finite-UNIV*) ∧ *of-phantom* (*finite-UNIV* :: '*b finite-UNIV*))
instance by *intro-classes* (*simp add*: *finite-UNIV-prod-def finite-UNIV finite-prod*)
end

instantiation *prod* :: (*card-UNIV*, *card-UNIV*) *card-UNIV* **begin**
definition *card-UNIV* = *Phantom*('*a* × '*b*)
(*of-phantom* (*card-UNIV* :: '*a card-UNIV*) ∗ *of-phantom* (*card-UNIV* :: '*b card-UNIV*))
instance by *intro-classes* (*simp add*: *card-UNIV-prod-def card-UNIV*)
end

instantiation *sum* :: (*finite-UNIV*, *finite-UNIV*) *finite-UNIV* **begin**
definition *finite-UNIV* = *Phantom*('*a* + '*b*)
(*of-phantom* (*finite-UNIV* :: '*a finite-UNIV*) ∧ *of-phantom* (*finite-UNIV* :: '*b finite-UNIV*))
instance
by *intro-classes* (*simp add*: *UNIV-Plus-UNIV*[*symmetric*] *finite-UNIV-sum-def finite-UNIV del*: *UNIV-Plus-UNIV*)
end

instantiation *sum* :: (*card-UNIV*, *card-UNIV*) *card-UNIV* **begin**
definition *card-UNIV* = *Phantom*('*a* + '*b*)
(*let ca* = *of-phantom* (*card-UNIV* :: '*a card-UNIV*);
cb = *of-phantom* (*card-UNIV* :: '*b card-UNIV*)
in if ca ≠ *0* ∧ *cb* ≠ *0 then ca* + *cb else 0*)
instance by *intro-classes* (*auto simp add*: *card-UNIV-sum-def card-UNIV card-UNIV-sum*)
end

instantiation *fun* :: (*finite-UNIV*, *card-UNIV*) *finite-UNIV* **begin**
definition *finite-UNIV* = *Phantom*('*a* ⇒ '*b*)
(*let cb* = *of-phantom* (*card-UNIV* :: '*b card-UNIV*)
in cb = *1* ∨ *of-phantom* (*finite-UNIV* :: '*a finite-UNIV*) ∧ *cb* ≠ *0*)
instance
by *intro-classes* (*auto simp add*: *finite-UNIV-fun-def Let-def card-UNIV finite-UNIV finite-UNIV-fun card-gt-0-iff*)
end

instantiation *fun* :: (*card-UNIV*, *card-UNIV*) *card-UNIV* **begin**
definition *card-UNIV* = *Phantom*('*a* ⇒ '*b*)
(*let ca* = *of-phantom* (*card-UNIV* :: '*a card-UNIV*);

```
      cb = of-phantom (card-UNIV :: 'b card-UNIV)
   in if ca ≠ 0 ∧ cb ≠ 0 ∨ cb = 1 then cb ^ ca else 0)
instance by intro-classes (simp add: card-UNIV-fun-def card-UNIV Let-def card-fun)
end

instantiation option :: (finite-UNIV) finite-UNIV begin
definition finite-UNIV = Phantom('a option) (of-phantom (finite-UNIV :: 'a
finite-UNIV))
instance by intro-classes (simp add: finite-UNIV-option-def finite-UNIV)
end

instantiation option :: (card-UNIV) card-UNIV begin
definition card-UNIV = Phantom('a option)
  (let c = of-phantom (card-UNIV :: 'a card-UNIV) in if c ≠ 0 then Suc c else 0)
instance by intro-classes (simp add: card-UNIV-option-def card-UNIV card-UNIV-option)
end

instantiation String.literal :: card-UNIV begin
definition finite-UNIV = Phantom(String.literal) False
definition card-UNIV = Phantom(String.literal) 0
instance
 by intro-classes (simp-all add: card-UNIV-literal-def finite-UNIV-literal-def infinite-literal
card-literal)
end

instantiation set :: (finite-UNIV) finite-UNIV begin
definition finite-UNIV = Phantom('a set) (of-phantom (finite-UNIV :: 'a finite-UNIV))
instance by intro-classes (simp add: finite-UNIV-set-def finite-UNIV Finite-Set.finite-set)
end

instantiation set :: (card-UNIV) card-UNIV begin
definition card-UNIV = Phantom('a set)
  (let c = of-phantom (card-UNIV :: 'a card-UNIV) in if c = 0 then 0 else 2 ^ c)
instance by intro-classes (simp add: card-UNIV-set-def card-UNIV-set card-UNIV)
end

lemma UNIV-finite-1: UNIV = set [finite-1.a1]
by(auto intro: finite-1.exhaust)

lemma UNIV-finite-2: UNIV = set [finite-2.a1, finite-2.a2]
by(auto intro: finite-2.exhaust)

lemma UNIV-finite-3: UNIV = set [finite-3.a1, finite-3.a2, finite-3.a3]
by(auto intro: finite-3.exhaust)

lemma UNIV-finite-4: UNIV = set [finite-4.a1, finite-4.a2, finite-4.a3, finite-4.a4]
by(auto intro: finite-4.exhaust)

lemma UNIV-finite-5:
```

UNIV = *set* [*finite-5*.a_1, *finite-5*.a_2, *finite-5*.a_3, *finite-5*.a_4, *finite-5*.a_5]
by(*auto intro*: *finite-5.exhaust*)

instantiation *Enum.finite-1* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*Enum.finite-1*) *True*
definition *card-UNIV* = *Phantom*(*Enum.finite-1*) *1*
instance
by *intro-classes* (*simp-all add*: *UNIV-finite-1 card-UNIV-finite-1-def finite-UNIV-finite-1-def*)
end

instantiation *Enum.finite-2* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*Enum.finite-2*) *True*
definition *card-UNIV* = *Phantom*(*Enum.finite-2*) *2*
instance
by *intro-classes* (*simp-all add*: *UNIV-finite-2 card-UNIV-finite-2-def finite-UNIV-finite-2-def*)
end

instantiation *Enum.finite-3* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*Enum.finite-3*) *True*
definition *card-UNIV* = *Phantom*(*Enum.finite-3*) *3*
instance
by *intro-classes* (*simp-all add*: *UNIV-finite-3 card-UNIV-finite-3-def finite-UNIV-finite-3-def*)
end

instantiation *Enum.finite-4* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*Enum.finite-4*) *True*
definition *card-UNIV* = *Phantom*(*Enum.finite-4*) *4*
instance
by *intro-classes* (*simp-all add*: *UNIV-finite-4 card-UNIV-finite-4-def finite-UNIV-finite-4-def*)
end

instantiation *Enum.finite-5* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*Enum.finite-5*) *True*
definition *card-UNIV* = *Phantom*(*Enum.finite-5*) *5*
instance
by *intro-classes* (*simp-all add*: *UNIV-finite-5 card-UNIV-finite-5-def finite-UNIV-finite-5-def*)
end

28.7 Code setup for sets

Implement *CARD*('*a*) via *card-UNIV-class.card-UNIV* and provide implementations for *finite*, *card*, *op* $\subseteq$, and *op* =if the calling context already provides *finite-UNIV* and *card-UNIV* instances. If we implemented the latter always via *card-UNIV-class.card-UNIV*, we would require instances of essentially all element types, i.e., a lot of instantiation proofs and – at run time – possibly slow dictionary constructions.

context
begin

qualified definition *card-UNIV'* :: *'a card-UNIV*
where [*code del*]: *card-UNIV' = Phantom('a) CARD('a)*

lemma *CARD-code* [*code-unfold*]:
CARD('a) = of-phantom (card-UNIV' :: 'a card-UNIV)
by(*simp add*: *card-UNIV'-def*)

lemma *card-UNIV'-code* [*code*]:
card-UNIV' = card-UNIV
by(*simp add*: *card-UNIV card-UNIV'-def*)

end

lemma *card-Compl*:
finite A ⟹ card (− A) = card (UNIV :: 'a set) − card (A :: 'a set)
by (*metis Compl-eq-Diff-UNIV card-Diff-subset top-greatest*)

context fixes *xs* :: *'a* :: *finite-UNIV list*
begin

qualified definition *finite'* :: *'a set ⇒ bool*
where [*simp*, *code del*, *code-abbrev*]: *finite' = finite*

lemma *finite'-code* [*code*]:
finite' (set xs) ⟷ True
finite' (List.coset xs) ⟷ of-phantom (finite-UNIV :: 'a finite-UNIV)
by(*simp-all add*: *card-gt-0-iff finite-UNIV*)

end

context fixes *xs* :: *'a* :: *card-UNIV list*
begin

qualified definition *card'* :: *'a set ⇒ nat*
where [*simp*, *code del*, *code-abbrev*]: *card' = card*

lemma *card'-code* [*code*]:
card' (set xs) = length (remdups xs)
card' (List.coset xs) = of-phantom (card-UNIV :: 'a card-UNIV) − length (remdups xs)
by(*simp-all add*: *List.card-set card-Compl card-UNIV*)

qualified definition *subset'* :: *'a set ⇒ 'a set ⇒ bool*
where [*simp*, *code del*, *code-abbrev*]: *subset' = op ⊆*

lemma *subset'-code* [*code*]:
subset' A (List.coset ys) ⟷ (∀ y ∈ set ys. y ∉ A)

subset' (*set ys*) *B* ⟷ (∀ *y* ∈ *set ys*. *y* ∈ *B*)
subset' (*List.coset xs*) (*set ys*) ⟷ (*let n* = *CARD*('*a*) *in n* > *0* ∧ *card*(*set* (*xs* @ *ys*)) = *n*)
by(*auto simp add*: *Let-def card-gt-0-iff dest*: *card-eq-UNIV-imp-eq-UNIV intro*: *arg-cong*[**where** *f*=*card*])
(*metis finite-compl finite-set rev-finite-subset*)

qualified definition *eq-set* :: '*a set* ⇒ '*a set* ⇒ *bool*
where [*simp*, *code del*, *code-abbrev*]: *eq-set* = *op* =

lemma *eq-set-code* [*code*]:
fixes *ys*
defines *rhs* ≡
let n = *CARD*('*a*)
in if n = *0 then False else*
let xs' = *remdups xs*; *ys'* = *remdups ys*
in length xs' + *length ys'* = *n* ∧ (∀ *x* ∈ *set xs'*. *x* ∉ *set ys'*) ∧ (∀ *y* ∈ *set ys'*. *y* ∉ *set xs'*)
shows *eq-set* (*List.coset xs*) (*set ys*) ⟷ *rhs*
and *eq-set* (*set ys*) (*List.coset xs*) ⟷ *rhs*
and *eq-set* (*set xs*) (*set ys*) ⟷ (∀ *x* ∈ *set xs*. *x* ∈ *set ys*) ∧ (∀ *y* ∈ *set ys*. *y* ∈ *set xs*)
and *eq-set* (*List.coset xs*) (*List.coset ys*) ⟷ (∀ *x* ∈ *set xs*. *x* ∈ *set ys*) ∧ (∀ *y* ∈ *set ys*. *y* ∈ *set xs*)
proof *goal-cases*
{
case *1*
show *?case* (**is** *?lhs* ⟷ *?rhs*)
proof
show *?rhs* **if** *?lhs*
using *that*
by (*auto simp add*: *rhs-def Let-def List.card-set*[*symmetric*]
card-Un-Int[**where** *A*=*set xs* **and** *B*=− *set xs*] *card-UNIV*
Compl-partition card-gt-0-iff dest: *sym*)(*metis finite-compl finite-set*)
show *?lhs* **if** *?rhs*
proof −
have ⟦ ∀ *y*∈*set xs*. *y* ∉ *set ys*; ∀ *x*∈*set ys*. *x* ∉ *set xs* ⟧ ⟹ *set xs* ∩ *set ys* = {} **by** *blast*
with *that* **show** *?thesis*
by (*auto simp add*: *rhs-def Let-def List.card-set*[*symmetric*]
card-UNIV card-gt-0-iff card-Un-Int[**where** *A*=*set xs* **and** *B*=*set ys*]
dest: *card-eq-UNIV-imp-eq-UNIV split*: *split-if-asm*)
qed
qed
}
moreover
case *2*
ultimately show *?case* **unfolding** *eq-set-def* **by** *blast*
next

```
  case 3
  show ?case unfolding eq-set-def List.coset-def by blast
next
  case 4
  show ?case unfolding eq-set-def List.coset-def by blast
qed

end
```

Provide more informative exceptions than Match for non-rewritten cases. If generated code raises one these exceptions, then a code equation calls the mentioned operator for an element type that is not an instance of *card-UNIV* and is therefore not implemented via *card-UNIV-class.card-UNIV*. Constrain the element type with sort *card-UNIV* to change this.

```
lemma card-coset-error [code]:
  card (List.coset xs) =
   Code.abort (STR ''card (List.coset -) requires type class instance card-UNIV'')
     (λ-. card (List.coset xs))
by(simp)

lemma coset-subseteq-set-code [code]:
  List.coset xs ⊆ set ys ⟷
  (if xs = [] ∧ ys = [] then False
   else Code.abort
    (STR ''subset-eq (List.coset -) (List.set -) requires type class instance card-UNIV'')
     (λ-. List.coset xs ⊆ set ys))
by simp

notepad begin — test code setup
have List.coset [True] = set [False] ∧
      List.coset [] ⊆ List.set [True, False] ∧
      finite (List.coset [True])
  by eval
end

end
```

29 Numeral Syntax for Types

```
theory Numeral-Type
imports Cardinality
begin
```

29.1 Numeral Types

```
typedef num0 = UNIV :: nat set ..
typedef num1 = UNIV :: unit set ..
```

typedef *'a bit0 = {0 ..< 2 * int CARD('a::finite)}*
proof
  show *0 ∈ {0 ..< 2 * int CARD('a)}*
    by *simp*
qed

typedef *'a bit1 = {0 ..< 1 + 2 * int CARD('a::finite)}*
proof
  show *0 ∈ {0 ..< 1 + 2 * int CARD('a)}*
    by *simp*
qed

lemma *card-num0 [simp]: CARD (num0) = 0*
  unfolding *type-definition.card [OF type-definition-num0]*
  by *simp*

lemma *infinite-num0: ¬ finite (UNIV :: num0 set)*
  using *card-num0[unfolded card-eq-0-iff]*
  by *simp*

lemma *card-num1 [simp]: CARD(num1) = 1*
  unfolding *type-definition.card [OF type-definition-num1]*
  by *(simp only: card-UNIV-unit)*

lemma *card-bit0 [simp]: CARD('a bit0) = 2 * CARD('a::finite)*
  unfolding *type-definition.card [OF type-definition-bit0]*
  by *simp*

lemma *card-bit1 [simp]: CARD('a bit1) = Suc (2 * CARD('a::finite))*
  unfolding *type-definition.card [OF type-definition-bit1]*
  by *simp*

instance *num1 :: finite*
proof
  show *finite (UNIV::num1 set)*
    unfolding *type-definition.univ [OF type-definition-num1]*
    using *finite* **by** *(rule finite-imageI)*
qed

instance *bit0 :: (finite) card2*
proof
  show *finite (UNIV::'a bit0 set)*
    unfolding *type-definition.univ [OF type-definition-bit0]*
    by *simp*
  show *2 ≤ CARD('a bit0)*
    by *simp*
qed

instance *bit1 :: (finite) card2*

proof
  show *finite* (*UNIV*::*'a bit1 set*)
    unfolding *type-definition.univ* [*OF type-definition-bit1*]
    by *simp*
  show *2* ≤ *CARD*(*'a bit1*)
    by *simp*
qed

29.2 Locales for for modular arithmetic subtypes

locale *mod-type* =
  fixes *n* :: *int*
  and *Rep* :: *'a*::{*zero*,*one*,*plus*,*times*,*uminus*,*minus*} ⇒ *int*
  and *Abs* :: *int* ⇒ *'a*::{*zero*,*one*,*plus*,*times*,*uminus*,*minus*}
  assumes *type*: *type-definition Rep Abs* {*0*..<*n*}
  and *size1*: *1* < *n*
  and *zero-def*: *0* = *Abs 0*
  and *one-def*: *1* = *Abs 1*
  and *add-def*: *x* + *y* = *Abs* ((*Rep x* + *Rep y*) *mod n*)
  and *mult-def*: *x* ∗ *y* = *Abs* ((*Rep x* ∗ *Rep y*) *mod n*)
  and *diff-def*: *x* − *y* = *Abs* ((*Rep x* − *Rep y*) *mod n*)
  and *minus-def*: − *x* = *Abs* ((− *Rep x*) *mod n*)
begin

lemma *size0*: *0* < *n*
using *size1* **by** *simp*

lemmas *definitions* =
  zero-def one-def add-def mult-def minus-def diff-def

lemma *Rep-less-n*: *Rep x* < *n*
by (*rule type-definition.Rep* [*OF type*, *simplified*, *THEN conjunct2*])

lemma *Rep-le-n*: *Rep x* ≤ *n*
by (*rule Rep-less-n* [*THEN order-less-imp-le*])

lemma *Rep-inject-sym*: *x* = *y* ⟷ *Rep x* = *Rep y*
by (*rule type-definition.Rep-inject* [*OF type*, *symmetric*])

lemma *Rep-inverse*: *Abs* (*Rep x*) = *x*
by (*rule type-definition.Rep-inverse* [*OF type*])

lemma *Abs-inverse*: *m* ∈ {*0*..<*n*} ⟹ *Rep* (*Abs m*) = *m*
by (*rule type-definition.Abs-inverse* [*OF type*])

lemma *Rep-Abs-mod*: *Rep* (*Abs* (*m mod n*)) = *m mod n*
by (*simp add*: *Abs-inverse pos-mod-conj* [*OF size0*])

lemma *Rep-Abs-0*: *Rep* (*Abs 0*) = *0*

by (*simp add*: *Abs-inverse size0*)

lemma *Rep-0*: *Rep 0 = 0*
by (*simp add*: *zero-def Rep-Abs-0*)

lemma *Rep-Abs-1*: *Rep* (*Abs 1*) = *1*
by (*simp add*: *Abs-inverse size1*)

lemma *Rep-1*: *Rep 1 = 1*
by (*simp add*: *one-def Rep-Abs-1*)

lemma *Rep-mod*: *Rep x mod n = Rep x*
apply (*rule-tac x=x* **in** *type-definition.Abs-cases* [*OF type*])
apply (*simp add*: *type-definition.Abs-inverse* [*OF type*])
apply (*simp add*: *mod-pos-pos-trivial*)
done

lemmas *Rep-simps* =
Rep-inject-sym Rep-inverse Rep-Abs-mod Rep-mod Rep-Abs-0 Rep-Abs-1

lemma *comm-ring-1*: *OFCLASS*('*a*, *comm-ring-1-class*)
apply (*intro-classes*, *unfold definitions*)
apply (*simp-all add*: *Rep-simps zmod-simps field-simps*)
done

end

locale *mod-ring* = *mod-type n Rep Abs*
for *n* :: *int*
and *Rep* :: '*a*::{*comm-ring-1*} ⇒ *int*
and *Abs* :: *int* ⇒ '*a*::{*comm-ring-1*}
begin

lemma *of-nat-eq*: *of-nat k = Abs* (*int k mod n*)
apply (*induct k*)
apply (*simp add*: *zero-def*)
apply (*simp add*: *Rep-simps add-def one-def zmod-simps ac-simps*)
done

lemma *of-int-eq*: *of-int z = Abs* (*z mod n*)
apply (*cases z rule*: *int-diff-cases*)
apply (*simp add*: *Rep-simps of-nat-eq diff-def zmod-simps*)
done

lemma *Rep-numeral*:
Rep (*numeral w*) = *numeral w mod n*
using *of-int-eq* [*of numeral w*]
by (*simp add*: *Rep-inject-sym Rep-Abs-mod*)

lemma *iszero-numeral*:
iszero (*numeral w*::'*a*) ⟷ *numeral w mod n* = *0*
by (*simp add*: *Rep-inject-sym Rep-numeral Rep-0 iszero-def*)

lemma *cases*:
assumes *1*: ⋀*z*. ⟦(*x*::'*a*) = *of-int z*; *0* ≤ *z*; *z* < *n*⟧ ⟹ *P*
shows *P*
apply (*cases x rule*: *type-definition.Abs-cases* [*OF type*])
apply (*rule-tac z*=*y* **in** *1*)
apply (*simp-all add*: *of-int-eq mod-pos-pos-trivial*)
done

lemma *induct*:
(⋀*z*. ⟦*0* ≤ *z*; *z* < *n*⟧ ⟹ *P* (*of-int z*)) ⟹ *P* (*x*::'*a*)
by (*cases x rule*: *cases*) *simp*

end

29.3 Ring class instances

Unfortunately *ring-1* instance is not possible for *num1*, since 0 and 1 are not distinct.

instantiation *num1* :: {*comm-ring*,*comm-monoid-mult*,*numeral*}
begin

lemma *num1-eq-iff*: (*x*::*num1*) = (*y*::*num1*) ⟷ *True*
by (*induct x*, *induct y*) *simp*

instance
by *standard* (*simp-all add*: *num1-eq-iff*)

end

instantiation
bit0 **and** *bit1* :: (*finite*) {*zero*,*one*,*plus*,*times*,*uminus*,*minus*}
begin

definition *Abs-bit0'* :: *int* ⇒ '*a bit0* **where**
Abs-bit0' x = *Abs-bit0* (*x mod int CARD*('*a bit0*))

definition *Abs-bit1'* :: *int* ⇒ '*a bit1* **where**
Abs-bit1' x = *Abs-bit1* (*x mod int CARD*('*a bit1*))

definition *0* = *Abs-bit0 0*
definition *1* = *Abs-bit0 1*
definition *x* + *y* = *Abs-bit0'* (*Rep-bit0 x* + *Rep-bit0 y*)
definition *x* ∗ *y* = *Abs-bit0'* (*Rep-bit0 x* ∗ *Rep-bit0 y*)
definition *x* − *y* = *Abs-bit0'* (*Rep-bit0 x* − *Rep-bit0 y*)
definition − *x* = *Abs-bit0'* (− *Rep-bit0 x*)

definition *0 = Abs-bit1 0*
definition *1 = Abs-bit1 1*
definition *x + y = Abs-bit1' (Rep-bit1 x + Rep-bit1 y)*
definition *x * y = Abs-bit1' (Rep-bit1 x * Rep-bit1 y)*
definition *x - y = Abs-bit1' (Rep-bit1 x - Rep-bit1 y)*
definition *- x = Abs-bit1' (- Rep-bit1 x)*

instance ..

end

interpretation *bit0*:
mod-type int CARD('a::finite bit0)
Rep-bit0 :: 'a::finite bit0 ⇒ int
Abs-bit0 :: int ⇒ 'a::finite bit0
apply (*rule mod-type.intro*)
apply (*simp add*: *int-mult type-definition-bit0*)
apply (*rule one-less-int-card*)
apply (*rule zero-bit0-def*)
apply (*rule one-bit0-def*)
apply (*rule plus-bit0-def* [*unfolded Abs-bit0'-def*])
apply (*rule times-bit0-def* [*unfolded Abs-bit0'-def*])
apply (*rule minus-bit0-def* [*unfolded Abs-bit0'-def*])
apply (*rule uminus-bit0-def* [*unfolded Abs-bit0'-def*])
done

interpretation *bit1*:
mod-type int CARD('a::finite bit1)
Rep-bit1 :: 'a::finite bit1 ⇒ int
Abs-bit1 :: int ⇒ 'a::finite bit1
apply (*rule mod-type.intro*)
apply (*simp add*: *int-mult type-definition-bit1*)
apply (*rule one-less-int-card*)
apply (*rule zero-bit1-def*)
apply (*rule one-bit1-def*)
apply (*rule plus-bit1-def* [*unfolded Abs-bit1'-def*])
apply (*rule times-bit1-def* [*unfolded Abs-bit1'-def*])
apply (*rule minus-bit1-def* [*unfolded Abs-bit1'-def*])
apply (*rule uminus-bit1-def* [*unfolded Abs-bit1'-def*])
done

instance *bit0* :: (*finite*) *comm-ring-1*
by (*rule bit0.comm-ring-1*)

instance *bit1* :: (*finite*) *comm-ring-1*
by (*rule bit1.comm-ring-1*)

interpretation *bit0*:

mod-ring int CARD('a::finite bit0)
Rep-bit0 :: 'a::finite bit0 ⇒ int
Abs-bit0 :: int ⇒ 'a::finite bit0
..

interpretation *bit1*:
mod-ring int CARD('a::finite bit1)
Rep-bit1 :: 'a::finite bit1 ⇒ int
Abs-bit1 :: int ⇒ 'a::finite bit1
..

Set up cases, induction, and arithmetic

lemmas *bit0-cases* [*case-names of-int*, *cases type*: *bit0*] = *bit0.cases*
lemmas *bit1-cases* [*case-names of-int*, *cases type*: *bit1*] = *bit1.cases*

lemmas *bit0-induct* [*case-names of-int*, *induct type*: *bit0*] = *bit0.induct*
lemmas *bit1-induct* [*case-names of-int*, *induct type*: *bit1*] = *bit1.induct*

lemmas *bit0-iszero-numeral* [*simp*] = *bit0.iszero-numeral*
lemmas *bit1-iszero-numeral* [*simp*] = *bit1.iszero-numeral*

lemmas [*simp*] = *eq-numeral-iff-iszero* [**where** *'a='a bit0*] **for** *dummy* :: *'a::finite*
lemmas [*simp*] = *eq-numeral-iff-iszero* [**where** *'a='a bit1*] **for** *dummy* :: *'a::finite*

29.4 Order instances

instantiation *bit0* **and** *bit1* :: (*finite*) *linorder* **begin**
definition $a < b \longleftrightarrow$ *Rep-bit0 a* $<$ *Rep-bit0 b*
definition $a \leq b \longleftrightarrow$ *Rep-bit0 a* $\leq$ *Rep-bit0 b*
definition $a < b \longleftrightarrow$ *Rep-bit1 a* $<$ *Rep-bit1 b*
definition $a \leq b \longleftrightarrow$ *Rep-bit1 a* $\leq$ *Rep-bit1 b*

instance
by(*intro-classes*)
(*auto simp add*: *less-eq-bit0-def less-bit0-def less-eq-bit1-def less-bit1-def Rep-bit0-inject Rep-bit1-inject*)
end

lemma (**in** *preorder*) *tranclp-less*: *op* $<^{++}$ = *op* $<$
by(*auto simp add*: *fun-eq-iff intro*: *less-trans elim*: *tranclp.induct*)

instance *bit0* **and** *bit1* :: (*finite*) *wellorder*
proof −
have *wf* {(*x* :: *'a bit0*, *y*). *x* < *y*}
by(*auto simp add*: *trancl-def tranclp-less intro*!: *finite-acyclic-wf acyclicI*)
thus *OFCLASS*('*a bit0*, *wellorder-class*)
by(*rule wf-wellorderI*) *intro-classes*
next
have *wf* {(*x* :: *'a bit1*, *y*). *x* < *y*}

by(*auto simp add*: *trancl-def tranclp-less intro*!: *finite-acyclic-wf acyclicI*)
thus *OFCLASS*('*a bit1*, *wellorder-class*)
by(*rule wf-wellorderI*) *intro-classes*
qed

29.5 Code setup and type classes for code generation

Code setup for *num0* and *num1*

definition *Num0* :: *num0* **where** *Num0* = *Abs-num0 0*
code-datatype *Num0*

instantiation *num0* :: *equal* **begin**
definition *equal-num0* :: *num0* ⇒ *num0* ⇒ *bool*
where *equal-num0* = *op* =
instance by *intro-classes* (*simp add*: *equal-num0-def*)
end

lemma *equal-num0-code* [*code*]:
equal-class.equal Num0 Num0 = *True*
by(*rule equal-refl*)

code-datatype *1* :: *num1*

instantiation *num1* :: *equal* **begin**
definition *equal-num1* :: *num1* ⇒ *num1* ⇒ *bool*
where *equal-num1* = *op* =
instance by *intro-classes* (*simp add*: *equal-num1-def*)
end

lemma *equal-num1-code* [*code*]:
equal-class.equal (*1* :: *num1*) *1* = *True*
by(*rule equal-refl*)

instantiation *num1* :: *enum* **begin**
definition *enum-class.enum* = [*1* :: *num1*]
definition *enum-class.enum-all P* = *P* (*1* :: *num1*)
definition *enum-class.enum-ex P* = *P* (*1* :: *num1*)
instance
by *intro-classes*
(*auto simp add*: *enum-num1-def enum-all-num1-def enum-ex-num1-def num1-eq-iff Ball-def*,
(*metis* (*full-types*) *num1-eq-iff*)+)
end

instantiation *num0* **and** *num1* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*num0*) *False*
definition *card-UNIV* = *Phantom*(*num0*) *0*
definition *finite-UNIV* = *Phantom*(*num1*) *True*
definition *card-UNIV* = *Phantom*(*num1*) *1*

instance
by *intro-classes*
(*simp-all add*: *finite-UNIV-num0-def card-UNIV-num0-def infinite-num0 finite-UNIV-num1-def card-UNIV-num1-def*)
end

Code setup for *'a bit0* and *'a bit1*

declare
bit0.Rep-inverse[*code abstype*]
bit0.Rep-0[*code abstract*]
bit0.Rep-1[*code abstract*]

lemma *Abs-bit0'-code* [*code abstract*]:
Rep-bit0 (*Abs-bit0'* *x* :: *'a* :: *finite bit0*) = *x mod int* (*CARD*(*'a bit0*))
by(*auto simp add*: *Abs-bit0'-def intro*!: *Abs-bit0-inverse*)

lemma *inj-on-Abs-bit0*:
inj-on (*Abs-bit0* :: *int* ⇒ *'a bit0*) {*0..<2* ∗ *int CARD*(*'a* :: *finite*)}
by(*auto intro*: *inj-onI simp add*: *Abs-bit0-inject*)

declare
bit1.Rep-inverse[*code abstype*]
bit1.Rep-0[*code abstract*]
bit1.Rep-1[*code abstract*]

lemma *Abs-bit1'-code* [*code abstract*]:
Rep-bit1 (*Abs-bit1'* *x* :: *'a* :: *finite bit1*) = *x mod int* (*CARD*(*'a bit1*))
by(*auto simp add*: *Abs-bit1'-def intro*!: *Abs-bit1-inverse*)

lemma *inj-on-Abs-bit1*:
inj-on (*Abs-bit1* :: *int* ⇒ *'a bit1*) {*0..<1* + *2* ∗ *int CARD*(*'a* :: *finite*)}
by(*auto intro*: *inj-onI simp add*: *Abs-bit1-inject*)

instantiation *bit0* **and** *bit1* :: (*finite*) *equal* **begin**

definition *equal-class.equal x y* ⟷ *Rep-bit0 x* = *Rep-bit0 y*
definition *equal-class.equal x y* ⟷ *Rep-bit1 x* = *Rep-bit1 y*

instance
by *intro-classes* (*simp-all add*: *equal-bit0-def equal-bit1-def Rep-bit0-inject Rep-bit1-inject*)

end

instantiation *bit0* :: (*finite*) *enum* **begin**
definition (*enum-class.enum* :: *'a bit0 list*) = *map* (*Abs-bit0'* ∘ *int*) (*upt 0* (*CARD*(*'a bit0*)))
definition *enum-class.enum-all P* = (∀ *b* :: *'a bit0* ∈ *set enum-class.enum*. *P b*)
definition *enum-class.enum-ex P* = (∃ *b* :: *'a bit0* ∈ *set enum-class.enum*. *P b*)

instance
proof(*intro-classes*)
show *distinct* (*enum-class.enum* :: *'a bit0 list*)
by (*simp add*: *enum-bit0-def distinct-map inj-on-def Abs-bit0'-def Abs-bit0-inject mod-pos-pos-trivial*)

show *univ-eq*: (*UNIV* :: *'a bit0 set*) = *set enum-class.enum*
unfolding *enum-bit0-def type-definition.Abs-image*[*OF type-definition-bit0*, *symmetric*]
by(*simp add*: *image-comp* [*symmetric*] *inj-on-Abs-bit0 card-image image-int-atLeastLessThan*)
(*auto intro*!: *image-cong*[*OF refl*] *simp add*: *Abs-bit0'-def mod-pos-pos-trivial*)

fix *P* :: *'a bit0* ⇒ *bool*
show *enum-class.enum-all P* = *Ball UNIV P*
and *enum-class.enum-ex P* = *Bex UNIV P*
by(*simp-all add*: *enum-all-bit0-def enum-ex-bit0-def univ-eq*)
qed

end

instantiation *bit1* :: (*finite*) *enum* **begin**
definition (*enum-class.enum* :: *'a bit1 list*) = *map* (*Abs-bit1'* ∘ *int*) (*upt 0* (*CARD*(*'a bit1*)))
definition *enum-class.enum-all P* = (∀ *b* :: *'a bit1* ∈ *set enum-class.enum*. *P b*)
definition *enum-class.enum-ex P* = (∃ *b* :: *'a bit1* ∈ *set enum-class.enum*. *P b*)

instance
proof(*intro-classes*)
show *distinct* (*enum-class.enum* :: *'a bit1 list*)
by(*simp only*: *Abs-bit1'-def zmod-int*[*symmetric*] *enum-bit1-def distinct-map Suc-eq-plus1 card-bit1 o-apply inj-on-def*)
(*clarsimp simp add*: *Abs-bit1-inject*)

show *univ-eq*: (*UNIV* :: *'a bit1 set*) = *set enum-class.enum*
unfolding *enum-bit1-def type-definition.Abs-image*[*OF type-definition-bit1*, *symmetric*]
by(*simp add*: *image-comp* [*symmetric*] *inj-on-Abs-bit1 card-image image-int-atLeastLessThan*)
(*auto intro*!: *image-cong*[*OF refl*] *simp add*: *Abs-bit1'-def mod-pos-pos-trivial*)

fix *P* :: *'a bit1* ⇒ *bool*
show *enum-class.enum-all P* = *Ball UNIV P*
and *enum-class.enum-ex P* = *Bex UNIV P*
by(*simp-all add*: *enum-all-bit1-def enum-ex-bit1-def univ-eq*)
qed

end

instantiation *bit0* **and** *bit1* :: (*finite*) *finite-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*'a bit0*) *True*

definition *finite-UNIV* = *Phantom*('*a bit1*) *True*
instance by *intro-classes* (*simp-all add*: *finite-UNIV-bit0-def finite-UNIV-bit1-def*)
end

instantiation *bit0* **and** *bit1* :: ({*finite*,*card-UNIV*}) *card-UNIV* **begin**
definition *card-UNIV* = *Phantom*('*a bit0*) (*2* ∗ *of-phantom* (*card-UNIV* :: '*a card-UNIV*))
definition *card-UNIV* = *Phantom*('*a bit1*) (*1* + *2* ∗ *of-phantom* (*card-UNIV* :: '*a card-UNIV*))
instance by *intro-classes* (*simp-all add*: *card-UNIV-bit0-def card-UNIV-bit1-def card-UNIV*)
end

29.6 Syntax

syntax
-NumeralType :: *num-token* => *type* (-)
-NumeralType0 :: *type* (*0*)
-NumeralType1 :: *type* (*1*)

translations
(*type*) *1* == (*type*) *num1*
(*type*) *0* == (*type*) *num0*

parse-translation ‹
let
fun mk-bintype n =
let
fun mk-bit 0 = *Syntax.const* @{*type-syntax bit0*}
| *mk-bit 1* = *Syntax.const* @{*type-syntax bit1*};
fun bin-of n =
if n = *1 then Syntax.const* @{*type-syntax num1*}
else if n = *0 then Syntax.const* @{*type-syntax num0*}
else if n = ~*1 then raise TERM* (*negative type numeral*, [])
else
let val (*q*, *r*) = *Integer.div-mod n 2*;
in mk-bit r $ *bin-of q end*;
in bin-of n end;

fun numeral-tr [*Free* (*str*, -)] = *mk-bintype* (*the* (*Int.fromString str*))
| *numeral-tr ts* = *raise TERM* (*numeral-tr*, *ts*);

in [(@{*syntax-const -NumeralType*}, *K numeral-tr*)] *end*;
›

print-translation ‹
let
fun int-of [] = *0*
| *int-of* (*b* :: *bs*) = *b* + *2* ∗ *int-of bs*;

```
  fun bin-of (Const (@{type-syntax num0}, -)) = []
    | bin-of (Const (@{type-syntax num1}, -)) = [1]
    | bin-of (Const (@{type-syntax bit0}, -) $ bs) = 0 :: bin-of bs
    | bin-of (Const (@{type-syntax bit1}, -) $ bs) = 1 :: bin-of bs
    | bin-of t = raise TERM (bin-of, [t]);

  fun bit-tr' b [t] =
      let
        val rev-digs = b :: bin-of t handle TERM - => raise Match
        val i = int-of rev-digs;
        val num = string-of-int (abs i);
      in
        Syntax.const @{syntax-const -NumeralType} $ Syntax.free num
      end
    | bit-tr' b - = raise Match;
 in
  [(@{type-syntax bit0}, K (bit-tr' 0)),
   (@{type-syntax bit1}, K (bit-tr' 1))]
 end;
›
```

29.7 Examples

lemma *CARD(0) = 0* **by** *simp*
lemma *CARD(17) = 17* **by** *simp*
lemma *8 * 11 ^ 3 - 6 = (2::5)* **by** *simp*

end

30 Definition of finite Cartesian product types.

theory *Finite-Cartesian-Product*
imports
Euclidean-Space
L2-Norm
~~/src/HOL/Library/Numeral-Type
begin

30.1 Finite Cartesian products, with indexing and lambdas.

typedef *('a, 'b) vec = UNIV :: ('b::finite) ⇒ 'a) set*
morphisms *vec-nth vec-lambda* **..**

notation
vec-nth (**infixl** $ *90*) **and**
vec-lambda (**binder** *χ 10*)

syntax *-finite-vec* :: *type* ⇒ *type* ⇒ *type* ((- ^/ -) [*15*, *16*] *15*)

parse-translation ‹
let
fun vec t u = Syntax.const @{type-syntax vec} $ t $ u;
fun finite-vec-tr [t, u] =
(case Term-Position.strip-positions u of
v as Free (x, -) =>
if Lexicon.is-tid x then
vec t (Syntax.const @{syntax-const -ofsort} $ v $
Syntax.const @{class-syntax finite})
else vec t u
| - => vec t u)
in
[(@{syntax-const -finite-vec}, K finite-vec-tr)]
end
›

lemma *vec-eq-iff*: (*x* = *y*) ⟷ (∀ *i*. *x*$*i* = *y*$*i*)
by (*simp add*: *vec-nth-inject* [*symmetric*] *fun-eq-iff*)

lemma *vec-lambda-beta* [*simp*]: *vec-lambda g* $ *i* = *g i*
by (*simp add*: *vec-lambda-inverse*)

lemma *vec-lambda-unique*: (∀ *i*. *f*$*i* = *g i*) ⟷ *vec-lambda g* = *f*
by (*auto simp add*: *vec-eq-iff*)

lemma *vec-lambda-eta*: (χ *i*. (*g*$*i*)) = *g*
by (*simp add*: *vec-eq-iff*)

30.2 Group operations and class instances

instantiation *vec* :: (*zero*, *finite*) *zero*
begin
definition *0* ≡ (χ *i*. *0*)
instance ..
end

instantiation *vec* :: (*plus*, *finite*) *plus*
begin
definition *op* + ≡ (λ *x y*. (χ *i*. *x*$*i* + *y*$*i*))
instance ..
end

instantiation *vec* :: (*minus*, *finite*) *minus*
begin
definition *op* − ≡ (λ *x y*. (χ *i*. *x*$*i* − *y*$*i*))
instance ..

end

instantiation *vec* :: (*uminus*, *finite*) *uminus*
begin
definition *uminus* ≡ (λ *x*. (χ *i*. − (*x*$*i*)))
instance ..
end

lemma *zero-index* [*simp*]: *0* $ *i* = *0*
unfolding *zero-vec-def* **by** *simp*

lemma *vector-add-component* [*simp*]: (*x* + *y*)$*i* = *x*$*i* + *y*$*i*
unfolding *plus-vec-def* **by** *simp*

lemma *vector-minus-component* [*simp*]: (*x* − *y*)$*i* = *x*$*i* − *y*$*i*
unfolding *minus-vec-def* **by** *simp*

lemma *vector-uminus-component* [*simp*]: (− *x*)$*i* = − (*x*$*i*)
unfolding *uminus-vec-def* **by** *simp*

instance *vec* :: (*semigroup-add*, *finite*) *semigroup-add*
by *standard* (*simp add*: *vec-eq-iff add.assoc*)

instance *vec* :: (*ab-semigroup-add*, *finite*) *ab-semigroup-add*
by *standard* (*simp add*: *vec-eq-iff add.commute*)

instance *vec* :: (*monoid-add*, *finite*) *monoid-add*
by *standard* (*simp-all add*: *vec-eq-iff*)

instance *vec* :: (*comm-monoid-add*, *finite*) *comm-monoid-add*
by *standard* (*simp add*: *vec-eq-iff*)

instance *vec* :: (*cancel-semigroup-add*, *finite*) *cancel-semigroup-add*
by *standard* (*simp-all add*: *vec-eq-iff*)

instance *vec* :: (*cancel-ab-semigroup-add*, *finite*) *cancel-ab-semigroup-add*
by *standard* (*simp-all add*: *vec-eq-iff diff-diff-eq*)

instance *vec* :: (*cancel-comm-monoid-add*, *finite*) *cancel-comm-monoid-add* **..**

instance *vec* :: (*group-add*, *finite*) *group-add*
by *standard* (*simp-all add*: *vec-eq-iff*)

instance *vec* :: (*ab-group-add*, *finite*) *ab-group-add*
by *standard* (*simp-all add*: *vec-eq-iff*)

30.3 Real vector space

instantiation *vec* :: (*real-vector*, *finite*) *real-vector*

begin

definition *scaleR* ≡ (λ *r x*. (χ *i*. *scaleR r* (*x*$*i*)))

lemma *vector-scaleR-component* [*simp*]: (*scaleR r x*)$*i* = *scaleR r* (*x*$*i*)
  unfolding *scaleR-vec-def* **by** *simp*

instance
  by *standard* (*simp-all add*: *vec-eq-iff scaleR-left-distrib scaleR-right-distrib*)

end

30.4 Topological space

instantiation *vec* :: (*topological-space*, *finite*) *topological-space*
begin

definition [*code del*]:
  open (*S* :: ('*a* ^ '*b*) *set*) ⟷
    (∀ *x*∈*S*. ∃ *A*. (∀ *i*. *open* (*A i*) ∧ *x*$*i* ∈ *A i*) ∧
      (∀ *y*. (∀ *i*. *y*$*i* ∈ *A i*) ⟶ *y* ∈ *S*))

instance proof
  show *open* (*UNIV* :: ('*a* ^ '*b*) *set*)
    unfolding *open-vec-def* **by** *auto*
next
  fix *S T* :: ('*a* ^ '*b*) *set*
  assume *open S open T* **thus** *open* (*S* ∩ *T*)
    unfolding *open-vec-def*
    apply *clarify*
    apply (*drule* (*1*) *bspec*)+
    apply (*clarify*, *rename-tac Sa Ta*)
    apply (*rule-tac x*=λ*i*. *Sa i* ∩ *Ta i* **in** *exI*)
    apply (*simp add*: *open-Int*)
    done
next
  fix *K* :: ('*a* ^ '*b*) *set set*
  assume ∀ *S*∈*K*. *open S* **thus** *open* (⋃ *K*)
    unfolding *open-vec-def*
    apply *clarify*
    apply (*drule* (*1*) *bspec*)
    apply (*drule* (*1*) *bspec*)
    apply *clarify*
    apply (*rule-tac x*=*A* **in** *exI*)
    apply *fast*
    done
qed

end

lemma *open-vector-box*: $\forall i.\ open\ (S\ i) \Longrightarrow open\ \{x.\ \forall i.\ x\ \$\ i \in S\ i\}$
unfolding *open-vec-def* **by** *auto*

lemma *open-vimage-vec-nth*: $open\ S \Longrightarrow open\ ((\lambda x.\ x\ \$\ i) -` S)$
unfolding *open-vec-def*
apply *clarify*
apply (*rule-tac x*=$\lambda k.\ if\ k = i\ then\ S\ else\ UNIV$ **in** *exI*, *simp*)
done

lemma *closed-vimage-vec-nth*: $closed\ S \Longrightarrow closed\ ((\lambda x.\ x\ \$\ i) -` S)$
unfolding *closed-open vimage-Compl* [*symmetric*]
by (*rule open-vimage-vec-nth*)

lemma *closed-vector-box*: $\forall i.\ closed\ (S\ i) \Longrightarrow closed\ \{x.\ \forall i.\ x\ \$\ i \in S\ i\}$
proof −
have $\{x.\ \forall i.\ x\ \$\ i \in S\ i\} = (\bigcap i.\ (\lambda x.\ x\ \$\ i) -` S\ i)$ **by** *auto*
thus $\forall i.\ closed\ (S\ i) \Longrightarrow closed\ \{x.\ \forall i.\ x\ \$\ i \in S\ i\}$
by (*simp add*: *closed-INT closed-vimage-vec-nth*)
qed

lemma *tendsto-vec-nth* [*tendsto-intros*]:
assumes $((\lambda x.\ f\ x) \longrightarrow a)\ net$
shows $((\lambda x.\ f\ x\ \$\ i) \longrightarrow a\ \$\ i)\ net$
proof (*rule topological-tendstoI*)
fix S **assume** $open\ S\ a\ \$\ i \in S$
then have $open\ ((\lambda y.\ y\ \$\ i) -` S)\ a \in ((\lambda y.\ y\ \$\ i) -` S)$
by (*simp-all add*: *open-vimage-vec-nth*)
with *assms* **have** *eventually* $(\lambda x.\ f\ x \in (\lambda y.\ y\ \$\ i) -` S)\ net$
by (*rule topological-tendstoD*)
then show *eventually* $(\lambda x.\ f\ x\ \$\ i \in S)\ net$
by *simp*
qed

lemma *isCont-vec-nth* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ f\ x\ \$\ i)\ a$
unfolding *isCont-def* **by** (*rule tendsto-vec-nth*)

lemma *vec-tendstoI*:
assumes $\bigwedge i.\ ((\lambda x.\ f\ x\ \$\ i) \longrightarrow a\ \$\ i)\ net$
shows $((\lambda x.\ f\ x) \longrightarrow a)\ net$
proof (*rule topological-tendstoI*)
fix S **assume** $open\ S$ **and** $a \in S$
then obtain A **where** A: $\bigwedge i.\ open\ (A\ i)\ \bigwedge i.\ a\ \$\ i \in A\ i$
and S: $\bigwedge y.\ \forall i.\ y\ \$\ i \in A\ i \Longrightarrow y \in S$
unfolding *open-vec-def* **by** *metis*
have $\bigwedge i.\ eventually\ (\lambda x.\ f\ x\ \$\ i \in A\ i)\ net$
using *assms A* **by** (*rule topological-tendstoD*)
hence *eventually* $(\lambda x.\ \forall i.\ f\ x\ \$\ i \in A\ i)\ net$
by (*rule eventually-all-finite*)

thus *eventually* $(\lambda x.\ f\ x \in S)$ *net*
by (*rule eventually-mono*, *simp add*: S)
qed

lemma *tendsto-vec-lambda* [*tendsto-intros*]:
assumes $\bigwedge i.\ ((\lambda x.\ f\ x\ i) \longrightarrow a\ i)$ *net*
shows $((\lambda x.\ \chi\ i.\ f\ x\ i) \longrightarrow (\chi\ i.\ a\ i))$ *net*
using *assms* **by** (*simp add*: *vec-tendstoI*)

lemma *open-image-vec-nth*: **assumes** *open* S **shows** *open* $((\lambda x.\ x\ \$\ i)\ `\ S)$
proof (*rule openI*)
fix a **assume** $a \in (\lambda x.\ x\ \$\ i)\ `\ S$
then obtain z **where** $a = z\ \$\ i$ **and** $z \in S$ **..**
then obtain A **where** A: $\forall i.\ open\ (A\ i) \wedge z\ \$\ i \in A\ i$
and S: $\forall y.\ (\forall i.\ y\ \$\ i \in A\ i) \longrightarrow y \in S$
using ‹*open S*› **unfolding** *open-vec-def* **by** *auto*
hence $A\ i \subseteq (\lambda x.\ x\ \$\ i)\ `\ S$
by (*clarsimp*, *rule-tac* $x{=}\chi\ j.\ if\ j = i\ then\ x\ else\ z\ \$\ j$ **in** *image-eqI*, *simp-all*)
hence *open* $(A\ i) \wedge a \in A\ i \wedge A\ i \subseteq (\lambda x.\ x\ \$\ i)\ `\ S$
using A ‹$a = z\ \$\ i$› **by** *simp*
then show $\exists T.\ open\ T \wedge a \in T \wedge T \subseteq (\lambda x.\ x\ \$\ i)\ `\ S$ **by** $-$ (*rule exI*)
qed

instance *vec* :: (*perfect-space*, *finite*) *perfect-space*
proof
fix x :: $'a\ \hat{}\ 'b$ **show** $\neg$ *open* $\{x\}$
proof
assume *open* $\{x\}$
hence $\forall i.\ open\ ((\lambda x.\ x\ \$\ i)\ `\ \{x\})$ **by** (*fast intro*: *open-image-vec-nth*)
hence $\forall i.\ open\ \{x\ \$\ i\}$ **by** *simp*
thus *False* **by** (*simp add*: *not-open-singleton*)
qed
qed

30.5 Metric space

instantiation *vec* :: (*metric-space*, *finite*) *dist*
begin

definition
dist x y = *setL2* $(\lambda i.\ dist\ (x\$i)\ (y\$i))$ *UNIV*

instance ..
end

instantiation *vec* :: (*metric-space*, *finite*) *uniformity-dist*
begin

definition [*code del*]:
(*uniformity* :: (('*a*, '*b*) *vec* × ('*a*, '*b*) *vec*) *filter*) =
(*INF e*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

instance
by *standard* (*rule uniformity-vec-def*)
end

declare *uniformity-Abort*[**where** '*a*='*a* :: *metric-space* ^ '*b* :: *finite*, *code*]

instantiation *vec* :: (*metric-space*, *finite*) *metric-space*
begin

lemma *dist-vec-nth-le*: *dist* (*x* $ *i*) (*y* $ *i*) ≤ *dist x y*
unfolding *dist-vec-def* **by** (*rule member-le-setL2*) *simp-all*

instance proof
fix *x y* :: '*a* ^ '*b*
show *dist x y* = *0* ⟷ *x* = *y*
unfolding *dist-vec-def*
by (*simp add*: *setL2-eq-0-iff vec-eq-iff*)
next
fix *x y z* :: '*a* ^ '*b*
show *dist x y* ≤ *dist x z* + *dist y z*
unfolding *dist-vec-def*
apply (*rule order-trans* [*OF* - *setL2-triangle-ineq*])
apply (*simp add*: *setL2-mono dist-triangle2*)
done
next
fix *S* :: ('*a* ^ '*b*) *set*
have ∗: *open S* ⟷ (∀ *x*∈*S*. ∃ *e*>*0*. ∀ *y*. *dist y x* < *e* ⟶ *y* ∈ *S*)
proof
assume *open S* **show** ∀ *x*∈*S*. ∃ *e*>*0*. ∀ *y*. *dist y x* < *e* ⟶ *y* ∈ *S*
proof
fix *x* **assume** *x* ∈ *S*
obtain *A* **where** *A*: ∀ *i*. *open* (*A i*) ∀ *i*. *x* $ *i* ∈ *A i*
and *S*: ∀ *y*. (∀ *i*. *y* $ *i* ∈ *A i*) ⟶ *y* ∈ *S*
using ‹*open S*› **and** ‹*x* ∈ *S*› **unfolding** *open-vec-def* **by** *metis*
have ∀ *i*∈*UNIV*. ∃ *r*>*0*. ∀ *y*. *dist y* (*x* $ *i*) < *r* ⟶ *y* ∈ *A i*
using *A* **unfolding** *open-dist* **by** *simp*
hence ∃ *r*. ∀ *i*∈*UNIV*. *0* < *r i* ∧ (∀ *y*. *dist y* (*x* $ *i*) < *r i* ⟶ *y* ∈ *A i*)
by (*rule finite-set-choice* [*OF finite*])
then obtain *r* **where** *r1*: ∀ *i*. *0* < *r i*
and *r2*: ∀ *i y*. *dist y* (*x* $ *i*) < *r i* ⟶ *y* ∈ *A i* **by** *fast*
have *0* < *Min* (*range r*) ∧ (∀ *y*. *dist y x* < *Min* (*range r*) ⟶ *y* ∈ *S*)
by (*simp add*: *r1 r2 S le-less-trans* [*OF dist-vec-nth-le*])
thus ∃ *e*>*0*. ∀ *y*. *dist y x* < *e* ⟶ *y* ∈ *S* **..**
qed
next

```
    assume ∗: ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S show open S
    proof (unfold open-vec-def, rule)
      fix x assume x ∈ S
      then obtain e where 0 < e and S: ∀ y. dist y x < e ⟶ y ∈ S
        using ∗ by fast
      def r ≡ λi::'b. e / sqrt (of-nat CARD('b))
      from ‹0 < e› have r: ∀ i. 0 < r i
        unfolding r-def by simp-all
      from ‹0 < e› have e: e = setL2 r UNIV
        unfolding r-def by (simp add: setL2-constant)
      def A ≡ λi. {y. dist (x $ i) y < r i}
      have ∀ i. open (A i) ∧ x $ i ∈ A i
        unfolding A-def by (simp add: open-ball r)
      moreover have ∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S
        by (simp add: A-def S dist-vec-def e setL2-strict-mono dist-commute)
      ultimately show ∃ A. (∀ i. open (A i) ∧ x $ i ∈ A i) ∧
        (∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S) by metis
    qed
  qed
  show open S = (∀ x∈S. ∀ F (x', y) in uniformity. x' = x ⟶ y ∈ S)
    unfolding ∗ eventually-uniformity-metric
    by (simp del: split-paired-All add: dist-vec-def dist-commute)
qed

end

lemma Cauchy-vec-nth:
  Cauchy (λn. X n) ⟹ Cauchy (λn. X n $ i)
  unfolding Cauchy-def by (fast intro: le-less-trans [OF dist-vec-nth-le])

lemma vec-CauchyI:
  fixes X :: nat ⇒ 'a::metric-space ^ 'n
  assumes X: ⋀i. Cauchy (λn. X n $ i)
  shows Cauchy (λn. X n)
proof (rule metric-CauchyI)
  fix r :: real assume 0 < r
  hence 0 < r / of-nat CARD('n) (is 0 < ?s) by simp
  def N ≡ λi. LEAST N. ∀ m≥N. ∀ n≥N. dist (X m $ i) (X n $ i) < ?s
  def M ≡ Max (range N)
  have ⋀i. ∃ N. ∀ m≥N. ∀ n≥N. dist (X m $ i) (X n $ i) < ?s
    using X ‹0 < ?s› by (rule metric-CauchyD)
  hence ⋀i. ∀ m≥N i. ∀ n≥N i. dist (X m $ i) (X n $ i) < ?s
    unfolding N-def by (rule LeastI-ex)
  hence M: ⋀i. ∀ m≥M. ∀ n≥M. dist (X m $ i) (X n $ i) < ?s
    unfolding M-def by simp
  {
    fix m n :: nat
    assume M ≤ m M ≤ n
    have dist (X m) (X n) = setL2 (λi. dist (X m $ i) (X n $ i)) UNIV
```

unfolding *dist-vec-def* **..**
also have $\ldots \leq$ *setsum* $(\lambda i.\ dist\ (X\ m\ \$\ i)\ (X\ n\ \$\ i))\ UNIV$
by (*rule setL2-le-setsum* [*OF zero-le-dist*])
also have $\ldots <$ *setsum* $(\lambda i{::}'n.\ ?s)\ UNIV$
by (*rule setsum-strict-mono*, *simp-all add*: $M\ \langle M \leq m \rangle\ \langle M \leq n \rangle$)
also have $\ldots = r$
by *simp*
finally have $dist\ (X\ m)\ (X\ n) < r$ **.**
}
hence $\forall\, m{\geq}M.\ \forall\, n{\geq}M.\ dist\ (X\ m)\ (X\ n) < r$
by *simp*
then show $\exists\, M.\ \forall\, m{\geq}M.\ \forall\, n{\geq}M.\ dist\ (X\ m)\ (X\ n) < r$ **..**
qed

instance *vec* :: (*complete-space*, *finite*) *complete-space*
proof
fix X :: $nat \Rightarrow {'a}\hat{}{'b}$ **assume** *Cauchy X*
have $\bigwedge i.\ (\lambda n.\ X\ n\ \$\ i) \longrightarrow lim\ (\lambda n.\ X\ n\ \$\ i)$
using *Cauchy-vec-nth* [*OF* $\langle Cauchy\ X \rangle$]
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
hence $X \longrightarrow$ *vec-lambda* $(\lambda i.\ lim\ (\lambda n.\ X\ n\ \$\ i))$
by (*simp add*: *vec-tendstoI*)
then show *convergent X*
by (*rule convergentI*)
qed

30.6 Normed vector space

instantiation *vec* :: (*real-normed-vector*, *finite*) *real-normed-vector*
begin

definition $norm\ x = setL2\ (\lambda i.\ norm\ (x\$i))\ UNIV$

definition $sgn\ (x{::}{'a}\hat{}{'b}) = scaleR\ (inverse\ (norm\ x))\ x$

instance proof
fix a :: *real* **and** $x\ y$ :: ${'a}\hat{}{'b}$
show $norm\ x = 0 \longleftrightarrow x = 0$
unfolding *norm-vec-def*
by (*simp add*: *setL2-eq-0-iff vec-eq-iff*)
show $norm\ (x + y) \leq norm\ x + norm\ y$
unfolding *norm-vec-def*
apply (*rule order-trans* [*OF - setL2-triangle-ineq*])
apply (*simp add*: *setL2-mono norm-triangle-ineq*)
done
show $norm\ (scaleR\ a\ x) = |a| * norm\ x$
unfolding *norm-vec-def*
by (*simp add*: *setL2-right-distrib*)
show $sgn\ x = scaleR\ (inverse\ (norm\ x))\ x$

by (*rule sgn-vec-def*)
show *dist x y = norm (x − y)*
unfolding *dist-vec-def norm-vec-def*
by (*simp add*: *dist-norm*)
qed

end

lemma *norm-nth-le*: *norm (x $ i) ≤ norm x*
unfolding *norm-vec-def*
by (*rule member-le-setL2*) *simp-all*

lemma *bounded-linear-vec-nth*: *bounded-linear (λx. x $ i)*
apply *standard*
apply (*rule vector-add-component*)
apply (*rule vector-scaleR-component*)
apply (*rule-tac x=1* **in** *exI*, *simp add*: *norm-nth-le*)
done

instance *vec* :: (*banach*, *finite*) *banach* **..**

30.7 Inner product space

instantiation *vec* :: (*real-inner*, *finite*) *real-inner*
begin

definition *inner x y = setsum (λi. inner (x$i) (y$i)) UNIV*

instance proof
fix *r* :: *real* **and** *x y z* :: *'a ^ 'b*
show *inner x y = inner y x*
unfolding *inner-vec-def*
by (*simp add*: *inner-commute*)
show *inner (x + y) z = inner x z + inner y z*
unfolding *inner-vec-def*
by (*simp add*: *inner-add-left setsum.distrib*)
show *inner (scaleR r x) y = r * inner x y*
unfolding *inner-vec-def*
by (*simp add*: *setsum-right-distrib*)
show *0 ≤ inner x x*
unfolding *inner-vec-def*
by (*simp add*: *setsum-nonneg*)
show *inner x x = 0 ⟷ x = 0*
unfolding *inner-vec-def*
by (*simp add*: *vec-eq-iff setsum-nonneg-eq-0-iff*)
show *norm x = sqrt (inner x x)*
unfolding *inner-vec-def norm-vec-def setL2-def*
by (*simp add*: *power2-norm-eq-inner*)
qed

end

30.8 Euclidean space

Vectors pointing along a single axis.

definition *axis k x* = (*χ i. if i = k then x else 0*)

lemma *axis-nth* [*simp*]: *axis i x $ i = x*
unfolding *axis-def* **by** *simp*

lemma *axis-eq-axis*: *axis i x = axis j y ⟷ x = y ∧ i = j ∨ x = 0 ∧ y = 0*
unfolding *axis-def vec-eq-iff* **by** *auto*

lemma *inner-axis-axis*:
inner (*axis i x*) (*axis j y*) = (*if i = j then inner x y else 0*)
unfolding *inner-vec-def*
apply (*cases i = j*)
apply *clarsimp*
apply (*subst setsum.remove* [*of - j*], *simp-all*)
apply (*rule setsum.neutral*, *simp add*: *axis-def*)
apply (*rule setsum.neutral*, *simp add*: *axis-def*)
done

lemma *setsum-single*:
assumes *finite A* **and** *k ∈ A* **and** *f k = y*
assumes ⋀*i. i ∈ A ⟹ i ≠ k ⟹ f i = 0*
shows (∑ *i∈A. f i*) = *y*
apply (*subst setsum.remove* [*OF assms*(*1*,*2*)])
apply (*simp add*: *setsum.neutral assms*(*3*,*4*))
done

lemma *inner-axis*: *inner x* (*axis i y*) = *inner* (*x $ i*) *y*
unfolding *inner-vec-def*
apply (*rule-tac k*=*i* **in** *setsum-single*)
apply *simp-all*
apply (*simp add*: *axis-def*)
done

instantiation *vec* :: (*euclidean-space*, *finite*) *euclidean-space*
begin

definition *Basis* = (⋃ *i*. ⋃ *u∈Basis*. {*axis i u*})

instance proof
show (*Basis* :: (*'a ^ 'b*) *set*) ≠ {}
unfolding *Basis-vec-def* **by** *simp*
next
show *finite* (*Basis* :: (*'a ^ 'b*) *set*)

```
    unfolding Basis-vec-def by simp
next
  fix u v :: 'a ^ 'b
  assume u ∈ Basis and v ∈ Basis
  thus inner u v = (if u = v then 1 else 0)
    unfolding Basis-vec-def
    by (auto simp add: inner-axis-axis axis-eq-axis inner-Basis)
next
  fix x :: 'a ^ 'b
  show (∀ u∈Basis. inner x u = 0) ⟷ x = 0
    unfolding Basis-vec-def
    by (simp add: inner-axis euclidean-all-zero-iff vec-eq-iff)
qed

lemma DIM-cart[simp]: DIM('a^'b) = CARD('b) * DIM('a)
  apply (simp add: Basis-vec-def)
  apply (subst card-UN-disjoint)
     apply simp
    apply simp
   apply (auto simp: axis-eq-axis) [1]
  apply (subst card-UN-disjoint)
    apply (auto simp: axis-eq-axis)
  done

end

end
```

31 Operator Norm

```
theory Operator-Norm
imports Complex-Main
begin
```

This formulation yields zero if $'a$ is the trivial vector space.

```
definition onorm :: ('a::real-normed-vector ⇒ 'b::real-normed-vector) ⇒ real
  where onorm f = (SUP x. norm (f x) / norm x)

lemma onorm-bound:
  assumes 0 ≤ b and ⋀x. norm (f x) ≤ b * norm x
  shows onorm f ≤ b
  unfolding onorm-def
proof (rule cSUP-least)
  fix x
  show norm (f x) / norm x ≤ b
    using assms by (cases x = 0) (simp-all add: pos-divide-le-eq)
qed simp
```

In non-trivial vector spaces, the first assumption is redundant.

lemma *onorm-le*:
fixes f :: $'a$::{*real-normed-vector*, *perfect-space*} ⇒ $'b$::*real-normed-vector*
assumes ⋀x. *norm* (f x) ≤ b ∗ *norm* x
shows *onorm* f ≤ b
proof (*rule onorm-bound* [*OF* - *assms*])
have {0::$'a$} ≠ *UNIV* **by** (*metis not-open-singleton open-UNIV*)
then obtain a :: $'a$ **where** a ≠ 0 **by** *fast*
have 0 ≤ b ∗ *norm* a
by (*rule order-trans* [*OF norm-ge-zero assms*])
with ‹a ≠ 0› **show** 0 ≤ b
by (*simp add*: *zero-le-mult-iff*)
qed

lemma *le-onorm*:
assumes *bounded-linear* f
shows *norm* (f x) / *norm* x ≤ *onorm* f
proof −
interpret f: *bounded-linear* f **by** *fact*
obtain b **where** 0 ≤ b **and** ∀x. *norm* (f x) ≤ *norm* x ∗ b
using f.*nonneg-bounded* **by** *auto*
then have ∀x. *norm* (f x) / *norm* x ≤ b
by (*clarify*, *case-tac* x = 0,
simp-all add: f.*zero pos-divide-le-eq mult.commute*)
then have *bdd-above* (*range* (λx. *norm* (f x) / *norm* x))
unfolding *bdd-above-def* **by** *fast*
with *UNIV-I* **show** *?thesis*
unfolding *onorm-def* **by** (*rule cSUP-upper*)
qed

lemma *onorm*:
assumes *bounded-linear* f
shows *norm* (f x) ≤ *onorm* f ∗ *norm* x
proof −
interpret f: *bounded-linear* f **by** *fact*
show *?thesis*
proof (*cases*)
assume x = 0
then show *?thesis* **by** (*simp add*: f.*zero*)
next
assume x ≠ 0
have *norm* (f x) / *norm* x ≤ *onorm* f
by (*rule le-onorm* [*OF assms*])
then show *norm* (f x) ≤ *onorm* f ∗ *norm* x
by (*simp add*: *pos-divide-le-eq* ‹x ≠ 0›)
qed
qed

lemma *onorm-pos-le*:
assumes f: *bounded-linear* f

shows $0 \leq$ *onorm f*
using *le-onorm* [*OF f*, **where** *x=0*] **by** *simp*

lemma *onorm-zero*: *onorm* ($\lambda x.\ 0$) = *0*
proof (*rule order-antisym*)
show *onorm* ($\lambda x.\ 0$) $\leq$ *0*
by (*simp add*: *onorm-bound*)
show *0* $\leq$ *onorm* ($\lambda x.\ 0$)
using *bounded-linear-zero* **by** (*rule onorm-pos-le*)
qed

lemma *onorm-eq-0*:
assumes *f*: *bounded-linear f*
shows *onorm f* = *0* $\longleftrightarrow$ ($\forall x.\ f\ x$ = *0*)
using *onorm* [*OF f*] **by** (*auto simp*: *fun-eq-iff* [*symmetric*] *onorm-zero*)

lemma *onorm-pos-lt*:
assumes *f*: *bounded-linear f*
shows *0* < *onorm f* $\longleftrightarrow$ $\neg$ ($\forall x.\ f\ x$ = *0*)
by (*simp add*: *less-le onorm-pos-le* [*OF f*] *onorm-eq-0* [*OF f*])

lemma *onorm-id-le*: *onorm* ($\lambda x.\ x$) $\leq$ *1*
by (*rule onorm-bound*) *simp-all*

lemma *onorm-id*: *onorm* ($\lambda x.\ x$::'*a*::{*real-normed-vector*, *perfect-space*}) = *1*
proof (*rule antisym*[*OF onorm-id-le*])
have {*0*::'*a*} $\neq$ *UNIV* **by** (*metis not-open-singleton open-UNIV*)
then obtain *x* :: '*a* **where** *x* $\neq$ *0* **by** *fast*
hence *1* $\leq$ *norm x* / *norm x*
by *simp*
also have ... $\leq$ *onorm* (λ*x*::'*a*. *x*)
by (*rule le-onorm*) (*rule bounded-linear-ident*)
finally show *1* $\leq$ *onorm* (λ*x*::'*a*. *x*) **.**
qed

lemma *onorm-compose*:
assumes *f*: *bounded-linear f*
assumes *g*: *bounded-linear g*
shows *onorm* (*f* $\circ$ *g*) $\leq$ *onorm f* $*$ *onorm g*
proof (*rule onorm-bound*)
show *0* $\leq$ *onorm f* $*$ *onorm g*
by (*intro mult-nonneg-nonneg onorm-pos-le f g*)
next
fix *x*
have *norm* (*f* (*g x*)) $\leq$ *onorm f* $*$ *norm* (*g x*)
by (*rule onorm* [*OF f*])
also have *onorm f* $*$ *norm* (*g x*) $\leq$ *onorm f* $*$ (*onorm g* $*$ *norm x*)
by (*rule mult-left-mono* [*OF onorm* [*OF g*] *onorm-pos-le* [*OF f*]])
finally show *norm* ((*f* $\circ$ *g*) *x*) $\leq$ *onorm f* $*$ *onorm g* $*$ *norm x*

```
    by (simp add: mult.assoc)
qed

lemma onorm-scaleR-lemma:
  assumes f: bounded-linear f
  shows onorm (λx. r *R f x) ≤ |r| * onorm f
proof (rule onorm-bound)
  show 0 ≤ |r| * onorm f
    by (intro mult-nonneg-nonneg onorm-pos-le abs-ge-zero f)
next
  fix x
  have |r| * norm (f x) ≤ |r| * (onorm f * norm x)
    by (intro mult-left-mono onorm abs-ge-zero f)
  then show norm (r *R f x) ≤ |r| * onorm f * norm x
    by (simp only: norm-scaleR mult.assoc)
qed

lemma onorm-scaleR:
  assumes f: bounded-linear f
  shows onorm (λx. r *R f x) = |r| * onorm f
proof (cases r = 0)
  assume r ≠ 0
  show ?thesis
  proof (rule order-antisym)
    show onorm (λx. r *R f x) ≤ |r| * onorm f
      using f by (rule onorm-scaleR-lemma)
  next
    have bounded-linear (λx. r *R f x)
      using bounded-linear-scaleR-right f by (rule bounded-linear-compose)
    then have onorm (λx. inverse r *R r *R f x) ≤ |inverse r| * onorm (λx. r
*R f x)
      by (rule onorm-scaleR-lemma)
    with ‹r ≠ 0› show |r| * onorm f ≤ onorm (λx. r *R f x)
      by (simp add: inverse-eq-divide pos-le-divide-eq mult.commute)
  qed
qed (simp add: onorm-zero)

lemma onorm-scaleR-left-lemma:
  assumes r: bounded-linear r
  shows onorm (λx. r x *R f) ≤ onorm r * norm f
proof (rule onorm-bound)
  fix x
  have norm (r x *R f) = norm (r x) * norm f
    by simp
  also have ... ≤ onorm r * norm x * norm f
    by (intro mult-right-mono onorm r norm-ge-zero)
  finally show norm (r x *R f) ≤ onorm r * norm f * norm x
    by (simp add: ac-simps)
qed (intro mult-nonneg-nonneg norm-ge-zero onorm-pos-le r)
```

lemma *onorm-scaleR-left*:
assumes *f*: *bounded-linear r*
shows *onorm* (λ*x*. *r x* ∗$_R$ *f*) = *onorm r* ∗ *norm f*
proof (*cases f* = *0*)
assume *f* ≠ *0*
show *?thesis*
proof (*rule order-antisym*)
show *onorm* (λ*x*. *r x* ∗$_R$ *f*) ≤ *onorm r* ∗ *norm f*
using *f* **by** (*rule onorm-scaleR-left-lemma*)
next
have *bl1*: *bounded-linear* (λ*x*. *r x* ∗$_R$ *f*)
by (*metis bounded-linear-scaleR-const f*)
have *bounded-linear* (λ*x*. *r x* ∗ *norm f*)
by (*metis bounded-linear-mult-const f*)
from *onorm-scaleR-left-lemma*[*OF this*, *of inverse* (*norm f*)]
have *onorm r* ≤ *onorm* (λ*x*. *r x* ∗ *norm f*) ∗ *inverse* (*norm f*)
using ‹*f* ≠ *0*›
by (*simp add*: *inverse-eq-divide*)
also have *onorm* (λ*x*. *r x* ∗ *norm f*) ≤ *onorm* (λ*x*. *r x* ∗$_R$ *f*)
by (*rule onorm-bound*)
(*auto simp*: *abs-mult bl1 onorm-pos-le intro*!: *order-trans*[*OF* - *onorm*])
finally show *onorm r* ∗ *norm f* ≤ *onorm* (λ*x*. *r x* ∗$_R$ *f*)
using ‹*f* ≠ *0*›
by (*simp add*: *inverse-eq-divide pos-le-divide-eq mult.commute*)
qed
qed (*simp add*: *onorm-zero*)

lemma *onorm-neg*:
shows *onorm* (λ*x*. − *f x*) = *onorm f*
unfolding *onorm-def* **by** *simp*

lemma *onorm-triangle*:
assumes *f*: *bounded-linear f*
assumes *g*: *bounded-linear g*
shows *onorm* (λ*x*. *f x* + *g x*) ≤ *onorm f* + *onorm g*
proof (*rule onorm-bound*)
show *0* ≤ *onorm f* + *onorm g*
by (*intro add-nonneg-nonneg onorm-pos-le f g*)
next
fix *x*
have *norm* (*f x* + *g x*) ≤ *norm* (*f x*) + *norm* (*g x*)
by (*rule norm-triangle-ineq*)
also have *norm* (*f x*) + *norm* (*g x*) ≤ *onorm f* ∗ *norm x* + *onorm g* ∗ *norm x*
by (*intro add-mono onorm f g*)
finally show *norm* (*f x* + *g x*) ≤ (*onorm f* + *onorm g*) ∗ *norm x*
by (*simp only*: *distrib-right*)
qed

lemma *onorm-triangle-le*:
assumes *bounded-linear f*
assumes *bounded-linear g*
assumes *onorm f + onorm g* $\leq e$
shows *onorm* $(\lambda x.\ f\ x + g\ x) \leq e$
using *assms* **by** (*rule onorm-triangle* [*THEN order-trans*])

lemma *onorm-triangle-lt*:
assumes *bounded-linear f*
assumes *bounded-linear g*
assumes *onorm f + onorm g* $< e$
shows *onorm* $(\lambda x.\ f\ x + g\ x) < e$
using *assms* **by** (*rule onorm-triangle* [*THEN order-le-less-trans*])

end

32 Uniform Limit and Uniform Convergence

theory *Uniform-Limit*
imports *Topology-Euclidean-Space*
begin

definition *uniformly-on* :: *'a set* $\Rightarrow$ (*'a* $\Rightarrow$ *'b*::*metric-space*) $\Rightarrow$ (*'a* $\Rightarrow$ *'b*) *filter*
where *uniformly-on S l* = (*INF e*:{*0* <..}. *principal* {*f*. $\forall x{\in}S.$ *dist* (*f x*) (*l x*) < *e*})

abbreviation
uniform-limit S f l $\equiv$ *filterlim f* (*uniformly-on S l*)

definition *uniformly-convergent-on* **where**
uniformly-convergent-on X f $\longleftrightarrow$ ($\exists\, l.$ *uniform-limit X f l sequentially*)

definition *uniformly-Cauchy-on* **where**
uniformly-Cauchy-on X f $\longleftrightarrow$ ($\forall\, e{>}0.\ \exists\, M.\ \forall\, x{\in}X.\ \forall\, (m{::}nat){\geq}M.\ \forall\, n{\geq}M.$ *dist* (*f m x*) (*f n x*) < *e*)

lemma *uniform-limit-iff*:
uniform-limit S f l F $\longleftrightarrow$ ($\forall\, e{>}0.\ \forall_F$ *n in F*. $\forall\, x{\in}S.$ *dist* (*f n x*) (*l x*) < *e*)
unfolding *filterlim-iff uniformly-on-def*
by (*subst eventually-INF-base*)
(*fastforce*
simp: *eventually-principal uniformly-on-def*
intro: *bexI*[**where** *x*=*min a b* **for** *a b*]
elim: *eventually-mono*)+

lemma *uniform-limitD*:
uniform-limit S f l F $\Longrightarrow$ *e* > *0* $\Longrightarrow$ $\forall_F$ *n in F*. $\forall\, x{\in}S.$ *dist* (*f n x*) (*l x*) < *e*
by (*simp add*: *uniform-limit-iff*)

lemma *uniform-limitI*:
(⋀*e*. *e* > *0* ⟹ ∀ *F* *n* *in* *F*. ∀ *x*∈*S*. *dist* (*f n x*) (*l x*) < *e*) ⟹ *uniform-limit S f l F*
by (*simp add*: *uniform-limit-iff*)

lemma *uniform-limit-sequentially-iff*:
uniform-limit S f l sequentially ⟷ (∀ *e*>*0*. ∃ *N*. ∀ *n*≥*N*. ∀ *x* ∈ *S*. *dist* (*f n x*) (*l x*) < *e*)
unfolding *uniform-limit-iff eventually-sequentially* **..**

lemma *uniform-limit-at-iff*:
uniform-limit S f l (*at x*) ⟷
(∀ *e*>*0*. ∃ *d*>*0*. ∀ *z*. *0* < *dist z x* ∧ *dist z x* < *d* ⟶ (∀ *x*∈*S*. *dist* (*f z x*) (*l x*) < *e*))
unfolding *uniform-limit-iff eventually-at2* **..**

lemma *uniform-limit-at-le-iff*:
uniform-limit S f l (*at x*) ⟷
(∀ *e*>*0*. ∃ *d*>*0*. ∀ *z*. *0* < *dist z x* ∧ *dist z x* < *d* ⟶ (∀ *x*∈*S*. *dist* (*f z x*) (*l x*) ≤ *e*))
unfolding *uniform-limit-iff eventually-at2*
by (*fastforce dest*: *spec*[**where** *x* = *e* / *2* **for** *e*])

lemma *swap-uniform-limit*:
assumes *f*: ∀ *F* *n* *in* *F*. (*f n* ⟶ *g n*) (*at x within S*)
assumes *g*: (*g* ⟶ *l*) *F*
assumes *uc*: *uniform-limit S f h F*
assumes ¬*trivial-limit F*
shows (*h* ⟶ *l*) (*at x within S*)
proof (*rule tendstoI*)
fix *e* :: *real*
def *e'* ≡ *e*/*3*
assume *0* < *e*
then have *0* < *e'* **by** (*simp add*: *e'-def*)
from *uniform-limitD*[*OF uc* ‹*0* < *e'*›]
have ∀ *F* *n* *in* *F*. ∀ *x*∈*S*. *dist* (*h x*) (*f n x*) < *e'*
by (*simp add*: *dist-commute*)
moreover
from *f*
have ∀ *F* *n* *in* *F*. ∀ *F* *x* *in* *at x within S*. *dist* (*g n*) (*f n x*) < *e'*
by *eventually-elim* (*auto dest*!: *tendstoD*[*OF* - ‹*0* < *e'*›] *simp*: *dist-commute*)
moreover
from *tendstoD*[*OF g* ‹*0* < *e'*›] **have** ∀ *F* *x* *in* *F*. *dist l* (*g x*) < *e'*
by (*simp add*: *dist-commute*)
ultimately
have ∀ *F* - *in* *F*. ∀ *F* *x* *in* *at x within S*. *dist* (*h x*) *l* < *e*
proof *eventually-elim*
case (*elim n*)
note *fh* = *elim*(*1*)

```
    note gl = elim(3)
    have ∀F x in at x within S. x ∈ S
      by (auto simp: eventually-at-filter)
    with elim(2)
    show ?case
    proof eventually-elim
      case (elim x)
      from fh[rule-format, OF ‹x ∈ S›] elim(1)
      have dist (h x) (g n) < e' + e'
        by (rule dist-triangle-lt[OF add-strict-mono])
      from dist-triangle-lt[OF add-strict-mono, OF this gl]
      show ?case by (simp add: e'-def)
    qed
  qed
  thus ∀F x in at x within S. dist (h x) l < e
    using eventually-happens by (metis ‹¬trivial-limit F›)
qed

lemma
  tendsto-uniform-limitI:
  assumes uniform-limit S f l F
  assumes x ∈ S
  shows ((λy. f y x) ⟶ l x) F
  using assms
  by (auto intro!: tendstoI simp: eventually-mono dest!: uniform-limitD)

lemma uniform-limit-theorem:
  assumes c: ∀F n in F. continuous-on A (f n)
  assumes ul: uniform-limit A f l F
  assumes ¬ trivial-limit F
  shows continuous-on A l
  unfolding continuous-on-def
proof safe
  fix x assume x ∈ A
  then have ∀F n in F. (f n ⟶ f n x) (at x within A) ((λn. f n x) ⟶ l x)
F
    using c ul
    by (auto simp: continuous-on-def eventually-mono tendsto-uniform-limitI)
  then show (l ⟶ l x) (at x within A)
    by (rule swap-uniform-limit) fact+
qed

lemma uniformly-Cauchy-onI:
  assumes ⋀e. e > 0 ⟹ ∃M. ∀x∈X. ∀m≥M. ∀n≥M. dist (f m x) (f n x) < e
  shows   uniformly-Cauchy-on X f
  using assms unfolding uniformly-Cauchy-on-def by blast

lemma uniformly-Cauchy-onI':
  assumes ⋀e. e > 0 ⟹ ∃M. ∀x∈X. ∀m≥M. ∀n>m. dist (f m x) (f n x) < e
```

shows *uniformly-Cauchy-on X f*
proof (*rule uniformly-Cauchy-onI*)
fix *e* :: *real* **assume** *e*: *e* > *0*
from *assms*[*OF this*] **obtain** *M*
where *M*: ⋀*x m n*. *x* ∈ *X* ⟹ *m* ≥ *M* ⟹ *n* > *m* ⟹ *dist* (*f m x*) (*f n x*) < *e* **by** *fast*
{
fix *x m n* **assume** *x*: *x* ∈ *X* **and** *m*: *m* ≥ *M* **and** *n*: *n* ≥ *M*
with *M*[*OF this*(*1*,*2*), *of n*] *M*[*OF this*(*1*,*3*), *of m*] *e* **have** *dist* (*f m x*) (*f n x*) < *e*
by (*cases m n rule*: *linorder-cases*) (*simp-all add*: *dist-commute*)
}
thus ∃ *M*. ∀ *x*∈*X*. ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*f m x*) (*f n x*) < *e* **by** *fast*
qed

lemma *uniformly-Cauchy-imp-Cauchy*:
uniformly-Cauchy-on X f ⟹ *x* ∈ *X* ⟹ *Cauchy* (λ*n*. *f n x*)
unfolding *Cauchy-def uniformly-Cauchy-on-def* **by** *fast*

lemma *uniform-limit-cong*:
fixes *f g* :: *'a* ⇒ *'b* ⇒ (*'c* :: *metric-space*) **and** *h i* :: *'b* ⇒ *'c*
assumes *eventually* (λ*y*. ∀ *x*∈*X*. *f y x* = *g y x*) *F*
assumes ⋀*x*. *x* ∈ *X* ⟹ *h x* = *i x*
shows *uniform-limit X f h F* ⟷ *uniform-limit X g i F*
proof −
{
fix *f g* :: *'a* ⇒ *'b* ⇒ *'c* **and** *h i* :: *'b* ⇒ *'c*
assume *C*: *uniform-limit X f h F* **and** *A*: *eventually* (λ*y*. ∀ *x*∈*X*. *f y x* = *g y x*) *F*
and *B*: ⋀*x*. *x* ∈ *X* ⟹ *h x* = *i x*
{
fix *e* ::*real* **assume** *e* > *0*
with *C* **have** *eventually* (λ*y*. ∀ *x*∈*X*. *dist* (*f y x*) (*h x*) < *e*) *F*
unfolding *uniform-limit-iff* **by** *blast*
with *A* **have** *eventually* (λ*y*. ∀ *x*∈*X*. *dist* (*g y x*) (*i x*) < *e*) *F*
by *eventually-elim* (*insert B*, *simp-all*)
}
hence *uniform-limit X g i F* **unfolding** *uniform-limit-iff* **by** *blast*
} **note** *A* = *this*
show *?thesis* **by** (*rule iffI*) (*erule A*; *insert assms*; *simp add*: *eq-commute*)+
qed

lemma *uniform-limit-cong'*:
fixes *f g* :: *'a* ⇒ *'b* ⇒ (*'c* :: *metric-space*) **and** *h i* :: *'b* ⇒ *'c*
assumes ⋀*y x*. *x* ∈ *X* ⟹ *f y x* = *g y x*
assumes ⋀*x*. *x* ∈ *X* ⟹ *h x* = *i x*
shows *uniform-limit X f h F* ⟷ *uniform-limit X g i F*
using *assms* **by** (*intro uniform-limit-cong always-eventually*) *blast*+

lemma *uniformly-convergent-uniform-limit-iff*:
uniformly-convergent-on X f ⟷ *uniform-limit X f* (λ*x*. *lim* (λ*n*. *f n x*)) *sequentially*
proof
assume *uniformly-convergent-on X f*
then obtain *l* **where** *l*: *uniform-limit X f l sequentially*
unfolding *uniformly-convergent-on-def* **by** *blast*
from *l* **have** *uniform-limit X f* (λ*x*. *lim* (λ*n*. *f n x*)) *sequentially* ⟷
uniform-limit X f l sequentially
by (*intro uniform-limit-cong*′ *limI tendsto-uniform-limitI*[*of f X l*]) *simp-all*
also note *l*
finally show *uniform-limit X f* (λ*x*. *lim* (λ*n*. *f n x*)) *sequentially* **.**
qed (*auto simp*: *uniformly-convergent-on-def*)

lemma *uniformly-convergentI*: *uniform-limit X f l sequentially* ⟹ *uniformly-convergent-on X f*
unfolding *uniformly-convergent-on-def* **by** *blast*

lemma *Cauchy-uniformly-convergent*:
fixes *f* :: *nat* ⇒ ′*a* ⇒ ′*b* :: *complete-space*
assumes *uniformly-Cauchy-on X f*
shows *uniformly-convergent-on X f*
unfolding *uniformly-convergent-uniform-limit-iff uniform-limit-iff*
proof *safe*
let *?f* = λ*x*. *lim* (λ*n*. *f n x*)
fix *e* :: *real* **assume** *e*: *e* > *0*
hence *e*/*2* > *0* **by** *simp*
with *assms* **obtain** *N* **where** *N*: ⋀*x m n*. *x* ∈ *X* ⟹ *m* ≥ *N* ⟹ *n* ≥ *N* ⟹ *dist* (*f m x*) (*f n x*) < *e*/*2*
unfolding *uniformly-Cauchy-on-def* **by** *fast*
show *eventually* (λ*n*. ∀*x*∈*X*. *dist* (*f n x*) (*?f x*) < *e*) *sequentially*
using *eventually-ge-at-top*[*of N*]
proof *eventually-elim*
fix *n* **assume** *n*: *n* ≥ *N*
show ∀*x*∈*X*. *dist* (*f n x*) (*?f x*) < *e*
proof
fix *x* **assume** *x*: *x* ∈ *X*
with *assms* **have** (λ*n*. *f n x*) ⟶ *?f x*
by (*auto dest*!: *Cauchy-convergent uniformly-Cauchy-imp-Cauchy simp*: *convergent-LIMSEQ-iff*)
with ‹*e*/*2* > *0*› **have** *eventually* (λ*m*. *m* ≥ *N* ∧ *dist* (*f m x*) (*?f x*) < *e*/*2*) *sequentially*
by (*intro tendstoD eventually-conj eventually-ge-at-top*)
then obtain *m* **where** *m*: *m* ≥ *N dist* (*f m x*) (*?f x*) < *e*/*2*
unfolding *eventually-at-top-linorder* **by** *blast*
have *dist* (*f n x*) (*?f x*) ≤ *dist* (*f n x*) (*f m x*) + *dist* (*f m x*) (*?f x*)
by (*rule dist-triangle*)
also from *x n* **have** ... < *e*/*2* + *e*/*2* **by** (*intro add-strict-mono N m*)
finally show *dist* (*f n x*) (*?f x*) < *e* **by** *simp*
qed

```
  qed
qed

lemma uniformly-convergent-imp-convergent:
  uniformly-convergent-on X f ⟹ x ∈ X ⟹ convergent (λn. f n x)
  unfolding uniformly-convergent-on-def convergent-def
  by (auto dest: tendsto-uniform-limitI)

lemma weierstrass-m-test-ev:
fixes f :: - ⇒ - ⇒ - :: banach
assumes eventually (λn. ∀x∈A. norm (f n x) ≤ M n) sequentially
assumes summable M
shows uniform-limit A (λn x. ∑i<n. f i x) (λx. suminf (λi. f i x)) sequentially
proof (rule uniform-limitI)
  fix e :: real
  assume 0 < e
  from suminf-exist-split[OF ‹0 < e› ‹summable M›]
  have ∀F k in sequentially. norm (∑i. M (i + k)) < e
    by (auto simp: eventually-sequentially)
  with eventually-all-ge-at-top[OF assms(1)]
    show ∀F n in sequentially. ∀x∈A. dist (∑i<n. f i x) (∑i. f i x) < e
  proof eventually-elim
    case (elim k)
    show ?case
    proof safe
      fix x assume x ∈ A
      have ∃N. ∀n≥N. norm (f n x) ≤ M n
        using assms(1) ‹x ∈ A› by (force simp: eventually-at-top-linorder)
      hence summable-norm-f: summable (λn. norm (f n x))
        by(rule summable-norm-comparison-test[OF - ‹summable M›])
      have summable-f: summable (λn. f n x)
        using summable-norm-cancel[OF summable-norm-f] .
      have summable-norm-f-plus-k: summable (λi. norm (f (i + k) x))
        using summable-ignore-initial-segment[OF summable-norm-f]
        by auto
      have summable-M-plus-k: summable (λi. M (i + k))
        using summable-ignore-initial-segment[OF ‹summable M›]
        by auto

      have dist (∑i<k. f i x) (∑i. f i x) = norm ((∑i. f i x) - (∑i<k. f i x))
        using dist-norm dist-commute by (subst dist-commute)
      also have ... = norm (∑i. f (i + k) x)
        using suminf-minus-initial-segment[OF summable-f, where k=k] by simp
      also have ... ≤ (∑i. norm (f (i + k) x))
        using summable-norm[OF summable-norm-f-plus-k] .
      also have ... ≤ (∑i. M (i + k))
        by (rule suminf-le[OF - summable-norm-f-plus-k summable-M-plus-k])
           (insert elim(1) ‹x ∈ A›, simp)
      finally show dist (∑i<k. f i x) (∑i. f i x) < e
```

using *elim* **by** *auto*
qed
qed
qed

Alternative version, formulated as in HOL Light

corollary *series-comparison-uniform*:
fixes f :: - ⇒ *nat* ⇒ - :: *banach*
assumes g: *summable* g **and** *le*: $\bigwedge n\ x.\ N \leq n \wedge x \in A \Longrightarrow norm(f\ x\ n) \leq g\ n$
shows $\exists l.\ \forall e.\ 0 < e \longrightarrow (\exists N.\ \forall n\ x.\ N \leq n \wedge x \in A \longrightarrow dist(setsum\ (f\ x)\ \{..<n\})\ (l\ x) < e)$
proof −
have *1*: $\forall_F\ n$ *in sequentially*. $\forall x{\in}A.\ norm\ (f\ x\ n) \leq g\ n$
using *le eventually-sequentially* **by** *auto*
show *?thesis*
apply (*rule-tac* $x{=}(\lambda x.\ \sum i.\ f\ x\ i)$ **in** *exI*)
apply (*metis* (*no-types*, *lifting*) *eventually-sequentially uniform-limitD* [*OF weierstrass-m-test-ev* [*OF 1 g*]])
done
qed

corollary *weierstrass-m-test*:
fixes f :: - ⇒ - ⇒ - :: *banach*
assumes $\bigwedge n\ x.\ x \in A \Longrightarrow norm\ (f\ n\ x) \leq M\ n$
assumes *summable M*
shows *uniform-limit A* $(\lambda n\ x.\ \sum i{<}n.\ f\ i\ x)$ $(\lambda x.\ suminf\ (\lambda i.\ f\ i\ x))$ *sequentially*
using *assms* **by** (*intro weierstrass-m-test-ev always-eventually*) *auto*

corollary *weierstrass-m-test′-ev*:
fixes f :: - ⇒ - ⇒ - :: *banach*
assumes *eventually* $(\lambda n.\ \forall x{\in}A.\ norm\ (f\ n\ x) \leq M\ n)$ *sequentially summable M*

shows *uniformly-convergent-on A* $(\lambda n\ x.\ \sum i{<}n.\ f\ i\ x)$
unfolding *uniformly-convergent-on-def* **by** (*rule exI*, *rule weierstrass-m-test-ev*[*OF assms*])

corollary *weierstrass-m-test′*:
fixes f :: - ⇒ - ⇒ - :: *banach*
assumes $\bigwedge n\ x.\ x \in A \Longrightarrow norm\ (f\ n\ x) \leq M\ n$ *summable M*
shows *uniformly-convergent-on A* $(\lambda n\ x.\ \sum i{<}n.\ f\ i\ x)$
unfolding *uniformly-convergent-on-def* **by** (*rule exI*, *rule weierstrass-m-test*[*OF assms*])

lemma *uniform-limit-eq-rhs*: *uniform-limit X f l F* $\Longrightarrow l = m \Longrightarrow$ *uniform-limit X f m F*
by *simp*

named-theorems *uniform-limit-intros introduction rules for uniform-limit*
setup ‹

```
  Global-Theory.add-thms-dynamic (@{binding uniform-limit-eq-intros},
    fn context =>
     Named-Theorems.get (Context.proof-of context) @{named-theorems uniform-limit-intros}
      |> map-filter (try (fn thm => @{thm uniform-limit-eq-rhs} OF [thm])))
›

lemma (in bounded-linear) uniform-limit[uniform-limit-intros]:
  assumes uniform-limit X g l F
  shows uniform-limit X (λa b. f (g a b)) (λa. f (l a)) F
proof (rule uniform-limitI)
  fix e::real
  from pos-bounded obtain K
    where K: ⋀x y. dist (f x) (f y) ≤ K * dist x y K > 0
    by (auto simp: ac-simps dist-norm diff[symmetric])
  assume 0 < e with ‹K > 0› have e / K > 0 by simp
  from uniform-limitD[OF assms this]
  show ∀F n in F. ∀x∈X. dist (f (g n x)) (f (l x)) < e
    by eventually-elim (metis le-less-trans mult.commute pos-less-divide-eq K)
qed

lemmas bounded-linear-uniform-limit-intros[uniform-limit-intros] =
  bounded-linear.uniform-limit[OF bounded-linear-Im]
  bounded-linear.uniform-limit[OF bounded-linear-Re]
  bounded-linear.uniform-limit[OF bounded-linear-cnj]
  bounded-linear.uniform-limit[OF bounded-linear-fst]
  bounded-linear.uniform-limit[OF bounded-linear-snd]
  bounded-linear.uniform-limit[OF bounded-linear-zero]
  bounded-linear.uniform-limit[OF bounded-linear-of-real]
  bounded-linear.uniform-limit[OF bounded-linear-inner-left]
  bounded-linear.uniform-limit[OF bounded-linear-inner-right]
  bounded-linear.uniform-limit[OF bounded-linear-divide]
  bounded-linear.uniform-limit[OF bounded-linear-scaleR-right]
  bounded-linear.uniform-limit[OF bounded-linear-mult-left]
  bounded-linear.uniform-limit[OF bounded-linear-mult-right]
  bounded-linear.uniform-limit[OF bounded-linear-scaleR-left]

lemmas uniform-limit-uminus[uniform-limit-intros] =
  bounded-linear.uniform-limit[OF bounded-linear-minus[OF bounded-linear-ident]]

lemma uniform-limit-add[uniform-limit-intros]:
  fixes f g::'a ⇒ 'b ⇒ 'c::real-normed-vector
  assumes uniform-limit X f l F
  assumes uniform-limit X g m F
  shows uniform-limit X (λa b. f a b + g a b) (λa. l a + m a) F
proof (rule uniform-limitI)
  fix e::real
  assume 0 < e
  hence 0 < e / 2 by simp
  from
```

```
    uniform-limitD[OF assms(1) this]
    uniform-limitD[OF assms(2) this]
  show ∀ F n in F. ∀ x∈X. dist (f n x + g n x) (l x + m x) < e
    by eventually-elim (simp add: dist-triangle-add-half)
qed

lemma uniform-limit-minus[uniform-limit-intros]:
  fixes f g::'a ⇒ 'b ⇒ 'c::real-normed-vector
  assumes uniform-limit X f l F
  assumes uniform-limit X g m F
  shows uniform-limit X (λa b. f a b − g a b) (λa. l a − m a) F
  unfolding diff-conv-add-uminus
  by (rule uniform-limit-intros assms)+

lemma (in bounded-bilinear) bounded-uniform-limit[uniform-limit-intros]:
  assumes uniform-limit X f l F
  assumes uniform-limit X g m F
  assumes bounded (m ' X)
  assumes bounded (l ' X)
  shows uniform-limit X (λa b. prod (f a b) (g a b)) (λa. prod (l a) (m a)) F
proof (rule uniform-limitI)
  fix e::real
  from pos-bounded obtain K where K:
    0 < K ⋀a b. norm (prod a b) ≤ norm a * norm b * K
    by auto
  hence sqrt (K*4) > 0 by simp

  from assms obtain Km Kl
  where Km: Km > 0 ⋀x. x ∈ X ⟹ norm (m x) ≤ Km
    and Kl: Kl > 0 ⋀x. x ∈ X ⟹ norm (l x) ≤ Kl
    by (auto simp: bounded-pos)
  hence K * Km * 4 > 0 K * Kl * 4 > 0
    using ‹K > 0›
    by simp-all
  assume 0 < e

  hence sqrt e > 0 by simp
  from uniform-limitD[OF assms(1) divide-pos-pos[OF this ‹sqrt (K*4) > 0›]]
    uniform-limitD[OF assms(2) divide-pos-pos[OF this ‹sqrt (K*4) > 0›]]
    uniform-limitD[OF assms(1) divide-pos-pos[OF ‹e > 0› ‹K * Km * 4 > 0›]]
    uniform-limitD[OF assms(2) divide-pos-pos[OF ‹e > 0› ‹K * Kl * 4 > 0›]]
  show ∀ F n in F. ∀ x∈X. dist (prod (f n x) (g n x)) (prod (l x) (m x)) < e
  proof eventually-elim
    case (elim n)
    show ?case
    proof safe
      fix x assume x ∈ X
      have dist (prod (f n x) (g n x)) (prod (l x) (m x)) ≤
        norm (prod (f n x − l x) (g n x − m x)) +
```

```
      norm (prod (f n x − l x) (m x)) +
      norm (prod (l x) (g n x − m x))
    by (auto simp: dist-norm prod-diff-prod intro: order-trans norm-triangle-ineq
add-mono)
   also note K(2)[of f n x − l x g n x − m x]
   also from elim(1)[THEN bspec, OF ‹- ∈ X›, unfolded dist-norm]
   have norm (f n x − l x) ≤ sqrt e / sqrt (K * 4)
    by simp
   also from elim(2)[THEN bspec, OF ‹- ∈ X›, unfolded dist-norm]
   have norm (g n x − m x) ≤ sqrt e / sqrt (K * 4)
    by simp
   also have sqrt e / sqrt (K * 4) * (sqrt e / sqrt (K * 4)) * K = e / 4
    using ‹K > 0› ‹e > 0› by auto
   also note K(2)[of f n x − l x m x]
   also note K(2)[of l x g n x − m x]
   also from elim(3)[THEN bspec, OF ‹- ∈ X›, unfolded dist-norm]
   have norm (f n x − l x) ≤ e / (K * Km * 4)
    by simp
   also from elim(4)[THEN bspec, OF ‹- ∈ X›, unfolded dist-norm]
   have norm (g n x − m x) ≤ e / (K * Kl * 4)
    by simp
   also note Kl(2)[OF ‹- ∈ X›]
   also note Km(2)[OF ‹- ∈ X›]
   also have e / (K * Km * 4) * Km * K = e / 4
    using ‹K > 0› ‹Km > 0› by simp
   also have  Kl * (e / (K * Kl * 4)) * K = e / 4
    using ‹K > 0› ‹Kl > 0› by simp
   also have e / 4 + e / 4 + e / 4 < e using ‹e > 0› by simp
   finally show dist (prod (f n x) (g n x)) (prod (l x) (m x)) < e
    using ‹K > 0› ‹Kl > 0› ‹Km > 0› ‹e > 0›
    by (simp add: algebra-simps mult-right-mono divide-right-mono)
  qed
 qed
qed

lemmas bounded-bilinear-bounded-uniform-limit-intros[uniform-limit-intros] =
 bounded-bilinear.bounded-uniform-limit[OF Inner-Product.bounded-bilinear-inner]
 bounded-bilinear.bounded-uniform-limit[OF Real-Vector-Spaces.bounded-bilinear-mult]
 bounded-bilinear.bounded-uniform-limit[OF Real-Vector-Spaces.bounded-bilinear-scaleR]

lemma metric-uniform-limit-imp-uniform-limit:
 assumes f: uniform-limit S f a F
 assumes le: eventually (λx. ∀ y∈S. dist (g x y) (b y) ≤ dist (f x y) (a y)) F
 shows uniform-limit S g b F
proof (rule uniform-limitI)
 fix e :: real assume 0 < e
 from uniform-limitD[OF f this] le
 show ∀ F x in F. ∀ y∈S. dist (g x y) (b y) < e
  by eventually-elim force
```

qed

lemma *uniform-limit-null-comparison*:
assumes $\forall_F\ x\ in\ F.\ \forall a{\in}S.\ norm\ (f\ x\ a) \leq g\ x\ a$
assumes *uniform-limit S g* (λ-. *0*) *F*
shows *uniform-limit S f* (λ-. *0*) *F*
using *assms*(*2*)
proof (*rule metric-uniform-limit-imp-uniform-limit*)
show $\forall_F\ x\ in\ F.\ \forall y{\in}S.\ dist\ (f\ x\ y)\ 0 \leq dist\ (g\ x\ y)\ 0$
using *assms*(*1*) **by** (*rule eventually-mono*) (*force simp add*: *dist-norm*)
qed

lemma *uniform-limit-on-union*:
uniform-limit I f g F ⟹ *uniform-limit J f g F* ⟹ *uniform-limit* $(I \cup J)$ *f g F*
by (*auto intro*!: *uniform-limitI dest*!: *uniform-limitD elim*: *eventually-elim2*)

lemma *uniform-limit-on-empty*:
uniform-limit {} *f g F*
by (*auto intro*!: *uniform-limitI*)

lemma *uniform-limit-on-UNION*:
assumes *finite S*
assumes $\bigwedge s.\ s \in S \Longrightarrow$ *uniform-limit* $(h\ s)$ *f g F*
shows *uniform-limit* (*UNION S h*) *f g F*
using *assms*
by *induct* (*auto intro*: *uniform-limit-on-empty uniform-limit-on-union*)

lemma *uniform-limit-on-Union*:
assumes *finite I*
assumes $\bigwedge J.\ J \in I \Longrightarrow$ *uniform-limit J f g F*
shows *uniform-limit* (*Union I*) *f g F*
by (*metis SUP-identity-eq assms uniform-limit-on-UNION*)

lemma *uniform-limit-on-subset*:
uniform-limit J f g F ⟹ $I \subseteq J$ ⟹ *uniform-limit I f g F*
by (*auto intro*!: *uniform-limitI dest*!: *uniform-limitD intro*: *eventually-mono*)

lemma *uniformly-convergent-add*:
uniformly-convergent-on A f ⟹ *uniformly-convergent-on A g*⟹
uniformly-convergent-on A (λ*k x*. *f k x* + *g k x* :: *'a* :: {*real-normed-algebra*})
unfolding *uniformly-convergent-on-def* **by** (*blast dest*: *uniform-limit-add*)

lemma *uniformly-convergent-minus*:
uniformly-convergent-on A f ⟹ *uniformly-convergent-on A g*⟹
uniformly-convergent-on A (λ*k x*. *f k x* − *g k x* :: *'a* :: {*real-normed-algebra*})
unfolding *uniformly-convergent-on-def* **by** (*blast dest*: *uniform-limit-minus*)

lemma *uniformly-convergent-mult*:
uniformly-convergent-on A f ⟹

uniformly-convergent-on A (λ*k x*. *c* ∗ *f k x* :: *'a* :: {*real-normed-algebra*})
unfolding *uniformly-convergent-on-def*
by (*blast dest*: *bounded-linear-uniform-limit-intros*(*13*))

end

33 Bounded Linear Function

theory *Bounded-Linear-Function*
imports
Topology-Euclidean-Space
Operator-Norm
begin

33.1 Intro rules for *bounded-linear*

named-theorems *bounded-linear-intros*

lemma *onorm-inner-left*:
assumes *bounded-linear r*
shows *onorm* (λ*x*. *r x* · *f*) ≤ *onorm r* ∗ *norm f*
proof (*rule onorm-bound*)
fix *x*
have *norm* (*r x* · *f*) ≤ *norm* (*r x*) ∗ *norm f*
by (*simp add*: *Cauchy-Schwarz-ineq2*)
also have ... ≤ *onorm r* ∗ *norm x* ∗ *norm f*
by (*intro mult-right-mono onorm assms norm-ge-zero*)
finally show *norm* (*r x* · *f*) ≤ *onorm r* ∗ *norm f* ∗ *norm x*
by (*simp add*: *ac-simps*)
qed (*intro mult-nonneg-nonneg norm-ge-zero onorm-pos-le assms*)

lemma *onorm-inner-right*:
assumes *bounded-linear r*
shows *onorm* (λ*x*. *f* · *r x*) ≤ *norm f* ∗ *onorm r*
apply (*subst inner-commute*)
apply (*rule onorm-inner-left*[*OF assms*, *THEN order-trans*])
apply *simp*
done

lemmas [*bounded-linear-intros*] =
bounded-linear-zero
bounded-linear-add
bounded-linear-const-mult
bounded-linear-mult-const
bounded-linear-scaleR-const
bounded-linear-const-scaleR
bounded-linear-ident
bounded-linear-setsum
bounded-linear-Pair

bounded-linear-sub
bounded-linear-fst-comp
bounded-linear-snd-comp
bounded-linear-inner-left-comp
bounded-linear-inner-right-comp

33.2 declaration of derivative/continuous/tendsto introduction rules for bounded linear functions

attribute-setup *bounded-linear* =
‹*Scan.succeed* (*Thm.declaration-attribute* (*fn thm* =>
fold (*fn* (*r*, *s*) => *Named-Theorems.add-thm s* (*thm RS r*))
[
(@{*thm bounded-linear.has-derivative*}, *Deriv.derivative-intros*),
(@{*thm bounded-linear.tendsto*}, *Topological-Spaces.tendsto-intros*),
(@{*thm bounded-linear.continuous*}, *Topological-Spaces.continuous-intros*),
(@{*thm bounded-linear.continuous-on*}, *Topological-Spaces.continuous-intros*),
(@{*thm bounded-linear.uniformly-continuous-on*}, *Topological-Spaces.continuous-intros*),
(@{*thm bounded-linear-compose*}, *Bounded-Linear-Function.bounded-linear-intros*)
]))›

attribute-setup *bounded-bilinear* =
‹*Scan.succeed* (*Thm.declaration-attribute* (*fn thm* =>
fold (*fn* (*r*, *s*) => *Named-Theorems.add-thm s* (*thm RS r*))
[
(@{*thm bounded-bilinear.FDERIV*}, *Deriv.derivative-intros*),
(@{*thm bounded-bilinear.tendsto*}, *Topological-Spaces.tendsto-intros*),
(@{*thm bounded-bilinear.continuous*}, *Topological-Spaces.continuous-intros*),
(@{*thm bounded-bilinear.continuous-on*}, *Topological-Spaces.continuous-intros*),
(@{*thm bounded-linear-compose*[*OF bounded-bilinear.bounded-linear-left*]},
Bounded-Linear-Function.bounded-linear-intros),
(@{*thm bounded-linear-compose*[*OF bounded-bilinear.bounded-linear-right*]},
Bounded-Linear-Function.bounded-linear-intros),
(@{*thm bounded-linear.uniformly-continuous-on*[*OF bounded-bilinear.bounded-linear-left*]},
Topological-Spaces.continuous-intros),
(@{*thm bounded-linear.uniformly-continuous-on*[*OF bounded-bilinear.bounded-linear-right*]},
Topological-Spaces.continuous-intros)
]))›

33.3 type of bounded linear functions

typedef (**overloaded**) ('*a*, '*b*) *blinfun* ((- ⇒$_L$ /-) [*22*, *21*] *21*) =
{*f*::'*a*::*real-normed-vector*⇒'*b*::*real-normed-vector*. *bounded-linear f*}
morphisms *blinfun-apply Blinfun*
by (*blast intro*: *bounded-linear-intros*)

declare [[*coercion*
blinfun-apply :: ('*a*::*real-normed-vector* ⇒$_L$ '*b*::*real-normed-vector*) ⇒ '*a* ⇒ '*b*]]

lemma *bounded-linear-blinfun-apply*[*bounded-linear-intros*]:
bounded-linear g $\Longrightarrow$ *bounded-linear* (λx. *blinfun-apply f* (*g x*))
by (*metis blinfun-apply mem-Collect-eq bounded-linear-compose*)

setup-lifting *type-definition-blinfun*

lemma *blinfun-eqI*: ($\bigwedge i$. *blinfun-apply x i* = *blinfun-apply y i*) $\Longrightarrow$ *x* = *y*
by *transfer auto*

lemma *bounded-linear-Blinfun-apply*: *bounded-linear f* $\Longrightarrow$ *blinfun-apply* (*Blinfun f*) = *f*
by (*auto simp*: *Blinfun-inverse*)

33.4 type class instantiations

instantiation *blinfun* :: (*real-normed-vector*, *real-normed-vector*) *real-normed-vector*
begin

lift-definition *norm-blinfun* :: *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *real* **is** *onorm* **.**

lift-definition *minus-blinfun* :: *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b*
is $\lambda f\ g\ x.\ f\ x - g\ x$
by (*rule bounded-linear-sub*)

definition *dist-blinfun* :: *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *real*
where *dist-blinfun a b* = *norm* (*a* − *b*)

definition [*code del*]:
(*uniformity* :: ((*'a* $\Rightarrow_L$ *'b*) × (*'a* $\Rightarrow_L$ *'b*)) *filter*) = (*INF e*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-blinfun* :: (*'a* $\Rightarrow_L$ *'b*) *set* $\Rightarrow$ *bool*
where [*code del*]: *open-blinfun S* = ($\forall\, x{\in}S.\ \forall_F\ (x', y)$ *in uniformity*. *x'* = *x* $\longrightarrow$ *y* $\in$ *S*)

lift-definition *uminus-blinfun* :: *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b* **is** $\lambda f\ x.\ -\ f\ x$
by (*rule bounded-linear-minus*)

lift-definition *zero-blinfun* :: *'a* $\Rightarrow_L$ *'b* **is** $\lambda x.\ 0$
by (*rule bounded-linear-zero*)

lift-definition *plus-blinfun* :: *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b*
is $\lambda f\ g\ x.\ f\ x + g\ x$
by (*metis bounded-linear-add*)

lift-definition *scaleR-blinfun*::*real* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b* **is** $\lambda r\ f\ x.\ r *_R f\ x$
by (*metis bounded-linear-compose bounded-linear-scaleR-right*)

definition *sgn-blinfun* :: *'a* $\Rightarrow_L$ *'b* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b*

where *sgn-blinfun x = scaleR (inverse (norm x)) x*

instance
apply *standard*
unfolding *dist-blinfun-def open-blinfun-def sgn-blinfun-def uniformity-blinfun-def*
apply (*rule refl* | (*transfer*, *force simp*: *onorm-triangle onorm-scaleR onorm-eq-0 algebra-simps*))+
done

end

declare *uniformity-Abort*[**where** *'a*=(*'a* :: *real-normed-vector*) ⇒$_L$ (*'b* :: *real-normed-vector*), *code*]

lemma *norm-blinfun-eqI*:
assumes *n* ≤ *norm* (*blinfun-apply f x*) / *norm x*
assumes ⋀*x*. *norm* (*blinfun-apply f x*) ≤ *n* ∗ *norm x*
assumes *0* ≤ *n*
shows *norm f* = *n*
by (*auto simp*: *norm-blinfun-def*
intro!: *antisym onorm-bound assms order-trans*[*OF - le-onorm*]
bounded-linear-intros)

lemma *norm-blinfun*: *norm* (*blinfun-apply f x*) ≤ *norm f* ∗ *norm x*
by *transfer* (*rule onorm*)

lemma *norm-blinfun-bound*: *0* ≤ *b* ⟹ (⋀*x*. *norm* (*blinfun-apply f x*) ≤ *b* ∗ *norm x*) ⟹ *norm f* ≤ *b*
by *transfer* (*rule onorm-bound*)

lemma *bounded-bilinear-blinfun-apply*[*bounded-bilinear*]: *bounded-bilinear blinfun-apply*
proof
fix *f g*::*'a* ⇒$_L$ *'b* **and** *a b*::*'a* **and** *r*::*real*
show (*f* + *g*) *a* = *f a* + *g a* (*r* ∗$_R$ *f*) *a* = *r* ∗$_R$ *f a*
by (*transfer*, *simp*)+
interpret *bounded-linear f* **for** *f*::*'a* ⇒$_L$ *'b*
by (*auto intro*!: *bounded-linear-intros*)
show *f* (*a* + *b*) = *f a* + *f b f* (*r* ∗$_R$ *a*) = *r* ∗$_R$ *f a*
by (*simp-all add*: *add scaleR*)
show ∃ *K*. ∀ *a b*. *norm* (*blinfun-apply a b*) ≤ *norm a* ∗ *norm b* ∗ *K*
by (*auto intro*!: *exI*[**where** *x=1*] *norm-blinfun*)
qed

interpretation *blinfun*: *bounded-bilinear blinfun-apply*
by (*rule bounded-bilinear-blinfun-apply*)

lemmas *bounded-linear-apply-blinfun*[*intro*, *simp*] = *blinfun.bounded-linear-left*

context *bounded-bilinear*
begin

named-theorems *bilinear-simps*

lemmas [*bilinear-simps*] =
add-left
add-right
diff-left
diff-right
minus-left
minus-right
scaleR-left
scaleR-right
zero-left
zero-right
setsum-left
setsum-right

end

instance *blinfun* :: (*banach*, *banach*) *banach*
proof
fix X::$nat \Rightarrow 'a \Rightarrow_L 'b$
assume *Cauchy X*
{
fix x::$'a$
{
fix x::$'a$
assume $norm\ x \leq 1$
have *Cauchy* $(\lambda n.\ X\ n\ x)$
proof (*rule CauchyI*)
fix e::*real*
assume $0 < e$
from *CauchyD*[*OF* ‹*Cauchy X*› ‹$0 < e$›] **obtain** M
where M: $\bigwedge m\ n.\ m \geq M \Longrightarrow n \geq M \Longrightarrow norm\ (X\ m - X\ n) < e$
by *auto*
show $\exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ norm\ (X\ m\ x - X\ n\ x) < e$
proof (*safe intro*!: *exI*[**where** $x{=}M$])
fix $m\ n$
assume *le*: $M \leq m\ M \leq n$
have $norm\ (X\ m\ x - X\ n\ x) = norm\ ((X\ m - X\ n)\ x)$
by (*simp add*: *blinfun.bilinear-simps*)
also have $\ldots \leq norm\ (X\ m - X\ n) * norm\ x$
by (*rule norm-blinfun*)
also have $\ldots \leq norm\ (X\ m - X\ n) * 1$
using ‹$norm\ x \leq 1$› *norm-ge-zero* **by** (*rule mult-left-mono*)
also have $\ldots = norm\ (X\ m - X\ n)$ **by** *simp*

```
      also have ... < e using le by fact
      finally show norm (X m x − X n x) < e .
    qed
  qed
  hence convergent (λn. X n x)
    by (metis Cauchy-convergent-iff)
 } note convergent-norm1 = this
 def y ≡ x /R norm x
 have y: norm y ≤ 1 and xy: x = norm x *R y
  by (simp-all add: y-def inverse-eq-divide)
 have convergent (λn. norm x *R X n y)
 by (intro bounded-bilinear.convergent[OF bounded-bilinear-scaleR] convergent-const
    convergent-norm1 y)
 also have (λn. norm x *R X n y) = (λn. X n x)
  by (subst xy) (simp add: blinfun.bilinear-simps)
 finally have convergent (λn. X n x) .
}
then obtain v where v: ⋀x. (λn. X n x) ⟶ v x
 unfolding convergent-def
 by metis

have Cauchy (λn. norm (X n))
proof (rule CauchyI)
 fix e::real
 assume e > 0
 from CauchyD[OF ‹Cauchy X› ‹0 < e›] obtain M
  where M: ⋀m n. m ≥ M ⟹ n ≥ M ⟹ norm (X m − X n) < e
  by auto
 show ∃ M. ∀ m≥M. ∀ n≥M. norm (norm (X m) − norm (X n)) < e
 proof (safe intro!: exI[where x=M])
  fix m n assume mn: m ≥ M n ≥ M
  have norm (norm (X m) − norm (X n)) ≤ norm (X m − X n)
   by (metis norm-triangle-ineq3 real-norm-def)
  also have ... < e using mn by fact
  finally show norm (norm (X m) − norm (X n)) < e .
 qed
qed
then obtain K where K: (λn. norm (X n)) ⟶ K
 unfolding Cauchy-convergent-iff convergent-def
 by metis

have bounded-linear v
proof
 fix x y and r::real
 from tendsto-add[OF v[of x] v [of y]] v[of x + y, unfolded blinfun.bilinear-simps]
    tendsto-scaleR[OF tendsto-const[of r] v[of x]] v[of r *R x, unfolded blin-
fun.bilinear-simps]
 show v (x + y) = v x + v y v (r *R x) = r *R v x
  by (metis (poly-guards-query) LIMSEQ-unique)+
```

```
    show ∃K. ∀x. norm (v x) ≤ norm x * K
    proof (safe intro!: exI[where x=K])
      fix x
      have norm (v x) ≤ K * norm x
       by (rule tendsto-le[OF - tendsto-mult[OF K tendsto-const] tendsto-norm[OF
v]])
          (auto simp: norm-blinfun)
      thus norm (v x) ≤ norm x * K
        by (simp add: ac-simps)
    qed
  qed
  hence Bv: ⋀x. (λn. X n x) ⟶ Blinfun v x
    by (auto simp: bounded-linear-Blinfun-apply v)

  have X ⟶ Blinfun v
  proof (rule LIMSEQ-I)
    fix r::real assume r > 0
    def r' ≡ r / 2
    have 0 < r' r' < r using ‹r > 0› by (simp-all add: r'-def)
    from CauchyD[OF ‹Cauchy X› ‹r' > 0›]
    obtain M where M: ⋀m n. m ≥ M ⟹ n ≥ M ⟹ norm (X m - X n) <
r'
      by metis
    show ∃no. ∀n≥no. norm (X n - Blinfun v) < r
    proof (safe intro!: exI[where x=M])
      fix n assume n: M ≤ n
      have norm (X n - Blinfun v) ≤ r'
      proof (rule norm-blinfun-bound)
        fix x
        have eventually (λm. m ≥ M) sequentially
          by (metis eventually-ge-at-top)
          hence ev-le: eventually (λm. norm (X n x - X m x) ≤ r' * norm x)
sequentially
        proof eventually-elim
          case (elim m)
          have norm (X n x - X m x) = norm ((X n - X m) x)
            by (simp add: blinfun.bilinear-simps)
          also have ... ≤ norm ((X n - X m)) * norm x
            by (rule norm-blinfun)
          also have ... ≤ r' * norm x
            using M[OF n elim] by (simp add: mult-right-mono)
          finally show ?case .
        qed
          have tendsto-v: (λm. norm (X n x - X m x)) ⟶ norm (X n x -
Blinfun v x)
          by (auto intro!: tendsto-intros Bv)
        show norm ((X n - Blinfun v) x) ≤ r' * norm x
        by (auto intro!: tendsto-ge-const tendsto-v ev-le simp: blinfun.bilinear-simps)
      qed (simp add: ‹0 < r'› less-imp-le)
```

thus *norm* (*X n* − *Blinfun v*) < *r*
by (*metis* ‹*r'* < *r*› *le-less-trans*)
qed
qed
thus *convergent X*
by (*rule convergentI*)
qed

33.5 On Euclidean Space

lemma *Zfun-setsum*:
assumes *finite s*
assumes *f*: ⋀*i*. *i* ∈ *s* ⟹ *Zfun* (*f i*) *F*
shows *Zfun* (λ*x*. *setsum* (λ*i*. *f i x*) *s*) *F*
using *assms* **by** *induct* (*auto intro*!: *Zfun-zero Zfun-add*)

lemma *norm-blinfun-euclidean-le*:
fixes *a*::'*a*::*euclidean-space* ⇒$_L$ 'b::*real-normed-vector*
shows *norm a* ≤ *setsum* (λ*x*. *norm* (*a x*)) *Basis*
apply (*rule norm-blinfun-bound*)
apply (*simp add*: *setsum-nonneg*)
apply (*subst euclidean-representation*[*symmetric*, **where** 'a='*a*])
apply (*simp only*: *blinfun.bilinear-simps setsum-left-distrib*)
apply (*rule order.trans*[*OF norm-setsum setsum-mono*])
apply (*simp add*: *abs-mult mult-right-mono ac-simps Basis-le-norm*)
done

lemma *tendsto-componentwise1*:
fixes *a*::'*a*::*euclidean-space* ⇒$_L$ 'b::*real-normed-vector*
and *b*::'*c* ⇒ '*a* ⇒$_L$ '*b*
assumes (⋀*j*. *j* ∈ *Basis* ⟹ ((λ*n*. *b n j*) ⟶ *a j*) *F*)
shows (*b* ⟶ *a*) *F*
proof −
have ⋀*j*. *j* ∈ *Basis* ⟹ *Zfun* (λ*x*. *norm* (*b x j* − *a j*)) *F*
using *assms* **unfolding** *tendsto-Zfun-iff Zfun-norm-iff* .
hence *Zfun* (λ*x*. ∑*j*∈*Basis*. *norm* (*b x j* − *a j*)) *F*
by (*auto intro*!: *Zfun-setsum*)
thus *?thesis*
unfolding *tendsto-Zfun-iff*
by (*rule Zfun-le*)
(*auto intro*!: *order-trans*[*OF norm-blinfun-euclidean-le*] *simp*: *blinfun.bilinear-simps*)
qed

lift-definition
blinfun-of-matrix::('*b*::*euclidean-space* ⇒ '*a*::*euclidean-space* ⇒ *real*) ⇒ '*a* ⇒$_L$ '*b*
is λ*a x*. ∑*i*∈*Basis*. ∑*j*∈*Basis*. ((*x* ∙ *j*) ∗ *a i j*) ∗$_R$ *i*
by (*intro bounded-linear-intros*)

lemma *blinfun-of-matrix-works*:

fixes f::$'a$::*euclidean-space* $\Rightarrow_L$ $'b$::*euclidean-space*
shows *blinfun-of-matrix* $(\lambda i\ j.\ (f\ j) \cdot i) = f$
proof (*transfer*, *rule*, *rule euclidean-eqI*)
fix f::$'a \Rightarrow 'b$ **and** x::$'a$ **and** b::$'b$ **assume** *bounded-linear* f **and** b: $b \in Basis$
then interpret *bounded-linear* f **by** *simp*
have $(\sum j{\in}Basis.\ \sum i{\in}Basis.\ (x \cdot i * (f\ i \cdot j)) *_R j) \cdot b$
$= (\sum j{\in}Basis.\ if\ j = b\ then\ (\sum i{\in}Basis.\ (x \cdot i * (f\ i \cdot j)))\ else\ 0)$
using b
by (*auto simp add*: *algebra-simps inner-setsum-left inner-Basis split*: *split-if intro*!: *setsum.cong*)
also have $\ldots = (\sum i{\in}Basis.\ (x \cdot i * (f\ i \cdot b)))$
using b **by** (*simp add*: *setsum.delta*)
also have $\ldots = f\ x \cdot b$
by (*subst linear-componentwise*[*symmetric*]) (*unfold-locales*, *rule*)
finally show $(\sum j{\in}Basis.\ \sum i{\in}Basis.\ (x \cdot i * (f\ i \cdot j)) *_R j) \cdot b = f\ x \cdot b$.
qed

lemma *blinfun-of-matrix-apply*:
blinfun-of-matrix $a\ x = (\sum i{\in}Basis.\ \sum j{\in}Basis.\ ((x \cdot j) * a\ i\ j) *_R i)$
by *transfer simp*

lemma *blinfun-of-matrix-minus*: *blinfun-of-matrix* x − *blinfun-of-matrix* y = *blinfun-of-matrix* $(x - y)$
by *transfer* (*auto simp*: *algebra-simps setsum-subtractf*)

lemma *norm-blinfun-of-matrix*:
norm (*blinfun-of-matrix* a) $\leq (\sum i{\in}Basis.\ \sum j{\in}Basis.\ |a\ i\ j|)$
apply (*rule norm-blinfun-bound*)
apply (*simp add*: *setsum-nonneg*)
apply (*simp only*: *blinfun-of-matrix-apply setsum-left-distrib*)
apply (*rule order-trans*[*OF norm-setsum setsum-mono*])
apply (*rule order-trans*[*OF norm-setsum setsum-mono*])
apply (*simp add*: *abs-mult mult-right-mono ac-simps Basis-le-norm*)
done

lemma *tendsto-blinfun-of-matrix*:
assumes $\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow ((\lambda n.\ b\ n\ i\ j) \longrightarrow a\ i\ j)\ F$
shows $((\lambda n.\ blinfun\text{-}of\text{-}matrix\ (b\ n)) \longrightarrow blinfun\text{-}of\text{-}matrix\ a)\ F$
proof −
have $\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow Zfun\ (\lambda x.\ norm\ (b\ x\ i\ j - a\ i\ j))\ F$
using *assms* **unfolding** *tendsto-Zfun-iff Zfun-norm-iff* .
hence *Zfun* $(\lambda x.\ (\sum i{\in}Basis.\ \sum j{\in}Basis.\ |b\ x\ i\ j - a\ i\ j|))\ F$
by (*auto intro*!: *Zfun-setsum*)
thus *?thesis*
unfolding *tendsto-Zfun-iff blinfun-of-matrix-minus*
by (*rule Zfun-le*) (*auto intro*!: *order-trans*[*OF norm-blinfun-of-matrix*])
qed

lemma *tendsto-componentwise*:

fixes $a::'a::euclidean\text{-}space \Rightarrow_L 'b::euclidean\text{-}space$
and $b::'c \Rightarrow 'a \Rightarrow_L 'b$
shows $(\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow ((\lambda n.\ b\ n\ j \cdot i) \longrightarrow a\ j \cdot i)\ F) \Longrightarrow (b \longrightarrow a)\ F$
apply (*subst blinfun-of-matrix-works*[*of a*, *symmetric*])
apply (*subst blinfun-of-matrix-works*[*of b x* **for** *x*, *symmetric*, *abs-def*])
by (*rule tendsto-blinfun-of-matrix*)

lemma
continuous-blinfun-componentwiseI:
fixes $f::'b::t2\text{-}space \Rightarrow 'a::euclidean\text{-}space \Rightarrow_L 'c::euclidean\text{-}space$
assumes $\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow continuous\ F\ (\lambda x.\ (f\ x)\ j \cdot i)$
shows *continuous F f*
using *assms* **by** (*auto simp*: *continuous-def intro*!: *tendsto-componentwise*)

lemma
continuous-blinfun-componentwiseI1:
fixes $f::'b::t2\text{-}space \Rightarrow 'a::euclidean\text{-}space \Rightarrow_L 'c::real\text{-}normed\text{-}vector$
assumes $\bigwedge i.\ i \in Basis \Longrightarrow continuous\ F\ (\lambda x.\ f\ x\ i)$
shows *continuous F f*
using *assms* **by** (*auto simp*: *continuous-def intro*!: *tendsto-componentwise1*)

lemma *bounded-linear-blinfun-matrix*: *bounded-linear* $(\lambda x.\ (x::-\Rightarrow_L -)\ j \cdot i)$
by (*auto intro*!: *bounded-linearI' bounded-linear-intros*)

lemma *continuous-blinfun-matrix*:
fixes $f::'b::t2\text{-}space \Rightarrow 'a::real\text{-}normed\text{-}vector \Rightarrow_L 'c::real\text{-}inner$
assumes *continuous F f*
shows *continuous F* $(\lambda x.\ (f\ x)\ j \cdot i)$
by (*rule bounded-linear.continuous*[*OF bounded-linear-blinfun-matrix assms*])

lemma *continuous-on-blinfun-matrix*:
fixes $f::'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}vector \Rightarrow_L 'c::real\text{-}inner$
assumes *continuous-on S f*
shows *continuous-on S* $(\lambda x.\ (f\ x)\ j \cdot i)$
using *assms*
by (*auto simp*: *continuous-on-eq-continuous-within continuous-blinfun-matrix*)

lemma *mult-if-delta*:
$(if\ P\ then\ (1::'a::comm\text{-}semiring\text{-}1)\ else\ 0) * q = (if\ P\ then\ q\ else\ 0)$
by *auto*

lemma *compact-blinfun-lemma*:
fixes $f :: nat \Rightarrow 'a::euclidean\text{-}space \Rightarrow_L 'b::euclidean\text{-}space$
assumes *bounded* (*range f*)
shows $\forall\, d{\subseteq}Basis.\ \exists\, l::'a \Rightarrow_L 'b.\ \exists\ r.$
subseq r $\wedge$ $(\forall\, e{>}0.\ eventually\ (\lambda n.\ \forall\, i{\in}d.\ dist\ (f\ (r\ n)\ i)\ (l\ i) < e)\ sequentially)$
by (*rule compact-lemma-general*[**where** *unproj* $= \lambda e.\ blinfun\text{-}of\text{-}matrix\ (\lambda i\ j.\ e\ j \cdot i)$])

(*auto intro*!: *euclidean-eqI*[**where** *'a='b*] *bounded-linear-image assms*
simp: *blinfun-of-matrix-works blinfun-of-matrix-apply inner-Basis mult-if-delta setsum.delta'*
scaleR-setsum-left[*symmetric*])

lemma *blinfun-euclidean-eqI*: ($\bigwedge i.\ i \in Basis \Longrightarrow blinfun\text{-}apply\ x\ i = blinfun\text{-}apply\ y\ i$) $\Longrightarrow x = y$
apply (*auto intro*!: *blinfun-eqI*)
apply (*subst* (*2*) *euclidean-representation*[*symmetric*, **where** *'a='a*])
apply (*subst* (*1*) *euclidean-representation*[*symmetric*, **where** *'a='a*])
apply (*simp add*: *blinfun.bilinear-simps*)
done

TODO: generalize (via *compact* (*cball ?x ?e*))?

instance *blinfun* :: (*euclidean-space*, *euclidean-space*) *heine-borel*
proof
fix *f* :: *nat* ⇒ *'a* ⇒$_L$ *'b*
assume *f*: *bounded* (*range f*)
then obtain *l*::*'a* ⇒$_L$ *'b* **and** *r* **where** *r*: *subseq r*
and *l*: ∀ *e*>*0*. *eventually* (λ*n*. ∀ *i*∈*Basis*. *dist* (*f* (*r n*) *i*) (*l i*) < *e*) *sequentially*
using *compact-blinfun-lemma* [*OF f*] **by** *blast*
{
fix *e*::*real*
let *?d* = *real-of-nat DIM*(*'a*) ∗ *real-of-nat DIM*(*'b*)
assume *e* > *0*
hence *e* / *?d* > *0* **by** (*simp add*: *DIM-positive*)
with *l* **have** *eventually* (λ*n*. ∀ *i*∈*Basis*. *dist* (*f* (*r n*) *i*) (*l i*) < *e* / *?d*) *sequentially*
by *simp*
moreover
{
fix *n*
assume *n*: ∀ *i*∈*Basis*. *dist* (*f* (*r n*) *i*) (*l i*) < *e* / *?d*
have *norm* (*f* (*r n*) − *l*) = *norm* (*blinfun-of-matrix* (λ*i j*. (*f* (*r n*) − *l*) *j* • *i*))
unfolding *blinfun-of-matrix-works* **..**
also note *norm-blinfun-of-matrix*
also have (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. |(*f* (*r n*) − *l*) *j* • *i*|) <
(∑ *i*∈(*Basis*::*'b set*). *e* / *real-of-nat DIM*(*'b*))
proof (*rule setsum-strict-mono*)
fix *i*::*'b* **assume** *i*: *i* ∈ *Basis*
have (∑ *j*::*'a*∈*Basis*. |(*f* (*r n*) − *l*) *j* • *i*|) < (∑ *j*::*'a*∈*Basis*. *e* / *?d*)
proof (*rule setsum-strict-mono*)
fix *j*::*'a* **assume** *j*: *j* ∈ *Basis*
have |(*f* (*r n*) − *l*) *j* • *i*| ≤ *norm* ((*f* (*r n*) − *l*) *j*)
by (*simp add*: *Basis-le-norm i*)
also have ... < *e* / *?d*
using *n i j* **by** (*auto simp*: *dist-norm blinfun.bilinear-simps*)
finally show |(*f* (*r n*) − *l*) *j* • *i*| < *e* / *?d* **by** *simp*

qed *simp-all*
also have $\ldots \leq e$ / *real-of-nat DIM*('*b*)
by *simp*
finally show $(\sum j{\in}Basis.\ |(f\ (r\ n) - l)\ j \cdot i|) < e$ / *real-of-nat DIM*('*b*)
by *simp*
qed *simp-all*
also have $\ldots \leq e$ **by** *simp*
finally have *dist* $(f\ (r\ n))\ l < e$
by (*auto simp*: *dist-norm*)
}
ultimately have *eventually* $(\lambda n.\ dist\ (f\ (r\ n))\ l < e)$ *sequentially*
using *eventually-elim2* **by** *force*
}
then have $*$: $((f \circ r) \longrightarrow l)$ *sequentially*
unfolding *o-def tendsto-iff* **by** *simp*
with *r* **show** $\exists l\ r.$ *subseq* $r \wedge ((f \circ r) \longrightarrow l)$ *sequentially*
by *auto*
qed

33.6 concrete bounded linear functions

lemma *transfer-bounded-bilinear-bounded-linearI*:
assumes $g = (\lambda i\ x.\ (blinfun\text{-}apply\ (f\ i)\ x))$
shows *bounded-bilinear g* = *bounded-linear f*
proof
assume *bounded-bilinear g*
then interpret *bounded-bilinear f* **by** (*simp add*: *assms*)
show *bounded-linear f*
proof (*unfold-locales*, *safe intro*!: *blinfun-eqI*)
fix *i*
show $f\ (x + y)\ i = (f\ x + f\ y)\ i\ f\ (r *_R x)\ i = (r *_R f\ x)\ i$ **for** *r x y*
by (*auto intro*!: *blinfun-eqI simp*: *blinfun.bilinear-simps*)
from - *nonneg-bounded* **show** $\exists K.\ \forall x.\ norm\ (f\ x) \leq norm\ x * K$
by (*rule ex-reg*) (*auto intro*!: *onorm-bound simp*: *norm-blinfun.rep-eq ac-simps*)
qed
qed (*auto simp*: *assms intro*!: *blinfun.comp*)

lemma *transfer-bounded-bilinear-bounded-linear*[*transfer-rule*]:
(*rel-fun* (*rel-fun op* = (*pcr-blinfun op* = *op* =)) *op* =) *bounded-bilinear bounded-linear*
by (*auto simp*: *pcr-blinfun-def cr-blinfun-def rel-fun-def OO-def*
intro!: *transfer-bounded-bilinear-bounded-linearI*)

context *bounded-bilinear*
begin

lift-definition *prod-left*::'*b* ⇒ '*a* ⇒$_L$ '*c* **is** $(\lambda b\ a.\ prod\ a\ b)$
by (*rule bounded-linear-left*)
declare *prod-left.rep-eq*[*simp*]

lemma *bounded-linear-prod-left*[*bounded-linear*]: *bounded-linear prod-left*
by *transfer* (*rule flip*)

lift-definition *prod-right*::$'a \Rightarrow 'b \Rightarrow_L 'c$ **is** $(\lambda a\ b.\ prod\ a\ b)$
by (*rule bounded-linear-right*)
declare *prod-right.rep-eq*[*simp*]

lemma *bounded-linear-prod-right*[*bounded-linear*]: *bounded-linear prod-right*
by *transfer* (*rule bounded-bilinear-axioms*)

end

lift-definition *id-blinfun*::$'a$::*real-normed-vector* $\Rightarrow_L 'a$ **is** $\lambda x.\ x$
by (*rule bounded-linear-ident*)

lemmas *blinfun-apply-id-blinfun*[*simp*] = *id-blinfun.rep-eq*

lemma *norm-blinfun-id*[*simp*]:
norm (*id-blinfun*::$'a$::{*real-normed-vector*, *perfect-space*} $\Rightarrow_L 'a$) = *1*
by *transfer* (*auto simp*: *onorm-id*)

lemma *norm-blinfun-id-le*:
norm (*id-blinfun*::$'a$::*real-normed-vector* $\Rightarrow_L 'a$) $\leq$ *1*
by *transfer* (*auto simp*: *onorm-id-le*)

lift-definition *fst-blinfun*::($'a$::*real-normed-vector* $\times$ $'b$::*real-normed-vector*) $\Rightarrow_L 'a$
is *fst*
by (*rule bounded-linear-fst*)

lemma *blinfun-apply-fst-blinfun*[*simp*]: *blinfun-apply fst-blinfun* = *fst*
by *transfer* (*rule refl*)

lift-definition *snd-blinfun*::($'a$::*real-normed-vector* $\times$ $'b$::*real-normed-vector*) $\Rightarrow_L$
$'b$ **is** *snd*
by (*rule bounded-linear-snd*)

lemma *blinfun-apply-snd-blinfun*[*simp*]: *blinfun-apply snd-blinfun* = *snd*
by *transfer* (*rule refl*)

lift-definition *blinfun-compose*::
$'a$::*real-normed-vector* $\Rightarrow_L$ $'b$::*real-normed-vector* $\Rightarrow$
$'c$::*real-normed-vector* $\Rightarrow_L 'a \Rightarrow$
$'c \Rightarrow_L 'b$ (**infixl** o_L *55*) **is** *op o*
parametric *comp-transfer*
unfolding *o-def*
by (*rule bounded-linear-compose*)

lemma *blinfun-apply-blinfun-compose*[*simp*]: $(a\ o_L\ b)\ c = a\ (b\ c)$
by (*simp add*: *blinfun-compose.rep-eq*)

lemma *norm-blinfun-compose*:
$norm\ (f\ o_L\ g) \leq norm\ f * norm\ g$
by *transfer* (*rule onorm-compose*)

lemma *bounded-bilinear-blinfun-compose*[*bounded-bilinear*]: *bounded-bilinear op* o_L
by *unfold-locales*
(*auto intro*!: *blinfun-eqI exI*[**where** *x=1*] *simp*: *blinfun.bilinear-simps norm-blinfun-compose*)

lift-definition *blinfun-inner-right*::'*a*::*real-inner* ⇒ '*a* ⇒$_L$ *real* **is** *op* ·
by (*rule bounded-linear-inner-right*)
declare *blinfun-inner-right.rep-eq*[*simp*]

lemma *bounded-linear-blinfun-inner-right*[*bounded-linear*]: *bounded-linear blinfun-inner-right*
by *transfer* (*rule bounded-bilinear-inner*)

lift-definition *blinfun-inner-left*::'*a*::*real-inner* ⇒ '*a* ⇒$_L$ *real* **is** $\lambda x\ y.\ y \cdot x$
by (*rule bounded-linear-inner-left*)
declare *blinfun-inner-left.rep-eq*[*simp*]

lemma *bounded-linear-blinfun-inner-left*[*bounded-linear*]: *bounded-linear blinfun-inner-left*
by *transfer* (*rule bounded-bilinear.flip*[*OF bounded-bilinear-inner*])

lift-definition *blinfun-scaleR-right*::*real* ⇒ '*a* ⇒$_L$ '*a*::*real-normed-vector* **is** *op* $*_R$
by (*rule bounded-linear-scaleR-right*)
declare *blinfun-scaleR-right.rep-eq*[*simp*]

lemma *bounded-linear-blinfun-scaleR-right*[*bounded-linear*]: *bounded-linear blinfun-scaleR-right*
by *transfer* (*rule bounded-bilinear-scaleR*)

lift-definition *blinfun-scaleR-left*::'*a*::*real-normed-vector* ⇒ *real* ⇒$_L$ '*a* **is** $\lambda x\ y.\ y$
$*_R\ x$
by (*rule bounded-linear-scaleR-left*)
lemmas [*simp*] = *blinfun-scaleR-left.rep-eq*

lemma *bounded-linear-blinfun-scaleR-left*[*bounded-linear*]: *bounded-linear blinfun-scaleR-left*
by *transfer* (*rule bounded-bilinear.flip*[*OF bounded-bilinear-scaleR*])

lift-definition *blinfun-mult-right*::'*a* ⇒ '*a* ⇒$_L$ '*a*::*real-normed-algebra* **is** *op* $*$
by (*rule bounded-linear-mult-right*)
declare *blinfun-mult-right.rep-eq*[*simp*]

lemma *bounded-linear-blinfun-mult-right*[*bounded-linear*]: *bounded-linear blinfun-mult-right*
by *transfer* (*rule bounded-bilinear-mult*)

lift-definition *blinfun-mult-left*::'*a*::*real-normed-algebra* ⇒ '*a* ⇒$_L$ '*a* **is** λ*x y*. *y* ∗ *x*
by (*rule bounded-linear-mult-left*)
lemmas [*simp*] = *blinfun-mult-left.rep-eq*

lemma *bounded-linear-blinfun-mult-left*[*bounded-linear*]: *bounded-linear blinfun-mult-left*
by *transfer* (*rule bounded-bilinear.flip*[*OF bounded-bilinear-mult*])

end

34 Multivariate calculus in Euclidean space

theory *Derivative*
imports *Brouwer-Fixpoint Operator-Norm Uniform-Limit Bounded-Linear-Function*
begin

lemma *onorm-inner-left*:
assumes *bounded-linear r*
shows *onorm* (λ*x*. *r x* • *f*) ≤ *onorm r* ∗ *norm f*
proof (*rule onorm-bound*)
fix *x*
have *norm* (*r x* • *f*) ≤ *norm* (*r x*) ∗ *norm f*
by (*simp add*: *Cauchy-Schwarz-ineq2*)
also have ... ≤ *onorm r* ∗ *norm x* ∗ *norm f*
by (*intro mult-right-mono onorm assms norm-ge-zero*)
finally show *norm* (*r x* • *f*) ≤ *onorm r* ∗ *norm f* ∗ *norm x*
by (*simp add*: *ac-simps*)
qed (*intro mult-nonneg-nonneg norm-ge-zero onorm-pos-le assms*)

lemma *onorm-inner-right*:
assumes *bounded-linear r*
shows *onorm* (λ*x*. *f* • *r x*) ≤ *norm f* ∗ *onorm r*
apply (*subst inner-commute*)
apply (*rule onorm-inner-left*[*OF assms*, *THEN order-trans*])
apply *simp*
done

declare *has-derivative-bounded-linear*[*dest*]

34.1 Derivatives

34.1.1 Combining theorems.

lemmas *has-derivative-id* = *has-derivative-ident*

lemmas *has-derivative-neg = has-derivative-minus*
lemmas *has-derivative-sub = has-derivative-diff*
lemmas *scaleR-right-has-derivative = has-derivative-scaleR-right*
lemmas *scaleR-left-has-derivative = has-derivative-scaleR-left*
lemmas *inner-right-has-derivative = has-derivative-inner-right*
lemmas *inner-left-has-derivative = has-derivative-inner-left*
lemmas *mult-right-has-derivative = has-derivative-mult-right*
lemmas *mult-left-has-derivative = has-derivative-mult-left*

lemma *has-derivative-add-const*:
(*f has-derivative f'*) *net* $\Longrightarrow$ ((λ*x*. *f x* + *c*) *has-derivative f'*) *net*
by (*intro derivative-eq-intros*) *auto*

34.2 Derivative with composed bilinear function.

lemma *has-derivative-bilinear-within*:
assumes (*f has-derivative f'*) (*at x within s*)
and (*g has-derivative g'*) (*at x within s*)
and *bounded-bilinear h*
shows ((λ*x*. *h* (*f x*) (*g x*)) *has-derivative* (λ*d*. *h* (*f x*) (*g' d*) + *h* (*f' d*) (*g x*)))
(*at x within s*)
using *bounded-bilinear.FDERIV*[*OF assms*(*3*,*1*,*2*)] .

lemma *has-derivative-bilinear-at*:
assumes (*f has-derivative f'*) (*at x*)
and (*g has-derivative g'*) (*at x*)
and *bounded-bilinear h*
shows ((λ*x*. *h* (*f x*) (*g x*)) *has-derivative* (λ*d*. *h* (*f x*) (*g' d*) + *h* (*f' d*) (*g x*)))
(*at x*)
using *has-derivative-bilinear-within*[*of f f' x UNIV g g' h*] *assms* **by** *simp*

These are the only cases we'll care about, probably.

lemma *has-derivative-within*: (*f has-derivative f'*) (*at x within s*) ⟷
bounded-linear f' ∧ ((λ*y*. (*1* / *norm*(*y* − *x*)) *$*$_R$ (*f y* − (*f x* + *f'* (*y* − *x*))))
⟶ *0*) (*at x within s*)
unfolding *has-derivative-def Lim*
by (*auto simp add*: *netlimit-within field-simps*)

lemma *has-derivative-at*: (*f has-derivative f'*) (*at x*) ⟷
bounded-linear f' ∧ ((λ*y*. (*1* / (*norm*(*y* − *x*))) *$*$_R$ (*f y* − (*f x* + *f'* (*y* − *x*))))
⟶ *0*) (*at x*)
using *has-derivative-within* [*of f f' x UNIV*]
by *simp*

More explicit epsilon-delta forms.

lemma *has-derivative-within'*:
(*f has-derivative f'*)(*at x within s*) ⟷
bounded-linear f' ∧
(∀ *e*>*0*. ∃ *d*>*0*. ∀ *x'*∈*s*. *0* < *norm* (*x'* − *x*) ∧ *norm* (*x'* − *x*) < *d* ⟶

norm $(f\,x' - f\,x - f'(x' - x)) \,/\, norm\ (x' - x) < e$)
unfolding *has-derivative-within Lim-within dist-norm*
unfolding *diff-0-right*
by (*simp add*: *diff-diff-eq*)

lemma *has-derivative-at'*:
(*f has-derivative f'*) (*at x*) $\longleftrightarrow$ *bounded-linear f'* $\wedge$
$(\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, x'.\ 0 < norm\ (x' - x) \wedge norm\ (x' - x) < d \longrightarrow$
norm $(f\,x' - f\,x - f'(x' - x)) \,/\, norm\ (x' - x) < e$)
using *has-derivative-within'* [*of f f' x UNIV*]
by *simp*

lemma *has-derivative-at-within*:
(*f has-derivative f'*) (*at x*) $\Longrightarrow$ (*f has-derivative f'*) (*at x within s*)
unfolding *has-derivative-within' has-derivative-at'*
by *blast*

lemma *has-derivative-within-open*:
$a \in s \Longrightarrow$ *open s* $\Longrightarrow$
(*f has-derivative f'*) (*at a within s*) $\longleftrightarrow$ (*f has-derivative f'*) (*at a*)
by (*simp only*: *at-within-interior interior-open*)

lemma *has-derivative-right*:
fixes *f* :: *real* $\Rightarrow$ *real*
and *y* :: *real*
shows (*f has-derivative* (*op* $*$ *y*)) (*at x within* ($\{x$ <..} $\cap$ *I*)) $\longleftrightarrow$
$((\lambda t.\ (f\,x - f\,t) \,/\, (x - t)) \longrightarrow y)$ (*at x within* ($\{x$ <..} $\cap$ *I*))
proof −
have $((\lambda t.\ (f\,t - (f\,x + y * (t - x))) \,/\, |t - x|) \longrightarrow 0)$ (*at x within* ($\{x$<..} $\cap$ *I*)) $\longleftrightarrow$
$((\lambda t.\ (f\,t - f\,x) \,/\, (t - x) - y) \longrightarrow 0)$ (*at x within* ($\{x$<..} $\cap$ *I*))
by (*intro Lim-cong-within*) (*auto simp add*: *diff-divide-distrib add-divide-distrib*)
also have ... $\longleftrightarrow$ $((\lambda t.\ (f\,t - f\,x) \,/\, (t - x)) \longrightarrow y)$ (*at x within* ($\{x$<..} $\cap$ *I*))
by (*simp add*: *Lim-null*[*symmetric*])
also have ... $\longleftrightarrow$ $((\lambda t.\ (f\,x - f\,t) \,/\, (x - t)) \longrightarrow y)$ (*at x within* ($\{x$<..} $\cap$ *I*))
by (*intro Lim-cong-within*) (*simp-all add*: *field-simps*)
finally show *?thesis*
by (*simp add*: *bounded-linear-mult-right has-derivative-within*)
qed

34.2.1 Caratheodory characterization

lemma *DERIV-within-iff*:
(*f has-field-derivative D*) (*at a within s*) $\longleftrightarrow$ $((\lambda z.\ (f\,z - f\,a) \,/\, (z - a)) \longrightarrow D)$ (*at a within s*)
proof −
have *1*: $\bigwedge w\ y.\ {}^{\sim}(w = a) ==> y \,/\, (w - a) - D = (y - (w - a)*D)/(w - a)$

by (*metis divide-diff-eq-iff eq-iff-diff-eq-0 mult.commute*)
show *?thesis*
apply (*simp add*: *has-field-derivative-def has-derivative-within bounded-linear-mult-right*)
apply (*simp add*: *LIM-zero-iff* [**where** *l* = *D*, *symmetric*])
apply (*simp add*: *Lim-within dist-norm*)
apply (*simp add*: *nonzero-norm-divide* [*symmetric*])
apply (*simp add*: *1 diff-diff-eq ac-simps*)
done
qed

lemma *DERIV-caratheodory-within*:
(*f has-field-derivative l*) (*at x within s*) ⟷
(∃ *g*. (∀ *z*. *f z* − *f x* = *g z* ∗ (*z* − *x*)) ∧ *continuous* (*at x within s*) *g* ∧ *g x* = *l*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
show *?rhs*
proof (*intro exI conjI*)
let *?g* = (%*z*. *if z* = *x then l else* (*f z* − *f x*) / (*z*−*x*))
show ∀ *z*. *f z* − *f x* = *?g z* ∗ (*z*−*x*) **by** *simp*
show *continuous* (*at x within s*) *?g* **using** ‹*?lhs*›
by (*auto simp add*: *continuous-within DERIV-within-iff cong*: *Lim-cong-within*)
show *?g x* = *l* **by** *simp*
qed
next
assume *?rhs*
then obtain *g* **where**
(∀ *z*. *f z* − *f x* = *g z* ∗ (*z*−*x*)) **and** *continuous* (*at x within s*) *g* **and** *g x* = *l*
by *blast*
thus *?lhs*
by (*auto simp add*: *continuous-within DERIV-within-iff cong*: *Lim-cong-within*)
qed

34.2.2 Limit transformation for derivatives

lemma *has-derivative-transform-within*:
assumes (*f has-derivative f'*) (*at x within s*)
and *0* < *d*
and *x* ∈ *s*
and ⋀*x'*. ⟦*x'* ∈ *s*; *dist x' x* < *d*⟧ ⟹ *f x'* = *g x'*
shows (*g has-derivative f'*) (*at x within s*)
using *assms*
unfolding *has-derivative-within*
by (*force simp add*: *intro*: *Lim-transform-within*)

lemma *has-derivative-transform-within-open*:
assumes (*f has-derivative f'*) (*at x*)
and *open s*
and *x* ∈ *s*

and $\bigwedge x.\ x{\in}s \Longrightarrow f\ x = g\ x$
shows $(g$ *has-derivative* $f')$ $(at\ x)$
using *assms* **unfolding** *has-derivative-at*
by (*force simp add*: *intro*: *Lim-transform-within-open*)

34.3 Differentiability

definition
$differentiable\text{-}on :: ('a{::}real\text{-}normed\text{-}vector \Rightarrow 'b{::}real\text{-}normed\text{-}vector) \Rightarrow 'a\ set \Rightarrow bool$
(**infix** *differentiable'-on 50*)
where f *differentiable-on* $s \longleftrightarrow (\forall x{\in}s.\ f\ differentiable\ (at\ x\ within\ s))$

lemma *differentiableI*: $(f$ *has-derivative* $f')\ net \Longrightarrow f\ differentiable\ net$
unfolding *differentiable-def*
by *auto*

lemma *differentiable-at-withinI*: $f\ differentiable\ (at\ x) \Longrightarrow f\ differentiable\ (at\ x\ within\ s)$
unfolding *differentiable-def*
using *has-derivative-at-within*
by *blast*

lemma *differentiable-at-imp-differentiable-on*:
$(\bigwedge x.\ x \in s \Longrightarrow f\ differentiable\ at\ x) \Longrightarrow f\ differentiable\text{-}on\ s$
by (*metis differentiable-at-withinI differentiable-on-def*)

corollary *differentiable-iff-scaleR*:
fixes $f :: real \Rightarrow 'a{::}real\text{-}normed\text{-}vector$
shows $f\ differentiable\ F \longleftrightarrow (\exists d.\ (f\ has\text{-}derivative\ (\lambda x.\ x *_R d))\ F)$
by (*auto simp*: *differentiable-def dest*: *has-derivative-linear linear-imp-scaleR*)

lemma *differentiable-within-open*:
assumes $a \in s$
and *open s*
shows $f\ differentiable\ (at\ a\ within\ s) \longleftrightarrow f\ differentiable\ (at\ a)$
using *assms*
by (*simp only*: *at-within-interior interior-open*)

lemma *differentiable-on-eq-differentiable-at*:
$open\ s \Longrightarrow f\ differentiable\text{-}on\ s \longleftrightarrow (\forall x{\in}s.\ f\ differentiable\ at\ x)$
unfolding *differentiable-on-def*
by (*metis at-within-interior interior-open*)

lemma *differentiable-transform-within*:
assumes $f\ differentiable\ (at\ x\ within\ s)$
and $0 < d$
and $x \in s$
and $\bigwedge x'.\ [\![x'{\in}s;\ dist\ x'\ x < d]\!] \Longrightarrow f\ x' = g\ x'$

shows *g differentiable* (*at x within s*)
using *assms has-derivative-transform-within* **unfolding** *differentiable-def*
by *blast*

34.4 Frechet derivative and Jacobian matrix

definition *frechet-derivative f net* = (*SOME f'*. (*f has-derivative f'*) *net*)

lemma *frechet-derivative-works*:
f differentiable net ⟷ (*f has-derivative* (*frechet-derivative f net*)) *net*
unfolding *frechet-derivative-def differentiable-def*
unfolding *some-eq-ex*[*of* λ *f'* . (*f has-derivative f'*) *net*] **..**

lemma *linear-frechet-derivative*: *f differentiable net* ⟹ *linear* (*frechet-derivative f net*)
unfolding *frechet-derivative-works has-derivative-def*
by (*auto intro*: *bounded-linear.linear*)

34.5 Differentiability implies continuity

lemma *differentiable-imp-continuous-within*:
f differentiable (*at x within s*) ⟹ *continuous* (*at x within s*) *f*
by (*auto simp*: *differentiable-def intro*: *has-derivative-continuous*)

lemma *differentiable-imp-continuous-on*:
f differentiable-on s ⟹ *continuous-on s f*
unfolding *differentiable-on-def continuous-on-eq-continuous-within*
using *differentiable-imp-continuous-within* **by** *blast*

lemma *differentiable-on-subset*:
f differentiable-on t ⟹ *s* ⊆ *t* ⟹ *f differentiable-on s*
unfolding *differentiable-on-def*
using *differentiable-within-subset*
by *blast*

lemma *differentiable-on-empty*: *f differentiable-on* {}
unfolding *differentiable-on-def*
by *auto*

Results about neighborhoods filter.

lemma *eventually-nhds-metric-le*:
eventually P (*nhds a*) = (∃ *d*>*0*. ∀ *x*. *dist x a* ≤ *d* ⟶ *P x*)
unfolding *eventually-nhds-metric* **by** (*safe*, *rule-tac x*=*d* / *2* **in** *exI*, *auto*)

lemma *le-nhds*: *F* ≤ *nhds a* ⟷ (∀ *S*. *open S* ∧ *a* ∈ *S* ⟶ *eventually* (λ*x*. *x* ∈ *S*) *F*)
unfolding *le-filter-def eventually-nhds* **by** (*fast elim*: *eventually-mono*)

lemma *le-nhds-metric*: *F* ≤ *nhds a* ⟷ (∀ *e*>*0*. *eventually* (λ*x*. *dist x a* < *e*) *F*)

unfolding *le-filter-def eventually-nhds-metric* **by** (*fast elim*: *eventually-mono*)

lemma *le-nhds-metric-le*: $F \leq nhds\ a \longleftrightarrow (\forall\, e{>}0.\ eventually\ (\lambda x.\ dist\ x\ a \leq e)\ F)$
unfolding *le-filter-def eventually-nhds-metric-le* **by** (*fast elim*: *eventually-mono*)

Several results are easier using a "multiplied-out" variant. (I got this idea from Dieudonne's proof of the chain rule).

lemma *has-derivative-within-alt*:
$(f\ has\text{-}derivative\ f')\ (at\ x\ within\ s) \longleftrightarrow bounded\text{-}linear\ f' \wedge$
$(\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, y{\in}s.\ norm(y - x) < d \longrightarrow norm\ (f\ y - f\ x - f'\ (y - x)) \leq e * norm\ (y - x))$
unfolding *has-derivative-within filterlim-def le-nhds-metric-le eventually-filtermap eventually-at dist-norm diff-diff-eq*
by (*force simp add*: *linear-0 bounded-linear.linear pos-divide-le-eq*)

lemma *has-derivative-within-alt2*:
$(f\ has\text{-}derivative\ f')\ (at\ x\ within\ s) \longleftrightarrow bounded\text{-}linear\ f' \wedge$
$(\forall\, e{>}0.\ eventually\ (\lambda y.\ norm\ (f\ y - f\ x - f'\ (y - x)) \leq e * norm\ (y - x))\ (at\ x\ within\ s))$
unfolding *has-derivative-within filterlim-def le-nhds-metric-le eventually-filtermap eventually-at dist-norm diff-diff-eq*
by (*force simp add*: *linear-0 bounded-linear.linear pos-divide-le-eq*)

lemma *has-derivative-at-alt*:
$(f\ has\text{-}derivative\ f')\ (at\ x) \longleftrightarrow$
$bounded\text{-}linear\ f' \wedge$
$(\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, y.\ norm(y - x) < d \longrightarrow norm\ (f\ y - f\ x - f'(y - x)) \leq e * norm\ (y - x))$
using *has-derivative-within-alt*[**where** *s*=*UNIV*]
by *simp*

34.6 The chain rule

lemma *diff-chain-within*[*derivative-intros*]:
assumes $(f\ has\text{-}derivative\ f')\ (at\ x\ within\ s)$
and $(g\ has\text{-}derivative\ g')\ (at\ (f\ x)\ within\ (f\ `\ s))$
shows $((g \circ f)\ has\text{-}derivative\ (g' \circ f'))(at\ x\ within\ s)$
using *has-derivative-in-compose*[*OF assms*]
by (*simp add*: *comp-def*)

lemma *diff-chain-at*[*derivative-intros*]:
$(f\ has\text{-}derivative\ f')\ (at\ x) \Longrightarrow$
$(g\ has\text{-}derivative\ g')\ (at\ (f\ x)) \Longrightarrow ((g \circ f)\ has\text{-}derivative\ (g' \circ f'))\ (at\ x)$
using *has-derivative-compose*[*of f f' x UNIV g g'*]
by (*simp add*: *comp-def*)

34.7 Composition rules stated just for differentiability

lemma *differentiable-chain-at*:

f differentiable (*at x*) ⟹
g differentiable (*at* (*f x*)) ⟹ (*g* ∘ *f*) *differentiable* (*at x*)
unfolding *differentiable-def*
by (*meson diff-chain-at*)

lemma *differentiable-chain-within*:
f differentiable (*at x within s*) ⟹
g differentiable (*at*(*f x*) *within* (*f ' s*)) ⟹ (*g* ∘ *f*) *differentiable* (*at x within s*)
unfolding *differentiable-def*
by (*meson diff-chain-within*)

34.8 Uniqueness of derivative

The general result is a bit messy because we need approachability of the limit point from any direction. But OK for nontrivial intervals etc.

lemma *frechet-derivative-unique-within*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*real-normed-vector*
assumes (*f has-derivative f'*) (*at x within s*)
and (*f has-derivative f''*) (*at x within s*)
and ∀ *i*∈*Basis*. ∀ *e*>*0*. ∃ *d*. *0* < |*d*| ∧ |*d*| < *e* ∧ (*x* + *d* ∗$_R$ *i*) ∈ *s*
shows *f'* = *f''*
proof −
note *as* = *assms*(*1*,*2*)[*unfolded has-derivative-def*]
then interpret *f'*: *bounded-linear f'* **by** *auto*
from *as* **interpret** *f''*: *bounded-linear f''* **by** *auto*
have *x islimpt s* **unfolding** *islimpt-approachable*
proof (*rule*, *rule*)
fix *e* :: *real*
assume *e* > *0*
obtain *d* **where** *0* < |*d*| **and** |*d*| < *e* **and** *x* + *d* ∗$_R$ (*SOME i*. *i* ∈ *Basis*) ∈ *s*
using *assms*(*3*) *SOME-Basis* ‹*e*>*0*› **by** *blast*
then show ∃ *x'*∈*s*. *x'* ≠ *x* ∧ *dist x' x* < *e*
apply (*rule-tac x*=*x* + *d* ∗$_R$ (*SOME i*. *i* ∈ *Basis*) **in** *bexI*)
unfolding *dist-norm*
apply (*auto simp*: *SOME-Basis nonzero-Basis*)
done
qed
then have ∗: *netlimit* (*at x within s*) = *x*
apply (*auto intro*!: *netlimit-within*)
by (*metis trivial-limit-within*)
show *?thesis*
apply (*rule linear-eq-stdbasis*)
unfolding *linear-conv-bounded-linear*
apply (*rule as*(*1*,*2*)[*THEN conjunct1*])+
proof (*rule*, *rule ccontr*)
fix *i* :: *'a*
assume *i*: *i* ∈ *Basis*
def *e* ≡ *norm* (*f' i* − *f'' i*)
assume *f' i* ≠ *f'' i*

```
    then have e > 0
      unfolding e-def by auto
    obtain d where d:
      0 < d
      (⋀xa. xa∈s ⟶ 0 < dist xa x ∧ dist xa x < d ⟶
          dist ((f xa − f x − f' (xa − x)) /R norm (xa − x) −
              (f xa − f x − f'' (xa − x)) /R norm (xa − x)) (0 − 0) < e)
      using tendsto-diff [OF as(1,2)[THEN conjunct2]]
      unfolding ∗ Lim-within
      using ‹e>0› by blast
    obtain c where c: 0 < |c| |c| < d ∧ x + c ∗R i ∈ s
      using assms(3) i d(1) by blast
    have ∗: norm (− ((1 / |c|) ∗R f' (c ∗R i)) + (1 / |c|) ∗R f'' (c ∗R i)) =
        norm ((1 / |c|) ∗R (− (f' (c ∗R i)) + f'' (c ∗R i)))
      unfolding scaleR-right-distrib by auto
    also have ... = norm ((1 / |c|) ∗R (c ∗R (− (f' i) + f'' i)))
      unfolding f'.scaleR f''.scaleR
      unfolding scaleR-right-distrib scaleR-minus-right
      by auto
    also have ... = e
      unfolding e-def
      using c(1)
      using norm-minus-cancel[of f' i − f'' i]
      by auto
    finally show False
      using c
      using d(2)[of x + c ∗R i]
      unfolding dist-norm
      unfolding f'.scaleR f''.scaleR f'.add f''.add f'.diff f''.diff
        scaleR-scaleR scaleR-right-diff-distrib scaleR-right-distrib
      using i
      by (auto simp: inverse-eq-divide)
  qed
qed

lemma frechet-derivative-unique-at:
  (f has-derivative f') (at x) ⟹ (f has-derivative f'') (at x) ⟹ f' = f''
  by (rule has-derivative-unique)

lemma frechet-derivative-unique-within-closed-interval:
  fixes f::'a::euclidean-space ⇒ 'b::real-normed-vector
  assumes ∀ i∈Basis. a•i < b•i
    and x ∈ cbox a b
    and (f has-derivative f' ) (at x within cbox a b)
    and (f has-derivative f'') (at x within cbox a b)
  shows f' = f''
  apply(rule frechet-derivative-unique-within)
  apply(rule assms(3,4))+
proof (rule, rule, rule)
```

```
  fix e :: real
  fix i :: 'a
  assume e > 0 and i: i ∈ Basis
  then show ∃ d. 0 < |d| ∧ |d| < e ∧ x + d *R i ∈ cbox a b
  proof (cases x•i = a•i)
    case True
    then show ?thesis
      apply (rule-tac x=(min (b•i − a•i)  e) / 2 in exI)
      using assms(1)[THEN bspec[where x=i]] and ‹e>0› and assms(2)
      unfolding mem-box
      using i
      apply (auto simp add: field-simps inner-simps inner-Basis)
      done
  next
    note * = assms(2)[unfolded mem-box, THEN bspec, OF i]
    case False
    moreover have a • i < x • i
      using False * by auto
    moreover {
      have a • i * 2 + min (x • i − a • i) e ≤ a•i *2 + x•i − a•i
        by auto
      also have ... = a•i + x•i
        by auto
      also have ... ≤ 2 * (x•i)
        using * by auto
      finally have a • i * 2 + min (x • i − a • i) e ≤ x • i * 2
        by auto
    }
    moreover have min (x • i − a • i) e ≥ 0
      using * and ‹e>0› by auto
    then have x • i * 2 ≤ b • i * 2 + min (x • i − a • i) e
      using * by auto
    ultimately show ?thesis
      apply (rule-tac x=− (min (x•i − a•i) e) / 2 in exI)
      using assms(1)[THEN bspec, OF i] and ‹e>0› and assms(2)
      unfolding mem-box
      using i
      apply (auto simp add: field-simps inner-simps inner-Basis)
      done
  qed
qed

lemma frechet-derivative-unique-within-open-interval:
  fixes f::'a::euclidean-space ⇒ 'b::real-normed-vector
  assumes x ∈ box a b
    and (f has-derivative f' ) (at x within box a b)
    and (f has-derivative f'') (at x within box a b)
  shows f' = f''
proof −
```

from *assms*(*1*) **have** ∗: *at x within box a b* = *at x*
by (*metis at-within-interior interior-open open-box*)
from *assms*(*2*,*3*) [*unfolded* ∗] **show** *f′* = *f″*
by (*rule frechet-derivative-unique-at*)
qed

lemma *frechet-derivative-at*:
(*f has-derivative f′*) (*at x*) ⟹ *f′* = *frechet-derivative f* (*at x*)
apply (*rule frechet-derivative-unique-at*[*of f*])
apply *assumption*
unfolding *frechet-derivative-works*[*symmetric*]
using *differentiable-def*
apply *auto*
done

lemma *frechet-derivative-within-cbox*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*real-normed-vector*
assumes ∀ *i*∈*Basis*. *a*·*i* < *b*·*i*
and *x* ∈ *cbox a b*
and (*f has-derivative f′*) (*at x within cbox a b*)
shows *frechet-derivative f* (*at x within cbox a b*) = *f′*
using *assms*
by (*metis Derivative.differentiableI frechet-derivative-unique-within-closed-interval frechet-derivative-works*)

34.9 The traditional Rolle theorem in one dimension

Derivatives of local minima and maxima are zero.

lemma *has-derivative-local-min*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *real*
assumes *deriv*: (*f has-derivative f′*) (*at x*)
assumes *min*: *eventually* (λ*y*. *f x* ≤ *f y*) (*at x*)
shows *f′* = (λ*h*. *0*)
proof
fix *h* :: *'a*
interpret *f′*: *bounded-linear f′*
using *deriv* **by** (*rule has-derivative-bounded-linear*)
show *f′ h* = *0*
proof (*cases h* = *0*)
assume *h* ≠ *0*
from *min* **obtain** *d* **where** *d1*: *0* < *d* **and** *d2*: ∀ *y*∈*ball x d*. *f x* ≤ *f y*
unfolding *eventually-at* **by** (*force simp*: *dist-commute*)
have *FDERIV* (λ*r*. *x* + *r* ∗*R* *h*) *0* :> (λ*r*. *r* ∗*R* *h*)
by (*intro derivative-eq-intros*) *auto*
then have *FDERIV* (λ*r*. *f* (*x* + *r* ∗*R* *h*)) *0* :> (λ*k*. *f′* (*k* ∗*R* *h*))
by (*rule has-derivative-compose*, *simp add*: *deriv*)
then have *DERIV* (λ*r*. *f* (*x* + *r* ∗*R* *h*)) *0* :> *f′ h*
unfolding *has-field-derivative-def* **by** (*simp add*: *f′.scaleR mult-commute-abs*)
moreover have *0* < *d* / *norm h* **using** *d1* **and** ‹*h* ≠ *0*› **by** *simp*

moreover have $\forall y.\ |0 - y| < d \ / \ norm\ h \longrightarrow f\ (x + 0 *_R h) \leq f\ (x + y *_R h)$
using ‹$h \neq 0$› **by** (*auto simp add*: *d2 dist-norm pos-less-divide-eq*)
ultimately show $f'\ h = 0$
by (*rule DERIV-local-min*)
qed (*simp add*: *f'.zero*)
qed

lemma *has-derivative-local-max*:
fixes $f :: 'a$::*real-normed-vector* $\Rightarrow$ *real*
assumes (*f has-derivative f'*) (*at x*)
assumes *eventually* $(\lambda y.\ f\ y \leq f\ x)$ (*at x*)
shows $f' = (\lambda h.\ 0)$
using *has-derivative-local-min* [*of* $\lambda x.\ -f\ x\ \lambda h.\ -f'\ h\ x$]
using *assms* **unfolding** *fun-eq-iff* **by** *simp*

lemma *differential-zero-maxmin*:
fixes f::$'a$::*real-normed-vector* $\Rightarrow$ *real*
assumes $x \in s$
and *open s*
and *deriv*: (*f has-derivative f'*) (*at x*)
and *mono*: $(\forall y{\in}s.\ f\ y \leq f\ x) \vee (\forall y{\in}s.\ f\ x \leq f\ y)$
shows $f' = (\lambda v.\ 0)$
using *mono*
proof
assume $\forall y{\in}s.\ f\ y \leq f\ x$
with ‹$x \in s$› **and** ‹*open s*› **have** *eventually* $(\lambda y.\ f\ y \leq f\ x)$ (*at x*)
unfolding *eventually-at-topological* **by** *auto*
with *deriv* **show** *?thesis*
by (*rule has-derivative-local-max*)
next
assume $\forall y{\in}s.\ f\ x \leq f\ y$
with ‹$x \in s$› **and** ‹*open s*› **have** *eventually* $(\lambda y.\ f\ x \leq f\ y)$ (*at x*)
unfolding *eventually-at-topological* **by** *auto*
with *deriv* **show** *?thesis*
by (*rule has-derivative-local-min*)
qed

lemma *differential-zero-maxmin-component*:
fixes $f :: 'a$::*euclidean-space* $\Rightarrow$ $'b$::*euclidean-space*
assumes *k*: $k \in Basis$
and *ball*: $0 < e\ (\forall y \in ball\ x\ e.\ (f\ y){\cdot}k \leq (f\ x){\cdot}k) \vee (\forall y{\in}ball\ x\ e.\ (f\ x){\cdot}k \leq (f\ y){\cdot}k)$
and *diff*: *f differentiable* (*at x*)
shows $(\sum j{\in}Basis.\ (frechet\text{-}derivative\ f\ (at\ x)\ j \cdot k) *_R j) = (0::'a)$ (**is** *?D k* = *0*)
proof −
let *?f'* = *frechet-derivative f* (*at x*)
have $x \in ball\ x\ e$ **using** ‹$0 < e$› **by** *simp*

```
  moreover have open (ball x e) by simp
  moreover have ((λx. f x · k) has-derivative (λh. ?f' h · k)) (at x)
    using bounded-linear-inner-left diff[unfolded frechet-derivative-works]
    by (rule bounded-linear.has-derivative)
  ultimately have (λh. frechet-derivative f (at x) h · k) = (λv. 0)
    using ball(2) by (rule differential-zero-maxmin)
  then show ?thesis
    unfolding fun-eq-iff by simp
qed

lemma rolle:
  fixes f :: real ⇒ real
  assumes a < b
    and f a = f b
    and continuous-on {a .. b} f
    and ∀x∈{a <..< b}. (f has-derivative f' x) (at x)
  shows ∃x∈{a <..< b}. f' x = (λv. 0)
proof −
  have ∃x∈box a b. (∀y∈box a b. f x ≤ f y) ∨ (∀y∈box a b. f y ≤ f x)
  proof −
    have (a + b) / 2 ∈ {a .. b}
      using assms(1) by auto
    then have ∗: {a .. b} ≠ {}
      by auto
    obtain d where d:
        d ∈cbox a b
        ∀y∈cbox a b. f y ≤ f d
      using continuous-attains-sup[OF compact-Icc ∗ assms(3)] by auto
    obtain c where c:
        c ∈ cbox a b
        ∀y∈cbox a b. f c ≤ f y
      using continuous-attains-inf[OF compact-Icc ∗ assms(3)] by auto
    show ?thesis
    proof (cases d ∈ box a b ∨ c ∈ box a b)
      case True
      then show ?thesis
        by (metis c(2) d(2) box-subset-cbox subset-iff)
    next
      def e ≡ (a + b) /2
      case False
      then have f d = f c
        using d c assms(2) by auto
      then have ⋀x. x ∈ {a..b} ⟹ f x = f d
        using c d
        by force
      then show ?thesis
        apply (rule-tac x=e in bexI)
        unfolding e-def
        using assms(1)
```

```
      apply auto
      done
  qed
 qed
 then obtain x where x: x ∈ {a <..< b} (∀ y∈{a <..< b}. f x ≤ f y) ∨ (∀ y∈{a
<..< b}. f y ≤ f x)
   by auto
 then have f' x = (λv. 0)
   apply (rule-tac differential-zero-maxmin[of x box a b f f' x])
   using assms
   apply auto
   done
 then show ?thesis
   by (metis x(1))
qed
```

34.10 One-dimensional mean value theorem

```
lemma mvt:
 fixes f :: real ⇒ real
 assumes a < b
   and continuous-on {a..b} f
 assumes ∀ x∈{a<..<b}. (f has-derivative (f' x)) (at x)
 shows ∃ x∈{a<..<b}. f b − f a = (f' x) (b − a)
proof −
 have ∃ x∈{a <..< b}. (λxa. f' x xa − (f b − f a) / (b − a) ∗ xa) = (λv. 0)
 proof (intro rolle[OF assms(1), of λx. f x − (f b − f a) / (b − a) ∗ x] ballI)
   fix x
   assume x: x ∈ {a <..< b}
   show ((λx. f x − (f b − f a) / (b − a) ∗ x) has-derivative
       (λxa. f' x xa − (f b − f a) / (b − a) ∗ xa)) (at x)
     by (intro derivative-intros assms(3)[rule-format,OF x])
 qed (insert assms(1,2), auto intro!: continuous-intros simp: field-simps)
 then obtain x where
   x ∈ {a <..< b}
   (λxa. f' x xa − (f b − f a) / (b − a) ∗ xa) = (λv. 0) ..
 then show ?thesis
   by (metis (hide-lams) assms(1) diff-gt-0-iff-gt eq-iff-diff-eq-0
     zero-less-mult-iff nonzero-mult-divide-cancel-right not-real-square-gt-zero
     times-divide-eq-left)
qed

lemma mvt-simple:
 fixes f :: real ⇒ real
 assumes a < b
   and ∀ x∈{a..b}. (f has-derivative f' x) (at x within {a..b})
 shows ∃ x∈{a<..<b}. f b − f a = f' x (b − a)
proof (rule mvt)
 have f differentiable-on {a..b}
```

```
    using assms(2) unfolding differentiable-on-def differentiable-def by fast
  then show continuous-on {a..b} f
    by (rule differentiable-imp-continuous-on)
  show ∀x∈{a<..<b}. (f has-derivative f' x) (at x)
  proof
    fix x
    assume x: x ∈ {a <..< b}
    show (f has-derivative f' x) (at x)
      unfolding at-within-open[OF x open-greaterThanLessThan,symmetric]
      apply (rule has-derivative-within-subset)
      apply (rule assms(2)[rule-format])
      using x
      apply auto
      done
  qed
qed (rule assms(1))

lemma mvt-very-simple:
  fixes f :: real ⇒ real
  assumes a ≤ b
    and ∀x∈{a .. b}. (f has-derivative f' x) (at x within {a .. b})
  shows ∃x∈{a .. b}. f b − f a = f' x (b − a)
proof (cases a = b)
  interpret bounded-linear f' b
    using assms(2) assms(1) by auto
  case True
  then show ?thesis
    apply (rule-tac x=a in bexI)
    using assms(2)[THEN bspec[where x=a]]
    unfolding has-derivative-def
    unfolding True
    using zero
    apply auto
    done
next
  case False
  then show ?thesis
    using mvt-simple[OF - assms(2)]
    using assms(1)
    by auto
qed
```

A nice generalization (see Havin's proof of 5.19 from Rudin's book).

```
lemma mvt-general:
  fixes f :: real ⇒ 'a::real-inner
  assumes a < b
    and continuous-on {a .. b} f
    and ∀x∈{a<..<b}. (f has-derivative f'(x)) (at x)
  shows ∃x∈{a<..<b}. norm (f b − f a) ≤ norm (f' x (b − a))
```

proof −
have $\exists x{\in}\{a<..<b\}.\ (f\ b - f\ a) \cdot f\ b - (f\ b - f\ a) \cdot f\ a = (f\ b - f\ a) \cdot f'\ x\ (b - a)$
apply (*rule mvt*)
apply (*rule assms*(*1*))
apply (*intro continuous-intros assms*(*2*))
using *assms*(*3*)
apply (*fast intro*: *has-derivative-inner-right*)
done
then obtain *x* **where** *x*:
$x \in \{a<..<b\}$
$(f\ b - f\ a) \cdot f\ b - (f\ b - f\ a) \cdot f\ a = (f\ b - f\ a) \cdot f'\ x\ (b - a)$ **..**
show *?thesis*
proof (*cases f a = f b*)
case *False*
have *norm* $(f\ b - f\ a) * norm\ (f\ b - f\ a) = (norm\ (f\ b - f\ a))^2$
by (*simp add*: *power2-eq-square*)
also have $\ldots = (f\ b - f\ a) \cdot (f\ b - f\ a)$
unfolding *power2-norm-eq-inner* **..**
also have $\ldots = (f\ b - f\ a) \cdot f'\ x\ (b - a)$
using *x*(*2*) **by** (*simp only*: *inner-diff-right*)
also have $\ldots \leq norm\ (f\ b - f\ a) * norm\ (f'\ x\ (b - a))$
by (*rule norm-cauchy-schwarz*)
finally show *?thesis*
using *False x*(*1*)
by (*auto simp add*: *mult-left-cancel*)
next
case *True*
then show *?thesis*
using *assms*(*1*)
apply (*rule-tac x*=(*a* + *b*) */2* **in** *bexI*)
apply *auto*
done
qed
qed

34.11 More general bound theorems

lemma *differentiable-bound-general*:
fixes *f* :: *real* ⇒ *'a*::*real-normed-vector*
assumes *a* < *b*
and *f-cont*: *continuous-on* {*a* .. *b*} *f*
and *phi-cont*: *continuous-on* {*a* .. *b*} φ
and *f'*: $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow (f\ has\text{-}vector\text{-}derivative\ f'\ x)\ (at\ x)$
and *phi'*: $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow (\varphi\ has\text{-}vector\text{-}derivative\ \varphi'\ x)\ (at\ x)$
and *bnd*: $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow norm\ (f'\ x) \leq \varphi'\ x$
shows *norm* $(f\ b - f\ a) \leq \varphi\ b - \varphi\ a$
proof −
{

```
  fix x assume x: a < x x < b
  have 0 ≤ norm (f' x) by simp
  also have ... ≤ φ' x using x by (auto intro!: bnd)
  finally have 0 ≤ φ' x .
 } note phi'-nonneg = this
 note f-tendsto = assms(2)[simplified continuous-on-def, rule-format]
 note phi-tendsto = assms(3)[simplified continuous-on-def, rule-format]
 {
  fix e::real assume e > 0
  def e2 ≡ e / 2 with ‹e > 0› have e2 > 0 by simp
  let ?le = λx1. norm (f x1 − f a) ≤ φ x1 − φ a + e * (x1 − a) + e
  def A ≡ {x2. a ≤ x2 ∧ x2 ≤ b ∧ (∀x1∈{a ..< x2}. ?le x1)}
  have A-subset: A ⊆ {a .. b} by (auto simp: A-def)
  {
   fix x2
   assume a: a ≤ x2 x2 ≤ b and le: ∀x1∈{a..<x2}. ?le x1
   have ?le x2 using ‹e > 0›
   proof cases
    assume x2 ≠ a with a have a < x2 by simp
    have at x2 within {a <..<x2} ≠ bot
     using ‹a < x2›
     by (auto simp: trivial-limit-within islimpt-in-closure)
    moreover
    have ((λx1. (φ x1 − φ a) + e * (x1 − a) + e) ⟶ (φ x2 − φ a) + e *
(x2 − a) + e) (at x2 within {a <..<x2})
      ((λx1. norm (f x1 − f a)) ⟶ norm (f x2 − f a)) (at x2 within {a
<..<x2})
     using a
     by (auto intro!: tendsto-eq-intros f-tendsto phi-tendsto
      intro: tendsto-within-subset[where S={a .. b}])
    moreover
    have eventually (λx. x > a) (at x2 within {a <..<x2})
     by (auto simp: eventually-at-filter)
    hence eventually ?le (at x2 within {a <..<x2})
     unfolding eventually-at-filter
     by eventually-elim (insert le, auto)
    ultimately
    show ?thesis
     by (rule tendsto-le)
   qed simp
  } note le-cont = this
  have a ∈ A
   using assms by (auto simp: A-def)
  hence [simp]: A ≠ {} by auto
  have A-ivl: ⋀x1 x2. x2 ∈ A ⟹ x1 ∈ {a ..x2} ⟹ x1 ∈ A
   by (simp add: A-def)
  have [simp]: bdd-above A by (auto simp: A-def)
  def y ≡ Sup A
  have y ≤ b
```

```
  unfolding y-def
  by (simp add: cSup-le-iff) (simp add: A-def)
 have leI: ⋀x x1. a ≤ x1 ⟹ x ∈ A ⟹ x1 < x ⟹ ?le x1
  by (auto simp: A-def intro!: le-cont)
have y-all-le: ∀x1∈{a..<y}. ?le x1
  by (auto simp: y-def less-cSup-iff leI)
have a ≤ y
  by (metis ‹a ∈ A› ‹bdd-above A› cSup-upper y-def)
have y ∈ A
  using y-all-le ‹a ≤ y› ‹y ≤ b›
  by (auto simp: A-def)
hence A = {a .. y}
  using A-subset
  by (auto simp: subset-iff y-def cSup-upper intro: A-ivl)
from le-cont[OF ‹a ≤ y› ‹y ≤ b› y-all-le] have le-y: ?le y .
{
  assume a ≠ y with ‹a ≤ y› have a < y by simp
  have y = b
  proof (rule ccontr)
    assume y ≠ b
    hence y < b using ‹y ≤ b› by simp
    let ?F = at y within {y..<b}
    from f' phi'
    have (f has-vector-derivative f' y) ?F
      and (φ has-vector-derivative φ' y) ?F
      using ‹a < y› ‹y < b›
     by (auto simp add: at-within-open[of - {a<..<b}] has-vector-derivative-def
        intro!: has-derivative-subset[where s={a<..<b} and t={y..<b}])
   hence ∀F x1 in ?F. norm (f x1 - f y - (x1 - y) *R f' y) ≤ e2 * |x1 - y|
       ∀F x1 in ?F. norm (φ x1 - φ y - (x1 - y) *R φ' y) ≤ e2 * |x1 - y|
      using ‹e2 > 0›
      by (auto simp: has-derivative-within-alt2 has-vector-derivative-def)
    moreover
    have ∀F x1 in ?F. y ≤ x1 ∀F x1 in ?F. x1 < b
      by (auto simp: eventually-at-filter)
    ultimately
    have ∀F x1 in ?F. norm (f x1 - f y) ≤ (φ x1 - φ y) + e * |x1 - y|
      (is ∀F x1 in ?F. ?le' x1)
    proof eventually-elim
      case (elim x1)
      from norm-triangle-ineq2[THEN order-trans, OF elim(1)]
      have norm (f x1 - f y) ≤ norm (f' y) * |x1 - y| + e2 * |x1 - y|
        by (simp add: ac-simps)
      also have norm (f' y) ≤ φ' y using bnd ‹a < y› ‹y < b› by simp
      also
      from elim have φ' y * |x1 - y| ≤ φ x1 - φ y + e2 * |x1 - y|
        by (simp add: ac-simps)
      finally
      have norm (f x1 - f y) ≤ φ x1 - φ y + e2 * |x1 - y| + e2 * |x1 - y|
```

```
      by (auto simp: mult-right-mono)
    thus ?case by (simp add: e2-def)
  qed
  moreover have ?le' y by simp
  ultimately obtain S
  where S: open S y ∈ S ⋀x. x∈S ⟹ x ∈ {y..<b} ⟹ ?le' x
    unfolding eventually-at-topological
    by metis
  from ‹open S› obtain d where d: ⋀x. dist x y < d ⟹ x ∈ S d > 0
    by (force simp: dist-commute open-dist ball-def dest!: bspec[OF - ‹y ∈ S›])
  def d' ≡ min ((y + b)/2) (y + (d/2))
  have d' ∈ A
    unfolding A-def
  proof safe
    show a ≤ d' using ‹a < y› ‹0 < d› ‹y < b› by (simp add: d'-def)
    show d' ≤ b using ‹y < b› by (simp add: d'-def min-def)
    fix x1
    assume x1: x1 ∈ {a..<d'}
    {
      assume x1 < y
      hence ?le x1
        using ‹x1 ∈ {a..<d'}› y-all-le by auto
    } moreover {
      assume x1 ≥ y
      hence x1': x1 ∈ S x1 ∈ {y..<b} using x1
        by (auto simp: d'-def dist-real-def intro!: d)
      have norm (f x1 - f a) ≤ norm (f x1 - f y) + norm (f y - f a)
        by (rule order-trans[OF - norm-triangle-ineq]) simp
      also note S(3)[OF x1']
      also note le-y
      finally have ?le x1
        using ‹x1 ≥ y› by (auto simp: algebra-simps)
    } ultimately show ?le x1 by arith
  qed
  hence d' ≤ y
    unfolding y-def
    by (rule cSup-upper) simp
  thus False using ‹d > 0› ‹y < b›
    by (simp add: d'-def min-def split: split-if-asm)
 qed
} moreover {
 assume a = y
 with ‹a < b› have y < b by simp
 with ‹a = y› f-cont phi-cont ‹e2 > 0›
 have 1: ∀F x in at y within {y..b}. dist (f x) (f y) < e2
  and 2: ∀F x in at y within {y..b}. dist (φ x) (φ y) < e2
   by (auto simp: continuous-on-def tendsto-iff)
 have 3: eventually (λx. y < x) (at y within {y..b})
   by (auto simp: eventually-at-filter)
```

```
    have 4: eventually (λx::real. x < b) (at y within {y..b})
      using - ‹y < b›
      by (rule order-tendstoD) (auto intro!: tendsto-eq-intros)
    from 1 2 3 4
    have eventually-le: eventually (λx. ?le x) (at y within {y .. b})
    proof eventually-elim
      case (elim x1)
      have norm (f x1 − f a) = norm (f x1 − f y)
        by (simp add: ‹a = y›)
      also have norm (f x1 − f y) ≤ e2
        using elim ‹a = y› by (auto simp : dist-norm intro!:  less-imp-le)
      also have ... ≤ e2 + (φ x1 − φ a + e2 + e ∗ (x1 − a))
        using ‹0 < e› elim
        by (intro add-increasing2[OF add-nonneg-nonneg order.refl])
          (auto simp: ‹a = y› dist-norm intro!: mult-nonneg-nonneg)
      also have ... = φ x1 − φ a + e ∗ (x1 − a) + e
        by (simp add: e2-def)
      finally show ?le x1 .
    qed
    from this[unfolded eventually-at-topological] ‹?le y›
    obtain S
    where S: open S y ∈ S ⋀x. x∈S ⟹ x ∈ {y..b} ⟹ ?le x
      by metis
    from ‹open S› obtain d where d: ⋀x. dist x y < d ⟹ x ∈ S d > 0
      by (force simp: dist-commute open-dist ball-def dest!: bspec[OF - ‹y ∈ S›])
    def d' ≡ min b (y + (d/2))
    have d' ∈ A
      unfolding A-def
    proof safe
      show a ≤ d' using ‹a = y› ‹0 < d› ‹y < b› by (simp add: d'-def)
      show d' ≤ b by (simp add: d'-def)
      fix x1
      assume x1 ∈ {a..<d'}
      hence x1 ∈ S x1 ∈ {y..b}
        by (auto simp: ‹a = y› d'-def dist-real-def intro!: d )
      thus ?le x1
        by (rule S)
    qed
    hence d' ≤ y
      unfolding y-def
      by (rule cSup-upper) simp
    hence y = b using ‹d > 0› ‹y < b›
      by (simp add: d'-def)
  } ultimately have y = b
    by auto
  with le-y have norm (f b − f a) ≤ φ b − φ a + e ∗ (b − a + 1)
    by (simp add: algebra-simps)
} note ∗ = this
{
```

```
    fix e::real assume e > 0
    hence norm (f b − f a) ≤ φ b − φ a + e
      using ∗[of e / (b − a + 1)] ‹a < b› by simp
  } thus ?thesis
    by (rule field-le-epsilon)
qed

lemma differentiable-bound:
  fixes f :: 'a::real-normed-vector ⇒ 'b::real-normed-vector
  assumes convex s
    and ∀x∈s. (f has-derivative f' x) (at x within s)
    and ∀x∈s. onorm (f' x) ≤ B
    and x: x ∈ s
    and y: y ∈ s
  shows norm (f x − f y) ≤ B ∗ norm (x − y)
proof −
  let ?p = λu. x + u ∗R (y − x)
  let ?φ = λh. h ∗ B ∗ norm (x − y)
  have ∗: ⋀u. u∈{0..1} ⟹ x + u ∗R (y − x) ∈ s
    using assms(1)[unfolded convex-alt,rule-format,OF x y]
    unfolding scaleR-left-diff-distrib scaleR-right-diff-distrib
    by (auto simp add: algebra-simps)
  have 0: continuous-on (?p ` {0..1}) f
    using ∗
    unfolding continuous-on-eq-continuous-within
    apply −
    apply rule
    apply (rule differentiable-imp-continuous-within)
    unfolding differentiable-def
    apply (rule-tac x=f' xa in exI)
    apply (rule has-derivative-within-subset)
    apply (rule assms(2)[rule-format])
    apply auto
    done
  from ∗ have 1: continuous-on {0 .. 1} (f ∘ ?p)
    by (intro continuous-intros 0)+
  {
    fix u::real assume u: u ∈{0 <..< 1}
    let ?u = ?p u
    interpret linear (f' ?u)
      using u by (auto intro!: has-derivative-linear assms(2)[rule-format] ∗)
    have (f ∘ ?p has-derivative (f' ?u) ∘ (λu. 0 + u ∗R (y − x))) (at u within box
0 1)
      apply (rule diff-chain-within)
      apply (rule derivative-intros)+
      apply (rule has-derivative-within-subset)
      apply (rule assms(2)[rule-format])
      using u ∗
      apply auto
```

```
    done
  hence ((f ∘ ?p) has-vector-derivative f' ?u (y − x)) (at u)
    by (simp add: has-derivative-within-open[OF u open-greaterThanLessThan]
      scaleR has-vector-derivative-def o-def)
 } note 2 = this
 {
  have continuous-on {0..1} ?φ
    by (rule continuous-intros)+
 } note 3 = this
 {
  fix u::real assume u: u ∈{0 <..< 1}
  have (?φ has-vector-derivative B * norm (x − y)) (at u)
    by (auto simp: has-vector-derivative-def intro!: derivative-eq-intros)
 } note 4 = this
 {
  fix u::real assume u: u ∈{0 <..< 1}
  let ?u = ?p u
  interpret bounded-linear (f' ?u)
    using u by (auto intro!: has-derivative-bounded-linear assms(2)[rule-format]
*)
  have norm (f' ?u (y − x)) ≤ onorm (f' ?u) * norm (y − x)
    by (rule onorm) fact
  also have onorm (f' ?u) ≤ B
    using u by (auto intro!: assms(3)[rule-format] *)
  finally have norm ((f' ?u) (y − x)) ≤ B * norm (x − y)
    by (simp add: mult-right-mono norm-minus-commute)
 } note 5 = this
 have norm (f x − f y) = norm ((f ∘ (λu. x + u *R (y − x))) 1 − (f ∘ (λu. x
+ u *R (y − x))) 0)
  by (auto simp add: norm-minus-commute)
 also
 from differentiable-bound-general[OF zero-less-one 1, OF 3 2 4 5]
 have norm ((f ∘ ?p) 1 − (f ∘ ?p) 0) ≤ B * norm (x − y)
  by simp
 finally show ?thesis .
qed

lemma
 differentiable-bound-segment:
 fixes f::'a::real-normed-vector ⇒ 'b::real-normed-vector
 assumes ⋀t. t ∈ {0..1} ⟹ x0 + t *R a ∈ G
 assumes f': ⋀x. x ∈ G ⟹ (f has-derivative f' x) (at x within G)
 assumes B: ∀x∈{0..1}. onorm (f' (x0 + x *R a)) ≤ B
 shows norm (f (x0 + a) − f x0) ≤ norm a * B
proof −
 let ?G = (λx. x0 + x *R a) ‘ {0..1}
 have ?G = op + x0 ‘ (λx. x *R a) ‘ {0..1} by auto
 also have convex ...
  by (intro convex-translation convex-scaled convex-real-interval)
```

finally have *convex ?G* **.**
moreover have *?G* $\subseteq$ *G x0* $\in$ *?G x0* + *a* $\in$ *?G* **using** *assms* **by** (*auto intro*: *image-eqI*[**where** *x=1*])
ultimately show *?thesis*
using *has-derivative-subset*[*OF f'* ‹*?G* $\subseteq$ *G*›] *B*
differentiable-bound[*of* (λ*x*. *x0* + *x* $*_R$ *a*) ‘ {*0..1*} *f f' B x0* + *a x0*]
by (*auto simp*: *ac-simps*)
qed

lemma *differentiable-bound-linearization*:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*
assumes ⋀*t*. *t* $\in$ {*0..1*} ⟹ *a* + *t* $*_R$ (*b* − *a*) $\in$ *S*
assumes *f'*[*derivative-intros*]: ⋀*x*. *x* $\in$ *S* ⟹ (*f has-derivative f' x*) (*at x within S*)
assumes *B*: ∀ *x*∈*S*. *onorm* (*f' x* − *f' x0*) ≤ *B*
assumes *x0* $\in$ *S*
shows *norm* (*f b* − *f a* − *f' x0* (*b* − *a*)) ≤ *norm* (*b* − *a*) ∗ *B*
proof −
def *g* ≡ λ*x*. *f x* − *f' x0 x*
have *g*: ⋀*x*. *x* $\in$ *S* ⟹ (*g has-derivative* (λ*i*. *f' x i* − *f' x0 i*)) (*at x within S*)
unfolding *g-def* **using** *assms*
by (*auto intro*!: *derivative-eq-intros*
bounded-linear.has-derivative[*OF has-derivative-bounded-linear*, *OF f'*])
from *B* **have** *B*: ∀ *x*∈{*0..1*}. *onorm* (λ*i*. *f'* (*a* + *x* $*_R$ (*b* − *a*)) *i* − *f' x0 i*) ≤ *B*
using *assms* **by** (*auto simp*: *fun-diff-def*)
from *differentiable-bound-segment*[*OF assms*(*1*) *g B*] ‹*x0* $\in$ *S*›
show *?thesis*
by (*simp add*: *g-def field-simps linear-sub*[*OF has-derivative-linear*[*OF f'*]])
qed

In particular.

lemma *has-derivative-zero-constant*:
fixes *f* :: '*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*
assumes *convex s*
and ⋀*x*. *x* $\in$ *s* ⟹ (*f has-derivative* (λ*h*. *0*)) (*at x within s*)
shows ∃ *c*. ∀ *x*∈*s*. *f x* = *c*
proof −
{ fix *x y* **assume** *x* $\in$ *s y* $\in$ *s*
then have *norm* (*f x* − *f y*) ≤ *0* ∗ *norm* (*x* − *y*)
using *assms* **by** (*intro differentiable-bound*[*of s*]) (*auto simp*: *onorm-zero*)
then have *f x* = *f y*
by *simp* **}**
then show *?thesis*
by *metis*
qed

lemma *has-field-derivative-zero-constant*:
assumes *convex s* ⋀*x*. *x* $\in$ *s* ⟹ (*f has-field-derivative 0*) (*at x within s*)

shows $\exists c.\ \forall x{\in}s.\ f\ (x) = (c :: 'a :: \textit{real-normed-field})$
proof (*rule has-derivative-zero-constant*)
have *A*: $op * 0 = (\lambda\text{-}.\ 0 :: 'a)$ **by** (*intro ext*) *simp*
fix x **assume** $x \in s$ **thus** (*f has-derivative* $(\lambda h.\ 0)$) (*at x within s*)
using *assms*(*2*)[*of x*] **by** (*simp add*: *has-field-derivative-def A*)
qed *fact*

lemma *has-derivative-zero-unique*:
fixes $f :: 'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-vector*
assumes *convex s*
and $\bigwedge x.\ x \in s \Longrightarrow$ (*f has-derivative* $(\lambda h.\ 0)$) (*at x within s*)
and $x \in s\ y \in s$
shows $f\ x = f\ y$
using *has-derivative-zero-constant*[*OF assms*(*1*,*2*)] *assms*(*3*−) **by** *force*

lemma *has-derivative-zero-unique-connected*:
fixes $f :: 'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-vector*
assumes *open s connected s*
assumes *f*: $\bigwedge x.\ x \in s \Longrightarrow$ (*f has-derivative* $(\lambda x.\ 0)$) (*at x*)
assumes $x \in s\ y \in s$
shows $f\ x = f\ y$
proof (*rule connected-local-const*[**where** *f*=*f*, *OF* ‹*connected s*› ‹$x{\in}s$› ‹$y{\in}s$›])
show $\forall\, a{\in}s.$ *eventually* $(\lambda b.\ f\ a = f\ b)$ (*at a within s*)
proof
fix a **assume** $a \in s$
with ‹*open s*› **obtain** e **where** $0 < e$ *ball a e* $\subseteq s$
by (*rule openE*)
then have $\exists c.\ \forall x{\in}$*ball a e*. $f\ x = c$
by (*intro has-derivative-zero-constant*)
(*auto simp*: *at-within-open*[*OF* - *open-ball*] *f convex-ball*)
with ‹$0{<}e$› **have** $\forall x{\in}$*ball a e*. $f\ a = f\ x$
by *auto*
then show *eventually* $(\lambda b.\ f\ a = f\ b)$ (*at a within s*)
using ‹$0{<}e$› **unfolding** *eventually-at-topological*
by (*intro exI*[*of* - *ball a e*]) *auto*
qed
qed

34.12 Differentiability of inverse function (most basic form)

lemma *has-derivative-inverse-basic*:
fixes $f :: 'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-vector*
assumes (*f has-derivative f'*) (*at* $(g\ y)$)
and *bounded-linear* g'
and $g' \circ f' = id$
and *continuous* (*at y*) *g*
and *open t*
and $y \in t$
and $\forall z{\in}t.\ f\ (g\ z) = z$

shows (*g has-derivative g'*) (*at y*)
proof −
interpret *f'*: *bounded-linear f'*
using *assms* **unfolding** *has-derivative-def* **by** *auto*
interpret *g'*: *bounded-linear g'*
using *assms* **by** *auto*
obtain *C* **where** *C*: *0 < C* $\bigwedge x$. *norm* (*g' x*) ≤ *norm x* ∗ *C*
using *bounded-linear.pos-bounded*[*OF assms*(*2*)] **by** *blast*
have *lem1*: ∀ *e*>*0*. ∃ *d*>*0*. ∀ *z*.
norm (*z* − *y*) < *d* ⟶ *norm* (*g z* − *g y* − *g'*(*z* − *y*)) ≤ *e* ∗ *norm* (*g z* − *g y*)
proof (*rule*, *rule*)
fix *e* :: *real*
assume *e* > *0*
with *C*(*1*) **have** ∗: *e* / *C* > *0* **by** *auto*
obtain *d0* **where** *d0*:
0 < *d0*
∀ *ya*. *norm* (*ya* − *g y*) < *d0* ⟶ *norm* (*f ya* − *f* (*g y*) − *f'* (*ya* − *g y*)) ≤ *e* / *C* ∗ *norm* (*ya* − *g y*)
using *assms*(*1*)
unfolding *has-derivative-at-alt*
using ∗ **by** *blast*
obtain *d1* **where** *d1*:
0 < *d1*
∀ *x*. *0* < *dist x y* ∧ *dist x y* < *d1* ⟶ *dist* (*g x*) (*g y*) < *d0*
using *assms*(*4*)
unfolding *continuous-at Lim-at*
using *d0*(*1*) **by** *blast*
obtain *d2* **where** *d2*:
0 < *d2*
∀ *ya*. *dist ya y* < *d2* ⟶ *ya* ∈ *t*
using *assms*(*5*)
unfolding *open-dist*
using *assms*(*6*) **by** *blast*
obtain *d* **where** *d*: *0* < *d d* < *d1 d* < *d2*
using *real-lbound-gt-zero*[*OF d1*(*1*) *d2*(*1*)] **by** *blast*
then show ∃ *d*>*0*. ∀ *z*. *norm* (*z* − *y*) < *d* ⟶ *norm* (*g z* − *g y* − *g'* (*z* − *y*)) ≤ *e* ∗ *norm* (*g z* − *g y*)
apply (*rule-tac x*=*d* **in** *exI*)
apply *rule*
defer
apply *rule*
apply *rule*
proof −
fix *z*
assume *as*: *norm* (*z* − *y*) < *d*
then have *z* ∈ *t*
using *d2 d* **unfolding** *dist-norm* **by** *auto*
have *norm* (*g z* − *g y* − *g'* (*z* − *y*)) ≤ *norm* (*g'* (*f* (*g z*) − *y* − *f'* (*g z* − *g y*)))

```
      unfolding g'.diff f'.diff
      unfolding assms(3)[unfolded o-def id-def, THEN fun-cong]
      unfolding assms(7)[rule-format,OF ‹z∈t›]
      apply (subst norm-minus-cancel[symmetric])
      apply auto
      done
    also have ... ≤ norm (f (g z) − y − f' (g z − g y)) * C
      by (rule C(2))
    also have ... ≤ (e / C) * norm (g z − g y) * C
      apply (rule mult-right-mono)
      apply (rule d0(2)[rule-format,unfolded assms(7)[rule-format,OF ‹y∈t›]])
      apply (cases z = y)
      defer
      apply (rule d1(2)[unfolded dist-norm,rule-format])
      using as d C d0
      apply auto
      done
    also have ... ≤ e * norm (g z − g y)
      using C by (auto simp add: field-simps)
    finally show norm (g z − g y − g' (z − y)) ≤ e * norm (g z − g y)
      by simp
  qed auto
 qed
 have *: (0::real) < 1 / 2
  by auto
 obtain d where d:
    0 < d
    ∀z. norm (z − y) < d ⟶ norm (g z − g y − g' (z − y)) ≤ 1 / 2 * norm
(g z − g y)
  using lem1 * by blast
 def B ≡ C * 2
 have B > 0
  unfolding B-def using C by auto
 have lem2: norm (g z − g y) ≤ B * norm (z − y) if z: norm(z − y) < d for z
 proof −
  have norm (g z − g y) ≤ norm(g' (z − y)) + norm ((g z − g y) − g'(z − y))
    by (rule norm-triangle-sub)
  also have ... ≤ norm (g' (z − y)) + 1 / 2 * norm (g z − g y)
    apply (rule add-left-mono)
    using d and z
    apply auto
    done
  also have ... ≤ norm (z − y) * C + 1 / 2 * norm (g z − g y)
    apply (rule add-right-mono)
    using C
    apply auto
    done
  finally show norm (g z − g y) ≤ B * norm (z − y)
    unfolding B-def
```

```
    by (auto simp add: field-simps)
  qed
  show ?thesis
    unfolding has-derivative-at-alt
    apply rule
    apply (rule assms)
    apply rule
    apply rule
  proof −
    fix e :: real
    assume e > 0
    then have ∗: e / B > 0 by (metis ‹B > 0› divide-pos-pos)
    obtain d′ where d′:
        0 < d′
       ∀ z. norm (z − y) < d′ ⟶ norm (g z − g y − g′ (z − y)) ≤ e / B ∗ norm
(g z − g y)
      using lem1 ∗ by blast
    obtain k where k: 0 < k k < d k < d′
      using real-lbound-gt-zero[OF d(1) d′(1)] by blast
    show ∃ d>0. ∀ ya. norm (ya − y) < d ⟶ norm (g ya − g y − g′ (ya − y))
≤ e ∗ norm (ya − y)
      apply (rule-tac x=k in exI)
      apply auto
    proof −
      fix z
      assume as: norm (z − y) < k
      then have norm (g z − g y − g′ (z − y)) ≤ e / B ∗ norm(g z − g y)
        using d′ k by auto
      also have ... ≤ e ∗ norm (z − y)
        unfolding times-divide-eq-left pos-divide-le-eq[OF ‹B>0›]
        using lem2[of z]
        using k as using ‹e > 0›
        by (auto simp add: field-simps)
      finally show norm (g z − g y − g′ (z − y)) ≤ e ∗ norm (z − y)
        by simp
    qed(insert k, auto)
  qed
qed
```

Simply rewrite that based on the domain point x.

```
lemma has-derivative-inverse-basic-x:
  fixes f :: 'a::real-normed-vector ⇒ 'b::real-normed-vector
  assumes (f has-derivative f′) (at x)
    and bounded-linear g′
    and g′ ∘ f′ = id
    and continuous (at (f x)) g
    and g (f x) = x
    and open t
    and f x ∈ t
```

```
    and ∀ y∈t. f (g y) = y
  shows (g has-derivative g') (at (f x))
  apply (rule has-derivative-inverse-basic)
  using assms
  apply auto
  done
```

This is the version in Dieudonne', assuming continuity of f and g.

```
lemma has-derivative-inverse-dieudonne:
  fixes f :: 'a::real-normed-vector ⇒ 'b::real-normed-vector
  assumes open s
    and open (f ' s)
    and continuous-on s f
    and continuous-on (f ' s) g
    and ∀ x∈s. g (f x) = x
    and x ∈ s
    and (f has-derivative f') (at x)
    and bounded-linear g'
    and g' ∘ f' = id
  shows (g has-derivative g') (at (f x))
  apply (rule has-derivative-inverse-basic-x[OF assms(7–9) - - assms(2)])
  using assms(3–6)
 unfolding continuous-on-eq-continuous-at[OF assms(1)] continuous-on-eq-continuous-at[OF
assms(2)]
  apply auto
  done
```

Here's the simplest way of not assuming much about g.

```
lemma has-derivative-inverse:
  fixes f :: 'a::real-normed-vector ⇒ 'b::real-normed-vector
  assumes compact s
    and x ∈ s
    and f x ∈ interior (f ' s)
    and continuous-on s f
    and ∀ y∈s. g (f y) = y
    and (f has-derivative f') (at x)
    and bounded-linear g'
    and g' ∘ f' = id
  shows (g has-derivative g') (at (f x))
proof –
  {
    fix y
    assume y ∈ interior (f ' s)
    then obtain x where x ∈ s and *: y = f x
      unfolding image-iff
      using interior-subset
      by auto
    have f (g y) = y
      unfolding * and assms(5)[rule-format,OF ‹x∈s›] ..
```

} note $* = this$
show *?thesis*
apply (*rule has-derivative-inverse-basic-x*[*OF assms*(*6*−*8*)])
apply (*rule continuous-on-interior*[*OF* - *assms*(*3*)])
apply (*rule continuous-on-inv*[*OF assms*(*4*,*1*)])
apply (*rule assms*(*2*,*5*) *assms*(*5*)[*rule-format*] *open-interior assms*(*3*))+
apply (*metis* $*$)
done
qed

34.13 Proving surjectivity via Brouwer fixpoint theorem

lemma *brouwer-surjective*:
fixes f :: $'n$::*euclidean-space* $\Rightarrow$ $'n$
assumes *compact* t
and *convex* t
and $t \neq \{\}$
and *continuous-on* t f
and $\forall x{\in}s.\ \forall y{\in}t.\ x + (y - f\ y) \in t$
and $x \in s$
shows $\exists y{\in}t.\ f\ y = x$
proof −
have $*$: $\bigwedge x\ y.\ f\ y = x \longleftrightarrow x + (y - f\ y) = y$
by (*auto simp add*: *algebra-simps*)
show *?thesis*
unfolding $*$
apply (*rule brouwer*[*OF assms*(*1*−*3*), *of* $\lambda y.\ x + (y - f\ y)$])
apply (*rule continuous-intros assms*)+
using *assms*(*4*−*6*)
apply *auto*
done
qed

lemma *brouwer-surjective-cball*:
fixes f :: $'n$::*euclidean-space* $\Rightarrow$ $'n$
assumes $e > 0$
and *continuous-on* (*cball a e*) f
and $\forall x{\in}s.\ \forall y{\in}cball\ a\ e.\ x + (y - f\ y) \in cball\ a\ e$
and $x \in s$
shows $\exists y{\in}cball\ a\ e.\ f\ y = x$
apply (*rule brouwer-surjective*)
apply (*rule compact-cball convex-cball*)+
unfolding *cball-eq-empty*
using *assms*
apply *auto*
done

See Sussmann: "Multidifferential calculus", Theorem 2.1.1

lemma *sussmann-open-mapping*:

```
  fixes f :: 'a::real-normed-vector ⇒ 'b::euclidean-space
  assumes open s
    and continuous-on s f
    and x ∈ s
    and (f has-derivative f') (at x)
    and bounded-linear g' f' ∘ g' = id
    and t ⊆ s
    and x ∈ interior t
  shows f x ∈ interior (f ' t)
proof −
  interpret f': bounded-linear f'
    using assms
    unfolding has-derivative-def
    by auto
  interpret g': bounded-linear g'
    using assms
    by auto
  obtain B where B: 0 < B ∀ x. norm (g' x) ≤ norm x * B
    using bounded-linear.pos-bounded[OF assms(5)] by blast
  hence *: 1 / (2 * B) > 0 by auto
  obtain e0 where e0:
      0 < e0
     ∀ y. norm (y − x) < e0 ⟶ norm (f y − f x − f' (y − x)) ≤ 1 / (2 * B) *
norm (y − x)
    using assms(4)
    unfolding has-derivative-at-alt
    using * by blast
  obtain e1 where e1: 0 < e1 cball x e1 ⊆ t
    using assms(8)
    unfolding mem-interior-cball
    by blast
  have *: 0 < e0 / B 0 < e1 / B using e0 e1 B by auto
  obtain e where e: 0 < e e < e0 / B e < e1 / B
    using real-lbound-gt-zero[OF *] by blast
  have ∀ z∈cball (f x) (e / 2). ∃ y∈cball (f x) e. f (x + g' (y − f x)) = z
    apply rule
    apply (rule brouwer-surjective-cball[where s=cball (f x) (e/2)])
    prefer 3
    apply rule
    apply rule
  proof−
    show continuous-on (cball (f x) e) (λy. f (x + g' (y − f x)))
      unfolding g'.diff
      apply (rule continuous-on-compose[of - - f, unfolded o-def])
      apply (rule continuous-intros linear-continuous-on[OF assms(5)])+
      apply (rule continuous-on-subset[OF assms(2)])
      apply rule
      apply (unfold image-iff)
      apply (erule bexE)
```

```
  proof−
    fix y z
    assume as: y ∈cball (f x) e z = x + (g' y − g' (f x))
    have dist x z = norm (g' (f x) − g' y)
      unfolding as(2) and dist-norm by auto
    also have ... ≤ norm (f x − y) * B
      unfolding g'.diff[symmetric]
      using B
      by auto
    also have ... ≤ e * B
      using as(1)[unfolded mem-cball dist-norm]
      using B
      by auto
    also have ... ≤ e1
      using e
      unfolding less-divide-eq
      using B
      by auto
    finally have z ∈ cball x e1
      unfolding mem-cball
      by force
    then show z ∈ s
      using e1 assms(7) by auto
  qed
next
  fix y z
  assume as: y ∈ cball (f x) (e / 2) z ∈ cball (f x) e
  have norm (g' (z − f x)) ≤ norm (z − f x) * B
    using B by auto
  also have ... ≤ e * B
    apply (rule mult-right-mono)
    using as(2)[unfolded mem-cball dist-norm] and B
    unfolding norm-minus-commute
    apply auto
    done
  also have ... < e0
    using e and B
    unfolding less-divide-eq
    by auto
  finally have *: norm (x + g' (z − f x) − x) < e0
    by auto
  have **: f x + f' (x + g' (z − f x) − x) = z
    using assms(6)[unfolded o-def id-def,THEN cong]
    by auto
  have norm (f x − (y + (z − f (x + g' (z − f x))))) ≤
      norm (f (x + g' (z − f x)) − z) + norm (f x − y)
    using norm-triangle-ineq[of f (x + g'(z − f x)) − z f x − y]
    by (auto simp add: algebra-simps)
  also have ... ≤ 1 / (B * 2) * norm (g' (z − f x)) + norm (f x − y)
```

```
    using e0(2)[rule-format, OF *]
    unfolding algebra-simps **
    by auto
  also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + e/2
    using as(1)[unfolded mem-cball dist-norm]
    by auto
  also have ... ≤ 1 / (B * 2) * B * norm (z - f x) + e/2
    using * and B
    by (auto simp add: field-simps)
  also have ... ≤ 1 / 2 * norm (z - f x) + e/2
    by auto
  also have ... ≤ e/2 + e/2
    apply (rule add-right-mono)
    using as(2)[unfolded mem-cball dist-norm]
    unfolding norm-minus-commute
    apply auto
    done
  finally show y + (z - f (x + g' (z - f x))) ∈ cball (f x) e
    unfolding mem-cball dist-norm
    by auto
qed (insert e, auto) note lem = this
show ?thesis
  unfolding mem-interior
  apply (rule-tac x=e/2 in exI)
  apply rule
  apply (rule divide-pos-pos)
  prefer 3
proof
  fix y
  assume y ∈ ball (f x) (e / 2)
  then have *: y ∈ cball (f x) (e / 2)
    by auto
  obtain z where z: z ∈ cball (f x) e f (x + g' (z - f x)) = y
    using lem * by blast
  then have norm (g' (z - f x)) ≤ norm (z - f x) * B
    using B
    by (auto simp add: field-simps)
  also have ... ≤ e * B
    apply (rule mult-right-mono)
    using z(1)
    unfolding mem-cball dist-norm norm-minus-commute
    using B
    apply auto
    done
  also have ... ≤ e1
    using e B unfolding less-divide-eq by auto
  finally have x + g'(z - f x) ∈ t
    apply -
    apply (rule e1(2)[unfolded subset-eq,rule-format])
```

```
    unfolding mem-cball dist-norm
    apply auto
    done
  then show y ∈ f ' t
    using z by auto
 qed (insert e, auto)
qed
```

Hence the following eccentric variant of the inverse function theorem. This has no continuity assumptions, but we do need the inverse function. We could put $f' \circ g = I$ but this happens to fit with the minimal linear algebra theory I've set up so far.

```
lemma right-inverse-linear:
 fixes f :: 'a::euclidean-space ⇒ 'a
 assumes lf: linear f
   and gf: f ∘ g = id
 shows linear g
proof −
 from gf have fi: surj f
   by (auto simp add: surj-def o-def id-def) metis
 from linear-surjective-isomorphism[OF lf fi]
 obtain h:: 'a ⇒ 'a where h: linear h ∀x. h (f x) = x ∀x. f (h x) = x
   by blast
 have h = g
   apply (rule ext)
   using gf h(2,3)
   apply (simp add: o-def id-def fun-eq-iff)
   apply metis
   done
 with h(1) show ?thesis by blast
qed

lemma has-derivative-inverse-strong:
 fixes f :: 'n::euclidean-space ⇒ 'n
 assumes open s
   and x ∈ s
   and continuous-on s f
   and ∀x∈s. g (f x) = x
   and (f has-derivative f') (at x)
   and f' ∘ g' = id
 shows (g has-derivative g') (at (f x))
proof −
 have linf: bounded-linear f'
   using assms(5) unfolding has-derivative-def by auto
 then have ling: bounded-linear g'
   unfolding linear-conv-bounded-linear[symmetric]
   apply −
   apply (rule right-inverse-linear)
   using assms(6)
```

```
    apply auto
    done
  moreover have g' ∘ f' = id
    using assms(6) linf ling
    unfolding linear-conv-bounded-linear[symmetric]
    using linear-inverse-left
    by auto
  moreover have ∗:∀ t⊆s. x ∈ interior t ⟶ f x ∈ interior (f ‘ t)
    apply clarify
    apply (rule sussmann-open-mapping)
    apply (rule assms ling)+
    apply auto
    done
  have continuous (at (f x)) g
    unfolding continuous-at Lim-at
  proof (rule, rule)
    fix e :: real
    assume e > 0
    then have f x ∈ interior (f ‘ (ball x e ∩ s))
      using ∗[rule-format,of ball x e ∩ s] ‹x ∈ s›
      by (auto simp add: interior-open[OF open-ball] interior-open[OF assms(1)])
    then obtain d where d: 0 < d ball (f x) d ⊆ f ‘ (ball x e ∩ s)
      unfolding mem-interior by blast
    show ∃ d>0. ∀ y. 0 < dist y (f x) ∧ dist y (f x) < d ⟶ dist (g y) (g (f x))
< e
      apply (rule-tac x=d in exI)
      apply rule
      apply (rule d(1))
      apply rule
      apply rule
    proof −
      fix y
      assume 0 < dist y (f x) ∧ dist y (f x) < d
      then have g y ∈ g ‘ f ‘ (ball x e ∩ s)
        using d(2)[unfolded subset-eq,THEN bspec[where x=y]]
        by (auto simp add: dist-commute)
      then have g y ∈ ball x e ∩ s
        using assms(4) by auto
      then show dist (g y) (g (f x)) < e
        using assms(4)[rule-format,OF ‹x ∈ s›]
        by (auto simp add: dist-commute)
    qed
  qed
  moreover have f x ∈ interior (f ‘ s)
    apply (rule sussmann-open-mapping)
    apply (rule assms ling)+
    using interior-open[OF assms(1)] and ‹x ∈ s›
    apply auto
    done
```

moreover have $f\ (g\ y) = y$ **if** $y \in$ *interior* $(f\ `\ s)$ **for** y
proof −
from *that* **have** $y \in f\ `\ s$
using *interior-subset* **by** *auto*
then obtain z **where** $z \in s\ y = f\ z$ **unfolding** *image-iff* **..**
then show *?thesis*
using *assms*(*4*) **by** *auto*
qed
ultimately show *?thesis* **using** *assms*
by (*metis has-derivative-inverse-basic-x open-interior*)
qed

A rewrite based on the other domain.

lemma *has-derivative-inverse-strong-x*:
fixes $f :: 'a$::*euclidean-space* $\Rightarrow 'a$
assumes *open s*
and $g\ y \in s$
and *continuous-on s f*
and $\forall x{\in}s.\ g\ (f\ x) = x$
and $(f$ *has-derivative* $f')\ (at\ (g\ y))$
and $f' \circ g' = id$
and $f\ (g\ y) = y$
shows $(g$ *has-derivative* $g')\ (at\ y)$
using *has-derivative-inverse-strong*[*OF assms*(*1*−*6*)]
unfolding *assms*(*7*)
by *simp*

On a region.

lemma *has-derivative-inverse-on*:
fixes $f :: 'n$::*euclidean-space* $\Rightarrow 'n$
assumes *open s*
and $\forall x{\in}s.\ (f$ *has-derivative* $f'(x))\ (at\ x)$
and $\forall x{\in}s.\ g\ (f\ x) = x$
and $f'\ x \circ g'\ x = id$
and $x \in s$
shows $(g$ *has-derivative* $g'(x))\ (at\ (f\ x))$
apply (*rule has-derivative-inverse-strong*[**where** $g'{=}g'\ x$ **and** $f{=}f$])
apply (*rule assms*)+
unfolding *continuous-on-eq-continuous-at*[*OF assms*(*1*)]
apply *rule*
apply (*rule differentiable-imp-continuous-within*)
unfolding *differentiable-def*
using *assms*
apply *auto*
done

Invertible derivative continous at a point implies local injectivity. It's only for this we need continuity of the derivative, except of course if we want the fact that the inverse derivative is also continuous. So if we know for some

other reason that the inverse function exists, it's OK.

lemma *has-derivative-locally-injective*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'm::euclidean\text{-}space$
assumes $a \in s$
and *open s*
and *bounded-linear* g'
and $g' \circ f' a = id$
and $\forall x{\in}s.\ (f \text{ has-derivative } f' x)\ (at\ x)$
and $\forall e{>}0.\ \exists d{>}0.\ \forall x.\ dist\ a\ x < d \longrightarrow onorm\ (\lambda v.\ f' x\ v - f' a\ v) < e$
obtains t **where** $a \in t$ *open* t $\forall x{\in}t.\ \forall x'{\in}t.\ f x' = f x \longrightarrow x' = x$
proof −
interpret *bounded-linear* g'
using *assms* **by** *auto*
note $f'g' = assms(4)[unfolded\ id\text{-}def\ o\text{-}def, THEN\ cong]$
have $g'\ (f'\ a\ (\sum Basis)) = (\sum Basis)\ (\sum Basis) \neq (0::'n)$
defer
apply (*subst euclidean-eq-iff*)
using $f'g'$
apply *auto*
done
then have $*$: $0 < onorm\ g'$
unfolding $onorm\text{-}pos\text{-}lt[OF\ assms(3)]$
by *fastforce*
def $k \equiv 1\ /\ onorm\ g'\ /\ 2$
have $*$: $k > 0$
unfolding *k-def* **using** $*$ **by** *auto*
obtain *d1* **where** *d1*:
$0 < d1$
$\bigwedge x.\ dist\ a\ x < d1 \Longrightarrow onorm\ (\lambda v.\ f' x\ v - f' a\ v) < k$
using $assms(6)$ $*$ **by** *blast*
from ‹*open s*› **obtain** *d2* **where** $d2 > 0$ *ball a d2* $\subseteq s$
using ‹$a{\in}s$› **..**
obtain *d2* **where** $d2 > 0$ *ball a d2* $\subseteq s$
using $assms(2,1)$ **..**
obtain *d2* **where** *d2*: $0 < d2$ *ball a d2* $\subseteq s$
using $assms(2)$
unfolding *open-contains-ball*
using ‹$a{\in}s$› **by** *blast*
obtain *d* **where** *d*: $0 < d\ d < d1\ d < d2$
using $real\text{-}lbound\text{-}gt\text{-}zero[OF\ d1(1)\ d2(1)]$ **by** *blast*
show *?thesis*
proof
show $a \in ball\ a\ d$
using *d* **by** *auto*
show $\forall x{\in}ball\ a\ d.\ \forall x'{\in}ball\ a\ d.\ f x' = f x \longrightarrow x' = x$
proof (*intro strip*)
fix *x y*
assume *as*: $x \in ball\ a\ d\ y \in ball\ a\ d\ f x = f y$
def $ph \equiv \lambda w.\ w - g'\ (f\ w - f\ x)$

```
    have ph':ph = g' ∘ (λw. f' a w − (f w − f x))
      unfolding ph-def o-def
      unfolding diff
      using f'g'
      by (auto simp add: algebra-simps)
    have norm (ph x − ph y) ≤ (1 / 2) * norm (x − y)
      apply (rule differentiable-bound[OF convex-ball - - as(1−2), where f'=λx
v. v − g'(f' x v)])
      apply (rule-tac[!] ballI)
    proof −
      fix u
      assume u: u ∈ ball a d
      then have u ∈ s
        using d d2 by auto
      have *: (λv. v − g' (f' u v)) = g' ∘ (λw. f' a w − f' u w)
        unfolding o-def and diff
        using f'g' by auto
      show (ph has-derivative (λv. v − g' (f' u v))) (at u within ball a d)
        unfolding ph' *
        apply (simp add: comp-def)
        apply (rule bounded-linear.has-derivative[OF assms(3)])
        apply (rule derivative-intros)
        defer
        apply (rule has-derivative-sub[where g'=λx.0,unfolded diff-0-right])
        apply (rule has-derivative-at-within)
        using assms(5) and ‹u ∈ s› ‹a ∈ s›
       apply (auto intro!: derivative-intros bounded-linear.has-derivative[of - λx.
x] has-derivative-bounded-linear)
        done
      have **: bounded-linear (λx. f' u x − f' a x) bounded-linear (λx. f' a x −
f' u x)
        apply (rule-tac[!] bounded-linear-sub)
        apply (rule-tac[!] has-derivative-bounded-linear)
        using assms(5) ‹u ∈ s› ‹a ∈ s›
        apply auto
        done
      have onorm (λv. v − g' (f' u v)) ≤ onorm g' * onorm (λw. f' a w − f' u
w)
        unfolding *
        apply (rule onorm-compose)
        apply (rule assms(3) **)+
        done
      also have ... ≤ onorm g' * k
        apply (rule mult-left-mono)
        using d1(2)[of u]
        using onorm-neg[where f=λx. f' u x − f' a x]
        using d and u and onorm-pos-le[OF assms(3)]
        apply (auto simp add: algebra-simps)
        done
```

```
      also have ... ≤ 1 / 2
        unfolding k-def by auto
      finally show onorm (λv. v − g' (f' u v)) ≤ 1 / 2 .
    qed
    moreover have norm (ph y − ph x) = norm (y − x)
      apply (rule arg-cong[where f=norm])
      unfolding ph-def
      using diff
      unfolding as
      apply auto
      done
    ultimately show x = y
      unfolding norm-minus-commute by auto
  qed
 qed auto
qed
```

34.14 Uniformly convergent sequence of derivatives

```
lemma has-derivative-sequence-lipschitz-lemma:
 fixes f :: nat ⇒ 'a::real-normed-vector ⇒ 'b::real-normed-vector
 assumes convex s
   and ∀n. ∀x∈s. ((f n) has-derivative (f' n x)) (at x within s)
   and ∀n≥N. ∀x∈s. ∀h. norm (f' n x h − g' x h) ≤ e ∗ norm h
   and 0 ≤ e
 shows ∀m≥N. ∀n≥N. ∀x∈s. ∀y∈s. norm ((f m x − f n x) − (f m y − f n y))
≤ 2 ∗ e ∗ norm (x − y)
proof rule+
 fix m n x y
 assume as: N ≤ m N ≤ n x ∈ s y ∈ s
 show norm ((f m x − f n x) − (f m y − f n y)) ≤ 2 ∗ e ∗ norm (x − y)
   apply (rule differentiable-bound[where f'=λx h. f' m x h − f' n x h, OF
assms(1) - - as(3−4)])
   apply (rule-tac[!] ballI)
 proof −
   fix x
   assume x ∈ s
   show ((λa. f m a − f n a) has-derivative (λh. f' m x h − f' n x h)) (at x
within s)
     by (rule derivative-intros assms(2)[rule-format] ‹x∈s›)+
   show onorm (λh. f' m x h − f' n x h) ≤ 2 ∗ e
   proof (rule onorm-bound)
     fix h
     have norm (f' m x h − f' n x h) ≤ norm (f' m x h − g' x h) + norm (f' n
x h − g' x h)
       using norm-triangle-ineq[of f' m x h − g' x h − f' n x h + g' x h]
       unfolding norm-minus-commute
       by (auto simp add: algebra-simps)
     also have ... ≤ e ∗ norm h + e ∗ norm h
```

```
      using assms(3)[rule-format,OF ‹N ≤ m› ‹x ∈ s›, of h]
      using assms(3)[rule-format,OF ‹N ≤ n› ‹x ∈ s›, of h]
      by (auto simp add: field-simps)
    finally show norm (f' m x h − f' n x h) ≤ 2 * e * norm h
      by auto
  qed (simp add: ‹0 ≤ e›)
 qed
qed

lemma has-derivative-sequence-lipschitz:
  fixes f :: nat ⇒ 'a::real-normed-vector ⇒ 'b::real-normed-vector
  assumes convex s
    and ∀ n. ∀ x∈s. ((f n) has-derivative (f' n x)) (at x within s)
    and ∀ e>0. ∃ N. ∀ n≥N. ∀ x∈s. ∀ h. norm (f' n x h − g' x h) ≤ e * norm h
  shows ∀ e>0. ∃ N. ∀ m≥N. ∀ n≥N. ∀ x∈s. ∀ y∈s.
    norm ((f m x − f n x) − (f m y − f n y)) ≤ e * norm (x − y)
proof (rule, rule)
  fix e :: real
  assume e > 0
  then have *: 2 * (1/2* e) = e 1/2 * e >0
    by auto
  obtain N where ∀ n≥N. ∀ x∈s. ∀ h. norm (f' n x h − g' x h) ≤ 1 / 2 * e *
norm h
    using assms(3) *(2) by blast
  then show ∃ N. ∀ m≥N. ∀ n≥N. ∀ x∈s. ∀ y∈s. norm (f m x − f n x − (f m y
− f n y)) ≤ e * norm (x − y)
    apply (rule-tac x=N in exI)
   apply (rule has-derivative-sequence-lipschitz-lemma[where e=1/2 *e, unfolded
*])
    using assms ‹e > 0›
    apply auto
    done
qed

lemma has-derivative-sequence:
  fixes f :: nat ⇒ 'a::real-normed-vector ⇒ 'b::banach
  assumes convex s
    and ∀ n. ∀ x∈s. ((f n) has-derivative (f' n x)) (at x within s)
    and ∀ e>0. ∃ N. ∀ n≥N. ∀ x∈s. ∀ h. norm (f' n x h − g' x h) ≤ e * norm h
    and x0 ∈ s
    and ((λn. f n x0) ⟶ l) sequentially
  shows ∃ g. ∀ x∈s. ((λn. f n x) ⟶ g x) sequentially ∧ (g has-derivative g'(x))
(at x within s)
proof −
  have lem1: ∀ e>0. ∃ N. ∀ m≥N. ∀ n≥N. ∀ x∈s. ∀ y∈s.
      norm ((f m x − f n x) − (f m y − f n y)) ≤ e * norm (x − y)
    using assms(1,2,3) by (rule has-derivative-sequence-lipschitz)
  have ∃ g. ∀ x∈s. ((λn. f n x) ⟶ g x) sequentially
    apply (rule bchoice)
```

```
  unfolding convergent-eq-cauchy
proof
  fix x
  assume x ∈ s
  show Cauchy (λn. f n x)
  proof (cases x = x0)
    case True
    then show ?thesis
      using LIMSEQ-imp-Cauchy[OF assms(5)] by auto
  next
    case False
    show ?thesis
      unfolding Cauchy-def
    proof (rule, rule)
      fix e :: real
      assume e > 0
      hence *: e / 2 > 0 e / 2 / norm (x - x0) > 0 using False by auto
      obtain M where M: ∀ m≥M. ∀ n≥M. dist (f m x0) (f n x0) < e / 2
        using LIMSEQ-imp-Cauchy[OF assms(5)]
        unfolding Cauchy-def
        using *(1) by blast
      obtain N where N:
        ∀ m≥N. ∀ n≥N.
          ∀ xa∈s. ∀ y∈s. norm (f m xa - f n xa - (f m y - f n y)) ≤
            e / 2 / norm (x - x0) * norm (xa - y)
      using lem1 *(2) by blast
      show ∃ M. ∀ m≥M. ∀ n≥M. dist (f m x) (f n x) < e
        apply (rule-tac x=max M N in exI)
      proof rule+
        fix m n
        assume as: max M N ≤m max M N≤n
        have dist (f m x) (f n x) ≤
            norm (f m x0 - f n x0) + norm (f m x - f n x - (f m x0 - f n x0))
          unfolding dist-norm
          by (rule norm-triangle-sub)
        also have ... ≤ norm (f m x0 - f n x0) + e / 2
          using N[rule-format,OF - - ‹x∈s› ‹x0∈s›, of m n] and as and False
          by auto
        also have ... < e / 2 + e / 2
          apply (rule add-strict-right-mono)
          using as and M[rule-format]
          unfolding dist-norm
          apply auto
          done
        finally show dist (f m x) (f n x) < e
          by auto
      qed
    qed
  qed
```

qed
then obtain g **where** g: $\forall x{\in}s.\ (\lambda n.\ f\ n\ x) \longrightarrow g\ x$ **..**
have *lem2*: $\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm\ ((f\ n\ x - f\ n\ y) - (g\ x - g\ y)) \leq e * norm\ (x - y)$
proof (*rule*, *rule*)
fix e :: *real*
assume $*$: $e > 0$
obtain N **where**
N: $\forall m{\geq}N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm\ (f\ m\ x - f\ n\ x - (f\ m\ y - f\ n\ y)) \leq e * norm\ (x - y)$
using *lem1* $*$ **by** *blast*
show $\exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall y{\in}s.\ norm\ (f\ n\ x - f\ n\ y - (g\ x - g\ y)) \leq e * norm\ (x - y)$
apply (*rule-tac* $x{=}N$ **in** *exI*)
proof *rule*+
fix $n\ x\ y$
assume *as*: $N \leq n\ x \in s\ y \in s$
have $((\lambda m.\ norm\ (f\ n\ x - f\ n\ y - (f\ m\ x - f\ m\ y))) \longrightarrow norm\ (f\ n\ x - f\ n\ y - (g\ x - g\ y)))$ *sequentially*
by (*intro tendsto-intros g*[*rule-format*] *as*)
moreover have *eventually* $(\lambda m.\ norm\ (f\ n\ x - f\ n\ y - (f\ m\ x - f\ m\ y)) \leq e * norm\ (x - y))$ *sequentially*
unfolding *eventually-sequentially*
apply (*rule-tac* $x{=}N$ **in** *exI*)
apply *rule*
apply *rule*
proof −
fix m
assume $N \leq m$
then show $norm\ (f\ n\ x - f\ n\ y - (f\ m\ x - f\ m\ y)) \leq e * norm\ (x - y)$
using N[*rule-format*, *of n m x y*] **and** *as*
by (*auto simp add*: *algebra-simps*)
qed
ultimately show $norm\ (f\ n\ x - f\ n\ y - (g\ x - g\ y)) \leq e * norm\ (x - y)$
by (*rule tendsto-ge-const*[*OF trivial-limit-sequentially*])
qed
qed
have $\forall x{\in}s.\ ((\lambda n.\ f\ n\ x) \longrightarrow g\ x)\ sequentially \wedge (g\ has\text{-}derivative\ g'\ x)\ (at\ x\ within\ s)$
unfolding *has-derivative-within-alt2*
proof (*intro ballI conjI*)
fix x
assume $x \in s$
then show $((\lambda n.\ f\ n\ x) \longrightarrow g\ x)\ sequentially$
by (*simp add*: *g*)
have *lem3*: $\forall u.\ ((\lambda n.\ f'\ n\ x\ u) \longrightarrow g'\ x\ u)\ sequentially$
unfolding *filterlim-def le-nhds-metric-le eventually-filtermap dist-norm*
proof (*intro allI impI*)
fix u

```
    fix e :: real
    assume e > 0
    show eventually (λn. norm (f' n x u - g' x u) ≤ e) sequentially
    proof (cases u = 0)
      case True
      have eventually (λn. norm (f' n x u - g' x u) ≤ e * norm u) sequentially
        using assms(3)[folded eventually-sequentially] and ‹0 < e› and ‹x ∈ s›
        by (fast elim: eventually-mono)
      then show ?thesis
        using ‹u = 0› and ‹0 < e› by (auto elim: eventually-mono)
    next
      case False
      with ‹0 < e› have 0 < e / norm u by simp
      then have eventually (λn. norm (f' n x u - g' x u) ≤ e / norm u * norm
u) sequentially
        using assms(3)[folded eventually-sequentially] and ‹x ∈ s›
        by (fast elim: eventually-mono)
      then show ?thesis
        using ‹u ≠ 0› by simp
    qed
  qed
  show bounded-linear (g' x)
  proof
    fix x' y z :: 'a
    fix c :: real
  note lin = assms(2)[rule-format,OF ‹x∈s›,THEN has-derivative-bounded-linear]
    show g' x (c *R x') = c *R g' x x'
      apply (rule tendsto-unique[OF trivial-limit-sequentially])
      apply (rule lem3[rule-format])
      unfolding lin[THEN bounded-linear.linear, THEN linear-cmul]
      apply (intro tendsto-intros)
      apply (rule lem3[rule-format])
      done
    show g' x (y + z) = g' x y + g' x z
      apply (rule tendsto-unique[OF trivial-limit-sequentially])
      apply (rule lem3[rule-format])
      unfolding lin[THEN bounded-linear.linear, THEN linear-add]
      apply (rule tendsto-add)
      apply (rule lem3[rule-format])+
      done
    obtain N where N: ∀h. norm (f' N x h - g' x h) ≤ 1 * norm h
      using assms(3) ‹x ∈ s› by (fast intro: zero-less-one)
    have bounded-linear (f' N x)
      using assms(2) ‹x ∈ s› by fast
    from bounded-linear.bounded [OF this]
    obtain K where K: ∀h. norm (f' N x h) ≤ norm h * K ..
    {
      fix h
      have norm (g' x h) = norm (f' N x h - (f' N x h - g' x h))
```

by *simp*
also have $\ldots \leq$ *norm* $(f'\ N\ x\ h) +$ *norm* $(f'\ N\ x\ h - g'\ x\ h)$
by (*rule norm-triangle-ineq4*)
also have $\ldots \leq$ *norm* $h * K + 1 *$ *norm* h
using $N\ K$ **by** (*fast intro*: *add-mono*)
finally have *norm* $(g'\ x\ h) \leq$ *norm* $h * (K + 1)$
by (*simp add*: *ring-distribs*)
}
then show $\exists K.\ \forall h.$ *norm* $(g'\ x\ h) \leq$ *norm* $h * K$ **by** *fast*
qed
show $\forall e{>}0.$ *eventually* $(\lambda y.$ *norm* $(g\ y - g\ x - g'\ x\ (y - x)) \leq e *$ *norm* $(y - x))$ (*at x within s*)
proof (*rule*, *rule*)
fix e :: *real*
assume $e > 0$
then have $*$: $e\ /\ 3 > 0$
by *auto*
obtain *N1* **where** *N1*: $\forall n{\geq}N1.\ \forall x{\in}s.\ \forall h.$ *norm* $(f'\ n\ x\ h - g'\ x\ h) \leq e\ /\ 3 *$ *norm* h
using *assms*(*3*) $*$ **by** *blast*
obtain *N2* **where**
N2: $\forall n{\geq}N2.\ \forall x{\in}s.\ \forall y{\in}s.$ *norm* $(f\ n\ x - f\ n\ y - (g\ x - g\ y)) \leq e\ /\ 3 *$ *norm* $(x - y)$
using *lem2* $*$ **by** *blast*
let *?N* = *max N1 N2*
have *eventually* $(\lambda y.$ *norm* $(f\ ?N\ y - f\ ?N\ x - f'\ ?N\ x\ (y - x)) \leq e\ /\ 3 *$ *norm* $(y - x))$ (*at x within s*)
using *assms*(*2*)[*unfolded has-derivative-within-alt2*] **and** ‹$x \in s$› **and** $*$ **by** *fast*
moreover have *eventually* $(\lambda y.\ y \in s)$ (*at x within s*)
unfolding *eventually-at* **by** (*fast intro*: *zero-less-one*)
ultimately show $\forall_F\ y$ *in at x within s.* *norm* $(g\ y - g\ x - g'\ x\ (y - x)) \leq e *$ *norm* $(y - x)$
proof (*rule eventually-elim2*)
fix y
assume $y \in s$
assume *norm* $(f\ ?N\ y - f\ ?N\ x - f'\ ?N\ x\ (y - x)) \leq e\ /\ 3 *$ *norm* $(y - x)$
moreover have *norm* $(g\ y - g\ x - (f\ ?N\ y - f\ ?N\ x)) \leq e\ /\ 3 *$ *norm* $(y - x)$
using *N2*[*rule-format*, *OF* - ‹$y \in s$› ‹$x \in s$›]
by (*simp add*: *norm-minus-commute*)
ultimately have *norm* $(g\ y - g\ x - f'\ ?N\ x\ (y - x)) \leq 2 * e\ /\ 3 *$ *norm* $(y - x)$
using *norm-triangle-le*[*of* $g\ y - g\ x - (f\ ?N\ y - f\ ?N\ x)\ f\ ?N\ y - f\ ?N\ x - f'\ ?N\ x\ (y - x)\ 2 * e\ /\ 3 *$ *norm* $(y - x)$]
by (*auto simp add*: *algebra-simps*)
moreover
have *norm* $(f'\ ?N\ x\ (y - x) - g'\ x\ (y - x)) \leq e\ /\ 3 *$ *norm* $(y - x)$

using *N1* ‹$x \in s$› **by** *auto*
ultimately show *norm* $(g\ y - g\ x - g'\ x\ (y - x)) \leq e * norm\ (y - x)$
using *norm-triangle-le*[*of* $g\ y - g\ x - f'\ (max\ N1\ N2)\ x\ (y - x)\ f'\ (max\ N1\ N2)\ x\ (y - x) - g'\ x\ (y - x)$]
by (*auto simp add*: *algebra-simps*)
qed
qed
qed
then show *?thesis* **by** *fast*
qed

Can choose to line up antiderivatives if we want.

lemma *has-antiderivative-sequence*:
fixes $f :: nat \Rightarrow 'a$::*real-normed-vector* $\Rightarrow 'b$::*banach*
assumes *convex s*
and $\forall n.\ \forall x{\in}s.\ ((f\ n)$ *has-derivative* $(f'\ n\ x))$ *(at x within s)*
and $\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall h.\ norm\ (f'\ n\ x\ h - g'\ x\ h) \leq e * norm\ h$
shows $\exists g.\ \forall x{\in}s.\ (g$ *has-derivative* $g'\ x)$ *(at x within s)*
proof (*cases* $s = \{\}$)
case *False*
then obtain *a* **where** $a \in s$
by *auto*
have $*$: $\bigwedge P\ Q.\ \exists g.\ \forall x{\in}s.\ P\ g\ x \wedge Q\ g\ x \Longrightarrow \exists g.\ \forall x{\in}s.\ Q\ g\ x$
by *auto*
show *?thesis*
apply (*rule* $*$)
apply (*rule has-derivative-sequence*[*OF assms*(*1*) - *assms*(*3*), *of* $\lambda n\ x.\ f\ n\ x + (f\ 0\ a - f\ n\ a)$])
apply (*metis assms*(*2*) *has-derivative-add-const*)
apply (*rule* ‹$a \in s$›)
apply *auto*
done
qed *auto*

lemma *has-antiderivative-limit*:
fixes $g' :: 'a$::*real-normed-vector* $\Rightarrow 'a \Rightarrow 'b$::*banach*
assumes *convex s*
and $\forall e{>}0.\ \exists f\ f'.\ \forall x{\in}s.$
$(f$ *has-derivative* $(f'\ x))$ *(at x within s)* $\wedge$ $(\forall h.\ norm\ (f'\ x\ h - g'\ x\ h) \leq e * norm\ h)$
shows $\exists g.\ \forall x{\in}s.\ (g$ *has-derivative* $g'\ x)$ *(at x within s)*
proof −
have $*$: $\forall n.\ \exists f\ f'.\ \forall x{\in}s.$
$(f$ *has-derivative* $(f'\ x))$ *(at x within s)* $\wedge$
$(\forall h.\ norm(f'\ x\ h - g'\ x\ h) \leq inverse\ (real\ (Suc\ n)) * norm\ h)$
by (*simp add*: *assms*(*2*))
obtain *f* **where**
$*$: $\forall x.\ \exists f'.\ \forall xa{\in}s.\ (f\ x$ *has-derivative* $f'\ xa)$ *(at xa within s)* $\wedge$
$(\forall h.\ norm\ (f'\ xa\ h - g'\ xa\ h) \leq inverse\ (real\ (Suc\ x)) * norm\ h)$

```
    using ∗[THEN choice] ..
  obtain f' where
    f: ∀x. ∀xa∈s. (f x has-derivative f' x xa) (at xa within s) ∧
      (∀h. norm (f' x xa h − g' xa h) ≤ inverse (real (Suc x)) ∗ norm h)
    using ∗[THEN choice] ..
  show ?thesis
    apply (rule has-antiderivative-sequence[OF assms(1), of f f'])
    defer
    apply rule
    apply rule
  proof −
    fix e :: real
    assume e > 0
    obtain N where N: inverse (real (Suc N)) < e
      using reals-Archimedean[OF ‹e>0›] ..
    show ∃N. ∀n≥N. ∀x∈s. ∀h. norm (f' n x h − g' x h) ≤ e ∗ norm h
      apply (rule-tac x=N in exI)
      apply rule
      apply rule
      apply rule
      apply rule
    proof −
      fix n x h
      assume n: N ≤ n and x: x ∈ s
      have ∗: inverse (real (Suc n)) ≤ e
        apply (rule order-trans[OF - N[THEN less-imp-le]])
        using n
        apply (auto simp add: field-simps)
        done
      show norm (f' n x h − g' x h) ≤ e ∗ norm h
       using f[rule-format,THEN conjunct2, OF x, of n, THEN spec[where x=h]]
        apply (rule order-trans)
        using N ∗
        apply (cases h = 0)
        apply auto
        done
    qed
  qed (insert f, auto)
qed
```

34.15 Differentiation of a series

```
lemma has-derivative-series:
  fixes f :: nat ⇒ 'a::real-normed-vector ⇒ 'b::banach
  assumes convex s
    and ⋀n x. x ∈ s ⟹ ((f n) has-derivative (f' n x)) (at x within s)
    and ∀e>0. ∃N. ∀n≥N. ∀x∈s. ∀h. norm (setsum (λi. f' i x h) {..<n} − g'
x h) ≤ e ∗ norm h
    and x ∈ s
```

and $(\lambda n.\ f\ n\ x)$ *sums* l
shows $\exists g.\ \forall x{\in}s.\ (\lambda n.\ f\ n\ x)$ *sums* $(g\ x) \wedge (g$ *has-derivative* $g'\ x)$ $(at\ x\ within\ s)$
unfolding *sums-def*
apply (*rule has-derivative-sequence*[*OF assms*(*1*) - *assms*(*3*)])
apply (*metis assms*(*2*) *has-derivative-setsum*)
using *assms*(*4*−*5*)
unfolding *sums-def*
apply *auto*
done

lemma *has-field-derivative-series*:
fixes $f :: nat \Rightarrow ('a :: \{real\text{-}normed\text{-}field, banach\}) \Rightarrow 'a$
assumes *convex s*
assumes $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n$ *has-field-derivative* $f'\ n\ x)$ $(at\ x\ within\ s)$
assumes *uniform-limit s* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)\ g'$ *sequentially*
assumes $x0 \in s$ *summable* $(\lambda n.\ f\ n\ x0)$
shows $\exists g.\ \forall x{\in}s.\ (\lambda n.\ f\ n\ x)$ *sums* $g\ x \wedge (g$ *has-field-derivative* $g'\ x)$ $(at\ x\ within\ s)$
unfolding *has-field-derivative-def*
proof (*rule has-derivative-series*)
show $\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall h.\ norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) \leq e * norm\ h$
proof (*intro allI impI*)
fix $e :: real$ **assume** $e > 0$
with *assms*(*3*) **obtain** N **where** N: $\bigwedge n\ x.\ n \geq N \Longrightarrow x \in s \Longrightarrow norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) < e$
unfolding *uniform-limit-iff eventually-at-top-linorder dist-norm* **by** *blast*
{
fix $n :: nat$ **and** $x\ h :: 'a$ **assume** nx: $n \geq N\ x \in s$
have $norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) = norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) * norm\ h$
by (*simp add*: *norm-mult* [*symmetric*] *ring-distribs setsum-left-distrib*)
also from N[*OF nx*] **have** $norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) \leq e$ **by** *simp*
hence $norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) * norm\ h \leq e * norm\ h$
by (*intro mult-right-mono*) *simp-all*
finally have $norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) \leq e * norm\ h$ **.**
}
thus $\exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall h.\ norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) \leq e * norm\ h$ **by** *blast*
qed
qed (*insert assms*, *auto simp*: *has-field-derivative-def*)

lemma *has-field-derivative-series'*:
fixes $f :: nat \Rightarrow ('a :: \{real\text{-}normed\text{-}field, banach\}) \Rightarrow 'a$
assumes *convex s*
assumes $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n$ *has-field-derivative* $f'\ n\ x)$ $(at\ x\ within\ s)$
assumes *uniformly-convergent-on s* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$
assumes $x0 \in s$ *summable* $(\lambda n.\ f\ n\ x0)$ $x \in interior\ s$

shows *summable* $(\lambda n.\ f\ n\ x)$ $((\lambda x.\ \sum n.\ f\ n\ x)$ *has-field-derivative* $(\sum n.\ f'\ n\ x))$ $(at\ x)$
proof −
from ‹$x \in$ *interior* s› **have** $x \in s$ **using** *interior-subset* **by** *blast*
def $g' \equiv \lambda x.\ \sum i.\ f'\ i\ x$
from *assms*(3) **have** *uniform-limit* s $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$ g' *sequentially*
by (*simp add*: *uniformly-convergent-uniform-limit-iff suminf-eq-lim g'-def*)
from *has-field-derivative-series*[*OF assms*(1,2) *this assms*(4,5)] **obtain** g **where** g:
$\bigwedge x.\ x \in s \Longrightarrow (\lambda n.\ f\ n\ x)$ *sums* $g\ x$
$\bigwedge x.\ x \in s \Longrightarrow (g$ *has-field-derivative* $g'\ x)$ $(at\ x$ *within* $s)$ **by** *blast*
from $g(1)$[*OF* ‹$x \in s$›] **show** *summable* $(\lambda n.\ f\ n\ x)$ **by** (*simp add*: *sums-iff*)
from $g(2)$[*OF* ‹$x \in s$›] ‹$x \in$ *interior* s› **have** $(g$ *has-field-derivative* $g'\ x)$ $(at\ x)$
by (*simp add*: *at-within-interior*[*of x s*])
also have $(g$ *has-field-derivative* $g'\ x)$ $(at\ x) \longleftrightarrow$
$((\lambda x.\ \sum n.\ f\ n\ x)$ *has-field-derivative* $g'\ x)$ $(at\ x)$
using *eventually-nhds-in-nhd*[*OF* ‹$x \in$ *interior* s›] *interior-subset*[*of s*] $g(1)$
by (*intro DERIV-cong-ev*) (*auto elim*!: *eventually-mono simp*: *sums-iff*)
finally show $((\lambda x.\ \sum n.\ f\ n\ x)$ *has-field-derivative* $g'\ x)$ $(at\ x)$ **.**
qed

lemma *differentiable-series*:
fixes f :: *nat* ⇒ ('*a* :: {*real-normed-field*,*banach*}) ⇒ '*a*
assumes *convex s open s*
assumes $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n$ *has-field-derivative* $f'\ n\ x)$ $(at\ x)$
assumes *uniformly-convergent-on* s $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$
assumes *x0* $\in s$ *summable* $(\lambda n.\ f\ n\ x0)$ **and** x: $x \in s$
shows *summable* $(\lambda n.\ f\ n\ x)$ **and** $(\lambda x.\ \sum n.\ f\ n\ x)$ *differentiable* $(at\ x)$
proof −
from *assms*(4) **obtain** g' **where** *A*: *uniform-limit* s $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$ g' *sequentially*
unfolding *uniformly-convergent-on-def* **by** *blast*
from x **and** ‹*open* s› **have** s: *at* x *within* $s = at\ x$ **by** (*rule at-within-open*)
have $\exists g.\ \forall x{\in}s.\ (\lambda n.\ f\ n\ x)$ *sums* $g\ x \wedge (g$ *has-field-derivative* $g'\ x)$ $(at\ x$ *within* $s)$
by (*intro has-field-derivative-series*[*of s f f' g' x0*] *assms A has-field-derivative-at-within*)
then obtain g **where** g: $\bigwedge x.\ x \in s \Longrightarrow (\lambda n.\ f\ n\ x)$ *sums* $g\ x$
$\bigwedge x.\ x \in s \Longrightarrow (g$ *has-field-derivative* $g'\ x)$ $(at\ x$ *within* $s)$ **by** *blast*
from g[*OF* x] **show** *summable* $(\lambda n.\ f\ n\ x)$ **by** (*auto simp*: *summable-def*)
from $g(2)$[*OF* x] **have** g': $(g$ *has-derivative op* $*$ $(g'\ x))$ $(at\ x)$
by (*simp add*: *has-field-derivative-def s*)
have $((\lambda x.\ \sum n.\ f\ n\ x)$ *has-derivative op* $*$ $(g'\ x))$ $(at\ x)$
by (*rule has-derivative-transform-within-open*[*OF* g' ‹*open* s› x])
(*insert g*, *auto simp*: *sums-iff*)
thus $(\lambda x.\ \sum n.\ f\ n\ x)$ *differentiable* $(at\ x)$ **unfolding** *differentiable-def*
by (*auto simp*: *summable-def differentiable-def has-field-derivative-def*)
qed

lemma *differentiable-series'*:

fixes $f :: nat \Rightarrow ('a :: \{real\text{-}normed\text{-}field, banach\}) \Rightarrow 'a$
assumes *convex s open s*
assumes $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n\ has\text{-}field\text{-}derivative\ f'\ n\ x)\ (at\ x)$
assumes *uniformly-convergent-on s* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$
assumes *x0* $\in$ *s summable* $(\lambda n.\ f\ n\ x0)$
shows $(\lambda x.\ \sum n.\ f\ n\ x)$ *differentiable* $(at\ x0)$
using *differentiable-series*[*OF assms*, *of x0*] ‹*x0* $\in$ *s*› **by** *blast+*

Considering derivative $real \Rightarrow 'b$ as a vector.

definition *vector-derivative f net* = $(SOME\ f'.\ (f\ has\text{-}vector\text{-}derivative\ f')\ net)$

lemma *vector-derivative-unique-within*:
assumes *not-bot*: *at x within s* $\neq$ *bot*
and *f'*: (*f has-vector-derivative f'*) (*at x within s*)
and *f''*: (*f has-vector-derivative f''*) (*at x within s*)
shows $f' = f''$
proof −
have $(\lambda x.\ x *_R f') = (\lambda x.\ x *_R f'')$
proof (*rule frechet-derivative-unique-within*)
show $\forall i{\in}Basis.\ \forall e{>}0.\ \exists d.\ 0 < |d| \wedge |d| < e \wedge x + d *_R i \in s$
proof *clarsimp*
fix *e* :: *real* **assume** $0 < e$
with *islimpt-approachable-real*[*of x s*] *not-bot*
obtain x' **where** $x' \in s\ x' \neq x\ |x' - x| < e$
by (*auto simp add*: *trivial-limit-within*)
then show $\exists d.\ d \neq 0 \wedge |d| < e \wedge x + d \in s$
by (*intro exI*[*of* - $x' - x$]) *auto*
qed
qed (*insert f' f''*, *auto simp*: *has-vector-derivative-def*)
then show *?thesis*
unfolding *fun-eq-iff* **by** (*metis scaleR-one*)
qed

lemma *vector-derivative-unique-at*:
(*f has-vector-derivative f'*) (*at x*) $\Longrightarrow$ (*f has-vector-derivative f''*) (*at x*) $\Longrightarrow$ $f' = f''$
by (*rule vector-derivative-unique-within*) *auto*

lemma *differentiableI-vector*: (*f has-vector-derivative y*) *F* $\Longrightarrow$ *f differentiable F*
by (*auto simp*: *differentiable-def has-vector-derivative-def*)

lemma *vector-derivative-works*:
f differentiable net $\longleftrightarrow$ (*f has-vector-derivative* (*vector-derivative f net*)) *net*
(**is** *?l* = *?r*)
proof
assume *?l*
obtain *f'* **where** *f'*: (*f has-derivative f'*) *net*
using ‹*?l*› **unfolding** *differentiable-def* **..**
then interpret *bounded-linear f'*

by *auto*
show *?r*
unfolding *vector-derivative-def has-vector-derivative-def*
by (*rule someI*[*of - f' 1*]) (*simp add*: *scaleR*[*symmetric*] *f'*)
qed (*auto simp*: *vector-derivative-def has-vector-derivative-def differentiable-def*)

lemma *vector-derivative-within*:
assumes *not-bot*: *at x within s* ≠ *bot* **and** *y*: (*f has-vector-derivative y*) (*at x within s*)
shows *vector-derivative f* (*at x within s*) = *y*
using *y*
by (*intro vector-derivative-unique-within*[*OF not-bot vector-derivative-works*[*THEN iffD1*] *y*])
(*auto simp*: *differentiable-def has-vector-derivative-def*)

lemma *frechet-derivative-eq-vector-derivative*:
assumes *f differentiable* (*at x*)
shows (*frechet-derivative f* (*at x*)) = (λ*r*. *r* ∗$_R$ *vector-derivative f* (*at x*))
using *assms*
by (*auto simp*: *differentiable-iff-scaleR vector-derivative-def has-vector-derivative-def*
intro: *someI frechet-derivative-at* [*symmetric*])

lemma *has-real-derivative*:
fixes *f* :: *real* ⇒ *real*
assumes (*f has-derivative f'*) *F*
obtains *c* **where** (*f has-real-derivative c*) *F*
proof −
obtain *c* **where** *f'* = (λ*x*. *x* ∗ *c*)
by (*metis assms has-derivative-bounded-linear real-bounded-linear*)
then show *?thesis*
by (*metis assms that has-field-derivative-def mult-commute-abs*)
qed

lemma *has-real-derivative-iff*:
fixes *f* :: *real* ⇒ *real*
shows (∃ *c*. (*f has-real-derivative c*) *F*) = (∃ *D*. (*f has-derivative D*) *F*)
by (*metis has-field-derivative-def has-real-derivative*)

definition *deriv* :: ('*a* ⇒ '*a*::*real-normed-field*) ⇒ '*a* ⇒ '*a* **where**
deriv f x ≡ *SOME D*. *DERIV f x* :> *D*

lemma *DERIV-imp-deriv*: *DERIV f x* :> *f'* ⟹ *deriv f x* = *f'*
unfolding *deriv-def* **by** (*metis some-equality DERIV-unique*)

lemma *DERIV-deriv-iff-has-field-derivative*:
DERIV f x :> *deriv f x* ⟷ (∃ *f'*. (*f has-field-derivative f'*) (*at x*))
by (*auto simp*: *has-field-derivative-def DERIV-imp-deriv*)

lemma *DERIV-deriv-iff-real-differentiable*:

fixes x :: *real*
shows *DERIV f x :> deriv f x* $\longleftrightarrow$ *f differentiable at x*
unfolding *differentiable-def* **by** (*metis DERIV-imp-deriv has-real-derivative-iff*)

lemma *real-derivative-chain*:
fixes x :: *real*
shows *f differentiable at x* $\Longrightarrow$ *g differentiable at* (*f x*)
$\Longrightarrow$ *deriv* (*g o f*) *x* = *deriv g* (*f x*) $*$ *deriv f x*
by (*metis DERIV-deriv-iff-real-differentiable DERIV-chain DERIV-imp-deriv*)
lemma *field-derivative-eq-vector-derivative*:
(*deriv f x*) = *vector-derivative f* (*at x*)
by (*simp add*: *mult.commute deriv-def vector-derivative-def has-vector-derivative-def has-field-derivative-def*)

lemma *islimpt-closure-open*:
fixes s :: *'a::perfect-space set*
assumes *open s* **and** t: t = *closure s x* $\in$ t
shows *x islimpt t*
proof *cases*
assume $x \in s$
{ fix T **assume** $x \in T$ *open T*
then have *open* ($s \cap T$)
using ‹*open s*› **by** *auto*
then have $s \cap T \neq \{x\}$
using *not-open-singleton*[*of x*] **by** *auto*
with ‹$x \in T$› ‹$x \in s$› **have** $\exists y{\in}t.\ y \in T \wedge y \neq x$
using *closure-subset*[*of s*] **by** (*auto simp*: t) **}**
then show *?thesis*
by (*auto intro*!: *islimptI*)
next
assume $x \notin s$ **with** t **show** *?thesis*
unfolding t *closure-def* **by** (*auto intro*: *islimpt-subset*)
qed

lemma *vector-derivative-unique-within-closed-interval*:
assumes *ab*: $a < b$ $x \in$ *cbox a b*
assumes D: (*f has-vector-derivative f'*) (*at x within cbox a b*) (*f has-vector-derivative f''*) (*at x within cbox a b*)
shows $f' = f''$
using *ab*
by (*intro vector-derivative-unique-within*[*OF - D*])
(*auto simp*: *trivial-limit-within intro*!: *islimpt-closure-open*[**where** s={a <..< b}])

lemma *vector-derivative-at*:
(*f has-vector-derivative f'*) (*at x*) $\Longrightarrow$ *vector-derivative f* (*at x*) = f'
by (*intro vector-derivative-within at-neq-bot*)

lemma *has-vector-derivative-id-at* [*simp*]: *vector-derivative* ($\lambda x.\ x$) (*at a*) = *1*

by (*simp add*: *vector-derivative-at*)

lemma *vector-derivative-minus-at* [*simp*]:
f differentiable at a
⟹ *vector-derivative* (λ*x*. − *f x*) (*at a*) = − *vector-derivative f* (*at a*)
by (*simp add*: *vector-derivative-at has-vector-derivative-minus vector-derivative-works* [*symmetric*])

lemma *vector-derivative-add-at* [*simp*]:
⟦*f differentiable at a*; *g differentiable at a*⟧
⟹ *vector-derivative* (λ*x*. *f x* + *g x*) (*at a*) = *vector-derivative f* (*at a*) + *vector-derivative g* (*at a*)
by (*simp add*: *vector-derivative-at has-vector-derivative-add vector-derivative-works* [*symmetric*])

lemma *vector-derivative-diff-at* [*simp*]:
⟦*f differentiable at a*; *g differentiable at a*⟧
⟹ *vector-derivative* (λ*x*. *f x* − *g x*) (*at a*) = *vector-derivative f* (*at a*) − *vector-derivative g* (*at a*)
by (*simp add*: *vector-derivative-at has-vector-derivative-diff vector-derivative-works* [*symmetric*])

lemma *vector-derivative-mult-at* [*simp*]:
fixes *f g* :: *real* ⇒ *'a* :: *real-normed-algebra*
shows ⟦*f differentiable at a*; *g differentiable at a*⟧
⟹ *vector-derivative* (λ*x*. *f x* ∗ *g x*) (*at a*) = *f a* ∗ *vector-derivative g* (*at a*) + *vector-derivative f* (*at a*) ∗ *g a*
by (*simp add*: *vector-derivative-at has-vector-derivative-mult vector-derivative-works* [*symmetric*])

lemma *vector-derivative-scaleR-at* [*simp*]:
⟦*f differentiable at a*; *g differentiable at a*⟧
⟹ *vector-derivative* (λ*x*. *f x* ∗*R* *g x*) (*at a*) = *f a* ∗*R* *vector-derivative g* (*at a*) + *vector-derivative f* (*at a*) ∗*R* *g a*
apply (*rule vector-derivative-at*)
apply (*rule has-vector-derivative-scaleR*)
apply (*auto simp*: *vector-derivative-works has-vector-derivative-def has-field-derivative-def mult-commute-abs*)
done

lemma *vector-derivative-within-closed-interval*:
assumes *ab*: *a* < *b x* ∈ *cbox a b*
assumes *f*: (*f has-vector-derivative f'*) (*at x within cbox a b*)
shows *vector-derivative f* (*at x within cbox a b*) = *f'*
by (*intro vector-derivative-unique-within-closed-interval*[*OF ab - f*]
vector-derivative-works[*THEN iffD1*] *differentiableI-vector*)
fact

lemma *has-vector-derivative-within-subset*:

(*f has-vector-derivative f'*) (*at x within s*) $\Longrightarrow$ $t \subseteq s$ $\Longrightarrow$ (*f has-vector-derivative f'*) (*at x within t*)
by (*auto simp*: *has-vector-derivative-def intro*: *has-derivative-within-subset*)

lemma *has-vector-derivative-at-within*:
(*f has-vector-derivative f'*) (*at x*) $\Longrightarrow$ (*f has-vector-derivative f'*) (*at x within s*)
unfolding *has-vector-derivative-def*
by (*rule has-derivative-at-within*)

lemma *has-vector-derivative-weaken*:
fixes *x D* **and** *f g s t*
assumes *f*: (*f has-vector-derivative D*) (*at x within t*)
and $x \in s$ $s \subseteq t$
and $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows (*g has-vector-derivative D*) (*at x within s*)
proof −
have (*f has-vector-derivative D*) (*at x within s*) $\longleftrightarrow$ (*g has-vector-derivative D*) (*at x within s*)
unfolding *has-vector-derivative-def has-derivative-iff-norm*
using *assms* **by** (*intro conj-cong Lim-cong-within refl*) *auto*
then show *?thesis*
using *has-vector-derivative-within-subset*[*OF f* ‹$s \subseteq t$›] **by** *simp*
qed

lemma *has-vector-derivative-transform-within*:
assumes (*f has-vector-derivative f'*) (*at x within s*)
and $0 < d$
and $x \in s$
and $\bigwedge x'.\ [\![x' \in s;\ \mathit{dist}\ x'\ x < d]\!] \Longrightarrow f\ x' = g\ x'$
shows (*g has-vector-derivative f'*) (*at x within s*)
using *assms*
unfolding *has-vector-derivative-def*
by (*rule has-derivative-transform-within*)

lemma *has-vector-derivative-transform-within-open*:
assumes (*f has-vector-derivative f'*) (*at x*)
and *open s*
and $x \in s$
and $\bigwedge y.\ y \in s \Longrightarrow f\ y = g\ y$
shows (*g has-vector-derivative f'*) (*at x*)
using *assms*
unfolding *has-vector-derivative-def*
by (*rule has-derivative-transform-within-open*)

lemma *vector-diff-chain-at*:
assumes (*f has-vector-derivative f'*) (*at x*)
and (*g has-vector-derivative g'*) (*at* (*f x*))
shows ((*g* ∘ *f*) *has-vector-derivative* (*f'* $*_R$ *g'*)) (*at x*)
using *assms*(*2*)

unfolding *has-vector-derivative-def*
apply −
apply (*drule diff-chain-at*[*OF assms*(*1*)[*unfolded has-vector-derivative-def*]])
apply (*simp only*: *o-def real-scaleR-def scaleR-scaleR*)
done

lemma *vector-diff-chain-within*:
assumes (*f has-vector-derivative f'*) (*at x within s*)
and (*g has-vector-derivative g'*) (*at* (*f x*) *within f ' s*)
shows ((*g* ∘ *f*) *has-vector-derivative* (*f'* *$_R$* *g'*)) (*at x within s*)
using *assms*(*2*)
unfolding *has-vector-derivative-def*
apply −
apply (*drule diff-chain-within*[*OF assms*(*1*)[*unfolded has-vector-derivative-def*]])
apply (*simp only*: *o-def real-scaleR-def scaleR-scaleR*)
done

lemma *vector-derivative-const-at* [*simp*]: *vector-derivative* (λ*x*. *c*) (*at a*) = *0*
by (*simp add*: *vector-derivative-at*)

lemma *vector-derivative-at-within-ivl*:
(*f has-vector-derivative f'*) (*at x*) ⟹
a ≤ *x* ⟹ *x* ≤ *b* ⟹ *a*<*b* ⟹ *vector-derivative f* (*at x within* {*a*..*b*}) = *f'*
using *has-vector-derivative-at-within vector-derivative-within-closed-interval* **by** *fastforce*

lemma *vector-derivative-chain-at*:
assumes *f differentiable at x* (*g differentiable at* (*f x*))
shows *vector-derivative* (*g* ∘ *f*) (*at x*) =
vector-derivative f (*at x*) *$*_R$* *vector-derivative g* (*at* (*f x*))
by (*metis vector-diff-chain-at vector-derivative-at vector-derivative-works assms*)

34.16 Relation between convexity and derivative

lemma *convex-on-imp-above-tangent*:
assumes *convex*: *convex-on A f* **and** *connected*: *connected A*
assumes *c*: *c* ∈ *interior A* **and** *x* : *x* ∈ *A*
assumes *deriv*: (*f has-field-derivative f'*) (*at c within A*)
shows *f x* − *f c* ≥ *f'* ∗ (*x* − *c*)
proof (*cases x c rule*: *linorder-cases*)
assume *xc*: *x* > *c*
let *?A'* = *interior A* ∩ {*c*<..}
from *c* **have** *c* ∈ *interior A* ∩ *closure* {*c*<..} **by** *auto*
also have ... ⊆ *closure* (*interior A* ∩ {*c*<..}) **by** (*intro open-inter-closure-subset*) *auto*
finally have *at c within ?A'* ≠ *bot* **by** (*subst at-within-eq-bot-iff*) *auto*
moreover from *deriv* **have** ((λ*y*. (*f y* − *f c*) / (*y* − *c*)) ⟶ *f'*) (*at c within ?A'*)
unfolding *DERIV-within-iff* **using** *interior-subset*[*of A*] **by** (*blast intro*: *tendsto-mono at-le*)

moreover from *eventually-at-right-real*[*OF xc*]
have *eventually* (λ*y*. (*f y* − *f c*) / (*y* − *c*) ≤ (*f x* − *f c*) / (*x* − *c*)) (*at-right c*)
proof *eventually-elim*
fix *y* **assume** *y*: *y* ∈ {*c*<..<*x*}
with *convex connected x c* **have** *f y* ≤ (*f x* − *f c*) / (*x* − *c*) ∗ (*y* − *c*) + *f c*
using *interior-subset*[*of A*]
by (*intro convex-onD-Icc*′ *convex-on-subset*[*OF convex*] *connected-contains-Icc*) *auto*
hence *f y* − *f c* ≤ (*f x* − *f c*) / (*x* − *c*) ∗ (*y* − *c*) **by** *simp*
thus (*f y* − *f c*) / (*y* − *c*) ≤ (*f x* − *f c*) / (*x* − *c*) **using** *y xc* **by** (*simp add*: *divide-simps*)
qed
hence *eventually* (λ*y*. (*f y* − *f c*) / (*y* − *c*) ≤ (*f x* − *f c*) / (*x* − *c*)) (*at c within ?A*′)
by (*blast intro*: *filter-leD at-le*)
ultimately have *f*′ ≤ (*f x* − *f c*) / (*x* − *c*) **by** (*rule tendsto-ge-const*)
thus *?thesis* **using** *xc* **by** (*simp add*: *field-simps*)
next
assume *xc*: *x* < *c*
let *?A*′ = *interior A* ∩ {..<*c*}
from *c* **have** *c* ∈ *interior A* ∩ *closure* {..<*c*} **by** *auto*
also have ... ⊆ *closure* (*interior A* ∩ {..<*c*}) **by** (*intro open-inter-closure-subset*) *auto*
finally have *at c within ?A*′ ≠ *bot* **by** (*subst at-within-eq-bot-iff*) *auto*
moreover from *deriv* **have** ((λ*y*. (*f y* − *f c*) / (*y* − *c*)) ⟶ *f*′) (*at c within ?A*′)
unfolding *DERIV-within-iff* **using** *interior-subset*[*of A*] **by** (*blast intro*: *tendsto-mono at-le*)
moreover from *eventually-at-left-real*[*OF xc*]
have *eventually* (λ*y*. (*f y* − *f c*) / (*y* − *c*) ≥ (*f x* − *f c*) / (*x* − *c*)) (*at-left c*)
proof *eventually-elim*
fix *y* **assume** *y*: *y* ∈ {*x*<..<*c*}
with *convex connected x c* **have** *f y* ≤ (*f x* − *f c*) / (*c* − *x*) ∗ (*c* − *y*) + *f c*
using *interior-subset*[*of A*]
by (*intro convex-onD-Icc*′′ *convex-on-subset*[*OF convex*] *connected-contains-Icc*) *auto*
hence *f y* − *f c* ≤ (*f x* − *f c*) ∗ ((*c* − *y*) / (*c* − *x*)) **by** *simp*
also have (*c* − *y*) / (*c* − *x*) = (*y* − *c*) / (*x* − *c*) **using** *y xc* **by** (*simp add*: *field-simps*)
finally show (*f y* − *f c*) / (*y* − *c*) ≥ (*f x* − *f c*) / (*x* − *c*) **using** *y xc*
by (*simp add*: *divide-simps*)
qed
hence *eventually* (λ*y*. (*f y* − *f c*) / (*y* − *c*) ≥ (*f x* − *f c*) / (*x* − *c*)) (*at c within ?A*′)
by (*blast intro*: *filter-leD at-le*)
ultimately have *f*′ ≥ (*f x* − *f c*) / (*x* − *c*) **by** (*rule tendsto-le-const*)
thus *?thesis* **using** *xc* **by** (*simp add*: *field-simps*)
qed *simp-all*

34.17 Partial derivatives

lemma *eventually-at-Pair-within-TimesI1*:
fixes $x::'a::metric\text{-}space$
assumes $\forall_F\ x'$ *in at* x *within* $X.\ P\ x'$
assumes $P\ x$
shows $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y.\ P\ x'$
proof −
from *assms*[*unfolded eventually-at-topological*]
obtain S **where** S: *open* $S\ x \in S \bigwedge x'.\ x' \in X \Longrightarrow x' \in S \Longrightarrow P\ x'$
by *metis*
show $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y.\ P\ x'$
unfolding *eventually-at-topological*
by (*auto intro*!: *exI*[**where** $x{=}S \times UNIV$] S *open-Times*)
qed

lemma *eventually-at-Pair-within-TimesI2*:
fixes $x::'a::metric\text{-}space$
assumes $\forall_F\ y'$ *in at* y *within* $Y.\ P\ y'$
assumes $P\ y$
shows $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y.\ P\ y'$
proof −
from *assms*[*unfolded eventually-at-topological*]
obtain S **where** S: *open* $S\ y \in S \bigwedge y'.\ y' \in Y \Longrightarrow y' \in S \Longrightarrow P\ y'$
by *metis*
show $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y.\ P\ y'$
unfolding *eventually-at-topological*
by (*auto intro*!: *exI*[**where** $x{=}UNIV \times S$] S *open-Times*)
qed

lemma *has-derivative-partialsI*:
assumes *fx*: $\bigwedge x\ y.\ x \in X \Longrightarrow y \in Y \Longrightarrow ((\lambda x.\ f\ x\ y)$ *has-derivative blinfun-apply* $(fx\ x\ y))$ $(at\ x\ within\ X)$
assumes *fy*: $\bigwedge x\ y.\ x \in X \Longrightarrow y \in Y \Longrightarrow ((\lambda y.\ f\ x\ y)$ *has-derivative blinfun-apply* $(fy\ x\ y))$ $(at\ y\ within\ Y)$
assumes *fx-cont*: *continuous-on* $(X \times Y)\ (\lambda(x, y).\ fx\ x\ y)$
assumes *fy-cont*: *continuous-on* $(X \times Y)\ (\lambda(x, y).\ fy\ x\ y)$
assumes $x \in X\ y \in Y$
assumes *convex X convex Y*
shows $((\lambda(x, y).\ f\ x\ y)$ *has-derivative* $(\lambda(tx, ty).\ fx\ x\ y\ tx + fy\ x\ y\ ty))$ $(at\ (x, y)\ within\ X \times Y)$
proof (*safe intro*!: *has-derivativeI tendstoI*, *goal-cases*)
case $(2\ e')$
def $e{\equiv}e'\ /\ 9$
have $e > 0$ **using** $\langle e' > 0\rangle$ **by** (*simp add*: *e-def*)

have $(x, y) \in X \times Y$ **using** *assms* **by** *auto*
from *fy-cont*[*unfolded continuous-on-eq-continuous-within*, *rule-format*, *OF this*, *unfolded continuous-within*, *THEN tendstoD*, *OF* $\langle e > 0\rangle$]
have $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y.\ dist\ (fy\ x'\ y')\ (fy\ x\ y) < e$

```
    by (auto simp: split-beta')
  from this[unfolded eventually-at] obtain d' where
    d' > 0
    ⋀x' y'. x' ∈ X ⟹ y' ∈ Y ⟹ (x', y') ≠ (x, y) ⟹ dist (x', y') (x, y) < d' ⟹
      dist (fy x' y') (fy x y) < e
    by auto
  then
  have d': x' ∈ X ⟹ y' ∈ Y ⟹ dist (x', y') (x, y) < d' ⟹ dist (fy x' y') (fy x y) < e
    for x' y'
    using ‹0 < e›
    by (cases (x', y') = (x, y)) auto
  def d ≡ d' / sqrt 2
  have d > 0 using ‹0 < d'› by (simp add: d-def)
  have d: x' ∈ X ⟹ y' ∈ Y ⟹ dist x' x < d ⟹ dist y' y < d ⟹ dist (fy x' y') (fy x y) < e
    for x' y'
    by (auto simp: dist-prod-def d-def intro!: d' real-sqrt-sum-squares-less)

  let ?S = ball y d ∩ Y
  have convex ?S
    by (auto intro!: convex-Int ‹convex Y›)
  {
    fix x'::'a and y'::'b
    assume x': x' ∈ X and y': y' ∈ Y
    assume dx': dist x' x < d and dy': dist y' y < d
    have norm (fy x' y' - fy x' y) ≤ dist (fy x' y') (fy x y) + dist (fy x' y) (fy x y)
      by norm
    also have dist (fy x' y') (fy x y) < e
      by (rule d; fact)
    also have dist (fy x' y) (fy x y) < e
      by (auto intro!: d simp: dist-prod-def x' ‹d > 0› ‹y ∈ Y› dx')
    finally
    have norm (fy x' y' - fy x' y) < e + e
      by arith
    then have onorm (blinfun-apply (fy x' y') - blinfun-apply (fy x' y)) < e + e
      by (auto simp: norm-blinfun.rep-eq blinfun.diff-left[abs-def] fun-diff-def)
  } note onorm = this

  have ev-mem: ∀F (x', y') in at (x, y) within X × Y. (x', y') ∈ X × Y
    using ‹x ∈ X› ‹y ∈ Y›
    by (auto simp: eventually-at intro!: zero-less-one)
  moreover
  have ev-dist: ∀F xy in at (x, y) within X × Y. dist xy (x, y) < d if d > 0 for d
    using eventually-at-ball[OF that]
    by (rule eventually-elim2) (auto simp: dist-commute intro!: eventually-True)
```

```
  note ev-dist[OF ‹0 < d›]
  ultimately
  have ∀F (x', y') in at (x, y) within X × Y.
    norm (f x' y' − f x' y − (fy x' y) (y' − y)) ≤ norm (y' − y) * (e + e)
  proof (eventually-elim, safe)
    fix x' y'
    assume x' ∈ X and y': y' ∈ Y
    assume dist: dist (x', y') (x, y) < d
    then have dx: dist x' x < d and dy: dist y' y < d
      unfolding dist-prod-def fst-conv snd-conv atomize-conj
      by (metis le-less-trans real-sqrt-sum-squares-ge1 real-sqrt-sum-squares-ge2)
    {
      fix t::real
      assume t ∈ {0 .. 1}
      then have y + t *R (y' − y) ∈ closed-segment y y'
        by (auto simp: closed-segment-def algebra-simps intro!: exI[where x=t])
      also
      have ... ⊆ ball y d ∩ Y
        using ‹y ∈ Y› ‹0 < d› dy y'
        by (intro ‹convex ?S›[unfolded convex-contains-segment, rule-format, of y
y'])
           (auto simp: dist-commute)
      finally have y + t *R (y' − y) ∈ ?S .
    } note seg = this

    have ∀x∈ball y d ∩ Y. onorm (blinfun-apply (fy x' x) − blinfun-apply (fy x'
y)) ≤ e + e
      by (safe intro!: onorm less-imp-le ‹x' ∈ X› dx) (auto simp: dist-commute ‹0
< d› ‹y ∈ Y›)
    with seg has-derivative-within-subset[OF assms(2)[OF ‹x' ∈ X›]]
    show norm (f x' y' − f x' y − (fy x' y) (y' − y)) ≤ norm (y' − y) * (e + e)
      by (rule differentiable-bound-linearization[where S=?S])
         (auto intro!: ‹0 < d› ‹y ∈ Y›)
  qed
  moreover
  let ?le = λx'. norm (f x' y − f x y − (fx x y) (x' − x)) ≤ norm (x' − x) * e
  from fx[OF ‹x ∈ X› ‹y ∈ Y›, unfolded has-derivative-within, THEN conjunct2,
THEN tendstoD, OF ‹0 < e›]
  have ∀F x' in at x within X. ?le x'
    by eventually-elim
       (auto simp: dist-norm divide-simps blinfun.bilinear-simps field-simps split:
split-if-asm)
  then have ∀F (x', y') in at (x, y) within X × Y. ?le x'
    by (rule eventually-at-Pair-within-TimesI1)
       (simp add: blinfun.bilinear-simps)
  moreover have ∀F (x', y') in at (x, y) within X × Y. norm ((x', y') − (x,
y)) ≠ 0
    unfolding norm-eq-zero right-minus-eq
    by (auto simp: eventually-at intro!: zero-less-one)
```

moreover
from *fy-cont*[*unfolded continuous-on-eq-continuous-within*, *rule-format*, *OF SigmaI*[*OF* ‹$x \in X$› ‹$y \in Y$›],
unfolded continuous-within, *THEN tendstoD*, *OF* ‹$0 < e$›]
have $\forall_F\ x'$ *in at* x *within* X. *norm* $(fy\ x'\ y - fy\ x\ y) < e$
unfolding *eventually-at*
using ‹$y \in Y$›
by (*auto simp*: *dist-prod-def dist-norm*)
then have $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y$. *norm* $(fy\ x'\ y - fy\ x\ y) < e$
by (*rule eventually-at-Pair-within-TimesI1*)
(*simp add*: *blinfun.bilinear-simps* ‹$0 < e$›)
ultimately
have $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y$.
norm $((f\ x'\ y' - f\ x\ y - (fx\ x\ y\ (x' - x) + fy\ x\ y\ (y' - y))) /_R$
norm $((x', y') - (x, y)))$
$< e'$
apply *eventually-elim*
proof *safe*
fix $x'\ y'$
have *norm* $(f\ x'\ y' - f\ x\ y - (fx\ x\ y\ (x' - x) + fy\ x\ y\ (y' - y))) \leq$
norm $(f\ x'\ y' - f\ x'\ y - fy\ x'\ y\ (y' - y)) +$
norm $(fy\ x\ y\ (y' - y) - fy\ x'\ y\ (y' - y)) +$
norm $(f\ x'\ y - f\ x\ y - fx\ x\ y\ (x' - x))$
by *norm*
also
assume *nz*: *norm* $((x', y') - (x, y)) \neq 0$
and *nfy*: *norm* $(fy\ x'\ y - fy\ x\ y) < e$
assume *norm* $(f\ x'\ y' - f\ x'\ y - blinfun\text{-}apply\ (fy\ x'\ y)\ (y' - y)) \leq norm\ (y' - y) * (e + e)$
also assume *norm* $(f\ x'\ y - f\ x\ y - blinfun\text{-}apply\ (fx\ x\ y)\ (x' - x)) \leq norm\ (x' - x) * e$
also
have *norm* $((fy\ x\ y)\ (y' - y) - (fy\ x'\ y)\ (y' - y)) \leq norm\ ((fy\ x\ y) - (fy\ x'\ y)) * norm\ (y' - y)$
by (*auto simp*: *blinfun.bilinear-simps*[*symmetric*] *intro*!: *norm-blinfun*)
also have $\ldots \leq (e + e) * norm\ (y' - y)$
using ‹$e > 0$› *nfy*
by (*auto simp*: *norm-minus-commute intro*!: *mult-right-mono*)
also have *norm* $(x' - x) * e \leq norm\ (x' - x) * (e + e)$
using ‹$0 < e$› **by** *simp*
also have *norm* $(y' - y) * (e + e) + (e + e) * norm\ (y' - y) + norm\ (x' - x) * (e + e) \leq$
$(norm\ (y' - y) + norm\ (x' - x)) * (4 * e)$
using ‹$e > 0$›
by (*simp add*: *algebra-simps*)
also have $\ldots \leq 2 * norm\ ((x', y') - (x, y)) * (4 * e)$
using ‹$0 < e$› *real-sqrt-sum-squares-ge1*[*of norm* $(x' - x)$ *norm* $(y' - y)$]
real-sqrt-sum-squares-ge2[*of norm* $(y' - y)$ *norm* $(x' - x)$]
by (*auto intro*!: *mult-right-mono simp*: *norm-prod-def*

simp del: *real-sqrt-sum-squares-ge1 real-sqrt-sum-squares-ge2*)
also have ... $\leq$ *norm* $((x', y') - (x, y)) * (8 * e)$
by *simp*
also have ... $<$ *norm* $((x', y') - (x, y)) * e'$
using ‹$0 < e'$› *nz*
by (*auto simp*: *e-def*)
finally show *norm* $((f\,x'\,y' - f\,x\,y - (fx\,x\,y\,(x' - x) + fy\,x\,y\,(y' - y))) /_R$ *norm* $((x', y') - (x, y))) < e'$
by (*auto simp*: *divide-simps dist-norm mult.commute*)
qed
then show *?case*
by *eventually-elim* (*auto simp*: *dist-norm field-simps*)
qed (*auto intro*!: *bounded-linear-intros simp*: *split-beta'*)

end

35 Kurzweil-Henstock Gauge Integration in many dimensions.

theory *Integration*
imports
Derivative
Uniform-Limit
~~/*src*/*HOL*/*Library*/*Indicator-Function*
begin

lemma *cSup-abs-le*:
fixes S :: ('*a*::{*linordered-idom*,*conditionally-complete-linorder*}) *set*
shows $S \neq \{\} \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow |x| \leq a) \Longrightarrow |Sup\ S| \leq a$
apply (*auto simp add*: *abs-le-iff intro*: *cSup-least*)
by (*metis bdd-aboveI cSup-upper neg-le-iff-le order-trans*)

lemma *cInf-abs-ge*:
fixes S :: ('*a*::{*linordered-idom*,*conditionally-complete-linorder*}) *set*
assumes $S \neq \{\}$ **and** *bdd*: $\bigwedge x.\ x \in S \Longrightarrow |x| \leq a$
shows $|Inf\ S| \leq a$
proof −
have *Sup* (*uminus* ‘ S) = − (*Inf* S)
proof (*rule antisym*)
show − (*Inf* S) $\leq$ *Sup*(*uminus* ‘ S)
apply (*subst minus-le-iff*)
apply (*rule cInf-greatest* [*OF* ‹$S \neq \{\}$›])
apply (*subst minus-le-iff*)
apply (*rule cSup-upper*, *force*)
using *bdd* **apply** (*force simp add*: *abs-le-iff bdd-above-def*)
done
next

```
    show Sup (uminus ‘ S) ≤ − Inf S
      apply (rule cSup-least)
       using ‹S ≠ {}› apply (force simp add: )
      apply clarsimp
      apply (rule cInf-lower)
      apply assumption
      using bdd apply (simp add: bdd-below-def)
      apply (rule-tac x=−a in exI)
      apply force
      done
  qed
  with cSup-abs-le [of uminus ‘ S] assms show ?thesis by fastforce
qed

lemma cSup-asclose:
  fixes S :: ('a::{linordered-idom,conditionally-complete-linorder}) set
  assumes S: S ≠ {}
    and b: ∀ x∈S. |x − l| ≤ e
  shows |Sup S − l| ≤ e
proof −
  have th: ⋀(x::'a) l e. |x − l| ≤ e ⟷ l − e ≤ x ∧ x ≤ l + e
    by arith
  have bdd-above S
    using b by (auto intro!: bdd-aboveI[of - l + e])
  with S b show ?thesis
    unfolding th by (auto intro!: cSup-upper2 cSup-least)
qed

lemma cInf-asclose:
  fixes S :: real set
  assumes S: S ≠ {}
    and b: ∀ x∈S. |x − l| ≤ e
  shows |Inf S − l| ≤ e
proof −
  have |− Sup (uminus ‘ S) − l| =  |Sup (uminus ‘ S) − (−l)|
    by auto
  also have ... ≤ e
    apply (rule cSup-asclose)
    using abs-minus-add-cancel b by (auto simp add: S)
  finally have |− Sup (uminus ‘ S) − l| ≤ e .
  then show ?thesis
    by (simp add: Inf-real-def)
qed

lemmas scaleR-simps = scaleR-zero-left scaleR-minus-left scaleR-left-diff-distrib
  scaleR-zero-right scaleR-minus-right scaleR-right-diff-distrib scaleR-eq-0-iff
  scaleR-cancel-left scaleR-cancel-right scaleR-add-right scaleR-add-left real-vector-class.scaleR-one

lemma real-arch-invD:
```

$0 < (e::real) \Longrightarrow (\exists\, n::nat.\ n \neq 0 \wedge 0 < inverse\ (real\ n) \wedge inverse\ (real\ n) < e)$
by (*subst*(*asm*) *real-arch-inv*)

35.1 Sundries

lemma *conjunctD2*: **assumes** $a \wedge b$ **shows** $a\ b$ **using** *assms* **by** *auto*
lemma *conjunctD3*: **assumes** $a \wedge b \wedge c$ **shows** $a\ b\ c$ **using** *assms* **by** *auto*
lemma *conjunctD4*: **assumes** $a \wedge b \wedge c \wedge d$ **shows** $a\ b\ c\ d$ **using** *assms* **by** *auto*

declare *norm-triangle-ineq4* [*intro*]

lemma *simple-image*: $\{f\ x\ |x\ .\ x \in s\} = f\ `\ s$
by *blast*

lemma *linear-simps*:
assumes *bounded-linear f*
shows
$f\ (a + b) = f\ a + f\ b$
$f\ (a - b) = f\ a - f\ b$
$f\ 0 = 0$
$f\ (-\ a) = -\ f\ a$
$f\ (s *_R v) = s *_R (f\ v)$
proof −
interpret *f*: *bounded-linear f* **by** *fact*
show $f\ (a + b) = f\ a + f\ b$ **by** (*rule f.add*)
show $f\ (a - b) = f\ a - f\ b$ **by** (*rule f.diff*)
show $f\ 0 = 0$ **by** (*rule f.zero*)
show $f\ (-\ a) = -\ f\ a$ **by** (*rule f.minus*)
show $f\ (s *_R v) = s *_R (f\ v)$ **by** (*rule f.scaleR*)
qed

lemma *bounded-linearI*:
assumes $\bigwedge x\ y.\ f\ (x + y) = f\ x + f\ y$
and $\bigwedge r\ x.\ f\ (r *_R x) = r *_R f\ x$
and $\bigwedge x.\ norm\ (f\ x) \leq norm\ x * K$
shows *bounded-linear f*
using *assms* **by** (*rule bounded-linear-intro*)

lemma *bounded-linear-component* [*intro*]: *bounded-linear* $(\lambda x::'a::euclidean\text{-}space.\ x \cdot k)$
by (*rule bounded-linear-inner-left*)

lemma *transitive-stepwise-lt-eq*:
assumes $(\bigwedge x\ y\ z::nat.\ R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z)$
shows $((\forall\, m.\ \forall\, n{>}m.\ R\ m\ n) \longleftrightarrow (\forall\, n.\ R\ n\ (Suc\ n)))$
(**is** *?l* = *?r*)
proof *safe*
assume *?r*

```
  fix n m :: nat
  assume m < n
  then show R m n
  proof (induct n arbitrary: m)
    case 0
    then show ?case by auto
  next
    case (Suc n)
    show ?case
    proof (cases m < n)
      case True
      show ?thesis
        apply (rule assms[OF Suc(1)[OF True]])
        using ‹?r›
        apply auto
        done
    next
      case False
      then have m = n
        using Suc(2) by auto
      then show ?thesis
        using ‹?r› by auto
    qed
  qed
qed auto

lemma transitive-stepwise-gt:
  assumes ⋀x y z. R x y ⟹ R y z ⟹ R x z ⋀n. R n (Suc n)
  shows ∀n>m. R m n
proof −
  have ∀m. ∀n>m. R m n
    apply (subst transitive-stepwise-lt-eq)
    apply (blast intro: assms)+
    done
  then show ?thesis by auto
qed

lemma transitive-stepwise-le-eq:
  assumes ⋀x. R x x ⋀x y z. R x y ⟹ R y z ⟹ R x z
  shows (∀m. ∀n≥m. R m n) ⟷ (∀n. R n (Suc n))
  (is ?l = ?r)
proof safe
  assume ?r
  fix m n :: nat
  assume m ≤ n
  then show R m n
  proof (induct n arbitrary: m)
    case 0
    with assms show ?case by auto
```

```
  next
    case (Suc n)
    show ?case
    proof (cases m ≤ n)
      case True
      with Suc.hyps ‹∀ n. R n (Suc n)› assms show ?thesis
        by blast
    next
      case False
      then have m = Suc n
        using Suc(2) by auto
      then show ?thesis
        using assms(1) by auto
    qed
  qed
qed auto

lemma transitive-stepwise-le:
  assumes ⋀x. R x x ⋀x y z. R x y ⟹ R y z ⟹ R x z
    and ⋀n. R n (Suc n)
  shows ∀ n≥m. R m n
proof −
  have ∀ m. ∀ n≥m. R m n
    apply (subst transitive-stepwise-le-eq)
    apply (blast intro: assms)+
    done
  then show ?thesis by auto
qed
```

35.2 Some useful lemmas about intervals.

```
lemma empty-as-interval: {} = cbox One (0::'a::euclidean-space)
  using nonempty-Basis
  by (fastforce simp add: set-eq-iff mem-box)

lemma interior-subset-union-intervals:
  assumes i = cbox a b
    and j = cbox c d
    and interior j ≠ {}
    and i ⊆ j ∪ s
    and interior i ∩ interior j = {}
  shows interior i ⊆ interior s
proof −
  have box a b ∩ cbox c d = {}
     using inter-interval-mixed-eq-empty[of c d a b] and assms(3,5)
     unfolding assms(1,2) interior-cbox by auto
  moreover
  have box a b ⊆ cbox c d ∪ s
    apply (rule order-trans,rule box-subset-cbox)
```

```
    using assms(4) unfolding assms(1,2)
    apply auto
    done
  ultimately
  show ?thesis
    unfolding assms interior-cbox
    by auto (metis IntI UnE empty-iff interior-maximal open-box subsetCE subsetI)
qed

lemma inter-interior-unions-intervals:
  fixes f::('a::euclidean-space) set set
  assumes finite f
    and open s
    and ∀ t∈f. ∃ a b. t = cbox a b
    and ∀ t∈f. s ∩ (interior t) = {}
  shows s ∩ interior (⋃f) = {}
proof (clarsimp simp only: all-not-in-conv [symmetric])
  fix x
  assume x: x ∈ s x ∈ interior (⋃f)
  have lem1: ⋀x e s U. ball x e ⊆ s ∩ interior U ⟷ ball x e ⊆ s ∩ U
    using interior-subset
   by auto (meson Topology-Euclidean-Space.open-ball contra-subsetD interior-maximal
mem-ball)
  have ∃ t∈f. ∃ x. ∃ e>0. ball x e ⊆ s ∩ t
    if finite f and ∀ t∈f. ∃ a b. t = cbox a b and ∃ x. x ∈ s ∩ interior (⋃f) for f
    using that
  proof (induct rule: finite-induct)
    case empty
    obtain x where x ∈ s ∩ interior (⋃{})
      using empty(2) ..
    then have False
      unfolding Union-empty interior-empty by auto
    then show ?case by auto
  next
    case (insert i f)
    obtain x where x: x ∈ s ∩ interior (⋃insert i f)
      using insert(5) ..
    then obtain e where e: 0 < e ∧ ball x e ⊆ s ∩ interior (⋃insert i f)
       unfolding open-contains-ball-eq[OF open-Int[OF assms(2) open-interior],
rule-format] ..
    obtain a where ∃ b. i = cbox a b
      using insert(4)[rule-format,OF insertI1] ..
    then obtain b where ab: i = cbox a b ..
    show ?case
    proof (cases x ∈ i)
      case False
      then have x ∈ UNIV − cbox a b
        unfolding ab by auto
      then obtain d where 0 < d ∧ ball x d ⊆ UNIV − cbox a b
```

```
    unfolding open-contains-ball-eq[OF open-Diff[OF open-UNIV closed-cbox],rule-format]
..
    then have 0 < d ball x (min d e) ⊆ UNIV − i
      unfolding ab ball-min-Int by auto
    then have ball x (min d e) ⊆ s ∩ interior (⋃f)
      using e unfolding lem1 unfolding  ball-min-Int by auto
    then have x ∈ s ∩ interior (⋃f) using ‹d>0› e by auto
    then have ∃t∈f. ∃x e. 0 < e ∧ ball x e ⊆ s ∩ t
      using insert.hyps(3) insert.prems(1) by blast
    then show ?thesis by auto
  next
    case True show ?thesis
    proof (cases x∈box a b)
      case True
      then obtain d where 0 < d ∧ ball x d ⊆ box a b
        unfolding open-contains-ball-eq[OF open-box,rule-format] ..
      then show ?thesis
        apply (rule-tac x=i in bexI, rule-tac x=x in exI, rule-tac x=min d e in
exI)
        unfolding ab
        using box-subset-cbox[of a b] and e
        apply fastforce+
        done
    next
      case False
      then obtain k where x·k ≤ a·k ∨ x·k ≥ b·k and k: k ∈ Basis
        unfolding mem-box by (auto simp add: not-less)
      then have x·k = a·k ∨ x·k = b·k
        using True unfolding ab and mem-box
          apply (erule-tac x = k in ballE)
          apply auto
          done
      then have ∃x. ball x (e/2) ⊆ s ∩ (⋃f)
      proof (rule disjE)
        let ?z = x − (e/2) *R k
        assume as: x·k = a·k
        have ball ?z (e / 2) ∩ i = {}
        proof (clarsimp simp only: all-not-in-conv [symmetric])
          fix y
          assume y ∈ ball ?z (e / 2) and yi: y ∈ i
          then have dist ?z y < e/2 by auto
          then have |(?z − y) · k| < e/2
            using Basis-le-norm[OF k, of ?z − y] unfolding dist-norm by auto
          then have y·k < a·k
            using e k
            by (auto simp add: field-simps abs-less-iff as inner-simps)
          then have y ∉ i
            unfolding ab mem-box by (auto intro!: bexI[OF - k])
          then show False using yi by auto
```

```
  qed
  moreover
  have ball ?z (e/2) ⊆ s ∩ (⋃ insert i f)
    apply (rule order-trans[OF - e[THEN conjunct2, unfolded lem1]])
  proof
    fix y
    assume as: y ∈ ball ?z (e/2)
    have norm (x − y) ≤ |e| / 2 + norm (x − y − (e / 2) *R k)
      apply (rule order-trans,rule norm-triangle-sub[of x − y (e/2) *R k])
      unfolding norm-scaleR norm-Basis[OF k]
      apply auto
      done
    also have ... < |e| / 2 + |e| / 2
      apply (rule add-strict-left-mono)
      using as e
      apply (auto simp add: field-simps dist-norm)
      done
    finally show y ∈ ball x e
     unfolding mem-ball dist-norm using e by (auto simp add:field-simps)
  qed
  ultimately show ?thesis
    apply (rule-tac x=?z in exI)
    unfolding Union-insert
    apply auto
    done
next
  let ?z = x + (e/2) *R k
  assume as: x•k = b•k
  have ball ?z (e / 2) ∩ i = {}
  proof (clarsimp simp only: all-not-in-conv [symmetric])
    fix y
    assume y ∈ ball ?z (e / 2) and yi: y ∈ i
    then have dist ?z y < e/2
      by auto
    then have |(?z − y) • k| < e/2
      using Basis-le-norm[OF k, of ?z − y]
      unfolding dist-norm by auto
    then have y•k > b•k
      using e k
      by (auto simp add:field-simps inner-simps inner-Basis as)
    then have y ∉ i
      unfolding ab mem-box by (auto intro!: bexI[OF - k])
    then show False using yi by auto
  qed
  moreover
  have ball ?z (e/2) ⊆ s ∩ (⋃ insert i f)
    apply (rule order-trans[OF - e[THEN conjunct2, unfolded lem1]])
  proof
    fix y
```

```
        assume as: y∈ ball ?z (e/2)
        have norm (x − y) ≤ |e| / 2 + norm (x − y + (e / 2) *R k)
         apply (rule order-trans,rule norm-triangle-sub[of x − y − (e/2) *R k])
          unfolding norm-scaleR
          apply (auto simp: k)
          done
        also have ... < |e| / 2 + |e| / 2
          apply (rule add-strict-left-mono)
          using as unfolding mem-ball dist-norm
          using e apply (auto simp add: field-simps)
          done
        finally show y ∈ ball x e
         unfolding mem-ball dist-norm using e by (auto simp add:field-simps)
       qed
       ultimately show ?thesis
        apply (rule-tac x=?z in exI)
        unfolding Union-insert
        apply auto
        done
     qed
     then obtain x where ball x (e / 2) ⊆ s ∩ ⋃f ..
     then have x ∈ s ∩ interior (⋃f)
      unfolding lem1[where U=⋃f, symmetric]
      using centre-in-ball e by auto
     then show ?thesis
      using insert.hyps(3) insert.prems(1) by blast
   qed
  qed
 qed
 from this[OF assms(1,3)] x
 obtain t x e where t ∈ f 0 < e ball x e ⊆ s ∩ t
  by blast
 then have x ∈ s x ∈ interior t
  using open-subset-interior[OF open-ball, of x e t]
  by auto
 then show False
  using ‹t ∈ f› assms(4) by auto
qed
```

35.3 Bounds on intervals where they exist.

```
definition interval-upperbound :: ('a::euclidean-space) set ⇒ 'a
 where interval-upperbound s = (∑ i∈Basis. (SUP x:s. x•i) *R i)

definition interval-lowerbound :: ('a::euclidean-space) set ⇒ 'a
  where interval-lowerbound s = (∑ i∈Basis. (INF x:s. x•i) *R i)

lemma interval-upperbound[simp]:
 ∀ i∈Basis. a•i ≤ b•i ⟹
```

interval-upperbound (*cbox a b*) = (*b*::′*a*::*euclidean-space*)
unfolding *interval-upperbound-def euclidean-representation-setsum cbox-def SUP-def*
by (*safe intro*!: *cSup-eq*) *auto*

lemma *interval-lowerbound*[*simp*]:
∀ *i*∈*Basis*. *a*•*i* ≤ *b*•*i* ⟹
interval-lowerbound (*cbox a b*) = (*a*::′*a*::*euclidean-space*)
unfolding *interval-lowerbound-def euclidean-representation-setsum cbox-def INF-def*
by (*safe intro*!: *cInf-eq*) *auto*

lemmas *interval-bounds* = *interval-upperbound interval-lowerbound*

lemma
fixes *X*::*real set*
shows *interval-upperbound-real*[*simp*]: *interval-upperbound X* = *Sup X*
and *interval-lowerbound-real*[*simp*]: *interval-lowerbound X* = *Inf X*
by (*auto simp*: *interval-upperbound-def interval-lowerbound-def SUP-def INF-def*)

lemma *interval-bounds*′[*simp*]:
assumes *cbox a b* ≠ {}
shows *interval-upperbound* (*cbox a b*) = *b*
and *interval-lowerbound* (*cbox a b*) = *a*
using *assms* **unfolding** *box-ne-empty* **by** *auto*

lemma *interval-upperbound-Times*:
assumes *A* ≠ {} **and** *B* ≠ {}
shows *interval-upperbound* (*A* × *B*) = (*interval-upperbound A*, *interval-upperbound B*)
proof−
from *assms* **have** *fst-image-times*′: *A* = *fst* ‘ (*A* × *B*) **by** *simp*
have (∑ *i*∈*Basis*. (*SUP x*:*A* × *B*. *x* • (*i*, *0*)) ∗*R i*) = (∑ *i*∈*Basis*. (*SUP x*:*A*. *x* • *i*) ∗*R i*)
by (*subst* (*2*) *fst-image-times*′) (*simp del*: *fst-image-times add*: *o-def inner-Pair-0*)
moreover from *assms* **have** *snd-image-times*′: *B* = *snd* ‘ (*A* × *B*) **by** *simp*
have (∑ *i*∈*Basis*. (*SUP x*:*A* × *B*. *x* • (*0*, *i*)) ∗*R i*) = (∑ *i*∈*Basis*. (*SUP x*:*B*. *x* • *i*) ∗*R i*)
by (*subst* (*2*) *snd-image-times*′) (*simp del*: *snd-image-times add*: *o-def inner-Pair-0*)
ultimately show *?thesis* **unfolding** *interval-upperbound-def*
by (*subst setsum-Basis-prod-eq*) (*auto simp add*: *setsum-prod*)
qed

lemma *interval-lowerbound-Times*:
assumes *A* ≠ {} **and** *B* ≠ {}
shows *interval-lowerbound* (*A* × *B*) = (*interval-lowerbound A*, *interval-lowerbound B*)
proof−
from *assms* **have** *fst-image-times*′: *A* = *fst* ‘ (*A* × *B*) **by** *simp*

have $(\sum i{\in}Basis.\ (INF\ x{:}A\ \times\ B.\ x\ \cdot\ (i,\ 0))\ *_R\ i) = (\sum i{\in}Basis.\ (INF\ x{:}A.\ x\ \cdot\ i)\ *_R\ i)$
by (*subst* (*2*) *fst-image-times'*) (*simp del*: *fst-image-times add*: *o-def inner-Pair-0*)
moreover from *assms* **have** *snd-image-times'*: $B = snd\ `\ (A \times B)$ **by** *simp*
have $(\sum i{\in}Basis.\ (INF\ x{:}A\ \times\ B.\ x\ \cdot\ (0,\ i))\ *_R\ i) = (\sum i{\in}Basis.\ (INF\ x{:}B.\ x\ \cdot\ i)\ *_R\ i)$
by (*subst* (*2*) *snd-image-times'*) (*simp del*: *snd-image-times add*: *o-def inner-Pair-0*)
ultimately show *?thesis* **unfolding** *interval-lowerbound-def*
by (*subst setsum-Basis-prod-eq*) (*auto simp add*: *setsum-prod*)
qed

35.4 Content (length, area, volume...) of an interval.

definition *content* $(s{::}('a{::}euclidean\text{-}space)\ set) =$
$(if\ s = \{\}\ then\ 0\ else\ (\prod i{\in}Basis.\ (interval\text{-}upperbound\ s){\cdot}i - (interval\text{-}lowerbound\ s){\cdot}i))$

lemma *interval-not-empty*: $\forall\, i{\in}Basis.\ a{\cdot}i \le b{\cdot}i \Longrightarrow cbox\ a\ b \neq \{\}$
unfolding *box-eq-empty* **unfolding** *not-ex not-less* **by** *auto*

lemma *content-cbox*:
fixes $a :: 'a{::}euclidean\text{-}space$
assumes $\forall\, i{\in}Basis.\ a{\cdot}i \le b{\cdot}i$
shows $content\ (cbox\ a\ b) = (\prod i{\in}Basis.\ b{\cdot}i - a{\cdot}i)$
using *interval-not-empty*[*OF assms*]
unfolding *content-def*
by *auto*

lemma *content-cbox'*:
fixes $a :: 'a{::}euclidean\text{-}space$
assumes $cbox\ a\ b \neq \{\}$
shows $content\ (cbox\ a\ b) = (\prod i{\in}Basis.\ b{\cdot}i - a{\cdot}i)$
using *assms box-ne-empty*(*1*) *content-cbox* **by** *blast*

lemma *content-real*: $a \le b \Longrightarrow content\ \{a..b\} = b - a$
by (*auto simp*: *interval-upperbound-def interval-lowerbound-def SUP-def INF-def content-def*)

lemma *abs-eq-content*: $|y - x| = (if\ x{\le}y\ then\ content\ \{x\ ..\ y\}\ else\ content\ \{y..x\})$
by (*auto simp*: *content-real*)

lemma *content-singleton*[*simp*]: $content\ \{a\} = 0$
proof −
have $content\ (cbox\ a\ a) = 0$
by (*subst content-cbox*) (*auto simp*: *ex-in-conv*)
then show *?thesis* **by** (*simp add*: *cbox-sing*)
qed

lemma *content-unit*[*iff*]: *content*(*cbox 0* (*One*::*'a*::*euclidean-space*)) = *1*
proof −
have ∗: ∀ *i*∈*Basis*. (*0*::*'a*)•*i* ≤ (*One*::*'a*)•*i*
by *auto*
have *0* ∈ *cbox 0* (*One*::*'a*)
unfolding *mem-box* **by** *auto*
then show *?thesis*
unfolding *content-def interval-bounds*[*OF* ∗] **using** *setprod.neutral-const* **by** *auto*
qed

lemma *content-pos-le*[*intro*]:
fixes *a*::*'a*::*euclidean-space*
shows *0* ≤ *content* (*cbox a b*)
proof (*cases cbox a b* = {})
case *False*
then have ∗: ∀ *i*∈*Basis*. *a* • *i* ≤ *b* • *i*
unfolding *box-ne-empty* **.**
have *0* ≤ (∏ *i*∈*Basis*. *interval-upperbound* (*cbox a b*) • *i* − *interval-lowerbound* (*cbox a b*) • *i*)
apply (*rule setprod-nonneg*)
unfolding *interval-bounds*[*OF* ∗]
using ∗
apply *auto*
done
also have ... = *content* (*cbox a b*) **using** *False* **by** (*simp add*: *content-def*)
finally show *?thesis* **.**
qed (*simp add*: *content-def*)

corollary *content-nonneg* [*simp*]:
fixes *a*::*'a*::*euclidean-space*
shows ~ *content* (*cbox a b*) < *0*
using *not-le* **by** *blast*

lemma *content-pos-lt*:
fixes *a* :: *'a*::*euclidean-space*
assumes ∀ *i*∈*Basis*. *a*•*i* < *b*•*i*
shows *0* < *content* (*cbox a b*)
using *assms*
by (*auto simp*: *content-def box-eq-empty intro*!: *setprod-pos*)

lemma *content-eq-0*:
content (*cbox a b*) = *0* ⟷ (∃ *i*∈*Basis*. *b*•*i* ≤ *a*•*i*)
by (*auto simp*: *content-def box-eq-empty intro*!: *setprod-pos bexI*)

lemma *cond-cases*: (*P* ⟹ *Q x*) ⟹ (¬ *P* ⟹ *Q y*) ⟹ *Q* (*if P then x else y*)
by *auto*

lemma *content-cbox-cases*:

content (cbox a (b::'a::euclidean-space)) =
(if ∀ i∈Basis. a•i ≤ b•i then setprod (λi. b•i − a•i) Basis else 0)
by (*auto simp*: *not-le content-eq-0 intro*: *less-imp-le content-cbox*)

lemma *content-eq-0-interior*: *content (cbox a b) = 0 ⟷ interior(cbox a b) = {}*
unfolding *content-eq-0 interior-cbox box-eq-empty*
by *auto*

lemma *content-pos-lt-eq*:
0 < content (cbox a (b::'a::euclidean-space)) ⟷ (∀ i∈Basis. a•i < b•i)
proof (*rule iffI*)
assume *0 < content (cbox a b)*
then have *content (cbox a b) ≠ 0* **by** *auto*
then show *∀ i∈Basis. a•i < b•i*
unfolding *content-eq-0 not-ex not-le* **by** *fastforce*
next
assume *∀ i∈Basis. a • i < b • i*
then show *0 < content (cbox a b)*
by (*metis content-pos-lt*)
qed

lemma *content-empty* [*simp*]: *content {} = 0*
unfolding *content-def* **by** *auto*

lemma *content-real-if* [*simp*]: *content {a..b} = (if a ≤ b then b − a else 0)*
by (*simp add*: *content-real*)

lemma *content-subset*:
assumes *cbox a b ⊆ cbox c d*
shows *content (cbox a b) ≤ content (cbox c d)*
proof (*cases cbox a b = {}*)
case *True*
then show *?thesis*
using *content-pos-le*[*of c d*] **by** *auto*
next
case *False*
then have *ab-ne*: *∀ i∈Basis. a • i ≤ b • i*
unfolding *box-ne-empty* **by** *auto*
then have *ab-ab*: *a∈cbox a b b∈cbox a b*
unfolding *mem-box* **by** *auto*
have *cbox c d ≠ {}* **using** *assms False* **by** *auto*
then have *cd-ne*: *∀ i∈Basis. c • i ≤ d • i*
using *assms* **unfolding** *box-ne-empty* **by** *auto*
have *⋀i. i ∈ Basis ⟹ 0 ≤ b • i − a • i*
using *ab-ne* **by** *auto*
moreover
have *⋀i. i ∈ Basis ⟹ b • i − a • i ≤ d • i − c • i*
using *assms*[*unfolded subset-eq mem-box*,*rule-format*,*OF ab-ab*(*2*)]
assms[*unfolded subset-eq mem-box*,*rule-format*,*OF ab-ab*(*1*)]

```
    by (metis diff-mono)
  ultimately show ?thesis
    unfolding content-def interval-bounds[OF ab-ne] interval-bounds[OF cd-ne]
    by (simp add: setprod-mono if-not-P[OF False] if-not-P[OF ‹cbox c d ≠ {}›])
qed

lemma content-lt-nz: 0 < content (cbox a b) ⟷ content (cbox a b) ≠ 0
  unfolding content-pos-lt-eq content-eq-0 unfolding not-ex not-le by fastforce

lemma content-times[simp]: content (A × B) = content A * content B
proof (cases A × B = {})
  let ?ub1 = interval-upperbound and ?lb1 = interval-lowerbound
  let ?ub2 = interval-upperbound and ?lb2 = interval-lowerbound
  assume nonempty: A × B ≠ {}
  hence content (A × B) = (∏ i∈Basis. (?ub1 A, ?ub2 B) · i − (?lb1 A, ?lb2 B)
· i)
    unfolding content-def by (simp add: interval-upperbound-Times interval-lowerbound-Times)
  also have ... = content A * content B unfolding content-def using nonempty
    apply (subst Basis-prod-def, subst setprod.union-disjoint, force, force, force,
simp)
    apply (subst (1 2) setprod.reindex, auto intro: inj-onI)
    done
  finally show ?thesis .
qed (auto simp: content-def)

lemma content-Pair: content (cbox (a,c) (b,d)) = content (cbox a b) * content
(cbox c d)
  by (simp add: cbox-Pair-eq)

lemma content-cbox-pair-eq0-D:
   content (cbox (a,c) (b,d)) = 0 ⟹ content (cbox a b) = 0 ∨ content (cbox c d)
= 0
  by (simp add: content-Pair)

lemma content-eq-0-gen:
  fixes s :: 'a::euclidean-space set
  assumes bounded s
  shows content s = 0 ⟷ (∃ i∈Basis. ∃ v. ∀ x ∈ s. x · i = v)  (is - = ?rhs)
proof safe
  assume content s = 0 then show ?rhs
    apply (clarsimp simp: ex-in-conv content-def split: split-if-asm)
    apply (rule-tac x=a in bexI)
    apply (rule-tac x=interval-lowerbound s · a in exI)
    apply (clarsimp simp: interval-upperbound-def interval-lowerbound-def)
    apply (drule cSUP-eq-cINF-D)
   apply (auto simp: bounded-inner-imp-bdd-above [OF assms]  bounded-inner-imp-bdd-below
[OF assms])
    done
next
```

fix *i a*
assume *i* ∈ *Basis* ∀ *x*∈*s*. *x* ∙ *i* = *a*
then show *content s* = *0*
apply (*clarsimp simp*: *content-def*)
apply (*rule-tac x*=*i* **in** *bexI*)
apply (*auto simp*: *interval-upperbound-def interval-lowerbound-def*)
done
qed

lemma *content-0-subset-gen*:
fixes *a* :: *'a*::*euclidean-space*
assumes *content t* = *0 s* ⊆ *t bounded t* **shows** *content s* = *0*
proof −
have *bounded s*
using *assms* **by** (*metis bounded-subset*)
then show *?thesis*
using *assms*
by (*auto simp*: *content-eq-0-gen*)
qed

lemma *content-0-subset*: ⟦*content*(*cbox a b*) = *0*; *s* ⊆ *cbox a b*⟧ ⟹ *content s* = *0*
by (*simp add*: *content-0-subset-gen bounded-cbox*)

35.5 The notion of a gauge — simply an open set containing the point.

definition *gauge d* ⟷ (∀ *x*. *x* ∈ *d x* ∧ *open* (*d x*))

lemma *gaugeI*:
assumes ⋀*x*. *x* ∈ *g x*
and ⋀*x*. *open* (*g x*)
shows *gauge g*
using *assms* **unfolding** *gauge-def* **by** *auto*

lemma *gaugeD*[*dest*]:
assumes *gauge d*
shows *x* ∈ *d x*
and *open* (*d x*)
using *assms* **unfolding** *gauge-def* **by** *auto*

lemma *gauge-ball-dependent*: ∀ *x*. *0* < *e x* ⟹ *gauge* (λ*x*. *ball x* (*e x*))
unfolding *gauge-def* **by** *auto*

lemma *gauge-ball*[*intro*]: *0* < *e* ⟹ *gauge* (λ*x*. *ball x e*)
unfolding *gauge-def* **by** *auto*

lemma *gauge-trivial*[*intro*!]: *gauge* (λ*x*. *ball x 1*)
by (*rule gauge-ball*) *auto*

lemma *gauge-inter*[*intro*]: *gauge d1* $\Longrightarrow$ *gauge d2* $\Longrightarrow$ *gauge* ($\lambda x.$ *d1 x* $\cap$ *d2 x*)
unfolding *gauge-def* **by** *auto*

lemma *gauge-inters*:
assumes *finite s*
and $\forall\, d{\in}s.$ *gauge* (*f d*)
shows *gauge* ($\lambda x.$ $\bigcap\{f\ d\ x \mid d.\ d \in s\}$)
proof −
have ∗: $\bigwedge x.$ $\{f\ d\ x \mid d.\ d \in s\} = (\lambda d.\ f\ d\ x)$ ‘ *s*
by *auto*
show *?thesis*
unfolding *gauge-def* **unfolding** ∗
using *assms* **unfolding** *Ball-def Inter-iff mem-Collect-eq gauge-def* **by** *auto*
qed

lemma *gauge-existence-lemma*:
($\forall\, x.\ \exists\, d ::$ *real*. *p x* $\longrightarrow$ *0* $<$ *d* $\wedge$ *q d x*) $\longleftrightarrow$ ($\forall\, x.\ \exists\, d{>}0.$ *p x* $\longrightarrow$ *q d x*)
by (*metis zero-less-one*)

35.6 Divisions.

definition *division-of* (**infixl** *division′-of 40*)
where
s division-of i $\longleftrightarrow$
finite s $\wedge$
($\forall\, k{\in}s.$ *k* $\subseteq$ *i* $\wedge$ *k* $\neq$ {} $\wedge$ ($\exists$ *a b.* *k* = *cbox a b*)) $\wedge$
($\forall\, k1{\in}s.\ \forall\, k2{\in}s.$ *k1* $\neq$ *k2* $\longrightarrow$ *interior*(*k1*) $\cap$ *interior*(*k2*) = {}) $\wedge$
($\bigcup s = i$)

lemma *division-ofD*[*dest*]:
assumes *s division-of i*
shows *finite s*
and $\bigwedge k.$ *k* $\in$ *s* $\Longrightarrow$ *k* $\subseteq$ *i*
and $\bigwedge k.$ *k* $\in$ *s* $\Longrightarrow$ *k* $\neq$ {}
and $\bigwedge k.$ *k* $\in$ *s* $\Longrightarrow$ $\exists$ *a b.* *k* = *cbox a b*
and $\bigwedge k1\ k2.$ *k1* $\in$ *s* $\Longrightarrow$ *k2* $\in$ *s* $\Longrightarrow$ *k1* $\neq$ *k2* $\Longrightarrow$ *interior*(*k1*) $\cap$ *interior*(*k2*)
= {}
and $\bigcup s = i$
using *assms* **unfolding** *division-of-def* **by** *auto*

lemma *division-ofI*:
assumes *finite s*
and $\bigwedge k.$ *k* $\in$ *s* $\Longrightarrow$ *k* $\subseteq$ *i*
and $\bigwedge k.$ *k* $\in$ *s* $\Longrightarrow$ *k* $\neq$ {}
and $\bigwedge k.$ *k* $\in$ *s* $\Longrightarrow$ $\exists$ *a b.* *k* = *cbox a b*
and $\bigwedge k1\ k2.$ *k1* $\in$ *s* $\Longrightarrow$ *k2* $\in$ *s* $\Longrightarrow$ *k1* $\neq$ *k2* $\Longrightarrow$ *interior k1* $\cap$ *interior k2* =
{}
and $\bigcup s = i$
shows *s division-of i*

using *assms* **unfolding** *division-of-def* **by** *auto*

lemma *division-of-finite*: *s division-of i* $\Longrightarrow$ *finite s*
unfolding *division-of-def* **by** *auto*

lemma *division-of-self*[*intro*]: *cbox a b* $\neq$ {} $\Longrightarrow$ {*cbox a b*} *division-of* (*cbox a b*)
unfolding *division-of-def* **by** *auto*

lemma *division-of-trivial*[*simp*]: *s division-of* {} $\longleftrightarrow$ *s* = {}
unfolding *division-of-def* **by** *auto*

lemma *division-of-sing*[*simp*]:
s division-of cbox a (*a*::'*a*::*euclidean-space*) $\longleftrightarrow$ *s* = {*cbox a a*}
(**is** *?l* = *?r*)
proof
assume *?r*
moreover
{ fix *k*
assume *s* = {{*a*}} *k*$\in$*s*
then have $\exists$ *x y*. *k* = *cbox x y*
apply (*rule-tac x*=*a* **in** *exI*)+
apply (*force simp*: *cbox-sing*)
done
}
ultimately show *?l*
unfolding *division-of-def cbox-sing* **by** *auto*
next
assume *?l*
note $*$ = *conjunctD4*[*OF this*[*unfolded division-of-def cbox-sing*]]
{
fix *x*
assume *x*: *x* $\in$ *s* **have** *x* = {*a*}
using $*$(*2*)[*rule-format*,*OF x*] **by** *auto*
}
moreover have *s* $\neq$ {}
using $*$(*4*) **by** *auto*
ultimately show *?r*
unfolding *cbox-sing* **by** *auto*
qed

lemma *elementary-empty*: **obtains** *p* **where** *p division-of* {}
unfolding *division-of-trivial* **by** *auto*

lemma *elementary-interval*: **obtains** *p* **where** *p division-of* (*cbox a b*)
by (*metis division-of-trivial division-of-self*)

lemma *division-contains*: *s division-of i* $\Longrightarrow$ $\forall$ *x*$\in$*i*. $\exists$ *k*$\in$*s*. *x* $\in$ *k*
unfolding *division-of-def* **by** *auto*

lemma *forall-in-division*:
d division-of i $\Longrightarrow$ $(\forall\, x{\in}d.\ P\ x) \longleftrightarrow (\forall\, a\ b.\ cbox\ a\ b \in d \longrightarrow P\ (cbox\ a\ b))$
unfolding *division-of-def* **by** *fastforce*

lemma *division-of-subset*:
assumes *p division-of* $(\bigcup p)$
and $q \subseteq p$
shows *q division-of* $(\bigcup q)$
proof (*rule division-ofI*)
note $*$ = *division-ofD*[*OF assms*(*1*)]
show *finite q*
using $*$(*1*) *assms*(*2*) *infinite-super* **by** *auto*
{
fix *k*
assume $k \in q$
then have *kp*: $k \in p$
using *assms*(*2*) **by** *auto*
show $k \subseteq \bigcup q$
using ‹$k \in q$› **by** *auto*
show $\exists\, a\ b.\ k = cbox\ a\ b$
using $*$(*4*)[*OF kp*] **by** *auto*
show $k \neq \{\}$
using $*$(*3*)[*OF kp*] **by** *auto*
}
fix *k1 k2*
assume $k1 \in q\ k2 \in q\ k1 \neq k2$
then have $**$: $k1 \in p\ k2 \in p\ k1 \neq k2$
using *assms*(*2*) **by** *auto*
show *interior k1* $\cap$ *interior k2* $= \{\}$
using $*$(*5*)[*OF* $**$] **by** *auto*
qed *auto*

lemma *division-of-union-self*[*intro*]: *p division-of s* $\Longrightarrow$ *p division-of* $(\bigcup p)$
unfolding *division-of-def* **by** *auto*

lemma *division-of-content-0*:
assumes *content* $(cbox\ a\ b) = 0$ *d division-of* $(cbox\ a\ b)$
shows $\forall\, k{\in}d.$ *content* $k = 0$
unfolding *forall-in-division*[*OF assms*(*2*)]
by (*metis antisym-conv assms content-pos-le content-subset division-ofD*(*2*))

lemma *division-inter*:
fixes *s1 s2* :: *'a*::*euclidean-space set*
assumes *p1 division-of s1*
and *p2 division-of s2*
shows $\{k1 \cap k2 \mid k1\ k2\ .k1 \in p1 \wedge k2 \in p2 \wedge k1 \cap k2 \neq \{\}\}$ *division-of* $(s1 \cap s2)$
(**is** *?A' division-of* -)
proof −

let *?A* = {*s*. *s* ∈ (λ(*k1*,*k2*). *k1* ∩ *k2*) ‘ (*p1* × *p2*) ∧ *s* ≠ {}}
have ∗: *?A′* = *?A* **by** *auto*
show *?thesis*
unfolding ∗
proof (*rule division-ofI*)
have *?A* ⊆ (λ(*x*, *y*). *x* ∩ *y*) ‘ (*p1* × *p2*)
by *auto*
moreover have *finite* (*p1* × *p2*)
using *assms* **unfolding** *division-of-def* **by** *auto*
ultimately show *finite ?A* **by** *auto*
have ∗: ⋀*s*. ⋃{*x*∈*s*. *x* ≠ {}} = ⋃*s*
by *auto*
show ⋃ *?A* = *s1* ∩ *s2*
apply (*rule set-eqI*)
unfolding ∗ **and** *Union-image-eq UN-iff*
using *division-ofD*(*6*)[*OF assms*(*1*)] **and** *division-ofD*(*6*)[*OF assms*(*2*)]
apply *auto*
done
{
fix *k*
assume *k* ∈ *?A*
then obtain *k1 k2* **where** *k*: *k* = *k1* ∩ *k2 k1* ∈ *p1 k2* ∈ *p2 k* ≠ {}
by *auto*
then show *k* ≠ {}
by *auto*
show *k* ⊆ *s1* ∩ *s2*
using *division-ofD*(*2*)[*OF assms*(*1*) *k*(*2*)] **and** *division-ofD*(*2*)[*OF assms*(*2*) *k*(*3*)]
unfolding *k* **by** *auto*
obtain *a1 b1* **where** *k1*: *k1* = *cbox a1 b1*
using *division-ofD*(*4*)[*OF assms*(*1*) *k*(*2*)] **by** *blast*
obtain *a2 b2* **where** *k2*: *k2* = *cbox a2 b2*
using *division-ofD*(*4*)[*OF assms*(*2*) *k*(*3*)] **by** *blast*
show ∃ *a b*. *k* = *cbox a b*
unfolding *k k1 k2* **unfolding** *inter-interval* **by** *auto*
}
fix *k1 k2*
assume *k1* ∈ *?A*
then obtain *x1 y1* **where** *k1*: *k1* = *x1* ∩ *y1 x1* ∈ *p1 y1* ∈ *p2 k1* ≠ {}
by *auto*
assume *k2* ∈ *?A*
then obtain *x2 y2* **where** *k2*: *k2* = *x2* ∩ *y2 x2* ∈ *p1 y2* ∈ *p2 k2* ≠ {}
by *auto*
assume *k1* ≠ *k2*
then have *th*: *x1* ≠ *x2* ∨ *y1* ≠ *y2*
unfolding *k1 k2* **by** *auto*
have ∗: *interior x1* ∩ *interior x2* = {} ∨ *interior y1* ∩ *interior y2* = {} ⟹
interior (*x1* ∩ *y1*) ⊆ *interior x1* ⟹ *interior* (*x1* ∩ *y1*) ⊆ *interior y1* ⟹
interior (*x2* ∩ *y2*) ⊆ *interior x2* ⟹ *interior* (*x2* ∩ *y2*) ⊆ *interior y2* ⟹

```
      interior (x1 ∩ y1) ∩ interior (x2 ∩ y2) = {} by auto
    show interior k1 ∩ interior k2 = {}
      unfolding k1 k2
      apply (rule *)
      using assms division-ofD(5) k1 k2(2) k2(3) th apply auto
      done
  qed
qed

lemma division-inter-1:
  assumes d division-of i
    and cbox a (b::'a::euclidean-space) ⊆ i
  shows {cbox a b ∩ k | k. k ∈ d ∧ cbox a b ∩ k ≠ {}} division-of (cbox a b)
proof (cases cbox a b = {})
  case True
  show ?thesis
    unfolding True and division-of-trivial by auto
next
  case False
  have *: cbox a b ∩ i = cbox a b using assms(2) by auto
  show ?thesis
    using division-inter[OF division-of-self[OF False] assms(1)]
    unfolding * by auto
qed

lemma elementary-inter:
  fixes s t :: 'a::euclidean-space set
  assumes p1 division-of s
    and p2 division-of t
  shows ∃p. p division-of (s ∩ t)
using assms division-inter by blast

lemma elementary-inters:
  assumes finite f
    and f ≠ {}
    and ∀s∈f. ∃p. p division-of (s::('a::euclidean-space) set)
  shows ∃p. p division-of (⋂f)
  using assms
proof (induct f rule: finite-induct)
  case (insert x f)
  show ?case
  proof (cases f = {})
    case True
    then show ?thesis
      unfolding True using insert by auto
  next
    case False
    obtain p where p division-of ⋂f
      using insert(3)[OF False insert(5)[unfolded ball-simps,THEN conjunct2]] ..
```

```
    moreover obtain px where px division-of x
      using insert(5)[rule-format,OF insertI1] ..
    ultimately show ?thesis
      by (simp add: elementary-inter Inter-insert)
  qed
qed auto

lemma division-disjoint-union:
  assumes p1 division-of s1
    and p2 division-of s2
    and interior s1 ∩ interior s2 = {}
  shows (p1 ∪ p2) division-of (s1 ∪ s2)
proof (rule division-ofI)
  note d1 = division-ofD[OF assms(1)]
  note d2 = division-ofD[OF assms(2)]
  show finite (p1 ∪ p2)
    using d1(1) d2(1) by auto
  show ⋃(p1 ∪ p2) = s1 ∪ s2
    using d1(6) d2(6) by auto
  {
    fix k1 k2
    assume as: k1 ∈ p1 ∪ p2 k2 ∈ p1 ∪ p2 k1 ≠ k2
    moreover
    let ?g=interior k1 ∩ interior k2 = {}
    {
      assume as: k1∈p1 k2∈p2
      have ?g
          using interior-mono[OF d1(2)[OF as(1)]] interior-mono[OF d2(2)[OF
as(2)]]
        using assms(3) by blast
    }
    moreover
    {
      assume as: k1∈p2 k2∈p1
      have ?g
          using interior-mono[OF d1(2)[OF as(2)]] interior-mono[OF d2(2)[OF
as(1)]]
        using assms(3) by blast
    }
    ultimately show ?g
      using d1(5)[OF - - as(3)] and d2(5)[OF - - as(3)] by auto
  }
  fix k
  assume k: k ∈ p1 ∪ p2
  show k ⊆ s1 ∪ s2
    using k d1(2) d2(2) by auto
  show k ≠ {}
    using k d1(3) d2(3) by auto
  show ∃ a b. k = cbox a b
```

```
    using k d1(4) d2(4) by auto
qed

lemma partial-division-extend-1:
  fixes a b c d :: 'a::euclidean-space
  assumes incl: cbox c d ⊆ cbox a b
    and nonempty: cbox c d ≠ {}
  obtains p where p division-of (cbox a b) cbox c d ∈ p
proof
  let ?B = λf::'a⇒'a × 'a.
    cbox (∑ i∈Basis. (fst (f i) · i) *R i) (∑ i∈Basis. (snd (f i) · i) *R i)
  def p ≡ ?B ' (Basis →E {(a, c), (c, d), (d, b)})

  show cbox c d ∈ p
    unfolding p-def
   by (auto simp add: box-eq-empty cbox-def intro!: image-eqI[where x=λ(i::'a)∈Basis.
(c, d)])
  {
    fix i :: 'a
    assume i ∈ Basis
    with incl nonempty have a · i ≤ c · i c · i ≤ d · i d · i ≤ b · i
      unfolding box-eq-empty subset-box by (auto simp: not-le)
  }
  note ord = this

  show p division-of (cbox a b)
  proof (rule division-ofI)
    show finite p
      unfolding p-def by (auto intro!: finite-PiE)
    {
      fix k
      assume k ∈ p
      then obtain f where f: f ∈ Basis →E {(a, c), (c, d), (d, b)} and k: k =
?B f
        by (auto simp: p-def)
      then show ∃ a b. k = cbox a b
        by auto
      have k ⊆ cbox a b ∧ k ≠ {}
      proof (simp add: k box-eq-empty subset-box not-less, safe)
        fix i :: 'a
        assume i: i ∈ Basis
        with f have f i = (a, c) ∨ f i = (c, d) ∨ f i = (d, b)
          by (auto simp: PiE-iff)
        with i ord[of i]
        show a · i ≤ fst (f i) · i snd (f i) · i ≤ b · i fst (f i) · i ≤ snd (f i) · i
          by auto
      qed
      then show k ≠ {} k ⊆ cbox a b
        by auto
```

```
    {
      fix l
      assume l ∈ p
      then obtain g where g: g ∈ Basis →E {(a, c), (c, d), (d, b)} and l: l =
?B g
        by (auto simp: p-def)
      assume l ≠ k
      have ∃ i∈Basis. f i ≠ g i
      proof (rule ccontr)
        assume ¬ ?thesis
        with f g have f = g
          by (auto simp: PiE-iff extensional-def intro!: ext)
        with ‹l ≠ k› show False
          by (simp add: l k)
      qed
      then obtain i where ∗: i ∈ Basis f i ≠ g i ..
      then have f i = (a, c) ∨ f i = (c, d) ∨ f i = (d, b)
                g i = (a, c) ∨ g i = (c, d) ∨ g i = (d, b)
        using f g by (auto simp: PiE-iff)
      with ∗ ord[of i] show interior l ∩ interior k = {}
        by (auto simp add: l k interior-cbox disjoint-interval intro!: bexI[of - i])
    }
    note ‹k ⊆ cbox a b›
  }
  moreover
  {
    fix x assume x: x ∈ cbox a b
    have ∀ i∈Basis. ∃ l. x · i ∈ {fst l · i .. snd l · i} ∧ l ∈ {(a, c), (c, d), (d, b)}
    proof
      fix i :: 'a
      assume i ∈ Basis
      with x ord[of i]
      have (a · i ≤ x · i ∧ x · i ≤ c · i) ∨ (c · i ≤ x · i ∧ x · i ≤ d · i) ∨
          (d · i ≤ x · i ∧ x · i ≤ b · i)
        by (auto simp: cbox-def)
      then show ∃ l. x · i ∈ {fst l · i .. snd l · i} ∧ l ∈ {(a, c), (c, d), (d, b)}
        by auto
    qed
    then obtain f where
      f: ∀ i∈Basis. x · i ∈ {fst (f i) · i..snd (f i) · i} ∧ f i ∈ {(a, c), (c, d), (d,
b)}
      unfolding bchoice-iff ..
    moreover from f have restrict f Basis ∈ Basis →E {(a, c), (c, d), (d, b)}
      by auto
    moreover from f have x ∈ ?B (restrict f Basis)
      by (auto simp: mem-box)
    ultimately have ∃ k∈p. x ∈ k
      unfolding p-def by blast
  }
```

```
    ultimately show ⋃p = cbox a b
      by auto
  qed
qed

lemma partial-division-extend-interval:
  assumes p division-of (⋃p) (⋃p) ⊆ cbox a b
  obtains q where p ⊆ q q division-of cbox a (b::'a::euclidean-space)
proof (cases p = {})
  case True
  obtain q where q division-of (cbox a b)
    by (rule elementary-interval)
  then show ?thesis
    using True that by blast
next
  case False
  note p = division-ofD[OF assms(1)]
  have div-cbox: ∀k∈p. ∃q. q division-of cbox a b ∧ k ∈ q
  proof
    fix k
    assume kp: k ∈ p
    obtain c d where k: k = cbox c d
      using p(4)[OF kp] by blast
    have ∗: cbox c d ⊆ cbox a b cbox c d ≠ {}
      using p(2,3)[OF kp, unfolded k] using assms(2)
      by (blast intro: order.trans)+
    obtain q where q division-of cbox a b cbox c d ∈ q
      by (rule partial-division-extend-1[OF ∗])
    then show ∃q. q division-of cbox a b ∧ k ∈ q
      unfolding k by auto
  qed
  obtain q where q: ⋀x. x ∈ p ⟹ q x division-of cbox a b ⋀x. x ∈ p ⟹ x ∈
q x
    using bchoice[OF div-cbox] by blast
  { fix x
    assume x: x ∈ p
    have q x division-of ⋃q x
      apply (rule division-ofI)
      using division-ofD[OF q(1)[OF x]]
      apply auto
      done }
  then have ⋀x. x ∈ p ⟹ ∃d. d division-of ⋃(q x − {x})
    by (meson Diff-subset division-of-subset)
  then have ∃d. d division-of ⋂((λi. ⋃(q i − {i})) ‘ p)
    apply −
    apply (rule elementary-inters [OF finite-imageI[OF p(1)]])
    apply (auto simp: False elementary-inters [OF finite-imageI[OF p(1)]])
    done
  then obtain d where d: d division-of ⋂((λi. ⋃(q i − {i})) ‘ p) ..
```

```
  have d ∪ p division-of cbox a b
  proof −
    have te: ⋀s f t. s ≠ {} ⟹ ∀i∈s. f i ∪ i = t ⟹ t = ⋂(f ‘ s) ∪ ⋃s by auto
    have cbox-eq: cbox a b = ⋂((λi. ⋃(q i − {i})) ‘ p) ∪ ⋃p
    proof (rule te[OF False], clarify)
      fix i
      assume i: i ∈ p
      show ⋃(q i − {i}) ∪ i = cbox a b
        using division-ofD(6)[OF q(1)[OF i]] using q(2)[OF i] by auto
    qed
    { fix k
      assume k: k ∈ p
      have ∗: ⋀u t s. t ∩ s = {} ⟹ u ⊆ s ⟹ u ∩ t = {}
        by auto
      have interior (⋂i∈p. ⋃(q i − {i})) ∩ interior k = {}
      proof (rule ∗[OF inter-interior-unions-intervals])
        note qk=division-ofD[OF q(1)[OF k]]
       show finite (q k − {k}) open (interior k) ∀t∈q k − {k}. ∃a b. t = cbox a b
          using qk by auto
        show ∀t∈q k − {k}. interior k ∩ interior t = {}
          using qk(5) using q(2)[OF k] by auto
        show interior (⋂i∈p. ⋃(q i − {i})) ⊆ interior (⋃(q k − {k}))
          apply (rule interior-mono)+
          using k
          apply auto
          done
      qed } note [simp] = this
    show d ∪ p division-of (cbox a b)
      unfolding cbox-eq
      apply (rule division-disjoint-union[OF d assms(1)])
      apply (rule inter-interior-unions-intervals)
      apply (rule p open-interior ballI)+
      apply simp-all
      done
  qed
  then show ?thesis
    by (meson Un-upper2 that)
qed

lemma elementary-bounded[dest]:
  fixes s :: 'a::euclidean-space set
  shows p division-of s ⟹ bounded s
  unfolding division-of-def by (metis bounded-Union bounded-cbox)

lemma elementary-subset-cbox:
  p division-of s ⟹ ∃a b. s ⊆ cbox a (b::'a::euclidean-space)
  by (meson elementary-bounded bounded-subset-cbox)

lemma division-union-intervals-exists:
```

fixes *a b* :: *'a*::*euclidean-space*
assumes *cbox a b* ≠ {}
obtains *p* **where** (*insert* (*cbox a b*) *p*) *division-of* (*cbox a b* ∪ *cbox c d*)
proof (*cases cbox c d* = {})
case *True*
show *?thesis*
apply (*rule that*[*of* {}])
unfolding *True*
using *assms*
apply *auto*
done
next
case *False*
show *?thesis*
proof (*cases cbox a b* ∩ *cbox c d* = {})
case *True*
show *?thesis*
apply (*rule that*[*of* {*cbox c d*}])
apply (*subst insert-is-Un*)
apply (*rule division-disjoint-union*)
using ‹*cbox c d* ≠ {}› *True assms interior-subset*
apply *auto*
done
next
case *False*
obtain *u v* **where** *uv*: *cbox a b* ∩ *cbox c d* = *cbox u v*
unfolding *inter-interval* **by** *auto*
have *uv-sub*: *cbox u v* ⊆ *cbox c d* **using** *uv* **by** *auto*
obtain *p* **where** *p division-of cbox c d cbox u v* ∈ *p*
by (*rule partial-division-extend-1*[*OF uv-sub False*[*unfolded uv*]])
note *p* = *this division-ofD*[*OF this*(*1*)]
have *interior* (*cbox a b* ∩ ⋃(*p* − {*cbox u v*})) = *interior*(*cbox u v* ∩ ⋃(*p* − {*cbox u v*}))
apply (*rule arg-cong*[*of - - interior*])
using *p*(*8*) *uv* **by** *auto*
also have ... = {}
unfolding *interior-Int*
apply (*rule inter-interior-unions-intervals*)
using *p*(*6*) *p*(*7*)[*OF p*(*2*)] *p*(*3*)
apply *auto*
done
finally have [*simp*]: *interior* (*cbox a b*) ∩ *interior* (⋃(*p* − {*cbox u v*})) = {}
by *simp*
have *cbe*: *cbox a b* ∪ *cbox c d* = *cbox a b* ∪ ⋃(*p* − {*cbox u v*})
using *p*(*8*) **unfolding** *uv*[*symmetric*] **by** *auto*
show *?thesis*
apply (*rule that*[*of p* − {*cbox u v*}])
apply (*simp add*: *cbe*)
apply (*subst insert-is-Un*)

```
    apply (rule division-disjoint-union)
    apply (simp-all add: assms division-of-self)
    by (metis Diff-subset division-of-subset p(1) p(8))
  qed
qed

lemma division-of-unions:
  assumes finite f
    and ⋀p. p ∈ f ⟹ p division-of (⋃p)
    and ⋀k1 k2. k1 ∈ ⋃f ⟹ k2 ∈ ⋃f ⟹ k1 ≠ k2 ⟹ interior k1 ∩ interior
k2 = {}
  shows ⋃f division-of ⋃⋃f
  using assms
  by (auto intro!: division-ofI)

lemma elementary-union-interval:
  fixes a b :: 'a::euclidean-space
  assumes p division-of ⋃p
  obtains q where q division-of (cbox a b ∪ ⋃p)
proof −
  note assm = division-ofD[OF assms]
  have lem1: ⋀f s. ⋃⋃(f ‘ s) = ⋃((λx. ⋃(f x)) ‘ s)
    by auto
  have lem2: ⋀f s. f ≠ {} ⟹ ⋃{s ∪ t |t. t ∈ f} = s ∪ ⋃f
    by auto
  {
    presume p = {} ⟹ thesis
      cbox a b = {} ⟹ thesis
      cbox a b ≠ {} ⟹ interior (cbox a b) = {} ⟹ thesis
      p ≠ {} ⟹ interior (cbox a b)≠{} ⟹ cbox a b ≠ {} ⟹ thesis
    then show thesis by auto
  next
    assume as: p = {}
    obtain p where p division-of (cbox a b)
      by (rule elementary-interval)
    then show thesis
      using as that by auto
  next
    assume as: cbox a b = {}
    show thesis
      using as assms that by auto
  next
    assume as: interior (cbox a b) = {} cbox a b ≠ {}
    show thesis
      apply (rule that[of insert (cbox a b) p],rule division-ofI)
      unfolding finite-insert
      apply (rule assm(1)) unfolding Union-insert
      using assm(2−4) as
      apply −
```

```
    apply (fast dest: assm(5))+
    done
  next
   assume as: p ≠ {} interior (cbox a b) ≠ {} cbox a b ≠ {}
   have ∀ k∈p. ∃ q. (insert (cbox a b) q) division-of (cbox a b ∪ k)
   proof
    fix k
    assume kp: k ∈ p
    from assm(4)[OF kp] obtain c d where k = cbox c d by blast
    then show ∃ q. (insert (cbox a b) q) division-of (cbox a b ∪ k)
     by (meson as(3) division-union-intervals-exists)
   qed
  from bchoice[OF this] obtain q where ∀ x∈p. insert (cbox a b) (q x) division-of
(cbox a b) ∪ x ..
   note q = division-ofD[OF this[rule-format]]
   let ?D = ⋃{insert (cbox a b) (q k) | k. k ∈ p}
   show thesis
   proof (rule that[OF division-ofI])
    have *: {insert (cbox a b) (q k) |k. k ∈ p} = (λk. insert (cbox a b) (q k)) ‘ p
     by auto
    show finite ?D
     using * assm(1) q(1) by auto
    show ⋃ ?D = cbox a b ∪ ⋃ p
     unfolding * lem1
     unfolding lem2[OF as(1), of cbox a b, symmetric]
     using q(6)
     by auto
    fix k
    assume k: k ∈ ?D
    then show k ⊆ cbox a b ∪ ⋃ p
     using q(2) by auto
    show k ≠ {}
     using q(3) k by auto
    show ∃ a b. k = cbox a b
     using q(4) k by auto
    fix k'
    assume k': k' ∈ ?D k ≠ k'
    obtain x where x: k ∈ insert (cbox a b) (q x) x∈p
     using k by auto
    obtain x' where x': k'∈insert (cbox a b) (q x') x'∈p
     using k' by auto
    show interior k ∩ interior k' = {}
    proof (cases x = x')
     case True
     show ?thesis
      using True k' q(5) x' x by auto
    next
     case False
     {
```

```
      presume k = cbox a b ⟹ ?thesis
        and k' = cbox a b ⟹ ?thesis
        and k ≠ cbox a b ⟹ k' ≠ cbox a b ⟹ ?thesis
      then show ?thesis by auto
    next
      assume as': k  = cbox a b
      show ?thesis
        using as' k' q(5) x' by auto
    next
      assume as': k' = cbox a b
      show ?thesis
        using as' k'(2) q(5) x by auto
    }
    assume as': k ≠ cbox a b k' ≠ cbox a b
    obtain c d where k: k = cbox c d
      using q(4)[OF x(2,1)] by blast
    have interior k ∩ interior (cbox a b) = {}
      using as' k'(2) q(5) x by auto
    then have interior k ⊆ interior x
    using interior-subset-union-intervals
      by (metis as(2) k q(2) x interior-subset-union-intervals)
    moreover
    obtain c d where c-d: k' = cbox c d
      using q(4)[OF x'(2,1)] by blast
    have interior k' ∩ interior (cbox a b) = {}
      using as'(2) q(5) x' by auto
    then have interior k' ⊆ interior x'
      by (metis as(2) c-d interior-subset-union-intervals q(2) x'(1) x'(2))
    ultimately show ?thesis
      using assm(5)[OF x(2) x'(2) False] by auto
  qed
 qed
}
qed

lemma elementary-unions-intervals:
 assumes fin: finite f
   and ⋀s. s ∈ f ⟹ ∃ a b. s = cbox a (b::'a::euclidean-space)
 obtains p where p division-of (⋃f)
proof −
 have ∃ p. p division-of (⋃f)
 proof (induct-tac f rule:finite-subset-induct)
   show ∃ p. p division-of ⋃{} using elementary-empty by auto
 next
   fix x F
   assume as: finite F x ∉ F ∃ p. p division-of ⋃F x∈f
   from this(3) obtain p where p: p division-of ⋃F ..
   from assms(2)[OF as(4)] obtain a b where x: x = cbox a b by blast
   have ∗: ⋃F = ⋃p
```

```
      using division-ofD[OF p] by auto
    show ∃p. p division-of ⋃insert x F
      using elementary-union-interval[OF p[unfolded *], of a b]
      unfolding Union-insert x * by metis
  qed (insert assms, auto)
  then show ?thesis
    using that by auto
qed

lemma elementary-union:
  fixes s t :: 'a::euclidean-space set
  assumes ps division-of s pt division-of t
  obtains p where p division-of (s ∪ t)
proof −
  have *: s ∪ t = ⋃ps ∪ ⋃pt
    using assms unfolding division-of-def by auto
  show ?thesis
    apply (rule elementary-unions-intervals[of ps ∪ pt])
    using assms apply auto
    by (simp add: * that)
qed

lemma partial-division-extend:
  fixes t :: 'a::euclidean-space set
  assumes p division-of s
    and q division-of t
    and s ⊆ t
  obtains r where p ⊆ r and r division-of t
proof −
  note divp = division-ofD[OF assms(1)] and divq = division-ofD[OF assms(2)]
  obtain a b where ab: t ⊆ cbox a b
    using elementary-subset-cbox[OF assms(2)] by auto
  obtain r1 where p ⊆ r1 r1 division-of (cbox a b)
    using assms
    by (metis ab dual-order.trans partial-division-extend-interval divp(6))
  note r1 = this division-ofD[OF this(2)]
  obtain p' where p' division-of ⋃(r1 − p)
    apply (rule elementary-unions-intervals[of r1 − p])
    using r1(3,6)
    apply auto
    done
  then obtain r2 where r2: r2 division-of (⋃(r1 − p)) ∩ (⋃q)
    by (metis assms(2) divq(6) elementary-inter)
  {
    fix x
    assume x: x ∈ t x ∉ s
    then have x∈⋃r1
      unfolding r1 using ab by auto
    then obtain r where r: r ∈ r1 x ∈ r
```

```
      unfolding Union-iff ..
    moreover
    have r ∉ p
    proof
      assume r ∈ p
      then have x ∈ s using divp(2) r by auto
      then show False using x by auto
    qed
    ultimately have x∈⋃(r1 − p) by auto
  }
  then have ∗: t = ⋃p ∪ (⋃(r1 − p) ∩ ⋃q)
    unfolding divp divq using assms(3) by auto
  show ?thesis
    apply (rule that[of p ∪ r2])
    unfolding ∗
    defer
    apply (rule division-disjoint-union)
    unfolding divp(6)
    apply(rule assms r2)+
  proof −
    have interior s ∩ interior (⋃(r1−p)) = {}
    proof (rule inter-interior-unions-intervals)
      show finite (r1 − p) and open (interior s) and ∀ t∈r1−p. ∃ a b. t = cbox a
b
        using r1 by auto
      have ∗: ⋀s. (⋀x. x ∈ s ⟹ False) ⟹ s = {}
        by auto
      show ∀ t∈r1−p. interior s ∩ interior t = {}
      proof
        fix m x
        assume as: m ∈ r1 − p
        have interior m ∩ interior (⋃p) = {}
        proof (rule inter-interior-unions-intervals)
          show finite p and open (interior m) and ∀ t∈p. ∃ a b. t = cbox a b
            using divp by auto
          show ∀ t∈p. interior m ∩ interior t = {}
            by (metis DiffD1 DiffD2 as r1(1) r1(7) set-rev-mp)
        qed
        then show interior s ∩ interior m = {}
          unfolding divp by auto
      qed
    qed
    then show interior s ∩ interior (⋃(r1−p) ∩ (⋃q)) = {}
      using interior-subset by auto
  qed auto
qed
```

35.7 Tagged (partial) divisions.

definition *tagged-partial-division-of* (**infixr** *tagged'-partial'-division'-of 40*)
where *s tagged-partial-division-of i* $\longleftrightarrow$
finite s $\wedge$
$(\forall x\ k.\ (x,\ k) \in s \longrightarrow x \in k \wedge k \subseteq i \wedge (\exists a\ b.\ k = cbox\ a\ b)) \wedge$
$(\forall x1\ k1\ x2\ k2.\ (x1,\ k1) \in s \wedge (x2,\ k2) \in s \wedge (x1,\ k1) \neq (x2,\ k2) \longrightarrow$
interior k1 $\cap$ *interior k2* $= \{\}$)

lemma *tagged-partial-division-ofD*[*dest*]:
assumes *s tagged-partial-division-of i*
shows *finite s*
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow x \in k$
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq i$
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow \exists a\ b.\ k = cbox\ a\ b$
and $\bigwedge x1\ k1\ x2\ k2.\ (x1,k1) \in s \Longrightarrow$
$(x2,\ k2) \in s \Longrightarrow (x1,\ k1) \neq (x2,\ k2) \Longrightarrow$ *interior k1* $\cap$ *interior k2* $= \{\}$
using *assms* **unfolding** *tagged-partial-division-of-def* **by** *blast+*

definition *tagged-division-of* (**infixr** *tagged'-division'-of 40*)
where *s tagged-division-of i* $\longleftrightarrow$ *s tagged-partial-division-of i* $\wedge$ $(\bigcup\{k.\ \exists x.\ (x,k) \in s\} = i)$

lemma *tagged-division-of-finite*: *s tagged-division-of i* $\Longrightarrow$ *finite s*
unfolding *tagged-division-of-def tagged-partial-division-of-def* **by** *auto*

lemma *tagged-division-of*:
s tagged-division-of i $\longleftrightarrow$
finite s $\wedge$
$(\forall x\ k.\ (x,\ k) \in s \longrightarrow x \in k \wedge k \subseteq i \wedge (\exists a\ b.\ k = cbox\ a\ b)) \wedge$
$(\forall x1\ k1\ x2\ k2.\ (x1,\ k1) \in s \wedge (x2,\ k2) \in s \wedge (x1,\ k1) \neq (x2,\ k2) \longrightarrow$
interior k1 $\cap$ *interior k2* $= \{\}) \wedge$
$(\bigcup\{k.\ \exists x.\ (x,k) \in s\} = i)$
unfolding *tagged-division-of-def tagged-partial-division-of-def* **by** *auto*

lemma *tagged-division-ofI*:
assumes *finite s*
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow x \in k$
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq i$
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow \exists a\ b.\ k = cbox\ a\ b$
and $\bigwedge x1\ k1\ x2\ k2.\ (x1,k1) \in s \Longrightarrow (x2,\ k2) \in s \Longrightarrow (x1,\ k1) \neq (x2,\ k2) \Longrightarrow$
interior k1 $\cap$ *interior k2* $= \{\}$
and $(\bigcup\{k.\ \exists x.\ (x,k) \in s\} = i)$
shows *s tagged-division-of i*
unfolding *tagged-division-of*
using *assms*
apply *auto*
apply *fastforce+*
done

lemma *tagged-division-ofD*[*dest*]:
assumes *s tagged-division-of i*
shows *finite s*
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow x \in k$
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq i$
and $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow \exists\, a\ b.\ k = cbox\ a\ b$
and $\bigwedge x1\ k1\ x2\ k2.\ (x1,\ k1) \in s \Longrightarrow (x2,\ k2) \in s \Longrightarrow (x1,\ k1) \neq (x2,\ k2) \Longrightarrow$
interior k1 $\cap$ *interior k2* $= \{\}$
and $(\bigcup\{k.\ \exists\, x.\ (x,k) \in s\} = i)$
using *assms* **unfolding** *tagged-division-of* **by** *blast+*

lemma *division-of-tagged-division*:
assumes *s tagged-division-of i*
shows (*snd* ' *s*) *division-of i*
proof (*rule division-ofI*)
note *assm* = *tagged-division-ofD*[*OF assms*]
show $\bigcup$(*snd* ' *s*) = *i finite* (*snd* ' *s*)
using *assm* **by** *auto*
fix *k*
assume *k*: *k* $\in$ *snd* ' *s*
then obtain *xk* **where** *xk*: (*xk*, *k*) $\in$ *s*
by *auto*
then show *k* $\subseteq$ *i k* $\neq$ {} $\exists$ *a b*. *k* = *cbox a b*
using *assm* **by** *fastforce+*
fix *k'*
assume *k'*: *k'* $\in$ *snd* ' *s k* $\neq$ *k'*
from *this*(*1*) **obtain** *xk'* **where** *xk'*: (*xk'*, *k'*) $\in$ *s*
by *auto*
then show *interior k* $\cap$ *interior k'* = {}
using *assm*(*5*) *k'*(*2*) *xk* **by** *blast*
qed

lemma *partial-division-of-tagged-division*:
assumes *s tagged-partial-division-of i*
shows (*snd* ' *s*) *division-of* $\bigcup$(*snd* ' *s*)
proof (*rule division-ofI*)
note *assm* = *tagged-partial-division-ofD*[*OF assms*]
show *finite* (*snd* ' *s*) $\bigcup$(*snd* ' *s*) = $\bigcup$(*snd* ' *s*)
using *assm* **by** *auto*
fix *k*
assume *k*: *k* $\in$ *snd* ' *s*
then obtain *xk* **where** *xk*: (*xk*, *k*) $\in$ *s*
by *auto*
then show *k* $\neq$ {} $\exists$ *a b*. *k* = *cbox a b k* $\subseteq$ $\bigcup$(*snd* ' *s*)
using *assm* **by** *auto*
fix *k'*
assume *k'*: *k'* $\in$ *snd* ' *s k* $\neq$ *k'*
from *this*(*1*) **obtain** *xk'* **where** *xk'*: (*xk'*, *k'*) $\in$ *s*
by *auto*

```
  then show interior k ∩ interior k' = {}
    using assm(5) k'(2) xk by auto
qed

lemma tagged-partial-division-subset:
  assumes s tagged-partial-division-of i
    and t ⊆ s
  shows t tagged-partial-division-of i
  using assms
  unfolding tagged-partial-division-of-def
  using finite-subset[OF assms(2)]
  by blast

lemma setsum-over-tagged-division-lemma:
  assumes p tagged-division-of i
    and ⋀u v. cbox u v ≠ {} ⟹ content (cbox u v) = 0 ⟹ d (cbox u v) = 0
  shows setsum (λ(x,k). d k) p = setsum d (snd ` p)
proof −
  have ∗: (λ(x,k). d k) = d ∘ snd
    unfolding o-def by (rule ext) auto
  note assm = tagged-division-ofD[OF assms(1)]
  show ?thesis
    unfolding ∗
  proof (rule setsum.reindex-nontrivial[symmetric])
    show finite p
      using assm by auto
    fix x y
    assume x∈p y∈p x≠y snd x = snd y
    obtain a b where ab: snd x = cbox a b
      using assm(4)[of fst x snd x] ‹x∈p› by auto
    have (fst x, snd y) ∈ p (fst x, snd y) ≠ y
      by (metis prod.collapse ‹x∈p› ‹snd x = snd y› ‹x ≠ y›)+
    with ‹x∈p› ‹y∈p› have interior (snd x) ∩ interior (snd y) = {}
      by (intro assm(5)[of fst x - fst y]) auto
    then have content (cbox a b) = 0
      unfolding ‹snd x = snd y›[symmetric] ab content-eq-0-interior by auto
    then have d (cbox a b) = 0
      using assm(2)[of fst x snd x] ‹x∈p› ab[symmetric] by (intro assms(2)) auto
    then show d (snd x) = 0
      unfolding ab by auto
  qed
qed

lemma tag-in-interval: p tagged-division-of i ⟹ (x, k) ∈ p ⟹ x ∈ i
  by auto

lemma tagged-division-of-empty: {} tagged-division-of {}
  unfolding tagged-division-of by auto
```

lemma *tagged-partial-division-of-trivial*[*simp*]: *p tagged-partial-division-of* {} ⟷ *p* = {}
unfolding *tagged-partial-division-of-def* **by** *auto*

lemma *tagged-division-of-trivial*[*simp*]: *p tagged-division-of* {} ⟷ *p* = {}
unfolding *tagged-division-of* **by** *auto*

lemma *tagged-division-of-self*: *x* ∈ *cbox a b* ⟹ {(*x*,*cbox a b*)} *tagged-division-of* (*cbox a b*)
by (*rule tagged-division-ofI*) *auto*

lemma *tagged-division-of-self-real*: *x* ∈ {*a* .. *b*::*real*} ⟹ {(*x*,{*a* .. *b*})} *tagged-division-of* {*a* .. *b*}
unfolding *box-real*[*symmetric*]
by (*rule tagged-division-of-self*)

lemma *tagged-division-union*:
assumes *p1 tagged-division-of s1*
and *p2 tagged-division-of s2*
and *interior s1* ∩ *interior s2* = {}
shows (*p1* ∪ *p2*) *tagged-division-of* (*s1* ∪ *s2*)
proof (*rule tagged-division-ofI*)
note *p1* = *tagged-division-ofD*[*OF assms*(*1*)]
note *p2* = *tagged-division-ofD*[*OF assms*(*2*)]
show *finite* (*p1* ∪ *p2*)
using *p1*(*1*) *p2*(*1*) **by** *auto*
show ⋃{*k*. ∃ *x*. (*x*, *k*) ∈ *p1* ∪ *p2*} = *s1* ∪ *s2*
using *p1*(*6*) *p2*(*6*) **by** *blast*
fix *x k*
assume *xk*: (*x*, *k*) ∈ *p1* ∪ *p2*
show *x* ∈ *k* ∃ *a b*. *k* = *cbox a b*
using *xk p1*(*2*,*4*) *p2*(*2*,*4*) **by** *auto*
show *k* ⊆ *s1* ∪ *s2*
using *xk p1*(*3*) *p2*(*3*) **by** *blast*
fix *x*′ *k*′
assume *xk*′: (*x*′, *k*′) ∈ *p1* ∪ *p2* (*x*, *k*) ≠ (*x*′, *k*′)
have ∗: ⋀*a b*. *a* ⊆ *s1* ⟹ *b* ⊆ *s2* ⟹ *interior a* ∩ *interior b* = {}
using *assms*(*3*) *interior-mono* **by** *blast*
show *interior k* ∩ *interior k*′ = {}
apply (*cases* (*x*, *k*) ∈ *p1*)
apply (*meson* ∗ *UnE assms*(*1*) *assms*(*2*) *p1*(*5*) *tagged-division-ofD*(*3*) *xk*′(*1*) *xk*′(*2*))
by (*metis* ∗ *UnE assms*(*1*) *assms*(*2*) *inf-sup-aci*(*1*) *p2*(*5*) *tagged-division-ofD*(*3*) *xk xk*′(*1*) *xk*′(*2*))
qed

lemma *tagged-division-unions*:
assumes *finite iset*
and ∀ *i*∈*iset*. *pfn i tagged-division-of i*

and $\forall i1{\in}iset.\ \forall i2{\in}iset.\ i1 \neq i2 \longrightarrow interior(i1) \cap interior(i2) = \{\}$
shows $\bigcup(pfn\ `\ iset)$ *tagged-division-of* $(\bigcup iset)$
proof (*rule tagged-division-ofI*)
note *assm* = *tagged-division-ofD*[*OF assms*(2)[*rule-format*]]
show *finite* $(\bigcup(pfn\ `\ iset))$
apply (*rule finite-Union*)
using *assms*
apply *auto*
done
have $\bigcup\{k.\ \exists x.\ (x,\ k) \in \bigcup(pfn\ `\ iset)\} = \bigcup((\lambda i.\ \bigcup\{k.\ \exists x.\ (x,\ k) \in pfn\ i\})\ `\ iset)$
by *blast*
also have $\ldots = \bigcup iset$
using *assm*(6) **by** *auto*
finally show $\bigcup\{k.\ \exists x.\ (x,\ k) \in \bigcup(pfn\ `\ iset)\} = \bigcup iset$ **.**
fix *x k*
assume *xk*: $(x,\ k) \in \bigcup(pfn\ `\ iset)$
then obtain *i* **where** *i*: $i \in iset\ (x,\ k) \in pfn\ i$
by *auto*
show $x \in k\ \exists a\ b.\ k = cbox\ a\ b\ k \subseteq \bigcup iset$
using *assm*(2−4)[*OF i*] **using** *i*(1) **by** *auto*
fix $x'\ k'$
assume *xk'*: $(x',\ k') \in \bigcup(pfn\ `\ iset)\ (x,\ k) \neq (x',\ k')$
then obtain i' **where** i': $i' \in iset\ (x',\ k') \in pfn\ i'$
by *auto*
have $*$: $\bigwedge a\ b.\ i \neq i' \Longrightarrow a \subseteq i \Longrightarrow b \subseteq i' \Longrightarrow interior\ a \cap interior\ b = \{\}$
using *i*(1) *i'*(1)
using *assms*(3)[*rule-format*] *interior-mono*
by *blast*
show $interior\ k \cap interior\ k' = \{\}$
apply (*cases* $i = i'$)
using *assm*(5) i' *i*(2) *xk'*(2) **apply** *blast*
using $*$ *assm*(3) i' *i* **by** *auto*
qed

lemma *tagged-partial-division-of-union-self*:
assumes *p tagged-partial-division-of s*
shows *p tagged-division-of* $(\bigcup(snd\ `\ p))$
apply (*rule tagged-division-ofI*)
using *tagged-partial-division-ofD*[*OF assms*]
apply *auto*
done

lemma *tagged-division-of-union-self*:
assumes *p tagged-division-of s*
shows *p tagged-division-of* $(\bigcup(snd\ `\ p))$
apply (*rule tagged-division-ofI*)
using *tagged-division-ofD*[*OF assms*]
apply *auto*

done

35.8 Fine-ness of a partition w.r.t. a gauge.

definition *fine* (**infixr** *fine 46*)
where d *fine* $s \longleftrightarrow (\forall (x,k) \in s.\ k \subseteq d\ x)$

lemma *fineI*:
assumes $\bigwedge x\ k.\ (x,\ k) \in s \Longrightarrow k \subseteq d\ x$
shows d *fine* s
using *assms* **unfolding** *fine-def* **by** *auto*

lemma *fineD*[*dest*]:
assumes d *fine* s
shows $\bigwedge x\ k.\ (x,k) \in s \Longrightarrow k \subseteq d\ x$
using *assms* **unfolding** *fine-def* **by** *auto*

lemma *fine-inter*: $(\lambda x.\ d1\ x \cap d2\ x)$ *fine* $p \longleftrightarrow$ *d1 fine* $p \wedge$ *d2 fine* p
unfolding *fine-def* **by** *auto*

lemma *fine-inters*:
$(\lambda x.\ \bigcap \{f\ d\ x \mid d.\ \ d \in s\})$ *fine* $p \longleftrightarrow (\forall\, d{\in}s.\ (f\ d)$ *fine* $p)$
unfolding *fine-def* **by** *blast*

lemma *fine-union*: d *fine* $p1 \Longrightarrow d$ *fine* $p2 \Longrightarrow d$ *fine* $(p1 \cup p2)$
unfolding *fine-def* **by** *blast*

lemma *fine-unions*: $(\bigwedge p.\ p \in ps \Longrightarrow d$ *fine* $p) \Longrightarrow d$ *fine* $(\bigcup ps)$
unfolding *fine-def* **by** *auto*

lemma *fine-subset*: $p \subseteq q \Longrightarrow d$ *fine* $q \Longrightarrow d$ *fine* p
unfolding *fine-def* **by** *blast*

35.9 Gauge integral. Define on compact intervals first, then use a limit.

definition *has-integral-compact-interval* (**infixr** *has'-integral'-compact'-interval 46*)
where $(f$ *has-integral-compact-interval* $y)\ i \longleftrightarrow$
$(\forall\, e{>}0.\ \exists\, d.$ *gauge* $d \wedge$
$(\forall\, p.\ p$ *tagged-division-of* $i \wedge d$ *fine* $p \longrightarrow$
norm $(setsum\ (\lambda(x,k).\ content\ k *_R f\ x)\ p - y) < e))$

definition *has-integral* ::
$('n{::}euclidean\text{-}space \Rightarrow 'b{::}real\text{-}normed\text{-}vector) \Rightarrow 'b \Rightarrow 'n\ set \Rightarrow bool$
(**infixr** *has'-integral 46*)
where $(f$ *has-integral* $y)\ i \longleftrightarrow$
(*if* $\exists\, a\ b.\ i = cbox\ a\ b$
then $(f$ *has-integral-compact-interval* $y)\ i$
else $(\forall\, e{>}0.\ \exists\, B{>}0.\ \forall\, a\ b.\ ball\ 0\ B \subseteq cbox\ a\ b \longrightarrow$

$(\exists\, z.\ ((\lambda x.\ \textit{if}\ x \in i\ \textit{then}\ f\ x\ \textit{else}\ 0)\ \textit{has-integral-compact-interval}\ z)\ (\textit{cbox}\ a\ b)$
$\wedge$
$\textit{norm}\ (z - y) < e)))$

lemma *has-integral*:
$(f\ \textit{has-integral}\ y)\ (\textit{cbox}\ a\ b) \longleftrightarrow$
$(\forall\, e{>}0.\ \exists\, d.\ \textit{gauge}\ d \wedge$
$(\forall\, p.\ p\ \textit{tagged-division-of}\ (\textit{cbox}\ a\ b) \wedge d\ \textit{fine}\ p \longrightarrow$
$\textit{norm}\ (\textit{setsum}\ (\lambda(x,k).\ \textit{content}(k) *_R f\ x)\ p - y) < e))$
unfolding *has-integral-def has-integral-compact-interval-def*
by *auto*

lemma *has-integral-real*:
$(f\ \textit{has-integral}\ y)\ \{a\ ..\ b{::}\textit{real}\} \longleftrightarrow$
$(\forall\, e{>}0.\ \exists\, d.\ \textit{gauge}\ d \wedge$
$(\forall\, p.\ p\ \textit{tagged-division-of}\ \{a\ ..\ b\} \wedge d\ \textit{fine}\ p \longrightarrow$
$\textit{norm}\ (\textit{setsum}\ (\lambda(x,k).\ \textit{content}(k) *_R f\ x)\ p - y) < e))$
unfolding *box-real*[*symmetric*]
by (*rule has-integral*)

lemma *has-integralD*[*dest*]:
assumes $(f\ \textit{has-integral}\ y)\ (\textit{cbox}\ a\ b)$
and $e > 0$
obtains d **where** *gauge d*
and $\bigwedge p.\ p\ \textit{tagged-division-of}\ (\textit{cbox}\ a\ b) \Longrightarrow d\ \textit{fine}\ p \Longrightarrow$
$\textit{norm}\ (\textit{setsum}\ (\lambda(x,k).\ \textit{content}(k) *_R f(x))\ p - y) < e$
using *assms* **unfolding** *has-integral* **by** *auto*

lemma *has-integral-alt*:
$(f\ \textit{has-integral}\ y)\ i \longleftrightarrow$
$(\textit{if}\ \exists\, a\ b.\ i = \textit{cbox}\ a\ b$
$\textit{then}\ (f\ \textit{has-integral}\ y)\ i$
$\textit{else}\ (\forall\, e{>}0.\ \exists\, B{>}0.\ \forall\, a\ b.\ \textit{ball}\ 0\ B \subseteq \textit{cbox}\ a\ b \longrightarrow$
$(\exists\, z.\ ((\lambda x.\ \textit{if}\ x \in i\ \textit{then}\ f(x)\ \textit{else}\ 0)\ \textit{has-integral}\ z)\ (\textit{cbox}\ a\ b) \wedge \textit{norm}\ (z - y) < e)))$
unfolding *has-integral*
unfolding *has-integral-compact-interval-def has-integral-def*
by *auto*

lemma *has-integral-altD*:
assumes $(f\ \textit{has-integral}\ y)\ i$
and $\neg\ (\exists\, a\ b.\ i = \textit{cbox}\ a\ b)$
and $e{>}0$
obtains B **where** $B > 0$
and $\forall\, a\ b.\ \textit{ball}\ 0\ B \subseteq \textit{cbox}\ a\ b \longrightarrow$
$(\exists\, z.\ ((\lambda x.\ \textit{if}\ x \in i\ \textit{then}\ f(x)\ \textit{else}\ 0)\ \textit{has-integral}\ z)\ (\textit{cbox}\ a\ b) \wedge \textit{norm}(z - y) < e)$
using *assms*
unfolding *has-integral*

unfolding *has-integral-compact-interval-def has-integral-def*
by *auto*

definition *integrable-on* (**infixr** *integrable'-on 46*)
where *f integrable-on i* $\longleftrightarrow$ ($\exists$ *y*. (*f has-integral y*) *i*)

definition *integral i f* = (*SOME y*. (*f has-integral y*) *i*)

lemma *integrable-integral*[*dest*]: *f integrable-on i* $\Longrightarrow$ (*f has-integral* (*integral i f*)) *i*
unfolding *integrable-on-def integral-def* **by** (*rule someI-ex*)

lemma *has-integral-integrable*[*intro*]: (*f has-integral i*) *s* $\Longrightarrow$ *f integrable-on s*
unfolding *integrable-on-def* **by** *auto*

lemma *has-integral-integral*: *f integrable-on s* $\longleftrightarrow$ (*f has-integral* (*integral s f*)) *s*
by *auto*

lemma *setsum-content-null*:
assumes *content* (*cbox a b*) = *0*
and *p tagged-division-of* (*cbox a b*)
shows *setsum* (λ(*x*,*k*). *content k* $*_R$ *f x*) *p* = (*0*::*'a*::*real-normed-vector*)
proof (*rule setsum.neutral*, *rule*)
fix *y*
assume *y*: *y* $\in$ *p*
obtain *x k* **where** *xk*: *y* = (*x*, *k*)
using *surj-pair*[*of y*] **by** *blast*
note *assm* = *tagged-division-ofD*(*3*−*4*)[*OF assms*(*2*) *y*[*unfolded xk*]]
from *this*(*2*) **obtain** *c d* **where** *k*: *k* = *cbox c d* **by** *blast*
have (λ(*x*, *k*). *content k* $*_R$ *f x*) *y* = *content k* $*_R$ *f x*
unfolding *xk* **by** *auto*
also have ... = *0*
using *content-subset*[*OF assm*(*1*)[*unfolded k*]] *content-pos-le*[*of c d*]
unfolding *assms*(*1*) *k*
by *auto*
finally show (λ(*x*, *k*). *content k* $*_R$ *f x*) *y* = *0* .
qed

35.10 Some basic combining lemmas.

lemma *tagged-division-unions-exists*:
assumes *finite iset*
and $\forall$ *i*$\in$*iset*. $\exists$ *p*. *p tagged-division-of i* $\wedge$ *d fine p*
and $\forall$ *i1*$\in$*iset*. $\forall$ *i2*$\in$*iset*. *i1* $\neq$ *i2* $\longrightarrow$ *interior i1* $\cap$ *interior i2* = {}
and $\bigcup$ *iset* = *i*
obtains *p* **where** *p tagged-division-of i* **and** *d fine p*
proof −
obtain *pfn* **where** *pfn*:
$\bigwedge$*x*. *x* $\in$ *iset* $\Longrightarrow$ *pfn x tagged-division-of x*

$\bigwedge x.\ x \in iset \Longrightarrow d\ fine\ pfn\ x$
using *bchoice*[*OF assms*(*2*)] **by** *auto*
show *thesis*
apply (*rule-tac p*=$\bigcup$(*pfn* ' *iset*) **in** *that*)
using *assms*(*1*) *assms*(*3*) *assms*(*4*) *pfn*(*1*) *tagged-division-unions* **apply** *force*
by (*metis* (*mono-tags*, *lifting*) *fine-unions imageE pfn*(*2*))
qed

35.11 The set we're concerned with must be closed.

lemma *division-of-closed*:
fixes $i :: 'n::euclidean\text{-}space\ set$
shows $s\ division\text{-}of\ i \Longrightarrow closed\ i$
unfolding *division-of-def* **by** *fastforce*

35.12 General bisection principle for intervals; might be useful elsewhere.

lemma *interval-bisection-step*:
fixes *type* :: $'a::euclidean\text{-}space$
assumes $P\ \{\}$
and $\forall s\ t.\ P\ s \wedge P\ t \wedge interior(s) \cap interior(t) = \{\} \longrightarrow P\ (s \cup t)$
and $\neg\ P\ (cbox\ a\ (b::'a))$
obtains *c d* **where** $\neg\ P\ (cbox\ c\ d)$
and $\forall i{\in}Basis.\ a{\cdot}i \le c{\cdot}i \wedge c{\cdot}i \le d{\cdot}i \wedge d{\cdot}i \le b{\cdot}i \wedge 2 * (d{\cdot}i - c{\cdot}i) \le b{\cdot}i - a{\cdot}i$
proof −
have $cbox\ a\ b \neq \{\}$
using *assms*(*1*,*3*) **by** *metis*
then have *ab*: $\bigwedge i.\ i{\in}Basis \Longrightarrow a \cdot i \le b \cdot i$
by (*force simp*: *mem-box*)
{ fix *f*
have $[\![finite\ f;$
$\bigwedge s.\ s{\in}f \Longrightarrow P\ s;$
$\bigwedge s.\ s{\in}f \Longrightarrow \exists a\ b.\ s = cbox\ a\ b;$
$\bigwedge s\ t.\ s{\in}f \Longrightarrow t{\in}f \Longrightarrow s \neq t \Longrightarrow interior\ s \cap interior\ t = \{\}]\!] \Longrightarrow P\ (\bigcup f)$
proof (*induct f rule*: *finite-induct*)
case *empty*
show *?case*
using *assms*(*1*) **by** *auto*
next
case (*insert x f*)
show *?case*
unfolding *Union-insert*
apply (*rule assms*(*2*)[*rule-format*])
using *inter-interior-unions-intervals* [*of f interior x*]
apply (*auto simp*: *insert*)
by (*metis IntI empty-iff insert.hyps*(*2*) *insert.prems*(*3*) *insert-iff*)

```
  qed
 } note UN-cases = this
 let ?A = {cbox c d | c d::'a. ∀ i∈Basis. (c•i = a•i) ∧ (d•i = (a•i + b•i) / 2) ∨
   (c•i = (a•i + b•i) / 2) ∧ (d•i = b•i)}
 let ?PP = λc d. ∀ i∈Basis. a•i ≤ c•i ∧ c•i ≤ d•i ∧ d•i ≤ b•i ∧ 2 * (d•i − c•i)
≤ b•i − a•i
 {
  presume ∀ c d. ?PP c d ⟶ P (cbox c d) ⟹ False
  then show thesis
   unfolding atomize-not not-all
   by (blast intro: that)
 }
 assume as: ∀ c d. ?PP c d ⟶ P (cbox c d)
 have P (⋃ ?A)
 proof (rule UN-cases)
  let ?B = (λs. cbox (∑ i∈Basis. (if i ∈ s then a•i else (a•i + b•i) / 2) *R i::'a)
    (∑ i∈Basis. (if i ∈ s then (a•i + b•i) / 2 else b•i) *R i)) ` {s. s ⊆ Basis}
  have ?A ⊆ ?B
  proof
   fix x
   assume x ∈ ?A
   then obtain c d
    where x: x = cbox c d
          ⋀i. i ∈ Basis ⟹
              c • i = a • i ∧ d • i = (a • i + b • i) / 2 ∨
              c • i = (a • i + b • i) / 2 ∧ d • i = b • i by blast
   show x ∈ ?B
    unfolding image-iff x
    apply (rule-tac x={i. i∈Basis ∧ c•i = a•i} in bexI)
    apply (rule arg-cong2 [where f = cbox])
    using x(2) ab
    apply (auto simp add: euclidean-eq-iff[where 'a='a])
    by fastforce
  qed
  then show finite ?A
   by (rule finite-subset) auto
 next
  fix s
  assume s ∈ ?A
  then obtain c d
   where s: s = cbox c d
         ⋀i. i ∈ Basis ⟹
             c • i = a • i ∧ d • i = (a • i + b • i) / 2 ∨
             c • i = (a • i + b • i) / 2 ∧ d • i = b • i
   by blast
  show P s
   unfolding s
   apply (rule as[rule-format])
   using ab s(2) by force
```

show $\exists\, a\ b.\ s = cbox\ a\ b$
unfolding s **by** *auto*
fix t
assume $t \in ?A$
then obtain $e\ f$ **where** t:
$t = cbox\ e\ f$
$\bigwedge i.\ i \in Basis \Longrightarrow$
$e \cdot i = a \cdot i \wedge f \cdot i = (a \cdot i + b \cdot i)\ /\ 2 \vee$
$e \cdot i = (a \cdot i + b \cdot i)\ /\ 2 \wedge f \cdot i = b \cdot i$
by *blast*
assume $s \neq t$
then have $\neg\ (c = e \wedge d = f)$
unfolding $s\ t$ **by** *auto*
then obtain i **where** $c{\cdot}i \neq e{\cdot}i \vee d{\cdot}i \neq f{\cdot}i$ **and** i': $i \in Basis$
unfolding *euclidean-eq-iff*[**where** $'a{=}'a$] **by** *auto*
then have i: $c{\cdot}i \neq e{\cdot}i\ d{\cdot}i \neq f{\cdot}i$
using $s(2)\ t(2)$ **apply** *fastforce*
using $t(2)[OF\ i']\ \langle c \cdot i \neq e \cdot i \vee d \cdot i \neq f \cdot i \rangle\ i'\ s(2)\ t(2)$ **by** *fastforce*
have $*$: $\bigwedge s\ t.\ (\bigwedge a.\ a \in s \Longrightarrow a \in t \Longrightarrow False) \Longrightarrow s \cap t = \{\}$
by *auto*
show $interior\ s \cap interior\ t = \{\}$
unfolding $s\ t$ *interior-cbox*
proof (*rule* $*$)
fix x
assume $x \in box\ c\ d\ x \in box\ e\ f$
then have x: $c{\cdot}i < d{\cdot}i\ e{\cdot}i < f{\cdot}i\ c{\cdot}i < f{\cdot}i\ e{\cdot}i < d{\cdot}i$
unfolding *mem-box* **using** i'
by *force+*
show *False* **using** $s(2)[OF\ i']$
proof *safe*
assume *as*: $c \cdot i = a \cdot i\ d \cdot i = (a \cdot i + b \cdot i)\ /\ 2$
show *False*
using $t(2)[OF\ i']$ **and** $i\ x$ **unfolding** *as* **by** (*fastforce simp add*:*field-simps*)
next
assume *as*: $c \cdot i = (a \cdot i + b \cdot i)\ /\ 2\ d \cdot i = b \cdot i$
show *False*
using $t(2)[OF\ i']$ **and** $i\ x$ **unfolding** *as* **by**(*fastforce simp add*:*field-simps*)
qed
qed
qed
also have $\bigcup ?A = cbox\ a\ b$
proof (*rule set-eqI*,*rule*)
fix x
assume $x \in \bigcup ?A$
then obtain $c\ d$ **where** x:
$x \in cbox\ c\ d$
$\bigwedge i.\ i \in Basis \Longrightarrow$
$c \cdot i = a \cdot i \wedge d \cdot i = (a \cdot i + b \cdot i)\ /\ 2 \vee$
$c \cdot i = (a \cdot i + b \cdot i)\ /\ 2 \wedge d \cdot i = b \cdot i$

```
      by blast
    show x∈cbox a b
      unfolding mem-box
    proof safe
      fix i :: 'a
      assume i: i ∈ Basis
      then show a · i ≤ x · i x · i ≤ b · i
        using x(2)[OF i] x(1)[unfolded mem-box,THEN bspec, OF i] by auto
    qed
  next
    fix x
    assume x: x ∈ cbox a b
    have ∀ i∈Basis.
      ∃ c d. (c = a•i ∧ d = (a•i + b•i) / 2 ∨ c = (a•i + b•i) / 2 ∧ d = b•i) ∧
c≤x•i ∧ x•i ≤ d
      (is ∀ i∈Basis. ∃ c d. ?P i c d)
      unfolding mem-box
    proof
      fix i :: 'a
      assume i: i ∈ Basis
      have ?P i (a•i) ((a · i + b · i) / 2) ∨ ?P i ((a · i + b · i) / 2) (b•i)
        using x[unfolded mem-box,THEN bspec, OF i] by auto
      then show ∃ c d. ?P i c d
        by blast
    qed
    then show x∈⋃ ?A
      unfolding Union-iff Bex-def mem-Collect-eq choice-Basis-iff
      apply auto
      apply (rule-tac x=cbox xa xaa in exI)
      unfolding mem-box
      apply auto
      done
  qed
  finally show False
    using assms by auto
qed

lemma interval-bisection:
  fixes type :: 'a::euclidean-space
  assumes P {}
    and (∀ s t. P s ∧ P t ∧ interior(s) ∩ interior(t) = {} ⟶ P(s ∪ t))
    and ¬ P (cbox a (b::'a))
  obtains x where x ∈ cbox a b
    and ∀ e>0. ∃ c d. x ∈ cbox c d ∧ cbox c d ⊆ ball x e ∧ cbox c d ⊆ cbox a b ∧
¬ P (cbox c d)
proof −
  have ∀ x. ∃ y. ¬ P (cbox (fst x) (snd x)) ⟶ (¬ P (cbox (fst y) (snd y)) ∧
    (∀ i∈Basis. fst x•i ≤ fst y•i ∧ fst y•i ≤ snd y•i ∧ snd y•i ≤ snd x•i ∧
      2 ∗ (snd y•i − fst y•i) ≤ snd x•i − fst x•i)) (is ∀ x. ?P x)
```

proof
show *?P x* **for** *x*
proof (*cases P* (*cbox* (*fst x*) (*snd x*)))
case *True*
then show *?thesis* **by** *auto*
next
case *as*: *False*
obtain *c d* **where** ¬ *P* (*cbox c d*)
∀ *i*∈*Basis*.
fst x · *i* ≤ *c* · *i* ∧
c · *i* ≤ *d* · *i* ∧
d · *i* ≤ *snd x* · *i* ∧
2 ∗ (*d* · *i* − *c* · *i*) ≤ *snd x* · *i* − *fst x* · *i*
by (*rule interval-bisection-step*[*of P*, *OF assms*(*1*−*2*) *as*])
then show *?thesis*
apply −
apply (*rule-tac x*=(*c*,*d*) **in** *exI*)
apply *auto*
done
qed
qed
then obtain *f* **where** *f*:
∀ *x*.
¬ *P* (*cbox* (*fst x*) (*snd x*)) ⟶
¬ *P* (*cbox* (*fst* (*f x*)) (*snd* (*f x*))) ∧
(∀ *i*∈*Basis*.
fst x · *i* ≤ *fst* (*f x*) · *i* ∧
fst (*f x*) · *i* ≤ *snd* (*f x*) · *i* ∧
snd (*f x*) · *i* ≤ *snd x* · *i* ∧
2 ∗ (*snd* (*f x*) · *i* − *fst* (*f x*) · *i*) ≤ *snd x* · *i* − *fst x* · *i*)
apply −
apply (*drule choice*)
apply *blast*
done
def *AB* ≡ λ*n*. (*f* ^^ *n*) (*a*,*b*)
def *A* ≡ λ*n*. *fst*(*AB n*)
def *B* ≡ λ*n*. *snd*(*AB n*)
note *ab-def* = *A-def B-def AB-def*
have *A 0* = *a B 0* = *b* ⋀*n*. ¬ *P* (*cbox* (*A*(*Suc n*)) (*B*(*Suc n*))) ∧
(∀ *i*∈*Basis*. *A*(*n*)·*i* ≤ *A*(*Suc n*)·*i* ∧ *A*(*Suc n*)·*i* ≤ *B*(*Suc n*)·*i* ∧ *B*(*Suc n*)·*i* ≤ *B*(*n*)·*i* ∧
2 ∗ (*B*(*Suc n*)·*i* − *A*(*Suc n*)·*i*) ≤ *B*(*n*)·*i* − *A*(*n*)·*i*) (**is** ⋀*n*. *?P n*)
proof −
show *A 0* = *a B 0* = *b*
unfolding *ab-def* **by** *auto*
note *S* = *ab-def funpow.simps o-def id-apply*
show *?P n* **for** *n*
proof (*induct n*)
case *0*

```
    then show ?case
      unfolding S
      apply (rule f[rule-format]) using assms(3)
      apply auto
      done
  next
    case (Suc n)
    show ?case
      unfolding S
      apply (rule f[rule-format])
      using Suc
      unfolding S
      apply auto
      done
  qed
qed
note AB = this(1-2) conjunctD2[OF this(3),rule-format]

have interv: ∃ n. ∀ x∈cbox (A n) (B n). ∀ y∈cbox (A n) (B n). dist x y < e
  if e: 0 < e for e
proof -
  obtain n where n: (∑ i∈Basis. b • i − a • i) / e < 2 ^ n
    using real-arch-pow2[of (setsum (λi. b•i − a•i) Basis) / e] ..
  show ?thesis
  proof (rule exI [where x=n], clarify)
    fix x y
    assume xy: x∈cbox (A n) (B n) y∈cbox (A n) (B n)
    have dist x y ≤ setsum (λi. |(x − y)•i|) Basis
      unfolding dist-norm by(rule norm-le-l1)
    also have ... ≤ setsum (λi. B n•i − A n•i) Basis
    proof (rule setsum-mono)
      fix i :: 'a
      assume i: i ∈ Basis
      show |(x − y) • i| ≤ B n • i − A n • i
        using xy[unfolded mem-box,THEN bspec, OF i]
        by (auto simp: inner-diff-left)
    qed
    also have ... ≤ setsum (λi. b•i − a•i) Basis / 2^n
      unfolding setsum-divide-distrib
    proof (rule setsum-mono)
      show B n • i − A n • i ≤ (b • i − a • i) / 2 ^ n if i: i ∈ Basis for i
      proof (induct n)
        case 0
        then show ?case
          unfolding AB by auto
      next
        case (Suc n)
        have B (Suc n) • i − A (Suc n) • i ≤ (B n • i − A n • i) / 2
          using AB(4)[of i n] using i by auto
```

```
        also have ... ≤ (b · i − a · i) / 2 ^ Suc n
          using Suc by (auto simp add: field-simps)
        finally show ?case .
      qed
    qed
    also have ... < e
      using n using e by (auto simp add: field-simps)
    finally show dist x y < e .
  qed
 qed
 {
  fix n m :: nat
  assume m ≤ n then have cbox (A n) (B n) ⊆ cbox (A m) (B m)
  proof (induction rule: inc-induct)
    case (step i)
    show ?case
      using AB(4) by (intro order-trans[OF step.IH] subset-box-imp) auto
  qed simp
 } note ABsubset = this
 have ∃ a. ∀ n. a∈ cbox (A n) (B n)
  by (rule decreasing-closed-nest[rule-format,OF closed-cbox - ABsubset interv])
    (metis nat.exhaust AB(1−3) assms(1,3))
 then obtain x0 where x0: ⋀n. x0 ∈ cbox (A n) (B n)
  by blast
 show thesis
 proof (rule that[rule-format, of x0])
  show x0∈cbox a b
    using x0[of 0] unfolding AB .
  fix e :: real
  assume e > 0
  from interv[OF this] obtain n
    where n: ∀ x∈cbox (A n) (B n). ∀ y∈cbox (A n) (B n). dist x y < e ..
  have ¬ P (cbox (A n) (B n))
    apply (cases 0 < n)
    using AB(3)[of n − 1] assms(3) AB(1−2)
    apply auto
    done
  moreover have cbox (A n) (B n) ⊆ ball x0 e
    using n using x0[of n] by auto
  moreover have cbox (A n) (B n) ⊆ cbox a b
    unfolding AB(1−2)[symmetric] by (rule ABsubset) auto
  ultimately show ∃ c d. x0 ∈ cbox c d ∧ cbox c d ⊆ ball x0 e ∧ cbox c d ⊆
cbox a b ∧ ¬ P (cbox c d)
    apply (rule-tac x=A n in exI)
    apply (rule-tac x=B n in exI)
    apply (auto simp: x0)
    done
 qed
qed
```

35.13 Cousin's lemma.

lemma *fine-division-exists*:
fixes *a b* :: *'a*::*euclidean-space*
assumes *gauge g*
obtains *p* **where** *p tagged-division-of* (*cbox a b*) *g fine p*
proof −
presume ¬ (∃*p*. *p tagged-division-of* (*cbox a b*) ∧ *g fine p*) ⟹ *False*
then obtain *p* **where** *p tagged-division-of* (*cbox a b*) *g fine p*
by *blast*
then show *thesis* **..**
next
assume *as*: ¬ (∃*p*. *p tagged-division-of* (*cbox a b*) ∧ *g fine p*)
obtain *x* **where** *x*:
x ∈ (*cbox a b*)
⋀*e*. *0* < *e* ⟹
∃*c d*.
x ∈ *cbox c d* ∧
cbox c d ⊆ *ball x e* ∧
cbox c d ⊆ (*cbox a b*) ∧
¬ (∃*p*. *p tagged-division-of cbox c d* ∧ *g fine p*)
apply (*rule interval-bisection*[*of* λ*s*. ∃*p*. *p tagged-division-of s* ∧ *g fine p*, *OF*
- - *as*])
apply (*simp add*: *fine-def*)
apply (*metis tagged-division-union fine-union*)
apply (*auto simp*:)
done
obtain *e* **where** *e*: *e* > *0 ball x e* ⊆ *g x*
using *gaugeD*[*OF assms*, *of x*] **unfolding** *open-contains-ball* **by** *auto*
from *x*(*2*)[*OF e*(*1*)]
obtain *c d* **where** *c-d*: *x* ∈ *cbox c d*
cbox c d ⊆ *ball x e*
cbox c d ⊆ *cbox a b*
¬ (∃*p*. *p tagged-division-of cbox c d* ∧ *g fine p*)
by *blast*
have *g fine* {(*x*, *cbox c d*)}
unfolding *fine-def* **using** *e* **using** *c-d*(*2*) **by** *auto*
then show *False*
using *tagged-division-of-self*[*OF c-d*(*1*)] **using** *c-d* **by** *auto*
qed

lemma *fine-division-exists-real*:
fixes *a b* :: *real*
assumes *gauge g*
obtains *p* **where** *p tagged-division-of* {*a* .. *b*} *g fine p*
by (*metis assms box-real*(*2*) *fine-division-exists*)

35.14 Basic theorems about integrals.

lemma *has-integral-unique*:

fixes *f* :: *'n::euclidean-space ⇒ 'a::real-normed-vector*
assumes (*f has-integral k1*) *i*
and (*f has-integral k2*) *i*
shows *k1 = k2*
proof (*rule ccontr*)
let *?e = norm (k1 − k2) / 2*
assume *as*: *k1 ≠ k2*
then have *e*: *?e > 0*
by *auto*
have *lem*: *False*
if *f-k1*: (*f has-integral k1*) (*cbox a b*)
and *f-k2*: (*f has-integral k2*) (*cbox a b*)
and *k1 ≠ k2*
for *f* :: *'n ⇒ 'a* **and** *a b k1 k2*
proof −
let *?e = norm (k1 − k2) / 2*
from ‹*k1 ≠ k2*› **have** *e*: *?e > 0* **by** *auto*
obtain *d1* **where** *d1*:
gauge d1
⋀*p*. *p tagged-division-of cbox a b* ⟹
d1 fine p ⟹ *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) − *k1*) < *norm* (*k1* − *k2*) / *2*
by (*rule has-integralD*[*OF f-k1 e*]) *blast*
obtain *d2* **where** *d2*:
gauge d2
⋀*p*. *p tagged-division-of cbox a b* ⟹
d2 fine p ⟹ *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) − *k2*) < *norm* (*k1* − *k2*) / *2*
by (*rule has-integralD*[*OF f-k2 e*]) *blast*
obtain *p* **where** *p*:
p tagged-division-of cbox a b
(λ*x*. *d1 x* ∩ *d2 x*) *fine p*
by (*rule fine-division-exists*[*OF gauge-inter*[*OF d1*(*1*) *d2*(*1*)]])
let *?c* = (∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*)
have *norm* (*k1 − k2*) ≤ *norm* (*?c − k2*) + *norm* (*?c − k1*)
using *norm-triangle-ineq4*[*of k1 − ?c k2 − ?c*]
by (*auto simp add*:*algebra-simps norm-minus-commute*)
also have ... < *norm* (*k1 − k2*) / *2* + *norm* (*k1 − k2*) / *2*
apply (*rule add-strict-mono*)
apply (*rule-tac*[!] *d2*(*2*) *d1*(*2*))
using *p* **unfolding** *fine-def*
apply *auto*
done
finally show *False* **by** *auto*
qed
{
presume ¬ (∃ *a b*. *i* = *cbox a b*) ⟹ *False*
then show *False*
using *as assms lem* **by** *blast*

```
  }
  assume as: ¬ (∃ a b. i = cbox a b)
  obtain B1 where B1:
      0 < B1
      ⋀a b. ball 0 B1 ⊆ cbox a b ⟹
        ∃ z. ((λx. if x ∈ i then f x else 0) has-integral z) (cbox a b) ∧
          norm (z − k1) < norm (k1 − k2) / 2
    by (rule has-integral-altD[OF assms(1) as,OF e]) blast
  obtain B2 where B2:
      0 < B2
      ⋀a b. ball 0 B2 ⊆ cbox a b ⟹
        ∃ z. ((λx. if x ∈ i then f x else 0) has-integral z) (cbox a b) ∧
          norm (z − k2) < norm (k1 − k2) / 2
    by (rule has-integral-altD[OF assms(2) as,OF e]) blast
  have ∃ a b::'n. ball 0 B1 ∪ ball 0 B2 ⊆ cbox a b
    apply (rule bounded-subset-cbox)
    using bounded-Un bounded-ball
    apply auto
    done
  then obtain a b :: 'n where ab: ball 0 B1 ⊆ cbox a b ball 0 B2 ⊆ cbox a b
    by blast
  obtain w where w:
    ((λx. if x ∈ i then f x else 0) has-integral w) (cbox a b)
    norm (w − k1) < norm (k1 − k2) / 2
    using B1(2)[OF ab(1)] by blast
  obtain z where z:
    ((λx. if x ∈ i then f x else 0) has-integral z) (cbox a b)
    norm (z − k2) < norm (k1 − k2) / 2
    using B2(2)[OF ab(2)] by blast
  have z = w
    using lem[OF w(1) z(1)] by auto
  then have norm (k1 − k2) ≤ norm (z − k2) + norm (w − k1)
    using norm-triangle-ineq4 [of k1 − w k2 − z]
    by (auto simp add: norm-minus-commute)
  also have ... < norm (k1 − k2) / 2 + norm (k1 − k2) / 2
    apply (rule add-strict-mono)
    apply (rule-tac[!] z(2) w(2))
    done
  finally show False by auto
qed

lemma integral-unique [intro]: (f has-integral y) k ⟹ integral k f = y
  unfolding integral-def
  by (rule some-equality) (auto intro: has-integral-unique)

lemma has-integral-is-0:
  fixes f :: 'n::euclidean-space ⇒ 'a::real-normed-vector
  assumes ∀ x∈s. f x = 0
  shows (f has-integral 0) s
```

proof −
have *lem*: $\bigwedge a\ b.\ \bigwedge f::'n \Rightarrow 'a.$
$(\forall x \in cbox\ a\ b.\ f(x) = 0) \Longrightarrow (f\ has\text{-}integral\ 0)\ (cbox\ a\ b)$
unfolding *has-integral*
proof *clarify*
fix *a b e*
fix $f :: 'n \Rightarrow 'a$
assume *as*: $\forall x \in cbox\ a\ b.\ f\ x = 0\ 0 < (e::real)$
have *norm* $((\sum (x,\ k) \in p.\ content\ k *_R f\ x) - 0) < e$
if *p*: *p tagged-division-of cbox a b* **for** *p*
proof −
have $(\sum (x,\ k) \in p.\ content\ k *_R f\ x) = 0$
proof (*rule setsum.neutral*, *rule*)
fix *x*
assume *x*: $x \in p$
have *f* (*fst x*) = 0
using *tagged-division-ofD*(2−3)[*OF p*, *of fst x snd x*] **using** *as x* **by** *auto*
then show $(\lambda(x,\ k).\ content\ k *_R f\ x)\ x = 0$
apply (*subst surjective-pairing*[*of x*])
unfolding *split-conv*
apply *auto*
done
qed
then show *?thesis*
using *as* **by** *auto*
qed
then show $\exists d.\ gauge\ d \wedge$
$(\forall p.\ p\ tagged\text{-}division\text{-}of\ (cbox\ a\ b) \wedge d\ fine\ p \longrightarrow norm\ ((\sum (x,\ k) \in p.$
$content\ k *_R f\ x) - 0) < e)$
by *auto*
qed
{
presume $\neg\ (\exists a\ b.\ s = cbox\ a\ b) \Longrightarrow$ *?thesis*
with *assms lem* **show** *?thesis*
by *blast*
}
have $*$: $(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0) = (\lambda x.\ 0)$
apply (*rule ext*)
using *assms*
apply *auto*
done
assume $\neg\ (\exists a\ b.\ s = cbox\ a\ b)$
then show *?thesis*
using *lem*
by (*subst has-integral-alt*) (*force simp add*: $*$)
qed

lemma *has-integral-0*[*simp*]: $((\lambda x::'n::euclidean\text{-}space.\ 0)\ has\text{-}integral\ 0)\ s$
by (*rule has-integral-is-0*) *auto*

lemma *has-integral-0-eq*[*simp*]: $((\lambda x.\ 0)$ *has-integral* $i)\ s \longleftrightarrow i = 0$
using *has-integral-unique*[*OF has-integral-0*] **by** *auto*

lemma *has-integral-linear*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::real\text{-}normed\text{-}vector$
assumes $(f$ *has-integral* $y)\ s$
and *bounded-linear* h
shows $((h \circ f)$ *has-integral* $((h\ y)))\ s$
proof −
interpret *bounded-linear* h
using *assms*(2) **.**
from *pos-bounded* **obtain** B **where** B: $0 < B\ \bigwedge x.\ norm\ (h\ x) \leq norm\ x * B$
by *blast*
have *lem*: $\bigwedge(f :: 'n \Rightarrow 'a)\ y\ a\ b.$
$(f$ *has-integral* $y)$ *(cbox* $a\ b) \Longrightarrow ((h \circ f)$ *has-integral* $h\ y)$ *(cbox* $a\ b)$
unfolding *has-integral*
proof (*clarify*, *goal-cases*)
case *prems*: (*1* $f\ y\ a\ b\ e$)
from *pos-bounded*
obtain B **where** B: $0 < B\ \bigwedge x.\ norm\ (h\ x) \leq norm\ x * B$
by *blast*
have $e\ /\ B > 0$ **using** *prems*(2) B **by** *simp*
then obtain g
where g: *gauge* g
$\bigwedge p.\ p$ *tagged-division-of* *(cbox* $a\ b) \Longrightarrow g$ *fine* $p \Longrightarrow$
$norm\ ((\sum (x,\ k) \in p.\ content\ k *_R f\ x) - y) < e\ /\ B$
using *prems*(1) **by** *auto*
{
fix p
assume *as*: p *tagged-division-of* *(cbox* $a\ b)\ g$ *fine* p
have *hc*: $\bigwedge x\ k.\ h\ ((\lambda(x,\ k).\ content\ k *_R f\ x)\ x) = (\lambda(x,\ k).\ h\ (content\ k *_R f\ x))\ x$
by *auto*
then have $(\sum (x,\ k) \in p.\ content\ k *_R (h \circ f)\ x) = setsum\ (h \circ (\lambda(x,\ k).\ content\ k *_R f\ x))\ p$
unfolding *o-def* **unfolding** *scaleR*[*symmetric*] *hc* **by** *simp*
also have $\ldots = h\ (\sum (x,\ k) \in p.\ content\ k *_R f\ x)$
using *setsum*[*of* $\lambda(x,k).\ content\ k *_R f\ x\ p$] **using** *as* **by** *auto*
finally have $(\sum (x,\ k) \in p.\ content\ k *_R (h \circ f)\ x) = h\ (\sum (x,\ k) \in p.\ content\ k *_R f\ x)$ **.**
then have $norm\ ((\sum (x,\ k) \in p.\ content\ k *_R (h \circ f)\ x) - h\ y) < e$
apply (*simp add*: *diff*[*symmetric*])
apply (*rule le-less-trans*[*OF* $B(2)$])
using $g(2)$[*OF as*] $B(1)$
apply (*auto simp add*: *field-simps*)
done
}
with g **show** *?case*

```
    by (rule-tac x=g in exI) auto
  qed
  {
    presume ¬ (∃ a b. s = cbox a b) ⟹ ?thesis
    then show ?thesis
      using assms(1) lem by blast
  }
  assume as: ¬ (∃ a b. s = cbox a b)
  then show ?thesis
  proof (subst has-integral-alt, clarsimp)
    fix e :: real
    assume e: e > 0
    have ∗: 0 < e/B using e B(1) by simp
    obtain M where M:
      M > 0
      ⋀a b. ball 0 M ⊆ cbox a b ⟹
       ∃ z. ((λx. if x ∈ s then f x else 0) has-integral z) (cbox a b) ∧ norm (z − y)
< e / B
      using has-integral-altD[OF assms(1) as ∗] by blast
    show ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
     (∃ z. ((λx. if x ∈ s then (h ∘ f) x else 0) has-integral z) (cbox a b) ∧ norm (z
− h y) < e)
    proof (rule-tac x=M in exI, clarsimp simp add: M, goal-cases)
      case prems: (1 a b)
      obtain z where z:
        ((λx. if x ∈ s then f x else 0) has-integral z) (cbox a b)
        norm (z − y) < e / B
        using M(2)[OF prems(1)] by blast
      have ∗: (λx. if x ∈ s then (h ∘ f) x else 0) = h ∘ (λx. if x ∈ s then f x else
0)
        using zero by auto
      show ?case
        apply (rule-tac x=h z in exI)
        apply (simp add: ∗ lem z(1))
        apply (metis B diff le-less-trans pos-less-divide-eq z(2))
        done
    qed
  qed
qed

lemma has-integral-scaleR-left:
  (f has-integral y) s ⟹ ((λx. f x ∗R c) has-integral (y ∗R c)) s
  using has-integral-linear[OF - bounded-linear-scaleR-left] by (simp add: comp-def)

lemma has-integral-mult-left:
  fixes c :: - :: {real-normed-algebra}
  shows (f has-integral y) s ⟹ ((λx. f x ∗ c) has-integral (y ∗ c)) s
  using has-integral-linear[OF - bounded-linear-mult-left] by (simp add: comp-def)
```

corollary *integral-mult-left*:
fixes *c*:: *'a*::*real-normed-algebra*
shows *f integrable-on s* $\Longrightarrow$ *integral s* $(\lambda x.\ f\ x * c) =$ *integral s f* $*$ *c*
by (*blast intro*: *has-integral-mult-left*)

lemma *has-integral-mult-right*:
fixes *c* :: *'a* :: *real-normed-algebra*
shows (*f has-integral y*) *i* $\Longrightarrow$ ((*λx*. *c* $*$ *f x*) *has-integral* (*c* $*$ *y*)) *i*
using *has-integral-linear*[*OF - bounded-linear-mult-right*] **by** (*simp add*: *comp-def*)

lemma *has-integral-cmul*: (*f has-integral k*) *s* $\Longrightarrow$ ((*λx*. *c* $*_R$ *f x*) *has-integral* (*c* $*_R$ *k*)) *s*
unfolding *o-def*[*symmetric*]
by (*metis has-integral-linear bounded-linear-scaleR-right*)

lemma *has-integral-cmult-real*:
fixes *c* :: *real*
assumes *c* $\neq$ *0* $\Longrightarrow$ (*f has-integral x*) *A*
shows ((*λx*. *c* $*$ *f x*) *has-integral c* $*$ *x*) *A*
proof (*cases c* = *0*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
from *has-integral-cmul*[*OF assms*[*OF this*], *of c*] **show** *?thesis*
unfolding *real-scaleR-def* **.**
qed

lemma *has-integral-neg*: (*f has-integral k*) *s* $\Longrightarrow$ ((*λx*. $-$(*f x*)) *has-integral* ($-k$)) *s*
by (*drule-tac c*=$-$*1* **in** *has-integral-cmul*) *auto*

lemma *has-integral-add*:
fixes *f* :: *'n*::*euclidean-space* $\Rightarrow$ *'a*::*real-normed-vector*
assumes (*f has-integral k*) *s*
and (*g has-integral l*) *s*
shows ((*λx*. *f x* + *g x*) *has-integral* (*k* + *l*)) *s*
proof $-$
have *lem*: ((*λx*. *f x* + *g x*) *has-integral* (*k* + *l*)) (*cbox a b*)
if *f-k*: (*f has-integral k*) (*cbox a b*)
and *g-l*: (*g has-integral l*) (*cbox a b*)
for *f* :: *'n* $\Rightarrow$ *'a* **and** *g a b k l*
unfolding *has-integral*
proof *clarify*
fix *e* :: *real*
assume *e*: *e* > *0*
then have $*$: *e* / *2* > *0*
by *auto*
obtain *d1* **where** *d1*:

gauge d1
$\bigwedge p$. *p tagged-division-of* (*cbox a b*) $\Longrightarrow$ *d1 fine p* $\Longrightarrow$
norm ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ *f x*) − *k*) < *e* / *2*
using *has-integralD*[*OF f-k* $*$] **by** *blast*
obtain *d2* **where** *d2*:
gauge d2
$\bigwedge p$. *p tagged-division-of* (*cbox a b*) $\Longrightarrow$ *d2 fine p* $\Longrightarrow$
norm ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ *g x*) − *l*) < *e* / *2*
using *has-integralD*[*OF g-l* $*$] **by** *blast*
show ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged-division-of* (*cbox a b*) ∧ *d fine p* ⟶
norm ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ (*f x* + *g x*)) − (*k* + *l*)) < *e*)
proof (*rule exI* [**where** *x*=λ*x*. (*d1 x*) ∩ (*d2 x*)], *clarsimp simp add*: *gauge-inter*[*OF d1(1) d2(1)*])
fix *p*
assume *as*: *p tagged-division-of* (*cbox a b*) (λ*x*. *d1 x* ∩ *d2 x*) *fine p*
have $*$: (∑ (*x*, *k*)∈*p*. *content k* $*_R$ (*f x* + *g x*)) =
(∑ (*x*, *k*)∈*p*. *content k* $*_R$ *f x*) + (∑ (*x*, *k*)∈*p*. *content k* $*_R$ *g x*)
unfolding *scaleR-right-distrib setsum.distrib*[*of* λ(*x*,*k*). *content k* $*_R$ *f x* λ(*x*,*k*). *content k* $*_R$ *g x p*,*symmetric*]
by (*rule setsum.cong*) *auto*
from *as* **have** *fine*: *d1 fine p d2 fine p*
unfolding *fine-inter* **by** *auto*
have *norm* ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ (*f x* + *g x*)) − (*k* + *l*)) =
norm (((∑ (*x*, *k*)∈*p*. *content k* $*_R$ *f x*) − *k*) + ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ *g x*) − *l*))
unfolding $*$ **by** (*auto simp add*: *algebra-simps*)
also have ... < *e*/*2* + *e*/*2*
apply (*rule le-less-trans*[*OF norm-triangle-ineq*])
using *as d1 d2 fine*
apply (*blast intro*: *add-strict-mono*)
done
finally show *norm* ((∑ (*x*, *k*)∈*p*. *content k* $*_R$ (*f x* + *g x*)) − (*k* + *l*)) < *e*
by *auto*
qed
qed
{
presume ¬ (∃ *a b*. *s* = *cbox a b*) $\Longrightarrow$ *?thesis*
then show *?thesis*
using *assms lem* **by** *force*
}
assume *as*: ¬ (∃ *a b*. *s* = *cbox a b*)
then show *?thesis*
proof (*subst has-integral-alt*, *clarsimp*, *goal-cases*)
case (*1 e*)
then have $*$: *e* / *2* > *0*
by *auto*
from *has-integral-altD*[*OF assms*(*1*) *as* $*$]
obtain *B1* **where** *B1*:
0 < *B1*

```
      ⋀a b. ball 0 B1 ⊆ cbox a b ⟹
        ∃z. ((λx. if x ∈ s then f x else 0) has-integral z) (cbox a b) ∧ norm (z −
k) < e / 2
     by blast
   from has-integral-altD[OF assms(2) as *]
   obtain B2 where B2:
      0 < B2
      ⋀a b. ball 0 B2 ⊆ (cbox a b) ⟹
        ∃z. ((λx. if x ∈ s then g x else 0) has-integral z) (cbox a b) ∧ norm (z −
l) < e / 2
     by blast
   show ?case
  proof (rule-tac x=max B1 B2 in exI, clarsimp simp add: max.strict-coboundedI1
B1)
     fix a b
     assume ball 0 (max B1 B2) ⊆ cbox a (b::'n)
     then have *: ball 0 B1 ⊆ cbox a (b::'n) ball 0 B2 ⊆ cbox a (b::'n)
       by auto
     obtain w where w:
       ((λx. if x ∈ s then f x else 0) has-integral w) (cbox a b)
       norm (w − k) < e / 2
       using B1(2)[OF *(1)] by blast
     obtain z where z:
       ((λx. if x ∈ s then g x else 0) has-integral z) (cbox a b)
       norm (z − l) < e / 2
       using B2(2)[OF *(2)] by blast
     have *: ⋀x. (if x ∈ s then f x + g x else 0) =
       (if x ∈ s then f x else 0) + (if x ∈ s then g x else 0)
       by auto
     show ∃z. ((λx. if x ∈ s then f x + g x else 0) has-integral z) (cbox a b) ∧
norm (z − (k + l)) < e
       apply (rule-tac x=w + z in exI)
       apply (simp add: lem[OF w(1) z(1), unfolded *[symmetric]])
       using norm-triangle-ineq[of w − k z − l] w(2) z(2)
       apply (auto simp add: field-simps)
       done
   qed
  qed
qed

lemma has-integral-sub:
  (f has-integral k) s ⟹ (g has-integral l) s ⟹
    ((λx. f x − g x) has-integral (k − l)) s
  using has-integral-add[OF - has-integral-neg, of f k s g l]
  unfolding algebra-simps
  by auto

lemma integral-0:
  integral s (λx::'n::euclidean-space. 0::'m::real-normed-vector) = 0
```

by (*rule integral-unique has-integral-0*)+

lemma *integral-add*: *f integrable-on s* $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$
integral s ($\lambda x.$ *f x* + *g x*) = *integral s f* + *integral s g*
by (*rule integral-unique*) (*metis integrable-integral has-integral-add*)

lemma *integral-cmul*: *f integrable-on s* $\Longrightarrow$ *integral s* ($\lambda x.$ *c* $*_R$ *f x*) = *c* $*_R$ *integral s f*
by (*rule integral-unique*) (*metis integrable-integral has-integral-cmul*)

lemma *integral-neg*: *f integrable-on s* $\Longrightarrow$ *integral s* ($\lambda x.$ − *f x*) = − *integral s f*
by (*rule integral-unique*) (*metis integrable-integral has-integral-neg*)

lemma *integral-diff*: *f integrable-on s* $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$
integral s ($\lambda x.$ *f x* − *g x*) = *integral s f* − *integral s g*
by (*rule integral-unique*) (*metis integrable-integral has-integral-sub*)

lemma *integrable-0*: ($\lambda x.$ *0*) *integrable-on s*
unfolding *integrable-on-def* **using** *has-integral-0* **by** *auto*

lemma *integrable-add*: *f integrable-on s* $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$ ($\lambda x.$ *f x* + *g x*) *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-add*)

lemma *integrable-cmul*: *f integrable-on s* $\Longrightarrow$ ($\lambda x.$ *c* $*_R$ *f*(*x*)) *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-cmul*)

lemma *integrable-on-cmult-iff*:
fixes *c* :: *real*
assumes *c* ≠ *0*
shows ($\lambda x.$ *c* ∗ *f x*) *integrable-on s* $\longleftrightarrow$ *f integrable-on s*
using *integrable-cmul*[*of* $\lambda x.$ *c* ∗ *f x s 1* / *c*] *integrable-cmul*[*of f s c*] ‹*c* ≠ *0*›
by *auto*

lemma *integrable-neg*: *f integrable-on s* $\Longrightarrow$ ($\lambda x.$ −*f*(*x*)) *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-neg*)

lemma *integrable-diff*:
f integrable-on s $\Longrightarrow$ *g integrable-on s* $\Longrightarrow$ ($\lambda x.$ *f x* − *g x*) *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-sub*)

lemma *integrable-linear*:
f integrable-on s $\Longrightarrow$ *bounded-linear h* $\Longrightarrow$ (*h* ∘ *f*) *integrable-on s*
unfolding *integrable-on-def* **by**(*auto intro*: *has-integral-linear*)

lemma *integral-linear*:
f integrable-on s $\Longrightarrow$ *bounded-linear h* $\Longrightarrow$ *integral s* (*h* ∘ *f*) = *h* (*integral s f*)
apply (*rule has-integral-unique* [**where** *i*=*s* **and** *f* = *h* ∘ *f*])
apply (*simp-all add*: *integrable-integral integrable-linear has-integral-linear*)

done

lemma *integral-component-eq*[*simp*]:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'm::euclidean\text{-}space$
assumes *f integrable-on s*
shows *integral s* $(\lambda x.\ f\ x \cdot k) =$ *integral s f* $\cdot\ k$
unfolding *integral-linear*[*OF assms*(*1*) *bounded-linear-component*,*unfolded o-def*]
..

lemma *has-integral-setsum*:
assumes *finite t*
and $\forall\, a{\in}t.$ ((*f a*) *has-integral* (*i a*)) *s*
shows (($\lambda x.$ *setsum* ($\lambda a.$ *f a x*) *t*) *has-integral* (*setsum i t*)) *s*
using *assms*(*1*) *subset-refl*[*of t*]
proof (*induct rule*: *finite-subset-induct*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x F*)
with *assms* **show** *?case*
by (*simp add*: *has-integral-add*)
qed

lemma *integral-setsum*:
$[\![$*finite t*; $\forall\, a{\in}t.$ (*f a*) *integrable-on s*$]\!] \Longrightarrow$
integral s ($\lambda x.$ *setsum* ($\lambda a.$ *f a x*) *t*) = *setsum* ($\lambda a.$ *integral s* (*f a*)) *t*
by (*auto intro*: *has-integral-setsum integrable-integral*)

lemma *integrable-setsum*:
$[\![$*finite t*; $\forall\, a{\in}t.$ (*f a*) *integrable-on s*$]\!] \Longrightarrow$ ($\lambda x.$ *setsum* ($\lambda a.$ *f a x*) *t*) *integrable-on s*
unfolding *integrable-on-def*
apply (*drule bchoice*)
using *has-integral-setsum*[*of t*]
apply *auto*
done

lemma *has-integral-eq*:
assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
and (*f has-integral k*) *s*
shows (*g has-integral k*) *s*
using *has-integral-sub*[*OF assms*(*2*), *of* $\lambda x.\ f\ x - g\ x\ 0$]
using *has-integral-is-0*[*of s* $\lambda x.\ f\ x - g\ x$]
using *assms*(*1*)
by *auto*

lemma *integrable-eq*: $(\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x) \Longrightarrow$ *f integrable-on s* $\Longrightarrow$ *g integrable-on s*
unfolding *integrable-on-def*

using *has-integral-eq*[*of s f g*] *has-integral-eq* **by** *blast*

lemma *has-integral-cong*:
assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows (*f has-integral i*) *s* = (*g has-integral i*) *s*
using *has-integral-eq*[*of s f g*] *has-integral-eq*[*of s g f*] *assms*
by *auto*

lemma *integral-cong*:
assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows *integral s f* = *integral s g*
unfolding *integral-def*
by (*metis assms has-integral-cong*)

lemma *has-integral-null* [*intro*]:
assumes *content*(*cbox a b*) = *0*
shows (*f has-integral 0*) (*cbox a b*)
proof −
have *gauge* (λ*x*. *ball x 1*)
by *auto*
moreover
{
fix *e* :: *real*
fix *p*
assume *e*: *e* > *0*
assume *p*: *p tagged-division-of* (*cbox a b*)
have *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗$_R$ *f x*) − *0*) = *0*
unfolding *norm-eq-zero diff-0-right*
using *setsum-content-null*[*OF assms*(*1*) *p*, *of f*] **.**
then have *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗$_R$ *f x*) − *0*) < *e*
using *e* **by** *auto*
}
ultimately show *?thesis*
by (*auto simp*: *has-integral*)
qed

lemma *has-integral-null-real* [*intro*]:
assumes *content* {*a* .. *b*::*real*} = *0*
shows (*f has-integral 0*) {*a* .. *b*}
by (*metis assms box-real*(*2*) *has-integral-null*)

lemma *has-integral-null-eq*[*simp*]: *content* (*cbox a b*) = *0* ⟹ (*f has-integral i*) (*cbox a b*) ⟷ *i* = *0*
by (*auto simp add*: *has-integral-null dest*!: *integral-unique*)

lemma *integral-null* [*simp*]: *content* (*cbox a b*) = *0* ⟹ *integral* (*cbox a b*) *f* = *0*
by (*metis has-integral-null integral-unique*)

lemma *integrable-on-null* [*intro*]: *content* (*cbox a b*) = *0* ⟹ *f integrable-on* (*cbox*

a b)
by (*simp add*: *has-integral-integrable*)

lemma *has-integral-empty*[*intro*]: (*f has-integral 0*) {}
by (*simp add*: *has-integral-is-0*)

lemma *has-integral-empty-eq*[*simp*]: (*f has-integral i*) {} ⟷ *i* = *0*
by (*auto simp add*: *has-integral-empty has-integral-unique*)

lemma *integrable-on-empty*[*intro*]: *f integrable-on* {}
unfolding *integrable-on-def* **by** *auto*

lemma *integral-empty*[*simp*]: *integral* {} *f* = *0*
by (*rule integral-unique*) (*rule has-integral-empty*)

lemma *has-integral-refl*[*intro*]:
fixes *a* :: *'a*::*euclidean-space*
shows (*f has-integral 0*) (*cbox a a*)
and (*f has-integral 0*) {*a*}
proof −
have ∗: {*a*} = *cbox a a*
apply (*rule set-eqI*)
unfolding *mem-box singleton-iff euclidean-eq-iff*[**where** *'a*=*'a*]
apply *safe*
prefer *3*
apply (*erule-tac x*=*b* **in** *ballE*)
apply (*auto simp add*: *field-simps*)
done
show (*f has-integral 0*) (*cbox a a*) (*f has-integral 0*) {*a*}
unfolding ∗
apply (*rule-tac*[!] *has-integral-null*)
unfolding *content-eq-0-interior*
unfolding *interior-cbox*
using *box-sing*
apply *auto*
done
qed

lemma *integrable-on-refl*[*intro*]: *f integrable-on cbox a a*
unfolding *integrable-on-def* **by** *auto*

lemma *integral-refl* [*simp*]: *integral* (*cbox a a*) *f* = *0*
by (*rule integral-unique*) *auto*

lemma *integral-singleton* [*simp*]: *integral* {*a*} *f* = *0*
by *auto*

lemma *integral-blinfun-apply*:
assumes *f integrable-on s*

shows *integral s* (λ*x*. *blinfun-apply h* (*f x*)) = *blinfun-apply h* (*integral s f*)
by (*subst integral-linear*[*symmetric*, *OF assms blinfun.bounded-linear-right*]) (*simp add*: *o-def*)

lemma *blinfun-apply-integral*:
assumes *f integrable-on s*
shows *blinfun-apply* (*integral s f*) *x* = *integral s* (λ*y*. *blinfun-apply* (*f y*) *x*)
by (*metis* (*no-types*, *lifting*) *assms blinfun.prod-left.rep-eq integral-blinfun-apply integral-cong*)

35.15 Cauchy-type criterion for integrability.

lemma *integrable-cauchy*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::{*real-normed-vector*,*complete-space*}
shows *f integrable-on cbox a b* ⟷
(∀ *e*>*0*.∃ *d*. *gauge d* ∧
(∀ *p1 p2*. *p1 tagged-division-of* (*cbox a b*) ∧ *d fine p1* ∧
p2 tagged-division-of (*cbox a b*) ∧ *d fine p2* ⟶
norm (*setsum* (λ(*x*,*k*). *content k* ∗*R f x*) *p1* −
setsum (λ(*x*,*k*). *content k* ∗*R f x*) *p2*) < *e*))
(**is** *?l* = (∀ *e*>*0*. ∃ *d*. *?P e d*))
proof
assume *?l*
then guess *y* **unfolding** *integrable-on-def has-integral* **..** **note** *y*=*this*
show ∀ *e*>*0*. ∃ *d*. *?P e d*
proof (*clarify*, *goal-cases*)
case (*1 e*)
then have *e*/*2* > *0* **by** *auto*
then guess *d*
apply −
apply (*drule y*[*rule-format*])
apply (*elim exE conjE*)
done
note *d*=*this*[*rule-format*]
show *?case*
proof (*rule-tac x*=*d* **in** *exI*, *clarsimp simp*: *d*)
fix *p1 p2*
assume *as*: *p1 tagged-division-of* (*cbox a b*) *d fine p1*
p2 tagged-division-of (*cbox a b*) *d fine p2*
show *norm* ((∑ (*x*, *k*)∈*p1*. *content k* ∗*R f x*) − (∑ (*x*, *k*)∈*p2*. *content k* ∗*R f x*)) < *e*
apply (*rule dist-triangle-half-l*[**where** *y*=*y*,*unfolded dist-norm*])
using *d*(*2*)[*OF conjI*[*OF as*(*1*−*2*)]] *d*(*2*)[*OF conjI*[*OF as*(*3*−*4*)]] **.**
qed
qed
next
assume ∀ *e*>*0*. ∃ *d*. *?P e d*
then have ∀ *n*::*nat*. ∃ *d*. *?P* (*inverse*(*of-nat* (*n* + *1*))) *d*
by *auto*

from *choice*[*OF this*] **guess** *d* **.. note** *d*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]
have $\bigwedge n.$ *gauge* ($\lambda x.$ $\bigcap$ {*d i x* |*i*. *i* ∈ {*0*..*n*}})
apply (*rule gauge-inters*)
using *d*(*1*)
apply *auto*
done
then have $\forall n.$ $\exists p.$ *p tagged-division-of* (*cbox a b*) ∧ ($\lambda x.$ $\bigcap$ {*d i x* |*i*. *i* ∈ {*0*..*n*}}) *fine p*
by (*meson fine-division-exists*)
from *choice*[*OF this*] **guess** *p* **.. note** *p* = *conjunctD2*[*OF this*[*rule-format*]]
have *dp*: $\bigwedge i$ *n*. *i*≤*n* ⟹ *d i fine p n*
using *p*(*2*) **unfolding** *fine-inters* **by** *auto*
have *Cauchy* ($\lambda n.$ *setsum* (λ(*x*,*k*). *content k* $*_R$ (*f x*)) (*p n*))
proof (*rule CauchyI*, *goal-cases*)
case (*1 e*)
then guess *N* **unfolding** *real-arch-inv*[*of e*] **.. note** *N*=*this*
show *?case*
apply (*rule-tac x*=*N* **in** *exI*)
proof *clarify*
fix *m n*
assume *mn*: *N* ≤ *m N* ≤ *n*
have ∗: *N* = (*N* − *1*) + *1* **using** *N* **by** *auto*
show *norm* ((∑ (*x*, *k*)∈*p m*. *content k* $*_R$ *f x*) − (∑ (*x*, *k*)∈*p n*. *content k* $*_R$ *f x*)) < *e*
apply (*rule less-trans*[*OF - N*[*THEN conjunct2*,*THEN conjunct2*]])
apply(*subst* ∗)
using *dp p*(*1*) *mn d*(*2*) **by** *auto*
qed
qed
then guess *y* **unfolding** *convergent-eq-cauchy*[*symmetric*] **.. note** *y*=*this*[*THEN LIMSEQ-D*]
show *?l*
unfolding *integrable-on-def has-integral*
proof (*rule-tac x*=*y* **in** *exI*, *clarify*)
fix *e* :: *real*
assume *e*>*0*
then have ∗:*e*/*2* > *0* **by** *auto*
then guess *N1* **unfolding** *real-arch-inv*[*of e*/*2*] **.. note** *N1*=*this*
then have *N1'*: *N1* = *N1* − *1* + *1*
by *auto*
guess *N2* **using** *y*[*OF* ∗] **.. note** *N2*=*this*
have *gauge* (*d* (*N1* + *N2*))
using *d* **by** *auto*
moreover
{
fix *q*
assume *as*: *q tagged-division-of* (*cbox a b*) *d* (*N1* + *N2*) *fine q*
have ∗: *inverse* (*of-nat* (*N1* + *N2* + *1*)) < *e* / *2*
apply (*rule less-trans*)

using *N1*
apply *auto*
done
have *norm* $((\sum (x,\ k){\in}q.\ content\ k *_R f\ x) - y) < e$
apply (*rule norm-triangle-half-r*)
apply (*rule less-trans*[*OF* - *]*)
apply (*subst N1'*, *rule d*(*2*)[*of p* (*N1*+*N2*)])
using *N1' as*(*1*) *as*(*2*) *dp*
apply (*simp add*: ‹∀ *x*. *p x tagged-division-of cbox a b* ∧ (λ*xa*. ⋂{*d i xa* |*i*. *i* ∈ {*0*..*x*}}) *fine p x*›)
using *N2 le-add2* **by** *blast*
}
ultimately show ∃ *d*. *gauge d* ∧
(∀ *p*. *p tagged-division-of* (*cbox a b*) ∧ *d fine p* ⟶
norm $((\sum (x,\ k){\in}p.\ content\ k *_R f\ x) - y) < e$)
by (*rule-tac x*=*d* (*N1* + *N2*) **in** *exI*) *auto*
qed
qed

35.16 Additivity of integral on abutting intervals.

lemma *interval-split*:
fixes *a* :: *'a*::*euclidean-space*
assumes *k* ∈ *Basis*
shows
cbox a b ∩ {*x*. *x*•*k* ≤ *c*} = *cbox a* ($\sum i{\in}Basis.$ (*if i* = *k then min* (*b*•*k*) *c else b*•*i*) **_R i*)
cbox a b ∩ {*x*. *x*•*k* ≥ *c*} = *cbox* ($\sum i{\in}Basis.$ (*if i* = *k then max* (*a*•*k*) *c else a*•*i*) **_R i*) *b*
apply (*rule-tac*[!] *set-eqI*)
unfolding *Int-iff mem-box mem-Collect-eq*
using *assms*
apply *auto*
done

lemma *content-split*:
fixes *a* :: *'a*::*euclidean-space*
assumes *k* ∈ *Basis*
shows *content* (*cbox a b*) = *content*(*cbox a b* ∩ {*x*. *x*•*k* ≤ *c*}) + *content*(*cbox a b* ∩ {*x*. *x*•*k* ≥ *c*})
proof *cases*
note *simps* = *interval-split*[*OF assms*] *content-cbox-cases*
have ***: *Basis* = *insert k* (*Basis* − {*k*}) ⋀*x*. *finite* (*Basis*−{*x*}) ⋀*x*. *x*∉*Basis*−{*x*}
using *assms* **by** *auto*
have ***: ⋀*X Y Z*. ($\prod i{\in}Basis.$ *Z i* (*if i* = *k then X else Y i*)) = *Z k X ** ($\prod i{\in}Basis{-}\{k\}.$ *Z i* (*Y i*))
($\prod i{\in}Basis.$ *b*•*i* − *a*•*i*) = ($\prod i{\in}Basis{-}\{k\}.$ *b*•*i* − *a*•*i*) * (*b*•*k* − *a*•*k*)
apply (*subst **(*1*))
defer

```
    apply (subst *(1))
    unfolding setprod.insert[OF *(2-)]
    apply auto
    done
  assume as: ∀i∈Basis. a · i ≤ b · i
  moreover
  have ⋀x. min (b · k) c = max (a · k) c ⟹
    x * (b·k - a·k) = x * (max (a · k) c - a · k) + x * (b · k - max (a · k) c)
    by (auto simp add: field-simps)
  moreover
  have **: (∏i∈Basis. ((∑i∈Basis. (if i = k then min (b · k) c else b · i) *R i)
· i - a · i)) =
      (∏i∈Basis. (if i = k then min (b · k) c else b · i) - a · i)
    (∏i∈Basis. b · i - ((∑i∈Basis. (if i = k then max (a · k) c else a · i) *R i)
· i)) =
      (∏i∈Basis. b · i - (if i = k then max (a · k) c else a · i))
    by (auto intro!: setprod.cong)
  have ¬ a · k ≤ c ⟹ ¬ c ≤ b · k ⟹ False
    unfolding not-le
    using as[unfolded ,rule-format,of k] assms
    by auto
  ultimately show ?thesis
    using assms
    unfolding simps **
    unfolding *(1)[of λi x. b·i - x] *(1)[of λi x. x - a·i]
    unfolding *(2)
    by auto
next
  assume ¬ (∀i∈Basis. a · i ≤ b · i)
  then have cbox a b = {}
    unfolding box-eq-empty by (auto simp: not-le)
  then show ?thesis
    by (auto simp: not-le)
qed

lemma division-split-left-inj:
  fixes type :: 'a::euclidean-space
  assumes d division-of i
    and k1 ∈ d
    and k2 ∈ d
    and k1 ≠ k2
    and k1 ∩ {x::'a. x·k ≤ c} = k2 ∩ {x. x·k ≤ c}
    and k: k∈Basis
  shows content(k1 ∩ {x. x·k ≤ c}) = 0
proof -
  note d=division-ofD[OF assms(1)]
  have *: ⋀(a::'a) b c. content (cbox a b ∩ {x. x·k ≤ c}) = 0 ⟷
    interior(cbox a b ∩ {x. x·k ≤ c}) = {}
    unfolding interval-split[OF k] content-eq-0-interior by auto
```

```
  guess u1 v1 using d(4)[OF assms(2)] by (elim exE) note uv1=this
  guess u2 v2 using d(4)[OF assms(3)] by (elim exE) note uv2=this
  have **: ⋀s t u. s ∩ t = {} ⟹ u ⊆ s ⟹ u ⊆ t ⟹ u = {}
    by auto
  show ?thesis
    unfolding uv1 uv2 *
    apply (rule **[OF d(5)[OF assms(2-4)]])
    apply (simp add: uv1)
    using assms(5) uv1 by auto
qed

lemma division-split-right-inj:
  fixes type :: 'a::euclidean-space
  assumes d division-of i
    and k1 ∈ d
    and k2 ∈ d
    and k1 ≠ k2
    and k1 ∩ {x::'a. x•k ≥ c} = k2 ∩ {x. x•k ≥ c}
    and k: k ∈ Basis
  shows content (k1 ∩ {x. x•k ≥ c}) = 0
proof -
  note d=division-ofD[OF assms(1)]
  have *: ⋀a b::'a. ⋀c. content(cbox a b ∩ {x. x•k ≥ c}) = 0 ⟷
    interior(cbox a b ∩ {x. x•k ≥ c}) = {}
    unfolding interval-split[OF k] content-eq-0-interior by auto
  guess u1 v1 using d(4)[OF assms(2)] by (elim exE) note uv1=this
  guess u2 v2 using d(4)[OF assms(3)] by (elim exE) note uv2=this
  have **: ⋀s t u. s ∩ t = {} ⟹ u ⊆ s ⟹ u ⊆ t ⟹ u = {}
    by auto
  show ?thesis
    unfolding uv1 uv2 *
    apply (rule **[OF d(5)[OF assms(2-4)]])
    apply (simp add: uv1)
    using assms(5) uv1 by auto
qed

lemma tagged-division-split-left-inj:
  fixes x1 :: 'a::euclidean-space
  assumes d: d tagged-division-of i
    and k12: (x1, k1) ∈ d
             (x2, k2) ∈ d
             k1 ≠ k2
             k1 ∩ {x. x•k ≤ c} = k2 ∩ {x. x•k ≤ c}
             k ∈ Basis
  shows content (k1 ∩ {x. x•k ≤ c}) = 0
proof -
  have *: ⋀a b c. (a,b) ∈ c ⟹ b ∈ snd ` c
    by force
  show ?thesis
```

using *k12*
by (*fastforce intro*!: *division-split-left-inj*[*OF division-of-tagged-division*[*OF d*]]
∗)
qed

lemma *tagged-division-split-right-inj*:
fixes *x1* :: *'a*::*euclidean-space*
assumes *d*: *d tagged-division-of i*
and *k12*: (*x1*, *k1*) ∈ *d*
(*x2*, *k2*) ∈ *d*
k1 ≠ *k2*
k1 ∩ {*x*. *x*•*k* ≥ *c*} = *k2* ∩ {*x*. *x*•*k* ≥ *c*}
k ∈ *Basis*
shows *content* (*k1* ∩ {*x*. *x*•*k* ≥ *c*}) = *0*
proof −
have ∗: ⋀*a b c*. (*a*,*b*) ∈ *c* ⟹ *b* ∈ *snd* ‘ *c*
by *force*
show *?thesis*
using *k12*
by (*fastforce intro*!: *division-split-right-inj*[*OF division-of-tagged-division*[*OF d*]] ∗)
qed

lemma *division-split*:
fixes *a* :: *'a*::*euclidean-space*
assumes *p division-of* (*cbox a b*)
and *k*: *k*∈*Basis*
shows {*l* ∩ {*x*. *x*•*k* ≤ *c*} | *l*. *l* ∈ *p* ∧ *l* ∩ {*x*. *x*•*k* ≤ *c*} ≠ {}} *division-of*(*cbox a b* ∩ {*x*. *x*•*k* ≤ *c*})
(**is** *?p1 division-of ?I1*)
and {*l* ∩ {*x*. *x*•*k* ≥ *c*} | *l*. *l* ∈ *p* ∧ *l* ∩ {*x*. *x*•*k* ≥ *c*} ≠ {}} *division-of* (*cbox a b* ∩ {*x*. *x*•*k* ≥ *c*})
(**is** *?p2 division-of ?I2*)
proof (*rule-tac*[!] *division-ofI*)
note *p* = *division-ofD*[*OF assms*(*1*)]
show *finite ?p1 finite ?p2*
using *p*(*1*) **by** *auto*
show ⋃ *?p1* = *?I1* ⋃ *?p2* = *?I2*
unfolding *p*(*6*)[*symmetric*] **by** *auto*
{
fix *k*
assume *k* ∈ *?p1*
then guess *l* **unfolding** *mem-Collect-eq* **by** (*elim exE conjE*) **note** *l*=*this*
guess *u v* **using** *p*(*4*)[*OF l*(*2*)] **by** (*elim exE*) **note** *uv*=*this*
show *k* ⊆ *?I1*
using *l p*(*2*) *uv* **by** *force*
show *k* ≠ {}
by (*simp add*: *l*)
show ∃ *a b*. *k* = *cbox a b*

```
    apply (simp add: l uv p(2-3)[OF l(2)])
    apply (subst interval-split[OF k])
    apply (auto intro: order.trans)
    done
  fix k'
  assume k' ∈ ?p1
  then guess l' unfolding mem-Collect-eq by (elim exE conjE) note l'=this
  assume k ≠ k'
  then show interior k ∩ interior k' = {}
    unfolding l l' using p(5)[OF l(2) l'(2)] by auto
 }
 {
  fix k
  assume k ∈ ?p2
  then guess l unfolding mem-Collect-eq by (elim exE conjE) note l=this
  guess u v using p(4)[OF l(2)] by (elim exE) note uv=this
  show k ⊆ ?I2
    using l p(2) uv by force
  show  k ≠ {}
    by (simp add: l)
  show  ∃ a b. k = cbox a b
    apply (simp add: l uv p(2-3)[OF l(2)])
    apply (subst interval-split[OF k])
    apply (auto intro: order.trans)
    done
  fix k'
  assume k' ∈ ?p2
  then guess l' unfolding mem-Collect-eq by (elim exE conjE) note l'=this
  assume k ≠ k'
  then show interior k ∩ interior k' = {}
    unfolding l l' using p(5)[OF l(2) l'(2)] by auto
 }
qed

lemma has-integral-split:
 fixes f :: 'a::euclidean-space ⇒ 'b::real-normed-vector
 assumes fi: (f has-integral i) (cbox a b ∩ {x. x•k ≤ c})
    and fj: (f has-integral j) (cbox a b ∩ {x. x•k ≥ c})
    and k: k ∈ Basis
 shows (f has-integral (i + j)) (cbox a b)
proof (unfold has-integral, rule, rule, goal-cases)
 case (1 e)
 then have e: e/2 > 0
   by auto
   obtain d1
   where d1: gauge d1
     and d1norm:
       ⋀p. ⟦p tagged-division-of cbox a b ∩ {x. x • k ≤ c};
            d1 fine p⟧ ⟹ norm ((∑(x, k) ∈ p. content k *R f x) - i) < e / 2
```

```
      apply (rule has-integralD[OF fi[unfolded interval-split[OF k]] e])
      apply (simp add: interval-split[symmetric] k)
      done
    obtain d2
    where d2: gauge d2
      and d2norm:
        ⋀p. ⟦p tagged-division-of cbox a b ∩ {x. c ≤ x · k};
              d2 fine p⟧ ⟹ norm ((∑(x, k) ∈ p. content k *R f x) − j) < e / 2
      apply (rule has-integralD[OF fj[unfolded interval-split[OF k]] e])
      apply (simp add: interval-split[symmetric] k)
      done
  let ?d = λx. if x·k = c then (d1 x ∩ d2 x) else ball x |x·k − c| ∩ d1 x ∩ d2 x
  have gauge ?d
    using d1 d2 unfolding gauge-def by auto
  then show ?case
  proof (rule-tac x=?d in exI, safe)
    fix p
    assume p tagged-division-of (cbox a b) ?d fine p
    note p = this tagged-division-ofD[OF this(1)]
    have xk-le-c: ⋀x kk. (x, kk) ∈ p ⟹ kk ∩ {x. x·k ≤ c} ≠ {} ⟹ x·k ≤ c
    proof −
      fix x kk
      assume as: (x, kk) ∈ p and kk: kk ∩ {x. x·k ≤ c} ≠ {}
      show x·k ≤ c
      proof (rule ccontr)
        assume **: ¬ ?thesis
        from this[unfolded not-le]
        have kk ⊆ ball x |x · k − c|
          using p(2)[unfolded fine-def, rule-format,OF as] by auto
        with kk obtain y where y: y ∈ ball x |x · k − c| y·k ≤ c
          by blast
        then have |x · k − y · k| < |x · k − c|
          using Basis-le-norm[OF k, of x − y]
          by (auto simp add: dist-norm inner-diff-left intro: le-less-trans)
        with y show False
          using ** by (auto simp add: field-simps)
      qed
    qed
    have xk-ge-c: ⋀x kk. (x, kk) ∈ p ⟹ kk ∩ {x. x·k ≥ c} ≠ {} ⟹ x·k ≥ c
    proof −
      fix x kk
      assume as: (x, kk) ∈ p and kk: kk ∩ {x. x·k ≥ c} ≠ {}
      show x·k ≥ c
      proof (rule ccontr)
        assume **: ¬ ?thesis
        from this[unfolded not-le] have kk ⊆ ball x |x · k − c|
          using p(2)[unfolded fine-def,rule-format,OF as,unfolded split-conv] by
auto
        with kk obtain y where y: y ∈ ball x |x · k − c| y·k ≥ c
```

```
      by blast
    then have |x · k − y · k| < |x · k − c|
      using Basis-le-norm[OF k, of x − y]
      by (auto simp add: dist-norm inner-diff-left intro: le-less-trans)
    with y show False
      using ** by (auto simp add: field-simps)
  qed
 qed

 have lem1: ⋀f P Q. (∀x k. (x, k) ∈ {(x, f k) | x k. P x k} ⟶ Q x k) ⟷
                (∀x k. P x k ⟶ Q x (f k))
  by auto
 have fin-finite: finite {(x,f k) | x k. (x,k) ∈ s ∧ P x k} if finite s for f s P
 proof −
  from that have finite ((λ(x, k). (x, f k)) ‘ s)
   by auto
  then show ?thesis
   by (rule rev-finite-subset) auto
 qed
 { fix g :: 'a set ⇒ 'a set
  fix i :: 'a × 'a set
  assume i ∈ (λ(x, k). (x, g k)) ‘ p − {(x, g k) |x k. (x, k) ∈ p ∧ g k ≠ {}}
  then obtain x k where xk:
      i = (x, g k)  (x, k) ∈ p
      (x, g k) ∉ {(x, g k) |x k. (x, k) ∈ p ∧ g k ≠ {}}
    by auto
  have content (g k) = 0
   using xk using content-empty by auto
  then have (λ(x, k). content k *R f x) i = 0
   unfolding xk split-conv by auto
 } note [simp] = this
 have lem3: ⋀g :: 'a set ⇒ 'a set. finite p ⟹
          setsum (λ(x, k). content k *R f x) {(x,g k) |x k. (x,k) ∈ p ∧ g k ≠
{}} =
          setsum (λ(x, k). content k *R f x) ((λ(x, k). (x, g k)) ‘ p)
  by (rule setsum.mono-neutral-left) auto
 let ?M1 = {(x, kk ∩ {x. x·k ≤ c}) |x kk. (x, kk) ∈ p ∧ kk ∩ {x. x·k ≤ c} ≠
{}}
 have d1-fine: d1 fine ?M1
  by (force intro: fineI dest: fineD[OF p(2)] simp add: split: split-if-asm)
 have norm ((∑(x, k)∈?M1. content k *R f x) − i) < e/2
 proof (rule d1norm [OF tagged-division-ofI d1-fine])
  show finite ?M1
   by (rule fin-finite p(3))+
  show ⋃{k. ∃x. (x, k) ∈ ?M1} = cbox a b ∩ {x. x·k ≤ c}
   unfolding p(8)[symmetric] by auto
  fix x l
  assume xl: (x, l) ∈ ?M1
  then guess x' l' unfolding mem-Collect-eq unfolding prod.inject by (elim
```

exE conjE) **note** *xl'=this*
show $x \in l\ l \subseteq cbox\ a\ b \cap \{x.\ x \cdot k \leq c\}$
unfolding *xl'*
using *p(4−6)[OF xl'(3)]* **using** *xl'(4)*
using *xk-le-c[OF xl'(3−4)]* **by** *auto*
show $\exists a\ b.\ l = cbox\ a\ b$
unfolding *xl'*
using *p(6)[OF xl'(3)]*
by (*fastforce simp add*: *interval-split*[*OF k*,**where** *c=c*])
fix *y r*
let *?goal* = *interior l* ∩ *interior r* = {}
assume *yr*: (*y*, *r*) ∈ *?M1*
then guess *y' r'* **unfolding** *mem-Collect-eq* **unfolding** *prod.inject* **by** (*elim exE conjE*) **note** *yr'=this*
assume *as*: (*x*, *l*) ≠ (*y*, *r*)
show *interior l* ∩ *interior r* = {}
proof (*cases l' = r' ⟶ x' = y'*)
case *False*
then show *?thesis*
using *p(7)[OF xl'(3) yr'(3)]* **using** *as* **unfolding** *xl' yr'* **by** *auto*
next
case *True*
then have *l' ≠ r'*
using *as* **unfolding** *xl' yr'* **by** *auto*
then show *?thesis*
using *p(7)[OF xl'(3) yr'(3)]* **using** *as* **unfolding** *xl' yr'* **by** *auto*
qed
qed
moreover
let *?M2* = {(*x*,*kk* ∩ {*x*. *x*⋅*k* ≥ *c*}) |*x kk*. (*x*,*kk*) ∈ *p* ∧ *kk* ∩ {*x*. *x*⋅*k* ≥ *c*} ≠ {}}
have *d2-fine*: *d2 fine ?M2*
by (*force intro*: *fineI dest*: *fineD*[*OF p(2)*] *simp add*: *split*: *split-if-asm*)
have *norm* ((∑ (*x*, *k*)∈*?M2*. *content k* ∗$_R$ *f x*) − *j*) < *e*/2
proof (*rule d2norm* [*OF tagged-division-ofI d2-fine*])
show *finite ?M2*
by (*rule fin-finite p(3)*)+
show ⋃{*k*. ∃*x*. (*x*, *k*) ∈ *?M2*} = *cbox a b* ∩ {*x*. *x*⋅*k* ≥ *c*}
unfolding *p(8)*[*symmetric*] **by** *auto*
fix *x l*
assume *xl*: (*x*, *l*) ∈ *?M2*
then guess *x' l'* **unfolding** *mem-Collect-eq* **unfolding** *prod.inject* **by** (*elim exE conjE*) **note** *xl'=this*
show $x \in l\ l \subseteq cbox\ a\ b \cap \{x.\ x \cdot k \geq c\}$
unfolding *xl'*
using *p(4−6)[OF xl'(3)] xl'(4) xk-ge-c[OF xl'(3−4)]*
by *auto*
show ∃*a b*. *l* = *cbox a b*
unfolding *xl'*

using *p*(6)[*OF xl'*(3)]
by (*fastforce simp add*: *interval-split*[*OF k*, **where** *c*=*c*])
fix *y r*
let *?goal* = *interior l* ∩ *interior r* = {}
assume *yr*: (*y*, *r*) ∈ *?M2*
then guess *y' r'* **unfolding** *mem-Collect-eq* **unfolding** *prod.inject* **by** (*elim exE conjE*) **note** *yr'*=*this*
assume *as*: (*x*, *l*) ≠ (*y*, *r*)
show *interior l* ∩ *interior r* = {}
proof (*cases l'* = *r'* ⟶ *x'* = *y'*)
case *False*
then show *?thesis*
using *p*(7)[*OF xl'*(3) *yr'*(3)] **using** *as* **unfolding** *xl' yr'* **by** *auto*
next
case *True*
then have *l'* ≠ *r'*
using *as* **unfolding** *xl' yr'* **by** *auto*
then show *?thesis*
using *p*(7)[*OF xl'*(3) *yr'*(3)] **using** *as* **unfolding** *xl' yr'* **by** *auto*
qed
qed
ultimately
have *norm* ((($\sum$(*x*, *k*)∈*?M1*. *content k* ∗$_R$ *f x*) − *i*) + (($\sum$(*x*, *k*)∈*?M2*. *content k* ∗$_R$ *f x*) − *j*)) < *e*/2 + *e*/2
using *norm-add-less* **by** *blast*
also **{**
have *eq0*: ⋀*x y*. *x* = (0::*real*) ⟹ *x* ∗$_R$ (*y*::'*b*) = 0
using *scaleR-zero-left* **by** *auto*
have *cont-eq*: ⋀*g*. (λ(*x*,*l*). *content l* ∗$_R$ *f x*) ∘ (λ(*x*,*l*). (*x*,*g l*)) = (λ(*x*,*l*). *content* (*g l*) ∗$_R$ *f x*)
by *auto*
have (($\sum$(*x*, *k*)∈*?M1*. *content k* ∗$_R$ *f x*) − *i*) + (($\sum$(*x*, *k*)∈*?M2*. *content k* ∗$_R$ *f x*) − *j*) =
($\sum$(*x*, *k*)∈*?M1*. *content k* ∗$_R$ *f x*) + ($\sum$(*x*, *k*)∈*?M2*. *content k* ∗$_R$ *f x*) − (*i* + *j*)
by *auto*
also have ... = ($\sum$(*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. *x* · *k* ≤ *c*}) ∗$_R$ *f x*) +
($\sum$(*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. *c* ≤ *x* · *k*}) ∗$_R$ *f x*) − (*i* + *j*)
unfolding *lem3*[*OF p*(3)]
by (*subst setsum.reindex-nontrivial*[*OF p*(3)], *auto intro*!: *k eq0 tagged-division-split-left-inj*[*OF p*(1)] *tagged-division-split-right-inj*[*OF p*(1)]
simp: *cont-eq*)+
also note *setsum.distrib*[*symmetric*]
also have ⋀*x*. *x* ∈ *p* ⟹
(λ(*x*,*ka*). *content* (*ka* ∩ {*x*. *x* · *k* ≤ *c*}) ∗$_R$ *f x*) *x* +
(λ(*x*,*ka*). *content* (*ka* ∩ {*x*. *c* ≤ *x* · *k*}) ∗$_R$ *f x*) *x* =
(λ(*x*,*ka*). *content ka* ∗$_R$ *f x*) *x*
proof *clarify*
fix *a b*

```
      assume (a, b) ∈ p
      from p(6)[OF this] guess u v by (elim exE) note uv=this
      then show content (b ∩ {x. x · k ≤ c}) *R f a + content (b ∩ {x. c ≤ x
· k}) *R f a =
        content b *R f a
        unfolding scaleR-left-distrib[symmetric]
        unfolding uv content-split[OF k,of u v c]
        by auto
    qed
    note setsum.cong [OF - this]
    finally have (∑(x, k)∈{(x, kk ∩ {x. x · k ≤ c}) |x kk. (x, kk) ∈ p ∧ kk ∩
{x. x · k ≤ c} ≠ {}}. content k *R f x) − i +
      ((∑(x, k)∈{(x, kk ∩ {x. c ≤ x · k}) |x kk. (x, kk) ∈ p ∧ kk ∩ {x. c ≤ x ·
k} ≠ {}}. content k *R f x) − j) =
      (∑(x, ka)∈p. content ka *R f x) − (i + j)
      by auto
  }
  finally show norm ((∑(x, k)∈p. content k *R f x) − (i + j)) < e
    by auto
 qed
qed
```

35.17 A sort of converse, integrability on subintervals.

```
lemma tagged-division-union-interval:
 fixes a :: 'a::euclidean-space
 assumes p1 tagged-division-of (cbox a b ∩ {x. x•k ≤ (c::real)})
   and p2 tagged-division-of (cbox a b ∩ {x. x•k ≥ c})
   and k: k ∈ Basis
 shows (p1 ∪ p2) tagged-division-of (cbox a b)
proof −
 have *: cbox a b = (cbox a b ∩ {x. x•k ≤ c}) ∪ (cbox a b ∩ {x. x•k ≥ c})
   by auto
 show ?thesis
   apply (subst *)
   apply (rule tagged-division-union[OF assms(1−2)])
   unfolding interval-split[OF k] interior-cbox
   using k
   apply (auto simp add: box-def elim!: ballE[where x=k])
   done
qed

lemma tagged-division-union-interval-real:
 fixes a :: real
 assumes p1 tagged-division-of ({a .. b} ∩ {x. x•k ≤ (c::real)})
   and p2 tagged-division-of ({a .. b} ∩ {x. x•k ≥ c})
   and k: k ∈ Basis
 shows (p1 ∪ p2) tagged-division-of {a .. b}
 using assms
```

unfolding *box-real*[*symmetric*]
by (*rule tagged-division-union-interval*)

lemma *has-integral-separate-sides*:
fixes $f :: 'a::euclidean\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}vector$
assumes (*f has-integral i*) (*cbox a b*)
and $e > 0$
and k: $k \in Basis$
obtains *d* **where** *gauge d*
$\forall p1\ p2.\ p1\ tagged\text{-}division\text{-}of\ (cbox\ a\ b \cap \{x.\ x{\cdot}k \leq c\}) \wedge d\ fine\ p1 \wedge$
p2 tagged-division-of $(cbox\ a\ b \cap \{x.\ x{\cdot}k \geq c\}) \wedge d\ fine\ p2 \longrightarrow$
$norm\ ((setsum\ (\lambda(x,k).\ content\ k *_R f\ x)\ p1 + setsum\ (\lambda(x,k).\ content\ k *_R f\ x)\ p2) - i) < e$
proof −
guess *d* **using** *has-integralD*[*OF assms*(1−2)] **. note** *d*=*this*
{ fix *p1 p2*
assume *p1 tagged-division-of* $(cbox\ a\ b) \cap \{x.\ x \cdot k \leq c\}$ *d fine p1*
note *p1*=*tagged-division-ofD*[*OF this*(1)] *this*
assume *p2 tagged-division-of* $(cbox\ a\ b) \cap \{x.\ c \leq x \cdot k\}$ *d fine p2*
note *p2*=*tagged-division-ofD*[*OF this*(1)] *this*
note *tagged-division-union-interval*[*OF p1*(7) *p2*(7)] **note** *p12* = *tagged-division-ofD*[*OF this*] *this*
{ fix *a b*
assume *ab*: $(a, b) \in p1 \cap p2$
have $(a, b) \in p1$
using *ab* **by** *auto*
with *p1* **obtain** *u v* **where** *uv*: $b = cbox\ u\ v$ **by** *auto*
have $b \subseteq \{x.\ x{\cdot}k = c\}$
using *ab p1*(3)[*of a b*] *p2*(3)[*of a b*] **by** *fastforce*
moreover
have $interior\ \{x::'a.\ x \cdot k = c\} = \{\}$
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *x* **where** *x*: $x \in interior\ \{x::'a.\ x{\cdot}k = c\}$
by *auto*
then guess *e* **unfolding** *mem-interior* **..** **note** *e*=*this*
have *x*: $x{\cdot}k = c$
using *x interior-subset* **by** *fastforce*
have $*$: $\bigwedge i.\ i \in Basis \Longrightarrow |(x - (x + (e\ /\ 2) *_R k)) \cdot i| = (if\ i = k\ then\ e/2\ else\ 0)$
using *e k* **by** (*auto simp*: *inner-simps inner-not-same-Basis*)
have $(\sum i{\in}Basis.\ |(x - (x + (e\ /\ 2\ ) *_R k)) \cdot i|) =$
$(\sum i{\in}Basis.\ (if\ i = k\ then\ e\ /\ 2\ else\ 0))$
using $*$ **by** (*blast intro*: *setsum.cong*)
also have ... $< e$
apply (*subst setsum.delta*)
using *e*
apply *auto*
done

```
      finally have x + (e/2) *R k ∈ ball x e
        unfolding mem-ball dist-norm by(rule le-less-trans[OF norm-le-l1])
      then have x + (e/2) *R k ∈ {x. x•k = c}
        using e by auto
      then show False
        unfolding mem-Collect-eq using e x k by (auto simp: inner-simps)
    qed
    ultimately have content b = 0
      unfolding uv content-eq-0-interior
      using interior-mono by blast
    then have content b *R f a = 0
      by auto
  }
  then have norm ((∑(x, k)∈p1. content k *R f x) + (∑(x, k)∈p2. content k
*R f x) − i) =
            norm ((∑(x, k)∈p1 ∪ p2. content k *R f x) − i)
    by (subst setsum.union-inter-neutral) (auto simp: p1 p2)
  also have ... < e
    by (rule k d(2) p12 fine-union p1 p2)+
  finally have norm ((∑(x, k)∈p1. content k *R f x) + (∑(x, k)∈p2. content
k *R f x) − i) < e .
 }
 then show ?thesis
  by (auto intro: that[of d] d elim: )
qed

lemma integrable-split[intro]:
 fixes f :: 'a::euclidean-space ⇒ 'b::{real-normed-vector,complete-space}
 assumes f integrable-on cbox a b
  and k: k ∈ Basis
 shows f integrable-on (cbox a b ∩ {x. x•k ≤ c}) (is ?t1)
  and f integrable-on (cbox a b ∩ {x. x•k ≥ c}) (is ?t2)
proof −
 guess y using assms(1) unfolding integrable-on-def .. note y=this
 def b' ≡ ∑ i∈Basis. (if i = k then min (b•k) c else b•i)*R i::'a
 def a' ≡ ∑ i∈Basis. (if i = k then max (a•k) c else a•i)*R i::'a
 show ?t1 ?t2
  unfolding interval-split[OF k] integrable-cauchy
  unfolding interval-split[symmetric,OF k]
 proof (rule-tac[!] allI impI)+
  fix e :: real
  assume e > 0
  then have e/2>0
   by auto
 from has-integral-separate-sides[OF y this k,of c] guess d . note d=this[rule-format]
  let ?P = λA. ∃ d. gauge d ∧ (∀ p1 p2. p1 tagged-division-of (cbox a b) ∩ A ∧
d fine p1 ∧
    p2 tagged-division-of (cbox a b) ∩ A ∧ d fine p2 ⟶
    norm ((∑(x, k)∈p1. content k *R f x) − (∑(x, k)∈p2. content k *R f x))
```

```
< e)
    show ?P {x. x · k ≤ c}
    proof (rule-tac x=d in exI, clarsimp simp add: d)
      fix p1 p2
      assume as: p1 tagged-division-of (cbox a b) ∩ {x. x · k ≤ c} d fine p1
                p2 tagged-division-of (cbox a b) ∩ {x. x · k ≤ c} d fine p2
      show norm ((∑(x, k)∈p1. content k *R f x) − (∑(x, k)∈p2. content k *R
f x)) < e
      proof (rule fine-division-exists[OF d(1), of a' b] )
        fix p
        assume p tagged-division-of cbox a' b d fine p
        then show ?thesis
          using as norm-triangle-half-l[OF d(2)[of p1 p] d(2)[of p2 p]]
          unfolding interval-split[OF k] b'-def[symmetric] a'-def[symmetric]
          by (auto simp add: algebra-simps)
      qed
    qed
    show ?P {x. x · k ≥ c}
    proof (rule-tac x=d in exI, clarsimp simp add: d)
      fix p1 p2
      assume as: p1 tagged-division-of (cbox a b) ∩ {x. x · k ≥ c} d fine p1
                p2 tagged-division-of (cbox a b) ∩ {x. x · k ≥ c} d fine p2
      show norm ((∑(x, k)∈p1. content k *R f x) − (∑(x, k)∈p2. content k *R
f x)) < e
      proof (rule fine-division-exists[OF d(1), of a b'] )
        fix p
        assume p tagged-division-of cbox a b' d fine p
        then show ?thesis
          using as norm-triangle-half-l[OF d(2)[of p p1] d(2)[of p p2]]
          unfolding interval-split[OF k] b'-def[symmetric] a'-def[symmetric]
          by (auto simp add: algebra-simps)
      qed
    qed
  qed
qed
```

35.18 Generalized notion of additivity.

```
definition neutral opp = (SOME x. ∀y. opp x y = y ∧ opp y x = y)

definition operative :: ('a ⇒ 'a ⇒ 'a) ⇒ (('b::euclidean-space) set ⇒ 'a) ⇒ bool
  where operative opp f ⟷
    (∀a b. content (cbox a b) = 0 ⟶ f (cbox a b) = neutral opp) ∧
    (∀a b c. ∀k∈Basis. f (cbox a b) = opp (f(cbox a b ∩ {x. x·k ≤ c})) (f (cbox
a b ∩ {x. x·k ≥ c})))

lemma operativeD[dest]:
  fixes type :: 'a::euclidean-space
  assumes operative opp f
```

shows $\bigwedge a\ b{::}'a.\ content\ (cbox\ a\ b) = 0 \Longrightarrow f\ (cbox\ a\ b) = neutral\ opp$
and $\bigwedge a\ b\ c\ k.\ k \in Basis \Longrightarrow f\ (cbox\ a\ b) =$
$opp\ (f\ (cbox\ a\ b \cap \{x.\ x{\cdot}k \leq c\}))\ (f\ (cbox\ a\ b \cap \{x.\ x{\cdot}k \geq c\}))$
using *assms* **unfolding** *operative-def* **by** *auto*

lemma *property-empty-interval*: $\forall\ a\ b.\ content\ (cbox\ a\ b) = 0 \longrightarrow P\ (cbox\ a\ b) \Longrightarrow P\ \{\}$
using *content-empty* **unfolding** *empty-as-interval* **by** *auto*

lemma *operative-empty*: *operative opp f* $\Longrightarrow f\ \{\} = neutral\ opp$
unfolding *operative-def* **by** (*rule property-empty-interval*) *auto*

35.19 Using additivity of lifted function to encode definedness.

fun *lifted* **where**
lifted (*opp* :: $'a \Rightarrow 'a \Rightarrow 'b$) (*Some x*) (*Some y*) = *Some* (*opp x y*)
| *lifted opp None* - = (*None*::$'b$ *option*)
| *lifted opp* - *None* = *None*

lemma *lifted-simp-1*[*simp*]: *lifted opp v None* = *None*
by (*induct v*) *auto*

definition *monoidal opp* $\longleftrightarrow$
$(\forall\ x\ y.\ opp\ x\ y = opp\ y\ x) \wedge$
$(\forall\ x\ y\ z.\ opp\ x\ (opp\ y\ z) = opp\ (opp\ x\ y)\ z) \wedge$
$(\forall\ x.\ opp\ (neutral\ opp)\ x = x)$

lemma *monoidalI*:
assumes $\bigwedge x\ y.\ opp\ x\ y = opp\ y\ x$
and $\bigwedge x\ y\ z.\ opp\ x\ (opp\ y\ z) = opp\ (opp\ x\ y)\ z$
and $\bigwedge x.\ opp\ (neutral\ opp)\ x = x$
shows *monoidal opp*
unfolding *monoidal-def* **using** *assms* **by** *fastforce*

lemma *monoidal-ac*:
assumes *monoidal opp*
shows [*simp*]: *opp* (*neutral opp*) *a* = *a*
and [*simp*]: *opp a* (*neutral opp*) = *a*
and *opp a b* = *opp b a*
and *opp* (*opp a b*) *c* = *opp a* (*opp b c*)
and *opp a* (*opp b c*) = *opp b* (*opp a c*)
using *assms* **unfolding** *monoidal-def* **by** *metis*+

lemma *neutral-lifted* [*cong*]:
assumes *monoidal opp*
shows *neutral* (*lifted opp*) = *Some* (*neutral opp*)
proof −
{ fix *x*

assume $\forall y.$ *lifted opp x y* $=$ *y* $\wedge$ *lifted opp y x* $=$ *y*
then have *Some* (*neutral opp*) $=$ *x*
apply (*induct x*)
apply *force*
by (*metis assms lifted.simps*(*1*) *monoidal-ac*(*2*) *option.inject*) **}**
note [*simp*] $=$ *this*
show *?thesis*
apply (*subst neutral-def*)
apply (*intro some-equality allI*)
apply (*induct-tac y*)
apply (*auto simp add*:*monoidal-ac*[*OF assms*])
done
qed

lemma *monoidal-lifted*[*intro*]:
assumes *monoidal opp*
shows *monoidal* (*lifted opp*)
unfolding *monoidal-def split-option-all neutral-lifted*[*OF assms*]
using *monoidal-ac*[*OF assms*]
by *auto*

definition *support opp f s* $= \{x.\ x \in s \wedge f\ x \neq$ *neutral opp*$\}$
definition *fold'* *opp e s* $=$ (*if finite s then Finite-Set.fold opp e s else e*)
definition *iterate opp s f* $=$ *fold'* $(\lambda x\ a.$ *opp* (*f x*) *a*) (*neutral opp*) (*support opp f s*)

lemma *support-subset*[*intro*]: *support opp f s* $\subseteq$ *s*
unfolding *support-def* **by** *auto*

lemma *support-empty*[*simp*]: *support opp f* $\{\} = \{\}$
using *support-subset*[*of opp f* $\{\}$] **by** *auto*

lemma *comp-fun-commute-monoidal*[*intro*]:
assumes *monoidal opp*
shows *comp-fun-commute opp*
unfolding *comp-fun-commute-def*
using *monoidal-ac*[*OF assms*]
by *auto*

lemma *support-clauses*:
$\bigwedge f\ g\ s.$ *support opp f* $\{\} = \{\}$
$\bigwedge f\ g\ s.$ *support opp f* (*insert x s*) $=$
(*if f*(*x*) $=$ *neutral opp then support opp f s else insert x* (*support opp f s*))
$\bigwedge f\ g\ s.$ *support opp f* $(s - \{x\}) =$ (*support opp f s*) $- \{x\}$
$\bigwedge f\ g\ s.$ *support opp f* $(s \cup t) =$ (*support opp f s*) $\cup$ (*support opp f t*)
$\bigwedge f\ g\ s.$ *support opp f* $(s \cap t) =$ (*support opp f s*) $\cap$ (*support opp f t*)
$\bigwedge f\ g\ s.$ *support opp f* $(s - t) =$ (*support opp f s*) $-$ (*support opp f t*)
$\bigwedge f\ g\ s.$ *support opp g* (*f* ‘ *s*) $=$ *f* ‘ (*support opp* $(g \circ f)$ *s*)
unfolding *support-def* **by** *auto*

lemma *finite-support*[*intro*]: *finite s* ⟹ *finite* (*support opp f s*)
unfolding *support-def* **by** *auto*

lemma *iterate-empty*[*simp*]: *iterate opp* {} *f* = *neutral opp*
unfolding *iterate-def fold'-def* **by** *auto*

lemma *iterate-insert*[*simp*]:
assumes *monoidal opp*
and *finite s*
shows *iterate opp* (*insert x s*) *f* =
(*if x* ∈ *s then iterate opp s f else opp* (*f x*) (*iterate opp s f*))
proof (*cases x* ∈ *s*)
case *True*
then show *?thesis* **by** (*auto simp*: *insert-absorb iterate-def*)
next
case *False*
note ∗ = *comp-fun-commute.comp-comp-fun-commute* [*OF comp-fun-commute-monoidal*[*OF assms*(*1*)]]
show *?thesis*
proof (*cases f x* = *neutral opp*)
case *True*
then show *?thesis*
using *assms* ‹*x* ∉ *s*›
by (*auto simp*: *iterate-def support-clauses*)
next
case *False*
with ‹*x* ∉ *s*› ‹*finite s*› *support-subset* **show** *?thesis*
apply (*simp add*: *iterate-def fold'-def support-clauses*)
apply (*subst comp-fun-commute.fold-insert*[*OF* ∗ *finite-support*, *simplified comp-def*])
apply (*force simp add*:)+
done
qed
qed

lemma *iterate-some*:
⟦*monoidal opp*; *finite s*⟧ ⟹ *iterate* (*lifted opp*) *s* (λ*x*. *Some*(*f x*)) = *Some* (*iterate opp s f*)
by (*erule finite-induct*) (*auto simp*: *monoidal-lifted*)

35.20 Two key instances of additivity.

lemma *neutral-add*[*simp*]: *neutral op* + = (*0*::*'a*::*comm-monoid-add*)
unfolding *neutral-def*
by (*force elim*: *allE* [**where** *x*=*0*])

lemma *operative-content*[*intro*]: *operative* (*op* +) *content*
by (*force simp add*: *operative-def content-split*[*symmetric*])

lemma *monoidal-monoid*[*intro*]: *monoidal* ((*op* +)::('*a*::*comm-monoid-add*) ⇒ '*a* ⇒ '*a*)
unfolding *monoidal-def neutral-add*
by (*auto simp add*: *algebra-simps*)

lemma *operative-integral*:
fixes *f* :: '*a*::*euclidean-space* ⇒ '*b*::*banach*
shows *operative* (*lifted*(*op* +)) (λ*i*. *if f integrable-on i then Some*(*integral i f*) *else None*)
unfolding *operative-def neutral-lifted*[*OF monoidal-monoid*] *neutral-add*
proof *safe*
fix *a b c*
fix *k* :: '*a*
assume *k*: *k* ∈ *Basis*
show (*if f integrable-on cbox a b then Some* (*integral* (*cbox a b*) *f*) *else None*) =
lifted op + (*if f integrable-on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*} *then Some* (*integral* (*cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}) *f*) *else None*)
(*if f integrable-on cbox a b* ∩ {*x*. *c* ≤ *x* • *k*} *then Some* (*integral* (*cbox a b* ∩ {*x*. *c* ≤ *x* • *k*}) *f*) *else None*)
proof (*cases f integrable-on cbox a b*)
case *True*
with *k* **show** *?thesis*
apply (*simp add*: *integrable-split*)
apply (*rule integral-unique* [*OF has-integral-split*[*OF - - k*]])
apply (*auto intro*: *integrable-integral*)
done
next
case *False*
have ¬ (*f integrable-on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}) ∨ ¬ (*f integrable-on cbox a b* ∩ {*x*. *c* ≤ *x* • *k*})
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *f integrable-on cbox a b*
unfolding *integrable-on-def*
apply (*rule-tac x*=*integral* (*cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}) *f* + *integral* (*cbox a b* ∩ {*x*. *x* • *k* ≥ *c*}) *f* **in** *exI*)
apply (*rule has-integral-split*[*OF - - k*])
apply (*auto intro*: *integrable-integral*)
done
then show *False*
using *False* **by** *auto*
qed
then show *?thesis*
using *False* **by** *auto*
qed
next
fix *a b* :: '*a*
assume *content* (*cbox a b*) = *0*

then show (*if f integrable-on cbox a b then Some* (*integral* (*cbox a b*) *f*) *else None*) = *Some 0*
using *has-integral-null-eq*
by (*auto simp*: *integrable-on-null*)
qed

35.21 Points of division of a partition.

definition *division-points* (*k*::('*a*::*euclidean-space*) *set*) *d* =
{(*j*,*x*). *j* ∈ *Basis* ∧ (*interval-lowerbound k*)·*j* < *x* ∧ *x* < (*interval-upperbound k*)·*j* ∧
(∃ *i*∈*d*. (*interval-lowerbound i*)·*j* = *x* ∨ (*interval-upperbound i*)·*j* = *x*)}

lemma *division-points-finite*:
fixes *i* :: '*a*::*euclidean-space set*
assumes *d division-of i*
shows *finite* (*division-points i d*)
proof −
note *assm* = *division-ofD*[*OF assms*]
let *?M* = λ*j*. {(*j*,*x*)|*x*. (*interval-lowerbound i*)·*j* < *x* ∧ *x* < (*interval-upperbound i*)·*j* ∧
(∃ *i*∈*d*. (*interval-lowerbound i*)·*j* = *x* ∨ (*interval-upperbound i*)·*j* = *x*)}
have ∗: *division-points i d* = ⋃ (*?M* ' *Basis*)
unfolding *division-points-def* **by** *auto*
show *?thesis*
unfolding ∗ **using** *assm* **by** *auto*
qed

lemma *division-points-subset*:
fixes *a* :: '*a*::*euclidean-space*
assumes *d division-of* (*cbox a b*)
and ∀ *i*∈*Basis*. *a*·*i* < *b*·*i* *a*·*k* < *c* *c* < *b*·*k*
and *k*: *k* ∈ *Basis*
shows *division-points* (*cbox a b* ∩ {*x*. *x*·*k* ≤ *c*}) {*l* ∩ {*x*. *x*·*k* ≤ *c*} | *l* . *l* ∈ *d* ∧ *l* ∩ {*x*. *x*·*k* ≤ *c*} ≠ {}} ⊆
division-points (*cbox a b*) *d* (**is** *?t1*)
and *division-points* (*cbox a b* ∩ {*x*. *x*·*k* ≥ *c*}) {*l* ∩ {*x*. *x*·*k* ≥ *c*} | *l* . *l* ∈ *d* ∧ ~(*l* ∩ {*x*. *x*·*k* ≥ *c*} = {})} ⊆
division-points (*cbox a b*) *d* (**is** *?t2*)
proof −
note *assm* = *division-ofD*[*OF assms*(*1*)]
have ∗: ∀ *i*∈*Basis*. *a*·*i* ≤ *b*·*i*
∀ *i*∈*Basis*. *a*·*i* ≤ (∑ *i*∈*Basis*. (*if i* = *k then min* (*b* · *k*) *c else* *b* · *i*) ∗$_R$ *i*) · *i*
∀ *i*∈*Basis*. (∑ *i*∈*Basis*. (*if i* = *k then max* (*a* · *k*) *c else a* · *i*) ∗$_R$ *i*) · *i* ≤ *b*·*i*
min (*b* · *k*) *c* = *c max* (*a* · *k*) *c* = *c*
using *assms* **using** *less-imp-le* **by** *auto*
show *?t1*
unfolding *division-points-def interval-split*[*OF k*, *of a b*]
unfolding *interval-bounds*[*OF* ∗(*1*)] *interval-bounds*[*OF* ∗(*2*)] *interval-bounds*[*OF*

```
*(3)]
    unfolding *
    apply (rule subsetI)
    unfolding mem-Collect-eq split-beta
    apply (erule bexE conjE)+
    apply (simp add: )
    apply (erule exE conjE)+
  proof
    fix i l x
    assume as:
      a · fst x < snd x snd x < (if fst x = k then c else b · fst x)
      interval-lowerbound i · fst x = snd x ∨ interval-upperbound i · fst x = snd x
      i = l ∩ {x. x · k ≤ c} l ∈ d l ∩ {x. x · k ≤ c} ≠ {}
      and fstx: fst x ∈ Basis
    from assm(4)[OF this(5)] guess u v apply−by(erule exE)+ note l=this
    have *: ∀ i∈Basis. u · i ≤ (∑ i∈Basis. (if i = k then min (v · k) c else v · i)
*R i) · i
      using as(6) unfolding l interval-split[OF k] box-ne-empty as .
    have **: ∀ i∈Basis. u·i ≤ v·i
      using l using as(6) unfolding box-ne-empty[symmetric] by auto
    show ∃ i∈d. interval-lowerbound i · fst x = snd x ∨ interval-upperbound i · fst
x = snd x
      apply (rule bexI[OF - ‹l ∈ d›])
      using as(1−3,5) fstx
       unfolding l interval-bounds[OF **] interval-bounds[OF *] interval-split[OF
k] as
      apply (auto split: split-if-asm)
      done
    show snd x < b · fst x
      using as(2) ‹c < b·k› by (auto split: split-if-asm)
  qed
  show ?t2
    unfolding division-points-def interval-split[OF k, of a b]
   unfolding interval-bounds[OF *(1)] interval-bounds[OF *(2)] interval-bounds[OF
*(3)]
    unfolding *
    unfolding subset-eq
    apply rule
    unfolding mem-Collect-eq split-beta
    apply (erule bexE conjE)+
    apply (simp only: mem-Collect-eq inner-setsum-left-Basis simp-thms)
    apply (erule exE conjE)+
  proof
    fix i l x
    assume as:
      (if fst x = k then c else a · fst x) < snd x snd x < b · fst x
      interval-lowerbound i · fst x = snd x ∨ interval-upperbound i · fst x = snd x
      i = l ∩ {x. c ≤ x · k} l ∈ d l ∩ {x. c ≤ x · k} ≠ {}
      and fstx: fst x ∈ Basis
```

```
    from assm(4)[OF this(5)] guess u v by (elim exE) note l=this
    have ∗: ∀ i∈Basis. (∑ i∈Basis. (if i = k then max (u · k) c else u · i) ∗R i) ·
i ≤ v · i
      using as(6) unfolding l interval-split[OF k] box-ne-empty as .
    have ∗∗: ∀ i∈Basis. u·i ≤ v·i
      using l using as(6) unfolding box-ne-empty[symmetric] by auto
    show ∃ i∈d. interval-lowerbound i · fst x = snd x ∨ interval-upperbound i · fst
x = snd x
      apply (rule bexI[OF - ‹l ∈ d›])
      using as(1−3,5) fstx
       unfolding l interval-bounds[OF ∗∗] interval-bounds[OF ∗] interval-split[OF
k] as
      apply (auto split: split-if-asm)
      done
    show a · fst x < snd x
      using as(1) ‹a·k < c› by (auto split: split-if-asm)
  qed
qed

lemma division-points-psubset:
  fixes a :: 'a::euclidean-space
  assumes d division-of (cbox a b)
      and ∀ i∈Basis. a·i < b·i  a·k < c c < b·k
      and l ∈ d
      and interval-lowerbound l·k = c ∨ interval-upperbound l·k = c
      and k: k ∈ Basis
  shows division-points (cbox a b ∩ {x. x·k ≤ c}) {l ∩ {x. x·k ≤ c} | l. l∈d ∧ l
∩ {x. x·k ≤ c} ≠ {}} ⊂
        division-points (cbox a b) d (is ?D1 ⊂ ?D)
    and division-points (cbox a b ∩ {x. x·k ≥ c}) {l ∩ {x. x·k ≥ c} | l. l∈d ∧ l
∩ {x. x·k ≥ c} ≠ {}} ⊂
        division-points (cbox a b) d (is ?D2 ⊂ ?D)
proof −
  have ab: ∀ i∈Basis. a·i ≤ b·i
    using assms(2) by (auto intro!:less-imp-le)
  guess u v using division-ofD(4)[OF assms(1,5)] by (elim exE) note l=this
  have uv: ∀ i∈Basis. u·i ≤ v·i ∀ i∈Basis. a·i ≤ u·i ∧ v·i ≤ b·i
    using division-ofD(2,2,3)[OF assms(1,5)] unfolding l box-ne-empty
    using subset-box(1)
    apply auto
    apply blast+
    done
  have ∗: interval-upperbound (cbox a b ∩ {x. x · k ≤ interval-upperbound l · k})
· k = interval-upperbound l · k
         interval-upperbound (cbox a b ∩ {x. x · k ≤ interval-lowerbound l · k}) ·
k = interval-lowerbound l · k
    unfolding l interval-split[OF k] interval-bounds[OF uv(1)]
    using uv[rule-format, of k] ab k
    by auto
```

have $\exists x.\ x \in ?D - ?D1$
using *assms*(3−)
unfolding *division-points-def interval-bounds*[*OF ab*]
apply −
apply (*erule disjE*)
apply (*rule-tac x*=(*k*,(*interval-lowerbound l*)·*k*) **in** *exI*, *force simp add*: ∗)
apply (*rule-tac x*=(*k*,(*interval-upperbound l*)·*k*) **in** *exI*, *force simp add*: ∗)
done
moreover have *?D1* ⊆ *?D*
by (*auto simp add*: *assms division-points-subset*)
ultimately show *?D1* ⊂ *?D*
by *blast*
have ∗: *interval-lowerbound* (*cbox a b* ∩ {*x*. *x* · *k* ≥ *interval-lowerbound l* · *k*}) · *k* = *interval-lowerbound l* · *k*
interval-lowerbound (*cbox a b* ∩ {*x*. *x* · *k* ≥ *interval-upperbound l* · *k*}) · *k* = *interval-upperbound l* · *k*
unfolding *l interval-split*[*OF k*] *interval-bounds*[*OF uv*(*1*)]
using *uv*[*rule-format*, *of k*] *ab k*
by *auto*
have $\exists x.\ x \in ?D - ?D2$
using *assms*(3−)
unfolding *division-points-def interval-bounds*[*OF ab*]
apply −
apply (*erule disjE*)
apply (*rule-tac x*=(*k*,(*interval-lowerbound l*)·*k*) **in** *exI*, *force simp add*: ∗)
apply (*rule-tac x*=(*k*,(*interval-upperbound l*)·*k*) **in** *exI*, *force simp add*: ∗)
done
moreover have *?D2* ⊆ *?D*
by (*auto simp add*: *assms division-points-subset*)
ultimately show *?D2* ⊂ *?D*
by *blast*
qed

35.22 Preservation by divisions and tagged divisions.

lemma *support-support*[*simp*]:*support opp f* (*support opp f s*) = *support opp f s*
unfolding *support-def* **by** *auto*

lemma *iterate-support*[*simp*]: *iterate opp* (*support opp f s*) *f* = *iterate opp s f*
unfolding *iterate-def support-support* **by** *auto*

lemma *iterate-expand-cases*:
iterate opp s f = (*if finite*(*support opp f s*) *then iterate opp* (*support opp f s*) *f else neutral opp*)
by (*simp add*: *iterate-def fold'-def*)

lemma *iterate-image*:
assumes *monoidal opp*
and *inj-on f s*

```
  shows iterate opp (f ‘ s) g = iterate opp s (g ∘ f)
proof −
  have ∗: iterate opp (f ‘ s) g = iterate opp s (g ∘ f)
    if finite s ∀ x∈s. ∀ y∈s. f x = f y ⟶ x = y for s
    using that
  proof (induct s)
    case empty
    then show ?case by simp
  next
    case insert
    with assms(1) show ?case by auto
  qed
  show ?thesis
    apply (cases finite (support opp g (f ‘ s)))
    prefer 2
      apply (metis finite-imageI iterate-expand-cases support-clauses(7))
   apply (subst (1) iterate-support[symmetric], subst (2) iterate-support[symmetric])
    unfolding support-clauses
    apply (rule ∗)
    apply (meson assms(2) finite-imageD subset-inj-on support-subset)
    apply (meson assms(2) inj-on-contraD rev-subsetD support-subset)
    done
qed



lemma iterate-nonzero-image-lemma:
  assumes monoidal opp
    and finite s g(a) = neutral opp
    and ∀ x∈s. ∀ y∈s. f x = f y ∧ x ≠ y ⟶ g(f x) = neutral opp
  shows iterate opp {f x | x. x ∈ s ∧ f x ≠ a} g = iterate opp s (g ∘ f)
proof −
  have ∗: {f x |x. x ∈ s ∧ f x ≠ a} = f ‘ {x. x ∈ s ∧ f x ≠ a}
    by auto
  have ∗∗: support opp (g ∘ f) {x ∈ s. f x ≠ a} = support opp (g ∘ f) s
    unfolding support-def using assms(3) by auto
  have inj: inj-on f (support opp (g ∘ f) {x ∈ s. f x ≠ a})
    apply (simp add: inj-on-def)
   apply (metis (mono-tags, lifting) assms(4) comp-def mem-Collect-eq support-def)
    done
  show ?thesis
    apply (subst iterate-support[symmetric])
    apply (simp add: ∗ support-clauses iterate-image[OF assms(1) inj])
    apply (simp add: iterate-def ∗∗)
    done
qed

lemma iterate-eq-neutral:
  assumes monoidal opp
```

```
    and ⋀x. x ∈ s ⟹ f x = neutral opp
  shows iterate opp s f = neutral opp
proof −
  have [simp]: support opp f s = {}
    unfolding support-def using assms(2) by auto
  show ?thesis
    by (subst iterate-support[symmetric]) simp
qed

lemma iterate-op:
  ⟦monoidal opp; finite s⟧
  ⟹ iterate opp s (λx. opp (f x) (g x)) = opp (iterate opp s f) (iterate opp s g)
by (erule finite-induct) (auto simp: monoidal-ac(4) monoidal-ac(5))

lemma iterate-eq:
  assumes monoidal opp
    and ⋀x. x ∈ s ⟹ f x = g x
  shows iterate opp s f = iterate opp s g
proof −
  have ∗: support opp g s = support opp f s
    unfolding support-def using assms(2) by auto
  show ?thesis
  proof (cases finite (support opp f s))
    case False
    then show ?thesis
      by (simp add: ∗ iterate-expand-cases)
  next
    case True
    def su ≡ support opp f s
    have fsu: finite su
      using True by (simp add: su-def)
    moreover
    { assume finite su su ⊆ s
      then have iterate opp su f = iterate opp su g
        by (induct su) (auto simp: assms)
    }
    ultimately have iterate opp (support opp f s) f = iterate opp (support opp g
s) g
      by (simp add: ∗ su-def support-subset)
    then show ?thesis
      by simp
  qed
qed

lemma nonempty-witness:
  assumes s ≠ {}
  obtains x where x ∈ s
  using assms by auto
```

lemma *operative-division*:
fixes *f* :: *'a*::*euclidean-space set* ⇒ *'b*
assumes *monoidal opp*
and *operative opp f*
and *d division-of* (*cbox a b*)
shows *iterate opp d f* = *f* (*cbox a b*)
proof −
def *C* ≡ *card* (*division-points* (*cbox a b*) *d*)
then show *?thesis*
using *assms*
proof (*induct C arbitrary*: *a b d rule*: *full-nat-induct*)
case (*1 a b d*)
show *?case*
proof (*cases content* (*cbox a b*) = *0*)
case *True*
show *iterate opp d f* = *f* (*cbox a b*)
unfolding *operativeD*(*1*)[*OF assms*(*2*) *True*]
proof (*rule iterate-eq-neutral*[*OF* ‹*monoidal opp*›])
fix *x*
assume *x*: *x* ∈ *d*
then show *f x* = *neutral opp*
by (*metis division-ofD*(*4*) *1*(*4*) *division-of-content-0*[*OF True*] *operativeD*(*1*)[*OF assms*(*2*)] *x*)
qed
next
case *False*
note *ab* = *this*[*unfolded content-lt-nz*[*symmetric*] *content-pos-lt-eq*]
then have *ab'*: ∀ *i*∈*Basis*. *a*•*i* ≤ *b*•*i*
by (*auto intro*!: *less-imp-le*)
show *iterate opp d f* = *f* (*cbox a b*)
proof (*cases division-points* (*cbox a b*) *d* = {})
case *True*
{ fix *u v* **and** *j* :: *'a*
assume *j*: *j* ∈ *Basis* **and** *as*: *cbox u v* ∈ *d*
then have *cbox u v* ≠ {}
using *1.prems*(*3*) **by** *blast*
then have *uv*: ∀ *i*∈*Basis*. *u*•*i* ≤ *v*•*i u*•*j* ≤ *v*•*j*
using *j* **unfolding** *box-ne-empty* **by** *auto*
have ∗: ⋀*p r Q*. ¬ *j*∈*Basis* ∨ *p* ∨ *r* ∨ (∀ *x*∈*d*. *Q x*) ⟹ *p* ∨ *r* ∨ *Q* (*cbox u v*)
using *as j* **by** *auto*
have (*j*, *u*•*j*) ∉ *division-points* (*cbox a b*) *d*
(*j*, *v*•*j*) ∉ *division-points* (*cbox a b*) *d* **using** *True* **by** *auto*
note *this*[*unfolded de-Morgan-conj division-points-def mem-Collect-eq split-conv interval-bounds*[*OF ab'*] *bex-simps*]
note ∗[*OF this*(*1*)] ∗[*OF this*(*2*)] **note** *this*[*unfolded interval-bounds*[*OF uv*(*1*)]]
moreover
have *a*•*j* ≤ *u*•*j v*•*j* ≤ *b*•*j*

using *division-ofD*(*2*,*2*,*3*)[*OF* ‹*d division-of cbox a b*› *as*]
apply (*metis j subset-box*(*1*) *uv*(*1*))
by (*metis* ‹*cbox u v* ⊆ *cbox a b*› *j subset-box*(*1*) *uv*(*1*))
ultimately have $u{\cdot}j = a{\cdot}j \wedge v{\cdot}j = a{\cdot}j \vee u{\cdot}j = b{\cdot}j \wedge v{\cdot}j = b{\cdot}j \vee u{\cdot}j = a{\cdot}j \wedge v{\cdot}j = b{\cdot}j$
unfolding *not-less de-Morgan-disj* **using** *ab*[*rule-format*,*of j*] *uv*(*2*) *j*
by *force* **}**
then have d': $\forall i{\in}d.\ \exists u\ v.\ i = cbox\ u\ v \wedge$
$(\forall j{\in}Basis.\ u{\cdot}j = a{\cdot}j \wedge v{\cdot}j = a{\cdot}j \vee u{\cdot}j = b{\cdot}j \wedge v{\cdot}j = b{\cdot}j \vee u{\cdot}j = a{\cdot}j \wedge v{\cdot}j = b{\cdot}j)$
unfolding *forall-in-division*[*OF 1*(*4*)]
by *blast*
have (*1*/*2*) $*_R$ (*a*+*b*) ∈ *cbox a b*
unfolding *mem-box* **using** *ab* **by**(*auto intro*!: *less-imp-le simp*: *inner-simps*)
note *this*[*unfolded division-ofD*(*6*)[*OF* ‹*d division-of cbox a b*›,*symmetric*] *Union-iff*]
then guess *i* **.. note** *i*=*this*
guess *u v* **using** d'[*rule-format*,*OF i*(*1*)] **by** (*elim exE conjE*) **note** *uv*=*this*
have *cbox a b* ∈ *d*
proof −
have *u* = *a v* = *b*
unfolding *euclidean-eq-iff*[**where** $'a{=}'a$]
proof *safe*
fix *j* :: $'a$
assume *j*: *j* ∈ *Basis*
note *i*(*2*)[*unfolded uv mem-box*,*rule-format*,*of j*]
then show *u* · *j* = *a* · *j* **and** *v* · *j* = *b* · *j*
using *uv*(*2*)[*rule-format*,*of j*] *j* **by** (*auto simp*: *inner-simps*)
qed
then have *i* = *cbox a b* **using** *uv* **by** *auto*
then show *?thesis* **using** *i* **by** *auto*
qed
then have *deq*: *d* = *insert* (*cbox a b*) (*d* − {*cbox a b*})
by *auto*
have *iterate opp* (*d* − {*cbox a b*}) *f* = *neutral opp*
proof (*rule iterate-eq-neutral*[*OF 1*(*2*)])
fix *x*
assume *x*: *x* ∈ *d* − {*cbox a b*}
then have *x*∈*d*
by *auto* **note** d'[*rule-format*,*OF this*]
then guess *u v* **by** (*elim exE conjE*) **note** *uv*=*this*
have $u \neq a \vee v \neq b$
using *x*[*unfolded uv*] **by** *auto*
then obtain *j* **where** $u{\cdot}j \neq a{\cdot}j \vee v{\cdot}j \neq b{\cdot}j$ **and** *j*: *j* ∈ *Basis*
unfolding *euclidean-eq-iff*[**where** $'a{=}'a$] **by** *auto*
then have $u{\cdot}j = v{\cdot}j$
using *uv*(*2*)[*rule-format*,*OF j*] **by** *auto*
then have *content* (*cbox u v*) = *0*
unfolding *content-eq-0* **using** *j*

by *force*
then show $f\ x = \mathit{neutral}\ \mathit{opp}$
unfolding *uv(1)* **by** (*rule operativeD(1)*[*OF 1(3)*])
qed
then show $\mathit{iterate}\ \mathit{opp}\ d\ f = f\ (\mathit{cbox}\ a\ b)$
apply (*subst deq*)
apply (*subst iterate-insert*[*OF 1(2)*])
using *1*
apply *auto*
done
next
case *False*
then have $\exists x.\ x \in \mathit{division\text{-}points}\ (\mathit{cbox}\ a\ b)\ d$
by *auto*
then guess $k\ c$
unfolding *split-paired-Ex division-points-def mem-Collect-eq split-conv*
apply (*elim exE conjE*)
done
note *this(2−4,1)* **note** *kc=this*[*unfolded interval-bounds*[*OF ab'*]]
from *this(3)* **guess** j **.. note** *j=this*
def $d1 \equiv \{l \cap \{x.\ x{\cdot}k \le c\} \mid l.\ l \in d \wedge l \cap \{x.\ x{\cdot}k \le c\} \neq \{\}\}$
def $d2 \equiv \{l \cap \{x.\ x{\cdot}k \ge c\} \mid l.\ l \in d \wedge l \cap \{x.\ x{\cdot}k \ge c\} \neq \{\}\}$
def $cb \equiv (\sum i{\in}\mathit{Basis}.\ (\mathit{if}\ i = k\ \mathit{then}\ c\ \mathit{else}\ b{\cdot}i) *_R i)::'a$
def $ca \equiv (\sum i{\in}\mathit{Basis}.\ (\mathit{if}\ i = k\ \mathit{then}\ c\ \mathit{else}\ a{\cdot}i) *_R i)::'a$
note *division-points-psubset*[*OF ‹d division-of cbox a b› ab kc(1−2) j*]
note *psubset-card-mono*[*OF - this(1)*] *psubset-card-mono*[*OF - this(2)*]
then have $*$: $(\mathit{iterate}\ \mathit{opp}\ d1\ f) = f\ (\mathit{cbox}\ a\ b \cap \{x.\ x{\cdot}k \le c\})$
$(\mathit{iterate}\ \mathit{opp}\ d2\ f) = f\ (\mathit{cbox}\ a\ b \cap \{x.\ x{\cdot}k \ge c\})$
unfolding *interval-split*[*OF kc(4)*]
apply (*rule-tac*[!] *1.hyps*[*rule-format*])
using *division-split*[*OF ‹d division-of cbox a b›*, **where** $k{=}k$ **and** $c{=}c$]
apply (*simp-all add*: *interval-split 1 kc d1-def d2-def division-points-finite*[*OF ‹d division-of cbox a b›*])
done
{ fix $l\ y$
assume *as*: $l \in d\ y \in d\ l \cap \{x.\ x \cdot k \le c\} = y \cap \{x.\ x \cdot k \le c\}\ l \neq y$
from *division-ofD(4)*[*OF ‹d division-of cbox a b› this(1)*] **guess** $u\ v$ **by** (*elim exE*) **note** *leq=this*
have $f\ (l \cap \{x.\ x \cdot k \le c\}) = \mathit{neutral}\ \mathit{opp}$
unfolding *leq interval-split*[*OF kc(4)*]
apply (*rule operativeD(1) 1*)+
unfolding *interval-split*[*symmetric*,*OF kc(4)*]
using *division-split-left-inj 1 as kc leq* **by** *blast*
} note *fxk-le = this*
{ fix $l\ y$
assume *as*: $l \in d\ y \in d\ l \cap \{x.\ c \le x \cdot k\} = y \cap \{x.\ c \le x \cdot k\}\ l \neq y$
from *division-ofD(4)*[*OF ‹d division-of cbox a b› this(1)*] **guess** $u\ v$ **by** (*elim exE*) **note** *leq=this*
have $f\ (l \cap \{x.\ x \cdot k \ge c\}) = \mathit{neutral}\ \mathit{opp}$

```
          unfolding leq interval-split[OF kc(4)]
          apply (rule operativeD(1) 1)+
          unfolding interval-split[symmetric,OF kc(4)]
          using division-split-right-inj 1 leq as kc by blast
        } note fxk-ge = this
        have f (cbox a b) = opp (iterate opp d1 f) (iterate opp d2 f) (is - = ?prev)
          unfolding *
          using assms(2) kc(4) by blast
        also have iterate opp d1 f = iterate opp d (λl. f(l ∩ {x. x•k ≤ c}))
          unfolding d1-def empty-as-interval
          apply (rule iterate-nonzero-image-lemma[unfolded o-def])
          apply (rule 1 division-of-finite operativeD[OF 1(3)])+
          apply (force simp add: empty-as-interval[symmetric] fxk-le)+
          done
        also have iterate opp d2 f = iterate opp d (λl. f(l ∩ {x. x•k ≥ c}))
          unfolding d2-def empty-as-interval
          apply (rule iterate-nonzero-image-lemma[unfolded o-def])
          apply (rule 1 division-of-finite operativeD[OF 1(3)])+
          apply (force simp add: empty-as-interval[symmetric] fxk-ge)+
          done
        also have *: ∀x∈d. f x = opp (f (x ∩ {x. x • k ≤ c})) (f (x ∩ {x. c ≤ x
• k}))
          unfolding forall-in-division[OF ‹d division-of cbox a b›]
          using assms(2) kc(4) by blast
        have opp (iterate opp d (λl. f (l ∩ {x. x • k ≤ c}))) (iterate opp d (λl. f
(l ∩ {x. c ≤ x • k}))) =
          iterate opp d f
          apply (subst(3) iterate-eq[OF - *[rule-format]])
          using 1
          apply (auto simp: iterate-op[symmetric])
          done
        finally show ?thesis by auto
      qed
    qed
  qed
qed

lemma iterate-image-nonzero:
  assumes monoidal opp
      and finite s
      and ⋀x y. ∀x∈s. ∀y∈s. x ≠ y ∧ f x = f y ⟶ g (f x) = neutral opp
    shows iterate opp (f ‘ s) g = iterate opp s (g ∘ f)
using assms
by (induct rule: finite-subset-induct[OF assms(2) subset-refl]) auto

lemma operative-tagged-division:
  assumes monoidal opp
      and operative opp f
      and d tagged-division-of (cbox a b)
```

shows *iterate opp d* (λ(*x*,*l*). *f l*) = *f* (*cbox a b*)
proof −
have ∗: (λ(*x*,*l*). *f l*) = *f* ∘ *snd*
unfolding *o-def* **by** *rule auto* **note** *tagged* = *tagged-division-ofD*[*OF assms*(*3*)]
{ fix *a b a′*
assume *as*: (*a*, *b*) ∈ *d* (*a′*, *b*) ∈ *d* (*a*, *b*) ≠ (*a′*, *b*)
have *f b* = *neutral opp*
using *tagged*(*4*)[*OF as*(*1*)]
apply *clarify*
apply (*rule operativeD*(*1*)[*OF assms*(*2*)])
by (*metis content-eq-0-interior inf.idem tagged-division-ofD*(*5*)[*OF assms*(*3*) *as*(*1*−*3*)])
}
then have *iterate opp d* (λ(*x*,*l*). *f l*) = *iterate opp* (*snd* ‘ *d*) *f*
unfolding ∗
by (*force intro*!: *assms iterate-image-nonzero*[*symmetric*, *OF - tagged-division-of-finite*])
also have ... = *f* (*cbox a b*)
using *operative-division*[*OF assms*(*1*−*2*) *division-of-tagged-division*[*OF assms*(*3*)]]
.
finally show *?thesis* .
qed

35.23 Additivity of content.

lemma *setsum-iterate*:
assumes *finite s*
shows *setsum f s* = *iterate op* + *s f*
proof −
have *setsum f s* = *setsum f* (*support op* + *f s*)
using *assms*
by (*auto simp*: *support-def intro*: *setsum.mono-neutral-right*)
then show *?thesis* **unfolding** *iterate-def fold′-def setsum.eq-fold*
by (*simp add*: *comp-def*)
qed

lemma *additive-content-division*:
d division-of (*cbox a b*) ⟹ *setsum content d* = *content* (*cbox a b*)
by (*metis division-ofD*(*1*) *monoidal-monoid operative-content operative-division setsum-iterate*)

lemma *additive-content-tagged-division*:
d tagged-division-of (*cbox a b*) ⟹ *setsum* (λ(*x*,*l*). *content l*) *d* = *content* (*cbox a b*)
unfolding *operative-tagged-division*[*OF monoidal-monoid operative-content assms*,*symmetric*]
using *setsum-iterate* **by** *blast*

35.24 Finally, the integral of a constant

lemma *has-integral-const* [*intro*]:
fixes *a b* :: *′a*::*euclidean-space*

shows $((\lambda x.\ c)$ *has-integral* $(content\ (cbox\ a\ b) *_R\ c))\ (cbox\ a\ b)$
apply (*auto intro*!: *exI* [**where** $x{=}\lambda x.$ *ball x 1*] *simp*: *split-def has-integral*)
apply (*subst scaleR-left.setsum*[*symmetric*, *unfolded o-def*])
apply (*subst additive-content-tagged-division*[*unfolded split-def*])
apply *auto*
done

lemma *has-integral-const-real* [*intro*]:
fixes *a b* :: *real*
shows $((\lambda x.\ c)$ *has-integral* $(content\ \{a\ ..\ b\} *_R\ c))\ \{a\ ..\ b\}$
by (*metis box-real*(*2*) *has-integral-const*)

lemma *integral-const* [*simp*]:
fixes *a b* :: $'a{::}euclidean\text{-}space$
shows *integral* $(cbox\ a\ b)\ (\lambda x.\ c) = content\ (cbox\ a\ b) *_R\ c$
by (*rule integral-unique*) (*rule has-integral-const*)

lemma *integral-const-real* [*simp*]:
fixes *a b* :: *real*
shows *integral* $\{a\ ..\ b\}\ (\lambda x.\ c) = content\ \{a\ ..\ b\} *_R\ c$
by (*metis box-real*(*2*) *integral-const*)

35.25 Bounds on the norm of Riemann sums and the integral itself.

lemma *dsum-bound*:
assumes *p division-of* $(cbox\ a\ b)$
and $norm\ c \le e$
shows $norm\ (setsum\ (\lambda l.\ content\ l *_R\ c)\ p) \le e * content(cbox\ a\ b)$
proof −
have *sumeq*: $(\sum i{\in}p.\ |content\ i|) = setsum\ content\ p$
apply (*rule setsum.cong*)
using *assms*
apply *simp*
apply (*metis abs-of-nonneg assms*(*1*) *content-pos-le division-ofD*(*4*))
done
have *e*: $0 \le e$
using *assms*(*2*) *norm-ge-zero order-trans* **by** *blast*
have $norm\ (setsum\ (\lambda l.\ content\ l *_R\ c)\ p) \le (\sum i{\in}p.\ norm\ (content\ i *_R\ c))$
using *norm-setsum* **by** *blast*
also have ... $\le e * (\sum i{\in}p.\ |content\ i|)$
apply (*simp add*: *setsum-right-distrib*[*symmetric*] *mult.commute*)
using *assms*(*2*) *mult-right-mono* **by** *blast*
also have ... $\le e * content\ (cbox\ a\ b)$
apply (*rule mult-left-mono* [*OF* - *e*])
apply (*simp add*: *sumeq*)
using *additive-content-division assms*(*1*) *eq-iff* **apply** *blast*
done
finally show *?thesis* **.**

qed

lemma *rsum-bound*:
assumes *p*: *p tagged-division-of* (*cbox a b*)
and ∀ *x*∈*cbox a b*. *norm* (*f x*) ≤ *e*
shows *norm* (*setsum* (λ(*x*,*k*). *content k* ∗$_R$ *f x*) *p*) ≤ *e* ∗ *content* (*cbox a b*)
proof (*cases cbox a b* = {})
case *True* **show** *?thesis*
using *p* **unfolding** *True tagged-division-of-trivial* **by** *auto*
next
case *False*
then have *e*: *e* ≥ *0*
by (*metis assms*(*2*) *norm-ge-zero order-trans nonempty-witness*)
have *setsum-le*: *setsum* (*content* ∘ *snd*) *p* ≤ *content* (*cbox a b*)
unfolding *additive-content-tagged-division*[*OF p*, *symmetric*] *split-def*
by (*auto intro*: *eq-refl*)
have *con*: ⋀*xk*. *xk* ∈ *p* ⟹ *0* ≤ *content* (*snd xk*)
using *tagged-division-ofD*(*4*) [*OF p*] *content-pos-le*
by *force*
have *norm*: ⋀*xk*. *xk* ∈ *p* ⟹ *norm* (*f* (*fst xk*)) ≤ *e*
unfolding *fst-conv* **using** *tagged-division-ofD*(*2*,*3*)[*OF p*] *assms*
by (*metis prod.collapse subset-eq*)
have *norm* (*setsum* (λ(*x*,*k*). *content k* ∗$_R$ *f x*) *p*) ≤ (∑ *i*∈*p*. *norm* (*case i of* (*x*, *k*) ⇒ *content k* ∗$_R$ *f x*))
by (*rule norm-setsum*)
also have ... ≤ *e* ∗ *content* (*cbox a b*)
unfolding *split-def norm-scaleR*
apply (*rule order-trans*[*OF setsum-mono*])
apply (*rule mult-left-mono*[*OF - abs-ge-zero*, *of - e*])
apply (*metis norm*)
unfolding *setsum-left-distrib*[*symmetric*]
using *con setsum-le*
apply (*auto simp*: *mult.commute intro*: *mult-left-mono* [*OF - e*])
done
finally show *?thesis* **.**
qed

lemma *rsum-diff-bound*:
assumes *p tagged-division-of* (*cbox a b*)
and ∀ *x*∈*cbox a b*. *norm* (*f x* − *g x*) ≤ *e*
shows *norm* (*setsum* (λ(*x*,*k*). *content k* ∗$_R$ *f x*) *p* − *setsum* (λ(*x*,*k*). *content k* ∗$_R$ *g x*) *p*) ≤
e ∗ *content* (*cbox a b*)
apply (*rule order-trans*[*OF - rsum-bound*[*OF assms*]])
apply (*simp add*: *split-def scaleR-diff-right setsum-subtractf eq-refl*)
done

lemma *has-integral-bound*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*real-normed-vector*

assumes $0 \leq B$
and $(f \text{ has-integral } i)\ (cbox\ a\ b)$
and $\forall x{\in}cbox\ a\ b.\ norm\ (f\ x) \leq B$
shows $norm\ i \leq B * content\ (cbox\ a\ b)$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then have $*$: $norm\ i - B * content\ (cbox\ a\ b) > 0$
by *auto*
from *assms(2)*[*unfolded has-integral*,*rule-format*,*OF* $*$]
guess d **by** (*elim exE conjE*) **note** d=*this*[*rule-format*]
from *fine-division-exists*[*OF this(1)*, *of a b*] **guess** p **. note** p=*this*
have $*$: $\bigwedge s\ B.\ norm\ s \leq B \Longrightarrow \neg\ norm\ (s - i) < norm\ i - B$
unfolding *not-less*
by (*metis norm-triangle-sub*[*of i*] *add.commute le-less-trans less-diff-eq linorder-not-le norm-minus-commute*)
show *False*
using *d(2)*[*OF conjI*[*OF p*]] $*$[*OF rsum-bound*[*OF p(1) assms(3)*]] **by** *auto*
qed

corollary *has-integral-bound-real*:
fixes f :: *real* $\Rightarrow$ *'b*::*real-normed-vector*
assumes $0 \leq B$
and $(f \text{ has-integral } i)\ \{a\ ..\ b\}$
and $\forall x{\in}\{a\ ..\ b\}.\ norm\ (f\ x) \leq B$
shows $norm\ i \leq B * content\ \{a\ ..\ b\}$
by (*metis assms box-real(2) has-integral-bound*)

corollary *integrable-bound*:
fixes f :: *'a*::*euclidean-space* $\Rightarrow$ *'b*::*real-normed-vector*
assumes $0 \leq B$
and f *integrable-on* $(cbox\ a\ b)$
and $\bigwedge x.\ x{\in}cbox\ a\ b \Longrightarrow norm\ (f\ x) \leq B$
shows $norm\ (integral\ (cbox\ a\ b)\ f) \leq B * content\ (cbox\ a\ b)$
by (*metis integrable-integral has-integral-bound assms*)

35.26 Similar theorems about relationship among components.

lemma *rsum-component-le*:
fixes f :: *'a*::*euclidean-space* $\Rightarrow$ *'b*::*euclidean-space*
assumes p *tagged-division-of* $(cbox\ a\ b)$
and $\forall x{\in}cbox\ a\ b.\ (f\ x){\cdot}i \leq (g\ x){\cdot}i$
shows $(setsum\ (\lambda(x,k).\ content\ k *_R f\ x)\ p){\cdot}i \leq (setsum\ (\lambda(x,k).\ content\ k *_R g\ x)\ p){\cdot}i$
unfolding *inner-setsum-left*
proof (*rule setsum-mono*, *clarify*)
fix $a\ b$
assume *ab*: $(a, b) \in p$
note *tagged* = *tagged-division-ofD(2–4)*[*OF assms(1) ab*]

```
  from this(3) guess u v by (elim exE) note b=this
  show (content b *R f a) · i ≤ (content b *R g a) · i
    unfolding b inner-simps real-scaleR-def
    apply (rule mult-left-mono)
    using assms(2) tagged
    by (auto simp add: content-pos-le)
qed

lemma has-integral-component-le:
  fixes f g :: 'a::euclidean-space ⇒ 'b::euclidean-space
  assumes k: k ∈ Basis
  assumes (f has-integral i) s (g has-integral j) s
    and ∀x∈s. (f x)·k ≤ (g x)·k
  shows i·k ≤ j·k
proof −
  have lem: i·k ≤ j·k
    if f-i: (f has-integral i) (cbox a b)
    and g-j: (g has-integral j) (cbox a b)
    and le: ∀x∈cbox a b. (f x)·k ≤ (g x)·k
    for a b i and j :: 'b and f g :: 'a ⇒ 'b
  proof (rule ccontr)
    assume ¬ ?thesis
    then have *: 0 < (i·k − j·k) / 3
      by auto
   guess d1 using f-i[unfolded has-integral,rule-format,OF *] by (elim exE conjE)
note d1=this[rule-format]
   guess d2 using g-j[unfolded has-integral,rule-format,OF *] by (elim exE conjE)
note d2=this[rule-format]
    obtain p where p: p tagged-division-of cbox a b d1 fine p d2 fine p
      using fine-division-exists[OF gauge-inter[OF d1(1) d2(1)], of a b] unfolding
fine-inter
       by metis
    note le-less-trans[OF Basis-le-norm[OF k]]
    then have |((∑(x, k)∈p. content k *R f x) − i) · k| < (i · k − j · k) / 3
              |((∑(x, k)∈p. content k *R g x) − j) · k| < (i · k − j · k) / 3
      using  k norm-bound-Basis-lt d1 d2 p
      by blast+
    then show False
      unfolding inner-simps
      using rsum-component-le[OF p(1) le]
      by (simp add: abs-real-def split: split-if-asm)
  qed
  show ?thesis
  proof (cases ∃a b. s = cbox a b)
    case True
    with lem assms show ?thesis
      by auto
  next
    case False
```

```
    show ?thesis
    proof (rule ccontr)
      assume ¬ i·k ≤ j·k
      then have ij: (i·k − j·k) / 3 > 0
        by auto
      note has-integral-altD[OF - False this]
    from this[OF assms(2)] this[OF assms(3)] guess B1 B2 . note B=this[rule-format]
      have bounded (ball 0 B1 ∪ ball (0::'a) B2)
        unfolding bounded-Un by(rule conjI bounded-ball)+
      from bounded-subset-cbox[OF this] guess a b by (elim exE)
      note ab = conjunctD2[OF this[unfolded Un-subset-iff]]
      guess w1 using B(2)[OF ab(1)] .. note w1=conjunctD2[OF this]
      guess w2 using B(4)[OF ab(2)] .. note w2=conjunctD2[OF this]
      have ∗: ⋀w1 w2 j i::real .|w1 − i| < (i − j) / 3 ⟹ |w2 − j| < (i − j) / 3
⟹ w1 ≤ w2 ⟹ False
        by (simp add: abs-real-def split: split-if-asm)
      note le-less-trans[OF Basis-le-norm[OF k]]
      note this[OF w1(2)] this[OF w2(2)]
      moreover
      have w1·k ≤ w2·k
        by (rule lem[OF w1(1) w2(1)]) (simp add: assms(4))
      ultimately show False
        unfolding inner-simps by(rule ∗)
    qed
  qed
qed

lemma integral-component-le:
  fixes g f :: 'a::euclidean-space ⇒ 'b::euclidean-space
  assumes k ∈ Basis
    and f integrable-on s g integrable-on s
    and ∀x∈s. (f x)·k ≤ (g x)·k
  shows (integral s f)·k ≤ (integral s g)·k
  apply (rule has-integral-component-le)
  using integrable-integral assms
  apply auto
  done

lemma has-integral-component-nonneg:
  fixes f :: 'a::euclidean-space ⇒ 'b::euclidean-space
  assumes k ∈ Basis
    and (f has-integral i) s
    and ∀x∈s. 0 ≤ (f x)·k
  shows 0 ≤ i·k
  using has-integral-component-le[OF assms(1) has-integral-0 assms(2)]
  using assms(3−)
  by auto

lemma integral-component-nonneg:
```

fixes $f :: 'a::euclidean\text{-}space \Rightarrow 'b::euclidean\text{-}space$
assumes $k \in Basis$
and f *integrable-on* s $\forall x{\in}s.\ 0 \leq (f\ x){\cdot}k$
shows $0 \leq (integral\ s\ f){\cdot}k$
apply (*rule has-integral-component-nonneg*)
using *assms*
apply *auto*
done

lemma *has-integral-component-neg*:
fixes $f :: 'a::euclidean\text{-}space \Rightarrow 'b::euclidean\text{-}space$
assumes $k \in Basis$
and (f *has-integral* i) s
and $\forall x{\in}s.\ (f\ x){\cdot}k \leq 0$
shows $i{\cdot}k \leq 0$
using *has-integral-component-le*[*OF assms*(*1*,*2*) *has-integral-0*] *assms*(*2*−)
by *auto*

lemma *has-integral-component-lbound*:
fixes $f :: 'a::euclidean\text{-}space \Rightarrow 'b::euclidean\text{-}space$
assumes (f *has-integral* i) (*cbox a b*)
and $\forall x{\in}cbox\ a\ b.\ B \leq f(x){\cdot}k$
and $k \in Basis$
shows $B * content\ (cbox\ a\ b) \leq i{\cdot}k$
using *has-integral-component-le*[*OF assms*(*3*) *has-integral-const assms*(*1*),*of* ($\sum i{\in}Basis.\ B *_R i$)::'*b*] *assms*(*2*−)
by (*auto simp add*: *field-simps*)

lemma *has-integral-component-ubound*:
fixes $f::'a::euclidean\text{-}space => 'b::euclidean\text{-}space$
assumes (f *has-integral* i) (*cbox a b*)
and $\forall x{\in}cbox\ a\ b.\ f\ x{\cdot}k \leq B$
and $k \in Basis$
shows $i{\cdot}k \leq B * content\ (cbox\ a\ b)$
using *has-integral-component-le*[*OF assms*(*3*,*1*) *has-integral-const*, *of* $\sum i{\in}Basis.\ B *_R i$] *assms*(*2*−)
by (*auto simp add*: *field-simps*)

lemma *integral-component-lbound*:
fixes $f :: 'a::euclidean\text{-}space \Rightarrow 'b::euclidean\text{-}space$
assumes f *integrable-on cbox a b*
and $\forall x{\in}cbox\ a\ b.\ B \leq f(x){\cdot}k$
and $k \in Basis$
shows $B * content\ (cbox\ a\ b) \leq (integral(cbox\ a\ b)\ f){\cdot}k$
apply (*rule has-integral-component-lbound*)
using *assms*
unfolding *has-integral-integral*
apply *auto*
done

lemma *integral-component-lbound-real*:
assumes *f integrable-on* {*a* ::*real* .. *b*}
and ∀ *x*∈{*a* .. *b*}. *B* ≤ *f*(*x*)•*k*
and *k* ∈ *Basis*
shows *B* ∗ *content* {*a* .. *b*} ≤ (*integral* {*a* .. *b*} *f*)•*k*
using *assms*
by (*metis box-real*(2) *integral-component-lbound*)

lemma *integral-component-ubound*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *f integrable-on cbox a b*
and ∀ *x*∈*cbox a b*. *f x*•*k* ≤ *B*
and *k* ∈ *Basis*
shows (*integral* (*cbox a b*) *f*)•*k* ≤ *B* ∗ *content* (*cbox a b*)
apply (*rule has-integral-component-ubound*)
using *assms*
unfolding *has-integral-integral*
apply *auto*
done

lemma *integral-component-ubound-real*:
fixes *f* :: *real* ⇒ *'a*::*euclidean-space*
assumes *f integrable-on* {*a* .. *b*}
and ∀ *x*∈{*a* .. *b*}. *f x*•*k* ≤ *B*
and *k* ∈ *Basis*
shows (*integral* {*a* .. *b*} *f*)•*k* ≤ *B* ∗ *content* {*a* .. *b*}
using *assms*
by (*metis box-real*(2) *integral-component-ubound*)

35.27 Uniform limit of integrable functions is integrable.

lemma *integrable-uniform-limit*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*banach*
assumes ∀ *e*>0. ∃ *g*. (∀ *x*∈*cbox a b*. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable-on* *cbox a b*
shows *f integrable-on cbox a b*
proof (*cases content* (*cbox a b*) > 0)
case *False* **then show** *?thesis*
using *has-integral-null*
by (*simp add*: *content-lt-nz integrable-on-def*)
next
case *True*
have ∗: ⋀*P*. ∀ *e*>(0::*real*). *P e* ⟹ ∀ *n*::*nat*. *P* (*inverse* (*real n* + 1))
by *auto*
from *choice*[*OF* ∗[*OF assms*]] **guess** *g* **.. note** *g*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]
from *choice*[*OF allI*[*OF g*(2)[*unfolded integrable-on-def*], *of* λ*x*. *x*]]
obtain *i* **where** *i*: ⋀*x*. (*g x has-integral i x*) (*cbox a b*)
by *auto*

```
  have Cauchy i
    unfolding Cauchy-def
  proof clarify
    fix e :: real
    assume e>0
    then have e / 4 / content (cbox a b) > 0
      using True by (auto simp add: field-simps)
    then obtain M :: nat
        where M: M ≠ 0 0 < inverse (real-of-nat M) inverse (of-nat M) < e / 4
/ content (cbox a b)
      by (subst (asm) real-arch-inv) auto
    show ∃ M. ∀ m≥M. ∀ n≥M. dist (i m) (i n) < e
    proof (rule exI [where x=M], clarify)
      fix m n
      assume m: M ≤ m and n: M ≤ n
      have e/4>0 using ‹e>0› by auto
      note * = i[unfolded has-integral,rule-format,OF this]
      from *[of m] guess gm by (elim conjE exE) note gm=this[rule-format]
      from *[of n] guess gn by (elim conjE exE) note gn=this[rule-format]
      from fine-division-exists[OF gauge-inter[OF gm(1) gn(1)], of a b]
      obtain p where p: p tagged-division-of cbox a b (λx. gm x ∩ gn x) fine p
        by auto
      { fix s1 s2 i1 and i2::'b
        assume no: norm(s2 − s1) ≤ e/2 norm (s1 − i1) < e/4 norm (s2 − i2)
< e/4
        have norm (i1 − i2) ≤ norm (i1 − s1) + norm (s1 − s2) + norm (s2 −
i2)
          using norm-triangle-ineq[of i1 − s1 s1 − i2]
          using norm-triangle-ineq[of s1 − s2 s2 − i2]
          by (auto simp add: algebra-simps)
        also have ... < e
          using no
          unfolding norm-minus-commute
          by (auto simp add: algebra-simps)
        finally have norm (i1 − i2) < e .
      } note triangle3 = this
      have finep: gm fine p gn fine p
        using fine-inter p  by auto
      { fix x
        assume x: x ∈ cbox a b
        have norm (f x − g n x) + norm (f x − g m x) ≤ inverse (real n + 1) +
inverse (real m + 1)
          using g(1)[OF x, of n] g(1)[OF x, of m] by auto
        also have ... ≤ inverse (real M) + inverse (real M)
          apply (rule add-mono)
          using M(2) m n by auto
        also have ... = 2 / real M
          unfolding divide-inverse by auto
        finally have norm (g n x − g m x) ≤ 2 / real M
```

```
        using norm-triangle-le[of g n x − f x f x − g m x 2 / real M]
        by (auto simp add: algebra-simps simp add: norm-minus-commute)
    } note norm-le = this
    have le-e2: norm ((∑(x, k)∈p. content k *R g n x) − (∑(x, k)∈p. content
k *R g m x)) ≤ e / 2
      apply (rule order-trans [OF rsum-diff-bound[OF p(1), where e=2 / real
M]])
      apply (blast intro: norm-le)
      using M True
      by (auto simp add: field-simps)
    then show dist (i m) (i n) < e
      unfolding dist-norm
      using gm gn p finep
      by (auto intro!: triangle3)
  qed
 qed
 then obtain s where s: i ⟶ s
  using convergent-eq-cauchy[symmetric] by blast
 show ?thesis
  unfolding integrable-on-def has-integral
 proof (rule-tac x=s in exI, clarify)
  fix e::real
  assume e: 0 < e
  then have *: e/3 > 0 by auto
  then obtain N1 where N1: ∀ n≥N1. norm (i n − s) < e / 3
   using LIMSEQ-D [OF s] by metis
  from e True have e / 3 / content (cbox a b) > 0
   by (auto simp add: field-simps)
  from real-arch-invD[OF this] guess N2 by (elim exE conjE) note N2=this
  from i[of N1 + N2,unfolded has-integral,rule-format,OF *] guess g′ .. note
g′=conjunctD2[OF this,rule-format]
  { fix sf sg i
   assume no: norm (sf − sg) ≤ e / 3
            norm(i − s) < e / 3
            norm (sg − i) < e / 3
   have norm (sf − s) ≤ norm (sf − sg) + norm (sg − i) + norm (i − s)
    using norm-triangle-ineq[of sf − sg sg − s]
    using norm-triangle-ineq[of sg − i i − s]
    by (auto simp add: algebra-simps)
   also have ... < e
    using no
    unfolding norm-minus-commute
    by (auto simp add: algebra-simps)
   finally have norm (sf − s) < e .
  } note lem = this
  { fix p
   assume p: p tagged-division-of (cbox a b) ∧ g′ fine p
   then have norm-less: norm ((∑(x, k)∈p. content k *R g (N1 + N2) x) −
i (N1 + N2)) < e / 3
```

using *g'* **by** *blast*
have *content (cbox a b) < e / 3 * (of-nat N2)*
using *N2* **unfolding** *inverse-eq-divide* **using** *True* **by** (*auto simp add*: *field-simps*)
moreover have *e / 3 * of-nat N2 ≤ e / 3 * (of-nat (N1 + N2) + 1)*
using ‹*e>0*› **by** *auto*
ultimately have *content (cbox a b) < e / 3 * (of-nat (N1 + N2) + 1)*
by *linarith*
then have *le-e3*: *inverse (real (N1 + N2) + 1) * content (cbox a b) ≤ e / 3*
unfolding *inverse-eq-divide*
by (*auto simp add*: *field-simps*)
have *ne3*: *norm (i (N1 + N2) − s) < e / 3*
using *N1* **by** *auto*
have *norm ((∑(x, k)∈p. content k *$_R$ f x) − s) < e*
apply (*rule lem*[*OF order-trans* [*OF - le-e3*] *ne3 norm-less*])
apply (*rule rsum-diff-bound*[*OF p*[*THEN conjunct1*]])
apply (*blast intro*: *g*)
done }
then show ∃ *d*. *gauge d* ∧
(∀ *p*. *p tagged-division-of cbox a b* ∧ *d fine p* ⟶ *norm ((∑(x, k)∈p. content k *$_R$ *f x) − s) < e*)
by (*blast intro*: *g'*)
qed
qed

lemmas *integrable-uniform-limit-real* = *integrable-uniform-limit* [**where** *'a=real*, *simplified*]

35.28 Negligible sets.

definition *negligible* (*s*:: *'a*::*euclidean-space set*) ⟷
(∀ *a b*. ((*indicator s* :: *'a*⇒*real*) *has-integral 0*) (*cbox a b*))

35.29 Negligibility of hyperplane.

lemma *setsum-nonzero-image-lemma*:
assumes *finite s*
and *g a = 0*
and ∀ *x*∈*s*. ∀ *y*∈*s*. *f x = f y* ∧ *x ≠ y* ⟶ *g (f x) = 0*
shows *setsum g {f x |x. x ∈ s* ∧ *f x ≠ a} = setsum (g ∘ f) s*
apply (*subst setsum-iterate*)
using *assms monoidal-monoid*
unfolding *setsum-iterate*[*OF assms*(*1*)]
apply (*auto intro*!: *iterate-nonzero-image-lemma*)
done

lemma *interval-doublesplit*:
fixes *a* :: *'a*::*euclidean-space*
assumes *k ∈ Basis*
shows *cbox a b* ∩ {*x* . |*x*•*k* − *c*| ≤ (*e*::*real*)} =

```
    cbox (∑ i∈Basis. (if i = k then max (a•k) (c − e) else a•i) *R i)
    (∑ i∈Basis. (if i = k then min (b•k) (c + e) else b•i) *R i)
proof −
  have *: ⋀x c e::real. |x − c| ≤ e ⟷ x ≥ c − e ∧ x ≤ c + e
    by auto
  have **: ⋀s P Q. s ∩ {x. P x ∧ Q x} = (s ∩ {x. Q x}) ∩ {x. P x}
    by blast
  show ?thesis
    unfolding * ** interval-split[OF assms] by (rule refl)
qed

lemma division-doublesplit:
  fixes a :: 'a::euclidean-space
  assumes p division-of (cbox a b)
    and k: k ∈ Basis
  shows {l ∩ {x. |x•k − c| ≤ e} |l. l ∈ p ∧ l ∩ {x. |x•k − c| ≤ e} ≠ {}}
        division-of (cbox a b ∩ {x. |x•k − c| ≤ e})
proof −
  have *: ⋀x c. |x − c| ≤ e ⟷ x ≥ c − e ∧ x ≤ c + e
    by auto
  have **: ⋀p q p' q'. p division-of q ⟹ p = p' ⟹ q = q' ⟹ p' division-of q'
    by auto
  note division-split(1)[OF assms, where c=c+e,unfolded interval-split[OF k]]
  note division-split(2)[OF this, where c=c−e and k=k,OF k]
  then show ?thesis
    apply (rule **)
    unfolding interval-doublesplit [OF k]
    using k
    apply (simp-all add: * interval-split)
    apply (rule equalityI, blast)
    apply clarsimp
    apply (rule-tac x=l ∩ {x. c + e ≥ x • k} in exI)
    apply auto
    done
qed

lemma content-doublesplit:
  fixes a :: 'a::euclidean-space
  assumes 0 < e
    and k: k ∈ Basis
  obtains d where 0 < d and content (cbox a b ∩ {x. |x•k − c| ≤ d}) < e
proof (cases content (cbox a b) = 0)
  case True
  then have ce: content (cbox a b) < e
    by (metis ‹0 < e›)
  show ?thesis
    apply (rule that[of 1])
    apply simp
    unfolding interval-doublesplit[OF k]
```

```
    apply (rule le-less-trans[OF content-subset ce])
    apply (auto simp: interval-doublesplit[symmetric] k)
    done
next
  case False
  def d ≡ e / 3 / setprod (λi. b·i − a·i) (Basis − {k})
  note False[unfolded content-eq-0 not-ex not-le, rule-format]
  then have ⋀x. x ∈ Basis ⟹ b·x > a·x
    by (auto simp add:not-le)
  then have prod0: 0 < setprod (λi. b·i − a·i) (Basis − {k})
    by (force simp add: setprod-pos field-simps)
  then have d > 0
    using assms
    by (auto simp add: d-def field-simps)
  then show ?thesis
  proof (rule that[of d])
    have ∗: Basis = insert k (Basis − {k})
      using k by auto
    have less-e: (min (b · k) (c + d) − max (a · k) (c − d)) ∗ (∏ i∈Basis − {k}.
b · i − a · i) < e
    proof −
      have (min (b · k) (c + d) − max (a · k) (c − d)) ≤ 2 ∗ d
        by auto
      also have ... < e / (∏ i∈Basis − {k}. b · i − a · i)
        unfolding d-def
        using assms prod0
        by (auto simp add: field-simps)
      finally show (min (b · k) (c + d) − max (a · k) (c − d)) ∗ (∏ i∈Basis −
{k}. b · i − a · i) < e
        unfolding pos-less-divide-eq[OF prod0] .
    qed
    show content (cbox a b ∩ {x. |x · k − c| ≤ d}) < e
    proof (cases cbox a b ∩ {x. |x · k − c| ≤ d} = {})
      case True
      then show ?thesis
        using assms by simp
    next
      case False
      then have
          (∏ i∈Basis − {k}. interval-upperbound (cbox a b ∩ {x. |x · k − c| ≤ d})
· i −
                interval-lowerbound (cbox a b ∩ {x. |x · k − c| ≤ d}) · i)
        = (∏ i∈Basis − {k}. b·i − a·i)
        by (simp add: box-eq-empty interval-doublesplit[OF k])
      then show content (cbox a b ∩ {x. |x · k − c| ≤ d}) < e
        unfolding content-def
        using assms False
        apply (subst ∗)
        apply (subst setprod.insert)
```

```
      apply (simp-all add: interval-doublesplit[OF k] box-eq-empty not-less less-e)
       done
    qed
  qed
qed

lemma negligible-standard-hyperplane[intro]:
  fixes k :: 'a::euclidean-space
  assumes k: k ∈ Basis
  shows negligible {x. x•k = c}
  unfolding negligible-def has-integral
proof (clarify, goal-cases)
  case (1 a b e)
  from this and k obtain d where d: 0 < d content (cbox a b ∩ {x. |x • k − c|
≤ d}) < e
    by (rule content-doublesplit)
  let ?i = indicator {x::'a. x•k = c} :: 'a⇒real
  show ?case
    apply (rule-tac x=λx. ball x d in exI)
    apply rule
    apply (rule gauge-ball)
    apply (rule d)
  proof (rule, rule)
    fix p
    assume p: p tagged-division-of (cbox a b) ∧ (λx. ball x d) fine p
    have ∗: (∑(x, ka)∈p. content ka ∗R ?i x) =
      (∑(x, ka)∈p. content (ka ∩ {x. |x•k − c| ≤ d}) ∗R ?i x)
      apply (rule setsum.cong)
      apply (rule refl)
      unfolding split-paired-all real-scaleR-def mult-cancel-right split-conv
      apply cases
      apply (rule disjI1)
      apply assumption
      apply (rule disjI2)
    proof −
      fix x l
      assume as: (x, l) ∈ p ?i x ≠ 0
      then have xk: x•k = c
        unfolding indicator-def
        apply −
        apply (rule ccontr)
        apply auto
        done
      show content l = content (l ∩ {x. |x • k − c| ≤ d})
        apply (rule arg-cong[where f=content])
        apply (rule set-eqI)
        apply rule
        apply rule
        unfolding mem-Collect-eq
```

```
      proof −
        fix y
        assume y: y ∈ l
          note p[THEN conjunct2,unfolded fine-def,rule-format,OF as(1),unfolded
split-conv]
        note this[unfolded subset-eq mem-ball dist-norm,rule-format,OF y]
        note le-less-trans[OF Basis-le-norm[OF k] this]
        then show |y · k − c| ≤ d
          unfolding inner-simps xk by auto
      qed auto
    qed
  note p'= tagged-division-ofD[OF p[THEN conjunct1]] and p''=division-of-tagged-division[OF
p[THEN conjunct1]]
    show norm ((∑(x, ka)∈p. content ka *R ?i x) − 0) < e
      unfolding diff-0-right *
      unfolding real-scaleR-def real-norm-def
      apply (subst abs-of-nonneg)
      apply (rule setsum-nonneg)
      apply rule
      unfolding split-paired-all split-conv
      apply (rule mult-nonneg-nonneg)
      apply (drule p'(4))
      apply (erule exE)+
      apply(rule-tac b=b in back-subst)
      prefer 2
      apply (subst(asm) eq-commute)
      apply assumption
      apply (subst interval-doublesplit[OF k])
      apply (rule content-pos-le)
      apply (rule indicator-pos-le)
    proof −
      have (∑(x, ka)∈p. content (ka ∩ {x. |x · k − c| ≤ d}) * ?i x) ≤
        (∑(x, ka)∈p. content (ka ∩ {x. |x · k − c| ≤ d}))
        apply (rule setsum-mono)
        unfolding split-paired-all split-conv
        apply (rule mult-right-le-one-le)
        apply (drule p'(4))
        apply (auto simp add:interval-doublesplit[OF k])
        done
      also have ... < e
        proof (subst setsum-over-tagged-division-lemma[OF p[THEN conjunct1]],
goal-cases)
        case prems: (1 u v)
        have content (cbox u v ∩ {x. |x · k − c| ≤ d}) ≤ content (cbox u v)
          unfolding interval-doublesplit[OF k]
          apply (rule content-subset)
          unfolding interval-doublesplit[symmetric,OF k]
          apply auto
          done
```

then show *?case*
unfolding *prems interval-doublesplit*[*OF k*]
by (*blast intro*: *antisym*)
next
have ∗: *setsum content* {*l* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*} |*l*. *l* ∈ *snd* ' *p* ∧ *l* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*} ≠ {}} ≥ *0*
apply (*rule setsum-nonneg*)
apply *rule*
unfolding *mem-Collect-eq image-iff*
apply (*erule exE bexE conjE*)+
unfolding *split-paired-all*
proof −
fix *x l a b*
assume *as*: *x* = *l* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*} (*a*, *b*) ∈ *p l* = *snd* (*a*, *b*)
guess *u v* **using** *p'*(*4*)[*OF as*(*2*)] **by** (*elim exE*) **note** ∗ = *this*
show *content x* ≥ *0*
unfolding *as snd-conv* ∗ *interval-doublesplit*[*OF k*]
by (*rule content-pos-le*)
qed
have ∗∗: *norm* (*1*::*real*) ≤ *1*
by *auto*
note *division-doublesplit*[*OF p'' k*,*unfolded interval-doublesplit*[*OF k*]]
note *dsum-bound*[*OF this* ∗∗,*unfolded interval-doublesplit*[*symmetric*,*OF k*]]
note *this*[*unfolded real-scaleR-def real-norm-def mult-1-right mult-1*, *of c d*]
note *le-less-trans*[*OF this d*(*2*)]
from *this*[*unfolded abs-of-nonneg*[*OF* ∗]]
show (∑ *ka*∈*snd* ' *p*. *content* (*ka* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*})) < *e*
apply (*subst setsum-nonzero-image-lemma*[*of snd* ' *p content* {}, *unfolded o-def*,*symmetric*])
apply (*rule finite-imageI p' content-empty*)+
unfolding *forall-in-division*[*OF p''*]
proof (*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*rule*,*erule conjE*)
fix *m n u v*
assume *as*:
cbox m n ∈ *snd* ' *p cbox u v* ∈ *snd* ' *p*
cbox m n ≠ *cbox u v*
cbox m n ∩ {*x*. |*x* · *k* − *c*| ≤ *d*} = *cbox u v* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*}
have (*cbox m n* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*}) ∩ (*cbox u v* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*}) ⊆ *cbox m n* ∩ *cbox u v*
by *blast*
note *interior-mono*[*OF this*, *unfolded division-ofD*(*5*)[*OF p'' as*(*1*−*3*)] *interior-Int*[*of cbox m n*]]
then have *interior* (*cbox m n* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*}) = {}
unfolding *as Int-absorb* **by** *auto*
then show *content* (*cbox m n* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*}) = *0*
unfolding *interval-doublesplit*[*OF k*] *content-eq-0-interior*[*symmetric*] .
qed
qed
finally show (∑ (*x*, *ka*)∈*p*. *content* (*ka* ∩ {*x*. |*x* · *k* − *c*| ≤ *d*}) ∗ *?i x*) < *e*

.
  qed
 qed
qed

35.30 A technical lemma about "refinement" of division.

lemma *tagged-division-finer*:
 fixes p :: ($'a$::*euclidean-space* × ($'a$::*euclidean-space set*)) *set*
 assumes p *tagged-division-of* (*cbox a b*)
  and *gauge d*
 obtains q **where** q *tagged-division-of* (*cbox a b*)
  and d *fine* q
  and ∀(*x*,*k*) ∈ p. $k \subseteq d(x) \longrightarrow (x,k) \in q$
proof −
 let *?P* = λp. p *tagged-partial-division-of* (*cbox a b*) ⟶ *gauge d* ⟶
  (∃ q. q *tagged-division-of* (⋃{k. ∃ x. (*x*,*k*) ∈ p}) ∧ d *fine* q ∧
   (∀(*x*,*k*) ∈ p. $k \subseteq d(x) \longrightarrow (x,k) \in q$))
 {
  have ∗: *finite p p tagged-partial-division-of* (*cbox a b*)
   using *assms*(*1*)
   unfolding *tagged-division-of-def*
   by *auto*
  presume ⋀p. *finite p* ⟹ *?P p*
  from *this*[*rule-format*,*OF* ∗ *assms*(*2*)] **guess** q **.. note** q=*this*
  then show *?thesis*
   apply −
   apply (*rule that*[*of q*])
   unfolding *tagged-division-ofD*[*OF assms*(*1*)]
   apply *auto*
   done
 }
 fix p :: ($'a$::*euclidean-space* × ($'a$::*euclidean-space set*)) *set*
 assume *as*: *finite p*
 show *?P p*
  apply *rule*
  apply *rule*
  using *as*
 proof (*induct p*)
  case *empty*
  show *?case*
   apply (*rule-tac x*={} **in** *exI*)
   unfolding *fine-def*
   apply *auto*
   done
 next
  case (*insert xk p*)
  guess *x k* **using** *surj-pair*[*of xk*] **by** (*elim exE*) **note** *xk*=*this*
  note *tagged-partial-division-subset*[*OF insert*(*4*) *subset-insertI*]

from *insert(3)*[*OF this insert(5)*] **guess** *q1* **.. note** *q1 = conjunctD3*[*OF this*]
have ∗: $\bigcup\{l.\ \exists y.\ (y,l) \in$ *insert xk p*$\} = k \cup \bigcup\{l.\ \exists y.\ (y,l) \in p\}$
unfolding *xk* **by** *auto*
note *p = tagged-partial-division-ofD*[*OF insert(4)*]
from *p(4)*[*unfolded xk*, *OF insertI1*] **guess** *u v* **by** (*elim exE*) **note** *uv=this*

have *finite* $\{k.\ \exists x.\ (x,\ k) \in p\}$
apply (*rule finite-subset*[*of - snd ' p*])
using *p*
apply *safe*
apply (*metis image-iff snd-conv*)
apply *auto*
done
then have *int*: *interior* (*cbox u v*) ∩ *interior* $(\bigcup\{k.\ \exists x.\ (x,\ k) \in p\}) = \{\}$
apply (*rule inter-interior-unions-intervals*)
apply (*rule open-interior*)
apply (*rule-tac*[!] *ballI*)
unfolding *mem-Collect-eq*
apply (*erule-tac*[!] *exE*)
apply (*drule p(4)*[*OF insertI2*])
apply *assumption*
apply (*rule p(5)*)
unfolding *uv xk*
apply (*rule insertI1*)
apply (*rule insertI2*)
apply *assumption*
using *insert(2)*
unfolding *uv xk*
apply *auto*
done
show *?case*
proof (*cases cbox u v* ⊆ *d x*)
case *True*
then show *?thesis*
apply (*rule-tac x*=$\{(x,$*cbox u v*$)\} \cup$ *q1* **in** *exI*)
apply *rule*
unfolding ∗ *uv*
apply (*rule tagged-division-union*)
apply (*rule tagged-division-of-self*)
apply (*rule p*[*unfolded xk uv*] *insertI1*)+
apply (*rule q1*)
apply (*rule int*)
apply *rule*
apply (*rule fine-union*)
apply (*subst fine-def*)
defer
apply (*rule q1*)
unfolding *Ball-def split-paired-All split-conv*
apply *rule*

```
      apply rule
      apply rule
      apply rule
      apply (erule insertE)
      apply (simp add: uv xk)
      apply (rule UnI2)
      apply (drule q1(3)[rule-format])
      unfolding xk uv
      apply auto
      done
  next
    case False
    from fine-division-exists[OF assms(2), of u v] guess q2 . note q2=this
    show ?thesis
      apply (rule-tac x=q2 ∪ q1 in exI)
      apply rule
      unfolding ∗ uv
      apply (rule tagged-division-union q2 q1 int fine-union)+
      unfolding Ball-def split-paired-All split-conv
      apply rule
      apply (rule fine-union)
      apply (rule q1 q2)+
      apply rule
      apply rule
      apply rule
      apply rule
      apply (erule insertE)
      apply (rule UnI2)
      apply (simp add: False uv xk)
      apply (drule q1(3)[rule-format])
      using False
      unfolding xk uv
      apply auto
      done
  qed
 qed
qed
```

35.31 Hence the main theorem about negligible sets.

```
lemma finite-product-dependent:
 assumes finite s
   and ⋀x. x ∈ s ⟹ finite (t x)
 shows finite {(i, j) |i j. i ∈ s ∧ j ∈ t i}
 using assms
proof induct
 case (insert x s)
 have ∗: {(i, j) |i j. i ∈ insert x s ∧ j ∈ t i} =
   (λy. (x,y)) ‘ (t x) ∪ {(i, j) |i j. i ∈ s ∧ j ∈ t i} by auto
```

```
  show ?case
    unfolding *
    apply (rule finite-UnI)
    using insert
    apply auto
    done
qed auto

lemma sum-sum-product:
  assumes finite s
    and ∀ i∈s. finite (t i)
  shows setsum (λi. setsum (x i) (t i)::real) s =
    setsum (λ(i,j). x i j) {(i,j) | i j. i ∈ s ∧ j ∈ t i}
  using assms
proof induct
  case (insert a s)
  have *: {(i, j) |i j. i ∈ insert a s ∧ j ∈ t i} =
    (λy. (a,y)) ` (t a) ∪ {(i, j) |i j. i ∈ s ∧ j ∈ t i} by auto
  show ?case
    unfolding *
    apply (subst setsum.union-disjoint)
    unfolding setsum.insert[OF insert(1-2)]
    prefer 4
    apply (subst insert(3))
    unfolding add-right-cancel
  proof -
    show setsum (x a) (t a) = (∑(xa, y)∈ Pair a ` t a. x xa y)
      apply (subst setsum.reindex)
      unfolding inj-on-def
      apply auto
      done
    show finite {(i, j) |i j. i ∈ s ∧ j ∈ t i}
      apply (rule finite-product-dependent)
      using insert
      apply auto
      done
  qed (insert insert, auto)
qed auto

lemma has-integral-negligible:
  fixes f :: 'b::euclidean-space ⇒ 'a::real-normed-vector
  assumes negligible s
    and ∀ x∈(t - s). f x = 0
  shows (f has-integral 0) t
proof -
  presume P: ⋀f::'b::euclidean-space ⇒ 'a.
    ⋀a b. ∀ x. x ∉ s ⟶ f x = 0 ⟹ (f has-integral 0) (cbox a b)
  let ?f = (λx. if x ∈ t then f x else 0)
  show ?thesis
```

```
    apply (rule-tac f=?f in has-integral-eq)
    unfolding if-P
    apply (rule refl)
    apply (subst has-integral-alt)
    apply cases
    apply (subst if-P, assumption)
    unfolding if-not-P
  proof −
    assume ∃ a b. t = cbox a b
    then guess a b apply − by (erule exE)+ note t = this
    show (?f has-integral 0) t
      unfolding t
      apply (rule P)
      using assms(2)
      unfolding t
      apply auto
      done
  next
    show ∀ e>0. ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
      (∃ z. ((λx. if x ∈ t then ?f x else 0) has-integral z) (cbox a b) ∧ norm (z −
0) < e)
      apply safe
      apply (rule-tac x=1 in exI)
      apply rule
      apply (rule zero-less-one)
      apply safe
      apply (rule-tac x=0 in exI)
      apply rule
      apply (rule P)
      using assms(2)
      apply auto
      done
  qed
next
  fix f :: 'b ⇒ 'a
  fix a b :: 'b
  assume assm: ∀ x. x ∉ s ⟶ f x = 0
  show (f has-integral 0) (cbox a b)
    unfolding has-integral
  proof (safe, goal-cases)
    case prems: (1 e)
    then have ⋀n. e / 2 / ((real n+1) * (2 ^ n)) > 0
      apply −
      apply (rule divide-pos-pos)
      defer
      apply (rule mult-pos-pos)
      apply (auto simp add:field-simps)
      done
    note assms(1)[unfolded negligible-def has-integral,rule-format,OF this,of a b]
```

```
    note allI[OF this,of λx. x]
    from choice[OF this] guess d .. note d=conjunctD2[OF this[rule-format]]
    show ?case
     apply (rule-tac x=λx. d (nat ⌊norm (f x)⌋) x in exI)
    proof safe
     show gauge (λx. d (nat ⌊norm (f x)⌋) x)
      using d(1) unfolding gauge-def by auto
     fix p
     assume as: p tagged-division-of (cbox a b) (λx. d (nat ⌊norm (f x)⌋) x) fine
p
     let ?goal = norm ((∑(x, k)∈p. content k *R f x) - 0) < e
     {
      presume p ≠ {} ⟹ ?goal
      then show ?goal
       apply (cases p = {})
       using prems
       apply auto
       done
     }
     assume as': p ≠ {}
     from real-arch-simple[of Max((λ(x,k). norm(f x)) ` p)] guess N ..
     then have N: ∀x∈(λ(x, k). norm (f x)) ` p. x ≤ real N
     by (meson Max-ge as(1) dual-order.trans finite-imageI tagged-division-of-finite)
      have ∀i. ∃q. q tagged-division-of (cbox a b) ∧ (d i) fine q ∧ (∀(x, k)∈p. k
⊆ (d i) x ⟶ (x, k) ∈ q)
       by (auto intro: tagged-division-finer[OF as(1) d(1)])
     from choice[OF this] guess q .. note q=conjunctD3[OF this[rule-format]]
     have *: ⋀i. (∑(x, k)∈q i. content k *R indicator s x) ≥ (0::real)
      apply (rule setsum-nonneg)
      apply safe
      unfolding real-scaleR-def
      apply (drule tagged-division-ofD(4)[OF q(1)])
      apply (auto intro: mult-nonneg-nonneg)
      done
     have **: finite s ⟹ finite t ⟹ (∀(x,y) ∈ t. (0::real) ≤ g(x,y)) ⟹
      (∀y∈s. ∃x. (x,y) ∈ t ∧ f(y) ≤ g(x,y)) ⟹ setsum f s ≤ setsum g t for f g
s t
      apply (rule setsum-le-included[of s t g snd f])
      prefer 4
      apply safe
      apply (erule-tac x=x in ballE)
      apply (erule exE)
      apply (rule-tac x=(xa,x) in bexI)
      apply auto
      done
     have norm ((∑(x, k)∈p. content k *R f x) - 0) ≤ setsum (λi. (real i + 1)
*
      norm (setsum (λ(x,k). content k *R indicator s x :: real) (q i))) {..N+1}
     unfolding real-norm-def setsum-right-distrib abs-of-nonneg[OF *] diff-0-right
```

```
  apply (rule order-trans)
  apply (rule norm-setsum)
  apply (subst sum-sum-product)
  prefer 3
proof (rule **, safe)
  show finite {(i, j) |i j. i ∈ {..N + 1} ∧ j ∈ q i}
    apply (rule finite-product-dependent)
    using q
    apply auto
    done
  fix i a b
  assume as'': (a, b) ∈ q i
  show 0 ≤ (real i + 1) * (content b *R indicator s a)
    unfolding real-scaleR-def
    using tagged-division-ofD(4)[OF q(1) as'']
    by (auto intro!: mult-nonneg-nonneg)
next
  fix i :: nat
  show finite (q i)
    using q by auto
next
  fix x k
  assume xk: (x, k) ∈ p
  def n ≡ nat ⌊norm (f x)⌋
  have *: norm (f x) ∈ (λ(x, k). norm (f x)) ` p
    using xk by auto
  have nfx: real n ≤ norm (f x) norm (f x) ≤ real n + 1
    unfolding n-def by auto
  then have n ∈ {0..N + 1}
    using N[rule-format,OF *] by auto
  moreover
  note as(2)[unfolded fine-def,rule-format,OF xk,unfolded split-conv]
  note q(3)[rule-format,OF xk,unfolded split-conv,rule-format,OF this]
  note this[unfolded n-def[symmetric]]
  moreover
  have norm (content k *R f x) ≤ (real n + 1) * (content k * indicator s x)
  proof (cases x ∈ s)
    case False
    then show ?thesis
      using assm by auto
  next
    case True
    have *: content k ≥ 0
      using tagged-division-ofD(4)[OF as(1) xk] by auto
    moreover
    have content k * norm (f x) ≤ content k * (real n + 1)
      apply (rule mult-mono)
      using nfx *
      apply auto
```

```
          done
        ultimately
        show ?thesis
          unfolding abs-mult
          using nfx True
          by (auto simp add: field-simps)
      qed
      ultimately show ∃y. (y, x, k) ∈ {(i, j) |i j. i ∈ {..N + 1} ∧ j ∈ q i} ∧
norm (content k *R f x) ≤
        (real y + 1) * (content k *R indicator s x)
        apply (rule-tac x=n in exI)
        apply safe
        apply (rule-tac x=n in exI)
        apply (rule-tac x=(x,k) in exI)
        apply safe
        apply auto
        done
    qed (insert as, auto)
    also have ... ≤ setsum (λi. e / 2 / 2 ^ i) {..N+1}
    proof (rule setsum-mono, goal-cases)
      case (1 i)
      then show ?case
        apply (subst mult.commute, subst pos-le-divide-eq[symmetric])
        using d(2)[rule-format, of q i i]
        using q[rule-format]
        apply (auto simp add: field-simps)
        done
    qed
    also have ... < e * inverse 2 * 2
      unfolding divide-inverse setsum-right-distrib[symmetric]
      apply (rule mult-strict-left-mono)
      unfolding power-inverse [symmetric] lessThan-Suc-atMost[symmetric]
      apply (subst geometric-sum)
      using prems
      apply auto
      done
    finally show ?goal by auto
  qed
 qed
qed

lemma has-integral-spike:
 fixes f :: 'b::euclidean-space ⇒ 'a::real-normed-vector
 assumes negligible s
   and (∀x∈(t - s). g x = f x)
   and (f has-integral y) t
 shows (g has-integral y) t
proof −
 {
```

```
    fix a b :: 'b
    fix f g :: 'b ⇒ 'a
    fix y :: 'a
    assume as: ∀x ∈ cbox a b − s. g x = f x (f has-integral y) (cbox a b)
    have ((λx. f x + (g x − f x)) has-integral (y + 0)) (cbox a b)
      apply (rule has-integral-add[OF as(2)])
      apply (rule has-integral-negligible[OF assms(1)])
      using as
      apply auto
      done
    then have (g has-integral y) (cbox a b)
      by auto
  } note * = this
  show ?thesis
    apply (subst has-integral-alt)
    using assms(2−)
    apply −
    apply (rule cond-cases)
    apply safe
    apply (rule *)
    apply assumption+
    apply (subst(asm) has-integral-alt)
    unfolding if-not-P
    apply (erule-tac x=e in allE)
    apply safe
    apply (rule-tac x=B in exI)
    apply safe
    apply (erule-tac x=a in allE)
    apply (erule-tac x=b in allE)
    apply safe
    apply (rule-tac x=z in exI)
    apply safe
    apply (rule *[where fa2=λx. if x∈t then f x else 0])
    apply auto
    done
qed

lemma has-integral-spike-eq:
  assumes negligible s
    and ∀x∈(t − s). g x = f x
  shows ((f has-integral y) t ⟷ (g has-integral y) t)
  apply rule
  apply (rule-tac[!] has-integral-spike[OF assms(1)])
  using assms(2)
  apply auto
  done

lemma integrable-spike:
  assumes negligible s
```

and $\forall x{\in}(t - s).\ g\ x = f\ x$
and *f integrable-on t*
shows *g integrable-on t*
using *assms*
unfolding *integrable-on-def*
apply −
apply (*erule exE*)
apply *rule*
apply (*rule has-integral-spike*)
apply *fastforce+*
done

lemma *integral-spike*:
assumes *negligible s*
and $\forall x{\in}(t - s).\ g\ x = f\ x$
shows *integral t f = integral t g*
unfolding *integral-def*
using *has-integral-spike-eq*[*OF assms*]
by *auto*

35.32 Some other trivialities about negligible sets.

lemma *negligible-subset*[*intro*]:
assumes *negligible s*
and $t \subseteq s$
shows *negligible t*
unfolding *negligible-def*
proof (*safe*, *goal-cases*)
case (*1 a b*)
show *?case*
using *assms*(*1*)[*unfolded negligible-def*,*rule-format*,*of a b*]
apply −
apply (*rule has-integral-spike*[*OF assms*(*1*)])
defer
apply *assumption*
using *assms*(*2*)
unfolding *indicator-def*
apply *auto*
done
qed

lemma *negligible-diff*[*intro?*]:
assumes *negligible s*
shows *negligible* (*s* − *t*)
using *assms* **by** *auto*

lemma *negligible-inter*:
assumes *negligible s* ∨ *negligible t*
shows *negligible* (*s* ∩ *t*)

using *assms* **by** *auto*

lemma *negligible-union*:
assumes *negligible s*
and *negligible t*
shows *negligible* (*s* ∪ *t*)
unfolding *negligible-def*
proof (*safe*, *goal-cases*)
case (*1 a b*)
note *assm* = *assms*[*unfolded negligible-def*,*rule-format*,*of a b*]
then show *?case*
apply (*subst has-integral-spike-eq*[*OF assms*(*2*)])
defer
apply *assumption*
unfolding *indicator-def*
apply *auto*
done
qed

lemma *negligible-union-eq*[*simp*]: *negligible* (*s* ∪ *t*) ⟷ *negligible s* ∧ *negligible t*
using *negligible-union* **by** *auto*

lemma *negligible-sing*[*intro*]: *negligible* {*a*::'*a*::*euclidean-space*}
using *negligible-standard-hyperplane*[*OF SOME-Basis*, *of a* · (*SOME i*. *i* ∈ *Basis*)] **by** *auto*

lemma *negligible-insert*[*simp*]: *negligible* (*insert a s*) ⟷ *negligible s*
apply (*subst insert-is-Un*)
unfolding *negligible-union-eq*
apply *auto*
done

lemma *negligible-empty*[*iff*]: *negligible* {}
by *auto*

lemma *negligible-finite*[*intro*]:
assumes *finite s*
shows *negligible s*
using *assms* **by** (*induct s*) *auto*

lemma *negligible-unions*[*intro*]:
assumes *finite s*
and ∀ *t*∈*s*. *negligible t*
shows *negligible*(⋃ *s*)
using *assms* **by** *induct auto*

lemma *negligible*:
negligible s ⟷ (∀ *t*::('*a*::*euclidean-space*) *set*. ((*indicator s*::'*a*⇒*real*) *has-integral 0*) *t*)

apply *safe*
defer
apply (*subst negligible-def*)
proof −
fix $t :: 'a\ set$
assume *as*: *negligible s*
have ∗: (λ*x*. *if x ∈ s ∩ t then 1 else 0*) = (λ*x*. *if x∈t then if x∈s then 1 else 0 else 0*)
by *auto*
show ((*indicator s*::′*a*⇒*real*) *has-integral 0*) *t*
apply (*subst has-integral-alt*)
apply *cases*
apply (*subst if-P*,*assumption*)
unfolding *if-not-P*
apply *safe*
apply (*rule as*[*unfolded negligible-def*,*rule-format*])
apply (*rule-tac x=1* **in** *exI*)
apply *safe*
apply (*rule zero-less-one*)
apply (*rule-tac x=0* **in** *exI*)
using *negligible-subset*[*OF as*,*of s ∩ t*]
unfolding *negligible-def indicator-def* [*abs-def*]
unfolding ∗
apply *auto*
done
qed *auto*

35.33 Finite case of the spike theorem is quite commonly needed.

lemma *has-integral-spike-finite*:
assumes *finite s*
and ∀ *x*∈*t*−*s*. *g x* = *f x*
and (*f has-integral y*) *t*
shows (*g has-integral y*) *t*
apply (*rule has-integral-spike*)
using *assms*
apply *auto*
done

lemma *has-integral-spike-finite-eq*:
assumes *finite s*
and ∀ *x*∈*t*−*s*. *g x* = *f x*
shows ((*f has-integral y*) *t* ⟷ (*g has-integral y*) *t*)
apply *rule*
apply (*rule-tac*[!] *has-integral-spike-finite*)
using *assms*
apply *auto*
done

lemma *integrable-spike-finite*:
assumes *finite s*
and $\forall x{\in}t{-}s.\ g\ x = f\ x$
and *f integrable-on t*
shows *g integrable-on t*
using *assms*
unfolding *integrable-on-def*
apply *safe*
apply (*rule-tac x=y* **in** *exI*)
apply (*rule has-integral-spike-finite*)
apply *auto*
done

35.34 In particular, the boundary of an interval is negligible.

lemma *negligible-frontier-interval*: *negligible*(*cbox* (*a*::'*a*::*euclidean-space*) *b* − *box a b*)
proof −
let *?A* = $\bigcup$((λ*k*. {*x*. *x*·*k* = *a*·*k*} ∪ {*x*::'*a*. *x*·*k* = *b*·*k*}) ' *Basis*)
have *cbox a b* − *box a b* ⊆ *?A*
apply *rule* **unfolding** *Diff-iff mem-box*
apply *simp*
apply(*erule conjE bexE*)+
apply(*rule-tac x=i* **in** *bexI*)
apply *auto*
done
then show *?thesis*
apply −
apply (*rule negligible-subset*[*of ?A*])
apply (*rule negligible-unions*[*OF finite-imageI*])
apply *auto*
done
qed

lemma *has-integral-spike-interior*:
assumes $\forall x{\in}box\ a\ b.\ g\ x = f\ x$
and (*f has-integral y*) (*cbox a b*)
shows (*g has-integral y*) (*cbox a b*)
apply (*rule has-integral-spike*[*OF negligible-frontier-interval - assms*(*2*)])
using *assms*(*1*)
apply *auto*
done

lemma *has-integral-spike-interior-eq*:
assumes $\forall x{\in}box\ a\ b.\ g\ x = f\ x$
shows (*f has-integral y*) (*cbox a b*) ⟷ (*g has-integral y*) (*cbox a b*)
apply *rule*
apply (*rule-tac*[!] *has-integral-spike-interior*)

using *assms*
apply *auto*
done

lemma *integrable-spike-interior*:
assumes $\forall\, x{\in}box\ a\ b.\ g\ x = f\ x$
and *f integrable-on cbox a b*
shows *g integrable-on cbox a b*
using *assms*
unfolding *integrable-on-def*
using *has-integral-spike-interior*[*OF assms*(*1*)]
by *auto*

35.35 Integrability of continuous functions.

lemma *neutral-and*[*simp*]: *neutral op* $\wedge$ $=$ *True*
unfolding *neutral-def* **by** (*rule some-equality*) *auto*

lemma *monoidal-and*[*intro*]: *monoidal op* $\wedge$
unfolding *monoidal-def* **by** *auto*

lemma *iterate-and*[*simp*]:
assumes *finite s*
shows (*iterate op* $\wedge$) *s p* $\longleftrightarrow$ $(\forall\, x{\in}s.\ p\ x)$
using *assms*
apply *induct*
unfolding *iterate-insert*[*OF monoidal-and*]
apply *auto*
done

lemma *operative-division-and*:
assumes *operative op* $\wedge$ *P*
and *d division-of* (*cbox a b*)
shows $(\forall\, i{\in}d.\ P\ i) \longleftrightarrow P$ (*cbox a b*)
using *operative-division*[*OF monoidal-and assms*] *division-of-finite*[*OF assms*(*2*)]
by *auto*

lemma *operative-approximable*:
fixes *f* :: *'b*::*euclidean-space* $\Rightarrow$ *'a*::*banach*
assumes $0 \le e$
shows *operative op* $\wedge$ $(\lambda i.\ \exists\, g.\ (\forall\, x{\in}i.\ norm\ (f\,x - g\ (x{::}'b)) \le e) \wedge g$ *integrable-on* $i)$
unfolding *operative-def neutral-and*
proof *safe*
fix *a b* :: *'b*
show $\exists\, g.\ (\forall\, x{\in}cbox\ a\ b.\ norm\ (f\,x - g\ x) \le e) \wedge g$ *integrable-on cbox a b*
if *content* (*cbox a b*) $= 0$
apply (*rule-tac x*=*f* **in** *exI*)
using *assms that*

apply (*auto intro*!: *integrable-on-null*)
done
{
fix *c g*
fix *k* :: *'b*
assume *as*: ∀ *x*∈*cbox a b*. *norm* (*f x* − *g x*) ≤ *e g integrable-on cbox a b*
assume *k*: *k* ∈ *Basis*
show ∃ *g*. (∀ *x*∈*cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable-on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}
∃ *g*. (∀ *x*∈*cbox a b* ∩ {*x*. *c* ≤ *x* • *k*}. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable-on cbox a b* ∩ {*x*. *c* ≤ *x* • *k*}
apply (*rule-tac*[!] *x*=*g* **in** *exI*)
using *as*(*1*) *integrable-split*[*OF as*(*2*) *k*]
apply *auto*
done
}
fix *c k g1 g2*
assume *as*: ∀ *x*∈*cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}. *norm* (*f x* − *g1 x*) ≤ *e g1 integrable-on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}
∀ *x*∈*cbox a b* ∩ {*x*. *c* ≤ *x* • *k*}. *norm* (*f x* − *g2 x*) ≤ *e g2 integrable-on cbox a b* ∩ {*x*. *c* ≤ *x* • *k*}
assume *k*: *k* ∈ *Basis*
let *?g* = λ*x*. *if x*•*k* = *c then f x else if x*•*k* ≤ *c then g1 x else g2 x*
show ∃ *g*. (∀ *x*∈*cbox a b*. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable-on cbox a b*
apply (*rule-tac x*=*?g* **in** *exI*)
apply *safe*
proof *goal-cases*
case (*1 x*)
then show *?case*
apply −
apply (*cases x*•*k*=*c*)
apply (*case-tac x*•*k* < *c*)
using *as assms*
apply *auto*
done
next
case *2*
presume *?g integrable-on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}
and *?g integrable-on cbox a b* ∩ {*x*. *x* • *k* ≥ *c*}
then guess *h1 h2* **unfolding** *integrable-on-def* **by** *auto*
from *has-integral-split*[*OF this k*] **show** *?case*
unfolding *integrable-on-def* **by** *auto*
next
show *?g integrable-on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*} *?g integrable-on cbox a b* ∩ {*x*. *x* • *k* ≥ *c*}
apply(*rule-tac*[!] *integrable-spike*[*OF negligible-standard-hyperplane*[*of k c*]])
using *k as*(*2*,*4*)
apply *auto*
done

```
  qed
qed

lemma approximable-on-division:
  fixes f :: 'b::euclidean-space ⇒ 'a::banach
  assumes 0 ≤ e
    and d division-of (cbox a b)
    and ∀ i∈d. ∃ g. (∀ x∈i. norm (f x − g x) ≤ e) ∧ g integrable-on i
  obtains g where ∀ x∈cbox a b. norm (f x − g x) ≤ e g integrable-on cbox a b
proof −
 note ∗ = operative-division[OF monoidal-and operative-approximable[OF assms(1)]
assms(2)]
  note this[unfolded iterate-and[OF division-of-finite[OF assms(2)]]]
  from assms(3)[unfolded this[of f]] guess g ..
  then show thesis
    apply −
    apply (rule that[of g])
    apply auto
    done
qed

lemma integrable-continuous:
  fixes f :: 'b::euclidean-space ⇒ 'a::banach
  assumes continuous-on (cbox a b) f
  shows f integrable-on cbox a b
proof (rule integrable-uniform-limit, safe)
  fix e :: real
  assume e: e > 0
 from compact-uniformly-continuous[OF assms compact-cbox,unfolded uniformly-continuous-on-def,rule-form
e] guess d ..
  note d=conjunctD2[OF this,rule-format]
  from fine-division-exists[OF gauge-ball[OF d(1)], of a b] guess p . note p=this
  note p' = tagged-division-ofD[OF p(1)]
  have ∗: ∀ i∈snd ‘ p. ∃ g. (∀ x∈i. norm (f x − g x) ≤ e) ∧ g integrable-on i
  proof (safe, unfold snd-conv)
    fix x l
    assume as: (x, l) ∈ p
    from p'(4)[OF this] guess a b by (elim exE) note l=this
    show ∃ g. (∀ x∈l. norm (f x − g x) ≤ e) ∧ g integrable-on l
      apply (rule-tac x=λy. f x in exI)
    proof safe
      show (λy. f x) integrable-on l
        unfolding integrable-on-def l
        apply rule
        apply (rule has-integral-const)
        done
      fix y
      assume y: y ∈ l
      note fineD[OF p(2) as,unfolded subset-eq,rule-format,OF this]
```

note *d(2)[OF - - this[unfolded mem-ball]]*
then show *norm (f y − f x) ≤ e*
using *y p'(2−3)[OF as]* **unfolding** *dist-norm l norm-minus-commute* **by** *fastforce*
qed
qed
from *e* **have** *e ≥ 0*
by *auto*
from *approximable-on-division[OF this division-of-tagged-division[OF p(1)] ∗]* **guess** *g* **.**
then show *∃ g. (∀ x∈cbox a b. norm (f x − g x) ≤ e) ∧ g integrable-on cbox a b*
by *auto*
qed

lemma *integrable-continuous-real*:
fixes *f :: real ⇒ 'a::banach*
assumes *continuous-on {a .. b} f*
shows *f integrable-on {a .. b}*
by (*metis assms box-real(2) integrable-continuous*)

35.36 Specialization of additivity to one dimension.

lemma
shows *real-inner-1-left*: *inner 1 x = x*
and *real-inner-1-right*: *inner x 1 = x*
by *simp-all*

lemma *content-real-eq-0*: *content {a .. b::real} = 0 ⟷ a ≥ b*
by (*metis atLeastatMost-empty-iff2 content-empty content-real diff-self eq-iff le-cases le-iff-diff-le-0*)

lemma *interval-real-split*:
{a .. b::real} ∩ {x. x ≤ c} = {a .. min b c}
{a .. b} ∩ {x. c ≤ x} = {max a c .. b}
apply (*metis Int-atLeastAtMostL1 atMost-def*)
apply (*metis Int-atLeastAtMostL2 atLeast-def*)
done

lemma *operative-1-lt*:
assumes *monoidal opp*
shows *operative opp f ⟷ ((∀ a b. b ≤ a ⟶ f {a .. b::real} = neutral opp) ∧ (∀ a b c. a < c ∧ c < b ⟶ opp (f {a .. c}) (f {c .. b}) = f {a .. b}))*
apply (*simp add*: *operative-def content-real-eq-0 del*: *content-real-if*)
proof *safe*
fix *a b c :: real*
assume *as*:
∀ a b c. f {a..b} = opp (f ({a..b} ∩ {x. x ≤ c})) (f ({a..b} ∩ Collect (op ≤ c)))
a < c

```
    c < b
    from this(2−) have cbox a b ∩ {x. x ≤ c} = cbox a c cbox a b ∩ {x. x ≥ c}
= cbox c b
      by (auto simp: mem-box)
    then show opp (f {a..c}) (f {c..b}) = f {a..b}
      unfolding as(1)[rule-format,of a b c] by auto
next
  fix a b c :: real
  assume as: ∀ a b. b ≤ a ⟶ f {a..b} = neutral opp
    ∀ a b c. a < c ∧ c < b ⟶ opp (f {a..c}) (f {c..b}) = f {a..b}
  show f {a..b} = opp (f ({a..b} ∩ {x. x ≤ c})) (f ({a..b} ∩ Collect (op ≤ c)))
  proof (cases c ∈ {a..b})
    case False
    then have c < a ∨ c > b by auto
    then show ?thesis
    proof
      assume c < a
      then have ∗: {a..b} ∩ {x. x ≤ c} = {1..0} {a..b} ∩ {x. c ≤ x} = {a..b}
        by auto
      show ?thesis
        unfolding ∗
        apply (subst as(1)[rule-format,of 0 1])
        using assms
        apply auto
        done
    next
      assume b < c
      then have ∗: {a..b} ∩ {x. x ≤ c} = {a..b} {a..b} ∩ {x. c ≤ x} = {1 .. 0}
        by auto
      show ?thesis
        unfolding ∗
        apply (subst as(1)[rule-format,of 0 1])
        using assms
        apply auto
        done
    qed
  next
    case True
    then have ∗: min (b) c = c max a c = c
      by auto
    have ∗∗: (1::real) ∈ Basis
      by simp
    have ∗∗∗: ⋀P Q. (∑ i∈Basis. (if i = 1 then P i else Q i) ∗R i) = (P 1::real)
      by simp
    show ?thesis
      unfolding interval-real-split unfolding ∗
    proof (cases c = a ∨ c = b)
      case False
      then show f {a..b} = opp (f {a..c}) (f {c..b})
```

```
      apply −
      apply (subst as(2)[rule-format])
      using True
      apply auto
      done
    next
     case True
     then show f {a..b} = opp (f {a..c}) (f {c..b})
     proof
      assume ∗: c = a
      then have f {a .. c} = neutral opp
       apply −
       apply (rule as(1)[rule-format])
       apply auto
       done
      then show ?thesis
       using assms unfolding ∗ by auto
     next
      assume ∗: c = b
      then have f {c .. b} = neutral opp
       apply −
       apply (rule as(1)[rule-format])
       apply auto
       done
      then show ?thesis
       using assms unfolding ∗ by auto
     qed
    qed
  qed
qed

lemma operative-1-le:
  assumes monoidal opp
  shows operative opp f ⟷ ((∀ a b. b ≤ a ⟶ f {a .. b::real} = neutral opp) ∧
    (∀ a b c. a ≤ c ∧ c ≤ b ⟶ opp (f {a .. c}) (f {c .. b}) = f {a .. b}))
  unfolding operative-1-lt[OF assms]
proof safe
  fix a b c :: real
  assume as:
    ∀ a b c. a ≤ c ∧ c ≤ b ⟶ opp (f {a..c}) (f {c..b}) = f {a..b}
    a < c
    c < b
  show opp (f {a..c}) (f {c..b}) = f {a..b}
    apply (rule as(1)[rule-format])
    using as(2−)
    apply auto
    done
next
  fix a b c :: real
```

assume $\forall\, a\ b.\ b \leq a \longrightarrow f\ \{a\ ..\ b\} = \mathit{neutral}\ \mathit{opp}$
and $\forall\, a\ b\ c.\ a < c \wedge c < b \longrightarrow \mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}$
and $a \leq c$
and $c \leq b$
note $\mathit{as} = \mathit{this}[\mathit{rule\text{-}format}]$
show $\mathit{opp}\ (f\ \{a..c\})\ (f\ \{c..b\}) = f\ \{a..b\}$
proof $(\mathit{cases}\ c = a \vee c = b)$
case *False*
then show *?thesis*
apply −
apply $(\mathit{subst}\ \mathit{as}(2))$
using $\mathit{as}(3-)$
apply *auto*
done
next
case *True*
then show *?thesis*
proof
assume $*$: $c = a$
then have $f\ \{a\ ..\ c\} = \mathit{neutral}\ \mathit{opp}$
apply −
apply $(\mathit{rule}\ \mathit{as}(1)[\mathit{rule\text{-}format}])$
apply *auto*
done
then show *?thesis*
using *assms* **unfolding** $*$ **by** *auto*
next
assume $*$: $c = b$
then have $f\ \{c\ ..\ b\} = \mathit{neutral}\ \mathit{opp}$
apply −
apply $(\mathit{rule}\ \mathit{as}(1)[\mathit{rule\text{-}format}])$
apply *auto*
done
then show *?thesis*
using *assms* **unfolding** $*$ **by** *auto*
qed
qed
qed

35.37 Special case of additivity we need for the FCT.

lemma *additive-tagged-division-1*:
fixes f :: *real* $\Rightarrow$ *'a*::*real-normed-vector*
assumes $a \leq b$
and p *tagged-division-of* $\{a..b\}$
shows $\mathit{setsum}\ (\lambda(x,k).\ f(\mathit{Sup}\ k) - f(\mathit{Inf}\ k))\ p = f\ b - f\ a$
proof −
let *?f* = $(\lambda k{::}(\mathit{real})\ \mathit{set}.\ \mathit{if}\ k = \{\}\ \mathit{then}\ 0\ \mathit{else}\ f(\mathit{interval\text{-}upperbound}\ k) - f(\mathit{interval\text{-}lowerbound}\ k))$

```
  have ***: ∀ i∈Basis. a · i ≤ b · i
    using assms by auto
  have *: operative op + ?f
    unfolding operative-1-lt[OF monoidal-monoid] box-eq-empty
    by auto
  have **: cbox a b ≠ {}
    using assms(1) by auto
 note operative-tagged-division[OF monoidal-monoid * assms(2)[simplified box-real[symmetric]]]
  note * = this[unfolded if-not-P[OF **] interval-bounds[OF ***],symmetric]
  show ?thesis
    unfolding *
    apply (subst setsum-iterate[symmetric])
    defer
    apply (rule setsum.cong)
    unfolding split-paired-all split-conv
    using assms(2)
    apply auto
    done
qed
```

35.38 A useful lemma allowing us to factor out the content size.

```
lemma has-integral-factor-content:
  (f has-integral i) (cbox a b) ⟷
    (∀ e>0. ∃ d. gauge d ∧ (∀ p. p tagged-division-of (cbox a b) ∧ d fine p ⟶
      norm (setsum (λ(x,k). content k *R f x) p − i) ≤ e * content (cbox a b)))
proof (cases content (cbox a b) = 0)
  case True
  show ?thesis
    unfolding has-integral-null-eq[OF True]
    apply safe
    apply (rule, rule, rule gauge-trivial, safe)
    unfolding setsum-content-null[OF True] True
    defer
    apply (erule-tac x=1 in allE)
    apply safe
    defer
    apply (rule fine-division-exists[of - a b])
    apply assumption
    apply (erule-tac x=p in allE)
    unfolding setsum-content-null[OF True]
    apply auto
    done
next
  case False
  note F = this[unfolded content-lt-nz[symmetric]]
  let ?P = λe opp. ∃ d. gauge d ∧
    (∀ p. p tagged-division-of (cbox a b) ∧ d fine p ⟶ opp (norm ((∑(x, k)∈p.
```

```
content k *R f x) - i)) e)
  show ?thesis
    apply (subst has-integral)
  proof safe
    fix e :: real
    assume e: e > 0
    {
      assume ∀ e>0. ?P e op <
      then show ?P (e * content (cbox a b)) op ≤
        apply (erule-tac x=e * content (cbox a b) in allE)
        apply (erule impE)
        defer
        apply (erule exE,rule-tac x=d in exI)
        using F e
        apply (auto simp add:field-simps)
        done
    }
    {
      assume ∀ e>0. ?P (e * content (cbox a b)) op ≤
      then show ?P e op <
        apply (erule-tac x=e / 2 / content (cbox a b) in allE)
        apply (erule impE)
        defer
        apply (erule exE,rule-tac x=d in exI)
        using F e
        apply (auto simp add: field-simps)
        done
    }
  qed
qed

lemma has-integral-factor-content-real:
  (f has-integral i) {a .. b::real} ⟷
    (∀ e>0. ∃ d. gauge d ∧ (∀ p. p tagged-division-of {a .. b} ∧ d fine p ⟶
      norm (setsum (λ(x,k). content k *R f x) p - i) ≤ e * content {a .. b} ))
  unfolding box-real[symmetric]
  by (rule has-integral-factor-content)
```

35.39 Fundamental theorem of calculus.

```
lemma interval-bounds-real:
  fixes q b :: real
  assumes a ≤ b
  shows Sup {a..b} = b
    and Inf {a..b} = a
  using assms by auto

lemma fundamental-theorem-of-calculus:
  fixes f :: real ⇒ 'a::banach
```

```
  assumes a ≤ b
    and ∀x∈{a .. b}. (f has-vector-derivative f' x) (at x within {a .. b})
  shows (f' has-integral (f b − f a)) {a .. b}
  unfolding has-integral-factor-content box-real[symmetric]
proof safe
  fix e :: real
  assume e: e > 0
 note assm = assms(2)[unfolded has-vector-derivative-def has-derivative-within-alt]
 have ∗: ⋀P Q. ∀x∈{a .. b}. P x ∧ (∀e>0. ∃d>0. Q x e d) ⟹ ∀x. ∃(d::real)>0.
x∈{a .. b} ⟶ Q x e d
    using e by blast
  note this[OF assm,unfolded gauge-existence-lemma]
  from choice[OF this,unfolded Ball-def[symmetric]] guess d ..
  note d=conjunctD2[OF this[rule-format],rule-format]
  show ∃d. gauge d ∧ (∀p. p tagged-division-of (cbox a b) ∧ d fine p ⟶
    norm ((∑(x, k)∈p. content k ∗R f' x) − (f b − f a)) ≤ e ∗ content (cbox a
b))
    apply (rule-tac x=λx. ball x (d x) in exI)
    apply safe
    apply (rule gauge-ball-dependent)
    apply rule
    apply (rule d(1))
  proof −
    fix p
    assume as: p tagged-division-of cbox a b (λx. ball x (d x)) fine p
     show norm ((∑(x, k)∈p. content k ∗R f' x) − (f b − f a)) ≤ e ∗ content
(cbox a b)
    unfolding content-real[OF assms(1), simplified box-real[symmetric]] additive-tagged-division-1[OF
assms(1) as(1)[simplified box-real],of f,symmetric]
     unfolding additive-tagged-division-1[OF assms(1) as(1)[simplified box-real],of
λx. x,symmetric]
      unfolding setsum-right-distrib
      defer
      unfolding setsum-subtractf[symmetric]
    proof (rule setsum-norm-le,safe)
      fix x k
      assume (x, k) ∈ p
      note xk = tagged-division-ofD(2−4)[OF as(1) this]
      from this(3) guess u v by (elim exE) note k=this
      have ∗: u ≤ v
        using xk unfolding k by auto
      have ball: ∀xa∈k. xa ∈ ball x (d x)
        using as(2)[unfolded fine-def,rule-format,OF ‹(x,k)∈p›,unfolded split-conv
subset-eq] .
      have norm ((v − u) ∗R f' x − (f v − f u)) ≤
        norm (f u − f x − (u − x) ∗R f' x) + norm (f v − f x − (v − x) ∗R f' x)
        apply (rule order-trans[OF - norm-triangle-ineq4])
        apply (rule eq-refl)
        apply (rule arg-cong[where f=norm])
```

```
      unfolding scaleR-diff-left
      apply (auto simp add:algebra-simps)
      done
    also have ... ≤ e * norm (u - x) + e * norm (v - x)
      apply (rule add-mono)
      apply (rule d(2)[of x u,unfolded o-def])
      prefer 4
      apply (rule d(2)[of x v,unfolded o-def])
      using ball[rule-format,of u] ball[rule-format,of v]
      using xk(1-2)
      unfolding k subset-eq
      apply (auto simp add:dist-real-def)
      done
    also have ... ≤ e * (Sup k - Inf k)
      unfolding k interval-bounds-real[OF *]
      using xk(1)
      unfolding k
      by (auto simp add: dist-real-def field-simps)
    finally show norm (content k *R f' x - (f (Sup k) - f (Inf k))) ≤
      e * (Sup k - Inf k)
      unfolding box-real k interval-bounds-real[OF *] content-real[OF *]
        interval-upperbound-real interval-lowerbound-real
        .
  qed
 qed
qed
```

35.40 Taylor series expansion

```
lemma (in bounded-bilinear) setsum-prod-derivatives-has-vector-derivative:
 assumes p>0
 and f0: Df 0 = f
 and Df: ⋀m t. m < p ⟹ a ≤ t ⟹ t ≤ b ⟹
   (Df m has-vector-derivative Df (Suc m) t) (at t within {a .. b})
 and g0: Dg 0 = g
 and Dg: ⋀m t. m < p ⟹ a ≤ t ⟹ t ≤ b ⟹
   (Dg m has-vector-derivative Dg (Suc m) t) (at t within {a .. b})
 and ivl: a ≤ t t ≤ b
 shows ((λt. ∑i<p. (-1) ^ i *R prod (Df i t) (Dg (p - Suc i) t))
   has-vector-derivative
     prod (f t) (Dg p t) - (-1) ^ p *R prod (Df p t) (g t))
   (at t within {a .. b})
 using assms
proof cases
 assume p: p ≠ 1
 def p' ≡ p - 2
 from assms p have p': {..<p} = {..Suc p'} p = Suc (Suc p')
   by (auto simp: p'-def)
 have *: ⋀i. i ≤ p' ⟹ Suc (Suc p' - i) = (Suc (Suc p') - i)
```

by *auto*
let *?f* = $\lambda i.\ (-1)\ \hat{}\ i\ *_R$ (*prod* (*Df i t*) (*Dg* ((*p* − *i*)) *t*))
have ($\sum i{<}p.\ (-1)\ \hat{}\ i\ *_R$ (*prod* (*Df i t*) (*Dg* (*Suc* (*p* − *Suc i*)) *t*) +
prod (*Df* (*Suc i*) *t*) (*Dg* (*p* − *Suc i*) *t*))) =
($\sum i{\leq}$(*Suc p'*). *?f i* − *?f* (*Suc i*))
by (*auto simp*: *algebra-simps p'*(*2*) *numeral-2-eq-2* ∗ *lessThan-Suc-atMost*)
also note *setsum-telescope*
finally
have ($\sum i{<}p.\ (-1)\ \hat{}\ i\ *_R$ (*prod* (*Df i t*) (*Dg* (*Suc* (*p* − *Suc i*)) *t*) +
prod (*Df* (*Suc i*) *t*) (*Dg* (*p* − *Suc i*) *t*)))
= *prod* (*f t*) (*Dg p t*) − (− *1*) ^ *p* $*_R$ *prod* (*Df p t*) (*g t*)
unfolding *p'*[*symmetric*]
by (*simp add*: *assms*)
thus *?thesis*
using *assms*
by (*auto intro*!: *derivative-eq-intros has-vector-derivative*)
qed (*auto intro*!: *derivative-eq-intros has-vector-derivative*)

lemma
fixes *f*::*real*⇒*'a*::*banach*
assumes *p*>*0*
and *f0*: *Df 0* = *f*
and *Df*: $\bigwedge m\ t.\ m < p \Longrightarrow a \leq t \Longrightarrow t \leq b \Longrightarrow$
(*Df m has-vector-derivative Df* (*Suc m*) *t*) (*at t within* {*a* .. *b*})
and *ivl*: *a* ≤ *b*
defines *i* ≡ $\lambda x.$ ((*b* − *x*) ^ (*p* − *1*) / *fact* (*p* − *1*)) $*_R$ *Df p x*
shows *taylor-has-integral*:
(*i has-integral f b* − ($\sum i{<}p.$ ((*b* − *a*) ^ *i* / *fact i*) $*_R$ *Df i a*)) {*a*..*b*}
and *taylor-integral*:
f b = ($\sum i{<}p.$ ((*b* − *a*) ^ *i* / *fact i*) $*_R$ *Df i a*) + *integral* {*a*..*b*} *i*
and *taylor-integrable*:
i integrable-on {*a* .. *b*}
proof *goal-cases*
case *1*
interpret *bounded-bilinear scaleR*::*real*⇒*'a*⇒*'a*
by (*rule bounded-bilinear-scaleR*)
def *g* ≡ $\lambda s.$ (*b* − *s*)^(*p* − *1*)/*fact* (*p* − *1*)
def *Dg* ≡ $\lambda n\ s.$ *if n* < *p then* (−*1*)^*n* ∗ (*b* − *s*)^(*p* − *1* − *n*) / *fact* (*p* − *1* − *n*) *else 0*
have *g0*: *Dg 0* = *g*
using ‹*p* > *0*›
by (*auto simp add*: *Dg-def divide-simps g-def split*: *split-if-asm*)
{
fix *m*
assume *p* > *Suc m*
hence *p* − *Suc m* = *Suc* (*p* − *Suc* (*Suc m*))
by *auto*
hence *real* (*p* − *Suc m*) ∗ *fact* (*p* − *Suc* (*Suc m*)) = *fact* (*p* − *Suc m*)
by *auto*

} note *fact-eq = this*
have *Dg*: $\bigwedge m\ t.\ m < p \Longrightarrow a \leq t \Longrightarrow t \leq b \Longrightarrow$
(*Dg m has-vector-derivative Dg (Suc m) t*) (*at t within* {*a .. b*})
unfolding *Dg-def*
by (*auto intro*!: *derivative-eq-intros simp*: *has-vector-derivative-def fact-eq divide-simps*)
let *?sum* = $\lambda t.\ \sum i{<}p.\ (-\ 1)\ \hat{}\ i *_R Dg\ i\ t *_R Df\ (p - Suc\ i)\ t$
from *setsum-prod-derivatives-has-vector-derivative*[*of - Dg - - - Df*,
OF ‹*p > 0*› *g0 Dg f0 Df*]
have *deriv*: $\bigwedge t.\ a \leq t \Longrightarrow t \leq b \Longrightarrow$
(*?sum has-vector-derivative*
g t $*_R$ *Df p t* $-$ $(-\ 1)\ \hat{}\ p *_R Dg\ p\ t *_R f\ t$) (*at t within* {*a..b*})
by *auto*
from *fundamental-theorem-of-calculus*[*rule-format*, *OF* ‹$a \leq b$› *deriv*]
have (*i has-integral ?sum b − ?sum a*) {*a .. b*}
by (*simp add*: *i-def g-def Dg-def*)
also
have *one*: $(-\ 1)\ \hat{}\ p' * (-\ 1)\ \hat{}\ p' = (1{::}real)$
and {..<*p*} ∩ {*i. p = Suc i*} = {*p − 1*}
for *p'*
using ‹*p > 0*›
by (*auto simp*: *power-mult-distrib*[*symmetric*])
then have *?sum b = f b*
using *Suc-pred'*[*OF* ‹*p > 0*›]
by (*simp add*: *diff-eq-eq Dg-def power-0-left le-Suc-eq if-distrib*
cond-application-beta setsum.If-cases f0)
also
have {..<*p*} = $(\lambda x.\ p - x - 1)$ ‘ {..<*p*}
proof *safe*
fix *x*
assume *x < p*
thus *x* ∈ $(\lambda x.\ p - x - 1)$ ‘ {..<*p*}
by (*auto intro*!: *image-eqI*[**where** $x = p - x - 1$])
qed *simp*
from *- this*
have *?sum a* = $(\sum i{<}p.\ ((b - a)\ \hat{}\ i\ /\ fact\ i) *_R Df\ i\ a)$
by (*rule setsum.reindex-cong*) (*auto simp add*: *inj-on-def Dg-def one*)
finally show *c*: *?case* **.**
case *2* **show** *?case* **using** *c integral-unique* **by** *force*
case *3* **show** *?case* **using** *c* **by** *force*
qed

35.41 Attempt a systematic general set of "offset" results for components.

lemma *gauge-modify*:
assumes ($\forall s.$ *open s* $\longrightarrow$ *open* {*x. f*(*x*) ∈ *s*}) *gauge d*
shows *gauge* ($\lambda x.$ {*y. f y* ∈ *d* (*f x*)})
using *assms*
unfolding *gauge-def*

apply *safe*
defer
apply (*erule-tac x=f x* **in** *allE*)
apply (*erule-tac x=d (f x)* **in** *allE*)
apply *auto*
done

35.42 Only need trivial subintervals if the interval itself is trivial.

lemma *division-of-nontrivial*:
fixes *s* :: *'a::euclidean-space set set*
assumes *s division-of (cbox a b)*
and *content (cbox a b) ≠ 0*
shows {*k*. *k ∈ s ∧ content k ≠ 0*} *division-of (cbox a b)*
using *assms(1)*
apply −
proof (*induct card s arbitrary*: *s rule*: *nat-less-induct*)
fix *s::'a set set*
assume *assm*: *s division-of (cbox a b)*
∀ m<card s. *∀ x*. *m = card x ⟶*
x division-of (cbox a b) ⟶ {*k ∈ x*. *content k ≠ 0*} *division-of (cbox a b)*
note *s = division-ofD[OF assm(1)]*
let *?thesis* = {*k ∈ s*. *content k ≠ 0*} *division-of (cbox a b)*
{
presume ∗: {*k ∈ s*. *content k ≠ 0*} *≠ s ⟹ ?thesis*
show *?thesis*
apply *cases*
defer
apply (*rule* ∗)
apply *assumption*
using *assm(1)*
apply *auto*
done
}
assume *noteq*: {*k ∈ s*. *content k ≠ 0*} *≠ s*
then obtain *k* **where** *k*: *k ∈ s content k = 0*
by *auto*
from *s(4)[OF k(1)]* **guess** *c d* **by** (*elim exE*) **note** *k=k this*
from *k* **have** *card s > 0*
unfolding *card-gt-0-iff* **using** *assm(1)* **by** *auto*
then have *card*: *card (s − {k}) < card s*
using *assm(1) k(1)*
apply (*subst card-Diff-singleton-if*)
apply *auto*
done
have ∗: *closed* (⋃(*s − {k}*))
apply (*rule closed-Union*)
defer

```
  apply rule
  apply (drule DiffD1,drule s(4))
  using assm(1)
  apply auto
  done
have k ⊆ ⋃(s − {k})
  apply safe
  apply (rule *[unfolded closed-limpt,rule-format])
  unfolding islimpt-approachable
proof safe
  fix x
  fix e :: real
  assume as: x ∈ k e > 0
  from k(2)[unfolded k content-eq-0] guess i ..
  then have i:c•i = d•i i∈Basis
    using s(3)[OF k(1),unfolded k] unfolding box-ne-empty by auto
  then have xi: x•i = d•i
    using as unfolding k mem-box by (metis antisym)
  def y ≡ ∑j∈Basis. (if j = i then if c•i ≤ (a•i + b•i) / 2 then c•i +
    min e (b•i − c•i) / 2 else c•i − min e (c•i − a•i) / 2 else x•j) *R j
  show ∃x'∈⋃(s − {k}). x' ≠ x ∧ dist x' x < e
    apply (rule-tac x=y in bexI)
  proof
    have d ∈ cbox c d
      using s(3)[OF k(1)]
      unfolding k box-eq-empty mem-box
      by (fastforce simp add: not-less)
    then have d ∈ cbox a b
      using s(2)[OF k(1)]
      unfolding k
      by auto
    note di = this[unfolded mem-box,THEN bspec[where x=i]]
    then have xyi: y•i ≠ x•i
      unfolding y-def i xi
      using as(2) assms(2)[unfolded content-eq-0] i(2)
      by (auto elim!: ballE[of - - i])
    then show y ≠ x
      unfolding euclidean-eq-iff[where 'a='a] using i by auto
    have *: Basis = insert i (Basis − {i})
      using i by auto
    have norm (y − x) < e + setsum (λi. 0) Basis
      apply (rule le-less-trans[OF norm-le-l1])
      apply (subst *)
      apply (subst setsum.insert)
      prefer 3
      apply (rule add-less-le-mono)
    proof −
      show |(y − x) • i| < e
        using di as(2) y-def i xi by (auto simp: inner-simps)
```

```
      show (∑ i∈Basis − {i}. |(y − x) · i|) ≤ (∑ i∈Basis. 0)
        unfolding y-def by (auto simp: inner-simps)
    qed auto
    then show dist y x < e
      unfolding dist-norm by auto
    have y ∉ k
      unfolding k mem-box
      apply rule
      apply (erule-tac x=i in ballE)
      using xyi k i xi
      apply auto
      done
    moreover
    have y ∈ ⋃ s
      using set-rev-mp[OF as(1) s(2)[OF k(1)]] as(2) di i
      unfolding s mem-box y-def
      by (auto simp: field-simps elim!: ballE[of - - i])
    ultimately
    show y ∈ ⋃ (s − {k}) by auto
  qed
 qed
 then have ⋃ (s − {k}) = cbox a b
  unfolding s(6)[symmetric] by auto
 then have {ka ∈ s − {k}. content ka ≠ 0} division-of (cbox a b)
  apply −
  apply (rule assm(2)[rule-format,OF card refl])
  apply (rule division-ofI)
  defer
  apply (rule-tac[1−4] s)
  using assm(1)
  apply auto
  done
 moreover
 have {ka ∈ s − {k}. content ka ≠ 0} = {k ∈ s. content k ≠ 0}
  using k by auto
 ultimately show ?thesis by auto
qed
```

35.43 Integrability on subintervals.

```
lemma operative-integrable:
 fixes f :: 'b::euclidean-space ⇒ 'a::banach
 shows operative op ∧ (λi. f integrable-on i)
 unfolding operative-def neutral-and
 apply safe
 apply (subst integrable-on-def)
 unfolding has-integral-null-eq
 apply (rule, rule refl)
 apply (rule, assumption, assumption)+
```

unfolding *integrable-on-def*
by (*auto intro*!: *has-integral-split*)

lemma *integrable-subinterval*:
fixes *f* :: *'b*::*euclidean-space* ⇒ *'a*::*banach*
assumes *f integrable-on cbox a b*
and *cbox c d* ⊆ *cbox a b*
shows *f integrable-on cbox c d*
apply (*cases cbox c d* = {})
defer
apply (*rule partial-division-extend-1*[*OF assms*(*2*)],*assumption*)
using *operative-division-and*[*OF operative-integrable*,*symmetric*,*of - - - f*] *assms*(*1*)
apply *auto*
done

lemma *integrable-subinterval-real*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *f integrable-on* {*a* .. *b*}
and {*c* .. *d*} ⊆ {*a* .. *b*}
shows *f integrable-on* {*c* .. *d*}
by (*metis assms*(*1*) *assms*(*2*) *box-real*(*2*) *integrable-subinterval*)

35.44 Combining adjacent intervals in 1 dimension.

lemma *has-integral-combine*:
fixes *a b c* :: *real*
assumes *a* ≤ *c*
and *c* ≤ *b*
and (*f has-integral i*) {*a* .. *c*}
and (*f has-integral* (*j*::*'a*::*banach*)) {*c* .. *b*}
shows (*f has-integral* (*i* + *j*)) {*a* .. *b*}
proof −
note *operative-integral*[*of f*, *unfolded operative-1-le*[*OF monoidal-lifted*[*OF monoidal-monoid*]]]
note *conjunctD2*[*OF this*,*rule-format*]
note ∗ = *this*(*2*)[*OF conjI*[*OF assms*(*1*−*2*)],*unfolded if-P*[*OF assms*(*3*)]]
then have *f integrable-on cbox a b*
apply −
apply (*rule ccontr*)
apply (*subst*(*asm*) *if-P*)
defer
apply (*subst*(*asm*) *if-P*)
using *assms*(*3*−)
apply *auto*
done
with ∗
show *?thesis*
apply −
apply (*subst*(*asm*) *if-P*)
defer

apply (*subst*(*asm*) *if-P*)
defer
apply (*subst*(*asm*) *if-P*)
unfolding *lifted.simps*
using *assms*(*3*−)
apply (*auto simp add*: *integrable-on-def integral-unique*)
done
qed

lemma *integral-combine*:
fixes *f* :: *real* ⇒ 'a::*banach*
assumes *a* ≤ *c*
and *c* ≤ *b*
and *f integrable-on* {*a* .. *b*}
shows *integral* {*a* .. *c*} *f* + *integral* {*c* .. *b*} *f* = *integral* {*a* .. *b*} *f*
apply (*rule integral-unique*[*symmetric*])
apply (*rule has-integral-combine*[*OF assms*(*1*−*2*)])
apply (*metis assms*(*2*) *assms*(*3*) *atLeastatMost-subset-iff box-real*(*2*) *content-pos-le*
content-real-eq-0 integrable-integral integrable-subinterval le-add-same-cancel2 monoid-add-class.add.left-neutra
by (*metis assms*(*1*) *assms*(*3*) *atLeastatMost-subset-iff box-real*(*2*) *content-pos-le*
content-real-eq-0 integrable-integral integrable-subinterval le-add-same-cancel1 monoid-add-class.add.right-neut

lemma *integrable-combine*:
fixes *f* :: *real* ⇒ 'a::*banach*
assumes *a* ≤ *c*
and *c* ≤ *b*
and *f integrable-on* {*a* .. *c*}
and *f integrable-on* {*c* .. *b*}
shows *f integrable-on* {*a* .. *b*}
using *assms*
unfolding *integrable-on-def*
by (*fastforce intro*!:*has-integral-combine*)

35.45 Reduce integrability to "local" integrability.

lemma *integrable-on-little-subintervals*:
fixes *f* :: 'b::*euclidean-space* ⇒ 'a::*banach*
assumes ∀ *x*∈*cbox a b*. ∃ *d*>*0*. ∀ *u v*. *x* ∈ *cbox u v* ∧ *cbox u v* ⊆ *ball x d* ∧ *cbox*
u v ⊆ *cbox a b* ⟶
f integrable-on cbox u v
shows *f integrable-on cbox a b*
proof −
have ∀ *x*. ∃ *d*. *x*∈*cbox a b* ⟶ *d*>*0* ∧ (∀ *u v*. *x* ∈ *cbox u v* ∧ *cbox u v* ⊆ *ball x d*
∧ *cbox u v* ⊆ *cbox a b* ⟶
f integrable-on cbox u v)
using *assms* **by** *auto*
note *this*[*unfolded gauge-existence-lemma*]
from *choice*[*OF this*] **guess** *d* **..** **note** *d*=*this*[*rule-format*]
guess *p*

```
    apply (rule fine-division-exists[OF gauge-ball-dependent,of d a b])
    using d
    by auto
  note p=this(1-2)
  note division-of-tagged-division[OF this(1)]
  note * = operative-division-and[OF operative-integrable,OF this,symmetric,of f]
  show ?thesis
    unfolding *
    apply safe
    unfolding snd-conv
  proof -
    fix x k
    assume (x, k) ∈ p
    note tagged-division-ofD(2-4)[OF p(1) this] fineD[OF p(2) this]
    then show f integrable-on k
      apply safe
      apply (rule d[THEN conjunct2,rule-format,of x])
      apply (auto intro: order.trans)
      done
  qed
qed
```

35.46 Second FCT or existence of antiderivative.

```
lemma integrable-const[intro]: (λx. c) integrable-on cbox a b
  unfolding integrable-on-def
  apply rule
  apply (rule has-integral-const)
  done

lemma integral-has-vector-derivative-continuous-at:
  fixes f :: real ⇒ 'a::banach
  assumes f: f integrable-on {a..b}
      and x: x ∈ {a..b}
      and fx: continuous (at x within {a..b}) f
  shows ((λu. integral {a..u} f) has-vector-derivative f x) (at x within {a..b})
proof -
  let ?I = λa b. integral {a..b} f
  { fix e::real
    assume e > 0
    obtain d where d>0 and d: ⋀x'. ⟦x' ∈ {a..b}; |x' - x| < d⟧ ⟹ norm(f x'
- f x) ≤ e
      using ‹e>0› fx by (auto simp: continuous-within-eps-delta dist-norm less-imp-le)
    have norm (integral {a..y} f - integral {a..x} f - (y - x) *R f x) ≤ e * |y
- x|
          if y: y ∈ {a..b} and yx: |y - x| < d for y
    proof (cases y < x)
      case False
      have f integrable-on {a..y}
```

```
      using f y by (simp add: integrable-subinterval-real)
    then have Idiff: ?I a y − ?I a x = ?I x y
      using False x by (simp add: algebra-simps integral-combine)
     have fux-int: ((λu. f u − f x) has-integral integral {x..y} f − (y − x) *R f
x) {x..y}
      apply (rule has-integral-sub)
    using x y apply (force intro: integrable-integral [OF integrable-subinterval-real
[OF f]])
      using has-integral-const-real [of f x x y] False
      apply (simp add: )
      done
    show ?thesis
      using False
      apply (simp add: abs-eq-content del: content-real-if)
      apply (rule has-integral-bound-real[where f=(λu. f u − f x)])
      using yx False d x y ‹e>0› apply (auto simp add: Idiff fux-int)
      done
  next
    case True
    have f integrable-on {a..x}
      using f x by (simp add: integrable-subinterval-real)
    then have Idiff: ?I a x − ?I a y = ?I y x
      using True x y by (simp add: algebra-simps integral-combine)
     have fux-int: ((λu. f u − f x) has-integral integral {y..x} f − (x − y) *R f
x) {y..x}
      apply (rule has-integral-sub)
    using x y apply (force intro: integrable-integral [OF integrable-subinterval-real
[OF f]])
      using has-integral-const-real [of f x y x] True
      apply (simp add: )
      done
    have norm (integral {a..x} f − integral {a..y} f − (x − y) *R f x) ≤ e * |y
− x|
      using True
      apply (simp add: abs-eq-content del: content-real-if)
      apply (rule has-integral-bound-real[where f=(λu. f u − f x)])
      using yx True d x y ‹e>0› apply (auto simp add: Idiff fux-int)
      done
    then show ?thesis
      by (simp add: algebra-simps norm-minus-commute)
  qed
   then have ∃ d>0. ∀ y∈{a..b}. |y − x| < d ⟶ norm (integral {a..y} f −
integral {a..x} f − (y − x) *R f x) ≤ e * |y − x|
    using ‹d>0› by blast
 }
 then show ?thesis
  by (simp add: has-vector-derivative-def has-derivative-within-alt bounded-linear-scaleR-left)
qed
```

lemma *integral-has-vector-derivative*:
fixes f :: *real* ⇒ *'a*::*banach*
assumes *continuous-on* {a .. b} f
and $x \in \{a$ .. $b\}$
shows ((λu. *integral* {a .. u} f) *has-vector-derivative* $f(x)$) (*at* x *within* {a .. b})
apply (*rule integral-has-vector-derivative-continuous-at* [*OF integrable-continuous-real*])
using *assms*
apply (*auto simp*: *continuous-on-eq-continuous-within*)
done

lemma *antiderivative-continuous*:
fixes q b :: *real*
assumes *continuous-on* {a .. b} f
obtains g **where** ∀ x∈{a .. b}. (g *has-vector-derivative* (f x::-::*banach*)) (*at* x *within* {a .. b})
apply (*rule that*)
apply *rule*
using *integral-has-vector-derivative*[*OF assms*]
apply *auto*
done

35.47 Combined fundamental theorem of calculus.

lemma *antiderivative-integral-continuous*:
fixes f :: *real* ⇒ *'a*::*banach*
assumes *continuous-on* {a .. b} f
obtains g **where** ∀ u∈{a .. b}. ∀ v ∈ {a .. b}. $u \leq v \longrightarrow$ (f *has-integral* (g v − g u)) {u .. v}
proof −
from *antiderivative-continuous*[*OF assms*] **guess** g **. note** g=*this*
show *?thesis*
apply (*rule that*[*of g*])
apply *safe*
proof *goal-cases*
case *prems*: (*1 u v*)
have ∀ x∈*cbox u v*. (g *has-vector-derivative f x*) (*at x within cbox u v*)
apply *rule*
apply (*rule has-vector-derivative-within-subset*)
apply (*rule g*[*rule-format*])
using *prems*(*1*,*2*)
apply *auto*
done
then show *?case*
using *fundamental-theorem-of-calculus*[*OF prems*(*3*), *of g f*] **by** *auto*
qed
qed

35.48 General "twiddling" for interval-to-interval function image.

lemma *has-integral-twiddle*:
assumes $0 < r$
and $\forall x.\ h(g\ x) = x$
and $\forall x.\ g(h\ x) = x$
and $\forall x.\ \textit{continuous}\ (\textit{at}\ x)\ g$
and $\forall u\ v.\ \exists w\ z.\ g\ `\ \textit{cbox}\ u\ v = \textit{cbox}\ w\ z$
and $\forall u\ v.\ \exists w\ z.\ h\ `\ \textit{cbox}\ u\ v = \textit{cbox}\ w\ z$
and $\forall u\ v.\ \textit{content}(g\ `\ \textit{cbox}\ u\ v) = r * \textit{content}\ (\textit{cbox}\ u\ v)$
and $(f\ \textit{has-integral}\ i)\ (\textit{cbox}\ a\ b)$
shows $((\lambda x.\ f(g\ x))\ \textit{has-integral}\ (1\ /\ r) *_R i)\ (h\ `\ \textit{cbox}\ a\ b)$
proof −
show *?thesis* **when** $*$: $\textit{cbox}\ a\ b \neq \{\} \Longrightarrow$ *?thesis*
apply *cases*
defer
apply (*rule* $*$)
apply *assumption*
proof *goal-cases*
case *prems*: *1*
then show *?thesis*
unfolding *prems assms*(*8*)[*unfolded prems has-integral-empty-eq*] **by** *auto*
qed
assume $\textit{cbox}\ a\ b \neq \{\}$
from *assms*(*6*)[*rule-format*,*of a b*] **guess** *w z* **by** (*elim exE*) **note** *wz*=*this*
have *inj*: *inj g inj h*
unfolding *inj-on-def*
apply *safe*
apply(*rule-tac*[!] *ccontr*)
using *assms*(*2*)
apply(*erule-tac x*=*x* **in** *allE*)
using *assms*(*2*)
apply(*erule-tac x*=*y* **in** *allE*)
defer
using *assms*(*3*)
apply (*erule-tac x*=*x* **in** *allE*)
using *assms*(*3*)
apply(*erule-tac x*=*y* **in** *allE*)
apply *auto*
done
show *?thesis*
unfolding *has-integral-def has-integral-compact-interval-def*
apply (*subst if-P*)
apply *rule*
apply *rule*
apply (*rule wz*)
proof *safe*
fix *e* :: *real*
assume *e*: $e > 0$

```
  with assms(1) have e * r > 0 by simp
   from assms(8)[unfolded has-integral,rule-format,OF this] guess d by (elim
exE conjE) note d=this[rule-format]
  def d' ≡ λx. {y. g y ∈ d (g x)}
  have d': ⋀x. d' x = {y. g y ∈ (d (g x))}
    unfolding d'-def ..
  show ∃d. gauge d ∧ (∀p. p tagged-division-of h ‘ cbox a b ∧ d fine p ⟶ norm
((∑(x, k)∈p. content k *R f (g x)) − (1 / r) *R i) < e)
  proof (rule-tac x=d' in exI, safe)
    show gauge d'
      using d(1)
      unfolding gauge-def d'
      using continuous-open-preimage-univ[OF assms(4)]
      by auto
    fix p
    assume as: p tagged-division-of h ‘ cbox a b d' fine p
    note p = tagged-division-ofD[OF as(1)]
   have (λ(x, k). (g x, g ‘ k)) ‘ p tagged-division-of (cbox a b) ∧ d fine (λ(x, k).
(g x, g ‘ k)) ‘ p
      unfolding tagged-division-of
    proof safe
      show finite ((λ(x, k). (g x, g ‘ k)) ‘ p)
        using as by auto
      show d fine (λ(x, k). (g x, g ‘ k)) ‘ p
        using as(2) unfolding fine-def d' by auto
      fix x k
      assume xk[intro]: (x, k) ∈ p
      show g x ∈ g ‘ k
        using p(2)[OF xk] by auto
      show ∃u v. g ‘ k = cbox u v
        using p(4)[OF xk] using assms(5−6) by auto
      {
        fix y
        assume y ∈ k
        then show g y ∈ cbox a b g y ∈ cbox a b
          using p(3)[OF xk,unfolded subset-eq,rule-format,of h (g y)]
          using assms(2)[rule-format,of y]
          unfolding inj-image-mem-iff[OF inj(2)]
          by auto
      }
      fix x' k'
      assume xk': (x', k') ∈ p
      fix z
      assume z ∈ interior (g ‘ k) and z ∈ interior (g ‘ k')
      then have *: interior (g ‘ k) ∩ interior (g ‘ k') ≠ {}
        by auto
      have same: (x, k) = (x', k')
        apply −
        apply (rule ccontr)
```

```
      apply (drule p(5)[OF xk xk'])
     proof −
      assume as: interior k ∩ interior k' = {}
      from nonempty-witness[OF *] guess z .
      then have z ∈ g ‘ (interior k ∩ interior k')
       using interior-image-subset[OF assms(4) inj(1)]
       unfolding image-Int[OF inj(1)]
       by auto
      then show False
       using as by blast
     qed
     then show g x = g x'
      by auto
     {
      fix z
      assume z ∈ k
      then show g z ∈ g ‘ k'
       using same by auto
     }
     {
      fix z
      assume z ∈ k'
      then show g z ∈ g ‘ k
       using same by auto
     }
    next
     fix x
     assume x ∈ cbox a b
     then have h x ∈ ⋃{k. ∃x. (x, k) ∈ p}
      using p(6) by auto
     then guess X unfolding Union-iff .. note X=this
     from this(1) guess y unfolding mem-Collect-eq ..
     then show x ∈ ⋃{k. ∃x. (x, k) ∈ (λ(x, k). (g x, g ‘ k)) ‘ p}
      apply −
      apply (rule-tac X=g ‘ X in UnionI)
      defer
      apply (rule-tac x=h x in image-eqI)
      using X(2) assms(3)[rule-format,of x]
      apply auto
      done
    qed
     note ** = d(2)[OF this]
     have *: inj-on (λ(x, k). (g x, g ‘ k)) p
      using inj(1) unfolding inj-on-def by fastforce
     have (∑(x, k)∈(λ(x, k). (g x, g ‘ k)) ‘ p. content k *R f x) − i = r *R
(∑(x, k)∈p. content k *R f (g x)) − i (is ?l = -)
      using assms(7)
      unfolding algebra-simps add-left-cancel scaleR-right.setsum
      by (subst setsum.reindex-bij-betw[symmetric, where h=λ(x, k). (g x, g ‘
```

```
k) and S=p])
          (auto intro!: * setsum.cong simp: bij-betw-def dest!: p(4))
     also have ... = r *R ((∑(x, k)∈p. content k *R f (g x)) - (1 / r) *R i)
(is - = ?r)
       unfolding scaleR-diff-right scaleR-scaleR
       using assms(1)
       by auto
     finally have *: ?l = ?r .
     show norm ((∑(x, k)∈p. content k *R f (g x)) - (1 / r) *R i) < e
       using **
       unfolding *
       unfolding norm-scaleR
       using assms(1)
       by (auto simp add:field-simps)
   qed
  qed
qed
```

35.49 Special case of a basic affine transformation.

```
lemma interval-image-affinity-interval:
  ∃u v. (λx. m *R (x::'a::euclidean-space) + c) ' cbox a b = cbox u v
  unfolding image-affinity-cbox
  by auto

lemma content-image-affinity-cbox:
  content((λx::'a::euclidean-space. m *R x + c) ' cbox a b) =
    |m| ^ DIM('a) * content (cbox a b) (is ?l = ?r)
proof -
  {
    presume *: cbox a b ≠ {} ⟹ ?thesis
    show ?thesis
      apply cases
      apply (rule *)
      apply assumption
      unfolding not-not
      using content-empty
      apply auto
      done
  }
  assume as: cbox a b ≠ {}
  show ?thesis
  proof (cases m ≥ 0)
    case True
    with as have cbox (m *R a + c) (m *R b + c) ≠ {}
      unfolding box-ne-empty
      apply (intro ballI)
      apply (erule-tac x=i in ballE)
      apply (auto simp: inner-simps intro!: mult-left-mono)
```

```
    done
  moreover from True have *: ⋀i. (m *R b + c) · i − (m *R a + c) · i = m
*R (b − a) · i
    by (simp add: inner-simps field-simps)
  ultimately show ?thesis
    by (simp add: image-affinity-cbox True content-cbox'
      setprod.distrib setprod-constant inner-diff-left)
 next
  case False
  with as have cbox (m *R b + c) (m *R a + c) ≠ {}
    unfolding box-ne-empty
    apply (intro ballI)
    apply (erule-tac x=i in ballE)
    apply (auto simp: inner-simps intro!: mult-left-mono)
    done
  moreover from False have *: ⋀i. (m *R a + c) · i − (m *R b + c) · i =
(−m) *R (b − a) · i
    by (simp add: inner-simps field-simps)
  ultimately show ?thesis using False
    by (simp add: image-affinity-cbox content-cbox'
      setprod.distrib[symmetric] setprod-constant[symmetric] inner-diff-left)
 qed
qed

lemma has-integral-affinity:
 fixes a :: 'a::euclidean-space
 assumes (f has-integral i) (cbox a b)
    and m ≠ 0
 shows ((λx. f(m *R x + c)) has-integral ((1 / (|m| ^ DIM('a))) *R i)) ((λx.
(1 / m) *R x + −((1 / m) *R c)) ' cbox a b)
 apply (rule has-integral-twiddle)
 using assms
 apply (safe intro!: interval-image-affinity-interval content-image-affinity-cbox)
 apply (rule zero-less-power)
 unfolding scaleR-right-distrib
 apply auto
 done

lemma integrable-affinity:
 assumes f integrable-on cbox a b
  and m ≠ 0
 shows (λx. f(m *R x + c)) integrable-on ((λx. (1 / m) *R x + −((1/m) *R
c)) ' cbox a b)
 using assms
 unfolding integrable-on-def
 apply safe
 apply (drule has-integral-affinity)
 apply auto
 done
```

lemmas *has-integral-affinity01 = has-integral-affinity* [*of - - 0 1*::*real*, *simplified*]

35.50 Special case of stretching coordinate axes separately.

lemma *image-stretch-interval*:
(λ*x*. ∑ *k*∈*Basis*. (*m k* ∗ (*x*•*k*)) ∗$_R$ *k*) ' *cbox a* (*b*::′*a*::*euclidean-space*) =
(*if* (*cbox a b*) = {} *then* {} *else*
cbox (∑ *k*∈*Basis*. (*min* (*m k* ∗ (*a*•*k*)) (*m k* ∗ (*b*•*k*))) ∗$_R$ *k*::′*a*)
(∑ *k*∈*Basis*. (*max* (*m k* ∗ (*a*•*k*)) (*m k* ∗ (*b*•*k*))) ∗$_R$ *k*))
proof *cases*
assume ∗: *cbox a b* ≠ {}
show *?thesis*
unfolding *box-ne-empty if-not-P*[*OF* ∗]
apply (*simp add*: *cbox-def image-Collect set-eq-iff euclidean-eq-iff* [**where** ′*a*=′*a*] *ball-conj-distrib*[*symmetric*])
apply (*subst choice-Basis-iff*[*symmetric*])
proof (*intro allI ball-cong refl*)
fix *x i* :: ′*a* **assume** *i* ∈ *Basis*
with ∗ **have** *a-le-b*: *a* · *i* ≤ *b* · *i*
unfolding *box-ne-empty* **by** *auto*
show (∃ *xa*. *x* · *i* = *m i* ∗ *xa* ∧ *a* · *i* ≤ *xa* ∧ *xa* ≤ *b* · *i*) ⟷
min (*m i* ∗ (*a* · *i*)) (*m i* ∗ (*b* · *i*)) ≤ *x* · *i* ∧ *x* · *i* ≤ *max* (*m i* ∗ (*a* · *i*)) (*m i* ∗ (*b* · *i*))
proof (*cases m i* = *0*)
case *True*
with *a-le-b* **show** *?thesis* **by** *auto*
next
case *False*
then have ∗: ⋀*a b*. *a* = *m i* ∗ *b* ⟷ *b* = *a* / *m i*
by (*auto simp add*: *field-simps*)
from *False* **have**
min (*m i* ∗ (*a* · *i*)) (*m i* ∗ (*b* · *i*)) = (*if* 0 < *m i then m i* ∗ (*a* · *i*) *else m i* ∗ (*b* · *i*))
max (*m i* ∗ (*a* · *i*)) (*m i* ∗ (*b* · *i*)) = (*if* 0 < *m i then m i* ∗ (*b* · *i*) *else m i* ∗ (*a* · *i*))
using *a-le-b* **by** (*auto simp*: *min-def max-def mult-le-cancel-left*)
with *False* **show** *?thesis* **using** *a-le-b*
unfolding ∗ **by** (*auto simp add*: *le-divide-eq divide-le-eq ac-simps*)
qed
qed
qed *simp*

lemma *interval-image-stretch-interval*:
∃ *u v*. (λ*x*. ∑ *k*∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*) ' *cbox a* (*b*::′*a*::*euclidean-space*) = *cbox u* (*v*::′*a*::*euclidean-space*)
unfolding *image-stretch-interval* **by** *auto*

lemma *content-image-stretch-interval*:

content ((λx::'a::*euclidean-space*. (∑ k∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*)::'a) ‘ *cbox a b*) =
|setprod m Basis| ∗ *content* (*cbox a b*)
proof (*cases cbox a b* = {})
case *True*
then show *?thesis*
unfolding *content-def image-is-empty image-stretch-interval if-P*[*OF True*] **by** *auto*
next
case *False*
then have (λx. (∑ k∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*)) ‘ *cbox a b* ≠ {}
by *auto*
then show *?thesis*
using *False*
unfolding *content-def image-stretch-interval*
apply −
unfolding *interval-bounds' if-not-P*
unfolding *abs-setprod setprod.distrib*[*symmetric*]
apply (*rule setprod.cong*)
apply (*rule refl*)
unfolding *lessThan-iff*
apply (*simp only*: *inner-setsum-left-Basis*)
proof −
fix *i* :: 'a
assume *i*: *i* ∈ *Basis*
have (*m i* < *0* ∨ *m i* > *0*) ∨ *m i* = *0*
by *auto*
then show *max* (*m i* ∗ (*a* • *i*)) (*m i* ∗ (*b* • *i*)) − *min* (*m i* ∗ (*a* • *i*)) (*m i* ∗ (*b* • *i*)) =
|m i| ∗ (*b* • *i* − *a* • *i*)
apply −
apply (*erule disjE*)+
unfolding *min-def max-def*
using *False*[*unfolded box-ne-empty*,*rule-format*,*of i*] *i*
apply (*auto simp add*:*field-simps not-le mult-le-cancel-left-neg mult-le-cancel-left-pos*)
done
qed
qed

lemma *has-integral-stretch*:
fixes *f* :: 'a::*euclidean-space* ⇒ 'b::*real-normed-vector*
assumes (*f has-integral i*) (*cbox a b*)
and ∀ *k*∈*Basis*. *m k* ≠ *0*
shows ((λx. *f* (∑ k∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*)) *has-integral*
((*1* / *|setprod m Basis|*) ∗$_R$ *i*)) ((λx. (∑ k∈*Basis*. (*1* / *m k* ∗ (*x*•*k*))∗$_R$ *k*)) ‘ *cbox a b*)
apply (*rule has-integral-twiddle*[**where** *f*=*f*])
unfolding *zero-less-abs-iff content-image-stretch-interval*
unfolding *image-stretch-interval empty-as-interval euclidean-eq-iff* [**where** 'a='a]

using *assms*
proof −
show ∀ *y*::′*a*. *continuous* (*at y*) (λ*x*. (∑ *k*∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*))
apply *rule*
apply (*rule linear-continuous-at*)
unfolding *linear-linear*
unfolding *linear-iff inner-simps euclidean-eq-iff* [**where** ′*a*=′*a*]
apply (*auto simp add*: *field-simps*)
done
qed *auto*

lemma *integrable-stretch*:
fixes *f* :: ′*a*::*euclidean-space* ⇒ ′*b*::*real-normed-vector*
assumes *f integrable-on cbox a b*
and ∀ *k*∈*Basis*. *m k* ≠ *0*
shows (λ*x*::′*a*. *f* (∑ *k*∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*)) *integrable-on*
((λ*x*. ∑ *k*∈*Basis*. (*1* / *m k* ∗ (*x*•*k*))∗$_R$ *k*) ‘ *cbox a b*)
using *assms*
unfolding *integrable-on-def*
apply −
apply (*erule exE*)
apply (*drule has-integral-stretch*)
apply *assumption*
apply *auto*
done

35.51 even more special cases.

lemma *uminus-interval-vector*[*simp*]:
fixes *a b* :: ′*a*::*euclidean-space*
shows *uminus* ‘ *cbox a b* = *cbox* (−*b*) (−*a*)
apply (*rule set-eqI*)
apply *rule*
defer
unfolding *image-iff*
apply (*rule-tac x*=−*x* **in** *bexI*)
apply (*auto simp add*:*minus-le-iff le-minus-iff mem-box*)
done

lemma *has-integral-reflect-lemma*[*intro*]:
assumes (*f has-integral i*) (*cbox a b*)
shows ((λ*x*. *f*(−*x*)) *has-integral i*) (*cbox* (−*b*) (−*a*))
using *has-integral-affinity*[*OF assms*, *of* −*1 0*]
by *auto*

lemma *has-integral-reflect-lemma-real*[*intro*]:
assumes (*f has-integral i*) {*a* .. *b*::*real*}
shows ((λ*x*. *f*(−*x*)) *has-integral i*) {−*b* .. −*a*}
using *assms*

unfolding *box-real*[*symmetric*]
by (*rule has-integral-reflect-lemma*)

lemma *has-integral-reflect*[*simp*]:
((λ*x*. *f* (−*x*)) *has-integral i*) (*cbox* (−*b*) (−*a*)) ⟷ (*f has-integral i*) (*cbox a b*)
apply *rule*
apply (*drule-tac*[!] *has-integral-reflect-lemma*)
apply *auto*
done

lemma *integrable-reflect*[*simp*]: (λ*x*. *f*(−*x*)) *integrable-on cbox* (−*b*) (−*a*) ⟷ *f integrable-on cbox a b*
unfolding *integrable-on-def* **by** *auto*

lemma *integrable-reflect-real*[*simp*]: (λ*x*. *f*(−*x*)) *integrable-on* {−*b* .. −*a*} ⟷ *f integrable-on* {*a* .. *b*::*real*}
unfolding *box-real*[*symmetric*]
by (*rule integrable-reflect*)

lemma *integral-reflect*[*simp*]: *integral* (*cbox* (−*b*) (−*a*)) (λ*x*. *f* (−*x*)) = *integral* (*cbox a b*) *f*
unfolding *integral-def* **by** *auto*

lemma *integral-reflect-real*[*simp*]: *integral* {−*b* .. −*a*} (λ*x*. *f* (−*x*)) = *integral* {*a* .. *b*::*real*} *f*
unfolding *box-real*[*symmetric*]
by (*rule integral-reflect*)

35.52 Stronger form of FCT; quite a tedious proof.

lemma *bgauge-existence-lemma*: (∀ *x*∈*s*. ∃ *d*::*real*. 0 < *d* ∧ *q d x*) ⟷ (∀ *x*. ∃ *d*>0. *x*∈*s* ⟶ *q d x*)
by (*meson zero-less-one*)

lemma *additive-tagged-division-1′*:
fixes *f* :: *real* ⇒ *'a*::*real-normed-vector*
assumes *a* ≤ *b*
and *p tagged-division-of* {*a*..*b*}
shows *setsum* (λ(*x*,*k*). *f* (*Sup k*) − *f*(*Inf k*)) *p* = *f b* − *f a*
using *additive-tagged-division-1*[*OF* - *assms*(*2*), *of f*]
using *assms*(*1*)
by *auto*

lemma *split-minus*[*simp*]: (λ(*x*, *k*). *f x k*) *x* − (λ(*x*, *k*). *g x k*) *x* = (λ(*x*, *k*). *f x k* − *g x k*) *x*
by (*simp add*: *split-def*)

lemma *norm-triangle-le-sub*: *norm x* + *norm y* ≤ *e* ⟹ *norm* (*x* − *y*) ≤ *e*
apply (*subst*(*asm*)(*2*) *norm-minus-cancel*[*symmetric*])

apply (*drule norm-triangle-le*)
apply (*auto simp add*: *algebra-simps*)
done

lemma *fundamental-theorem-of-calculus-interior*:
fixes f :: *real* ⇒ 'a::*real-normed-vector*
assumes $a \leq b$
and *continuous-on* $\{a\ ..\ b\}$ f
and $\forall x{\in}\{a <..< b\}$. ($f$ *has-vector-derivative* $f'(x)$) (*at* x)
shows (f' *has-integral* ($f\ b - f\ a$)) $\{a\ ..\ b\}$
proof −
{
presume ∗: $a < b \Longrightarrow$ *?thesis*
show *?thesis*
proof (*cases* $a < b$)
case *True*
then show *?thesis* **by** (*rule* ∗)
next
case *False*
then have $a = b$
using *assms*(*1*) **by** *auto*
then have ∗: *cbox* $a\ b = \{b\}$ $f\ b - f\ a = 0$
by (*auto simp add*: *order-antisym*)
show *?thesis*
unfolding ∗(*2*)
unfolding *content-eq-0*
using ∗ ‹$a = b$›
by (*auto simp*: *ex-in-conv*)
qed
}
assume *ab*: $a < b$
let *?P* = $\lambda e.\ \exists d.$ *gauge* $d \wedge (\forall p.$ p *tagged-division-of* $\{a\ ..\ b\} \wedge d$ *fine* $p \longrightarrow$
norm $((\sum (x, k){\in}p.$ *content* $k *_R f'\ x) - (f\ b - f\ a)) \leq e *$ *content* $\{a\ ..\ b\})$
{ presume $\bigwedge e.\ e > 0 \Longrightarrow$ *?P* e **then show** *?thesis* **unfolding** *has-integral-factor-content-real*
by *auto* **}**
fix e :: *real*
assume *e*: $e > 0$
note *assms*(*3*)[*unfolded has-vector-derivative-def has-derivative-at-alt ball-conj-distrib*]
note *conjunctD2*[*OF this*]
note *bounded=this*(*1*) **and** *this*(*2*)
from *this*(*2*) **have** $\forall x{\in}box\ a\ b.\ \exists d{>}0.\ \forall y.$ *norm* $(y - x) < d \longrightarrow$
norm $(f\ y - f\ x - (y - x) *_R f'\ x) \leq e/2 *$ *norm* $(y - x)$
apply −
apply *safe*
apply (*erule-tac* $x{=}x$ **in** *ballE*)
apply (*erule-tac* $x{=}e/2$ **in** *allE*)
using e
apply *auto*
done

note *this*[*unfolded bgauge-existence-lemma*]
from *choice*[*OF this*] **guess** d **..**
note *conjunctD2*[*OF this*[*rule-format*]]
note d = *this*[*rule-format*]
have *bounded* (f ‘ *cbox a b*)
apply (*rule compact-imp-bounded compact-continuous-image*)+
using *compact-cbox assms*
apply *auto*
done
from *this*[*unfolded bounded-pos*] **guess** B **..** **note** B = *this*[*rule-format*]

have $\exists da.\ 0 < da \wedge (\forall c.\ a \leq c \wedge \{a\ ..\ c\} \subseteq \{a\ ..\ b\} \wedge \{a\ ..\ c\} \subseteq \mathit{ball}\ a\ da \longrightarrow$
norm (*content* $\{a\ ..\ c\}$ $*_R$ f' a − (f c − f a)) ≤ (e * (b − a)) / *4*)
proof −
have $a \in \{a\ ..\ b\}$
using *ab* **by** *auto*
note *assms*(*2*)[*unfolded continuous-on-eq-continuous-within*,*rule-format*,*OF this*]
note $*$ = *this*[*unfolded continuous-within Lim-within*,*rule-format*]
have (e * (b − a)) / *8* > *0*
using *e ab* **by** (*auto simp add*: *field-simps*)
from $*$[*OF this*] **guess** k **..** **note** k = *conjunctD2*[*OF this*,*rule-format*]
have $\exists l.\ 0 < l \wedge$ *norm*(l $*_R$ f' a) ≤ (e * (b − a)) / *8*
proof (*cases* f' a = *0*)
case *True*
thus *?thesis* **using** *ab e* **by** *auto*
next
case *False*
then show *?thesis*
apply (*rule-tac x*=(e * (b − a)) / *8* / *norm* (f' a) **in** *exI*)
using *ab e*
apply (*auto simp add*: *field-simps*)
done
qed
then guess l **..** **note** l = *conjunctD2*[*OF this*]
show *?thesis*
apply (*rule-tac x*=*min k l* **in** *exI*)
apply *safe*
unfolding *min-less-iff-conj*
apply *rule*
apply (*rule l k*)+
proof −
fix c
assume *as*: $a \leq c$ $\{a\ ..\ c\} \subseteq \{a\ ..\ b\}$ $\{a\ ..\ c\} \subseteq \mathit{ball}\ a$ (*min k l*)
note *as'* = *this*[*unfolded subset-eq Ball-def mem-ball dist-real-def mem-box*]
have *norm* ((c − a) $*_R$ f' a − (f c − f a)) ≤ *norm* ((c − a) $*_R$ f' a) + *norm* (f c − f a)
by (*rule norm-triangle-ineq4*)
also have ... ≤ e * (b − a) / *8* + e * (b − a) / *8*
proof (*rule add-mono*)

```
      have |c − a| ≤ |l|
        using as' by auto
      then show norm ((c − a) *R f' a) ≤ e * (b − a) / 8
        apply −
        apply (rule order-trans[OF - l(2)])
        unfolding norm-scaleR
        apply (rule mult-right-mono)
        apply auto
        done
    next
      show norm (f c − f a) ≤ e * (b − a) / 8
        apply (rule less-imp-le)
        apply (cases a = c)
        defer
        apply (rule k(2)[unfolded dist-norm])
        using as' e ab
        apply (auto simp add: field-simps)
        done
    qed
    finally show norm (content {a .. c} *R f' a − (f c − f a)) ≤ e * (b − a) /
4
      unfolding content-real[OF as(1)] by auto
  qed
 qed
 then guess da .. note da=conjunctD2[OF this,rule-format]

 have ∃ db>0. ∀ c≤b. {c .. b} ⊆ {a .. b} ∧ {c .. b} ⊆ ball b db ⟶
   norm (content {c .. b} *R f' b − (f b − f c)) ≤ (e * (b − a)) / 4
 proof −
  have b ∈ {a .. b}
    using ab by auto
  note assms(2)[unfolded continuous-on-eq-continuous-within,rule-format,OF this]
  note * = this[unfolded continuous-within Lim-within,rule-format] have (e * (b
− a)) / 8 > 0
    using e ab by (auto simp add: field-simps)
  from *[OF this] guess k .. note k = conjunctD2[OF this,rule-format]
  have ∃ l. 0 < l ∧ norm (l *R f' b) ≤ (e * (b − a)) / 8
  proof (cases f' b = 0)
    case True
    thus ?thesis using ab e by auto
  next
    case False
    then show ?thesis
      apply (rule-tac x=(e * (b − a)) / 8 / norm (f' b) in exI)
      using ab e
      apply (auto simp add: field-simps)
      done
  qed
  then guess l .. note l = conjunctD2[OF this]
```

```
    show ?thesis
      apply (rule-tac x=min k l in exI)
      apply safe
      unfolding min-less-iff-conj
      apply rule
      apply (rule l k)+
    proof −
      fix c
      assume as: c ≤ b {c..b} ⊆ {a..b} {c..b} ⊆ ball b (min k l)
      note as' = this[unfolded subset-eq Ball-def mem-ball dist-real-def mem-box]
     have norm ((b − c) *R f' b − (f b − f c)) ≤ norm ((b − c) *R f' b) + norm
(f b − f c)
        by (rule norm-triangle-ineq4)
      also have ... ≤ e * (b − a) / 8 + e * (b − a) / 8
      proof (rule add-mono)
        have |c − b| ≤ |l|
          using as' by auto
        then show norm ((b − c) *R f' b) ≤ e * (b − a) / 8
          apply −
          apply (rule order-trans[OF - l(2)])
          unfolding norm-scaleR
          apply (rule mult-right-mono)
          apply auto
          done
      next
        show norm (f b − f c) ≤ e * (b − a) / 8
          apply (rule less-imp-le)
          apply (cases b = c)
          defer
          apply (subst norm-minus-commute)
          apply (rule k(2)[unfolded dist-norm])
          using as' e ab
          apply (auto simp add: field-simps)
          done
      qed
     finally show norm (content {c .. b} *R f' b − (f b − f c)) ≤ e * (b − a) / 4
        unfolding content-real[OF as(1)] by auto
    qed
  qed
  then guess db .. note db=conjunctD2[OF this,rule-format]

  let ?d = (λx. ball x (if x=a then da else if x=b then db else d x))
  show ?P e
    apply (rule-tac x=?d in exI)
  proof (safe, goal-cases)
    case 1
    show ?case
      apply (rule gauge-ball-dependent)
      using ab db(1) da(1) d(1)
```

```
      apply auto
      done
  next
    case as: (2 p)
    let ?A = {t. fst t ∈ {a, b}}
    note p = tagged-division-ofD[OF as(1)]
    have pA: p = (p ∩ ?A) ∪ (p - ?A) finite (p ∩ ?A) finite (p - ?A) (p ∩ ?A)
∩ (p - ?A) = {}
      using as by auto
    note ∗ = additive-tagged-division-1'[OF assms(1) as(1), symmetric]
    have ∗∗: ⋀n1 s1 n2 s2::real. n2 ≤ s2 / 2 ⟹ n1 - s1 ≤ s2 / 2 ⟹ n1 +
n2 ≤ s1 + s2
      by arith
    show ?case
    unfolding content-real[OF assms(1)] and ∗[of λx. x] ∗[of f] setsum-subtractf[symmetric]
split-minus
      unfolding setsum-right-distrib
      apply (subst(2) pA)
      apply (subst pA)
      unfolding setsum.union-disjoint[OF pA(2-)]
    proof (rule norm-triangle-le, rule ∗∗, goal-cases)
      case 1
      show ?case
        apply (rule order-trans)
        apply (rule setsum-norm-le)
        defer
        apply (subst setsum-divide-distrib)
        apply (rule order-refl)
        apply safe
        apply (unfold not-le o-def split-conv fst-conv)
      proof (rule ccontr)
        fix x k
        assume xk: (x, k) ∈ p
          e ∗ (Sup k -  Inf k) / 2 <
            norm (content k ∗R f' x - (f (Sup k) - f (Inf k)))
        from p(4)[OF this(1)] guess u v by (elim exE) note k=this
        then have u ≤ v and uv: {u, v} ⊆ cbox u v
          using p(2)[OF xk(1)] by auto
          note result = xk(2)[unfolded k box-real interval-bounds-real[OF this(1)]
content-real[OF this(1)]]

        assume as': x ≠ a x ≠ b
        then have x ∈ box a b
          using p(2-3)[OF xk(1)] by (auto simp: mem-box)
        note ∗ = d(2)[OF this]
        have norm ((v - u) ∗R f' (x) - (f (v) - f (u))) =
          norm ((f (u) - f (x) - (u - x) ∗R f' (x)) - (f (v) - f (x) - (v - x)
∗R f' (x)))
          apply (rule arg-cong[of - - norm])
```

```
      unfolding scaleR-left.diff
      apply auto
      done
    also have ... ≤ e / 2 * norm (u − x) + e / 2 * norm (v − x)
      apply (rule norm-triangle-le-sub)
      apply (rule add-mono)
      apply (rule-tac[!] *)
      using fineD[OF as(2) xk(1)] as'
      unfolding k subset-eq
      apply −
      apply (erule-tac x=u in ballE)
      apply (erule-tac[3] x=v in ballE)
      using uv
      apply (auto simp:dist-real-def)
      done
    also have ... ≤ e / 2 * norm (v − u)
      using p(2)[OF xk(1)]
      unfolding k
      by (auto simp add: field-simps)
    finally have e * (v − u) / 2 < e * (v − u) / 2
      apply −
      apply (rule less-le-trans[OF result])
      using uv
      apply auto
      done
    then show False by auto
  qed
next
  have *: ⋀x s1 s2::real. 0 ≤ s1 ⟹ x ≤ (s1 + s2) / 2 ⟹ x − s1 ≤ s2 / 2
    by auto
  case 2
  show ?case
    apply (rule *)
    apply (rule setsum-nonneg)
    apply rule
    apply (unfold split-paired-all split-conv)
    defer
    unfolding setsum.union-disjoint[OF pA(2−),symmetric] pA(1)[symmetric]
    unfolding setsum-right-distrib[symmetric]
    apply (subst additive-tagged-division-1[OF - as(1)])
    apply (rule assms)
  proof −
    fix x k
    assume (x, k) ∈ p ∩ {t. fst t ∈ {a, b}}
    note xk=IntD1[OF this]
    from p(4)[OF this] guess u v by (elim exE) note uv=this
    with p(2)[OF xk] have cbox u v ≠ {}
      by auto
    then show 0 ≤ e * ((Sup k) − (Inf k))
```

unfolding *uv* **using** *e* **by** (*auto simp add*: *field-simps*)
next
have ∗: ⋀*s f t e*. *setsum f s* = *setsum f t* ⟹ *norm* (*setsum f t*) ≤ *e* ⟹ *norm* (*setsum f s*) ≤ *e*
by *auto*
show *norm* (∑ (*x*, *k*)∈*p* ∩ *?A*. *content k* ∗$_R$ *f*′ *x* −
(*f* ((*Sup k*)) − *f* ((*Inf k*)))) ≤ *e* ∗ (*b* − *a*) / *2*
apply (*rule* ∗[**where** *t1*=*p* ∩ {*t*. *fst t* ∈ {*a*, *b*} ∧ *content*(*snd t*) ≠ *0*}])
apply (*rule setsum.mono-neutral-right*[*OF pA*(*2*)])
defer
apply *rule*
unfolding *split-paired-all split-conv o-def*
proof *goal-cases*
fix *x k*
assume (*x*, *k*) ∈ *p* ∩ {*t*. *fst t* ∈ {*a*, *b*}} − *p* ∩ {*t*. *fst t* ∈ {*a*, *b*} ∧ *content* (*snd t*) ≠ *0*}
then have *xk*: (*x*, *k*) ∈ *p content k* = *0*
by *auto*
from *p*(*4*)[*OF xk*(*1*)] **guess** *u v* **by** (*elim exE*) **note** *uv*=*this*
have *k* ≠ {}
using *p*(*2*)[*OF xk*(*1*)] **by** *auto*
then have ∗: *u* = *v*
using *xk*
unfolding *uv content-eq-0 box-eq-empty*
by *auto*
then show *content k* ∗$_R$ (*f*′ (*x*)) − (*f* ((*Sup k*)) − *f* ((*Inf k*))) = *0*
using *xk* **unfolding** *uv* **by** *auto*
next
have ∗: *p* ∩ {*t*. *fst t* ∈ {*a*, *b*} ∧ *content*(*snd t*) ≠ *0*} =
{*t*. *t*∈*p* ∧ *fst t* = *a* ∧ *content*(*snd t*) ≠ *0*} ∪ {*t*. *t*∈*p* ∧ *fst t* = *b* ∧ *content*(*snd t*) ≠ *0*}
by *blast*
have ∗∗: *norm* (*setsum f s*) ≤ *e*
if ∀ *x y*. *x* ∈ *s* ∧ *y* ∈ *s* ⟶ *x* = *y*
and ∀ *x*. *x* ∈ *s* ⟶ *norm* (*f x*) ≤ *e*
and *e* > *0*
for *s f* **and** *e* :: *real*
proof (*cases s* = {})
case *True*
with *that* **show** *?thesis* **by** *auto*
next
case *False*
then obtain *x* **where** *x* ∈ *s*
by *auto*
then have ∗: *s* = {*x*}
using *that*(*1*) **by** *auto*
then show *?thesis*
using ‹*x* ∈ *s*› *that*(*2*) **by** *auto*
qed

```
case 2
show ?case
  apply (subst *)
  apply (subst setsum.union-disjoint)
  prefer 4
  apply (rule order-trans[of - e * (b - a)/4 + e * (b - a)/4])
  apply (rule norm-triangle-le,rule add-mono)
  apply (rule-tac[1-2] **)
proof -
  let ?B = λx. {t ∈ p. fst t = x ∧ content (snd t) ≠ 0}
  have pa: ∃ v. k = cbox a v ∧ a ≤ v if (a, k) ∈ p for k
  proof -
    guess u v using p(4)[OF that] by (elim exE) note uv=this
    have *: u ≤ v
      using p(2)[OF that] unfolding uv by auto
    have u: u = a
    proof (rule ccontr)
      have u ∈ cbox u v
        using p(2-3)[OF that(1)] unfolding uv by auto
      have u ≥ a
        using p(2-3)[OF that(1)] unfolding uv subset-eq by auto
      moreover assume ¬ ?thesis
      ultimately have u > a by auto
      then show False
        using p(2)[OF that(1)] unfolding uv by (auto simp add:)
    qed
    then show ?thesis
      apply (rule-tac x=v in exI)
      unfolding uv
      using *
      apply auto
      done
  qed
  have pb: ∃ v. k = cbox v b ∧ b ≥ v if (b, k) ∈ p for k
  proof -
    guess u v using p(4)[OF that] by (elim exE) note uv=this
    have *: u ≤ v
      using p(2)[OF that] unfolding uv by auto
    have u: v = b
    proof (rule ccontr)
      have u ∈ cbox u v
        using p(2-3)[OF that(1)] unfolding uv by auto
      have v ≤ b
        using p(2-3)[OF that(1)] unfolding uv subset-eq by auto
      moreover assume ¬ ?thesis
      ultimately have v < b by auto
      then show False
        using p(2)[OF that(1)] unfolding uv by (auto simp add:)
    qed
```

```
    then show ?thesis
      apply (rule-tac x=u in exI)
      unfolding uv
      using ∗
      apply auto
      done
  qed
  show ∀ x y. x ∈ ?B a ∧ y ∈ ?B a ⟶ x = y
    apply (rule,rule,rule,unfold split-paired-all)
    unfolding mem-Collect-eq fst-conv snd-conv
    apply safe
  proof −
    fix x k k′
    assume k: (a, k) ∈ p (a, k′) ∈ p content k ≠ 0 content k′ ≠ 0
    guess v using pa[OF k(1)] .. note v = conjunctD2[OF this]
    guess v′ using pa[OF k(2)] .. note v′ = conjunctD2[OF this] let ?v
= min v v′
    have box a ?v ⊆ k ∩ k′
      unfolding v v′ by (auto simp add: mem-box)
    note interior-mono[OF this,unfolded interior-Int]
    moreover have (a + ?v)/2 ∈ box a ?v
      using k(3−)
      unfolding v v′ content-eq-0 not-le
      by (auto simp add: mem-box)
    ultimately have (a + ?v)/2 ∈ interior k ∩ interior k′
      unfolding interior-open[OF open-box] by auto
    then have ∗: k = k′
      apply −
      apply (rule ccontr)
      using p(5)[OF k(1−2)]
      apply auto
      done
    { assume x ∈ k then show x ∈ k′ unfolding ∗ . }
    { assume x ∈ k′ then show x ∈ k unfolding ∗ . }
  qed
  show ∀ x y. x ∈ ?B b ∧ y ∈ ?B b ⟶ x = y
    apply rule
    apply rule
    apply rule
    apply (unfold split-paired-all)
    unfolding mem-Collect-eq fst-conv snd-conv
    apply safe
  proof −
    fix x k k′
    assume k: (b, k) ∈ p (b, k′) ∈ p content k ≠ 0 content k′ ≠ 0
    guess v using pb[OF k(1)] .. note v = conjunctD2[OF this]
    guess v′ using pb[OF k(2)] .. note v′ = conjunctD2[OF this]
    let ?v = max v v′
    have box ?v b ⊆ k ∩ k′
```

```
            unfolding v v' by (auto simp: mem-box)
            note interior-mono[OF this,unfolded interior-Int]
          moreover have ((b + ?v)/2) ∈ box ?v b
              using k(3−) unfolding v v' content-eq-0 not-le by (auto simp:
mem-box)
          ultimately have ((b + ?v)/2) ∈ interior k ∩ interior k'
            unfolding interior-open[OF open-box] by auto
          then have ∗: k = k'
            apply −
            apply (rule ccontr)
            using p(5)[OF k(1−2)]
            apply auto
            done
          { assume x ∈ k then show x ∈ k' unfolding ∗ . }
          { assume x ∈ k' then show x∈k unfolding ∗ . }
        qed

        let ?a = a and ?b = b
         show ∀x. x ∈ ?B a ⟶ norm ((λ(x, k). content k ∗R f' x − (f (Sup
k) −
          f (Inf k))) x) ≤ e ∗ (b − a) / 4
          apply rule
          apply rule
          unfolding mem-Collect-eq
          unfolding split-paired-all fst-conv snd-conv
        proof (safe, goal-cases)
          case prems: 1
          guess v using pa[OF prems(1)] .. note v = conjunctD2[OF this]
          have ?a ∈ {?a..v}
            using v(2) by auto
          then have v ≤ ?b
            using p(3)[OF prems(1)] unfolding subset-eq v by auto
          moreover have {?a..v} ⊆ ball ?a da
            using fineD[OF as(2) prems(1)]
            apply −
            apply (subst(asm) if-P)
            apply (rule refl)
            unfolding subset-eq
            apply safe
            apply (erule-tac x= x in ballE)
            apply (auto simp add:subset-eq dist-real-def v)
            done
          ultimately show ?case
            unfolding v interval-bounds-real[OF v(2)] box-real
            apply −
            apply(rule da(2)[of v])
            using prems fineD[OF as(2) prems(1)]
            unfolding v content-eq-0
            apply auto
```

```
          done
        qed
        show ∀x. x ∈ ?B b ⟶ norm ((λ(x, k). content k *R f' x −
          (f (Sup k) − f (Inf k))) x) ≤ e * (b − a) / 4
          apply rule
          apply rule
          unfolding mem-Collect-eq
          unfolding split-paired-all fst-conv snd-conv
        proof (safe, goal-cases)
          case prems: 1
          guess v using pb[OF prems(1)] .. note v = conjunctD2[OF this]
          have ?b ∈ {v.. ?b}
            using v(2) by auto
          then have v ≥ ?a using p(3)[OF prems(1)]
            unfolding subset-eq v by auto
          moreover have {v..?b} ⊆ ball ?b db
            using fineD[OF as(2) prems(1)]
            apply −
            apply (subst(asm) if-P, rule refl)
            unfolding subset-eq
            apply safe
            apply (erule-tac x= x in ballE)
            using ab
            apply (auto simp add:subset-eq v dist-real-def)
            done
          ultimately show ?case
            unfolding v
            unfolding interval-bounds-real[OF v(2)] box-real
            apply −
            apply(rule db(2)[of v])
            using prems fineD[OF as(2) prems(1)]
            unfolding v content-eq-0
            apply auto
            done
        qed
      qed (insert p(1) ab e, auto simp add: field-simps)
    qed auto
  qed
 qed
qed
```

35.53 Stronger form with finite number of exceptional points.

```
lemma fundamental-theorem-of-calculus-interior-strong:
 fixes f :: real ⇒ 'a::banach
 assumes finite s
   and a ≤ b
   and continuous-on {a .. b} f
```

```
    and ∀x∈{a <..< b} − s. (f has-vector-derivative f'(x)) (at x)
  shows (f' has-integral (f b − f a)) {a .. b}
  using assms
proof (induct card s arbitrary: s a b)
  case 0
  show ?case
    apply (rule fundamental-theorem-of-calculus-interior)
    using 0
    apply auto
    done
next
  case (Suc n)
  from this(2) guess c s'
    apply −
    apply (subst(asm) eq-commute)
    unfolding card-Suc-eq
    apply (subst(asm)(2) eq-commute)
    apply (elim exE conjE)
    done
  note cs = this[rule-format]
  show ?case
  proof (cases c ∈ box a b)
    case False
    then show ?thesis
      apply −
      apply (rule Suc(1)[OF cs(3) - Suc(4,5)])
      apply safe
      defer
      apply (rule Suc(6)[rule-format])
      using Suc(3)
      unfolding cs
      apply auto
      done
  next
    have ∗: f b − f a = (f c − f a) + (f b − f c)
      by auto
    case True
    then have a ≤ c c ≤ b
      by (auto simp: mem-box)
    then show ?thesis
      apply (subst ∗)
      apply (rule has-integral-combine)
      apply assumption+
      apply (rule-tac[!] Suc(1)[OF cs(3)])
      using Suc(3)
      unfolding cs
    proof −
      show continuous-on {a .. c} f continuous-on {c .. b} f
        apply (rule-tac[!] continuous-on-subset[OF Suc(5)])
```

```
      using True
      apply (auto simp: mem-box)
      done
    let ?P = λi j. ∀x∈{i <..< j} - s'. (f has-vector-derivative f' x) (at x)
    show ?P a c ?P c b
      apply safe
      apply (rule-tac[!] Suc(6)[rule-format])
      using True
      unfolding cs
      apply (auto simp: mem-box)
      done
  qed auto
 qed
qed

lemma fundamental-theorem-of-calculus-strong:
 fixes f :: real ⇒ 'a::banach
 assumes finite s
  and a ≤ b
  and continuous-on {a .. b} f
  and ∀x∈{a .. b} - s. (f has-vector-derivative f'(x)) (at x)
 shows (f' has-integral (f b - f a)) {a .. b}
 apply (rule fundamental-theorem-of-calculus-interior-strong[OF assms(1-3), of
f ])
 using assms(4)
 apply (auto simp: mem-box)
 done

lemma indefinite-integral-continuous-left:
 fixes f:: real ⇒ 'a::banach
 assumes f integrable-on {a .. b}
  and a < c
  and c ≤ b
  and e > 0
 obtains d where d > 0
  and ∀t. c - d < t ∧ t ≤ c ⟶ norm (integral {a .. c} f - integral {a .. t}
f) < e
proof -
 have ∃w>0. ∀t. c - w < t ∧ t < c ⟶ norm (f c) * norm(c - t) < e / 3
 proof (cases f c = 0)
  case False
  hence 0 < e / 3 / norm (f c) using ‹e>0› by simp
  then show ?thesis
   apply -
   apply rule
   apply rule
   apply assumption
   apply safe
  proof -
```

```
    fix t
    assume as: t < c and c − e / 3 / norm (f c) < t
    then have c − t < e / 3 / norm (f c)
      by auto
    then have norm (c − t) < e / 3 / norm (f c)
      using as by auto
    then show norm (f c) ∗ norm (c − t) < e / 3
      using False
      apply −
      apply (subst mult.commute)
      apply (subst pos-less-divide-eq[symmetric])
      apply auto
      done
  qed
 next
  case True
  show ?thesis
    apply (rule-tac x=1 in exI)
    unfolding True
    using ‹e > 0›
    apply auto
    done
 qed
 then guess w .. note w = conjunctD2[OF this,rule-format]

 have ∗: e / 3 > 0
  using assms by auto
 have f integrable-on {a .. c}
  apply (rule integrable-subinterval-real[OF assms(1)])
  using assms(2−3)
  apply auto
  done
 from integrable-integral[OF this,unfolded has-integral-real,rule-format,OF ∗] guess
d1 ..
 note d1 = conjunctD2[OF this,rule-format]
 def d ≡ λx. ball x w ∩ d1 x
 have gauge d
  unfolding d-def using w(1) d1 by auto
 note this[unfolded gauge-def,rule-format,of c]
 note conjunctD2[OF this]
 from this(2)[unfolded open-contains-ball,rule-format,OF this(1)] guess k ..
 note k=conjunctD2[OF this]

 let ?d = min k (c − a) / 2
 show ?thesis
  apply (rule that[of ?d])
  apply safe
 proof −
  show ?d > 0
```

```
    using k(1) using assms(2) by auto
  fix t
  assume as: c − ?d < t t ≤ c
  let ?thesis = norm (integral ({a .. c}) f − integral ({a .. t}) f) < e
  {
    presume ∗: t < c ⟹ ?thesis
    show ?thesis
      apply (cases t = c)
      defer
      apply (rule ∗)
      apply (subst less-le)
      using ‹e > 0› as(2)
      apply auto
      done
  }
  assume t < c

  have f integrable-on {a .. t}
    apply (rule integrable-subinterval-real[OF assms(1)])
    using assms(2−3) as(2)
    apply auto
    done
   from integrable-integral[OF this,unfolded has-integral-real,rule-format,OF ∗]
guess d2 ..
  note d2 = conjunctD2[OF this,rule-format]
  def d3 ≡ λx. if x ≤ t then d1 x ∩ d2 x else d1 x
  have gauge d3
    using d2(1) d1(1) unfolding d3-def gauge-def by auto
  from fine-division-exists-real[OF this, of a t] guess p . note p=this
  note p'=tagged-division-ofD[OF this(1)]
  have pt: ∀(x,k)∈p. x ≤ t
  proof (safe, goal-cases)
    case prems: 1
    from p'(2,3)[OF prems] show ?case
      by auto
  qed
  with p(2) have d2 fine p
    unfolding fine-def d3-def
    apply safe
    apply (erule-tac x=(a,b) in ballE)+
    apply auto
    done
  note d2-fin = d2(2)[OF conjI[OF p(1) this]]

  have ∗: {a .. c} ∩ {x. x · 1 ≤ t} = {a .. t} {a .. c} ∩ {x. x · 1 ≥ t} = {t ..
c}
    using assms(2−3) as by (auto simp add: field-simps)
  have p ∪ {(c, {t .. c})} tagged-division-of {a .. c} ∧ d1 fine p ∪ {(c, {t .. c})}
    apply rule
```

```
    apply (rule tagged-division-union-interval-real[of - - - 1 t])
    unfolding *
    apply (rule p)
    apply (rule tagged-division-of-self-real)
    unfolding fine-def
    apply safe
   proof −
    fix x k y
    assume (x,k) ∈ p and y ∈ k
    then show y ∈ d1 x
     using p(2) pt
     unfolding fine-def d3-def
     apply −
     apply (erule-tac x=(x,k) in ballE)+
     apply auto
     done
   next
    fix x assume x ∈ {t..c}
    then have dist c x < k
     unfolding dist-real-def
     using as(1)
     by (auto simp add: field-simps)
    then show x ∈ d1 c
     using k(2)
     unfolding d-def
     by auto
   qed (insert as(2), auto) note d1-fin = d1(2)[OF this]

   have *: integral {a .. c} f − integral {a .. t} f = −(((c − t) *R f c + (∑(x,
k)∈p. content k *R f x)) −
    integral {a .. c} f) + ((∑(x, k)∈p. content k *R f x) − integral {a .. t} f)
+ (c − t) *R f c
    e = (e/3 + e/3) + e/3
    by auto
   have **: (∑(x, k)∈p ∪ {(c, {t .. c})}. content k *R f x) =
    (c − t) *R f c + (∑(x, k)∈p. content k *R f x)
   proof −
    have **: ⋀x F. F ∪ {x} = insert x F
     by auto
    have (c, cbox t c) ∉ p
    proof (safe, goal-cases)
     case prems: 1
     from p'(2−3)[OF prems] have c ∈ cbox a t
      by auto
     then show False using ‹t < c›
      by auto
    qed
    then show ?thesis
     unfolding ** box-real
```

```
      apply −
      apply (subst setsum.insert)
      apply (rule p')
      unfolding split-conv
      defer
      apply (subst content-real)
      using as(2)
      apply auto
      done
  qed
  have ***: c − w < t ∧ t < c
  proof −
    have c − k < t
      using ‹k>0› as(1) by (auto simp add: field-simps)
    moreover have k ≤ w
      apply (rule ccontr)
      using k(2)
      unfolding subset-eq
      apply (erule-tac x=c + ((k + w)/2) in ballE)
      unfolding d-def
      using ‹k > 0› ‹w > 0›
      apply (auto simp add: field-simps not-le not-less dist-real-def)
      done
    ultimately show ?thesis using ‹t < c›
      by (auto simp add: field-simps)
  qed
  show ?thesis
    unfolding *(1)
    apply (subst *(2))
    apply (rule norm-triangle-lt add-strict-mono)+
    unfolding norm-minus-cancel
    apply (rule d1-fin[unfolded **])
    apply (rule d2-fin)
    using w(2)[OF ***]
    unfolding norm-scaleR
    apply (auto simp add: field-simps)
    done
 qed
qed

lemma indefinite-integral-continuous-right:
 fixes f :: real ⇒ 'a::banach
 assumes f integrable-on {a .. b}
   and a ≤ c
   and c < b
   and e > 0
 obtains d where 0 < d
   and ∀t. c ≤ t ∧ t < c + d ⟶ norm (integral {a .. c} f − integral {a .. t}
f) < e
```

proof −
have ∗: (λx. f (− x)) *integrable-on* {−b .. −a} − b < − c − c ≤ − a
using *assms* **by** *auto*
from *indefinite-integral-continuous-left*[*OF* ∗ ‹e>0›] **guess** d **. note** d = *this*
let *?d* = *min d* (b − c)
show *?thesis*
apply (*rule that*[*of ?d*])
apply *safe*
proof −
show *0* < *?d*
using *d*(*1*) *assms*(*3*) **by** *auto*
fix t :: *real*
assume *as*: c ≤ t t < c + *?d*
have ∗: *integral* {a .. c} f = *integral* {a .. b} f − *integral* {c .. b} f
integral {a .. t} f = *integral* {a .. b} f − *integral* {t .. b} f
unfolding *algebra-simps*
apply (*rule-tac*[!] *integral-combine*)
using *assms as*
apply *auto*
done
have (− c) − d < (− t) ∧ − t ≤ − c
using *as* **by** *auto* **note** *d*(*2*)[*rule-format*,*OF this*]
then show *norm* (*integral* {a .. c} f − *integral* {a .. t} f) < e
unfolding ∗
unfolding *integral-reflect*
apply (*subst norm-minus-commute*)
apply (*auto simp add*: *algebra-simps*)
done
qed
qed

lemma *indefinite-integral-continuous*:
fixes f :: *real* ⇒ 'a::*banach*
assumes f *integrable-on* {a .. b}
shows *continuous-on* {a .. b} (λx. *integral* {a .. x} f)
proof (*unfold continuous-on-iff*, *safe*)
fix x e :: *real*
assume *as*: x ∈ {a .. b} e > *0*
let *?thesis* = ∃ d>*0*. ∀ x'∈{a .. b}. *dist* x' x < d ⟶ *dist* (*integral* {a .. x'} f)
(*integral* {a .. x} f) < e
{
presume ∗: a < b ⟹ *?thesis*
show *?thesis*
apply *cases*
apply (*rule* ∗)
apply *assumption*
proof *goal-cases*
case *1*
then have *cbox* a b = {x}

```
      using as(1)
      apply −
      apply (rule set-eqI)
      apply auto
      done
    then show ?case using ‹e > 0› by auto
  qed
 }
 assume a < b
 have (x = a ∨ x = b) ∨ (a < x ∧ x < b)
  using as(1) by auto
 then show ?thesis
  apply (elim disjE)
 proof −
  assume x = a
  have a ≤ a ..
  from indefinite-integral-continuous-right[OF assms(1) this ‹a<b› ‹e>0›] guess
d . note d=this
  show ?thesis
   apply rule
   apply rule
   apply (rule d)
   apply safe
   apply (subst dist-commute)
   unfolding ‹x = a› dist-norm
   apply (rule d(2)[rule-format])
   apply auto
   done
 next
  assume x = b
  have b ≤ b ..
  from indefinite-integral-continuous-left[OF assms(1) ‹a<b› this ‹e>0›] guess
d . note d=this
  show ?thesis
   apply rule
   apply rule
   apply (rule d)
   apply safe
   apply (subst dist-commute)
   unfolding ‹x = b› dist-norm
   apply (rule d(2)[rule-format])
   apply auto
   done
 next
  assume a < x ∧ x < b
  then have xl: a < x x ≤ b and xr: a ≤ x x < b
   by auto
  from indefinite-integral-continuous-left [OF assms(1) xl ‹e>0›] guess d1 .
note d1=this
```

from *indefinite-integral-continuous-right*[*OF assms*(*1*) *xr* ‹*e*>*0*›] **guess** *d2* **.**
note *d2*=*this*
show *?thesis*
apply (*rule-tac x*=*min d1 d2* **in** *exI*)
proof *safe*
show *0* < *min d1 d2*
using *d1 d2* **by** *auto*
fix *y*
assume *y* ∈ {*a* .. *b*} **and** *dist y x* < *min d1 d2*
then show *dist* (*integral* {*a* .. *y*} *f*) (*integral* {*a* .. *x*} *f*) < *e*
apply (*subst dist-commute*)
apply (*cases y* < *x*)
unfolding *dist-norm*
apply (*rule d1*(*2*)[*rule-format*])
defer
apply (*rule d2*(*2*)[*rule-format*])
unfolding *not-less*
apply (*auto simp add*: *field-simps*)
done
qed
qed
qed

35.54 This doesn't directly involve integration, but that gives an easy proof.

lemma *has-derivative-zero-unique-strong-interval*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *finite k*
and *continuous-on* {*a* .. *b*} *f*
and *f a* = *y*
and ∀ *x*∈({*a* .. *b*} − *k*). (*f has-derivative* (λ*h*. *0*)) (*at x within* {*a* .. *b*}) *x* ∈ {*a* .. *b*}
shows *f x* = *y*
proof −
have *ab*: *a* ≤ *b*
using *assms* **by** *auto*
have ∗: *a* ≤ *x*
using *assms*(*5*) **by** *auto*
have ((λ*x*. *0*::*'a*) *has-integral f x* − *f a*) {*a* .. *x*}
apply (*rule fundamental-theorem-of-calculus-interior-strong*[*OF assms*(*1*) ∗])
apply (*rule continuous-on-subset*[*OF assms*(*2*)])
defer
apply *safe*
unfolding *has-vector-derivative-def*
apply (*subst has-derivative-within-open*[*symmetric*])
apply *assumption*
apply (*rule open-greaterThanLessThan*)
apply (*rule has-derivative-within-subset*[**where** *s*={*a* .. *b*}])

```
    using assms(4) assms(5)
    apply (auto simp: mem-box)
    done
  note this[unfolded *]
  note has-integral-unique[OF has-integral-0 this]
  then show ?thesis
    unfolding assms by auto
qed
```

35.55 Generalize a bit to any convex set.

```
lemma has-derivative-zero-unique-strong-convex:
  fixes f :: 'a::euclidean-space ⇒ 'b::banach
  assumes convex s
    and finite k
    and continuous-on s f
    and c ∈ s
    and f c = y
    and ∀ x∈(s - k). (f has-derivative (λh. 0)) (at x within s)
    and x ∈ s
  shows f x = y
proof -
  {
    presume *: x ≠ c ⟹ ?thesis
    show ?thesis
      apply cases
      apply (rule *)
      apply assumption
      unfolding assms(5)[symmetric]
      apply auto
      done
  }
  assume x ≠ c
  note conv = assms(1)[unfolded convex-alt,rule-format]
  have as1: continuous-on {0 ..1} (f ∘ (λt. (1 - t) *R c + t *R x))
    apply (rule continuous-intros)+
    apply (rule continuous-on-subset[OF assms(3)])
    apply safe
    apply (rule conv)
    using assms(4,7)
    apply auto
    done
  have *: t = xa if (1 - t) *R c + t *R x = (1 - xa) *R c + xa *R x for t xa
  proof -
    from that have (t - xa) *R x = (t - xa) *R c
      unfolding scaleR-simps by (auto simp add: algebra-simps)
    then show ?thesis
      using ‹x ≠ c› by auto
  qed
```

```
  have as2: finite {t. ((1 - t) *R c + t *R x) ∈ k}
    using assms(2)
    apply (rule finite-surj[where f=λz. SOME t. (1−t) *R c + t *R x = z])
    apply safe
    unfolding image-iff
    apply rule
    defer
    apply assumption
    apply (rule sym)
    apply (rule some-equality)
    defer
    apply (drule *)
    apply auto
    done
  have (f ∘ (λt. (1 - t) *R c + t *R x)) 1 = y
    apply (rule has-derivative-zero-unique-strong-interval[OF as2 as1, of ])
    unfolding o-def
    using assms(5)
    defer
    apply −
    apply rule
  proof −
    fix t
    assume as: t ∈ {0 .. 1} − {t. (1 - t) *R c + t *R x ∈ k}
    have *: c − t *R c + t *R x ∈ s − k
      apply safe
      apply (rule conv[unfolded scaleR-simps])
      using ‹x ∈ s› ‹c ∈ s› as
      by (auto simp add: algebra-simps)
    have (f ∘ (λt. (1 - t) *R c + t *R x) has-derivative (λx. 0) ∘ (λz. (0 − z *R
c) + z *R x))
      (at t within {0 .. 1})
      apply (intro derivative-eq-intros)
      apply simp-all
      apply (simp add: field-simps)
      unfolding scaleR-simps
      apply (rule has-derivative-within-subset,rule assms(6)[rule-format])
      apply (rule *)
      apply safe
      apply (rule conv[unfolded scaleR-simps])
      using ‹x ∈ s› ‹c ∈ s›
      apply auto
      done
    then show ((λxa. f ((1 - xa) *R c + xa *R x)) has-derivative (λh. 0)) (at t
within {0 .. 1})
      unfolding o-def .
  qed auto
  then show ?thesis
    by auto
```

qed

Also to any open connected set with finite set of exceptions. Could generalize to locally convex set with limpt-free set of exceptions.

lemma *has-derivative-zero-unique-strong-connected*:
fixes f :: *'a::euclidean-space* ⇒ *'b::banach*
assumes *connected s*
and *open s*
and *finite k*
and *continuous-on s f*
and $c \in s$
and $f\ c = y$
and $\forall x{\in}(s - k).\ (f\ has\text{-}derivative\ (\lambda h.\ 0))\ (at\ x\ within\ s)$
and $x{\in}s$
shows $f\ x = y$
proof −
have $\{x \in s.\ f\ x \in \{y\}\} = \{\} \vee \{x \in s.\ f\ x \in \{y\}\} = s$
apply (*rule assms*(*1*)[*unfolded connected-clopen*,*rule-format*])
apply *rule*
defer
apply (*rule continuous-closedin-preimage*[*OF assms*(*4*) *closed-singleton*])
apply (*rule open-openin-trans*[*OF assms*(*2*)])
unfolding *open-contains-ball*
proof *safe*
fix x
assume $x \in s$
from *assms*(*2*)[*unfolded open-contains-ball*,*rule-format*,*OF this*] **guess** e **.. note** e=*conjunctD2*[*OF this*]
show $\exists e{>}0.\ ball\ x\ e \subseteq \{xa \in s.\ f\ xa \in \{f\ x\}\}$
apply *rule*
apply *rule*
apply (*rule e*)
proof *safe*
fix y
assume *y*: $y \in ball\ x\ e$
then show $y \in s$
using e **by** *auto*
show $f\ y = f\ x$
apply (*rule has-derivative-zero-unique-strong-convex*[*OF convex-ball*])
apply (*rule assms*)
apply (*rule continuous-on-subset*)
apply (*rule assms*)
apply (*rule e*)+
apply (*subst centre-in-ball*)
apply (*rule e*)
apply *rule*
apply *safe*
apply (*rule has-derivative-within-subset*)
apply (*rule assms*(*7*)[*rule-format*])

```
      using y e
      apply auto
      done
  qed
 qed
 then show ?thesis
  using ‹x ∈ s› ‹f c = y› ‹c ∈ s› by auto
qed

lemma has-derivative-zero-connected-constant:
 fixes f :: 'a::euclidean-space ⇒ 'b::banach
 assumes connected s
    and open s
    and finite k
    and continuous-on s f
    and ∀x∈(s - k). (f has-derivative (λh. 0)) (at x within s)
  obtains c where ⋀x. x ∈ s ⟹ f(x) = c
proof (cases s = {})
 case True
 then show ?thesis
by (metis empty-iff that)
next
 case False
 then obtain c where c ∈ s
  by (metis equals0I)
 then show ?thesis
  by (metis has-derivative-zero-unique-strong-connected assms that)
qed
```

35.56 Integrating characteristic function of an interval

```
lemma has-integral-restrict-open-subinterval:
 fixes f :: 'a::euclidean-space ⇒ 'b::banach
 assumes (f has-integral i) (cbox c d)
  and cbox c d ⊆ cbox a b
 shows ((λx. if x ∈ box c d then f x else 0) has-integral i) (cbox a b)
proof -
 def g ≡ λx. if x ∈box c d then f x else 0
 {
  presume ∗: cbox c d ≠ {} ⟹ ?thesis
  show ?thesis
   apply cases
   apply (rule ∗)
   apply assumption
  proof goal-cases
   case prems: 1
   then have ∗: box c d = {}
    by (metis bot.extremum-uniqueI box-subset-cbox)
   show ?thesis
```

```
      using assms(1)
      unfolding ∗
      using prems
      by auto
    qed
  }
  assume cbox c d ≠ {}
  from partial-division-extend-1[OF assms(2) this] guess p . note p=this
  note mon = monoidal-lifted[OF monoidal-monoid]
  note operat = operative-division[OF this operative-integral p(1), symmetric]
  let ?P = (if g integrable-on cbox a b then Some (integral (cbox a b) g) else None)
= Some i
  {
    presume ?P
    then have g integrable-on cbox a b ∧ integral (cbox a b) g = i
      apply −
      apply cases
      apply (subst(asm) if-P)
      apply assumption
      apply auto
      done
    then show ?thesis
      using integrable-integral
      unfolding g-def
      by auto
  }

  note iterate-eq-neutral[OF mon,unfolded neutral-lifted[OF monoidal-monoid]]
  note ∗ = this[unfolded neutral-add]
  have iterate:iterate (lifted op +) (p − {cbox c d})
    (λi. if g integrable-on i then Some (integral i g) else None) = Some 0
  proof (rule ∗)
    fix x
    assume x: x ∈ p − {cbox c d}
    then have x ∈ p
      by auto
    note div = division-ofD(2−5)[OF p(1) this]
    from div(3) guess u v by (elim exE) note uv=this
    have interior x ∩ interior (cbox c d) = {}
      using div(4)[OF p(2)] x by auto
    then have (g has-integral 0) x
      unfolding uv
      apply −
      apply (rule has-integral-spike-interior[where f=λx. 0])
      unfolding g-def interior-cbox
      apply auto
      done
    then show (if g integrable-on x then Some (integral x g) else None) = Some 0
      by auto
```

```
  qed

  have *: p = insert (cbox c d) (p - {cbox c d})
    using p by auto
  have **: g integrable-on cbox c d
    apply (rule integrable-spike-interior[where f=f])
    unfolding g-def
    defer
    apply (rule has-integral-integrable)
    using assms(1)
    apply auto
    done
  moreover
  have integral (cbox c d) g = i
    apply (rule has-integral-unique[OF - assms(1)])
    apply (rule has-integral-spike-interior[where f=g])
    defer
    apply (rule integrable-integral[OF **])
    unfolding g-def
    apply auto
    done
  ultimately show ?P
    unfolding operat
    apply (subst *)
    apply (subst iterate-insert)
    apply rule+
    unfolding iterate
    defer
    apply (subst if-not-P)
    defer
    using p
    apply auto
    done
qed

lemma has-integral-restrict-closed-subinterval:
  fixes f :: 'a::euclidean-space ⇒ 'b::banach
  assumes (f has-integral i) (cbox c d)
    and cbox c d ⊆ cbox a b
  shows ((λx. if x ∈ cbox c d then f x else 0) has-integral i) (cbox a b)
proof -
  note has-integral-restrict-open-subinterval[OF assms]
  note * = has-integral-spike[OF negligible-frontier-interval - this]
  show ?thesis
    apply (rule *[of c d])
    using box-subset-cbox[of c d]
    apply auto
    done
qed
```

lemma *has-integral-restrict-closed-subintervals-eq*:
fixes $f :: 'a::euclidean\text{-}space \Rightarrow 'b::banach$
assumes $cbox\ c\ d \subseteq cbox\ a\ b$
shows $((\lambda x.\ if\ x \in cbox\ c\ d\ then\ f\ x\ else\ 0)\ has\text{-}integral\ i)\ (cbox\ a\ b) \longleftrightarrow (f\ has\text{-}integral\ i)\ (cbox\ c\ d)$
(**is** *?l = ?r*)
proof (*cases cbox c d = {}*)
case *False*
let *?g = λx. if x ∈ cbox c d then f x else 0*
show *?thesis*
apply *rule*
defer
apply (*rule has-integral-restrict-closed-subinterval*[*OF - assms*])
apply *assumption*
proof −
assume *?l*
then have *?g integrable-on cbox c d*
using *assms has-integral-integrable integrable-subinterval* **by** *blast*
then have ∗: *f integrable-on cbox c d*
apply −
apply (*rule integrable-eq*)
apply *auto*
done
then have *i = integral (cbox c d) f*
apply −
apply (*rule has-integral-unique*)
apply (*rule ‹?l›*)
apply (*rule has-integral-restrict-closed-subinterval*[*OF - assms*])
apply *auto*
done
then show *?r*
using ∗ **by** *auto*
qed
qed *auto*

Hence we can apply the limit process uniformly to all integrals.

lemma *has-integral′*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
shows $(f\ has\text{-}integral\ i)\ s \longleftrightarrow$
$(\forall e{>}0.\ \exists B{>}0.\ \forall a\ b.\ ball\ 0\ B \subseteq cbox\ a\ b \longrightarrow$
$(\exists z.\ ((\lambda x.\ if\ x \in s\ then\ f(x)\ else\ 0)\ has\text{-}integral\ z)\ (cbox\ a\ b) \wedge norm(z - i) < e))$
(**is** *?l* $\longleftrightarrow$ $(\forall e{>}0.\ ?r\ e)$)
proof −
{
presume ∗: $\exists a\ b.\ s = cbox\ a\ b \Longrightarrow$ *?thesis*
show *?thesis*
apply *cases*

```
    apply (rule *)
    apply assumption
    apply (subst has-integral-alt)
    apply auto
    done
 }
 assume ∃ a b. s = cbox a b
 then guess a b by (elim exE) note s=this
 from bounded-cbox[of a b, unfolded bounded-pos] guess B ..
 note B = conjunctD2[OF this,rule-format] show ?thesis
  apply safe
 proof −
  fix e :: real
  assume ?l and e > 0
  show ?r e
    apply (rule-tac x=B+1 in exI)
    apply safe
    defer
    apply (rule-tac x=i in exI)
  proof
    fix c d :: 'n
    assume as: ball 0 (B+1) ⊆ cbox c d
    then show ((λx. if x ∈ s then f x else 0) has-integral i) (cbox c d)
      unfolding s
      apply −
      apply (rule has-integral-restrict-closed-subinterval)
      apply (rule ‹?l›[unfolded s])
      apply safe
      apply (drule B(2)[rule-format])
      unfolding subset-eq
      apply (erule-tac x=x in ballE)
      apply (auto simp add: dist-norm)
      done
  qed (insert B ‹e>0›, auto)
 next
  assume as: ∀ e>0. ?r e
  from this[rule-format,OF zero-less-one] guess C .. note C=conjunctD2[OF
this,rule-format]
  def c ≡ (∑ i∈Basis. (− max B C) *R i)::'n
  def d ≡ (∑ i∈Basis. max B C *R i)::'n
  have c-d: cbox a b ⊆ cbox c d
    apply safe
    apply (drule B(2))
    unfolding mem-box
  proof
    fix x i
    show c • i ≤ x • i ∧ x • i ≤ d • i if norm x ≤ B and i ∈ Basis
      using that and Basis-le-norm[OF ‹i∈Basis›, of x]
      unfolding c-def d-def
```

```
    by (auto simp add: field-simps setsum-negf)
  qed
  have ball 0 C ⊆ cbox c d
    apply (rule subsetI)
    unfolding mem-box mem-ball dist-norm
  proof
    fix x i :: 'n
    assume x: norm (0 − x) < C and i: i ∈ Basis
    show c · i ≤ x · i ∧ x · i ≤ d · i
      using Basis-le-norm[OF i, of x] and x i
      unfolding c-def d-def
      by (auto simp: setsum-negf)
  qed
  from C(2)[OF this] have ∃ y. (f has-integral y) (cbox a b)
    unfolding has-integral-restrict-closed-subintervals-eq[OF c-d,symmetric]
    unfolding s
    by auto
  then guess y .. note y=this

  have y = i
  proof (rule ccontr)
    assume ¬ ?thesis
    then have 0 < norm (y − i)
      by auto
  from as[rule-format,OF this] guess C .. note C=conjunctD2[OF this,rule-format]
    def c ≡ (∑ i∈Basis. (− max B C) *R i)::'n
    def d ≡ (∑ i∈Basis. max B C *R i)::'n
    have c-d: cbox a b ⊆ cbox c d
      apply safe
      apply (drule B(2))
      unfolding mem-box
    proof
      fix x i :: 'n
      assume norm x ≤ B and i ∈ Basis
      then show c · i ≤ x · i ∧ x · i ≤ d · i
        using Basis-le-norm[of i x]
        unfolding c-def d-def
        by (auto simp add: field-simps setsum-negf)
    qed
    have ball 0 C ⊆ cbox c d
      apply (rule subsetI)
      unfolding mem-box mem-ball dist-norm
    proof
      fix x i :: 'n
      assume norm (0 − x) < C and i ∈ Basis
      then show c · i ≤ x · i ∧ x · i ≤ d · i
        using Basis-le-norm[of i x]
        unfolding c-def d-def
        by (auto simp: setsum-negf)
```

```
      qed
     note C(2)[OF this] then guess z .. note z = conjunctD2[OF this, unfolded
s]
      note this[unfolded has-integral-restrict-closed-subintervals-eq[OF c-d]]
      then have z = y and norm (z − i) < norm (y − i)
        apply −
        apply (rule has-integral-unique[OF - y(1)])
        apply assumption
        apply assumption
        done
      then show False
        by auto
    qed
    then show ?l
      using y
      unfolding s
      by auto
  qed
qed

lemma has-integral-le:
  fixes f :: 'n::euclidean-space ⇒ real
  assumes (f has-integral i) s
    and (g has-integral j) s
    and ∀x∈s. f x ≤ g x
  shows i ≤ j
  using has-integral-component-le[OF - assms(1−2), of 1]
  using assms(3)
  by auto

lemma integral-le:
  fixes f :: 'n::euclidean-space ⇒ real
  assumes f integrable-on s
    and g integrable-on s
    and ∀x∈s. f x ≤ g x
  shows integral s f ≤ integral s g
  by (rule has-integral-le[OF assms(1,2)[unfolded has-integral-integral] assms(3)])

lemma has-integral-nonneg:
  fixes f :: 'n::euclidean-space ⇒ real
  assumes (f has-integral i) s
    and ∀x∈s. 0 ≤ f x
  shows 0 ≤ i
  using has-integral-component-nonneg[of 1 f i s]
  unfolding o-def
  using assms
  by auto

lemma integral-nonneg:
```

fixes $f :: 'n::euclidean\text{-}space \Rightarrow real$
assumes f *integrable-on* s
and $\forall x{\in}s.\ 0 \leq f\ x$
shows $0 \leq$ *integral* $s\ f$
by (*rule has-integral-nonneg*[*OF assms*(*1*)[*unfolded has-integral-integral*] *assms*(*2*)])

Hence a general restriction property.

lemma *has-integral-restrict*[*simp*]:
assumes $s \subseteq t$
shows $((\lambda x.\ if\ x \in s\ then\ f\ x\ else\ (0::'a::banach))\ has\text{-}integral\ i)\ t \longleftrightarrow (f\ has\text{-}integral\ i)\ s$
proof −
have $*$: $\bigwedge x.\ (if\ x \in t\ then\ if\ x \in s\ then\ f\ x\ else\ 0\ else\ 0) = (if\ x{\in}s\ then\ f\ x\ else\ 0)$
using *assms* **by** *auto*
show *?thesis*
apply (*subst*(*2*) *has-integral'*)
apply (*subst has-integral'*)
unfolding $*$
apply *rule*
done
qed

lemma *has-integral-restrict-univ*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
shows $((\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0)\ has\text{-}integral\ i)\ UNIV \longleftrightarrow (f\ has\text{-}integral\ i)\ s$
by *auto*

lemma *has-integral-on-superset*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
assumes $\forall x.\ x \notin s \longrightarrow f\ x = 0$
and $s \subseteq t$
and $(f\ has\text{-}integral\ i)\ s$
shows $(f\ has\text{-}integral\ i)\ t$
proof −
have $(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0) = (\lambda x.\ if\ x \in t\ then\ f\ x\ else\ 0)$
apply *rule*
using *assms*(*1*−*2*)
apply *auto*
done
then show *?thesis*
using *assms*(*3*)
apply (*subst has-integral-restrict-univ*[*symmetric*])
apply (*subst*(*asm*) *has-integral-restrict-univ*[*symmetric*])
apply *auto*
done
qed

lemma *integrable-on-superset*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
assumes $\forall x.\ x \notin s \longrightarrow f\ x = 0$
and $s \subseteq t$
and f *integrable-on* s
shows f *integrable-on* t
using *assms*
unfolding *integrable-on-def*
by (*auto intro*:*has-integral-on-superset*)

lemma *integral-restrict-univ*[*intro*]:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
shows f *integrable-on* $s \Longrightarrow$ *integral UNIV* $(\lambda x.$ *if* $x \in s$ *then* $f\ x$ *else* $0) =$ *integral* $s\ f$
apply (*rule integral-unique*)
unfolding *has-integral-restrict-univ*
apply *auto*
done

lemma *integrable-restrict-univ*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
shows $(\lambda x.$ *if* $x \in s$ *then* $f\ x$ *else* $0)$ *integrable-on UNIV* $\longleftrightarrow$ f *integrable-on* s
unfolding *integrable-on-def*
by *auto*

lemma *negligible-on-intervals*: *negligible* $s \longleftrightarrow (\forall\ a\ b.$ *negligible*$(s \cap cbox\ a\ b))$ (**is** *?l* $\longleftrightarrow$ *?r*)
proof
assume *?r*
show *?l*
unfolding *negligible-def*
proof *safe*
fix $a\ b$
show (*indicator s has-integral 0*) (*cbox a b*)
apply (*rule has-integral-negligible*[*OF* ‹*?r*›[*rule-format*,*of a b*]])
unfolding *indicator-def*
apply *auto*
done
qed
qed *auto*

lemma *has-integral-spike-set-eq*:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow 'a::banach$
assumes *negligible* $((s - t) \cup (t - s))$
shows $(f$ *has-integral* $y)\ s \longleftrightarrow (f$ *has-integral* $y)\ t$
unfolding *has-integral-restrict-univ*[*symmetric*,*of f*]
apply (*rule has-integral-spike-eq*[*OF assms*])
by (*auto split*: *split-if-asm*)

lemma *has-integral-spike-set*[*dest*]:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *negligible* ((*s* − *t*) ∪ (*t* − *s*))
and (*f has-integral y*) *s*
shows (*f has-integral y*) *t*
using *assms has-integral-spike-set-eq*
by *auto*

lemma *integrable-spike-set*[*dest*]:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *negligible* ((*s* − *t*) ∪ (*t* − *s*))
and *f integrable-on s*
shows *f integrable-on t*
using *assms*(*2*)
unfolding *integrable-on-def*
unfolding *has-integral-spike-set-eq*[*OF assms*(*1*)] **.**

lemma *integrable-spike-set-eq*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *negligible* ((*s* − *t*) ∪ (*t* − *s*))
shows *f integrable-on s* ⟷ *f integrable-on t*
apply *rule*
apply (*rule-tac*[!] *integrable-spike-set*)
using *assms*
apply *auto*
done

35.57 More lemmas that are useful later

lemma *has-integral-subset-component-le*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'m*::*euclidean-space*
assumes *k*: *k* ∈ *Basis*
and *as*: *s* ⊆ *t* (*f has-integral i*) *s* (*f has-integral j*) *t* ∀ *x*∈*t*. *0* ≤ *f*(*x*)•*k*
shows *i*•*k* ≤ *j*•*k*
proof −
note *has-integral-restrict-univ*[*symmetric*, *of f*]
note *as*(*2*−*3*)[*unfolded this*] **note** ∗ = *has-integral-component-le*[*OF k this*]
show *?thesis*
apply (*rule* ∗)
using *as*(*1*,*4*)
apply *auto*
done
qed

lemma *has-integral-subset-le*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *real*
assumes *s* ⊆ *t*
and (*f has-integral i*) *s*
and (*f has-integral j*) *t*

```
    and ∀x∈t. 0 ≤ f x
  shows i ≤ j
  using has-integral-subset-component-le[OF - assms(1), of 1 f i j]
  using assms
  by auto

lemma integral-subset-component-le:
  fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
  assumes k ∈ Basis
    and s ⊆ t
    and f integrable-on s
    and f integrable-on t
    and ∀x ∈ t. 0 ≤ f x • k
  shows (integral s f)•k ≤ (integral t f)•k
  apply (rule has-integral-subset-component-le)
  using assms
  apply auto
  done

lemma integral-subset-le:
  fixes f :: 'n::euclidean-space ⇒ real
  assumes s ⊆ t
    and f integrable-on s
    and f integrable-on t
    and ∀x ∈ t. 0 ≤ f x
  shows integral s f ≤ integral t f
  apply (rule has-integral-subset-le)
  using assms
  apply auto
  done

lemma has-integral-alt':
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  shows (f has-integral i) s ⟷ (∀a b. (λx. if x ∈ s then f x else 0) integrable-on
cbox a b) ∧
    (∀e>0. ∃B>0. ∀a b. ball 0 B ⊆ cbox a b ⟶
      norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) − i) < e)
  (is ?l = ?r)
proof
  assume ?r
  show ?l
    apply (subst has-integral')
    apply safe
  proof goal-cases
    case (1 e)
   from ‹?r›[THEN conjunct2,rule-format,OF this] guess B .. note B=conjunctD2[OF
this]
    show ?case
      apply rule
```

```
    apply rule
    apply (rule B)
    apply safe
    apply (rule-tac x=integral (cbox a b) (λx. if x ∈ s then f x else 0) in exI)
    apply (drule B(2)[rule-format])
    using integrable-integral[OF ‹?r›[THEN conjunct1,rule-format]]
    apply auto
    done
 qed
next
 assume ?l note as = this[unfolded has-integral'[of f],rule-format]
 let ?f = λx. if x ∈ s then f x else 0
 show ?r
 proof safe
  fix a b :: 'n
  from as[OF zero-less-one] guess B .. note B=conjunctD2[OF this,rule-format]
  let ?a = ∑ i∈Basis. min (a•i) (-B) *R i::'n
  let ?b = ∑ i∈Basis. max (b•i) B *R i::'n
  show ?f integrable-on cbox a b
  proof (rule integrable-subinterval[of - ?a ?b])
   have ball 0 B ⊆ cbox ?a ?b
    apply (rule subsetI)
    unfolding mem-ball mem-box dist-norm
   proof (rule, goal-cases)
    case (1 x i)
    then show ?case using Basis-le-norm[of i x]
     by (auto simp add:field-simps)
   qed
   from B(2)[OF this] guess z .. note conjunct1[OF this]
   then show ?f integrable-on cbox ?a ?b
    unfolding integrable-on-def by auto
   show cbox a b ⊆ cbox ?a ?b
    apply safe
    unfolding mem-box
    apply rule
    apply (erule-tac x=i in ballE)
    apply auto
    done
  qed

  fix e :: real
  assume e > 0
  from as[OF this] guess B .. note B=conjunctD2[OF this,rule-format]
  show ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
   norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) - i) < e
   apply rule
   apply rule
   apply (rule B)
   apply safe
```

```
    proof goal-cases
      case 1
      from B(2)[OF this] guess z .. note z=conjunctD2[OF this]
      from integral-unique[OF this(1)] show ?case
        using z(2) by auto
    qed
  qed
qed
```

35.58 Continuity of the integral (for a 1-dimensional interval).

```
lemma integrable-alt:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  shows f integrable-on s ⟷
    (∀ a b. (λx. if x ∈ s then f x else 0) integrable-on cbox a b) ∧
    (∀ e>0. ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d ⟶
    norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) −
      integral (cbox c d) (λx. if x ∈ s then f x else 0)) < e)
  (is ?l = ?r)
proof
  assume ?l
  then guess y unfolding integrable-on-def .. note this[unfolded has-integral-alt'[of
f]]
  note y=conjunctD2[OF this,rule-format]
  show ?r
    apply safe
    apply (rule y)
  proof goal-cases
    case (1 e)
    then have e/2 > 0
      by auto
    from y(2)[OF this] guess B .. note B=conjunctD2[OF this,rule-format]
    show ?case
      apply rule
      apply rule
      apply (rule B)
      apply safe
    proof goal-cases
      case prems: (1 a b c d)
      show ?case
        apply (rule norm-triangle-half-l)
        using B(2)[OF prems(1)] B(2)[OF prems(2)]
        apply auto
        done
    qed
  qed
next
  assume ?r
```

note *as* = *conjunctD2*[*OF this*,*rule-format*]
let *?cube* = $\lambda n.$ *cbox* ($\sum i{\in}$*Basis*. − *real n* $*_R$ *i*::′*n*) ($\sum i{\in}$*Basis*. *real n* $*_R$ *i*)
have *Cauchy* ($\lambda n.$ *integral* (*?cube n*) ($\lambda x.$ *if x* ∈ *s then f x else 0*))
proof (*unfold Cauchy-def*, *safe*, *goal-cases*)
  case (*1 e*)
  from *as*(*2*)[*OF this*] **guess** *B* **..** **note** *B* = *conjunctD2*[*OF this*,*rule-format*]
  from *real-arch-simple*[*of B*] **guess** *N* **..** **note** *N* = *this*
  {
    fix *n*
    assume *n*: *n* ≥ *N*
    have *ball 0 B* ⊆ *?cube n*
      apply (*rule subsetI*)
      unfolding *mem-ball mem-box dist-norm*
    proof (*rule*, *goal-cases*)
      case (*1 x i*)
      then show *?case*
        using *Basis-le-norm*[*of i x*] ‹*i*∈*Basis*›
        using *n N*
        by (*auto simp add*: *field-simps setsum-negf*)
    qed
  }
  then show *?case*
    apply −
    apply (*rule-tac x*=*N* **in** *exI*)
    apply *safe*
    unfolding *dist-norm*
    apply (*rule B*(*2*))
    apply *auto*
    done
qed
from *this*[*unfolded convergent-eq-cauchy*[*symmetric*]] **guess** *i* **..**
note *i* = *this*[*THEN LIMSEQ-D*]

show *?l* **unfolding** *integrable-on-def has-integral-alt*′[*of f*]
  apply (*rule-tac x*=*i* **in** *exI*)
  apply *safe*
  apply (*rule as*(*1*)[*unfolded integrable-on-def*])
proof *goal-cases*
  case (*1 e*)
  then have ∗: *e*/*2* > *0* **by** *auto*
  from *i*[*OF this*] **guess** *N* **..** **note** *N* =*this*[*rule-format*]
  from *as*(*2*)[*OF* ∗] **guess** *B* **..** **note** *B*=*conjunctD2*[*OF this*,*rule-format*]
  let *?B* = *max* (*real N*) *B*
  show *?case*
    apply (*rule-tac x*=*?B* **in** *exI*)
  proof *safe*
    show *0* < *?B*
      using *B*(*1*) **by** *auto*
    fix *a b* :: ′*n*

```
      assume ab: ball 0 ?B ⊆ cbox a b
      from real-arch-simple[of ?B] guess n .. note n=this
      show norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) − i) < e
        apply (rule norm-triangle-half-l)
        apply (rule B(2))
        defer
        apply (subst norm-minus-commute)
        apply (rule N[of n])
      proof safe
        show N ≤ n
          using n by auto
        fix x :: 'n
        assume x: x ∈ ball 0 B
        then have x ∈ ball 0 ?B
          by auto
        then show x ∈ cbox a b
          using ab by blast
        show x ∈ ?cube n
          using x
          unfolding mem-box mem-ball dist-norm
          apply −
        proof (rule, goal-cases)
          case (1 i)
          then show ?case
            using Basis-le-norm[of i x] ‹i ∈ Basis›
            using n
            by (auto simp add: field-simps setsum-negf)
        qed
      qed
    qed
  qed
qed

lemma integrable-altD:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes f integrable-on s
  shows ⋀a b. (λx. if x ∈ s then f x else 0) integrable-on cbox a b
    and ⋀e. e > 0 ⟹ ∃B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c
d ⟶
      norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) − integral (cbox c d)
(λx. if x ∈ s then f x else 0)) < e
  using assms[unfolded integrable-alt[of f]] by auto

lemma integrable-on-subcbox:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes f integrable-on s
    and cbox a b ⊆ s
  shows f integrable-on cbox a b
  apply (rule integrable-eq)
```

defer
apply (*rule integrable-altD*(*1*)[*OF assms*(*1*)])
using *assms*(*2*)
apply *auto*
done

35.59 A straddling criterion for integrability

lemma *integrable-straddle-interval*:
fixes *f* :: '*n*::*euclidean-space* ⇒ *real*
assumes ∀ *e*>*0*. ∃ *g h i j*. (*g has-integral i*) (*cbox a b*) ∧ (*h has-integral j*) (*cbox a b*) ∧
norm (*i* − *j*) < *e* ∧ (∀ *x*∈*cbox a b*. (*g x*) ≤ *f x* ∧ *f x* ≤ *h x*)
shows *f integrable-on cbox a b*
proof (*subst integrable-cauchy*, *safe*, *goal-cases*)
case (*1 e*)
then have *e*: *e*/*3* > *0*
by *auto*
note *assms*[*rule-format*,*OF this*]
then guess *g h i j* **by** (*elim exE conjE*) **note** *obt* = *this*
from *obt*(*1*)[*unfolded has-integral*[*of g*], *rule-format*, *OF e*] **guess** *d1* **.. note** *d1*=*conjunctD2*[*OF this*,*rule-format*]
from *obt*(*2*)[*unfolded has-integral*[*of h*], *rule-format*, *OF e*] **guess** *d2* **.. note** *d2*=*conjunctD2*[*OF this*,*rule-format*]
show *?case*
apply (*rule-tac x*=λ*x*. *d1 x* ∩ *d2 x* **in** *exI*)
apply (*rule conjI gauge-inter d1 d2*)+
unfolding *fine-inter*
proof (*safe*, *goal-cases*)
have ∗∗: ⋀*i j g1 g2 h1 h2 f1 f2*. *g1* − *h2* ≤ *f1* − *f2* ⟹ *f1* − *f2* ≤ *h1* − *g2* ⟹
|*i* − *j*| < *e* / *3* ⟹ |*g2* − *i*| < *e* / *3* ⟹ |*g1* − *i*| < *e* / *3* ⟹
|*h2* − *j*| < *e* / *3* ⟹ |*h1* − *j*| < *e* / *3* ⟹ |*f1* − *f2*| < *e*
using ‹*e* > *0*› **by** *arith*
case *prems*: (*1 p1 p2*)
note *tagged-division-ofD*(*2*−*4*) **note** ∗ = *this*[*OF prems*(*1*)] *this*[*OF prems*(*4*)]

have (∑ (*x*, *k*)∈*p1*. *content k* ∗*R f x*) − (∑ (*x*, *k*)∈*p1*. *content k* ∗*R g x*) ≥ *0*
and *0* ≤ (∑ (*x*, *k*)∈*p2*. *content k* ∗*R h x*) − (∑ (*x*, *k*)∈*p2*. *content k* ∗*R f x*)
and (∑ (*x*, *k*)∈*p2*. *content k* ∗*R f x*) − (∑ (*x*, *k*)∈*p2*. *content k* ∗*R g x*) ≥ *0*
and *0* ≤ (∑ (*x*, *k*)∈*p1*. *content k* ∗*R h x*) − (∑ (*x*, *k*)∈*p1*. *content k* ∗*R f x*)
unfolding *setsum-subtractf*[*symmetric*]
apply −
apply (*rule-tac*[!] *setsum-nonneg*)
apply *safe*
unfolding *real-scaleR-def right-diff-distrib*[*symmetric*]
apply (*rule-tac*[!] *mult-nonneg-nonneg*)
proof −
fix *a b*

```
      assume ab: (a, b) ∈ p1
      show 0 ≤ content b
        using ∗(3)[OF ab]
        apply safe
        apply (rule content-pos-le)
        done
      then show 0 ≤ content b .
      show 0 ≤ f a − g a 0 ≤ h a − f a
        using ∗(1−2)[OF ab]
        using obt(4)[rule-format,of a]
        by auto
    next
      fix a b
      assume ab: (a, b) ∈ p2
      show 0 ≤ content b
        using ∗(6)[OF ab]
        apply safe
        apply (rule content-pos-le)
        done
      then show 0 ≤ content b .
      show 0 ≤ f a − g a and 0 ≤ h a − f a
        using ∗(4−5)[OF ab] using obt(4)[rule-format,of a] by auto
    qed
    then show ?case
      apply −
      unfolding real-norm-def
      apply (rule ∗∗)
      defer
      defer
      unfolding real-norm-def[symmetric]
      apply (rule obt(3))
      apply (rule d1(2)[OF conjI[OF prems(4,5)]])
      apply (rule d1(2)[OF conjI[OF prems(1,2)]])
      apply (rule d2(2)[OF conjI[OF prems(4,6)]])
      apply (rule d2(2)[OF conjI[OF prems(1,3)]])
      apply auto
      done
  qed
qed

lemma integrable-straddle:
  fixes f :: 'n::euclidean-space ⇒ real
  assumes ∀ e>0. ∃ g h i j. (g has-integral i) s ∧ (h has-integral j) s ∧
    norm (i − j) < e ∧ (∀ x∈s. g x ≤ f x ∧ f x ≤ h x)
  shows f integrable-on s
proof −
  have ⋀a b. (λx. if x ∈ s then f x else 0) integrable-on cbox a b
  proof (rule integrable-straddle-interval, safe, goal-cases)
    case (1 a b e)
```

then have $*$: $e/4 > 0$
by *auto*
from *assms*[*rule-format*,*OF this*] **guess** $g\ h\ i\ j$ **by** (*elim exE conjE*) **note** *obt*=*this*
note *obt*(*1*)[*unfolded has-integral-alt'*[*of g*]]
note *conjunctD2*[*OF this*, *rule-format*]
note g = *this*(*1*) **and** *this*(*2*)[*OF* $*$]
from *this*(*2*) **guess** *B1* **..** **note** *B1* = *conjunctD2*[*OF this*,*rule-format*]
note *obt*(*2*)[*unfolded has-integral-alt'*[*of h*]]
note *conjunctD2*[*OF this*, *rule-format*]
note h = *this*(*1*) **and** *this*(*2*)[*OF* $*$]
from *this*(*2*) **guess** *B2* **..** **note** *B2* = *conjunctD2*[*OF this*,*rule-format*]
def $c \equiv \sum i{\in}Basis.\ min\ (a{\cdot}i)\ (-\ (max\ B1\ B2))\ *_R\ i::'n$
def $d \equiv \sum i{\in}Basis.\ max\ (b{\cdot}i)\ (max\ B1\ B2)\ *_R\ i::'n$
have $*$: *ball 0 B1* $\subseteq$ *cbox c d ball 0 B2* $\subseteq$ *cbox c d*
apply *safe*
unfolding *mem-ball mem-box dist-norm*
apply (*rule-tac*[!] *ballI*)
proof *goal-cases*
case (*1 x i*)
then show *?case* **using** *Basis-le-norm*[*of i x*]
unfolding *c-def d-def* **by** *auto*
next
case (*2 x i*)
then show *?case* **using** *Basis-le-norm*[*of i x*]
unfolding *c-def d-def* **by** *auto*
qed
have $**$: $\bigwedge ch\ cg\ ag\ ah::real.\ norm\ (ah - ag) \le norm\ (ch - cg) \Longrightarrow norm\ (cg - i) < e\ /\ 4 \Longrightarrow$
$norm\ (ch - j) < e\ /\ 4 \Longrightarrow norm\ (ag - ah) < e$
using *obt*(*3*)
unfolding *real-norm-def*
by *arith*
show *?case*
apply (*rule-tac x*=$\lambda x.$ *if* $x \in s$ *then g x else 0* **in** *exI*)
apply (*rule-tac x*=$\lambda x.$ *if* $x \in s$ *then h x else 0* **in** *exI*)
apply (*rule-tac x*=*integral* (*cbox a b*) ($\lambda x.$ *if* $x \in s$ *then g x else 0*) **in** *exI*)
apply (*rule-tac x*=*integral* (*cbox a b*) ($\lambda x.$ *if* $x \in s$ *then h x else 0*) **in** *exI*)
apply *safe*
apply (*rule-tac*[*1*−*2*] *integrable-integral*,*rule g*)
apply (*rule h*)
apply (*rule* $**$[*OF* - *B1*(*2*)[*OF* $*$(*1*)] *B2*(*2*)[*OF* $*$(*2*)]])
proof −
have $*$: $\bigwedge x\ f\ g.$ (*if* $x \in s$ *then f x else 0*) − (*if* $x \in s$ *then g x else 0*) =
(*if* $x \in s$ *then f x* − *g x else* (*0*::*real*))
by *auto*
note $**$ = *abs-of-nonneg*[*OF integral-nonneg*[*OF integrable-diff*, *OF h g*]]
show *norm* (*integral* (*cbox a b*) ($\lambda x.$ *if* $x \in s$ *then h x else 0*) −
integral (*cbox a b*) ($\lambda x.$ *if* $x \in s$ *then g x else 0*)) $\le$

```
      norm (integral (cbox c d) (λx. if x ∈ s then h x else 0) −
        integral (cbox c d) (λx. if x ∈ s then g x else 0))
      unfolding integral-diff[OF h g,symmetric] real-norm-def
      apply (subst **)
      defer
      apply (subst **)
      defer
      apply (rule has-integral-subset-le)
      defer
      apply (rule integrable-integral integrable-diff h g)+
    proof safe
      fix x
      assume x ∈ cbox a b
      then show x ∈ cbox c d
        unfolding mem-box c-def d-def
        apply −
        apply rule
        apply (erule-tac x=i in ballE)
        apply auto
        done
    qed (insert obt(4), auto)
  qed (insert obt(4), auto)
 qed
 note interv = this

 show ?thesis
  unfolding integrable-alt[of f]
  apply safe
  apply (rule interv)
 proof goal-cases
  case (1 e)
  then have *: e/3 > 0
   by auto
  from assms[rule-format,OF this] guess g h i j by (elim exE conjE) note
obt=this
  note obt(1)[unfolded has-integral-alt'[of g]]
  note conjunctD2[OF this, rule-format]
  note g = this(1) and this(2)[OF *]
  from this(2) guess B1 .. note B1 = conjunctD2[OF this,rule-format]
  note obt(2)[unfolded has-integral-alt'[of h]]
  note conjunctD2[OF this, rule-format]
  note h = this(1) and this(2)[OF *]
  from this(2) guess B2 .. note B2 = conjunctD2[OF this,rule-format]
  show ?case
   apply (rule-tac x=max B1 B2 in exI)
   apply safe
   apply (rule max.strict-coboundedI1)
   apply (rule B1)
  proof −
```

fix *a b c d* :: *'n*
assume *as*: *ball 0* (*max B1 B2*) ⊆ *cbox a b ball 0* (*max B1 B2*) ⊆ *cbox c d*
have ∗∗: *ball 0 B1* ⊆ *ball* (*0*::*'n*) (*max B1 B2*) *ball 0 B2* ⊆ *ball* (*0*::*'n*) (*max B1 B2*)
by *auto*
have ∗: ⋀*ga gc ha hc fa fc*::*real*.
|*ga* − *i*| < *e* / *3* ∧ |*gc* − *i*| < *e* / *3* ∧ |*ha* − *j*| < *e* / *3* ∧
|*hc* − *j*| < *e* / *3* ∧ |*i* − *j*| < *e* / *3* ∧ *ga* ≤ *fa* ∧ *fa* ≤ *ha* ∧ *gc* ≤ *fc* ∧ *fc* ≤ *hc* ⟹
|*fa* − *fc*| < *e*
by (*simp add*: *abs-real-def split*: *split-if-asm*)
show *norm* (*integral* (*cbox a b*) (λ*x*. *if x* ∈ *s then f x else 0*) − *integral* (*cbox c d*)
(λ*x*. *if x* ∈ *s then f x else 0*)) < *e*
unfolding *real-norm-def*
apply (*rule* ∗)
apply *safe*
unfolding *real-norm-def*[*symmetric*]
apply (*rule B1*(*2*))
apply (*rule order-trans*)
apply (*rule* ∗∗)
apply (*rule as*(*1*))
apply (*rule B1*(*2*))
apply (*rule order-trans*)
apply (*rule* ∗∗)
apply (*rule as*(*2*))
apply (*rule B2*(*2*))
apply (*rule order-trans*)
apply (*rule* ∗∗)
apply (*rule as*(*1*))
apply (*rule B2*(*2*))
apply (*rule order-trans*)
apply (*rule* ∗∗)
apply (*rule as*(*2*))
apply (*rule obt*)
apply (*rule-tac*[!] *integral-le*)
using *obt*
apply (*auto intro*!: *h g interv*)
done
qed
qed
qed

35.60 Adding integrals over several sets

lemma *has-integral-union*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes (*f has-integral i*) *s*
and (*f has-integral j*) *t*

```
    and negligible (s ∩ t)
  shows (f has-integral (i + j)) (s ∪ t)
proof −
  note ∗ = has-integral-restrict-univ[symmetric, of f]
  show ?thesis
    unfolding ∗
    apply (rule has-integral-spike[OF assms(3)])
    defer
    apply (rule has-integral-add[OF assms(1−2)[unfolded ∗]])
    apply auto
    done
qed

lemma has-integral-unions:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes finite t
    and ∀ s∈t. (f has-integral (i s)) s
    and ∀ s∈t. ∀ s'∈t. s ≠ s' ⟶ negligible (s ∩ s')
  shows (f has-integral (setsum i t)) (⋃ t)
proof −
  note ∗ = has-integral-restrict-univ[symmetric, of f]
  have ∗∗: negligible (⋃((λ(a,b). a ∩ b) ` {(a,b). a ∈ t ∧ b ∈ {y. y ∈ t ∧ a ≠
y}}))
    apply (rule negligible-unions)
    apply (rule finite-imageI)
    apply (rule finite-subset[of - t × t])
    defer
    apply (rule finite-cartesian-product[OF assms(1,1)])
    using assms(3)
    apply auto
    done
  note assms(2)[unfolded ∗]
  note has-integral-setsum[OF assms(1) this]
  then show ?thesis
    unfolding ∗
    apply −
    apply (rule has-integral-spike[OF ∗∗])
    defer
    apply assumption
    apply safe
  proof goal-cases
    case prems: (1 x)
    then show ?case
    proof (cases x ∈ ⋃ t)
      case True
      then guess s unfolding Union-iff .. note s=this
      then have ∗: ∀ b∈t. x ∈ b ⟷ b = s
        using prems(3) by blast
      show ?thesis
```

```
      unfolding if-P[OF True]
      apply (rule trans)
      defer
      apply (rule setsum.cong)
      apply (rule refl)
      apply (subst *)
      apply assumption
      apply (rule refl)
      unfolding setsum.delta[OF assms(1)]
      using s
      apply auto
      done
    qed auto
  qed
qed
```

In particular adding integrals over a division, maybe not of an interval.

```
lemma has-integral-combine-division:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes d division-of s
    and ∀k∈d. (f has-integral (i k)) k
  shows (f has-integral (setsum i d)) s
proof −
  note d = division-ofD[OF assms(1)]
  show ?thesis
    unfolding d(6)[symmetric]
    apply (rule has-integral-unions)
    apply (rule d assms)+
    apply rule
    apply rule
    apply rule
  proof goal-cases
    case prems: (1 s s')
     from d(4)[OF this(1)] d(4)[OF this(2)] guess a c b d by (elim exE) note
obt=this
    from d(5)[OF prems] show ?case
      unfolding obt interior-cbox
      apply −
      apply (rule negligible-subset[of (cbox a b−box a b) ∪ (cbox c d−box c d)])
      apply (rule negligible-union negligible-frontier-interval)+
      apply auto
      done
  qed
qed

lemma integral-combine-division-bottomup:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes d division-of s
    and ∀k∈d. f integrable-on k
```

shows *integral s f = setsum (λi. integral i f) d*
apply (*rule integral-unique*)
apply (*rule has-integral-combine-division*[*OF assms*(*1*)])
using *assms*(*2*)
unfolding *has-integral-integral*
apply *assumption*
done

lemma *has-integral-combine-division-topdown*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *f integrable-on s*
and *d division-of k*
and *k* ⊆ *s*
shows (*f has-integral* (*setsum* (*λi. integral i f*) *d*)) *k*
apply (*rule has-integral-combine-division*[*OF assms*(*2*)])
apply *safe*
unfolding *has-integral-integral*[*symmetric*]
proof *goal-cases*
case (*1 k*)
from *division-ofD*(*2*,*4*)[*OF assms*(*2*) *this*]
show *?case*
apply *safe*
apply (*rule integrable-on-subcbox*)
apply (*rule assms*)
using *assms*(*3*)
apply *auto*
done
qed

lemma *integral-combine-division-topdown*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *f integrable-on s*
and *d division-of s*
shows *integral s f = setsum* (*λi. integral i f*) *d*
apply (*rule integral-unique*)
apply (*rule has-integral-combine-division-topdown*)
using *assms*
apply *auto*
done

lemma *integrable-combine-division*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *d division-of s*
and ∀ *i*∈*d*. *f integrable-on i*
shows *f integrable-on s*
using *assms*(*2*)
unfolding *integrable-on-def*
by (*metis has-integral-combine-division*[*OF assms*(*1*)])

lemma *integrable-on-subdivision*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *d division-of i*
and *f integrable-on s*
and *i* ⊆ *s*
shows *f integrable-on i*
apply (*rule integrable-combine-division assms*)+
apply *safe*
proof *goal-cases*
case *1*
note *division-ofD*(*2*,*4*)[*OF assms*(*1*) *this*]
then show *?case*
apply *safe*
apply (*rule integrable-on-subcbox*[*OF assms*(*2*)])
using *assms*(*3*)
apply *auto*
done
qed

35.61 Also tagged divisions

lemma *has-integral-combine-tagged-division*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *p tagged-division-of s*
and ∀(*x*,*k*) ∈ *p*. (*f has-integral* (*i k*)) *k*
shows (*f has-integral* (*setsum* (λ(*x*,*k*). *i k*) *p*)) *s*
proof −
have ∗: (*f has-integral* (*setsum* (λ*k*. *integral k f*) (*snd* ‘ *p*))) *s*
apply (*rule has-integral-combine-division*)
apply (*rule division-of-tagged-division*[*OF assms*(*1*)])
using *assms*(*2*)
unfolding *has-integral-integral*[*symmetric*]
apply *safe*
apply *auto*
done
then show *?thesis*
apply −
apply (*rule subst*[**where** *P*=λ*i*. (*f has-integral i*) *s*])
defer
apply *assumption*
apply (*rule trans*[*of* - *setsum* (λ(*x*,*k*). *integral k f*) *p*])
apply (*subst eq-commute*)
apply (*rule setsum-over-tagged-division-lemma*[*OF assms*(*1*)])
apply (*rule integral-null*)
apply *assumption*
apply (*rule setsum.cong*)
using *assms*(*2*)
apply *auto*
done

qed

lemma *integral-combine-tagged-division-bottomup*:
fixes f :: *'n::euclidean-space* ⇒ *'a::banach*
assumes p *tagged-division-of* (*cbox a b*)
and ∀(*x*,*k*)∈p. f *integrable-on k*
shows *integral* (*cbox a b*) f = *setsum* (λ(*x*,*k*). *integral k f*) p
apply (*rule integral-unique*)
apply (*rule has-integral-combine-tagged-division*[*OF assms*(*1*)])
using *assms*(*2*)
apply *auto*
done

lemma *has-integral-combine-tagged-division-topdown*:
fixes f :: *'n::euclidean-space* ⇒ *'a::banach*
assumes f *integrable-on cbox a b*
and p *tagged-division-of* (*cbox a b*)
shows (*f has-integral* (*setsum* (λ(*x*,*k*). *integral k f*) p)) (*cbox a b*)
apply (*rule has-integral-combine-tagged-division*[*OF assms*(*2*)])
apply *safe*
proof *goal-cases*
case *1*
note *tagged-division-ofD*(*3*−*4*)[*OF assms*(*2*) *this*]
then show *?case*
using *integrable-subinterval*[*OF assms*(*1*)] **by** *blast*
qed

lemma *integral-combine-tagged-division-topdown*:
fixes f :: *'n::euclidean-space* ⇒ *'a::banach*
assumes f *integrable-on cbox a b*
and p *tagged-division-of* (*cbox a b*)
shows *integral* (*cbox a b*) f = *setsum* (λ(*x*,*k*). *integral k f*) p
apply (*rule integral-unique*)
apply (*rule has-integral-combine-tagged-division-topdown*)
using *assms*
apply *auto*
done

35.62 Henstock's lemma

lemma *henstock-lemma-part1*:
fixes f :: *'n::euclidean-space* ⇒ *'a::banach*
assumes f *integrable-on cbox a b*
and $e > 0$
and *gauge d*
and (∀ p. p *tagged-division-of* (*cbox a b*) ∧ d *fine* p ⟶
norm (*setsum* (λ(*x*,*k*). *content k* ∗$_R$ f *x*) p − *integral*(*cbox a b*) f) < e)
and p: p *tagged-partial-division-of* (*cbox a b*) d *fine* p
shows *norm* (*setsum* (λ(*x*,*k*). *content k* ∗$_R$ f *x* − *integral k f*) p) ≤ e

(**is** *?x* ≤ *e*)
proof −
{ presume ⋀*k*. *0*<*k* ⟹ *?x* ≤ *e* + *k* **then show** *?thesis* **by** (*blast intro*: *field-le-epsilon*) **}**
fix *k* :: *real*
assume *k*: *k* > *0*
note *p'* = *tagged-partial-division-ofD*[*OF p*(*1*)]
have ⋃(*snd* ' *p*) ⊆ *cbox a b*
using *p'*(*3*) **by** *fastforce*
note *partial-division-of-tagged-division*[*OF p*(*1*)] *this*
from *partial-division-extend-interval*[*OF this*] **guess** *q* **. note** *q*=*this* **and** *q'* = *division-ofD*[*OF this*(*2*)]
def *r* ≡ *q* − *snd* ' *p*
have *snd* ' *p* ∩ *r* = {}
unfolding *r-def* **by** *auto*
have *r*: *finite r*
using *q'* **unfolding** *r-def* **by** *auto*

have ∀ *i*∈*r*. ∃ *p*. *p tagged-division-of i* ∧ *d fine p* ∧
norm (*setsum* (λ(*x*,*j*). *content j* ∗*R f x*) *p* − *integral i f*) < *k* / (*real* (*card r*) + *1*)
apply *safe*
proof *goal-cases*
case (*1 i*)
then have *i*: *i* ∈ *q*
unfolding *r-def* **by** *auto*
from *q'*(*4*)[*OF this*] **guess** *u v* **by** (*elim exE*) **note** *uv*=*this*
have ∗: *k* / (*real* (*card r*) + *1*) > *0* **using** *k* **by** *simp*
have *f integrable-on cbox u v*
apply (*rule integrable-subinterval*[*OF assms*(*1*)])
using *q'*(*2*)[*OF i*]
unfolding *uv*
apply *auto*
done
note *integrable-integral*[*OF this*, *unfolded has-integral*[*of f*]]
from *this*[*rule-format*,*OF* ∗] **guess** *dd* **.. note** *dd*=*conjunctD2*[*OF this*,*rule-format*]
note *gauge-inter*[*OF* ‹*gauge d*› *dd*(*1*)]
from *fine-division-exists*[*OF this*,*of u v*] **guess** *qq* **.**
then show *?case*
apply (*rule-tac x*=*qq* **in** *exI*)
using *dd*(*2*)[*of qq*]
unfolding *fine-inter uv*
apply *auto*
done
qed
from *bchoice*[*OF this*] **guess** *qq* **.. note** *qq*=*this*[*rule-format*]

let *?p* = *p* ∪ ⋃(*qq* ' *r*)
have *norm* ((∑ (*x*, *k*)∈*?p*. *content k* ∗*R f x*) − *integral* (*cbox a b*) *f*) < *e*

```
  apply (rule assms(4)[rule-format])
 proof
  show d fine ?p
   apply (rule fine-union)
   apply (rule p)
   apply (rule fine-unions)
   using qq
   apply auto
   done
  note ∗ = tagged-partial-division-of-union-self[OF p(1)]
  have p ∪ ⋃(qq ‘ r) tagged-division-of ⋃(snd ‘ p) ∪ ⋃r
   using r
  proof (rule tagged-division-union[OF ∗ tagged-division-unions], goal-cases)
   case 1
   then show ?case
    using qq by auto
  next
   case 2
   then show ?case
    apply rule
    apply rule
    apply rule
    apply(rule q′(5))
    unfolding r-def
    apply auto
    done
  next
   case 3
   then show ?case
    apply (rule inter-interior-unions-intervals)
    apply fact
    apply rule
    apply rule
    apply (rule q′)
    defer
    apply rule
    apply (subst Int-commute)
    apply (rule inter-interior-unions-intervals)
    apply (rule finite-imageI)
    apply (rule p′)
    apply rule
    defer
    apply rule
    apply (rule q′)
    using q(1) p′
    unfolding r-def
    apply auto
    done
  qed
```

```
    moreover have ⋃(snd ‘ p) ∪ ⋃r = cbox a b and {qq i |i. i ∈ r} = qq ‘ r
      unfolding Union-Un-distrib[symmetric] r-def
      using q
      by auto
    ultimately show ?p tagged-division-of (cbox a b)
      by fastforce
  qed

  then have norm ((∑(x, k)∈p. content k *R f x) + (∑(x, k)∈⋃(qq ‘ r). content
k *R f x) −
    integral (cbox a b) f) < e
    apply (subst setsum.union-inter-neutral[symmetric])
    apply (rule p')
    prefer 3
    apply assumption
    apply rule
    apply (rule finite-imageI)
    apply (rule r)
    apply safe
    apply (drule qq)
  proof −
    fix x l k
    assume as: (x, l) ∈ p (x, l) ∈ qq k k ∈ r
    note qq[OF this(3)]
    note tagged-division-ofD(3,4)[OF conjunct1[OF this] as(2)]
    from this(2) guess u v by (elim exE) note uv=this
    have l∈snd ‘ p unfolding image-iff apply(rule-tac x=(x,l) in bexI) using as
by auto
    then have l ∈ q k ∈ q l ≠ k
      using as(1,3) q(1) unfolding r-def by auto
    note q'(5)[OF this]
    then have interior l = {}
      using interior-mono[OF ‹l ⊆ k›] by blast
    then show content l *R f x = 0
      unfolding uv content-eq-0-interior[symmetric] by auto
  qed auto

  then have norm ((∑(x, k)∈p. content k *R f x) + setsum (setsum (λ(x, k).
content k *R f x))
    (qq ‘ r) − integral (cbox a b) f) < e
    apply (subst (asm) setsum.Union-comp)
    prefer 2
    unfolding split-paired-all split-conv image-iff
    apply (erule bexE)+
  proof −
    fix x m k l T1 T2
    assume (x, m) ∈ T1 (x, m) ∈ T2 T1 ≠ T2 k ∈ r l ∈ r T1 = qq k T2 = qq l
    note as = this(1−5)[unfolded this(6−)]
    note kl = tagged-division-ofD(3,4)[OF qq[THEN conjunct1]]
```

```
    from this(2)[OF as(4,1)] guess u v by (elim exE) note uv=this
    have ∗: interior (k ∩ l) = {}
      unfolding interior-Int
      apply (rule q')
      using as
      unfolding r-def
      apply auto
      done
    have interior m = {}
      unfolding subset-empty[symmetric]
      unfolding ∗[symmetric]
      apply (rule interior-mono)
      using kl(1)[OF as(4,1)] kl(1)[OF as(5,2)]
      apply auto
      done
    then show content m ∗R f x = 0
      unfolding uv content-eq-0-interior[symmetric]
      by auto
  qed (insert qq, auto)

  then have ∗∗: norm ((∑(x, k)∈p. content k ∗R f x) + setsum (setsum (λ(x,
k). content k ∗R f x) ∘ qq) r −
    integral (cbox a b) f) < e
    apply (subst (asm) setsum.reindex-nontrivial)
    apply fact
    apply (rule setsum.neutral)
    apply rule
    unfolding split-paired-all split-conv
    defer
    apply assumption
  proof −
    fix k l x m
    assume as: k ∈ r l ∈ r k ≠ l qq k = qq l (x, m) ∈ qq k
    note tagged-division-ofD(6)[OF qq[THEN conjunct1]]
    from this[OF as(1)] this[OF as(2)] show content m ∗R f x = 0
      using as(3) unfolding as by auto
  qed

  have ∗: norm (cp − ip) ≤ e + k
    if norm ((cp + cr) − i) < e
    and norm (cr − ir) < k
    and ip + ir = i
    for ir ip i cr cp
  proof −
    from that show ?thesis
      using norm-triangle-le[of cp + cr − i − (cr − ir)]
      unfolding that(3)[symmetric] norm-minus-cancel
      by (auto simp add: algebra-simps)
  qed
```

```
  have ?x = norm ((∑(x, k)∈p. content k *R f x) - (∑(x, k)∈p. integral k f))
    unfolding split-def setsum-subtractf ..
  also have ... ≤ e + k
    apply (rule *[OF **, where ir1=setsum (λk. integral k f) r])
  proof goal-cases
    case 1
    have *: k * real (card r) / (1 + real (card r)) < k
      using k by (auto simp add: field-simps)
    show ?case
      apply (rule le-less-trans[of - setsum (λx. k / (real (card r) + 1)) r])
      unfolding setsum-subtractf[symmetric]
      apply (rule setsum-norm-le)
      apply rule
      apply (drule qq)
      defer
      unfolding divide-inverse setsum-left-distrib[symmetric]
      unfolding divide-inverse[symmetric]
      using * apply (auto simp add: field-simps)
      done
  next
    case 2
    have *: (∑(x, k)∈p. integral k f) = (∑k∈snd ` p. integral k f)
      apply (subst setsum.reindex-nontrivial)
      apply fact
      unfolding split-paired-all snd-conv split-def o-def
    proof -
      fix x l y m
      assume as: (x, l) ∈ p (y, m) ∈ p (x, l) ≠ (y, m) l = m
      from p'(4)[OF as(1)] guess u v by (elim exE) note uv=this
      show integral l f = 0
        unfolding uv
        apply (rule integral-unique)
        apply (rule has-integral-null)
        unfolding content-eq-0-interior
        using p'(5)[OF as(1-3)]
        unfolding uv as(4)[symmetric]
        apply auto
        done
    qed auto
    from q(1) have **: snd ` p ∪ q = q by auto
    show ?case
      unfolding integral-combine-division-topdown[OF assms(1) q(2)] * r-def
     using ** q'(1) p'(1) setsum.union-disjoint [of snd ` p q - snd ` p λk. integral
k f, symmetric]
      by simp
  qed
  finally show ?x ≤ e + k .
qed
```

lemma *henstock-lemma-part2*:
fixes *f* :: *'m*::*euclidean-space* ⇒ *'n*::*euclidean-space*
assumes *f integrable-on cbox a b*
and *e* > *0*
and *gauge d*
and ∀ *p*. *p tagged-division-of* (*cbox a b*) ∧ *d fine p* ⟶
norm (*setsum* (λ(*x*,*k*). *content k* ∗*R f x*) *p* − *integral* (*cbox a b*) *f*) < *e*
and *p tagged-partial-division-of* (*cbox a b*)
and *d fine p*
shows *setsum* (λ(*x*,*k*). *norm* (*content k* ∗*R f x* − *integral k f*)) *p* ≤ *2* ∗ *real* (*DIM*(*'n*)) ∗ *e*
unfolding *split-def*
apply (*rule setsum-norm-allsubsets-bound*)
defer
apply (*rule henstock-lemma-part1*[*unfolded split-def*,*OF assms*(*1*−*3*)])
apply *safe*
apply (*rule assms*[*rule-format*,*unfolded split-def*])
defer
apply (*rule tagged-partial-division-subset*)
apply (*rule assms*)
apply *assumption*
apply (*rule fine-subset*)
apply *assumption*
apply (*rule assms*)
using *assms*(*5*)
apply *auto*
done

lemma *henstock-lemma*:
fixes *f* :: *'m*::*euclidean-space* ⇒ *'n*::*euclidean-space*
assumes *f integrable-on cbox a b*
and *e* > *0*
obtains *d* **where** *gauge d*
and ∀ *p*. *p tagged-partial-division-of* (*cbox a b*) ∧ *d fine p* ⟶
setsum (λ(*x*,*k*). *norm*(*content k* ∗*R f x* − *integral k f*)) *p* < *e*
proof −
have ∗: *e* / (*2* ∗ (*real DIM*(*'n*) + *1*)) > *0* **using** *assms*(*2*) **by** *simp*
from *integrable-integral*[*OF assms*(*1*),*unfolded has-integral*[*of f*],*rule-format*,*OF this*]
guess *d* **..** **note** *d* = *conjunctD2*[*OF this*]
show *thesis*
apply (*rule that*)
apply (*rule d*)
proof (*safe*, *goal-cases*)
case (*1 p*)
note ∗ = *henstock-lemma-part2*[*OF assms*(*1*) ∗ *d this*]
show *?case*
apply (*rule le-less-trans*[*OF* ∗])

```
    using ‹e > 0›
    apply (auto simp add: field-simps)
    done
 qed
qed
```

35.63 Geometric progression

FIXME: Should one or more of these theorems be moved to `~~/src/HOL/Set_Interval.thy`, alongside *geometric-sum*?

```
lemma sum-gp-basic:
 fixes x :: 'a::ring-1
 shows (1 − x) ∗ setsum (λi. x^i) {0 .. n} = (1 − x^(Suc n))
proof −
 def y ≡ 1 − x
 have y ∗ (∑ i=0..n. (1 − y) ^ i) = 1 − (1 − y) ^ Suc n
   by (induct n) (simp-all add: algebra-simps)
 then show ?thesis
   unfolding y-def by simp
qed

lemma sum-gp-multiplied:
 assumes mn: m ≤ n
 shows ((1::'a::{field}) − x) ∗ setsum (op ^ x) {m..n} = x^m − x^ Suc n
 (is ?lhs = ?rhs)
proof −
 let ?S = {0..(n − m)}
 from mn have mn': n − m ≥ 0
   by arith
 let ?f = op + m
 have i: inj-on ?f ?S
   unfolding inj-on-def by auto
 have f: ?f ‘ ?S = {m..n}
   using mn
   apply (auto simp add: image-iff Bex-def)
   apply presburger
   done
 have th: op ^ x ∘ op + m = (λi. x^m ∗ x^i)
   by (rule ext) (simp add: power-add power-mult)
 from setsum.reindex[OF i, of op ^ x, unfolded f th setsum-right-distrib[symmetric]]
 have ?lhs = x^m ∗ ((1 − x) ∗ setsum (op ^ x) {0..n − m})
   by simp
 then show ?thesis
   unfolding sum-gp-basic
   using mn
   by (simp add: field-simps power-add[symmetric])
qed

lemma sum-gp:
```

```
  setsum (op ^ (x::'a::{field})) {m .. n} =
    (if n < m then 0
     else if x = 1 then of-nat ((n + 1) - m)
     else (x^ m - x^ (Suc n)) / (1 - x))
proof -
  {
    assume nm: n < m
    then have ?thesis by simp
  }
  moreover
  {
    assume ¬ n < m
    then have nm: m ≤ n
      by arith
    {
      assume x: x = 1
      then have ?thesis
        by simp
    }
    moreover
    {
      assume x: x ≠ 1
      then have nz: 1 - x ≠ 0
        by simp
      from sum-gp-multiplied[OF nm, of x] nz have ?thesis
        by (simp add: field-simps)
    }
    ultimately have ?thesis by blast
  }
  ultimately show ?thesis by blast
qed

lemma sum-gp-offset:
  setsum (op ^ (x::'a::{field})) {m .. m+n} =
    (if x = 1 then of-nat n + 1 else x^m * (1 - x^Suc n) / (1 - x))
  unfolding sum-gp[of x m m + n] power-Suc
  by (simp add: field-simps power-add)
```

35.64 Monotone convergence (bounded interval first)

```
lemma monotone-convergence-interval:
  fixes f :: nat ⇒ 'n::euclidean-space ⇒ real
  assumes ∀ k. (f k) integrable-on cbox a b
    and ∀ k. ∀ x∈cbox a b.(f k x) ≤ f (Suc k) x
    and ∀ x∈cbox a b. ((λk. f k x) ⟶ g x) sequentially
    and bounded {integral (cbox a b) (f k) | k . k ∈ UNIV}
  shows g integrable-on cbox a b ∧ ((λk. integral (cbox a b) (f k)) ⟶ integral
(cbox a b) g) sequentially
proof (cases content (cbox a b) = 0)
```

case *True*
show *?thesis*
using *integrable-on-null*[*OF True*]
unfolding *integral-null*[*OF True*]
using *tendsto-const*
by *auto*
next
case *False*
have *fg*: $\forall x{\in}cbox\ a\ b.\ \forall k.\ (f\ k\ x) \cdot 1 \leq (g\ x) \cdot 1$
proof *safe*
fix *x k*
assume *x*: $x \in cbox\ a\ b$
note $* = Lim\text{-}component\text{-}ge[OF\ assms(3)[rule\text{-}format,\ OF\ x]\ trivial\text{-}limit\text{-}sequentially]$
show $f\ k\ x \cdot 1 \leq g\ x \cdot 1$
apply (*rule* $*$)
unfolding *eventually-sequentially*
apply (*rule-tac x=k* **in** *exI*)
apply −
apply (*rule transitive-stepwise-le*)
using *assms*(*2*)[*rule-format*, *OF x*]
apply *auto*
done
qed
have $\exists i.\ ((\lambda k.\ integral\ (cbox\ a\ b)\ (f\ k)) \longrightarrow i)\ sequentially$
apply (*rule bounded-increasing-convergent*)
defer
apply *rule*
apply (*rule integral-le*)
apply *safe*
apply (*rule assms*(*1*−*2*)[*rule-format*])+
using *assms*(*4*)
apply *auto*
done
then guess *i* **..** **note** *i=this*
have *i'*: $\bigwedge k.\ (integral(cbox\ a\ b)\ (f\ k)) \leq i{\cdot}1$
apply (*rule Lim-component-ge*)
apply (*rule i*)
apply (*rule trivial-limit-sequentially*)
unfolding *eventually-sequentially*
apply (*rule-tac x=k* **in** *exI*)
apply (*rule transitive-stepwise-le*)
prefer *3*
unfolding *inner-simps real-inner-1-right*
apply (*rule integral-le*)
apply (*rule assms*(*1*−*2*)[*rule-format*])+
using *assms*(*2*)
apply *auto*
done

```
  have (g has-integral i) (cbox a b)
    unfolding has-integral
  proof (safe, goal-cases)
    case e: (1 e)
    then have ∀ k. (∃ d. gauge d ∧ (∀ p. p tagged-division-of (cbox a b) ∧ d fine p
⟶
      norm ((∑ (x, ka)∈p. content ka *R f k x) − integral (cbox a b) (f k)) < e /
2 ^ (k + 2)))
      apply −
      apply rule
      apply (rule assms(1)[unfolded has-integral-integral has-integral,rule-format])
      apply auto
      done
   from choice[OF this] guess c .. note c=conjunctD2[OF this[rule-format],rule-format]

    have ∃ r. ∀ k≥r. 0 ≤ i·1 − (integral (cbox a b) (f k)) ∧ i·1 − (integral (cbox
a b) (f k)) < e / 4
    proof −
      have e/4 > 0
        using e by auto
      from LIMSEQ-D [OF i this] guess r ..
      then show ?thesis
        apply (rule-tac x=r in exI)
        apply rule
        apply (erule-tac x=k in allE)
        subgoal for k using i'[of k] by auto
        done
    qed
    then guess r .. note r=conjunctD2[OF this[rule-format]]

    have ∀ x∈cbox a b. ∃ n≥r. ∀ k≥n. 0 ≤ (g x)·1 − (f k x)·1 ∧
      (g x)·1 − (f k x)·1 < e / (4 * content(cbox a b))
    proof (rule, goal-cases)
      case prems: (1 x)
      have e / (4 * content (cbox a b)) > 0
        using ‹e>0› False content-pos-le[of a b] by auto
      from assms(3)[rule-format, OF prems, THEN LIMSEQ-D, OF this]
      guess n .. note n=this
      then show ?case
        apply (rule-tac x=n + r in exI)
        apply safe
        apply (erule-tac[2−3] x=k in allE)
        unfolding dist-real-def
        using fg[rule-format, OF prems]
        apply (auto simp add: field-simps)
        done
    qed
   from bchoice[OF this] guess m .. note m=conjunctD2[OF this[rule-format],rule-format]
    def d ≡ λx. c (m x) x
```

```
    show ?case
      apply (rule-tac x=d in exI)
    proof safe
      show gauge d
        using c(1) unfolding gauge-def d-def by auto
    next
      fix p
      assume p: p tagged-division-of (cbox a b) d fine p
      note p'=tagged-division-ofD[OF p(1)]
      have ∃ a. ∀ x∈p. m (fst x) ≤ a
        by (metis finite-imageI finite-nat-set-iff-bounded-le p'(1) rev-image-eqI)
      then guess s .. note s=this
      have *: ∀ a b c d. norm(a − b) ≤ e / 4 ∧ norm(b − c) < e / 2 ∧
        norm (c − d) < e / 4 ⟶ norm (a − d) < e
      proof (safe, goal-cases)
        case (1 a b c d)
        then show ?case
          using norm-triangle-lt[of a − b b − c 3* e/4]
            norm-triangle-lt[of a − b + (b − c) c − d e]
          unfolding norm-minus-cancel
          by (auto simp add: algebra-simps)
      qed
      show norm ((∑(x, k)∈p. content k *R g x) − i) < e
        apply (rule *[rule-format,where
          b=∑(x, k)∈p. content k *R f (m x) x and c=∑(x, k)∈p. integral k (f
(m x))])
      proof (safe, goal-cases)
        case 1
        show ?case
          apply (rule order-trans[of - ∑(x, k)∈p. content k * (e / (4 * content
(cbox a b)))])
          unfolding setsum-subtractf[symmetric]
          apply (rule order-trans)
          apply (rule norm-setsum)
          apply (rule setsum-mono)
          unfolding split-paired-all split-conv
       unfolding split-def setsum-left-distrib[symmetric] scaleR-diff-right[symmetric]
          unfolding additive-content-tagged-division[OF p(1), unfolded split-def]
        proof −
          fix x k
          assume xk: (x, k) ∈ p
          then have x: x ∈ cbox a b
            using p'(2−3)[OF xk] by auto
          from p'(4)[OF xk] guess u v by (elim exE) note uv=this
           show norm (content k *R (g x − f (m x) x)) ≤ content k * (e / (4 *
content (cbox a b)))
            unfolding norm-scaleR uv
            unfolding abs-of-nonneg[OF content-pos-le]
```

```
          apply (rule mult-left-mono)
          using m(2)[OF x,of m x]
          apply auto
          done
      qed (insert False, auto)

    next
      case 2
      show ?case
        apply (rule le-less-trans[of - norm (∑j = 0..s.
          ∑(x, k)∈{xk∈p. m (fst xk) = j}. content k *R f (m x) x − integral k
(f (m x)))])
        apply (subst setsum-group)
        apply fact
        apply (rule finite-atLeastAtMost)
        defer
        apply (subst split-def)+
        unfolding setsum-subtractf
        apply rule
      proof −
        show norm (∑j = 0..s. ∑(x, k)∈{xk ∈ p.
          m (fst xk) = j}. content k *R f (m x) x − integral k (f (m x))) < e / 2
          apply (rule le-less-trans[of - setsum (λi. e / 2^(i+2)) {0..s}])
          apply (rule setsum-norm-le)
        proof
          show (∑i = 0..s. e / 2 ^ (i + 2)) < e / 2
            unfolding power-add divide-inverse inverse-mult-distrib
          unfolding setsum-right-distrib[symmetric] setsum-left-distrib[symmetric]
            unfolding power-inverse [symmetric] sum-gp
            apply(rule mult-strict-left-mono[OF - e])
            unfolding power2-eq-square
            apply auto
            done
          fix t
          assume t ∈ {0..s}
          show norm (∑(x, k)∈{xk ∈ p. m (fst xk) = t}. content k *R f (m x)
x −
            integral k (f (m x))) ≤ e / 2 ^ (t + 2)
            apply (rule order-trans
              [of - norm (setsum (λ(x,k). content k *R f t x − integral k (f t)) {xk
∈ p. m (fst xk) = t})])
            apply (rule eq-refl)
            apply (rule arg-cong[where f=norm])
            apply (rule setsum.cong)
            apply (rule refl)
            defer
            apply (rule henstock-lemma-part1)
            apply (rule assms(1)[rule-format])
            apply (simp add: e)
```

```
            apply safe
            apply (rule c)+
            apply rule
            apply assumption+
            apply (rule tagged-partial-division-subset[of p])
            apply (rule p(1)[unfolded tagged-division-of-def, THEN conjunct1])
            defer
            unfolding fine-def
            apply safe
            apply (drule p(2)[unfolded fine-def, rule-format])
            unfolding d-def
            apply auto
            done
          qed
        qed (insert s, auto)
      next
        case 3
      note comb = integral-combine-tagged-division-topdown[OF assms(1)[rule-format]
p(1)]
        have *: ⋀sr sx ss ks kr::real. kr = sr ⟶ ks = ss ⟶
          ks ≤ i ∧ sr ≤ sx ∧ sx ≤ ss ∧ 0 ≤ i•1 − kr•1 ∧ i•1 − kr•1 < e/4 ⟶
|sx − i| < e/4
          by auto
        show ?case
          unfolding real-norm-def
          apply (rule *[rule-format])
          apply safe
          apply (rule comb[of r])
          apply (rule comb[of s])
          apply (rule i'[unfolded real-inner-1-right])
          apply (rule-tac[1−2] setsum-mono)
          unfolding split-paired-all split-conv
          apply (rule-tac[1−2] integral-le[OF ])
        proof safe
          show 0 ≤ i•1 − (integral (cbox a b) (f r))•1
            using r(1) by auto
          show i•1 − (integral (cbox a b) (f r))•1 < e / 4
            using r(2) by auto
          fix x k
          assume xk: (x, k) ∈ p
          from p'(4)[OF this] guess u v by (elim exE) note uv=this
          show f r integrable-on k
            and f s integrable-on k
            and f (m x) integrable-on k
            and f (m x) integrable-on k
            unfolding uv
            apply (rule-tac[!] integrable-on-subcbox[OF assms(1)[rule-format]])
            using p'(3)[OF xk]
            unfolding uv
```

```
        apply auto
        done
      fix y
      assume y ∈ k
      then have y ∈ cbox a b
        using p'(3)[OF xk] by auto
      then have *: ⋀m. ∀n≥m. f m y ≤ f n y
        apply -
        apply (rule transitive-stepwise-le)
        using assms(2)
        apply auto
        done
      show f r y ≤ f (m x) y and f (m x) y ≤ f s y
        apply (rule-tac[!] *[rule-format])
        using s[rule-format,OF xk] m(1)[of x] p'(2-3)[OF xk]
        apply auto
        done
    qed
   qed
  qed
 qed note * = this

 have integral (cbox a b) g = i
  by (rule integral-unique) (rule *)
 then show ?thesis
  using i * by auto
qed

lemma monotone-convergence-increasing:
 fixes f :: nat ⇒ 'n::euclidean-space ⇒ real
 assumes ∀k. (f k) integrable-on s
  and ∀k. ∀x∈s. (f k x) ≤ (f (Suc k) x)
  and ∀x∈s. ((λk. f k x) ⟶ g x) sequentially
  and bounded {integral s (f k)| k. True}
 shows g integrable-on s ∧ ((λk. integral s (f k)) ⟶ integral s g) sequentially
proof -
 have lem: g integrable-on s ∧ ((λk. integral s (f k)) ⟶ integral s g) sequentially
  if ∀k. ∀x∈s. 0 ≤ f k x
  and ∀k. (f k) integrable-on s
  and ∀k. ∀x∈s. f k x ≤ f (Suc k) x
  and ∀x∈s. ((λk. f k x) ⟶ g x) sequentially
  and bounded {integral s (f k)| k. True}
  for f :: nat ⇒ 'n::euclidean-space ⇒ real and g s
 proof -
  note assms=that[rule-format]
  have ∀x∈s. ∀k. (f k x)•1 ≤ (g x)•1
   apply safe
   apply (rule Lim-component-ge)
   apply (rule that(4)[rule-format])
```

apply *assumption*
apply (*rule trivial-limit-sequentially*)
unfolding *eventually-sequentially*
apply (*rule-tac x=k* **in** *exI*)
apply (*rule transitive-stepwise-le*)
using *that(3)*
apply *auto*
done
note *fg=this*[*rule-format*]

have $\exists i.\ ((\lambda k.\ \mathit{integral}\ s\ (f\ k)) \longrightarrow i)\ \mathit{sequentially}$
apply (*rule bounded-increasing-convergent*)
apply (*rule that(5)*)
apply *rule*
apply (*rule integral-le*)
apply (*rule that(2)*[*rule-format*])+
using *that(3)*
apply *auto*
done
then guess *i* **..** **note** *i=this*
have $\bigwedge k.\ \forall x{\in}s.\ \forall n{\geq}k.\ f\ k\ x \leq f\ n\ x$
apply *rule*
apply (*rule transitive-stepwise-le*)
using *that(3)*
apply *auto*
done
then have *i'*: $\forall k.\ (\mathit{integral}\ s\ (f\ k))\cdot 1 \leq i\cdot 1$
apply −
apply *rule*
apply (*rule Lim-component-ge*)
apply (*rule i*)
apply (*rule trivial-limit-sequentially*)
unfolding *eventually-sequentially*
apply (*rule-tac x=k* **in** *exI*)
apply *safe*
apply (*rule integral-component-le*)
apply *simp*
apply (*rule that(2)*[*rule-format*])+
apply *auto*
done

note *int* = *assms(2)*[*unfolded integrable-alt*[*of - s*],*THEN conjunct1*,*rule-format*]
have *ifif*: $\bigwedge k\ t.\ (\lambda x.\ \mathit{if}\ x \in t\ \mathit{then\ if}\ x \in s\ \mathit{then}\ f\ k\ x\ \mathit{else}\ 0\ \mathit{else}\ 0) =$
$(\lambda x.\ \mathit{if}\ x \in t \cap s\ \mathit{then}\ f\ k\ x\ \mathit{else}\ 0)$
by (*rule ext*) *auto*
have *int'*: $\bigwedge k\ a\ b.\ f\ k\ \mathit{integrable\text{-}on}\ \mathit{cbox}\ a\ b \cap s$
apply (*subst integrable-restrict-univ*[*symmetric*])
apply (*subst ifif*[*symmetric*])
apply (*subst integrable-restrict-univ*)

```
    apply (rule int)
    done
  have ⋀a b. (λx. if x ∈ s then g x else 0) integrable-on cbox a b ∧
    ((λk. integral (cbox a b) (λx. if x ∈ s then f k x else 0)) ⟶
    integral (cbox a b) (λx. if x ∈ s then g x else 0)) sequentially
  proof (rule monotone-convergence-interval, safe, goal-cases)
    case 1
    show ?case by (rule int)
  next
    case (2 - - - x)
    then show ?case
      apply (cases x ∈ s)
      using assms(3)
      apply auto
      done
  next
    case (3 - - x)
    then show ?case
      apply (cases x ∈ s)
      using assms(4)
      apply auto
      done
  next
    case (4 a b)
    note * = integral-nonneg
    have ⋀k. norm (integral (cbox a b) (λx. if x ∈ s then f k x else 0)) ≤ norm
(integral s (f k))
      unfolding real-norm-def
      apply (subst abs-of-nonneg)
      apply (rule *[OF int])
      apply safe
      apply (case-tac x ∈ s)
      apply (drule assms(1))
      prefer 3
      apply (subst abs-of-nonneg)
      apply (rule *[OF assms(2) that(1)[THEN spec]])
      apply (subst integral-restrict-univ[symmetric,OF int])
      unfolding ifif
      unfolding integral-restrict-univ[OF int']
      apply (rule integral-subset-le[OF - int' assms(2)])
      using assms(1)
      apply auto
      done
    then show ?case
      using assms(5)
      unfolding bounded-iff
      apply safe
      apply (rule-tac x=aa in exI)
      apply safe
```

```
      apply (erule-tac x=integral s (f k) in ballE)
      apply (rule order-trans)
      apply assumption
      apply auto
      done
  qed
  note g = conjunctD2[OF this]

  have (g has-integral i) s
    unfolding has-integral-alt'
    apply safe
    apply (rule g(1))
  proof goal-cases
    case (1 e)
    then have e/4>0
      by auto
    from LIMSEQ-D [OF i this] guess N .. note N=this
    note assms(2)[of N,unfolded has-integral-integral has-integral-alt'[of f N]]
      from this[THEN conjunct2,rule-format,OF ‹e/4>0›] guess B .. note
B=conjunctD2[OF this]
    show ?case
      apply rule
      apply rule
      apply (rule B)
      apply safe
    proof −
      fix a b :: 'n
      assume ab: ball 0 B ⊆ cbox a b
      from ‹e > 0› have e/2 > 0
        by auto
      from LIMSEQ-D [OF g(2)[of a b] this] guess M .. note M=this
      have **: norm (integral (cbox a b) (λx. if x ∈ s then f N x else 0) − i) <
e/2
        apply (rule norm-triangle-half-l)
        using B(2)[rule-format,OF ab] N[rule-format,of N]
        apply −
        defer
        apply (subst norm-minus-commute)
        apply auto
        done
      have *: ⋀f1 f2 g. |f1 − i| < e / 2 ⟶ |f2 − g| < e / 2 ⟶
        f1 ≤ f2 ⟶ f2 ≤ i ⟶ |g − i| < e
        unfolding real-inner-1-right by arith
      show norm (integral (cbox a b) (λx. if x ∈ s then g x else 0) − i) < e
        unfolding real-norm-def
        apply (rule *[rule-format])
        apply (rule **[unfolded real-norm-def])
        apply (rule M[rule-format,of M + N,unfolded real-norm-def])
        apply (rule le-add1)
```

```
        apply (rule integral-le[OF int int])
        defer
      apply (rule order-trans[OF - i'[rule-format,of M + N,unfolded real-inner-1-right]])
       proof (safe, goal-cases)
        case (2 x)
        have ⋀m. x ∈ s ⟹ ∀ n≥m. (f m x)·1 ≤ (f n x)·1
          apply (rule transitive-stepwise-le)
          using assms(3)
          apply auto
          done
        then show ?case
          by auto
       next
        case 1
        show ?case
          apply (subst integral-restrict-univ[symmetric,OF int])
          unfolding ifif integral-restrict-univ[OF int']
          apply (rule integral-subset-le[OF - int'])
          using assms
          apply auto
          done
       qed
     qed
   qed
   then show ?thesis
     apply safe
     defer
     apply (drule integral-unique)
     using i
     apply auto
     done
 qed

 have sub: ⋀k. integral s (λx. f k x − f 0 x) = integral s (f k) − integral s (f 0)
   apply (subst integral-diff)
   apply (rule assms(1)[rule-format])+
   apply rule
   done
 have ⋀x m. x ∈ s ⟹ ∀ n≥m. f m x ≤ f n x
   apply (rule transitive-stepwise-le)
   using assms(2)
   apply auto
   done
 note ∗ = this[rule-format]
 have (λx. g x − f 0 x) integrable-on s ∧ ((λk. integral s (λx. f (Suc k) x − f 0
x)) ⟶
    integral s (λx. g x − f 0 x)) sequentially
   apply (rule lem)
   apply safe
```

proof *goal-cases*
case (*1 k x*)
then show *?case*
using ∗[*of x 0 Suc k*] **by** *auto*
next
case (*2 k*)
then show *?case*
apply (*rule integrable-diff*)
using *assms*(*1*)
apply *auto*
done
next
case (*3 k x*)
then show *?case*
using ∗[*of x Suc k Suc* (*Suc k*)] **by** *auto*
next
case (*4 x*)
then show *?case*
apply −
apply (*rule tendsto-diff*)
using *LIMSEQ-ignore-initial-segment*[*OF assms*(*3*)[*rule-format*],*of x 1*]
apply *auto*
done
next
case *5*
then show *?case*
using *assms*(*4*)
unfolding *bounded-iff*
apply *safe*
apply (*rule-tac x*=*a* + *norm* (*integral s* (λ*x*. *f 0 x*)) **in** *exI*)
apply *safe*
apply (*erule-tac x*=*integral s* (λ*x*. *f* (*Suc k*) *x*) **in** *ballE*)
unfolding *sub*
apply (*rule order-trans*[*OF norm-triangle-ineq4*])
apply *auto*
done
qed
note *conjunctD2*[*OF this*]
note *tendsto-add*[*OF this*(*2*) *tendsto-const*[*of integral s* (*f 0*)]]
integrable-add[*OF this*(*1*) *assms*(*1*)[*rule-format*,*of 0*]]
then show *?thesis*
unfolding *sub*
apply −
apply *rule*
defer
apply (*subst*(*asm*) *integral-diff*)
using *assms*(*1*)
apply *auto*
apply (*rule LIMSEQ-imp-Suc*)

apply *assumption*
done
qed

lemma *has-integral-monotone-convergence-increasing*:
fixes $f :: nat \Rightarrow 'a::euclidean\text{-}space \Rightarrow real$
assumes f: $\bigwedge k.\ (f\ k\ has\text{-}integral\ x\ k)\ s$
assumes $\bigwedge k\ x.\ x \in s \Longrightarrow f\ k\ x \leq f\ (Suc\ k)\ x$
assumes $\bigwedge x.\ x \in s \Longrightarrow (\lambda k.\ f\ k\ x) \longrightarrow g\ x$
assumes $x \longrightarrow x'$
shows $(g\ has\text{-}integral\ x')\ s$
proof −
have *x-eq*: $x = (\lambda i.\ integral\ s\ (f\ i))$
by (*simp add*: *integral-unique*[*OF f*])
then have x: $\{integral\ s\ (f\ k)\ |k.\ True\} = range\ x$
by *auto*

have $g\ integrable\text{-}on\ s \wedge (\lambda k.\ integral\ s\ (f\ k)) \longrightarrow integral\ s\ g$
proof (*intro monotone-convergence-increasing allI ballI assms*)
show *bounded* $\{integral\ s\ (f\ k)\ |k.\ True\}$
unfolding x **by** (*rule convergent-imp-bounded*) *fact*
qed (*auto intro*: f)
moreover then have $integral\ s\ g = x'$
by (*intro LIMSEQ-unique*[*OF* - ‹$x \longrightarrow x'$›]) (*simp add*: *x-eq*)
ultimately show *?thesis*
by (*simp add*: *has-integral-integral*)
qed

lemma *monotone-convergence-decreasing*:
fixes $f :: nat \Rightarrow 'n::euclidean\text{-}space \Rightarrow real$
assumes $\forall k.\ (f\ k)\ integrable\text{-}on\ s$
and $\forall k.\ \forall x{\in}s.\ f\ (Suc\ k)\ x \leq f\ k\ x$
and $\forall x{\in}s.\ ((\lambda k.\ f\ k\ x) \longrightarrow g\ x)\ sequentially$
and *bounded* $\{integral\ s\ (f\ k)|\ k.\ True\}$
shows $g\ integrable\text{-}on\ s \wedge ((\lambda k.\ integral\ s\ (f\ k)) \longrightarrow integral\ s\ g)\ sequentially$
proof −
note *assm* = *assms*[*rule-format*]
have $*$: $\{integral\ s\ (\lambda x.\ -\ f\ k\ x)\ |k.\ True\} = op\ *_R\ (-\ 1)\ `\ \{integral\ s\ (f\ k)|\ k.\ True\}$
apply *safe*
unfolding *image-iff*
apply (*rule-tac x*=*integral s* (*f k*) **in** *bexI*)
prefer *3*
apply (*rule-tac x*=*k* **in** *exI*)
unfolding *integral-neg*[*OF assm*(*1*)]
apply *auto*
done
have $(\lambda x.\ -\ g\ x)\ integrable\text{-}on\ s \wedge$
$((\lambda k.\ integral\ s\ (\lambda x.\ -\ f\ k\ x)) \longrightarrow integral\ s\ (\lambda x.\ -\ g\ x))\ sequentially$

apply (*rule monotone-convergence-increasing*)
apply *safe*
apply (*rule integrable-neg*)
apply (*rule assm*)
defer
apply (*rule tendsto-minus*)
apply (*rule assm*)
apply *assumption*
unfolding ∗
apply (*rule bounded-scaling*)
using *assm*
apply *auto*
done
note ∗ = *conjunctD2*[*OF this*]
show *?thesis*
apply *rule*
using *integrable-neg*[*OF* ∗(*1*)]
defer
using *tendsto-minus*[*OF* ∗(*2*)]
unfolding *integral-neg*[*OF assm*(*1*)]
unfolding *integral-neg*[*OF* ∗(*1*),*symmetric*]
apply *auto*
done
qed

35.65 Absolute integrability (this is the same as Lebesgue integrability)

definition *absolutely-integrable-on* (**infixr** *absolutely′-integrable′-on 46*)
where *f absolutely-integrable-on s* ⟷ *f integrable-on s* ∧ (λ*x*. (*norm*(*f x*))) *integrable-on s*

lemma *absolutely-integrable-onI*[*intro?*]:
f integrable-on s ⟹
(λ*x*. (*norm*(*f x*))) *integrable-on s* ⟹ *f absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def*
by *auto*

lemma *absolutely-integrable-onD*[*dest*]:
assumes *f absolutely-integrable-on s*
shows *f integrable-on s*
and (λ*x*. *norm* (*f x*)) *integrable-on s*
using *assms*
unfolding *absolutely-integrable-on-def*
by *auto*

lemma *integral-norm-bound-integral*:

```
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes f integrable-on s
    and g integrable-on s
    and ∀x∈s. norm (f x) ≤ g x
  shows norm (integral s f) ≤ integral s g
proof −
  have ∗: ⋀x y. (∀e::real. 0 < e ⟶ x < y + e) ⟹ x ≤ y
    apply (rule ccontr)
    apply (erule-tac x=x − y in allE)
    apply auto
    done
  have norm: norm ig < dia + e
    if norm sg ≤ dsa
    and |dsa − dia| < e / 2
    and norm (sg − ig) < e / 2
    for e dsa dia and sg ig :: 'a
    apply (rule le-less-trans[OF norm-triangle-sub[of ig sg]])
    apply (subst real-sum-of-halves[of e,symmetric])
    unfolding add.assoc[symmetric]
    apply (rule add-le-less-mono)
    defer
    apply (subst norm-minus-commute)
    apply (rule that(3))
    apply (rule order-trans[OF that(1)])
    using that(2)
    apply arith
    done
  have lem: norm (integral(cbox a b) f) ≤ integral (cbox a b) g
    if f integrable-on cbox a b
    and g integrable-on cbox a b
    and ∀x∈cbox a b. norm (f x) ≤ g x
    for f :: 'n ⇒ 'a and g a b
  proof (rule ∗[rule-format])
    fix e :: real
    assume e > 0
    then have ∗: e/2 > 0
      by auto
    from integrable-integral[OF that(1),unfolded has-integral[of f],rule-format,OF
∗]
    guess d1 .. note d1 = conjunctD2[OF this,rule-format]
    from integrable-integral[OF that(2),unfolded has-integral[of g],rule-format,OF
∗]
    guess d2 .. note d2 = conjunctD2[OF this,rule-format]
    note gauge-inter[OF d1(1) d2(1)]
    from fine-division-exists[OF this, of a b] guess p . note p=this
    show norm (integral (cbox a b) f) < integral (cbox a b) g + e
      apply (rule norm)
      defer
      apply (rule d2(2)[OF conjI[OF p(1)],unfolded real-norm-def])
```

```
    defer
    apply (rule d1(2)[OF conjI[OF p(1)]])
    defer
    apply (rule setsum-norm-le)
   proof safe
    fix x k
    assume (x, k) ∈ p
    note as = tagged-division-ofD(2–4)[OF p(1) this]
    from this(3) guess u v by (elim exE) note uv=this
    show norm (content k *R f x) ≤ content k *R g x
     unfolding uv norm-scaleR
     unfolding abs-of-nonneg[OF content-pos-le] real-scaleR-def
     apply (rule mult-left-mono)
     using that(3) as
     apply auto
     done
   qed (insert p[unfolded fine-inter], auto)
  qed

  { presume ⋀e. 0 < e ⟹ norm (integral s f) < integral s g + e
   then show ?thesis by (rule *[rule-format]) auto }
  fix e :: real
  assume e > 0
  then have e: e/2 > 0
   by auto
  note assms(1)[unfolded integrable-alt[of f]] note f=this[THEN conjunct1,rule-format]
  note assms(2)[unfolded integrable-alt[of g]] note g=this[THEN conjunct1,rule-format]
  from integrable-integral[OF assms(1),unfolded has-integral'[of f],rule-format,OF
e]
  guess B1 .. note B1=conjunctD2[OF this[rule-format],rule-format]
  from integrable-integral[OF assms(2),unfolded has-integral'[of g],rule-format,OF
e]
  guess B2 .. note B2=conjunctD2[OF this[rule-format],rule-format]
  from bounded-subset-cbox[OF bounded-ball, of 0::'n max B1 B2]
  guess a b by (elim exE) note ab=this[unfolded ball-max-Un]

  have ball 0 B1 ⊆ cbox a b
   using ab by auto
  from B1(2)[OF this] guess z .. note z=conjunctD2[OF this]
  have ball 0 B2 ⊆ cbox a b
   using ab by auto
  from B2(2)[OF this] guess w .. note w=conjunctD2[OF this]

  show norm (integral s f) < integral s g + e
   apply (rule norm)
   apply (rule lem[OF f g, of a b])
   unfolding integral-unique[OF z(1)] integral-unique[OF w(1)]
   defer
   apply (rule w(2)[unfolded real-norm-def])
```

```
    apply (rule z(2))
    apply safe
    apply (case-tac x ∈ s)
    unfolding if-P
    apply (rule assms(3)[rule-format])
    apply auto
    done
qed

lemma integral-norm-bound-integral-component:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  fixes g :: 'n ⇒ 'b::euclidean-space
  assumes f integrable-on s
    and g integrable-on s
    and ∀x∈s. norm(f x) ≤ (g x)•k
  shows norm (integral s f) ≤ (integral s g)•k
proof −
  have norm (integral s f) ≤ integral s ((λx. x • k) ∘ g)
    apply (rule integral-norm-bound-integral[OF assms(1)])
    apply (rule integrable-linear[OF assms(2)])
    apply rule
    unfolding o-def
    apply (rule assms)
    done
  then show ?thesis
    unfolding o-def integral-component-eq[OF assms(2)] .
qed

lemma has-integral-norm-bound-integral-component:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  fixes g :: 'n ⇒ 'b::euclidean-space
  assumes (f has-integral i) s
    and (g has-integral j) s
    and ∀x∈s. norm (f x) ≤ (g x)•k
  shows norm i ≤ j•k
  using integral-norm-bound-integral-component[of f s g k]
  unfolding integral-unique[OF assms(1)] integral-unique[OF assms(2)]
  using assms
  by auto

lemma absolutely-integrable-le:
  fixes f :: 'n::euclidean-space ⇒ 'a::banach
  assumes f absolutely-integrable-on s
  shows norm (integral s f) ≤ integral s (λx. norm (f x))
  apply (rule integral-norm-bound-integral)
  using assms
  apply auto
  done
```

lemma *absolutely-integrable-0*[*intro*]:
(λ*x*. *0*) *absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def*
by *auto*

lemma *absolutely-integrable-cmul*[*intro*]:
f absolutely-integrable-on s ⟹
(λ*x*. *c* *$_R$ *f x*) *absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def*
using *integrable-cmul*[*of f s c*]
using *integrable-cmul*[*of* λ*x*. *norm* (*f x*) *s* |*c*|]
by *auto*

lemma *absolutely-integrable-neg*[*intro*]:
f absolutely-integrable-on s ⟹
(λ*x*. −*f*(*x*)) *absolutely-integrable-on s*
apply (*drule absolutely-integrable-cmul*[**where** *c*=−*1*])
apply *auto*
done

lemma *absolutely-integrable-norm*[*intro*]:
f absolutely-integrable-on s ⟹
(λ*x*. *norm* (*f x*)) *absolutely-integrable-on s*
unfolding *absolutely-integrable-on-def*
by *auto*

lemma *absolutely-integrable-abs*[*intro*]:
f absolutely-integrable-on s ⟹
(λ*x*. |*f x*::*real*|) *absolutely-integrable-on s*
apply (*drule absolutely-integrable-norm*)
unfolding *real-norm-def*
apply *assumption*
done

lemma *absolutely-integrable-on-subinterval*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
shows *f absolutely-integrable-on s* ⟹
cbox a b ⊆ *s* ⟹ *f absolutely-integrable-on cbox a b*
unfolding *absolutely-integrable-on-def*
by (*metis integrable-on-subcbox*)

lemma *absolutely-integrable-bounded-variation*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'a*::*banach*
assumes *f absolutely-integrable-on UNIV*
obtains *B* **where** ∀ *d*. *d division-of* (⋃ *d*) ⟶ *setsum* (λ*k*. *norm*(*integral k f*))
d ≤ *B*
apply (*rule that*[*of integral UNIV* (λ*x*. *norm* (*f x*))])
apply *safe*
proof *goal-cases*

case *prems*: (*1 d*)
note *d* = *division-ofD*[*OF prems*(*2*)]
have (∑ *k*∈*d*. *norm* (*integral k f*)) ≤ (∑ *i*∈*d*. *integral i* (λ*x*. *norm* (*f x*)))
apply (*rule setsum-mono*,*rule absolutely-integrable-le*)
apply (*drule d*(*4*))
apply *safe*
apply (*rule absolutely-integrable-on-subinterval*[*OF assms*])
apply *auto*
done
also have ... ≤ *integral* (⋃ *d*) (λ*x*. *norm* (*f x*))
apply (*subst integral-combine-division-topdown*[*OF - prems*(*2*)])
using *integrable-on-subdivision*[*OF prems*(*2*)]
using *assms*
apply *auto*
done
also have ... ≤ *integral UNIV* (λ*x*. *norm* (*f x*))
apply (*rule integral-subset-le*)
using *integrable-on-subdivision*[*OF prems*(*2*)]
using *assms*
apply *auto*
done
finally show *?case* **.**
qed

lemma *helplemma*:
assumes *setsum* (λ*x*. *norm* (*f x* − *g x*)) *s* < *e*
and *finite s*
shows |*setsum* (λ*x*. *norm*(*f x*)) *s* − *setsum* (λ*x*. *norm*(*g x*)) *s*| < *e*
unfolding *setsum-subtractf*[*symmetric*]
apply (*rule le-less-trans*[*OF setsum-abs*])
apply (*rule le-less-trans*[*OF - assms*(*1*)])
apply (*rule setsum-mono*)
apply (*rule norm-triangle-ineq3*)
done

lemma *bounded-variation-absolutely-integrable-interval*:
fixes *f* :: *'n*::*euclidean-space* ⇒ *'m*::*euclidean-space*
assumes *f*: *f integrable-on cbox a b*
and ∗: ∀ *d*. *d division-of* (*cbox a b*) ⟶ *setsum* (λ*k*. *norm*(*integral k f*)) *d* ≤ *B*
shows *f absolutely-integrable-on cbox a b*
proof −
let *?f* = λ*d*. ∑ *k*∈*d*. *norm* (*integral k f*) **and** *?D* = {*d*. *d division-of* (*cbox a b*)}
have *D-1*: *?D* ≠ {}
by (*rule elementary-interval*[*of a b*]) *auto*
have *D-2*: *bdd-above* (*?f*‘*?D*)
by (*metis* ∗ *mem-Collect-eq bdd-aboveI2*)
note *D* = *D-1 D-2*
let *?S* = *SUP x*:*?D*. *?f x*

show *?thesis*
apply (*rule absolutely-integrable-onI* [*OF f has-integral-integrable*])
apply (*subst has-integral*[*of - ?S*])
apply *safe*
proof *goal-cases*
case *e*: (*1 e*)
then have *?S − e / 2 < ?S* **by** *simp*
then obtain *d* **where** *d*: *d division-of* (*cbox a b*) *?S − e / 2 <* (∑ *k*∈*d*. *norm* (*integral k f*))
unfolding *less-cSUP-iff*[*OF D*] **by** *auto*
note *d′ = division-ofD*[*OF this*(*1*)]

have ∀ *x*. ∃ *e*>*0*. ∀ *i*∈*d*. *x* ∉ *i* ⟶ *ball x e* ∩ *i* = {}
proof
fix *x*
have ∃ *da*>*0*. ∀ *xa*∈⋃ {*i* ∈ *d*. *x* ∉ *i*}. *da* ≤ *dist x xa*
apply (*rule separate-point-closed*)
apply (*rule closed-Union*)
apply (*rule finite-subset*[*OF - d′*(*1*)])
using *d′*(*4*)
apply *auto*
done
then show ∃ *e*>*0*. ∀ *i*∈*d*. *x* ∉ *i* ⟶ *ball x e* ∩ *i* = {}
by *force*
qed
from *choice*[*OF this*] **guess** *k* **.. note** *k*=*conjunctD2*[*OF this*[*rule-format*],*rule-format*]

have *e*/*2* > *0*
using *e* **by** *auto*
from *henstock-lemma*[*OF assms*(*1*) *this*] **guess** *g* **. note** *g*=*this*[*rule-format*]
let *?g* = λ*x*. *g x* ∩ *ball x* (*k x*)
show *?case*
apply (*rule-tac x*=*?g* **in** *exI*)
apply *safe*
proof −
show *gauge ?g*
using *g*(*1*) *k*(*1*)
unfolding *gauge-def*
by *auto*
fix *p*
assume *p tagged-division-of* (*cbox a b*) **and** *?g fine p*
note *p* = *this*(*1*) *conjunctD2*[*OF this*(*2*)[*unfolded fine-inter*]]
note *p′* = *tagged-division-ofD*[*OF p*(*1*)]
def *p′* ≡ {(*x*,*k*) | *x k*. ∃ *i l*. *x* ∈ *i* ∧ *i* ∈ *d* ∧ (*x*,*l*) ∈ *p* ∧ *k* = *i* ∩ *l*}
have *gp′*: *g fine p′*
using *p*(*2*)
unfolding *p′-def fine-def*
by *auto*
have *p″*: *p′ tagged-division-of* (*cbox a b*)

```
  apply (rule tagged-division-ofI)
proof −
  show finite p'
    apply (rule finite-subset[of - (λ(k,(x,l)). (x,k ∩ l)) `
      {(k,xl) | k xl. k ∈ d ∧ xl ∈ p}])
    unfolding p'-def
    defer
    apply (rule finite-imageI,rule finite-product-dependent[OF d'(1) p'(1)])
    apply safe
    unfolding image-iff
    apply (rule-tac x=(i,x,l) in bexI)
    apply auto
    done
  fix x k
  assume (x, k) ∈ p'
  then have ∃ i l. x ∈ i ∧ i ∈ d ∧ (x, l) ∈ p ∧ k = i ∩ l
    unfolding p'-def by auto
  then guess i l by (elim exE) note il=conjunctD4[OF this]
  show x ∈ k and k ⊆ cbox a b
    using p'(2−3)[OF il(3)] il by auto
  show ∃ a b. k = cbox a b
    unfolding il using p'(4)[OF il(3)] d'(4)[OF il(2)]
    apply safe
    unfolding inter-interval
    apply auto
    done
next
  fix x1 k1
  assume (x1, k1) ∈ p'
  then have ∃ i l. x1 ∈ i ∧ i ∈ d ∧ (x1, l) ∈ p ∧ k1 = i ∩ l
    unfolding p'-def by auto
  then guess i1 l1 by (elim exE) note il1=conjunctD4[OF this]
  fix x2 k2
  assume (x2,k2)∈p'
  then have ∃ i l. x2 ∈ i ∧ i ∈ d ∧ (x2, l) ∈ p ∧ k2 = i ∩ l
    unfolding p'-def by auto
  then guess i2 l2 by (elim exE) note il2=conjunctD4[OF this]
  assume (x1, k1) ≠ (x2, k2)
  then have interior i1 ∩ interior i2 = {} ∨ interior l1 ∩ interior l2 = {}
    using d'(5)[OF il1(2) il2(2)] p'(5)[OF il1(3) il2(3)]
    unfolding il1 il2
    by auto
  then show interior k1 ∩ interior k2 = {}
    unfolding il1 il2 by auto
next
  have ∗: ∀(x, X) ∈ p'. X ⊆ cbox a b
    unfolding p'-def using d' by auto
  show ⋃{k. ∃ x. (x, k) ∈ p'} = cbox a b
    apply rule
```

```
      apply (rule Union-least)
      unfolding mem-Collect-eq
      apply (erule exE)
      apply (drule *[rule-format])
      apply safe
     proof −
      fix y
      assume y: y ∈ cbox a b
      then have ∃ x l. (x, l) ∈ p ∧ y∈l
       unfolding p'(6)[symmetric] by auto
      then guess x l by (elim exE) note xl=conjunctD2[OF this]
      then have ∃ k. k ∈ d ∧ y ∈ k
       using y unfolding d'(6)[symmetric] by auto
      then guess i .. note i = conjunctD2[OF this]
      have x ∈ i
       using fineD[OF p(3) xl(1)]
       using k(2)[OF i(1), of x]
       using i(2) xl(2)
       by auto
      then show y ∈ ⋃{k. ∃ x. (x, k) ∈ p'}
       unfolding p'-def Union-iff
       apply (rule-tac x=i ∩ l in bexI)
       using i xl
       apply auto
       done
     qed
    qed

    then have (∑ (x, k)∈p'. norm (content k *R f x − integral k f)) < e / 2
     apply −
     apply (rule g(2)[rule-format])
     unfolding tagged-division-of-def
     apply safe
     apply (rule gp')
     done
    then have **: |(∑ (x,k)∈p'. norm (content k *R f x)) − (∑ (x,k)∈p'. norm
(integral k f))| < e / 2
     unfolding split-def
     using p''
     by (force intro!: helplemma)

    have p'alt: p' = {(x,(i ∩ l)) | x i l. (x,l) ∈ p ∧ i ∈ d ∧ i ∩ l ≠ {}}
    proof (safe, goal-cases)
     case prems: (2 - - x i l)
     have x ∈ i
      using fineD[OF p(3) prems(1)] k(2)[OF prems(2), of x] prems(4−)
      by auto
     then have (x, i ∩ l) ∈ p'
      unfolding p'-def
```

using *prems*
apply *safe*
apply (*rule-tac x=x* **in** *exI*)
apply (*rule-tac x=i ∩ l* **in** *exI*)
apply *safe*
using *prems*
apply *auto*
done
then show *?case*
using *prems(3)* **by** *auto*
next
fix *x k*
assume $(x, k) \in p'$
then have $\exists i\ l.\ x \in i \wedge i \in d \wedge (x, l) \in p \wedge k = i \cap l$
unfolding *p'-def* **by** *auto*
then guess *i l* **by** (*elim exE*) **note** *il=conjunctD4*[*OF this*]
then show $\exists y\ i\ l.\ (x, k) = (y, i \cap l) \wedge (y, l) \in p \wedge i \in d \wedge i \cap l \neq \{\}$
apply (*rule-tac x=x* **in** *exI*)
apply (*rule-tac x=i* **in** *exI*)
apply (*rule-tac x=l* **in** *exI*)
using *p'(2)*[*OF il(3)*]
apply *auto*
done
qed
have *sum-p'*: $(\sum (x, k) \in p'.\ norm\ (integral\ k\ f)) = (\sum k \in snd\ `\ p'.\ norm\ (integral\ k\ f))$
apply (*subst setsum-over-tagged-division-lemma*[*OF p'',of* $\lambda k.\ norm\ (integral\ k\ f)$])
unfolding *norm-eq-zero*
apply (*rule integral-null*)
apply *assumption*
apply *rule*
done
note *snd-p = division-ofD*[*OF division-of-tagged-division*[*OF p(1)*]]

have $*$: $\bigwedge sni\ sni'\ sf\ sf'.\ |sf' - sni'| < e\ /\ 2 \longrightarrow\ ?S - e\ /\ 2 < sni \wedge sni' \leq\ ?S \wedge$
$sni \leq sni' \wedge sf' = sf \longrightarrow |sf - ?S| < e$
by *arith*
show $norm\ ((\sum (x, k) \in p.\ content\ k *_R norm\ (f\ x)) - ?S) < e$
unfolding *real-norm-def*
apply (*rule* $*$[*rule-format*,*OF* $**$])
apply *safe*
apply(*rule d(2)*)
proof *goal-cases*
case *1*
show *?case*
by (*auto simp*: *sum-p' division-of-tagged-division*[*OF p''*] *D intro*!: *cSUP-upper*)

```
    next
      case 2
      have *: {k ∩ l | k l. k ∈ d ∧ l ∈ snd ‘ p} =
        (λ(k,l). k ∩ l) ‘ {(k,l)|k l. k ∈ d ∧ l ∈ snd ‘ p}
        by auto
      have (∑k∈d. norm (integral k f)) ≤ (∑i∈d. ∑l∈snd ‘ p. norm (integral
(i ∩ l) f))
      proof (rule setsum-mono, goal-cases)
        case k: (1 k)
        from d'(4)[OF this] guess u v by (elim exE) note uv=this
        def d' ≡ {cbox u v ∩ l |l. l ∈ snd ‘ p ∧  cbox u v ∩ l ≠ {}}
        note uvab = d'(2)[OF k[unfolded uv]]
        have d' division-of cbox u v
          apply (subst d'-def)
          apply (rule division-inter-1)
          apply (rule division-of-tagged-division[OF p(1)])
          apply (rule uvab)
          done
        then have norm (integral k f) ≤ setsum (λk. norm (integral k f)) d'
          unfolding uv
          apply (subst integral-combine-division-topdown[of - - d'])
          apply (rule integrable-on-subcbox[OF assms(1) uvab])
          apply assumption
          apply (rule setsum-norm-le)
          apply auto
          done
        also have ... = (∑k∈{k ∩ l |l. l ∈ snd ‘ p}. norm (integral k f))
          apply (rule setsum.mono-neutral-left)
          apply (subst simple-image)
          apply (rule finite-imageI)+
          apply fact
          unfolding d'-def uv
          apply blast
        proof (rule, goal-cases)
          case prems: (1 i)
          then have i ∈ {cbox u v ∩ l |l. l ∈ snd ‘ p}
            by auto
          from this[unfolded mem-Collect-eq] guess l .. note l=this
          then have cbox u v ∩ l = {}
            using prems by auto
          then show ?case
            using l by auto
        qed
        also have ... = (∑l∈snd ‘ p. norm (integral (k ∩ l) f))
          unfolding simple-image
          apply (rule setsum.reindex-nontrivial [unfolded o-def])
          apply (rule finite-imageI)
          apply (rule p')
        proof goal-cases
```

```
      case prems: (1 l y)
      have interior (k ∩ l) ⊆ interior (l ∩ y)
        apply (subst(2) interior-Int)
        apply (rule Int-greatest)
        defer
        apply (subst prems(4))
        apply auto
        done
      then have ∗: interior (k ∩ l) = {}
        using snd-p(5)[OF prems(1−3)] by auto
       from d'(4)[OF k] snd-p(4)[OF prems(1)] guess u1 v1 u2 v2 by (elim
exE) note uv=this
      show ?case
        using ∗
        unfolding uv inter-interval content-eq-0-interior[symmetric]
        by auto
    qed
    finally show ?case .
  qed
  also have ... = (∑(i,l)∈{(i, l) |i l. i ∈ d ∧ l ∈ snd ' p}. norm (integral
(i∩l) f))
    apply (subst sum-sum-product[symmetric])
    apply fact
    using p'(1)
    apply auto
    done
  also have ... = (∑x∈{(i, l) |i l. i ∈ d ∧ l ∈ snd ' p}. norm (integral
(case-prod op ∩ x) f))
    unfolding split-def ..
  also have ... = (∑k∈{i ∩ l |i l. i ∈ d ∧ l ∈ snd ' p}. norm (integral k f))
    unfolding ∗
    apply (rule setsum.reindex-nontrivial [symmetric, unfolded o-def])
    apply (rule finite-product-dependent)
    apply fact
    apply (rule finite-imageI)
    apply (rule p')
    unfolding split-paired-all mem-Collect-eq split-conv o-def
  proof −
    note ∗ = division-ofD(4,5)[OF division-of-tagged-division,OF p(1)]
    fix l1 l2 k1 k2
    assume as:
      (l1, k1) ≠ (l2, k2)
      l1 ∩ k1 = l2 ∩ k2
      ∃i l. (l1, k1) = (i, l) ∧ i ∈ d ∧ l ∈ snd ' p
      ∃i l. (l2, k2) = (i, l) ∧ i ∈ d ∧ l ∈ snd ' p
    then have l1 ∈ d and k1 ∈ snd ' p
      by auto from d'(4)[OF this(1)] ∗(1)[OF this(2)]
    guess u1 v1 u2 v2 by (elim exE) note uv=this
    have l1 ≠ l2 ∨ k1 ≠ k2
```

```
    using as by auto
  then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2 = {}
    apply −
    apply (erule disjE)
    apply (rule disjI2)
    apply (rule d'(5))
    prefer 4
    apply (rule disjI1)
    apply (rule ∗)
    using as
    apply auto
    done
  moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
    using as(2) by auto
  ultimately have interior(l1 ∩ k1) = {}
    by auto
  then show norm (integral (l1 ∩ k1) f) = 0
    unfolding uv inter-interval
    unfolding content-eq-0-interior[symmetric]
    by auto
qed
also have ... = (∑(x, k)∈p'. norm (integral k f))
  unfolding sum-p'
  apply (rule setsum.mono-neutral-right)
  apply (subst ∗)
  apply (rule finite-imageI[OF finite-product-dependent])
  apply fact
  apply (rule finite-imageI[OF p'(1)])
  apply safe
proof goal-cases
  case (2 i ia l a b)
  then have ia ∩ b = {}
    unfolding p'alt image-iff Bex-def not-ex
    apply (erule-tac x=(a, ia ∩ b) in allE)
    apply auto
    done
  then show ?case
    by auto
next
  case (1 x a b)
  then show ?case
    unfolding p'-def
    apply safe
    apply (rule-tac x=i in exI)
    apply (rule-tac x=l in exI)
    unfolding snd-conv image-iff
    apply safe
    apply (rule-tac x=(a,l) in bexI)
    apply auto
```

```
        done
      qed
      finally show ?case .
    next
      case 3
      let ?S = {(x, i ∩ l) |x i l. (x, l) ∈ p ∧ i ∈ d}
      have Sigma-alt: ⋀s t. s × t = {(i, j) |i j. i ∈ s ∧ j ∈ t}
        by auto
      have ∗: ?S = (λ(xl,i). (fst xl, snd xl ∩ i)) ‘ (p × d)
        apply safe
        unfolding image-iff
        apply (rule-tac x=((x,l),i) in bexI)
        apply auto
        done
      note pdfin = finite-cartesian-product[OF p'(1) d'(1)]
      have (∑(x, k)∈p'. norm (content k ∗R f x)) = (∑(x, k)∈?S. |content k|
∗ norm (f x))
        unfolding norm-scaleR
        apply (rule setsum.mono-neutral-left)
        apply (subst ∗)
        apply (rule finite-imageI)
        apply fact
        unfolding p'alt
        apply blast
        apply safe
        apply (rule-tac x=x in exI)
        apply (rule-tac x=i in exI)
        apply (rule-tac x=l in exI)
        apply auto
        done
      also have ... = (∑((x,l),i)∈p × d. |content (l ∩ i)| ∗ norm (f x))
        unfolding ∗
        apply (subst setsum.reindex-nontrivial)
        apply fact
        unfolding split-paired-all
        unfolding o-def split-def snd-conv fst-conv mem-Sigma-iff prod.inject
        apply (elim conjE)
      proof −
        fix x1 l1 k1 x2 l2 k2
        assume as: (x1, l1) ∈ p (x2, l2) ∈ p k1 ∈ d k2 ∈ d
          x1 = x2 l1 ∩ k1 = l2 ∩ k2 ¬ ((x1 = x2 ∧ l1 = l2) ∧ k1 = k2)
        from d'(4)[OF as(3)] p'(4)[OF as(1)] guess u1 v1 u2 v2 by (elim exE)
note uv=this
        from as have l1 ≠ l2 ∨ k1 ≠ k2
          by auto
       then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2 = {}
          apply −
          apply (erule disjE)
          apply (rule disjI2)
```

```
    defer
    apply (rule disjI1)
    apply (rule d'(5)[OF as(3−4)])
    apply assumption
    apply (rule p'(5)[OF as(1−2)])
    apply auto
    done
  moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
    unfolding as ..
  ultimately have interior (l1 ∩ k1) = {}
    by auto
  then show |content (l1 ∩ k1)| * norm (f x1) = 0
    unfolding uv inter-interval
    unfolding content-eq-0-interior[symmetric]
    by auto
qed safe
also have ... = (∑(x, k)∈p. content k *R norm (f x))
  unfolding Sigma-alt
  apply (subst sum-sum-product[symmetric])
  apply (rule p')
  apply rule
  apply (rule d')
  apply (rule setsum.cong)
  apply (rule refl)
  unfolding split-paired-all split-conv
proof −
  fix x l
  assume as: (x, l) ∈ p
  note xl = p'(2−4)[OF this]
  from this(3) guess u v by (elim exE) note uv=this
  have (∑i∈d. |content (l ∩ i)|) = (∑k∈d. content (k ∩ cbox u v))
    apply (rule setsum.cong)
    apply (rule refl)
    apply (drule d'(4))
    apply safe
    apply (subst Int-commute)
    unfolding inter-interval uv
    apply (subst abs-of-nonneg)
    apply auto
    done
  also have ... = setsum content {k ∩ cbox u v| k. k ∈ d}
    unfolding simple-image
    apply (rule setsum.reindex-nontrivial [unfolded o-def, symmetric])
    apply (rule d')
  proof goal-cases
    case prems: (1 k y)
    from d'(4)[OF this(1)] d'(4)[OF this(2)]
    guess u1 v1 u2 v2 by (elim exE) note uv=this
    have {} = interior ((k ∩ y) ∩ cbox u v)
```

```
            apply (subst interior-Int)
            using d'(5)[OF prems(1–3)]
            apply auto
            done
          also have ... = interior (y ∩ (k ∩ cbox u v))
            by auto
          also have ... = interior (k ∩ cbox u v)
            unfolding prems(4) by auto
          finally show ?case
            unfolding uv inter-interval content-eq-0-interior ..
        qed
        also have ... = setsum content {cbox u v ∩ k |k. k ∈ d ∧ cbox u v ∩ k
≠ {}}
          apply (rule setsum.mono-neutral-right)
          unfolding simple-image
          apply (rule finite-imageI)
          apply (rule d')
          apply blast
          apply safe
          apply (rule-tac x=k in exI)
        proof goal-cases
          case prems: (1 i k)
          from d'(4)[OF this(1)] guess a b by (elim exE) note ab=this
          have interior (k ∩ cbox u v) ≠ {}
            using prems(2)
            unfolding ab inter-interval content-eq-0-interior
            by auto
          then show ?case
            using prems(1)
            using interior-subset[of k ∩ cbox u v]
            by auto
        qed
        finally show (∑ i∈d. |content (l ∩ i)| * norm (f x)) = content l *R norm
(f x)
          unfolding setsum-left-distrib[symmetric] real-scaleR-def
            apply (subst(asm) additive-content-division[OF division-inter-1[OF
d(1)]])
          using xl(2)[unfolded uv]
          unfolding uv
          apply auto
          done
      qed
      finally show ?case .
    qed
  qed
 qed
qed

lemma bounded-variation-absolutely-integrable:
```

fixes f :: $'n$::*euclidean-space* $\Rightarrow$ $'m$::*euclidean-space*
assumes f *integrable-on UNIV*
and $\forall\, d.\ d$ *division-of* $(\bigcup d) \longrightarrow$ *setsum* $(\lambda k.\ norm\ (integral\ k\ f))\ d \leq B$
shows f *absolutely-integrable-on UNIV*
proof (*rule absolutely-integrable-onI*, *fact*, *rule*)
let *?f* $= \lambda d.\ \sum k{\in}d.\ norm\ (integral\ k\ f)$ **and** *?D* $= \{d.\ d$ *division-of* $(\bigcup d)\}$
have *D-1*: *?D* $\neq \{\}$
by (*rule elementary-interval*) *auto*
have *D-2*: *bdd-above* (*?f*`*?D*)
by (*intro bdd-aboveI2*[**where** $M{=}B$] *assms*(*2*)[*rule-format*]) *simp*
note *D = D-1 D-2*
let *?S = SUP d*:*?D*. *?f d*
have *f-int*: $\bigwedge a\ b.$ f *absolutely-integrable-on cbox a b*
apply (*rule bounded-variation-absolutely-integrable-interval*[**where** $B{=}B$])
apply (*rule integrable-on-subcbox*[*OF assms*(*1*)])
defer
apply *safe*
apply (*rule assms*(*2*)[*rule-format*])
apply *auto*
done
show $((\lambda x.\ norm\ (f\ x))$ *has-integral ?S*) *UNIV*
apply (*subst has-integral-alt'*)
apply *safe*
proof *goal-cases*
case (*1 a b*)
show *?case*
using *f-int*[*of a b*] **by** *auto*
next
case *prems*: (*2 e*)
have $\exists\, y{\in}$*setsum* $(\lambda k.\ norm\ (integral\ k\ f))$ ` $\{d.\ d$ *division-of* $\bigcup d\}.\ \neg\ y \leq$ *?S* $- e$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then have *?S* $\leq$ *?S* $- e$
by (*intro cSUP-least*[*OF D*(*1*)]) *auto*
then show *False*
using *prems* **by** *auto*
qed
then obtain K **where** $*$: $\exists\, x{\in}\{d.\ d$ *division-of* $\bigcup d\}.\ K = (\sum k{\in}x.\ norm\ (integral\ k\ f))$
SUPREMUM $\{d.\ d$ *division-of* $\bigcup d\}$ $(setsum\ (\lambda k.\ norm\ (integral\ k\ f))) - e < K$
by (*auto simp add*: *image-iff not-le*)
from *this*(*1*) **obtain** d **where** d *division-of* $\bigcup d$
and $K = (\sum k{\in}d.\ norm\ (integral\ k\ f))$
by *auto*
note $d = this(1)\ *(2)$[*unfolded this*(*2*)]
note d'=*division-ofD*[*OF this*(*1*)]
have *bounded* $(\bigcup d)$

```
    by (rule elementary-bounded,fact)
  from this[unfolded bounded-pos] obtain K where
    K: 0 < K ∀x∈⋃ d. norm x ≤ K by auto
  show ?case
    apply (rule-tac x=K + 1 in exI)
    apply safe
  proof −
    fix a b :: 'n
    assume ab: ball 0 (K + 1) ⊆ cbox a b
    have ∗: ∀s s1. ?S − e < s1 ∧ s1 ≤ s ∧ s < ?S + e ⟶ |s − ?S| < e
      by arith
    show norm (integral (cbox a b) (λx. if x ∈ UNIV then norm (f x) else 0) −
?S) < e
      unfolding real-norm-def
      apply (rule ∗[rule-format])
      apply safe
      apply (rule d(2))
    proof goal-cases
      case 1
      have (∑ k∈d. norm (integral k f)) ≤ setsum (λk. integral k (λx. norm (f
x))) d
        apply (rule setsum-mono)
        apply (rule absolutely-integrable-le)
        apply (drule d'(4))
        apply safe
        apply (rule f-int)
        done
      also have ... = integral (⋃ d) (λx. norm (f x))
        apply (rule integral-combine-division-bottomup[symmetric])
        apply (rule d)
        unfolding forall-in-division[OF d(1)]
        using f-int
        apply auto
        done
      also have ... ≤ integral (cbox a b) (λx. if x ∈ UNIV then norm (f x) else
0)
      proof −
        have ⋃ d ⊆ cbox a b
          apply rule
          apply (drule K(2)[rule-format])
          apply (rule ab[unfolded subset-eq,rule-format])
          apply (auto simp add: dist-norm)
          done
        then show ?thesis
          apply −
          apply (subst if-P)
          apply rule
          apply (rule integral-subset-le)
          defer
```

apply (*rule integrable-on-subdivision*[*of - - - cbox a b*])
apply (*rule d*)
using *f-int*[*of a b*]
apply *auto*
done
qed
finally show *?case* **.**
next
note *f = absolutely-integrable-onD*[*OF f-int*[*of a b*]]
note ∗ = *this*(*2*)[*unfolded has-integral-integral has-integral*[*of* λ*x. norm* (*f x*)],*rule-format*]
have *e/2>0*
using ‹*e > 0*› **by** *auto*
from ∗ [*OF this*] **obtain** *d1* **where**
d1: *gauge d1* ∀ *p. p tagged-division-of* (*cbox a b*) ∧ *d1 fine p* ⟶
norm ((∑ (*x*, *k*)∈*p. content k* ∗$_R$ *norm* (*f x*)) − *integral* (*cbox a b*) (λ*x. norm* (*f x*))) < *e* / *2*
by *auto*
from *henstock-lemma* [*OF f*(*1*) ‹*e/2>0*›] **obtain** *d2* **where**
d2: *gauge d2* ∀ *p. p tagged-partial-division-of* (*cbox a b*) ∧ *d2 fine p* ⟶
(∑ (*x*, *k*)∈*p. norm* (*content k* ∗$_R$ *f x* − *integral k f*)) < *e* / *2* **.**
obtain *p* **where**
p: *p tagged-division-of* (*cbox a b*) *d1 fine p d2 fine p*
by (*rule fine-division-exists* [*OF gauge-inter* [*OF d1*(*1*) *d2*(*1*)], *of a b*])
(*auto simp add*: *fine-inter*)
have ∗: ⋀*sf sf' si di. sf' = sf* ⟶ *si* ≤ *?S* ⟶ |*sf* − *si*| < *e* / *2* ⟶
|*sf'* − *di*| < *e* / *2* ⟶ *di* < *?S* + *e*
by *arith*
show *integral* (*cbox a b*) (λ*x. if x* ∈ *UNIV then norm* (*f x*) *else 0*) < *?S* + *e*
apply (*subst if-P*)
apply *rule*
proof (*rule* ∗[*rule-format*])
show |(∑ (*x*,*k*)∈*p. norm* (*content k* ∗$_R$ *f x*)) − (∑ (*x*,*k*)∈*p. norm* (*integral k f*))| < *e* / *2*
unfolding *split-def*
apply (*rule helplemma*)
using *d2*(*2*)[*rule-format*,*of p*]
using *p*(*1*,*3*)
unfolding *tagged-division-of-def split-def*
apply *auto*
done
show |(∑ (*x*, *k*)∈*p. content k* ∗$_R$ *norm* (*f x*)) − *integral* (*cbox a b*) (λ*x. norm*(*f x*))| < *e* / *2*
using *d1*(*2*)[*rule-format*,*OF conjI*[*OF p*(*1*,*2*)]]
by (*simp only*: *real-norm-def*)
show (∑ (*x*, *k*)∈*p. content k* ∗$_R$ *norm* (*f x*)) = (∑ (*x*, *k*)∈*p. norm* (*content k* ∗$_R$ *f x*))
apply (*rule setsum.cong*)

```
          apply (rule refl)
          unfolding split-paired-all split-conv
          apply (drule tagged-division-ofD(4)[OF p(1)])
          unfolding norm-scaleR
          apply (subst abs-of-nonneg)
          apply auto
          done
        show (∑(x, k)∈p. norm (integral k f)) ≤ ?S
          using partial-division-of-tagged-division[of p cbox a b] p(1)
          apply (subst setsum-over-tagged-division-lemma[OF p(1)])
          apply (simp add: integral-null)
          apply (intro cSUP-upper2[OF D(2), of snd ` p])
          apply (auto simp: tagged-partial-division-of-def)
          done
      qed
    qed
  qed (insert K, auto)
 qed
qed

lemma absolutely-integrable-restrict-univ:
 (λx. if x ∈ s then f x else (0::'a::banach)) absolutely-integrable-on UNIV ⟷
   f absolutely-integrable-on s
 unfolding absolutely-integrable-on-def if-distrib norm-zero integrable-restrict-univ
..

lemma absolutely-integrable-add[intro]:
 fixes f g :: 'n::euclidean-space ⇒ 'm::euclidean-space
 assumes f absolutely-integrable-on s
   and g absolutely-integrable-on s
 shows (λx. f x + g x) absolutely-integrable-on s
proof −
 let ?P = ⋀f g::'n ⇒ 'm. f absolutely-integrable-on UNIV ⟹
   g absolutely-integrable-on UNIV ⟹ (λx. f x + g x) absolutely-integrable-on
UNIV
 {
   presume as: PROP ?P
   note a = absolutely-integrable-restrict-univ[symmetric]
   have ∗: ⋀x. (if x ∈ s then f x else 0) + (if x ∈ s then g x else 0) =
     (if x ∈ s then f x + g x else 0) by auto
   show ?thesis
     apply (subst a)
     using as[OF assms[unfolded a[of f] a[of g]]]
     apply (simp only: ∗)
     done
 }
 fix f g :: 'n ⇒ 'm
 assume assms: f absolutely-integrable-on UNIV g absolutely-integrable-on UNIV
 note absolutely-integrable-bounded-variation
```

from *this*[*OF assms*(*1*)] *this*[*OF assms*(*2*)] **guess** *B1 B2* **. note** *B*=*this*[*rule-format*]
show (λ*x*. *f x* + *g x*) *absolutely-integrable-on UNIV*
apply (*rule bounded-variation-absolutely-integrable*[*of* - *B1*+*B2*])
apply (*rule integrable-add*)
prefer *3*
apply *safe*
proof *goal-cases*
case *prems*: (*1 d*)
have ⋀*k*. *k* ∈ *d* ⟹ *f integrable-on k* ∧ *g integrable-on k*
apply (*drule division-ofD*(*4*)[*OF prems*])
apply *safe*
apply (*rule-tac*[!] *integrable-on-subcbox*[*of* - *UNIV*])
using *assms*
apply *auto*
done
then have (∑ *k*∈*d*. *norm* (*integral k* (λ*x*. *f x* + *g x*))) ≤
(∑ *k*∈*d*. *norm* (*integral k f*)) + (∑ *k*∈*d*. *norm* (*integral k g*))
apply −
unfolding *setsum.distrib* [*symmetric*]
apply (*rule setsum-mono*)
apply (*subst integral-add*)
prefer *3*
apply (*rule norm-triangle-ineq*)
apply *auto*
done
also have ... ≤ *B1* + *B2*
using *B*(*1*)[*OF prems*] *B*(*2*)[*OF prems*] **by** *auto*
finally show *?case* **.**
qed (*insert assms*, *auto*)
qed

lemma *absolutely-integrable-sub*[*intro*]:
fixes *f g* :: *'n*::*euclidean-space* ⇒ *'m*::*euclidean-space*
assumes *f absolutely-integrable-on s*
and *g absolutely-integrable-on s*
shows (λ*x*. *f x* − *g x*) *absolutely-integrable-on s*
using *absolutely-integrable-add*[*OF assms*(*1*) *absolutely-integrable-neg*[*OF assms*(*2*)]]
by (*simp add*: *algebra-simps*)

lemma *absolutely-integrable-linear*:
fixes *f* :: *'m*::*euclidean-space* ⇒ *'n*::*euclidean-space*
and *h* :: *'n*::*euclidean-space* ⇒ *'p*::*euclidean-space*
assumes *f absolutely-integrable-on s*
and *bounded-linear h*
shows (*h* ∘ *f*) *absolutely-integrable-on s*
proof −
{
presume *as*: ⋀*f*::*'m* ⇒ *'n*. ⋀*h*::*'n* ⇒ *'p*. *f absolutely-integrable-on UNIV* ⟹
bounded-linear h ⟹ (*h* ∘ *f*) *absolutely-integrable-on UNIV*

```
    note a = absolutely-integrable-restrict-univ[symmetric]
    show ?thesis
      apply (subst a)
      using as[OF assms[unfolded a[of f] a[of g]]]
      apply (simp only: o-def if-distrib linear-simps[OF assms(2)])
      done
  }
  fix f :: 'm ⇒ 'n
  fix h :: 'n ⇒ 'p
  assume assms: f absolutely-integrable-on UNIV bounded-linear h
 from absolutely-integrable-bounded-variation[OF assms(1)] guess B . note B=this
 from bounded-linear.pos-bounded[OF assms(2)] guess b .. note b=conjunctD2[OF
this]
  show (h ∘ f) absolutely-integrable-on UNIV
    apply (rule bounded-variation-absolutely-integrable[of - B * b])
    apply (rule integrable-linear[OF - assms(2)])
    apply safe
  proof goal-cases
    case prems: (2 d)
    have (∑ k∈d. norm (integral k (h ∘ f))) ≤ setsum (λk. norm(integral k f)) d
* b
      unfolding setsum-left-distrib
      apply (rule setsum-mono)
    proof goal-cases
      case (1 k)
      from division-ofD(4)[OF prems this]
      guess u v by (elim exE) note uv=this
      have *: f integrable-on k
        unfolding uv
        apply (rule integrable-on-subcbox[of - UNIV])
        using assms
        apply auto
        done
      note this[unfolded has-integral-integral]
      note has-integral-linear[OF this assms(2)] integrable-linear[OF * assms(2)]
     note * = has-integral-unique[OF this(2)[unfolded has-integral-integral] this(1)]
      show ?case
        unfolding * using b by auto
    qed
    also have ... ≤ B * b
      apply (rule mult-right-mono)
      using B prems b
      apply auto
      done
    finally show ?case .
  qed (insert assms, auto)
qed

lemma absolutely-integrable-setsum:
```

fixes $f :: 'a \Rightarrow 'n::euclidean\text{-}space \Rightarrow 'm::euclidean\text{-}space$
assumes *finite t*
and $\bigwedge a.\ a \in t \Longrightarrow (f\ a)$ *absolutely-integrable-on s*
shows $(\lambda x.\ setsum\ (\lambda a.\ f\ a\ x)\ t)$ *absolutely-integrable-on s*
using *assms*(*1*,*2*)
by *induct auto*

lemma *absolutely-integrable-vector-abs*:
fixes $f :: 'a::euclidean\text{-}space => 'b::euclidean\text{-}space$
and $T :: 'c::euclidean\text{-}space \Rightarrow 'b$
assumes *f*: *f absolutely-integrable-on s*
shows $(\lambda x.\ (\sum i{\in}Basis.\ |f\ x{\cdot}T\ i|\ *_R\ i))$ *absolutely-integrable-on s*
(**is** *?Tf absolutely-integrable-on s*)
proof −
have *if-distrib*: $\bigwedge P\ A\ B\ x.\ (if\ P\ then\ A\ else\ B)\ *_R\ x = (if\ P\ then\ A\ *_R\ x\ else\ B\ *_R\ x)$
by *simp*
have $*$: $\bigwedge x.\ ?Tf\ x = (\sum i{\in}Basis.$
$((\lambda y.\ (\sum j{\in}Basis.\ (if\ j = i\ then\ y\ else\ 0)\ *_R\ j))\ o$
$(\lambda x.\ (norm\ (\sum j{\in}Basis.\ (if\ j = i\ then\ f\ x{\cdot}T\ i\ else\ 0)\ *_R\ j))))\ x)$
by (*simp add*: *comp-def if-distrib setsum.If-cases*)
show *?thesis*
unfolding $*$
apply (*rule absolutely-integrable-setsum*[*OF finite-Basis*])
apply (*rule absolutely-integrable-linear*)
apply (*rule absolutely-integrable-norm*)
apply (*rule absolutely-integrable-linear*[*OF f*, *unfolded o-def*])
apply (*auto simp*: *linear-linear euclidean-eq-iff*[**where** $'a='c$] *inner-simps intro*!: *linearI*)
done
qed

lemma *absolutely-integrable-max*:
fixes $f\ g :: 'm::euclidean\text{-}space \Rightarrow 'n::euclidean\text{-}space$
assumes *f absolutely-integrable-on s*
and *g absolutely-integrable-on s*
shows $(\lambda x.\ (\sum i{\in}Basis.\ max\ (f(x){\cdot}i)\ (g(x){\cdot}i)\ *_R\ i)::'n)$ *absolutely-integrable-on s*
proof −
have $*$:$\bigwedge x.\ (1\ /\ 2)\ *_R\ (((\sum i{\in}Basis.\ |(f\ x\ -\ g\ x)\ \cdot\ i|\ *_R\ i)::'n) + (f\ x + g\ x)) =$
$(\sum i{\in}Basis.\ max\ (f(x){\cdot}i)\ (g(x){\cdot}i)\ *_R\ i)$
unfolding *euclidean-eq-iff*[**where** $'a='n$] **by** (*auto simp*: *inner-simps*)
note *absolutely-integrable-sub*[*OF assms*] *absolutely-integrable-add*[*OF assms*]
note *absolutely-integrable-vector-abs*[*OF this*(*1*), **where** $T=\lambda x.\ x$] *this*(*2*)
note *absolutely-integrable-add*[*OF this*]
note *absolutely-integrable-cmul*[*OF this*, *of 1/2*]
then show *?thesis* **unfolding** $*$ **.**
qed

lemma *absolutely-integrable-min*:
fixes *f g*::*'m*::*euclidean-space* ⇒ *'n*::*euclidean-space*
assumes *f absolutely-integrable-on s*
and *g absolutely-integrable-on s*
shows (λ*x*. (∑ *i*∈*Basis*. *min* (*f*(*x*)•*i*) (*g*(*x*)•*i*) *∗*$_R$ *i*)::*'n*) *absolutely-integrable-on s*
proof −
have *∗*:⋀*x*. (*1* / *2*) *∗*$_R$ ((*f x* + *g x*) − (∑ *i*∈*Basis*. |(*f x* − *g x*) • *i*| *∗*$_R$ *i*::*'n*)) = (∑ *i*∈*Basis*. *min* (*f*(*x*)•*i*) (*g*(*x*)•*i*) *∗*$_R$ *i*)
unfolding *euclidean-eq-iff*[**where** *'a*=*'n*] **by** (*auto simp*: *inner-simps*)

note *absolutely-integrable-add*[*OF assms*] *absolutely-integrable-sub*[*OF assms*]
note *this*(*1*) *absolutely-integrable-vector-abs*[*OF this*(*2*), **where** *T*=λ*x*. *x*]
note *absolutely-integrable-sub*[*OF this*]
note *absolutely-integrable-cmul*[*OF this*,*of 1*/*2*]
then show *?thesis* **unfolding** *∗* **.**
qed

lemma *absolutely-integrable-abs-eq*:
fixes *f*::*'n*::*euclidean-space* ⇒ *'m*::*euclidean-space*
shows *f absolutely-integrable-on s* ⟷ *f integrable-on s* ∧
(λ*x*. (∑ *i*∈*Basis*. |*f x*•*i*| *∗*$_R$ *i*)::*'m*) *integrable-on s*
(**is** *?l* = *?r*)
proof
assume *?l*
then show *?r*
apply −
apply *rule*
defer
apply (*drule absolutely-integrable-vector-abs*)
apply *auto*
done
next
assume *?r*
{
presume *lem*: ⋀*f*::*'n* ⇒ *'m*. *f integrable-on UNIV* ⟹
(λ*x*. (∑ *i*∈*Basis*. |*f x*•*i*| *∗*$_R$ *i*)::*'m*) *integrable-on UNIV* ⟹
f absolutely-integrable-on UNIV
have *∗*: ⋀*x*. (∑ *i*∈*Basis*. |(*if x* ∈ *s then f x else 0*) • *i*| *∗*$_R$ *i*) =
(*if x* ∈ *s then* (∑ *i*∈*Basis*. |*f x* • *i*| *∗*$_R$ *i*) *else* (*0*::*'m*))
unfolding *euclidean-eq-iff*[**where** *'a*=*'m*]
by *auto*
show *?l*
apply (*subst absolutely-integrable-restrict-univ*[*symmetric*])
apply (*rule lem*)
unfolding *integrable-restrict-univ ∗*
using ‹*?r*›
apply *auto*

```
    done
  }
  fix f :: 'n ⇒ 'm
 assume assms: f integrable-on UNIV (λx. (∑ i∈Basis. |f x•i| *R i)::'m) integrable-on
UNIV
  let ?B = ∑ i∈Basis. integral UNIV (λx. (∑ i∈Basis. |f x•i| *R i)::'m) • i
  show f absolutely-integrable-on UNIV
   apply (rule bounded-variation-absolutely-integrable[OF assms(1), where B=?B])
    apply safe
  proof goal-cases
    case d: (1 d)
    note d'=division-ofD[OF d]
    have (∑ k∈d. norm (integral k f)) ≤
      (∑ k∈d. setsum (op • (integral k (λx. (∑ i∈Basis. |f x•i| *R i)::'m))) Basis)
      apply (rule setsum-mono)
      apply (rule order-trans[OF norm-le-l1])
      apply (rule setsum-mono)
      unfolding lessThan-iff
    proof −
      fix k
      fix i :: 'm
      assume k ∈ d and i: i ∈ Basis
      from d'(4)[OF this(1)] guess a b by (elim exE) note ab=this
      show |integral k f • i| ≤ integral k (λx. (∑ i∈Basis. |f x•i| *R i)::'m) • i
        apply (rule abs-leI)
        unfolding inner-minus-left[symmetric]
        defer
        apply (subst integral-neg[symmetric])
        defer
        apply (rule-tac[1−2] integral-component-le[OF i])
        apply (rule integrable-neg)
        using integrable-on-subcbox[OF assms(1),of a b]
          integrable-on-subcbox[OF assms(2),of a b] i
        unfolding ab
        apply auto
        done
    qed
     also have ... ≤ setsum (op • (integral UNIV (λx. (∑ i∈Basis. |f x•i| *R
i)::'m))) Basis
      apply (subst setsum.commute)
      apply (rule setsum-mono)
    proof goal-cases
      case (1 j)
      have ∗: (λx. ∑ i∈Basis. |f x•i| *R i::'m) integrable-on ⋃ d
        using integrable-on-subdivision[OF d assms(2)] by auto
      have (∑ i∈d. integral i (λx. ∑ i∈Basis. |f x•i| *R i::'m) • j) =
        integral (⋃ d) (λx. ∑ i∈Basis. |f x•i| *R i::'m) • j
      unfolding inner-setsum-left[symmetric] integral-combine-division-topdown[OF
∗ d] ..
```

```
    also have ... ≤ integral UNIV (λx. ∑i∈Basis. |f x•i| *R i::'m) • j
      apply (rule integral-subset-component-le)
      using assms * ‹j ∈ Basis›
      apply auto
      done
    finally show ?case .
  qed
  finally show ?case .
 qed
qed

lemma nonnegative-absolutely-integrable:
 fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
 assumes ∀x∈s. ∀i∈Basis. 0 ≤ f x • i
   and f integrable-on s
 shows f absolutely-integrable-on s
 unfolding absolutely-integrable-abs-eq
 apply rule
 apply (rule assms)thm integrable-eq
 apply (rule integrable-eq[of - f])
 using assms
 apply (auto simp: euclidean-eq-iff[where 'a='m])
 done

lemma absolutely-integrable-integrable-bound:
 fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
 assumes ∀x∈s. norm (f x) ≤ g x
   and f integrable-on s
   and g integrable-on s
 shows f absolutely-integrable-on s
proof −
 {
  presume *: ⋀f::'n ⇒ 'm. ⋀g. ∀x. norm (f x) ≤ g x ⟹ f integrable-on UNIV
⟹
     g integrable-on UNIV ⟹ f absolutely-integrable-on UNIV
   show ?thesis
     apply (subst absolutely-integrable-restrict-univ[symmetric])
     apply (rule *[of - λx. if x∈s then g x else 0])
     using assms
     unfolding integrable-restrict-univ
     apply auto
     done
 }
 fix g
 fix f :: 'n ⇒ 'm
 assume assms: ∀x. norm (f x) ≤ g x f integrable-on UNIV g integrable-on UNIV
 show f absolutely-integrable-on UNIV
  apply (rule bounded-variation-absolutely-integrable[OF assms(2),where B=integral
UNIV g])
```

```
    apply safe
  proof goal-cases
    case d: (1 d)
    note d'=division-ofD[OF d]
    have (∑ k∈d. norm (integral k f)) ≤ (∑ k∈d. integral k g)
      apply (rule setsum-mono)
      apply (rule integral-norm-bound-integral)
      apply (drule-tac[!] d'(4))
      apply safe
      apply (rule-tac[1-2] integrable-on-subcbox)
      using assms
      apply auto
      done
    also have ... = integral (⋃ d) g
      apply (rule integral-combine-division-bottomup[symmetric])
      apply (rule d)
      apply safe
      apply (drule d'(4))
      apply safe
      apply (rule integrable-on-subcbox[OF assms(3)])
      apply auto
      done
    also have ... ≤ integral UNIV g
      apply (rule integral-subset-le)
      defer
      apply (rule integrable-on-subdivision[OF d,of - UNIV])
      prefer 4
      apply rule
      apply (rule-tac y=norm (f x) in order-trans)
      using assms
      apply auto
      done
    finally show ?case .
  qed
qed

lemma absolutely-integrable-absolutely-integrable-bound:
  fixes f :: 'n::euclidean-space ⇒ 'm::euclidean-space
    and g :: 'n::euclidean-space ⇒ 'p::euclidean-space
  assumes ∀ x∈s. norm (f x) ≤ norm (g x)
    and f integrable-on s
    and g absolutely-integrable-on s
  shows f absolutely-integrable-on s
  apply (rule absolutely-integrable-integrable-bound[of s f λx. norm (g x)])
  using assms
  apply auto
  done

lemma absolutely-integrable-inf-real:
```

```
  assumes finite k
    and k ≠ {}
    and ∀ i∈k. (λx. (fs x i)::real) absolutely-integrable-on s
  shows (λx. (Inf ((fs x) ' k))) absolutely-integrable-on s
  using assms
proof induct
  case (insert a k)
  let ?P = (λx.
    if fs x ' k = {} then fs x a
    else min (fs x a) (Inf (fs x ' k))) absolutely-integrable-on s
  show ?case
    unfolding image-insert
    apply (subst Inf-insert-finite)
    apply (rule finite-imageI[OF insert(1)])
  proof (cases k = {})
    case True
    then show ?P
      apply (subst if-P)
      defer
      apply (rule insert(5)[rule-format])
      apply auto
      done
  next
    case False
    then show ?P
      apply (subst if-not-P)
      defer
     apply (rule absolutely-integrable-min[where 'n=real, unfolded Basis-real-def,
simplified])
      defer
      apply(rule insert(3)[OF False])
      using insert(5)
      apply auto
      done
  qed
next
  case empty
  then show ?case by auto
qed

lemma absolutely-integrable-sup-real:
  assumes finite k
    and k ≠ {}
    and ∀ i∈k. (λx. (fs x i)::real) absolutely-integrable-on s
  shows (λx. (Sup ((fs x) ' k))) absolutely-integrable-on s
  using assms
proof induct
  case (insert a k)
  let ?P = (λx.
```

if fs x ‘ k = {} then fs x a
else max (fs x a) (Sup (fs x ‘ k))) absolutely-integrable-on s
show *?case*
unfolding *image-insert*
apply (*subst Sup-insert-finite*)
apply (*rule finite-imageI*[*OF insert*(*1*)])
proof (*cases k = {}*)
case *True*
then show *?P*
apply (*subst if-P*)
defer
apply (*rule insert*(*5*)[*rule-format*])
apply *auto*
done
next
case *False*
then show *?P*
apply (*subst if-not-P*)
defer
apply (*rule absolutely-integrable-max*[**where** *'n=real*, *unfolded Basis-real-def*, *simplified*])
defer
apply (*rule insert*(*3*)[*OF False*])
using *insert*(*5*)
apply *auto*
done
qed
qed *auto*

35.66 differentiation under the integral sign

lemma *tube-lemma*:
assumes *compact K*
assumes *open W*
assumes {*x0*} × *K* ⊆ *W*
shows ∃ *X0*. *x0* ∈ *X0* ∧ *open X0* ∧ *X0* × *K* ⊆ *W*
proof −
{
fix *y* **assume** *y* ∈ *K*
then have (*x0*, *y*) ∈ *W* **using** *assms* **by** *auto*
with ‹*open W*›
have ∃ *X0 Y*. *open X0* ∧ *open Y* ∧ *x0* ∈ *X0* ∧ *y* ∈ *Y* ∧ *X0* × *Y* ⊆ *W*
by (*rule open-prod-elim*) *blast*
} then obtain *X0 Y* **where**
∀ *y* ∈ *K*. *open* (*X0 y*) ∧ *open* (*Y y*) ∧ *x0* ∈ *X0 y* ∧ *y* ∈ *Y y* ∧ *X0 y* × *Y y* ⊆ *W*
by *metis*
moreover
then have ∀ *t*∈*Y ‘ K*. *open t K* ⊆ ⋃(*Y ‘ K*) **by** *auto*

```
  with ‹compact K› obtain CC where CC ⊆ Y ‘ K finite CC K ⊆ ⋃ CC
    by (rule compactE)
  moreover
  then obtain c where c:
    ⋀C. C ∈ CC ⟹ c C ∈ K ∧ C = Y (c C)
    by (force intro!: choice)
  ultimately show ?thesis
    by (force intro!: exI[where x=⋂ C∈CC. X0 (c C)])
qed

lemma continuous-on-prod-compactE:
  fixes fx::'a::topological-space × 'b::topological-space ⇒ 'c::metric-space
    and e::real
  assumes cont-fx: continuous-on (U × C) fx
  assumes compact C
  assumes [intro]: x0 ∈ U
  notes [continuous-intros] = continuous-on-compose2[OF cont-fx]
  assumes e > 0
  obtains X0 where x0 ∈ X0 open X0
    ∀x∈X0 ∩ U. ∀t ∈ C. dist (fx (x, t)) (fx (x0, t)) ≤ e
proof −
  def psi ≡ λ(x, t). dist (fx (x, t)) (fx (x0, t))
  def W0 ≡ {(x, t) ∈ U × C. psi (x, t) < e}
  have W0-eq: W0 = psi −‘ {..<e} ∩ U × C
    by (auto simp: vimage-def W0-def)
  have open {..<e} by simp
  have continuous-on (U × C) psi
    by (auto intro!: continuous-intros simp: psi-def split-beta')
 from this[unfolded continuous-on-open-invariant, rule-format, OF ‹open {..<e}›]
  obtain W where W: open W W ∩ U × C = W0 ∩ U × C
    unfolding W0-eq by blast
  have {x0} × C ⊆ W ∩ U × C
    unfolding W
    by (auto simp: W0-def psi-def ‹0 < e›)
  then have {x0} × C ⊆ W by blast
  from tube-lemma[OF ‹compact C› ‹open W› this]
  obtain X0 where X0: x0 ∈ X0 open X0 X0 × C ⊆ W
    by blast

  have ∀x∈X0 ∩ U. ∀t ∈ C. dist (fx (x, t)) (fx (x0, t)) ≤ e
  proof safe
    fix x assume x: x ∈ X0 x ∈ U
    fix t assume t: t ∈ C
    have dist (fx (x, t)) (fx (x0, t)) = psi (x, t)
      by (auto simp: psi-def)
    also
    {
      have (x, t) ∈ X0 × C
        using t x
```

```
      by auto
    also note ‹... ⊆ W›
    finally have (x, t) ∈ W .
    with t x have (x, t) ∈ W ∩ U × C
      by blast
    also note ‹W ∩ U × C = W0 ∩ U × C›
    finally  have psi (x, t) < e
      by (auto simp: W0-def)
  }
  finally show dist (fx (x, t)) (fx (x0, t)) ≤ e by simp
 qed
 from X0(1,2) this show ?thesis ..
qed

lemma integral-continuous-on-param:
 fixes f::'a::topological-space ⇒ 'b::euclidean-space ⇒ 'c::banach
 assumes cont-fx: continuous-on (U × cbox a b) (λ(x, t). f x t)
 shows continuous-on U (λx. integral (cbox a b) (f x))
proof cases
 assume content (cbox a b) ≠ 0
 then have ne: cbox a b ≠ {} by auto

 note [continuous-intros] =
   continuous-on-compose2[OF cont-fx, where f=λy. Pair x y for x,
     unfolded split-beta fst-conv snd-conv]

 show ?thesis
   unfolding continuous-on-def
 proof (safe intro!: tendstoI)
   fix e'::real and x
   assume e' > 0
   def e ≡ e' / (content (cbox a b) + 1)
  have e > 0 using ‹e' > 0› by (auto simp: e-def intro!: divide-pos-pos add-nonneg-pos)
   assume x ∈ U
   from continuous-on-prod-compactE[OF cont-fx compact-cbox ‹x ∈ U› ‹0 < e›]
   obtain X0 where X0: x ∈ X0 open X0
     and fx-bound: ⋀y t. y ∈ X0 ∩ U ⟹ t ∈ cbox a b ⟹ norm (f y t − f x t)
≤ e
     unfolding split-beta fst-conv snd-conv dist-norm
     by metis
   have ∀F y in at x within U. y ∈ X0 ∩ U
     using X0(1) X0(2) eventually-at-topological by auto
    then show ∀F y in at x within U. dist (integral (cbox a b) (f y)) (integral
(cbox a b) (f x)) < e'
   proof eventually-elim
     case (elim y)
     have dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x)) =
       norm (integral (cbox a b) (λt. f y t − f x t))
       using elim ‹x ∈ U›
```

```
      unfolding dist-norm
      by (subst integral-diff)
        (auto intro!: integrable-continuous continuous-intros)
    also have ... ≤ e * content (cbox a b)
      using elim ‹x ∈ U›
      by (intro integrable-bound)
        (auto intro!: fx-bound ‹x ∈ U › less-imp-le[OF ‹0 < e›]
          integrable-continuous continuous-intros)
    also have ... < e'
      using ‹0 < e'› ‹e > 0›
      by (auto simp: e-def divide-simps)
    finally show dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x)) < e' .
  qed
 qed
qed (auto intro!: continuous-on-const)

lemma eventually-closed-segment:
 fixes x0::'a::real-normed-vector
 assumes open X0 x0 ∈ X0
 shows ∀F x in at x0 within U. closed-segment x0 x ⊆ X0
proof −
 from openE[OF assms]
 obtain e where e: 0 < e ball x0 e ⊆ X0 .
 then have ∀F x in at x0 within U. x ∈ ball x0 e
   by (auto simp: dist-commute eventually-at)
 then show ?thesis
 proof eventually-elim
   case (elim x)
   have x0 ∈ ball x0 e using ‹e > 0› by simp
   from convex-ball[unfolded convex-contains-segment, rule-format, OF this elim]
   have closed-segment x0 x ⊆ ball x0 e .
   also note ‹... ⊆ X0›
   finally show ?case .
 qed
qed

lemma leibniz-rule:
 fixes f::'a::banach ⇒ 'b::euclidean-space ⇒ 'c::banach
 assumes fx: ⋀x t. x ∈ U ⟹ t ∈ cbox a b ⟹
   ((λx. f x t) has-derivative blinfun-apply (fx x t)) (at x within U)
 assumes integrable-f2: ⋀x. x ∈ U ⟹ f x integrable-on cbox a b
 assumes cont-fx: continuous-on (U × (cbox a b)) (λ(x, t). fx x t)
 assumes [intro]: x0 ∈ U
 assumes convex U
 shows
   ((λx. integral (cbox a b) (f x)) has-derivative integral (cbox a b) (fx x0)) (at x0
within U)
   (is (?F has-derivative ?dF) -)
proof cases
```

```
  assume content (cbox a b) ≠ 0
  then have ne: cbox a b ≠ {} by auto
  note [continuous-intros] =
    continuous-on-compose2[OF cont-fx, where f=λy. Pair x y for x,
      unfolded split-beta fst-conv snd-conv]
  show ?thesis
 proof (intro has-derivativeI bounded-linear-scaleR-left tendstoI, fold norm-conv-dist)
    have cont-f1: ⋀t. t ∈ cbox a b ⟹ continuous-on U (λx. f x t)
    by (auto simp: continuous-on-eq-continuous-within intro!: has-derivative-continuous
fx)
    note [continuous-intros] = continuous-on-compose2[OF cont-f1]
    fix e'::real
    assume e' > 0
    def e ≡ e' / (content (cbox a b) + 1)
   have e > 0 using ‹e' > 0› by (auto simp: e-def intro!: divide-pos-pos add-nonneg-pos)
    from continuous-on-prod-compactE[OF cont-fx compact-cbox ‹x0 ∈ U› ‹e > 0›]
    obtain X0 where X0: x0 ∈ X0 open X0
      and fx-bound: ⋀x t. x ∈ X0 ∩ U ⟹ t ∈ cbox a b ⟹ norm (fx x t − fx x0
t) ≤ e
      unfolding split-beta fst-conv snd-conv
      by (metis dist-norm)

    note eventually-closed-segment[OF ‹open X0› ‹x0 ∈ X0›, of U]
    moreover
    have ∀F x in at x0 within U. x ∈ X0
      using ‹open X0› ‹x0 ∈ X0› eventually-at-topological by blast
    moreover have ∀F x in at x0 within U. x ≠ x0
      by (auto simp: eventually-at-filter)
    moreover have ∀F x in at x0 within U. x ∈ U
      by (auto simp: eventually-at-filter)
    ultimately
     show ∀F x in at x0 within U. norm ((?F x − ?F x0 − ?dF (x − x0)) /R
norm (x − x0)) < e'
    proof eventually-elim
      case (elim x)
      from elim have 0 < norm (x − x0) by simp
      have closed-segment x0 x ⊆ U
        by (rule ‹convex U›[unfolded convex-contains-segment, rule-format, OF ‹x0
∈ U› ‹x ∈ U›])
      from elim have [intro]: x ∈ U by auto

      have ?F x − ?F x0 − ?dF (x − x0) =
        integral (cbox a b) (λy. f x y − f x0 y − fx x0 y (x − x0))
        (is - = ?id)
        using ‹x ≠ x0›
        by (subst blinfun-apply-integral integral-diff,
            auto intro!: integrable-diff integrable-f2 continuous-intros
              intro: integrable-continuous)+
      also
```

```
    {
      fix t assume t: t ∈ (cbox a b)
      have seg: ⋀t. t ∈ {0..1} ⟹ x0 + t *R (x − x0) ∈ X0 ∩ U
        using ‹closed-segment x0 x ⊆ U›
          ‹closed-segment x0 x ⊆ X0›
        by (force simp: closed-segment-def algebra-simps)
      from t have deriv:
        ((λx. f x t) has-derivative (fx y t)) (at y within X0 ∩ U)
        if y ∈ X0 ∩ U for y
        unfolding has-vector-derivative-def[symmetric]
        using that ‹x ∈ X0›
        by (intro has-derivative-within-subset[OF fx]) auto
      have ∀ x ∈ X0 ∩ U. onorm (blinfun-apply (fx x t) − (fx x0 t)) ≤ e
        using fx-bound t
     by (auto simp add: norm-blinfun-def fun-diff-def blinfun.bilinear-simps[symmetric])
      from differentiable-bound-linearization[OF seg deriv this] X0
      have norm (f x t − f x0 t − fx x0 t (x − x0)) ≤ e * norm (x − x0)
        by (auto simp add: ac-simps)
    }
    then have norm ?id ≤ integral (cbox a b) (λ-. e * norm (x − x0))
      by (intro integral-norm-bound-integral)
        (auto intro!: continuous-intros integrable-diff integrable-f2
          intro: integrable-continuous)
    also have ... = content (cbox a b) * e * norm (x − x0)
      by simp
    also have ... < e' * norm (x − x0)
      using ‹e' > 0› content-pos-le[of a b]
      by (intro mult-strict-right-mono[OF - ‹0 < norm (x − x0)›])
        (auto simp: divide-simps e-def)
    finally have norm (?F x − ?F x0 − ?dF (x − x0)) < e' * norm (x − x0) .
    then show ?case
      by (auto simp: divide-simps)
  qed
 qed (rule blinfun.bounded-linear-right)
qed (auto intro!: derivative-eq-intros simp: blinfun.bilinear-simps)

lemma
  has-vector-derivative-eq-has-derivative-blinfun:
  (f has-vector-derivative f') (at x within U) ⟷
    (f has-derivative blinfun-scaleR-left f') (at x within U)
  by (simp add: has-vector-derivative-def)

lemma leibniz-rule-vector-derivative:
  fixes f::real ⇒ 'b::euclidean-space ⇒ 'c::banach
  assumes fx: ⋀x t. x ∈ U ⟹ t ∈ cbox a b ⟹
      ((λx. f x t) has-vector-derivative (fx x t)) (at x within U)
  assumes integrable-f2: ⋀x. x ∈ U ⟹ (f x) integrable-on cbox a b
  assumes cont-fx: continuous-on (U × cbox a b) (λ(x, t). fx x t)
  assumes U: x0 ∈ U convex U
```

shows ((λx. *integral* (*cbox a b*) (*f x*)) *has-vector-derivative integral* (*cbox a b*) (*fx x0*))
(*at x0 within U*)
proof −
note [*continuous-intros*] =
continuous-on-compose2[*OF cont-fx*, **where** *f*=λy. *Pair x y* **for** *x*,
unfolded split-beta fst-conv snd-conv]
have ∗: *blinfun-scaleR-left* (*integral* (*cbox a b*) (*fx x0*)) =
integral (*cbox a b*) (λt. *blinfun-scaleR-left* (*fx x0 t*))
by (*subst integral-linear*[*symmetric*])
(*auto simp*: *has-vector-derivative-def o-def*
intro!: *integrable-continuous U continuous-intros bounded-linear-intros*)
show *?thesis*
unfolding *has-vector-derivative-eq-has-derivative-blinfun*
apply (*rule has-derivative-eq-rhs*)
apply (*rule leibniz-rule*[*OF - integrable-f2 - U*, **where** *fx*=λx t. *blinfun-scaleR-left* (*fx x t*)])
using *fx cont-fx*
apply (*auto simp*: *has-vector-derivative-def* ∗ *split-beta intro*!: *continuous-intros*)
done
qed

lemma
has-field-derivative-eq-has-derivative-blinfun:
(*f has-field-derivative f'*) (*at x within U*) ⟷ (*f has-derivative blinfun-mult-right f'*) (*at x within U*)
by (*simp add*: *has-field-derivative-def*)

lemma *leibniz-rule-field-derivative*:
fixes *f*::'*a*::{*real-normed-field*, *banach*} ⇒ '*b*::*euclidean-space* ⇒ '*a*
assumes *fx*: ⋀x t. *x* ∈ *U* ⟹ *t* ∈ *cbox a b* ⟹ ((λx. *f x t*) *has-field-derivative fx x t*) (*at x within U*)
assumes *integrable-f2*: ⋀x. *x* ∈ *U* ⟹ (*f x*) *integrable-on cbox a b*
assumes *cont-fx*: *continuous-on* (*U* × (*cbox a b*)) (λ(*x*, *t*). *fx x t*)
assumes *U*: *x0* ∈ *U convex U*
shows ((λx. *integral* (*cbox a b*) (*f x*)) *has-field-derivative integral* (*cbox a b*) (*fx x0*)) (*at x0 within U*)
proof −
note [*continuous-intros*] =
continuous-on-compose2[*OF cont-fx*, **where** *f*=λy. *Pair x y* **for** *x*,
unfolded split-beta fst-conv snd-conv]
have ∗: *blinfun-mult-right* (*integral* (*cbox a b*) (*fx x0*)) =
integral (*cbox a b*) (λt. *blinfun-mult-right* (*fx x0 t*))
by (*subst integral-linear*[*symmetric*])
(*auto simp*: *has-vector-derivative-def o-def*
intro!: *integrable-continuous U continuous-intros bounded-linear-intros*)
show *?thesis*
unfolding *has-field-derivative-eq-has-derivative-blinfun*
apply (*rule has-derivative-eq-rhs*)

apply (*rule leibniz-rule*[*OF - integrable-f2 - U*, **where** *fx*=λ*x t*. *blinfun-mult-right* (*fx x t*)])
using *fx cont-fx*
apply (*auto simp*: *has-field-derivative-def* ∗ *split-beta intro*!: *continuous-intros*)
done
qed

35.67 Exchange uniform limit and integral

lemma
uniform-limit-integral:
fixes *f*::*'a* ⇒ *'b*::*euclidean-space* ⇒ *'c*::*banach*
assumes *u*: *uniform-limit* (*cbox a b*) *f g F*
assumes *c*: ⋀*n*. *continuous-on* (*cbox a b*) (*f n*)
assumes [*simp*]: *F* ≠ *bot*
obtains *I J* **where**
⋀*n*. (*f n has-integral I n*) (*cbox a b*)
(*g has-integral J*) (*cbox a b*)
(*I* ⟶ *J*) *F*
proof −
have *fi*[*simp*]: *f n integrable-on* (*cbox a b*) **for** *n*
by (*auto intro*!: *integrable-continuous assms*)
then obtain *I* **where** *I*: ⋀*n*. (*f n has-integral I n*) (*cbox a b*)
by *atomize-elim* (*auto simp*: *integrable-on-def intro*!: *choice*)

moreover

have *gi*[*simp*]: *g integrable-on* (*cbox a b*)
by (*auto intro*!: *integrable-continuous uniform-limit-theorem*[*OF - u*] *eventuallyI c*)
then obtain *J* **where** *J*: (*g has-integral J*) (*cbox a b*)
by *blast*

moreover

have (*I* ⟶ *J*) *F*
proof *cases*
assume *content* (*cbox a b*) = *0*
hence *I* = (λ-. *0*) *J* = *0*
by (*auto intro*!: *has-integral-unique I J*)
thus *?thesis* **by** *simp*
next
assume *content-nonzero*: *content* (*cbox a b*) ≠ *0*
show *?thesis*
proof (*rule tendstoI*)
fix *e*::*real*
assume *e* > *0*
def *e'* ≡ *e* / *2*
with ‹*e* > *0*› **have** *e'* > *0* **by** *simp*

then have $\forall_F\ n\ in\ F.\ \forall x{\in}cbox\ a\ b.\ norm\ (f\ n\ x - g\ x) < e'\ /\ content\ (cbox\ a\ b)$
using *u content-nonzero content-pos-le*[*of a b*]
by (*auto simp*: *uniform-limit-iff dist-norm*)
then show $\forall_F\ n\ in\ F.\ dist\ (I\ n)\ J < e$
proof *eventually-elim*
case (*elim n*)
have $I\ n = integral\ (cbox\ a\ b)\ (f\ n)$
$J = integral\ (cbox\ a\ b)\ g$
using *I*[*of n*] *J* **by** (*simp-all add*: *integral-unique*)
then have $dist\ (I\ n)\ J = norm\ (integral\ (cbox\ a\ b)\ (\lambda x.\ f\ n\ x - g\ x))$
by (*simp add*: *integral-diff dist-norm*)
also have $\ldots \le integral\ (cbox\ a\ b)\ (\lambda x.\ (e'\ /\ content\ (cbox\ a\ b)))$
using *elim*
by (*intro integral-norm-bound-integral*)
(*auto intro*!: *integrable-diff absolutely-integrable-onI*)
also have $\ldots < e$
using ‹$0 < e$›
by (*simp add*: *e'-def*)
finally show *?case* **.**
qed
qed
qed
ultimately show *?thesis* **..**
qed

35.68 Dominated convergence

lemma *dominated-convergence*:
fixes $f :: nat \Rightarrow 'n{::}euclidean\text{-}space \Rightarrow real$
assumes $\bigwedge k.\ (f\ k)\ integrable\text{-}on\ s\ h\ integrable\text{-}on\ s$
and $\bigwedge k.\ \forall x \in s.\ norm\ (f\ k\ x) \le h\ x$
and $\forall x \in s.\ ((\lambda k.\ f\ k\ x) \longrightarrow g\ x)\ sequentially$
shows *g integrable-on s*
and $((\lambda k.\ integral\ s\ (f\ k)) \longrightarrow integral\ s\ g)\ sequentially$
proof −
have *bdd-below*[*simp*]: $\bigwedge x\ P.\ x \in s \Longrightarrow bdd\text{-}below\ \{f\ n\ x\ |n.\ P\ n\}$
proof (*safe intro*!: *bdd-belowI*)
fix *n x* **show** $x \in s \Longrightarrow -h\ x \le f\ n\ x$
using *assms*(*3*)[*rule-format*, *of x n*] **by** *simp*
qed
have *bdd-above*[*simp*]: $\bigwedge x\ P.\ x \in s \Longrightarrow bdd\text{-}above\ \{f\ n\ x\ |n.\ P\ n\}$
proof (*safe intro*!: *bdd-aboveI*)
fix *n x* **show** $x \in s \Longrightarrow f\ n\ x \le h\ x$
using *assms*(*3*)[*rule-format*, *of x n*] **by** *simp*
qed

have $\bigwedge m.\ (\lambda x.\ Inf\ \{f\ j\ x\ |j.\ m \le j\})\ integrable\text{-}on\ s \wedge$
$((\lambda k.\ integral\ s\ (\lambda x.\ Inf\ \{f\ j\ x\ |j.\ j \in \{m..m + k\}\})) \longrightarrow$

```
  integral s (λx. Inf {f j x |j. m ≤ j}))sequentially
 proof (rule monotone-convergence-decreasing, safe)
  fix m :: nat
  show bounded {integral s (λx. Inf {f j x |j. j ∈ {m..m + k}}) |k. True}
   unfolding bounded-iff
   apply (rule-tac x=integral s h in exI)
  proof safe
   fix k :: nat
   show norm (integral s (λx. Inf {f j x |j. j ∈ {m..m + k}})) ≤ integral s h
    apply (rule integral-norm-bound-integral)
    unfolding simple-image
    apply (rule absolutely-integrable-onD)
    apply (rule absolutely-integrable-inf-real)
    prefer 5
    unfolding real-norm-def
    apply rule
    apply (rule cInf-abs-ge)
    prefer 5
    apply rule
    apply (rule-tac g = h in absolutely-integrable-integrable-bound)
    using assms
    unfolding real-norm-def
    apply auto
    done
  qed
  fix k :: nat
  show (λx. Inf {f j x |j. j ∈ {m..m + k}}) integrable-on s
   unfolding simple-image
   apply (rule absolutely-integrable-onD)
   apply (rule absolutely-integrable-inf-real)
   prefer 3
   using absolutely-integrable-integrable-bound[OF assms(3,1,2)]
   apply auto
   done
  fix x
  assume x: x ∈ s
  show Inf {f j x |j. j ∈ {m..m + Suc k}} ≤ Inf {f j x |j. j ∈ {m..m + k}}
   by (rule cInf-superset-mono) auto
  let ?S = {f j x| j. m ≤ j}
  show ((λk. Inf {f j x |j. j ∈ {m..m + k}}) ⟶ Inf ?S) sequentially
  proof (rule LIMSEQ-I, goal-cases)
   case r: (1 r)

   have ∃ y∈?S. y < Inf ?S + r
    by (subst cInf-less-iff[symmetric]) (auto simp: ‹x∈s› r)
   then obtain N where N: f N x < Inf ?S + r m ≤ N
    by blast

   show ?case
```

```
      apply (rule-tac x=N in exI)
      apply safe
    proof goal-cases
      case prems: (1 n)
      have *: ⋀y ix. y < Inf ?S + r ⟶ Inf ?S ≤ ix ⟶ ix ≤ y ⟶ |ix − Inf
?S| < r
        by arith
      show ?case
        unfolding real-norm-def
          apply (rule *[rule-format, OF N(1)])
          apply (rule cInf-superset-mono, auto simp: ‹x∈s›) []
          apply (rule cInf-lower)
          using N prems
          apply auto []
          apply simp
          done
    qed
  qed
 qed
 note dec1 = conjunctD2[OF this]

 have ⋀m. (λx. Sup {f j x |j. m ≤ j}) integrable-on s ∧
  ((λk. integral s (λx. Sup {f j x |j. j ∈ {m..m + k}})) ⟶⟶
  integral s (λx. Sup {f j x |j. m ≤ j})) sequentially
 proof (rule monotone-convergence-increasing,safe)
  fix m :: nat
  show bounded {integral s (λx. Sup {f j x |j. j ∈ {m..m + k}}) |k. True}
   unfolding bounded-iff
   apply (rule-tac x=integral s h in exI)
  proof safe
   fix k :: nat
   show norm (integral s (λx. Sup {f j x |j. j ∈ {m..m + k}})) ≤ integral s h
    apply (rule integral-norm-bound-integral) unfolding simple-image
    apply (rule absolutely-integrable-onD)
    apply(rule absolutely-integrable-sup-real)
    prefer 5 unfolding real-norm-def
    apply rule
    apply (rule cSup-abs-le)
    using assms
    apply (force simp add: )
    prefer 4
    apply rule
    apply (rule-tac g=h in absolutely-integrable-integrable-bound)
    using assms
    unfolding real-norm-def
    apply auto
    done
  qed
  fix k :: nat
```

```
    show (λx. Sup {f j x |j. j ∈ {m..m + k}}) integrable-on s
      unfolding simple-image
      apply (rule absolutely-integrable-onD)
      apply (rule absolutely-integrable-sup-real)
      prefer 3
      using absolutely-integrable-integrable-bound[OF assms(3,1,2)]
      apply auto
      done
    fix x
    assume x: x∈s
    show Sup {f j x |j. j ∈ {m..m + Suc k}} ≥ Sup {f j x |j. j ∈ {m..m + k}}
      by (rule cSup-subset-mono) auto
    let ?S = {f j x| j. m ≤ j}
    show ((λk. Sup {f j x |j. j ∈ {m..m + k}}) ⟶ Sup ?S) sequentially
    proof (rule LIMSEQ-I, goal-cases)
      case r: (1 r)
      have ∃ y∈?S. Sup ?S − r < y
        by (subst less-cSup-iff[symmetric]) (auto simp: r ‹x∈s›)
      then obtain N where N: Sup ?S − r < f N x m ≤ N
        by blast

      show ?case
        apply (rule-tac x=N in exI)
        apply safe
      proof goal-cases
        case prems: (1 n)
       have ∗: ⋀y ix. Sup ?S − r < y ⟶ ix ≤ Sup ?S ⟶ y ≤ ix ⟶ |ix − Sup
?S| < r
          by arith
        show ?case
          apply simp
          apply (rule ∗[rule-format, OF N(1)])
          apply (rule cSup-subset-mono, auto simp: ‹x∈s›) []
          apply (subst cSup-upper)
          using N prems
          apply auto
          done
      qed
    qed
  qed
  note inc1 = conjunctD2[OF this]

  have g integrable-on s ∧
    ((λk. integral s (λx. Inf {f j x |j. k ≤ j})) ⟶ integral s g) sequentially
    apply (rule monotone-convergence-increasing,safe)
    apply fact
  proof −
    show bounded {integral s (λx. Inf {f j x |j. k ≤ j}) |k. True}
      unfolding bounded-iff apply(rule-tac x=integral s h in exI)
```

```
    proof safe
      fix k :: nat
      show norm (integral s (λx. Inf {f j x |j. k ≤ j})) ≤ integral s h
        apply (rule integral-norm-bound-integral)
        apply fact+
        unfolding real-norm-def
        apply rule
        apply (rule cInf-abs-ge)
        using assms(3)
        apply auto
        done
    qed
    fix k :: nat and x
    assume x: x ∈ s

    have *: ⋀x y::real. x ≥ − y ⟹ − x ≤ y by auto
    show Inf {f j x |j. k ≤ j} ≤ Inf {f j x |j. Suc k ≤ j}
      by (intro cInf-superset-mono) (auto simp: ‹x∈s›)

    show (λk::nat. Inf {f j x |j. k ≤ j}) ⟶ g x
    proof (rule LIMSEQ-I, goal-cases)
      case r: (1 r)
      then have 0<r/2
        by auto
     from assms(4)[THEN bspec, THEN LIMSEQ-D, OF x this] guess N .. note
N = this
      show ?case
        apply (rule-tac x=N in exI)
        apply safe
        unfolding real-norm-def
        apply (rule le-less-trans[of - r/2])
        apply (rule cInf-asclose)
        apply safe
        defer
        apply (rule less-imp-le)
        using N r
        apply auto
        done
    qed
  qed
  note inc2 = conjunctD2[OF this]

  have g integrable-on s ∧
    ((λk. integral s (λx. Sup {f j x |j. k ≤ j})) ⟶ integral s g) sequentially
    apply (rule monotone-convergence-decreasing,safe)
    apply fact
  proof −
    show bounded {integral s (λx. Sup {f j x |j. k ≤ j}) |k. True}
      unfolding bounded-iff
```

```
    apply (rule-tac x=integral s h in exI)
   proof safe
    fix k :: nat
    show norm (integral s (λx. Sup {f j x |j. k ≤ j})) ≤ integral s h
     apply (rule integral-norm-bound-integral)
     apply fact+
     unfolding real-norm-def
     apply rule
     apply (rule cSup-abs-le)
     using assms(3)
     apply auto
     done
   qed
   fix k :: nat
   fix x
   assume x: x ∈ s

   show Sup {f j x |j. k ≤ j} ≥ Sup {f j x |j. Suc k ≤ j}
    by (rule cSup-subset-mono) (auto simp: ‹x∈s›)
   show ((λk. Sup {f j x |j. k ≤ j}) ⟶ g x) sequentially
   proof (rule LIMSEQ-I, goal-cases)
    case r: (1 r)
    then have 0<r/2
     by auto
    from assms(4)[THEN bspec, THEN LIMSEQ-D, OF x this] guess N .. note
N=this
    show ?case
     apply (rule-tac x=N in exI,safe)
     unfolding real-norm-def
     apply (rule le-less-trans[of - r/2])
     apply (rule cSup-asclose)
     apply safe
     defer
     apply (rule less-imp-le)
     using N r
     apply auto
     done
   qed
  qed
  note dec2 = conjunctD2[OF this]

  show g integrable-on s by fact
  show ((λk. integral s (f k)) ⟶ integral s g) sequentially
  proof (rule LIMSEQ-I, goal-cases)
   case r: (1 r)
  from LIMSEQ-D [OF inc2(2) r] guess N1 .. note N1=this[unfolded real-norm-def]
  from LIMSEQ-D [OF dec2(2) r] guess N2 .. note N2=this[unfolded real-norm-def]
   show ?case
   proof (rule-tac x=N1+N2 in exI, safe)
```

fix *n*
assume *n*: $n \geq N1 + N2$
have $*$: $\bigwedge i0\ i\ i1\ g.\ |i0 - g| < r \longrightarrow |i1 - g| < r \longrightarrow i0 \leq i \longrightarrow i \leq i1 \longrightarrow |i - g| < r$
by *arith*
show *norm* (*integral s* (*f n*) − *integral s g*) < *r*
unfolding *real-norm-def*
proof (*rule* $*$[*rule-format*,*OF N1*[*rule-format*] *N2*[*rule-format*], *of n n*])
show *integral s* ($\lambda x.$ *Inf* {*f j x* |*j*. *n* ≤ *j*}) ≤ *integral s* (*f n*)
by (*rule integral-le*[*OF dec1*(*1*) *assms*(*1*)]) (*auto intro*!: *cInf-lower*)
show *integral s* (*f n*) ≤ *integral s* ($\lambda x.$ *Sup* {*f j x* |*j*. *n* ≤ *j*})
by (*rule integral-le*[*OF assms*(*1*) *inc1*(*1*)]) (*auto intro*!: *cSup-upper*)
qed (*insert n*, *auto*)
qed
qed
qed

lemma *has-integral-dominated-convergence*:
fixes *f* :: *nat* ⇒ *'n*::*euclidean-space* ⇒ *real*
assumes $\bigwedge k.$ (*f k has-integral y k*) *s h integrable-on s*
$\bigwedge k.\ \forall x{\in}s.$ *norm* (*f k x*) ≤ *h x* $\forall x{\in}s.$ ($\lambda k.$ *f k x*) ⟶ *g x*
and *x*: *y* ⟶ *x*
shows (*g has-integral x*) *s*
proof −
have *int-f*: $\bigwedge k.$ (*f k*) *integrable-on s*
using *assms* **by** (*auto simp*: *integrable-on-def*)
have (*g has-integral* (*integral s g*)) *s*
by (*intro integrable-integral dominated-convergence*[*OF int-f assms*(*2*)]) *fact*+
moreover have *integral s g* = *x*
proof (*rule LIMSEQ-unique*)
show ($\lambda i.$ *integral s* (*f i*)) ⟶ *x*
using *integral-unique*[*OF assms*(*1*)] *x* **by** *simp*
show ($\lambda i.$ *integral s* (*f i*)) ⟶ *integral s g*
by (*intro dominated-convergence*[*OF int-f assms*(*2*)]) *fact*+
qed
ultimately show *?thesis*
by *simp*
qed

35.69 Compute a double integral using iterated integrals and switching the order of integration

lemma *setcomp-dot1*: {*z*. *P* (*z* · (*i*,*0*))} = {(*x*,*y*). *P*(*x* · *i*)}
by *auto*

lemma *setcomp-dot2*: {*z*. *P* (*z* · (*0*,*i*))} = {(*x*,*y*). *P*(*y* · *i*)}
by *auto*

lemma *Sigma-Int-Paircomp1*: (*Sigma A B*) ∩ {(*x*, *y*). *P x*} = *Sigma* (*A* ∩ {*x*. *P*

$x\}$) B
by *blast*

lemma *Sigma-Int-Paircomp2*: (*Sigma A B*) ∩ {(*x*, *y*). *P y*} = *Sigma A* (λ*z*. *B z* ∩ {*y*. *P y*})
by *blast*

lemma *continuous-on-imp-integrable-on-Pair1*:
fixes *f* :: - ⇒ *'b*::*banach*
assumes *con*: *continuous-on* (*cbox* (*a*,*c*) (*b*,*d*)) *f* **and** *x*: *x* ∈ *cbox a b*
shows (λ*y*. *f* (*x*, *y*)) *integrable-on* (*cbox c d*)
proof −
have *f* ∘ (λ*y*. (*x*, *y*)) *integrable-on* (*cbox c d*)
apply (*rule integrable-continuous*)
apply (*rule continuous-on-compose* [*OF* - *continuous-on-subset* [*OF con*]])
using *x*
apply (*auto intro*: *continuous-on-Pair continuous-on-const continuous-on-id continuous-on-subset con*)
done
then show *?thesis*
by (*simp add*: *o-def*)
qed

lemma *integral-integrable-2dim*:
fixes *f* :: (*'a*::*euclidean-space* ∗ *'b*::*euclidean-space*) ⇒ *'c*::*banach*
assumes *continuous-on* (*cbox* (*a*,*c*) (*b*,*d*)) *f*
shows (λ*x*. *integral* (*cbox c d*) (λ*y*. *f* (*x*,*y*))) *integrable-on cbox a b*
proof (*cases content*(*cbox c d*) = *0*)
case *True*
then show *?thesis*
by (*simp add*: *True integrable-const*)
next
case *False*
have *uc*: *uniformly-continuous-on* (*cbox* (*a*,*c*) (*b*,*d*)) *f*
by (*simp add*: *assms compact-cbox compact-uniformly-continuous*)
{ fix *x*::*'a* **and** *e*::*real*
assume *x*: *x* ∈ *cbox a b* **and** *e*: *0* < *e*
then have *e2-gt*: *0* < *e* / *2* / *content* (*cbox c d*) **and** *e2-less*: *e* / *2* / *content* (*cbox c d*) ∗ *content* (*cbox c d*) < *e*
by (*auto simp*: *False content-lt-nz e*)
then obtain *dd*
where *dd*: ⋀*x x'*. ⟦*x*∈*cbox* (*a*, *c*) (*b*, *d*); *x'*∈*cbox* (*a*, *c*) (*b*, *d*); *norm* (*x'* − *x*) < *dd*⟧
⟹ *norm* (*f x'* − *f x*) ≤ *e* / (*2* ∗ *content* (*cbox c d*)) *dd*>*0*
using *uc* [*unfolded uniformly-continuous-on-def*, *THEN spec*, *of e* / (*2* ∗ *content* (*cbox c d*))]
by (*auto simp*: *dist-norm intro*: *less-imp-le*)
have ∃ *delta*>*0*. ∀ *x'*∈*cbox a b*. *norm* (*x'* − *x*) < *delta* ⟶ *norm* (*integral* (*cbox c d*) (λ*u*. *f* (*x'*, *u*) − *f* (*x*, *u*))) < *e*

```
    apply (rule-tac x=dd in exI)
    using dd e2-gt assms x
    apply clarify
    apply (rule le-less-trans [OF - e2-less])
    apply (rule integrable-bound)
    apply (auto intro: integrable-diff continuous-on-imp-integrable-on-Pair1)
    done
 } note * = this
 show ?thesis
  apply (rule integrable-continuous)
 apply (simp add: * continuous-on-iff dist-norm integral-diff [symmetric] continuous-on-imp-integrable-on-Pa
[OF assms])
  done
qed

lemma norm-diff2: ⟦y = y1 + y2; x = x1 + x2; e = e1 + e2; norm(y1 - x1)
≤ e1; norm(y2 - x2) ≤ e2⟧
        ⟹ norm(y - x) ≤ e
using norm-triangle-mono [of y1 - x1 e1 y2 - x2 e2]
 by (simp add: add-diff-add)

lemma integral-split:
 fixes f :: 'a::euclidean-space ⇒ 'b::{real-normed-vector,complete-space}
 assumes f: f integrable-on (cbox a b)
    and k: k ∈ Basis
 shows integral (cbox a b) f =
        integral (cbox a b ∩ {x. x•k ≤ c}) f +
        integral (cbox a b ∩ {x. x•k ≥ c}) f
 apply (rule integral-unique [OF has-integral-split [where c=c]])
 using k f
 apply (auto simp: has-integral-integral [symmetric])
 done

lemma integral-swap-operative:
 fixes f :: ('a::euclidean-space * 'b::euclidean-space) ⇒ 'c::banach
 assumes f: continuous-on s f and e: 0 < e
  shows operative(op ∧)
        (λk. ∀ a b c d.
            cbox (a,c) (b,d) ⊆ k ∧ cbox (a,c) (b,d) ⊆ s
            ⟶ norm(integral (cbox (a,c) (b,d)) f -
                    integral (cbox a b) (λx. integral (cbox c d) (λy. f((x,y)))))
               ≤ e * content (cbox (a,c) (b,d)))
proof (auto simp: operative-def)
 fix a::'a and c::'b and b::'a and d::'b and u::'a and v::'a and w::'b and z::'b
 assume c0: content (cbox (a, c) (b, d)) = 0
   and cb1: cbox (u, w) (v, z) ⊆ cbox (a, c) (b, d)
   and cb2: cbox (u, w) (v, z) ⊆ s
 have c0': content (cbox (u, w) (v, z)) = 0
  by (fact content-0-subset [OF c0 cb1])
```

show *norm (integral (cbox (u,w) (v,z)) f − integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))))*
≤ e ∗ content (cbox (u,w) (v,z))
using *content-cbox-pair-eq0-D [OF c0']*
by (*force simp add*: *c0'*)
next
fix *a*::'*a* **and** *c*::'*b* **and** *b*::'*a* **and** *d*::'*b*
and *M*::*real* **and** *i*::'*a* **and** *j*::'*b*
and *u*::'*a* **and** *v*::'*a* **and** *w*::'*b* **and** *z*::'*b*
assume *ij*: *(i,j) ∈ Basis*
and *n1*: *∀ a' b' c' d'.*
cbox (a',c') (b',d') ⊆ cbox (a,c) (b,d) ∧
cbox (a',c') (b',d') ⊆ {x. x · (i,j) ≤ M} ∧ cbox (a',c') (b',d') ⊆ s ⟶
norm (integral (cbox (a',c') (b',d')) f − integral (cbox a' b') (λx. integral (cbox c' d') (λy. f (x,y))))
≤ e ∗ content (cbox (a',c') (b',d'))
and *n2*: *∀ a' b' c' d'.*
cbox (a',c') (b',d') ⊆ cbox (a,c) (b,d) ∧
cbox (a',c') (b',d') ⊆ {x. M ≤ x · (i,j)} ∧ cbox (a',c') (b',d') ⊆ s ⟶
norm (integral (cbox (a',c') (b',d')) f − integral (cbox a' b') (λx. integral (cbox c' d') (λy. f (x,y))))
≤ e ∗ content (cbox (a',c') (b',d'))
and *subs*: *cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d) cbox (u,w) (v,z) ⊆ s*
have *fcont*: *continuous-on (cbox (u, w) (v, z)) f*
using *assms(1) continuous-on-subset subs(2)* **by** *blast*
then have *fint*: *f integrable-on cbox (u, w) (v, z)*
by (*metis integrable-continuous*)
consider *i ∈ Basis j=0 | j ∈ Basis i=0* **using** *ij*
by (*auto simp*: *Euclidean-Space.Basis-prod-def*)
then show *norm (integral (cbox (u,w) (v,z)) f − integral (cbox u v) (λx. integral (cbox w z) (λy. f (x,y))))*
≤ e ∗ content (cbox (u,w) (v,z)) (**is** *?normle*)
proof *cases*
case *1*
then have *e*: *e ∗ content (cbox (u, w) (v, z)) =*
e ∗ (content (cbox u v ∩ {x. x · i ≤ M}) ∗ content (cbox w z)) +
e ∗ (content (cbox u v ∩ {x. M ≤ x · i}) ∗ content (cbox w z))
by (*simp add*: *content-split* [**where** *c=M*] *content-Pair algebra-simps*)
have ∗: *integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =*
integral (cbox u v ∩ {x. x · i ≤ M}) (λx. integral (cbox w z) (λy. f (x, y))) +
integral (cbox u v ∩ {x. M ≤ x · i}) (λx. integral (cbox w z) (λy. f (x, y)))
using *1 f subs integral-integrable-2dim continuous-on-subset*
by (*blast intro*: *integral-split*)
show *?normle*
apply (*rule norm-diff2* [*OF integral-split* [**where** *c=M*, *OF fint ij*] ∗ *e*])

```
      using 1 subs
      apply (simp-all add: cbox-Pair-eq setcomp-dot1 [of λu. M≤u] setcomp-dot1
[of λu. u≤M] Sigma-Int-Paircomp1)
      apply (simp-all add: interval-split ij)
      apply (simp-all add: cbox-Pair-eq [symmetric] content-Pair [symmetric])
      apply (force simp add: interval-split [symmetric] intro!: n1 [rule-format])
      apply (force simp add: interval-split [symmetric] intro!: n2 [rule-format])
      done
  next
    case 2
    then have e: e * content (cbox (u, w) (v, z)) =
                  e * (content (cbox u v) * content (cbox w z ∩ {x. x · j ≤ M})) +
                  e * (content (cbox u v) * content (cbox w z ∩ {x. M ≤ x · j}))
      by (simp add: content-split [where c=M] content-Pair algebra-simps)
    have (λx. integral (cbox w z ∩ {x. x · j ≤ M}) (λy. f (x, y))) integrable-on
cbox u v
           (λx. integral (cbox w z ∩ {x. M ≤ x · j}) (λy. f (x, y))) integrable-on
cbox u v
      using 2 subs
      apply (simp-all add: interval-split)
      apply (rule-tac [!] integral-integrable-2dim [OF continuous-on-subset [OF f]])
      apply (auto simp: cbox-Pair-eq interval-split [symmetric])
      done
    with 2 have *: integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =
                   integral (cbox u v) (λx. integral (cbox w z ∩ {x. x · j ≤ M}) (λy.
f (x, y))) +
                   integral (cbox u v) (λx. integral (cbox w z ∩ {x. M ≤ x · j}) (λy.
f (x, y)))
      by (simp add: integral-add [symmetric] integral-split [symmetric]
            continuous-on-imp-integrable-on-Pair1 [OF fcont] cong: integral-cong)
    show ?normle
      apply (rule norm-diff2 [OF integral-split [where c=M, OF fint ij] * e])
      using 2 subs
      apply (simp-all add: cbox-Pair-eq setcomp-dot2 [of λu. M≤u] setcomp-dot2
[of λu. u≤M] Sigma-Int-Paircomp2)
      apply (simp-all add: interval-split ij)
      apply (simp-all add: cbox-Pair-eq [symmetric] content-Pair [symmetric])
      apply (force simp add: interval-split [symmetric] intro!: n1 [rule-format])
      apply (force simp add: interval-split [symmetric] intro!: n2 [rule-format])
      done
  qed
qed

lemma integral-Pair-const:
    integral (cbox (a,c) (b,d)) (λx. k) =
     integral (cbox a b) (λx. integral (cbox c d) (λy. k))
  by (simp add: content-Pair)

lemma norm-minus2: norm (x1−x2, y1−y2) = norm (x2−x1, y2−y1)
```

```
  by (simp add: norm-minus-eqI)

lemma integral-prod-continuous:
  fixes f :: ('a::euclidean-space * 'b::euclidean-space) ⇒ 'c::banach
  assumes continuous-on (cbox (a,c) (b,d)) f (is continuous-on ?CBOX f)
    shows integral (cbox (a,c) (b,d)) f = integral (cbox a b) (λx. integral (cbox c
d) (λy. f(x,y)))
proof (cases content ?CBOX = 0)
  case True
  then show ?thesis
    by (auto simp: content-Pair)
next
  case False
  then have cbp: content ?CBOX > 0
    using content-lt-nz by blast
  have norm (integral ?CBOX f − integral (cbox a b) (λx. integral (cbox c d) (λy.
f (x,y)))) = 0
  proof (rule dense-eq0-I, simp)
    fix e::real  assume 0 < e
    with cbp have e': 0 < e / content ?CBOX
      by simp
    have f-int-acbd: f integrable-on cbox (a,c) (b,d)
      by (rule integrable-continuous [OF assms])
    { fix p
      assume p: p division-of cbox (a,c) (b,d)
       note opd1 = operative-division-and [OF integral-swap-operative [OF assms
e'], THEN iffD1,
                        THEN spec, THEN spec, THEN spec, THEN spec, of p (a,c)
(b,d) a c b d]
      have (⋀t u v w z.
            ⟦t ∈ p; cbox (u,w) (v,z) ⊆ t;  cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d)⟧ ⟹
             norm (integral (cbox (u,w) (v,z)) f − integral (cbox u v) (λx. integral
(cbox w z) (λy. f (x,y))))
             ≤ e * content (cbox (u,w) (v,z)) / content?CBOX)
           ⟹
           norm (integral ?CBOX f − integral (cbox a b) (λx. integral (cbox c d)
(λy. f (x,y)))) ≤ e
        using opd1 [OF p] False  by simp
    } note op-acbd = this
    { fix k::real and p and u::'a and v w and z::'b and t1 t2 l
      assume k: 0 < k
        and nf: ⋀x y u v.
                ⟦x ∈ cbox a b; y ∈ cbox c d; u ∈ cbox a b; v∈cbox c d; norm (u−x,
v−y) < k⟧
                ⟹ norm (f(u,v) − f(x,y)) < e / (2 * (content ?CBOX))
        and p-acbd: p tagged-division-of cbox (a,c) (b,d)
        and fine: (λx. ball x k) fine p  ((t1,t2), l) ∈ p
        and uwvz-sub: cbox (u,w) (v,z) ⊆ l cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d)
      have t: t1 ∈ cbox a b t2 ∈ cbox c d
```

by (*meson fine p-acbd cbox-Pair-iff tag-in-interval*)+
have *ls*: $l \subseteq ball\ (t1,t2)\ k$
using *fine* **by** (*simp add*: *fine-def Ball-def*)
{ fix *x1 x2*
assume *xuvwz*: *x1* $\in$ *cbox u v x2* $\in$ *cbox w z*
then have *x*: *x1* $\in$ *cbox a b x2* $\in$ *cbox c d*
using *uwvz-sub* **by** *auto*
have *norm* $(x1 - t1,\ x2 - t2) < k$
using *xuvwz ls uwvz-sub* **unfolding** *ball-def*
by (*force simp add*: *cbox-Pair-eq dist-norm norm-minus2*)
then have *norm* $(f\ (x1,x2) - f\ (t1,t2)) \leq e$ / *content ?CBOX* / *2*
using *nf* [*OF t x*] **by** *simp*
} note $nf' = this$
have *f-int-uwvz*: *f integrable-on cbox* (u,w) (v,z)
using *f-int-acbd uwvz-sub integrable-on-subcbox* **by** *blast*
have *f-int-uv*: $\bigwedge x.\ x \in cbox\ u\ v \Longrightarrow (\lambda y.\ f\ (x,y))$ *integrable-on cbox w z*
using *assms continuous-on-subset uwvz-sub*
by (*blast intro*: *continuous-on-imp-integrable-on-Pair1*)
have *1*: *norm* (*integral* (*cbox* (u,w) (v,z)) *f* $-$ *integral* (*cbox* (u,w) (v,z)) $(\lambda x.\ f\ (t1,t2))$)
$\leq e *$ *content* (*cbox* (u,w) (v,z)) / *content ?CBOX* / *2*
apply (*simp only*: *integral-diff* [*symmetric*] *f-int-uwvz integrable-const*)
apply (*rule order-trans* [*OF integrable-bound* [*of e* / *content ?CBOX* / *2*]])
using *cbp* e' nf'
apply (*auto simp*: *integrable-diff f-int-uwvz integrable-const*)
done
have *int-integrable*: $(\lambda x.\ integral\ (cbox\ w\ z)\ (\lambda y.\ f\ (x,\ y)))$ *integrable-on cbox u v*
using *assms integral-integrable-2dim continuous-on-subset uwvz-sub*(*2*) **by** *blast*
have *normint-wz*:
$\bigwedge x.\ x \in cbox\ u\ v \Longrightarrow$
norm (*integral* (*cbox w z*) $(\lambda y.\ f\ (x,\ y))$ $-$ *integral* (*cbox w z*) $(\lambda y.\ f\ (t1,\ t2))$)
$\leq e *$ *content* (*cbox w z*) / *content* (*cbox* $(a,\ c)$ $(b,\ d)$) / *2*
apply (*simp only*: *integral-diff* [*symmetric*] *f-int-uv integrable-const*)
apply (*rule order-trans* [*OF integrable-bound* [*of e* / *content ?CBOX* / *2*]])
using *cbp* e' nf'
apply (*auto simp*: *integrable-diff f-int-uv integrable-const*)
done
have *norm* (*integral* (*cbox u v*)
$(\lambda x.\ integral\ (cbox\ w\ z)\ (\lambda y.\ f\ (x,y)) - integral\ (cbox\ w\ z)\ (\lambda y.\ f\ (t1,t2))))$
$\leq e *$ *content* (*cbox w z*) / *content ?CBOX* / *2* $*$ *content* (*cbox u v*)
apply (*rule integrable-bound* [*OF - - normint-wz*])
using *cbp* e'
apply (*auto simp*: *divide-simps content-pos-le integrable-diff int-integrable integrable-const*)
done

```
    also have ... ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX / 2
      by (simp add: content-Pair divide-simps)
     finally have 2: norm (integral (cbox u v) (λx. integral (cbox w z) (λy. f
(x,y))) −
                    integral (cbox u v) (λx. integral (cbox w z) (λy. f (t1,t2))))
                ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX / 2
      by (simp only: integral-diff [symmetric] int-integrable integrable-const)
     have norm-xx: ⟦x′ = y′; norm(x − x′) ≤ e/2; norm(y − y′) ≤ e/2⟧ ⟹
norm(x − y) ≤ e for x::'c and y x′ y′ e
      using norm-triangle-mono [of x−y′ e/2 y′−y e/2] real-sum-of-halves
      by (simp add: norm-minus-commute)
    have norm (integral (cbox (u,w) (v,z)) f − integral (cbox u v) (λx. integral
(cbox w z) (λy. f (x,y))))
          ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX
      by (rule norm-xx [OF integral-Pair-const 1 2])
  } note * = this
  show norm (integral ?CBOX f − integral (cbox a b) (λx. integral (cbox c d)
(λy. f (x,y)))) ≤ e
    using compact-uniformly-continuous [OF assms compact-cbox]
    apply (simp add: uniformly-continuous-on-def dist-norm)
    apply (drule-tac x=e / 2 / content?CBOX in spec)
    using cbp ‹0 < e›
    apply (auto simp: zero-less-mult-iff)
    apply (rename-tac k)
   apply (rule-tac e1=k in fine-division-exists [OF gauge-ball, where a = (a,c)
and b = (b,d)])
    apply assumption
    apply (rule op-acbd)
    apply (erule division-of-tagged-division)
    using *
    apply auto
    done
 qed
 then show ?thesis
  by simp
qed

lemma swap-continuous:
 assumes continuous-on (cbox (a,c) (b,d)) (λ(x,y). f x y)
  shows continuous-on (cbox (c,a) (d,b)) (λ(x, y). f y x)
proof −
 have (λ(x, y). f y x) = (λ(x, y). f x y) ∘ prod.swap
  by auto
 then show ?thesis
  apply (rule ssubst)
  apply (rule continuous-on-compose)
  apply (simp add: split-def)
  apply (rule continuous-intros | simp add: assms)+
  done
```

qed

lemma *integral-swap-2dim*:
fixes *f* :: [*'a*::*euclidean-space*, *'b*::*euclidean-space*] ⇒ *'c*::*banach*
assumes *continuous-on* (*cbox* (*a*,*c*) (*b*,*d*)) (λ(*x*,*y*). *f x y*)
shows *integral* (*cbox* (*a*, *c*) (*b*, *d*)) (λ(*x*, *y*). *f x y*) = *integral* (*cbox* (*c*, *a*) (*d*, *b*)) (λ(*x*, *y*). *f y x*)
proof −
have ((λ(*x*, *y*). *f x y*) *has-integral integral* (*cbox* (*c*, *a*) (*d*, *b*)) (λ(*x*, *y*). *f y x*)) (*prod.swap* ‘ (*cbox* (*c*, *a*) (*d*, *b*)))
apply (*rule has-integral-twiddle* [*of 1 prod.swap prod.swap* λ(*x*,*y*). *f y x integral* (*cbox* (*c*, *a*) (*d*, *b*)) (λ(*x*, *y*). *f y x*), *simplified*])
apply (*auto simp*: *isCont-swap content-Pair has-integral-integral* [*symmetric*] *integrable-continuous swap-continuous assms*)
done
then show *?thesis*
by *force*
qed

theorem *integral-swap-continuous*:
fixes *f* :: [*'a*::*euclidean-space*, *'b*::*euclidean-space*] ⇒ *'c*::*banach*
assumes *continuous-on* (*cbox* (*a*,*c*) (*b*,*d*)) (λ(*x*,*y*). *f x y*)
shows *integral* (*cbox a b*) (λ*x*. *integral* (*cbox c d*) (*f x*)) =
integral (*cbox c d*) (λ*y*. *integral* (*cbox a b*) (λ*x*. *f x y*))
proof −
have *integral* (*cbox a b*) (λ*x*. *integral* (*cbox c d*) (*f x*)) = *integral* (*cbox* (*a*, *c*) (*b*, *d*)) (λ(*x*, *y*). *f x y*)
using *integral-prod-continuous* [*OF assms*] **by** *auto*
also have ... = *integral* (*cbox* (*c*, *a*) (*d*, *b*)) (λ(*x*, *y*). *f y x*)
by (*rule integral-swap-2dim* [*OF assms*])
also have ... = *integral* (*cbox c d*) (λ*y*. *integral* (*cbox a b*) (λ*x*. *f x y*))
using *integral-prod-continuous* [*OF swap-continuous*] *assms*
by *auto*
finally show *?thesis* .
qed

end

36 Instantiates the finite Cartesian product of Euclidean spaces as a Euclidean space.

theory *Cartesian-Euclidean-Space*
imports *Finite-Cartesian-Product Integration*
begin

lemma *delta-mult-idempotent*:
(*if k*=*a then 1 else* (*0*::*'a*::*semiring-1*)) ∗ (*if k*=*a then 1 else 0*) = (*if k*=*a then 1 else 0*)

by (*cases k=a*) *auto*

lemma *setsum-UNIV-sum*:
fixes $g :: 'a::finite + 'b::finite \Rightarrow$ -
shows $(\sum x{\in}UNIV.\ g\ x) = (\sum x{\in}UNIV.\ g\ (Inl\ x)) + (\sum x{\in}UNIV.\ g\ (Inr\ x))$
apply (*subst UNIV-Plus-UNIV* [*symmetric*])
apply (*subst setsum.Plus*)
apply *simp-all*
done

lemma *setsum-mult-product*:
$setsum\ h\ \{..<A * B :: nat\} = (\sum i{\in}\{..<A\}.\ \sum j{\in}\{..<B\}.\ h\ (j + i * B))$
unfolding *setsum-nat-group*[*of h B A*, *unfolded atLeast0LessThan*, *symmetric*]
proof (*rule setsum.cong*, *simp*, *rule setsum.reindex-cong*)
fix *i*
show *inj-on* $(\lambda j.\ j + i * B)\ \{..<B\}$ **by** (*auto intro*!: *inj-onI*)
show $\{i * B..<i * B + B\} = (\lambda j.\ j + i * B)\ `\ \{..<B\}$
proof *safe*
fix *j* **assume** $j \in \{i * B..<i * B + B\}$
then show $j \in (\lambda j.\ j + i * B)\ `\ \{..<B\}$
by (*auto intro*!: *image-eqI*[*of - - j − i * B*])
qed *simp*
qed *simp*

36.1 Basic componentwise operations on vectors.

instantiation *vec* :: (*times*, *finite*) *times*
begin

definition $op * \equiv (\lambda\ x\ y.\ \ (\chi\ i.\ (x\$i) * (y\$i)))$
instance ..

end

instantiation *vec* :: (*one*, *finite*) *one*
begin

definition $1 \equiv (\chi\ i.\ 1)$
instance ..

end

instantiation *vec* :: (*ord*, *finite*) *ord*
begin

definition $x \leq y \longleftrightarrow (\forall i.\ x\$i \leq y\$i)$
definition $x < (y::'a\hat{}'b) \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
instance ..

end

The ordering on one-dimensional vectors is linear.

class *cart-one* =
  assumes *UNIV-one*: *card* (*UNIV* :: *'a set*) = *Suc 0*
begin

subclass *finite*
proof
  from *UNIV-one* **show** *finite* (*UNIV* :: *'a set*)
    by (*auto intro*!: *card-ge-0-finite*)
qed

end

instance *vec*:: (*order*, *finite*) *order*
  by *standard* (*auto simp*: *less-eq-vec-def less-vec-def vec-eq-iff*
      intro: *order.trans order.antisym order.strict-implies-order*)

instance *vec* :: (*linorder*, *cart-one*) *linorder*
proof
  obtain *a* :: *'b* **where** *all*: $\bigwedge P.\ (\forall\, i.\ P\ i) \longleftrightarrow P\ a$
  proof −
    have *card* (*UNIV* :: *'b set*) = *Suc 0* **by** (*rule UNIV-one*)
    then obtain *b* :: *'b* **where** *UNIV* = {*b*} **by** (*auto iff*: *card-Suc-eq*)
    then have $\bigwedge P.\ (\forall\, i{\in}UNIV.\ P\ i) \longleftrightarrow P\ b$ **by** *auto*
    then show *thesis* **by** (*auto intro*: *that*)
  qed
  fix *x y* :: *'a^'b::cart-one*
  note [*simp*] = *less-eq-vec-def less-vec-def all vec-eq-iff field-simps*
  show $x \leq y \vee y \leq x$ **by** *auto*
qed

Constant Vectors

definition *vec x* = (χ *i*. *x*)

lemma *interval-cbox-cart*: {*a*::*real^'n..b*} = *cbox a b*
  by (*auto simp add*: *less-eq-vec-def mem-box Basis-vec-def inner-axis*)

Also the scalar-vector multiplication.

definition *vector-scalar-mult*:: *'a*::*times* ⇒ *'a ^ 'n* ⇒ *'a ^ 'n* (**infixl** ∗*s 70*)
  where *c* ∗*s x* = (χ *i*. *c* ∗ (*x*$*i*))

36.2 A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space.

lemma *setsum-cong-aux*:
  ($\bigwedge x.\ x \in A \Longrightarrow f\ x = g\ x$) ⟹ *setsum f A* = *setsum g A*
  by (*auto intro*: *setsum.cong*)

hide-fact (**open**) *setsum-cong-aux*

method-setup *vector* = ‹
let
  val ss1 =
    simpset-of (*put-simpset HOL-basic-ss* @{*context*}
      addsimps [@{*thm setsum.distrib*} *RS sym*,
      @{*thm setsum-subtractf*} *RS sym*, @{*thm setsum-right-distrib*},
      @{*thm setsum-left-distrib*}, @{*thm setsum-negf*} *RS sym*])
  val ss2 =
    simpset-of (@{*context*} *addsimps*
          [@{*thm plus-vec-def*}, @{*thm times-vec-def*},
           @{*thm minus-vec-def*}, @{*thm uminus-vec-def*},
           @{*thm one-vec-def*}, @{*thm zero-vec-def*}, @{*thm vec-def*},
           @{*thm scaleR-vec-def*},
           @{*thm vec-lambda-beta*}, @{*thm vector-scalar-mult-def*}])
  fun vector-arith-tac ctxt ths =
    simp-tac (*put-simpset ss1 ctxt*)
   THEN' (*fn i => resolve-tac ctxt* @{*thms Cartesian-Euclidean-Space.setsum-cong-aux*}
i
        ORELSE resolve-tac ctxt @{*thms setsum.neutral*} *i*
            ORELSE simp-tac (*put-simpset HOL-basic-ss ctxt addsimps* [@{*thm*
vec-eq-iff}]) *i*)
    (* *THEN' TRY o clarify-tac HOL-cs THEN'* (*TRY o rtac* @{*thm iffI*}) *)
    THEN' asm-full-simp-tac (*put-simpset ss2 ctxt addsimps ths*)
in
  Attrib.thms >> (*fn ths => fn ctxt => SIMPLE-METHOD'* (*vector-arith-tac*
ctxt ths))
end
› *lift trivial vector statements to real arith statements*

lemma *vec-0*[*simp*]: *vec 0 = 0* **by** *vector*
lemma *vec-1*[*simp*]: *vec 1 = 1* **by** *vector*

lemma *vec-inj*[*simp*]: *vec x = vec y* ⟷ *x = y* **by** *vector*

lemma *vec-in-image-vec*: *vec x* ∈ (*vec* ‘ *S*) ⟷ *x* ∈ *S* **by** *auto*

lemma *vec-add*: *vec*(*x* + *y*) = *vec x* + *vec y* **by** *vector*
lemma *vec-sub*: *vec*(*x* − *y*) = *vec x* − *vec y* **by** *vector*
lemma *vec-cmul*: *vec*(*c* ∗ *x*) = *c* ∗*s vec x* **by** *vector*
lemma *vec-neg*: *vec*(− *x*) = − *vec x* **by** *vector*

lemma *vec-setsum*:
  assumes *finite S*
  shows *vec*(*setsum f S*) = *setsum* (*vec* ∘ *f*) *S*
  using *assms*
proof *induct*

case *empty*
then show *?case* **by** *simp*
next
case *insert*
then show *?case* **by** (*auto simp add*: *vec-add*)
qed

Obvious "component-pushing".

lemma *vec-component* [*simp*]: *vec x $ i = x*
by *vector*

lemma *vector-mult-component* [*simp*]: *(x * y)$i = x$i * y$i*
by *vector*

lemma *vector-smult-component* [*simp*]: *(c *s y)$i = c * (y$i)*
by *vector*

lemma *cond-component*: *(if b then x else y)$i = (if b then x$i else y$i)* **by** *vector*

lemmas *vector-component =*
vec-component vector-add-component vector-mult-component
vector-smult-component vector-minus-component vector-uminus-component
vector-scaleR-component cond-component

36.3 Some frequently useful arithmetic lemmas over vectors.

instance *vec* :: (*semigroup-mult*, *finite*) *semigroup-mult*
by *standard* (*vector mult.assoc*)

instance *vec* :: (*monoid-mult*, *finite*) *monoid-mult*
by *standard vector+*

instance *vec* :: (*ab-semigroup-mult*, *finite*) *ab-semigroup-mult*
by *standard* (*vector mult.commute*)

instance *vec* :: (*comm-monoid-mult*, *finite*) *comm-monoid-mult*
by *standard vector*

instance *vec* :: (*semiring*, *finite*) *semiring*
by *standard* (*vector field-simps*)+

instance *vec* :: (*semiring-0*, *finite*) *semiring-0*
by *standard* (*vector field-simps*)+
instance *vec* :: (*semiring-1*, *finite*) *semiring-1*
by *standard vector*
instance *vec* :: (*comm-semiring*, *finite*) *comm-semiring*
by *standard* (*vector field-simps*)+

instance *vec* :: (*comm-semiring-0*, *finite*) *comm-semiring-0* **..**

instance *vec* :: (*cancel-comm-monoid-add*, *finite*) *cancel-comm-monoid-add* **..**
instance *vec* :: (*semiring-0-cancel*, *finite*) *semiring-0-cancel* **..**
instance *vec* :: (*comm-semiring-0-cancel*, *finite*) *comm-semiring-0-cancel* **..**
instance *vec* :: (*ring*, *finite*) *ring* **..**
instance *vec* :: (*semiring-1-cancel*, *finite*) *semiring-1-cancel* **..**
instance *vec* :: (*comm-semiring-1*, *finite*) *comm-semiring-1* **..**

instance *vec* :: (*ring-1*, *finite*) *ring-1* **..**

instance *vec* :: (*real-algebra*, *finite*) *real-algebra*
by *standard* (*simp-all add*: *vec-eq-iff*)

instance *vec* :: (*real-algebra-1*, *finite*) *real-algebra-1* **..**

lemma *of-nat-index*: (*of-nat n* :: *'a*::*semiring-1* ^*'n*)$*i* = *of-nat n*
proof (*induct n*)
case *0*
then show *?case* **by** *vector*
next
case *Suc*
then show *?case* **by** *vector*
qed

lemma *one-index* [*simp*]: (*1* :: *'a* :: *one* ^ *'n*) $ *i* = *1*
by *vector*

lemma *neg-one-index* [*simp*]: (− *1* :: *'a* :: {*one*, *uminus*} ^ *'n*) $ *i* = − *1*
by *vector*

instance *vec* :: (*semiring-char-0*, *finite*) *semiring-char-0*
proof
fix *m n* :: *nat*
show *inj* (*of-nat* :: *nat* ⇒ *'a* ^ *'b*)
by (*auto intro*!: *injI simp add*: *vec-eq-iff of-nat-index*)
qed

instance *vec* :: (*numeral*, *finite*) *numeral* **..**
instance *vec* :: (*semiring-numeral*, *finite*) *semiring-numeral* **..**

lemma *numeral-index* [*simp*]: *numeral w* $ *i* = *numeral w*
by (*induct w*) (*simp-all only*: *numeral.simps vector-add-component one-index*)

lemma *neg-numeral-index* [*simp*]: − *numeral w* $ *i* = − *numeral w*
by (*simp only*: *vector-uminus-component numeral-index*)

instance *vec* :: (*comm-ring-1*, *finite*) *comm-ring-1* **..**
instance *vec* :: (*ring-char-0*, *finite*) *ring-char-0* **..**

lemma *vector-smult-assoc*: *a* ∗*s* (*b* ∗*s x*) = ((*a*::*'a*::*semigroup-mult*) ∗ *b*) ∗*s x*

by (*vector mult.assoc*)
lemma *vector-sadd-rdistrib*: $((a::'a::semiring) + b) *s x = a *s x + b *s x$
by (*vector field-simps*)
lemma *vector-add-ldistrib*: $(c::'a::semiring) *s (x + y) = c *s x + c *s y$
by (*vector field-simps*)
lemma *vector-smult-lzero*[*simp*]: $(0::'a::mult\text{-}zero) *s x = 0$ **by** *vector*
lemma *vector-smult-lid*[*simp*]: $(1::'a::monoid\text{-}mult) *s x = x$ **by** *vector*
lemma *vector-ssub-ldistrib*: $(c::'a::ring) *s (x - y) = c *s x - c *s y$
by (*vector field-simps*)
lemma *vector-smult-rneg*: $(c::'a::ring) *s -x = -(c *s x)$ **by** *vector*
lemma *vector-smult-lneg*: $-(c::'a::ring) *s x = -(c *s x)$ **by** *vector*
lemma *vector-sneg-minus1*: $-x = (-1::'a::ring\text{-}1) *s x$ **by** *vector*
lemma *vector-smult-rzero*[*simp*]: $c *s 0 = (0::'a::mult\text{-}zero\ \hat{}\ 'n)$ **by** *vector*
lemma *vector-sub-rdistrib*: $((a::'a::ring) - b) *s x = a *s x - b *s x$
by (*vector field-simps*)

lemma *vec-eq*[*simp*]: $(vec\ m = vec\ n) \longleftrightarrow (m = n)$
by (*simp add*: *vec-eq-iff*)

lemma *norm-eq-0-imp*: $norm\ x = 0 ==> x = (0::real\ \hat{}\ 'n)$ **by** (*metis norm-eq-zero*)
lemma *vector-mul-eq-0*[*simp*]: $(a *s x = 0) \longleftrightarrow a = (0::'a::idom) \vee x = 0$
by *vector*
lemma *vector-mul-lcancel*[*simp*]: $a *s x = a *s y \longleftrightarrow a = (0::real) \vee x = y$
by (*metis eq-iff-diff-eq-0 vector-mul-eq-0 vector-ssub-ldistrib*)
lemma *vector-mul-rcancel*[*simp*]: $a *s x = b *s x \longleftrightarrow (a::real) = b \vee x = 0$
by (*metis eq-iff-diff-eq-0 vector-mul-eq-0 vector-sub-rdistrib*)
lemma *vector-mul-lcancel-imp*: $a \neq (0::real) ==>\ a *s x = a *s y ==> (x = y)$
by (*metis vector-mul-lcancel*)
lemma *vector-mul-rcancel-imp*: $x \neq 0 \Longrightarrow (a::real) *s x = b *s x ==> a = b$
by (*metis vector-mul-rcancel*)

lemma *component-le-norm-cart*: $|x\$i| <= norm\ x$
apply (*simp add*: *norm-vec-def*)
apply (*rule member-le-setL2*, *simp-all*)
done

lemma *norm-bound-component-le-cart*: $norm\ x <= e ==> |x\$i| <= e$
by (*metis component-le-norm-cart order-trans*)

lemma *norm-bound-component-lt-cart*: $norm\ x < e ==> |x\$i| < e$
by (*metis component-le-norm-cart le-less-trans*)

lemma *norm-le-l1-cart*: $norm\ x <= setsum(\lambda i.\ |x\$i|)\ UNIV$
by (*simp add*: *norm-vec-def setL2-le-setsum*)

lemma *scalar-mult-eq-scaleR*: $c *s x = c *_R x$
unfolding *scaleR-vec-def vector-scalar-mult-def* **by** *simp*

lemma *dist-mul*[*simp*]: *dist* (*c* ∗*s* *x*) (*c* ∗*s* *y*) = |*c*| ∗ *dist* *x* *y*
unfolding *dist-norm scalar-mult-eq-scaleR*
unfolding *scaleR-right-diff-distrib*[*symmetric*] **by** *simp*

lemma *setsum-component* [*simp*]:
fixes *f*:: *'a* ⇒ (*'b*::*comm-monoid-add*) ^*'n*
shows (*setsum f S*)$*i* = *setsum* (λ*x*. (*f x*)$*i*) *S*
proof (*cases finite S*)
case *True*
then show *?thesis* **by** *induct simp-all*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *setsum-eq*: *setsum f S* = (χ *i*. *setsum* (λ*x*. (*f x*)$*i*) *S*)
by (*simp add*: *vec-eq-iff*)

lemma *setsum-cmul*:
fixes *f*:: *'c* ⇒ (*'a*::*semiring-1*) ^*'n*
shows *setsum* (λ*x*. *c* ∗*s* *f x*) *S* = *c* ∗*s* *setsum f S*
by (*simp add*: *vec-eq-iff setsum-right-distrib*)

lemma *setsum-norm-allsubsets-bound-cart*:
fixes *f*:: *'a* ⇒ *real* ^*'n*
assumes *fP*: *finite P* **and** *fPs*: ⋀*Q*. *Q* ⊆ *P* ⟹ *norm* (*setsum f Q*) ≤ *e*
shows *setsum* (λ*x*. *norm* (*f x*)) *P* ≤ *2* ∗ *real CARD*(*'n*) ∗ *e*
using *setsum-norm-allsubsets-bound*[*OF assms*]
by *simp*

36.4 Matrix operations

Matrix notation. NB: an MxN matrix is of type ((*'a*, *'n*) *vec*, *'m*) *vec*, not ((*'a*, *'m*) *vec*, *'n*) *vec*

definition *matrix-matrix-mult* :: (*'a*::*semiring-1*) ^*'n*^*'m* ⇒ *'a* ^*'p*^*'n* ⇒ *'a* ^ *'p* ^*'m*
(**infixl** ∗∗ *70*)
where *m* ∗∗ *m'* == (χ *i j*. *setsum* (λ*k*. ((*m*$*i*)$*k*) ∗ ((*m'*$*k*)$*j*)) (*UNIV* :: *'n set*)) ::*'a* ^ *'p* ^*'m*

definition *matrix-vector-mult* :: (*'a*::*semiring-1*) ^*'n*^*'m* ⇒ *'a* ^*'n* ⇒ *'a* ^ *'m*
(**infixl** ∗*v* *70*)
where *m* ∗*v* *x* ≡ (χ *i*. *setsum* (λ*j*. ((*m*$*i*)$*j*) ∗ (*x*$*j*)) (*UNIV* ::*'n set*)) :: *'a*^*'m*

definition *vector-matrix-mult* :: *'a* ^ *'m* ⇒ (*'a*::*semiring-1*) ^*'n*^*'m* ⇒ *'a* ^*'n*
(**infixl** *v*∗ *70*)
where *v* *v*∗ *m* == (χ *j*. *setsum* (λ*i*. ((*m*$*i*)$*j*) ∗ (*v*$*i*)) (*UNIV* :: *'m set*)) :: *'a*^*'n*

definition (*mat*::'*a*::*zero* => '*a* ^'*n*^'*n*) *k* = (χ *i j*. *if i* = *j then k else* 0)
definition *transpose* **where**
(*transpose*::'*a*^'*n*^'*m* ⇒ '*a*^'*m*^'*n*) *A* = (χ *i j*. ((*A*$*j*)$*i*))
definition (*row*::'*m* => '*a* ^'*n*^'*m* ⇒ '*a* ^'*n*) *i A* = (χ *j*. ((*A*$*i*)$*j*))
definition (*column*::'*n* =>'*a*^'*n*^'*m* =>'*a*^'*m*) *j A* = (χ *i*. ((*A*$*i*)$*j*))
definition *rows*(*A*::'*a*^'*n*^'*m*) = { *row i A* | *i*. *i* ∈ (*UNIV* :: '*m set*)}
definition *columns*(*A*::'*a*^'*n*^'*m*) = { *column i A* | *i*. *i* ∈ (*UNIV* :: '*n set*)}

lemma *mat-0*[*simp*]: *mat* 0 = 0 **by** (*vector mat-def*)
lemma *matrix-add-ldistrib*: (*A* ** (*B* + *C*)) = (*A* ** *B*) + (*A* ** *C*)
by (*vector matrix-matrix-mult-def setsum.distrib*[*symmetric*] *field-simps*)

lemma *matrix-mul-lid*:
fixes *A* :: '*a*::*semiring-1* ^ '*m* ^ '*n*
shows *mat* 1 ** *A* = *A*
apply (*simp add*: *matrix-matrix-mult-def mat-def*)
apply *vector*
apply (*auto simp only*: *if-distrib cond-application-beta setsum.delta'*[*OF finite*]
mult-1-left mult-zero-left if-True UNIV-I)
done

lemma *matrix-mul-rid*:
fixes *A* :: '*a*::*semiring-1* ^ '*m* ^ '*n*
shows *A* ** *mat* 1 = *A*
apply (*simp add*: *matrix-matrix-mult-def mat-def*)
apply *vector*
apply (*auto simp only*: *if-distrib cond-application-beta setsum.delta*[*OF finite*]
mult-1-right mult-zero-right if-True UNIV-I cong: *if-cong*)
done

lemma *matrix-mul-assoc*: *A* ** (*B* ** *C*) = (*A* ** *B*) ** *C*
apply (*vector matrix-matrix-mult-def setsum-right-distrib setsum-left-distrib mult.assoc*)
apply (*subst setsum.commute*)
apply *simp*
done

lemma *matrix-vector-mul-assoc*: *A* *v* (*B* *v* *x*) = (*A* ** *B*) *v* *x*
apply (*vector matrix-matrix-mult-def matrix-vector-mult-def*
setsum-right-distrib setsum-left-distrib mult.assoc)
apply (*subst setsum.commute*)
apply *simp*
done

lemma *matrix-vector-mul-lid*: *mat* 1 *v* *x* = (*x*::'*a*::*semiring-1* ^ '*n*)
apply (*vector matrix-vector-mult-def mat-def*)
apply (*simp add*: *if-distrib cond-application-beta setsum.delta' cong del*: *if-weak-cong*)
done

lemma *matrix-transpose-mul*:
transpose(*A* ** *B*) = *transpose B* ** *transpose* (*A*::*'a*::*comm-semiring-1* ^ - ^ -)
by (*simp add*: *matrix-matrix-mult-def transpose-def vec-eq-iff mult.commute*)

lemma *matrix-eq*:
fixes *A B* :: *'a*::*semiring-1* ^ *'n* ^ *'m*
shows *A* = *B* ⟷ (∀ *x*. *A* *v *x* = *B* *v *x*) (**is** *?lhs* ⟷ *?rhs*)
apply *auto*
apply (*subst vec-eq-iff*)
apply *clarify*
apply (*clarsimp simp add*: *matrix-vector-mult-def if-distrib cond-application-beta vec-eq-iff cong del*: *if-weak-cong*)
apply (*erule-tac x*=*axis ia 1* **in** *allE*)
apply (*erule-tac x*=*i* **in** *allE*)
apply (*auto simp add*: *if-distrib cond-application-beta axis-def setsum.delta*[*OF finite*] *cong del*: *if-weak-cong*)
done

lemma *matrix-vector-mul-component*: ((*A*::*real* ^ - ^ -) *v *x*)$*k* = (*A*$*k*) · *x*
by (*simp add*: *matrix-vector-mult-def inner-vec-def*)

lemma *dot-lmul-matrix*: ((*x*::*real* ^ -) *v** *A*) · *y* = *x* · (*A* *v *y*)
apply (*simp add*: *inner-vec-def matrix-vector-mult-def vector-matrix-mult-def setsum-left-distrib setsum-right-distrib ac-simps*)
apply (*subst setsum.commute*)
apply *simp*
done

lemma *transpose-mat*: *transpose* (*mat n*) = *mat n*
by (*vector transpose-def mat-def*)

lemma *transpose-transpose*: *transpose*(*transpose A*) = *A*
by (*vector transpose-def*)

lemma *row-transpose*:
fixes *A*:: *'a*::*semiring-1* ^ - ^ -
shows *row i* (*transpose A*) = *column i A*
by (*simp add*: *row-def column-def transpose-def vec-eq-iff*)

lemma *column-transpose*:
fixes *A*:: *'a*::*semiring-1* ^ - ^ -
shows *column i* (*transpose A*) = *row i A*
by (*simp add*: *row-def column-def transpose-def vec-eq-iff*)

lemma *rows-transpose*: *rows*(*transpose* (*A*::*'a*::*semiring-1* ^ - ^ -)) = *columns A*
by (*auto simp add*: *rows-def columns-def row-transpose intro*: *set-eqI*)

lemma *columns-transpose*: *columns*(*transpose* (*A*::*'a*::*semiring-1* ^ - ^ -)) = *rows A*
by (*metis transpose-transpose rows-transpose*)

Two sometimes fruitful ways of looking at matrix-vector multiplication.

lemma *matrix-mult-dot*: $A *v x = (\chi\ i.\ A\$i \cdot x)$
by (*simp add*: *matrix-vector-mult-def inner-vec-def*)

lemma *matrix-mult-vsum*:
$(A::'a::comm\text{-}semiring\text{-}1\hat{}'n\hat{}'m) *v x = setsum\ (\lambda i.\ (x\$i) *s column\ i\ A)\ (UNIV::'n\ set)$
by (*simp add*: *matrix-vector-mult-def vec-eq-iff column-def mult.commute*)

lemma *vector-componentwise*:
$(x::'a::ring\text{-}1\hat{}'n) = (\chi\ j.\ \sum i{\in}UNIV.\ (x\$i) * (axis\ i\ 1 :: 'a\hat{}'n)\ \$\ j)$
by (*simp add*: *axis-def if-distrib setsum.If-cases vec-eq-iff*)

lemma *basis-expansion*: $setsum\ (\lambda i.\ (x\$i) *s\ axis\ i\ 1)\ UNIV = (x::('a::ring\text{-}1)\hat{}'n)$
by (*auto simp add*: *axis-def vec-eq-iff if-distrib setsum.If-cases cong del*: *if-weak-cong*)

lemma *linear-componentwise*:
fixes $f::$ *real* $\hat{}'m \Rightarrow$ *real* $\hat{}$ -
assumes *lf*: *linear f*
shows $(f\ x)\$j = setsum\ (\lambda i.\ (x\$i) * (f\ (axis\ i\ 1)\$j))\ (UNIV :: 'm\ set)$ (**is** *?lhs = ?rhs*)
proof −
let *?M* $= (UNIV :: 'm\ set)$
let *?N* $= (UNIV :: 'n\ set)$
have $?rhs = (setsum\ (\lambda i.(x\$i) *_R f\ (axis\ i\ 1)\ )\ ?M)\$j$
unfolding *setsum-component* **by** *simp*
then show *?thesis*
unfolding *linear-setsum-mul*[*OF lf*, *symmetric*]
unfolding *scalar-mult-eq-scaleR*[*symmetric*]
unfolding *basis-expansion*
by *simp*
qed

Inverse matrices (not necessarily square)

definition
$invertible(A::'a::semiring\text{-}1\hat{}'n\hat{}'m) \longleftrightarrow (\exists A'::'a\hat{}'m\hat{}'n.\ A ** A' = mat\ 1 \wedge A' ** A = mat\ 1)$

definition
$matrix\text{-}inv(A::'a::semiring\text{-}1\hat{}'n\hat{}'m) =$
$(SOME\ A'::'a\hat{}'m\hat{}'n.\ A ** A' = mat\ 1 \wedge A' ** A = mat\ 1)$

Correspondence between matrices and linear operators.

definition $matrix :: ('a::\{plus,times,\ one,\ zero\}\hat{}'m \Rightarrow 'a\ \hat{}\ 'n) \Rightarrow 'a\hat{}'m\hat{}'n$
where $matrix\ f = (\chi\ i\ j.\ (f(axis\ j\ 1))\$i)$

lemma *matrix-vector-mul-linear*: $linear(\lambda x.\ A *v (x::real\ \hat{}\ -))$
by (*simp add*: *linear-iff matrix-vector-mult-def vec-eq-iff*

field-simps setsum-right-distrib setsum.distrib)

lemma *matrix-works*:
assumes *lf*: *linear f*
shows *matrix f* $*v$ *x* = *f* (*x*::*real* ^ '*n*)
apply (*simp add*: *matrix-def matrix-vector-mult-def vec-eq-iff mult.commute*)
apply *clarify*
apply (*rule linear-componentwise*[*OF lf*, *symmetric*])
done

lemma *matrix-vector-mul*: *linear f* ==> *f* = (λ*x*. *matrix f* $*v* (*x*::*real* ^ '*n*))
by (*simp add*: *ext matrix-works*)

lemma *matrix-of-matrix-vector-mul*: *matrix*(λ*x*. *A* $*v* (*x* :: *real* ^ '*n*)) = *A*
by (*simp add*: *matrix-eq matrix-vector-mul-linear matrix-works*)

lemma *matrix-compose*:
assumes *lf*: *linear* (*f*::*real*^'*n* ⇒ *real*^'*m*)
and *lg*: *linear* (*g*::*real*^'*m* ⇒ *real*^-)
shows *matrix* (*g* ∘ *f*) = *matrix g* ** *matrix f*
using *lf lg linear-compose*[*OF lf lg*] *matrix-works*[*OF linear-compose*[*OF lf lg*]]
by (*simp add*: *matrix-eq matrix-works matrix-vector-mul-assoc*[*symmetric*] *o-def*)

lemma *matrix-vector-column*:
(*A*::'*a*::*comm-semiring-1*^'*n*^-) $*v* *x* = *setsum* (λ*i*. (*x*$*i*) $*s* ((*transpose A*)$*i*))
(*UNIV*:: '*n set*)
by (*simp add*: *matrix-vector-mult-def transpose-def vec-eq-iff mult.commute*)

lemma *adjoint-matrix*: *adjoint*(λ*x*. (*A*::*real*^'*n*^'*m*) $*v* *x*) = (λ*x*. *transpose A* $*v* *x*)
apply (*rule adjoint-unique*)
apply (*simp add*: *transpose-def inner-vec-def matrix-vector-mult-def*
setsum-left-distrib setsum-right-distrib)
apply (*subst setsum.commute*)
apply (*auto simp add*: *ac-simps*)
done

lemma *matrix-adjoint*: **assumes** *lf*: *linear* (*f* :: *real*^'*n* ⇒ *real* ^'*m*)
shows *matrix*(*adjoint f*) = *transpose*(*matrix f*)
apply (*subst matrix-vector-mul*[*OF lf*])
unfolding *adjoint-matrix matrix-of-matrix-vector-mul*
apply *rule*
done

36.5 lambda skolemization on cartesian products

lemma *lambda-skolem*: (∀ *i*. ∃ *x*. *P i x*) ⟷
(∃ *x*::'*a* ^ '*n*. ∀ *i*. *P i* (*x* $ *i*)) (**is** *?lhs* ⟷ *?rhs*)
proof −

```
  let ?S = (UNIV :: 'n set)
  { assume H: ?rhs
    then have ?lhs by auto }
  moreover
  { assume H: ?lhs
    then obtain f where f:∀ i. P i (f i) unfolding choice-iff by metis
    let ?x = (χ i. (f i)) :: 'a ^ 'n
    { fix i
      from f have P i (f i) by metis
      then have P i (?x $ i) by auto
    }
    hence ∀ i. P i (?x$i) by metis
    hence ?rhs by metis }
  ultimately show ?thesis by metis
qed

lemma vector-sub-project-orthogonal-cart: (b::real^'n) • (x - ((b • x) / (b • b)) *s
b) = 0
  unfolding inner-simps scalar-mult-eq-scaleR by auto

lemma left-invertible-transpose:
  (∃(B). B ** transpose (A) = mat (1::'a::comm-semiring-1)) ⟷ (∃(B). A **
B = mat 1)
  by (metis matrix-transpose-mul transpose-mat transpose-transpose)

lemma right-invertible-transpose:
  (∃(B). transpose (A) ** B = mat (1::'a::comm-semiring-1)) ⟷ (∃(B). B **
A = mat 1)
  by (metis matrix-transpose-mul transpose-mat transpose-transpose)

lemma matrix-left-invertible-injective:
  (∃B. (B::real^'m^'n) ** (A::real^'n^'m) = mat 1) ⟷ (∀x y. A *v x = A *v y
⟶ x = y)
proof −
  { fix B:: real^'m^'n and x y assume B: B ** A = mat 1 and xy: A *v x =
A*v y
    from xy have B*v (A *v x) = B *v (A*v y) by simp
    hence x = y
      unfolding matrix-vector-mul-assoc B matrix-vector-mul-lid . }
  moreover
  { assume A: ∀x y. A *v x = A *v y ⟶ x = y
    hence i: inj (op *v A) unfolding inj-on-def by auto
    from linear-injective-left-inverse[OF matrix-vector-mul-linear i]
    obtain g where g: linear g g ∘ op *v A = id by blast
    have matrix g ** A = mat 1
     unfolding matrix-eq matrix-vector-mul-lid matrix-vector-mul-assoc[symmetric]
matrix-works[OF g(1)]
      using g(2) by (simp add: fun-eq-iff)
    then have ∃B. (B::real ^'m^'n) ** A = mat 1 by blast }
```

```
  ultimately show ?thesis by blast
qed

lemma matrix-left-invertible-ker:
  (∃ B. (B::real ^'m^'n) ** (A::real^'n^'m) = mat 1) ⟷ (∀ x. A *v x = 0 ⟶ x
= 0)
  unfolding matrix-left-invertible-injective
  using linear-injective-0[OF matrix-vector-mul-linear, of A]
  by (simp add: inj-on-def)

lemma matrix-right-invertible-surjective:
  (∃ B. (A::real^'n^'m) ** (B::real^'m^'n) = mat 1) ⟷ surj (λx. A *v x)
proof −
  { fix B :: real ^'m^'n
    assume AB: A ** B = mat 1
    { fix x :: real ^ 'm
      have A *v (B *v x) = x
        by (simp add: matrix-vector-mul-lid matrix-vector-mul-assoc AB) }
    hence surj (op *v A) unfolding surj-def by metis }
  moreover
  { assume sf: surj (op *v A)
    from linear-surjective-right-inverse[OF matrix-vector-mul-linear sf]
    obtain g:: real ^'m ⇒ real ^'n where g: linear g op *v A ∘ g = id
      by blast

    have A ** (matrix g) = mat 1
      unfolding matrix-eq  matrix-vector-mul-lid
        matrix-vector-mul-assoc[symmetric] matrix-works[OF g(1)]
      using g(2) unfolding o-def fun-eq-iff id-def
      .
    hence ∃ B. A ** (B::real^'m^'n) = mat 1 by blast
  }
  ultimately show ?thesis unfolding surj-def by blast
qed

lemma matrix-left-invertible-independent-columns:
  fixes A :: real^'n^'m
  shows (∃ (B::real ^'m^'n). B ** A = mat 1) ⟷
     (∀ c. setsum (λi. c i *s column i A) (UNIV :: 'n set) = 0 ⟶ (∀ i. c i = 0))
    (is ?lhs ⟷ ?rhs)
proof −
  let ?U = UNIV :: 'n set
  { assume k: ∀ x. A *v x = 0 ⟶ x = 0
    { fix c i
      assume c: setsum (λi. c i *s column i A) ?U = 0 and i: i ∈ ?U
      let ?x = χ i. c i
      have th0:A *v ?x = 0
        using c
        unfolding matrix-mult-vsum vec-eq-iff
```

```
      by auto
    from k[rule-format, OF th0] i
    have c i = 0 by (vector vec-eq-iff)}
  hence ?rhs by blast }
 moreover
 { assume H: ?rhs
  { fix x assume x: A *v x = 0
    let ?c = λi. ((x$i ):: real)
    from H[rule-format, of ?c, unfolded matrix-mult-vsum[symmetric], OF x]
    have x = 0 by vector }
 }
 ultimately show ?thesis unfolding matrix-left-invertible-ker by blast
qed

lemma matrix-right-invertible-independent-rows:
 fixes A :: real^'n^'m
 shows (∃(B::real^'m^'n). A ** B = mat 1) ⟷
   (∀c. setsum (λi. c i *s row i A) (UNIV :: 'm set) = 0 ⟶ (∀i. c i = 0))
 unfolding left-invertible-transpose[symmetric]
   matrix-left-invertible-independent-columns
 by (simp add: column-transpose)

lemma matrix-right-invertible-span-columns:
 (∃(B::real ^'n^'m). (A::real ^'m^'n) ** B = mat 1) ⟷
   span (columns A) = UNIV (is ?lhs = ?rhs)
proof −
 let ?U = UNIV :: 'm set
 have fU: finite ?U by simp
 have lhseq: ?lhs ⟷ (∀y. ∃(x::real^'m). setsum (λi. (x$i) *s column i A) ?U
= y)
   unfolding matrix-right-invertible-surjective matrix-mult-vsum surj-def
   apply (subst eq-commute)
   apply rule
   done
 have rhseq: ?rhs ⟷ (∀x. x ∈ span (columns A)) by blast
 { assume h: ?lhs
  { fix x:: real ^'n
    from h[unfolded lhseq, rule-format, of x] obtain y :: real ^'m
      where y: setsum (λi. (y$i) *s column i A) ?U = x by blast
    have x ∈ span (columns A)
      unfolding y[symmetric]
      apply (rule span-setsum)
      apply clarify
      unfolding scalar-mult-eq-scaleR
      apply (rule span-mul)
      apply (rule span-superset)
      unfolding columns-def
      apply blast
      done
```

```
  }
  then have ?rhs unfolding rhseq by blast }
 moreover
 { assume h:?rhs
  let ?P = λ(y::real ^'n). ∃(x::real^'m). setsum (λi. (x$i) *s column i A) ?U =
y
  { fix y
    have ?P y
    proof (rule span-induct-alt[of ?P columns A, folded scalar-mult-eq-scaleR])
      show ∃x::real ^ 'm. setsum (λi. (x$i) *s column i A) ?U = 0
        by (rule exI[where x=0], simp)
    next
      fix c y1 y2
      assume y1: y1 ∈ columns A and y2: ?P y2
      from y1 obtain i where i: i ∈ ?U y1 = column i A
        unfolding columns-def by blast
      from y2 obtain x:: real ^'m where
        x: setsum (λi. (x$i) *s column i A) ?U = y2 by blast
      let ?x = (χ j. if j = i then c + (x$i) else (x$j))::real^'m
      show ?P (c*s y1 + y2)
        proof (rule exI[where x= ?x], vector, auto simp add: i x[symmetric]
if-distrib distrib-left cond-application-beta cong del: if-weak-cong)
        fix j
        have th: ∀ xa ∈ ?U. (if xa = i then (c + (x$i)) * ((column xa A)$j)
          else (x$xa) * ((column xa A$j))) = (if xa = i then c * ((column i A)$j)
else 0) + ((x$xa) * ((column xa A)$j))
          using i(1) by (simp add: field-simps)
        have setsum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
          else (x$xa) * ((column xa A$j))) ?U = setsum (λxa. (if xa = i then c
* ((column i A)$j) else 0) + ((x$xa) * ((column xa A)$j))) ?U
          apply (rule setsum.cong[OF refl])
          using th apply blast
          done
        also have ... = setsum (λxa. if xa = i then c * ((column i A)$j) else 0)
?U + setsum (λxa. ((x$xa) * ((column xa A)$j))) ?U
          by (simp add: setsum.distrib)
        also have ... = c * ((column i A)$j) + setsum (λxa. ((x$xa) * ((column
xa A)$j))) ?U
          unfolding setsum.delta[OF fU]
          using i(1) by simp
        finally show setsum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
          else (x$xa) * ((column xa A$j))) ?U = c * ((column i A)$j) + setsum
(λxa. ((x$xa) * ((column xa A)$j))) ?U .
      qed
    next
      show y ∈ span (columns A)
        unfolding h by blast
    qed
  }
```

```
    then have ?lhs unfolding lhseq ..
  }
  ultimately show ?thesis by blast
qed

lemma matrix-left-invertible-span-rows:
  (∃(B::real^'m^'n). B ** (A::real^'n^'m) = mat 1) ⟷ span (rows A) = UNIV
  unfolding right-invertible-transpose[symmetric]
  unfolding columns-transpose[symmetric]
  unfolding matrix-right-invertible-span-columns
  ..
```

The same result in terms of square matrices.

```
lemma matrix-left-right-inverse:
  fixes A A' :: real ^'n^'n
  shows A ** A' = mat 1 ⟷ A' ** A = mat 1
proof −
  { fix A A' :: real ^'n^'n
    assume AA': A ** A' = mat 1
    have sA: surj (op *v A)
      unfolding surj-def
      apply clarify
      apply (rule-tac x=(A' *v y) in exI)
      apply (simp add: matrix-vector-mul-assoc AA' matrix-vector-mul-lid)
      done
    from linear-surjective-isomorphism[OF matrix-vector-mul-linear sA]
    obtain f' :: real ^'n ⇒ real ^'n
      where f': linear f' ∀ x. f' (A *v x) = x ∀ x. A *v f' x = x by blast
    have th: matrix f' ** A = mat 1
      by (simp add: matrix-eq matrix-works[OF f'(1)]
       matrix-vector-mul-assoc[symmetric] matrix-vector-mul-lid f'(2)[rule-format])
    hence (matrix f' ** A) ** A' = mat 1 ** A' by simp
    hence matrix f' = A'
    by (simp add: matrix-mul-assoc[symmetric] AA' matrix-mul-rid matrix-mul-lid)
    hence matrix f' ** A = A' ** A by simp
    hence A' ** A = mat 1 by (simp add: th)
  }
  then show ?thesis by blast
qed
```

Considering an n-element vector as an n-by-1 or 1-by-n matrix.

```
definition rowvector v = (χ i j. (v$j))

definition columnvector v = (χ i j. (v$i))

lemma transpose-columnvector: transpose(columnvector v) = rowvector v
  by (simp add: transpose-def rowvector-def columnvector-def vec-eq-iff)

lemma transpose-rowvector: transpose(rowvector v) = columnvector v
```

by (*simp add*: *transpose-def columnvector-def rowvector-def vec-eq-iff*)

lemma *dot-rowvector-columnvector*: *columnvector* (*A* ∗*v v*) = *A* ∗∗ *columnvector v*
by (*vector columnvector-def matrix-matrix-mult-def matrix-vector-mult-def*)

lemma *dot-matrix-product*:
(*x*::*real*^′*n*) · *y* = (((*rowvector x* ::*real*^′*n*^*1*) ∗∗ (*columnvector y* :: *real*^*1*^′*n*))$*1*)$*1*
by (*vector matrix-matrix-mult-def rowvector-def columnvector-def inner-vec-def*)

lemma *dot-matrix-vector-mul*:
fixes *A B* :: *real* ^′*n* ^′*n* **and** *x y* :: *real* ^′*n*
shows (*A* ∗*v x*) · (*B* ∗*v y*) =
(((*rowvector x* :: *real*^′*n*^*1*) ∗∗ ((*transpose A* ∗∗ *B*) ∗∗ (*columnvector y* :: *real* ^*1*^′*n*)))$*1*)$*1*
unfolding *dot-matrix-product transpose-columnvector*[*symmetric*]
dot-rowvector-columnvector matrix-transpose-mul matrix-mul-assoc **..**

lemma *infnorm-cart*:*infnorm* (*x*::*real*^′*n*) = *Sup* {|*x*$*i*| |*i*. *i*∈*UNIV*}
by (*simp add*: *infnorm-def inner-axis Basis-vec-def*) (*metis* (*lifting*) *inner-axis real-inner-1-right*)

lemma *component-le-infnorm-cart*: |*x*$*i*| ≤ *infnorm* (*x*::*real*^′*n*)
using *Basis-le-infnorm*[*of axis i 1 x*]
by (*simp add*: *Basis-vec-def axis-eq-axis inner-axis*)

lemma *continuous-component*: *continuous F f* ⟹ *continuous F* (λ*x*. *f x* $ *i*)
unfolding *continuous-def* **by** (*rule tendsto-vec-nth*)

lemma *continuous-on-component*: *continuous-on s f* ⟹ *continuous-on s* (λ*x*. *f x* $ *i*)
unfolding *continuous-on-def* **by** (*fast intro*: *tendsto-vec-nth*)

lemma *closed-positive-orthant*: *closed* {*x*::*real*^′*n*. ∀ *i*. *0* ≤*x*$*i*}
by (*simp add*: *Collect-all-eq closed-INT closed-Collect-le*)

lemma *bounded-component-cart*: *bounded s* ⟹ *bounded* ((λ*x*. *x* $ *i*) ‘ *s*)
unfolding *bounded-def*
apply *clarify*
apply (*rule-tac x*=*x* $ *i* **in** *exI*)
apply (*rule-tac x*=*e* **in** *exI*)
apply *clarify*
apply (*rule order-trans* [*OF dist-vec-nth-le*], *simp*)
done

lemma *compact-lemma-cart*:
fixes *f* :: *nat* ⇒ ′*a*::*heine-borel* ^ ′*n*
assumes *f*: *bounded* (*range f*)

```
  shows ∃ l r. subseq r ∧
        (∀ e>0. eventually (λn. ∀ i∈d. dist (f (r n) $ i) (l $ i) < e) sequentially)
    (is ?th d)
proof −
  have ∀ d' ⊆ d. ?th d'
    by (rule compact-lemma-general[where unproj=vec-lambda])
      (auto intro!: f bounded-component-cart simp: vec-lambda-eta)
  then show ?th d by simp
qed

instance vec :: (heine-borel, finite) heine-borel
proof
  fix f :: nat ⇒ 'a ^ 'b
  assume f: bounded (range f)
  then obtain l r where r: subseq r
       and l: ∀ e>0. eventually (λn. ∀ i∈UNIV. dist (f (r n) $ i) (l $ i) < e)
sequentially
    using compact-lemma-cart [OF f] by blast
  let ?d = UNIV::'b set
  { fix e::real assume e>0
    hence 0 < e / (real-of-nat (card ?d))
     using zero-less-card-finite divide-pos-pos[of e, of real-of-nat (card ?d)] by auto
    with l have eventually (λn. ∀ i. dist (f (r n) $ i) (l $ i) < e / (real-of-nat
(card ?d))) sequentially
      by simp
    moreover
    { fix n
      assume n: ∀ i. dist (f (r n) $ i) (l $ i) < e / (real-of-nat (card ?d))
      have dist (f (r n)) l ≤ (∑ i∈?d. dist (f (r n) $ i) (l $ i))
        unfolding dist-vec-def using zero-le-dist by (rule setL2-le-setsum)
      also have ... < (∑ i∈?d. e / (real-of-nat (card ?d)))
        by (rule setsum-strict-mono) (simp-all add: n)
      finally have dist (f (r n)) l < e by simp
    }
    ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
      by (rule eventually-mono)
  }
  hence ((f ∘ r) ⟶ l) sequentially unfolding o-def tendsto-iff by simp
  with r show ∃ l r. subseq r ∧ ((f ∘ r) ⟶ l) sequentially by auto
qed

lemma interval-cart:
  fixes a :: real^'n
  shows box a b = {x::real^'n. ∀ i. a$i < x$i ∧ x$i < b$i}
    and cbox a b = {x::real^'n. ∀ i. a$i ≤ x$i ∧ x$i ≤ b$i}
  by (auto simp add: set-eq-iff less-vec-def less-eq-vec-def mem-box Basis-vec-def
inner-axis)

lemma mem-interval-cart:
```

fixes $a :: real\hat{}'n$
shows $x \in box\ a\ b \longleftrightarrow (\forall i.\ a\$i < x\$i \wedge x\$i < b\$i)$
and $x \in cbox\ a\ b \longleftrightarrow (\forall i.\ a\$i \leq x\$i \wedge x\$i \leq b\$i)$
using *interval-cart*[*of a b*] **by** (*auto simp add*: *set-eq-iff less-vec-def less-eq-vec-def*)

lemma *interval-eq-empty-cart*:
fixes $a :: real\hat{}'n$
shows ($box\ a\ b = \{\} \longleftrightarrow (\exists i.\ b\$i \leq a\$i)$) (**is** *?th1*)
and ($cbox\ a\ b = \{\} \longleftrightarrow (\exists i.\ b\$i < a\$i)$) (**is** *?th2*)
proof −
{ fix *i x* **assume** *as*:$b\$i \leq a\$i$ **and** *x*:$x \in box\ a\ b$
hence $a\ \$\ i < x\ \$\ i \wedge x\ \$\ i < b\ \$\ i$ **unfolding** *mem-interval-cart* **by** *auto*
hence $a\$i < b\$i$ **by** *auto*
hence *False* **using** *as* **by** *auto* **}**
moreover
{ assume *as*:$\forall i.\ \neg\ (b\$i \leq a\$i)$
let *?x* = $(1/2) *_R (a + b)$
{ fix *i*
have $a\$i < b\$i$ **using** *as*[*THEN spec*[**where** *x*=*i*]] **by** *auto*
hence $a\$i < ((1/2) *_R (a+b))\ \$\ i\ ((1/2) *_R (a+b))\ \$\ i < b\$i$
unfolding *vector-smult-component* **and** *vector-add-component*
by *auto* **}**
hence $box\ a\ b \neq \{\}$ **using** *mem-interval-cart*(*1*)[*of ?x a b*] **by** *auto* **}**
ultimately show *?th1* **by** *blast*

{ fix *i x* **assume** *as*:$b\$i < a\$i$ **and** *x*:$x \in cbox\ a\ b$
hence $a\ \$\ i \leq x\ \$\ i \wedge x\ \$\ i \leq b\ \$\ i$ **unfolding** *mem-interval-cart* **by** *auto*
hence $a\$i \leq b\$i$ **by** *auto*
hence *False* **using** *as* **by** *auto* **}**
moreover
{ assume *as*:$\forall i.\ \neg\ (b\$i < a\$i)$
let *?x* = $(1/2) *_R (a + b)$
{ fix *i*
have $a\$i \leq b\$i$ **using** *as*[*THEN spec*[**where** *x*=*i*]] **by** *auto*
hence $a\$i \leq ((1/2) *_R (a+b))\ \$\ i\ ((1/2) *_R (a+b))\ \$\ i \leq b\$i$
unfolding *vector-smult-component* **and** *vector-add-component*
by *auto* **}**
hence $cbox\ a\ b \neq \{\}$ **using** *mem-interval-cart*(*2*)[*of ?x a b*] **by** *auto* **}**
ultimately show *?th2* **by** *blast*
qed

lemma *interval-ne-empty-cart*:
fixes $a :: real\hat{}'n$
shows $cbox\ a\ b \neq \{\} \longleftrightarrow (\forall i.\ a\$i \leq b\$i)$
and $box\ a\ b \neq \{\} \longleftrightarrow (\forall i.\ a\$i < b\$i)$
unfolding *interval-eq-empty-cart*[*of a b*] **by** (*auto simp add*: *not-less not-le*)

lemma *subset-interval-imp-cart*:

fixes $a :: real\hat{}'n$
shows $(\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i) \Longrightarrow cbox\ c\ d \subseteq cbox\ a\ b$
and $(\forall i.\ a\$i < c\$i \wedge d\$i < b\$i) \Longrightarrow cbox\ c\ d \subseteq box\ a\ b$
and $(\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i) \Longrightarrow box\ c\ d \subseteq cbox\ a\ b$
and $(\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i) \Longrightarrow box\ c\ d \subseteq box\ a\ b$
unfolding *subset-eq*[*unfolded Ball-def*] **unfolding** *mem-interval-cart*
by (*auto intro*: *order-trans less-le-trans le-less-trans less-imp-le*)

lemma *interval-sing*:
fixes $a :: 'a::linorder\hat{}'n$
shows $\{a\ ..\ a\} = \{a\} \wedge \{a<..<a\} = \{\}$
apply (*auto simp add*: *set-eq-iff less-vec-def less-eq-vec-def vec-eq-iff*)
done

lemma *subset-interval-cart*:
fixes $a :: real\hat{}'n$
shows $cbox\ c\ d \subseteq cbox\ a\ b \longleftrightarrow (\forall i.\ c\$i \leq d\$i) \longrightarrow (\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i)$ (**is** *?th1*)
and $cbox\ c\ d \subseteq box\ a\ b \longleftrightarrow (\forall i.\ c\$i \leq d\$i) \longrightarrow (\forall i.\ a\$i < c\$i \wedge d\$i < b\$i)$ (**is** *?th2*)
and $box\ c\ d \subseteq cbox\ a\ b \longleftrightarrow (\forall i.\ c\$i < d\$i) \longrightarrow (\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i)$ (**is** *?th3*)
and $box\ c\ d \subseteq box\ a\ b \longleftrightarrow (\forall i.\ c\$i < d\$i) \longrightarrow (\forall i.\ a\$i \leq c\$i \wedge d\$i \leq b\$i)$ (**is** *?th4*)
using *subset-box*[*of c d a b*] **by** (*simp-all add*: *Basis-vec-def inner-axis*)

lemma *disjoint-interval-cart*:
fixes $a::real\hat{}'n$
shows $cbox\ a\ b \cap cbox\ c\ d = \{\} \longleftrightarrow (\exists i.\ (b\$i < a\$i \vee d\$i < c\$i \vee b\$i < c\$i \vee d\$i < a\$i))$ (**is** *?th1*)
and $cbox\ a\ b \cap box\ c\ d = \{\} \longleftrightarrow (\exists i.\ (b\$i < a\$i \vee d\$i \leq c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$ (**is** *?th2*)
and $box\ a\ b \cap cbox\ c\ d = \{\} \longleftrightarrow (\exists i.\ (b\$i \leq a\$i \vee d\$i < c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$ (**is** *?th3*)
and $box\ a\ b \cap box\ c\ d = \{\} \longleftrightarrow (\exists i.\ (b\$i \leq a\$i \vee d\$i \leq c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$ (**is** *?th4*)
using *disjoint-interval*[*of a b c d*] **by** (*simp-all add*: *Basis-vec-def inner-axis*)

lemma *inter-interval-cart*:
fixes $a :: real\hat{}'n$
shows $cbox\ a\ b \cap cbox\ c\ d = \{(\chi\ i.\ max\ (a\$i)\ (c\$i))\ ..\ (\chi\ i.\ min\ (b\$i)\ (d\$i))\}$
unfolding *inter-interval*
by (*auto simp*: *mem-box less-eq-vec-def*)
(*auto simp*: *Basis-vec-def inner-axis*)

lemma *closed-interval-left-cart*:
fixes $b :: real\hat{}'n$
shows *closed* $\{x::real\hat{}'n.\ \forall i.\ x\$i \leq b\$i\}$
by (*simp add*: *Collect-all-eq closed-INT closed-Collect-le*)

lemma *closed-interval-right-cart*:
fixes *a*::*real^'n*
shows *closed* {*x*::*real^'n*. ∀ *i*. *a*$*i* ≤ *x*$*i*}
by (*simp add*: *Collect-all-eq closed-INT closed-Collect-le*)

lemma *is-interval-cart*:
is-interval (*s*::(*real^'n*) *set*) ⟷
(∀ *a*∈*s*. ∀ *b*∈*s*. ∀ *x*. (∀ *i*. ((*a*$*i* ≤ *x*$*i* ∧ *x*$*i* ≤ *b*$*i*) ∨ (*b*$*i* ≤ *x*$*i* ∧ *x*$*i* ≤ *a*$*i*))) ⟶ *x* ∈ *s*)
by (*simp add*: *is-interval-def Ball-def Basis-vec-def inner-axis imp-ex*)

lemma *closed-halfspace-component-le-cart*: *closed* {*x*::*real^'n*. *x*$*i* ≤ *a*}
by (*simp add*: *closed-Collect-le*)

lemma *closed-halfspace-component-ge-cart*: *closed* {*x*::*real^'n*. *x*$*i* ≥ *a*}
by (*simp add*: *closed-Collect-le*)

lemma *open-halfspace-component-lt-cart*: *open* {*x*::*real^'n*. *x*$*i* < *a*}
by (*simp add*: *open-Collect-less*)

lemma *open-halfspace-component-gt-cart*: *open* {*x*::*real^'n*. *x*$*i* > *a*}
by (*simp add*: *open-Collect-less*)

lemma *Lim-component-le-cart*:
fixes *f* :: *'a* ⇒ *real^'n*
assumes (*f* ⟶⟶ *l*) *net* ¬ (*trivial-limit net*) *eventually* (λ*x*. *f x* $*i* ≤ *b*) *net*
shows *l*$*i* ≤ *b*
by (*rule tendsto-le*[*OF assms*(*2*) *tendsto-const tendsto-vec-nth*, *OF assms*(*1*, *3*)])

lemma *Lim-component-ge-cart*:
fixes *f* :: *'a* ⇒ *real^'n*
assumes (*f* ⟶⟶ *l*) *net* ¬ (*trivial-limit net*) *eventually* (λ*x*. *b* ≤ (*f x*)$*i*) *net*
shows *b* ≤ *l*$*i*
by (*rule tendsto-le*[*OF assms*(*2*) *tendsto-vec-nth tendsto-const*, *OF assms*(*1*, *3*)])

lemma *Lim-component-eq-cart*:
fixes *f* :: *'a* ⇒ *real^'n*
assumes *net*: (*f* ⟶⟶ *l*) *net* ~(*trivial-limit net*) **and** *ev*:*eventually* (λ*x*. *f*(*x*)$*i* = *b*) *net*
shows *l*$*i* = *b*
using *ev*[*unfolded order-eq-iff eventually-conj-iff*] **and**
Lim-component-ge-cart[*OF net*, *of b i*] **and**
Lim-component-le-cart[*OF net*, *of i b*] **by** *auto*

lemma *connected-ivt-component-cart*:
fixes *x* :: *real^'n*
shows *connected s* ⟹ *x* ∈ *s* ⟹ *y* ∈ *s* ⟹ *x*$*k* ≤ *a* ⟹ *a* ≤ *y*$*k* ⟹ (∃ *z*∈*s*. *z*$*k* = *a*)

using *connected-ivt-hyperplane*[*of s x y axis k 1 a*]
by (*auto simp add*: *inner-axis inner-commute*)

lemma *subspace-substandard-cart*: *subspace* {*x*::*real*^-. (∀ *i*. *P i* ⟶ *x*$*i* = *0*)}
unfolding *subspace-def* **by** *auto*

lemma *closed-substandard-cart*:
closed {*x*::*'a*::*real-normed-vector* ^ *'n*. ∀ *i*. *P i* ⟶ *x*$*i* = *0*}
proof −
{ fix *i*::*'n*
have *closed* {*x*::*'a* ^ *'n*. *P i* ⟶ *x*$*i* = *0*}
by (*cases P i*) (*simp-all add*: *closed-Collect-eq*) **}**
thus *?thesis*
unfolding *Collect-all-eq* **by** (*simp add*: *closed-INT*)
qed

lemma *dim-substandard-cart*: *dim* {*x*::*real*^*'n*. ∀ *i*. *i* ∉ *d* ⟶ *x*$*i* = *0*} = *card d*
(**is** *dim ?A* = -)
proof −
let *?a* = λ*x*. *axis x 1* :: *real*^*'n*
have ∗: {*x*. ∀ *i*∈*Basis*. *i* ∉ *?a* ‘ *d* ⟶ *x* • *i* = *0*} = *?A*
by (*auto simp*: *image-iff Basis-vec-def axis-eq-axis inner-axis*)
have *?a* ‘ *d* ⊆ *Basis*
by (*auto simp*: *Basis-vec-def*)
thus *?thesis*
using *dim-substandard*[*of ?a* ‘ *d*] *card-image*[*of ?a d*]
by (*auto simp*: *axis-eq-axis inj-on-def* ∗)
qed

lemma *affinity-inverses*:
assumes *m0*: *m* ≠ (*0*::*'a*::*field*)
shows (λ*x*. *m* ∗*s x* + *c*) ∘ (λ*x*. *inverse*(*m*) ∗*s x* + (−(*inverse*(*m*) ∗*s c*))) = *id*
(λ*x*. *inverse*(*m*) ∗*s x* + (−(*inverse*(*m*) ∗*s c*))) ∘ (λ*x*. *m* ∗*s x* + *c*) = *id*
using *m0*
apply (*auto simp add*: *fun-eq-iff vector-add-ldistrib diff-conv-add-uminus simp del*: *add-uminus-conv-diff*)
apply (*simp-all add*: *vector-smult-lneg*[*symmetric*] *vector-smult-assoc vector-sneg-minus1*[*symmetric*])
done

lemma *vector-affinity-eq*:
assumes *m0*: (*m*::*'a*::*field*) ≠ *0*
shows *m* ∗*s x* + *c* = *y* ⟷ *x* = *inverse m* ∗*s y* + −(*inverse m* ∗*s c*)
proof
assume *h*: *m* ∗*s x* + *c* = *y*
hence *m* ∗*s x* = *y* − *c* **by** (*simp add*: *field-simps*)
hence *inverse m* ∗*s* (*m* ∗*s x*) = *inverse m* ∗*s* (*y* − *c*) **by** *simp*
then show *x* = *inverse m* ∗*s y* + − (*inverse m* ∗*s c*)
using *m0* **by** (*simp add*: *vector-smult-assoc vector-ssub-ldistrib*)

next
assume *h*: $x = inverse\ m\ *s\ y + -\ (inverse\ m\ *s\ c)$
show $m\ *s\ x + c = y$ **unfolding** *h*
using *m0* **by** (*simp add*: *vector-smult-assoc vector-ssub-ldistrib*)
qed

lemma *vector-eq-affinity*:
$(m::'a::field) \neq 0 ==> (y = m\ *s\ x + c \longleftrightarrow inverse(m)\ *s\ y + -(inverse(m)\ *s\ c) = x)$
using *vector-affinity-eq*[**where** *m*=*m* **and** *x*=*x* **and** *y*=*y* **and** *c*=*c*]
by *metis*

lemma *vector-cart*:
fixes $f :: real\hat{}'n \Rightarrow real$
shows $(\chi\ i.\ f\ (axis\ i\ 1)) = (\sum i{\in}Basis.\ f\ i\ *_R\ i)$
unfolding *euclidean-eq-iff*[**where** *'a*=$real\hat{}'n$]
by *simp* (*simp add*: *Basis-vec-def inner-axis*)

lemma *const-vector-cart*:$((\chi\ i.\ d)::real\hat{}'n) = (\sum i{\in}Basis.\ d\ *_R\ i)$
by (*rule vector-cart*)

36.6 Convex Euclidean Space

lemma *Cart-1*:$(1::real\hat{}'n) = \sum Basis$
using *const-vector-cart*[*of 1*] **by** (*simp add*: *one-vec-def*)

declare *vector-add-ldistrib*[*simp*] *vector-ssub-ldistrib*[*simp*] *vector-smult-assoc*[*simp*] *vector-smult-rneg*[*simp*]
declare *vector-sadd-rdistrib*[*simp*] *vector-sub-rdistrib*[*simp*]

lemmas *vector-component-simps* = *vector-minus-component vector-smult-component vector-add-component less-eq-vec-def vec-lambda-beta vector-uminus-component*

lemma *convex-box-cart*:
assumes $\bigwedge i.\ convex\ \{x.\ P\ i\ x\}$
shows $convex\ \{x.\ \forall i.\ P\ i\ (x\$i)\}$
using *assms* **unfolding** *convex-def* **by** *auto*

lemma *convex-positive-orthant-cart*: $convex\ \{x::real\hat{}'n.\ (\forall i.\ 0 \leq x\$i)\}$
by (*rule convex-box-cart*) (*simp add*: *atLeast-def*[*symmetric*] *convex-real-interval*)

lemma *unit-interval-convex-hull-cart*:
$cbox\ (0::real\hat{}'n)\ 1 = convex\ hull\ \{x.\ \forall i.\ (x\$i = 0) \vee (x\$i = 1)\}$
unfolding *Cart-1 unit-interval-convex-hull*[**where** *'a*=$real\hat{}'n$] *box-real*[*symmetric*]
by (*rule arg-cong*[**where** $f{=}\lambda x.\ convex\ hull\ x$]) (*simp add*: *Basis-vec-def inner-axis*)

lemma *cube-convex-hull-cart*:
assumes $0 < d$
obtains $s::(real\hat{}'n)\ set$

where *finite s cbox* $(x - (\chi\ i.\ d))\ (x + (\chi\ i.\ d)) = convex\ hull\ s$
proof −
from *assms* **obtain** *s* **where** *finite s*
and *cbox* $(x - setsum\ (op\ *_R\ d)\ Basis)\ (x + setsum\ (op\ *_R\ d)\ Basis) = convex\ hull\ s$
by (*rule cube-convex-hull*)
with *that*[*of s*] **show** *thesis*
by (*simp add*: *const-vector-cart*)
qed

36.7 Derivative

definition *jacobian f net* = *matrix*(*frechet-derivative f net*)

lemma *jacobian-works*:
$(f::(real\hat{}'a) \Rightarrow (real\hat{}'b))$ *differentiable net* $\longleftrightarrow$
(*f has-derivative* $(\lambda h.\ (jacobian\ f\ net) *v\ h))$ *net*
apply *rule*
unfolding *jacobian-def*
apply (*simp only*: *matrix-works*[*OF linear-frechet-derivative*]) **defer**
apply (*rule differentiableI*)
apply *assumption*
unfolding *frechet-derivative-works*
apply *assumption*
done

36.8 Component of the differential must be zero if it exists at a local maximum or minimum for that corresponding component.

lemma *differential-zero-maxmin-cart*:
fixes $f::real\hat{}'a \Rightarrow real\hat{}'b$
assumes $0 < e$ $((\forall\ y \in ball\ x\ e.\ (f\ y)\$k \leq (f\ x)\$k) \vee (\forall\ y \in ball\ x\ e.\ (f\ x)\$k \leq (f\ y)\$k))$
f differentiable $(at\ x)$
shows *jacobian f* $(at\ x)\ \$\ k = 0$
using *differential-zero-maxmin-component*[*of axis k 1 e x f*] *assms*
vector-cart[*of* $\lambda j.$ *frechet-derivative f* $(at\ x)\ j\ \$\ k$]
by (*simp add*: *Basis-vec-def axis-eq-axis inner-axis jacobian-def matrix-def*)

36.9 Lemmas for working on $(real,\ 1)\ vec$

lemma *forall-1*[*simp*]: $(\forall\ i::1.\ P\ i) \longleftrightarrow P\ 1$
by (*metis* (*full-types*) *num1-eq-iff*)

lemma *ex-1*[*simp*]: $(\exists\ x::1.\ P\ x) \longleftrightarrow P\ 1$
by *auto* (*metis* (*full-types*) *num1-eq-iff*)

lemma *exhaust-2*:

fixes $x :: 2$
shows $x = 1 \vee x = 2$
proof (*induct x*)
case (*of-int z*)
then have $0 <= z$ **and** $z < 2$ **by** *simp-all*
then have $z = 0 \mid z = 1$ **by** *arith*
then show *?case* **by** *auto*
qed

lemma *forall-2*: $(\forall i::2.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2$
by (*metis exhaust-2*)

lemma *exhaust-3*:
fixes $x :: 3$
shows $x = 1 \vee x = 2 \vee x = 3$
proof (*induct x*)
case (*of-int z*)
then have $0 <= z$ **and** $z < 3$ **by** *simp-all*
then have $z = 0 \vee z = 1 \vee z = 2$ **by** *arith*
then show *?case* **by** *auto*
qed

lemma *forall-3*: $(\forall i::3.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3$
by (*metis exhaust-3*)

lemma *UNIV-1* [*simp*]: $UNIV = \{1::1\}$
by (*auto simp add*: *num1-eq-iff*)

lemma *UNIV-2*: $UNIV = \{1::2, 2::2\}$
using *exhaust-2* **by** *auto*

lemma *UNIV-3*: $UNIV = \{1::3, 2::3, 3::3\}$
using *exhaust-3* **by** *auto*

lemma *setsum-1*: $setsum\ f\ (UNIV::1\ set) = f\ 1$
unfolding *UNIV-1* **by** *simp*

lemma *setsum-2*: $setsum\ f\ (UNIV::2\ set) = f\ 1 + f\ 2$
unfolding *UNIV-2* **by** *simp*

lemma *setsum-3*: $setsum\ f\ (UNIV::3\ set) = f\ 1 + f\ 2 + f\ 3$
unfolding *UNIV-3* **by** (*simp add*: *ac-simps*)

instantiation *num1* :: *cart-one*
begin

instance
proof
show $CARD(1) = Suc\ 0$ **by** *auto*

qed

end

36.10 The collapse of the general concepts to dimension one.

lemma *vector-one*: $(x::'a\ \hat{}1) = (\chi\ i.\ (x\$1))$
by (*simp add*: *vec-eq-iff*)

lemma *forall-one*: $(\forall (x::'a\ \hat{}1).\ P\ x) \longleftrightarrow (\forall x.\ P(\chi\ i.\ x))$
apply *auto*
apply (*erule-tac x*= *x*$1 **in** *allE*)
apply (*simp only*: *vector-one*[*symmetric*])
done

lemma *norm-vector-1*: *norm* $(x :: -\hat{}1) =$ *norm* $(x\$1)$
by (*simp add*: *norm-vec-def*)

lemma *norm-real*: *norm*$(x::real\ \hat{}\ 1) = |x\$1|$
by (*simp add*: *norm-vector-1*)

lemma *dist-real*: *dist*$(x::real\ \hat{}\ 1)\ y = |(x\$1) - (y\$1)|$
by (*auto simp add*: *norm-real dist-norm*)

36.11 Explicit vector construction from lists.

definition *vector* $l = (\chi\ i.\ foldr\ (\lambda x\ f\ n.\ fun\text{-}upd\ (f\ (n+1))\ n\ x)\ l\ (\lambda n\ x.\ 0)\ 1\ i)$

lemma *vector-1*: $(vector[x])\ \$1 = x$
unfolding *vector-def* **by** *simp*

lemma *vector-2*:
$(vector[x,y])\ \$1 = x$
$(vector[x,y] :: 'a\hat{}2)\$2 = (y::'a::zero)$
unfolding *vector-def* **by** *simp-all*

lemma *vector-3*:
$(vector\ [x,y,z] ::('a::zero)\hat{}3)\$1 = x$
$(vector\ [x,y,z] ::('a::zero)\hat{}3)\$2 = y$
$(vector\ [x,y,z] ::('a::zero)\hat{}3)\$3 = z$
unfolding *vector-def* **by** *simp-all*

lemma *forall-vector-1*: $(\forall v::'a::zero\hat{}1.\ P\ v) \longleftrightarrow (\forall x.\ P(vector[x]))$
apply *auto*
apply (*erule-tac x*=*v*$1 **in** *allE*)
apply (*subgoal-tac vector* $[v\$1] = v$)
apply *simp*
apply (*vector vector-def*)
apply *simp*
done

lemma *forall-vector-2*: $(\forall\, v::'a::zero\hat{}2.\ P\ v) \longleftrightarrow (\forall\, x\ y.\ P(vector[x,\ y]))$
apply *auto*
apply (*erule-tac x=v$1* **in** *allE*)
apply (*erule-tac x=v$2* **in** *allE*)
apply (*subgoal-tac vector* $[v\$1,\ v\$2] = v$)
apply *simp*
apply (*vector vector-def*)
apply (*simp add*: *forall-2*)
done

lemma *forall-vector-3*: $(\forall\, v::'a::zero\hat{}3.\ P\ v) \longleftrightarrow (\forall\, x\ y\ z.\ P(vector[x,\ y,\ z]))$
apply *auto*
apply (*erule-tac x=v$1* **in** *allE*)
apply (*erule-tac x=v$2* **in** *allE*)
apply (*erule-tac x=v$3* **in** *allE*)
apply (*subgoal-tac vector* $[v\$1,\ v\$2,\ v\$3] = v$)
apply *simp*
apply (*vector vector-def*)
apply (*simp add*: *forall-3*)
done

lemma *bounded-linear-component-cart*[*intro*]: *bounded-linear* $(\lambda x::real\hat{}'n.\ x\ \$\ k)$
apply (*rule bounded-linearI*[**where** *K=1*])
using *component-le-norm-cart*[*of - k*] **unfolding** *real-norm-def* **by** *auto*

lemma *integral-component-eq-cart*[*simp*]:
fixes $f :: 'n::euclidean\text{-}space \Rightarrow real\hat{}'m$
assumes *f integrable-on s*
shows *integral s* $(\lambda x.\ f\ x\ \$\ k) =$ *integral s f* $\$\ k$
using *integral-linear*[*OF assms*(*1*) *bounded-linear-component-cart*,*unfolded o-def*]
.

lemma *interval-split-cart*:
$\{a..b::real\hat{}'n\} \cap \{x.\ x\$k \le c\} = \{a\ ..\ (\chi\ i.\ if\ i = k\ then\ min\ (b\$k)\ c\ else\ b\$i)\}$
$cbox\ a\ b \cap \{x.\ x\$k \ge c\} = \{(\chi\ i.\ if\ i = k\ then\ max\ (a\$k)\ c\ else\ a\$i)\ ..\ b\}$
apply (*rule-tac*[!] *set-eqI*)
unfolding *Int-iff mem-interval-cart mem-Collect-eq interval-cbox-cart*
unfolding *vec-lambda-beta*
by *auto*

end

37 Fashoda meet theorem

theory *Fashoda*
imports *Brouwer-Fixpoint Path-Connected Cartesian-Euclidean-Space*
begin

lemma *cart-eq-inner-axis*: $a \$ i = a \cdot axis\ i\ 1$
by (*simp add*: *inner-axis*)

lemma *axis-in-Basis*: $a \in Basis \Longrightarrow axis\ i\ a \in Basis$
by (*auto simp add*: *Basis-vec-def axis-eq-axis*)

37.1 Bijections between intervals.

definition *interval-bij* :: $'a \times 'a \Rightarrow 'a \times 'a \Rightarrow 'a \Rightarrow 'a$::*euclidean-space*
where *interval-bij* =
$(\lambda(a, b)\ (u, v)\ x.\ (\sum i{\in}Basis.\ (u{\cdot}i + (x{\cdot}i - a{\cdot}i)\ /\ (b{\cdot}i - a{\cdot}i) * (v{\cdot}i - u{\cdot}i))\ *_R\ i))$

lemma *interval-bij-affine*:
interval-bij $(a,b)\ (u,v) = (\lambda x.\ (\sum i{\in}Basis.\ ((v{\cdot}i - u{\cdot}i)\ /\ (b{\cdot}i - a{\cdot}i) * (x{\cdot}i))\ *_R\ i) +$
$(\sum i{\in}Basis.\ (u{\cdot}i - (v{\cdot}i - u{\cdot}i)\ /\ (b{\cdot}i - a{\cdot}i) * (a{\cdot}i))\ *_R\ i))$
by (*auto simp*: *setsum.distrib*[*symmetric*] *scaleR-add-left*[*symmetric*] *interval-bij-def fun-eq-iff*
field-simps inner-simps add-divide-distrib[*symmetric*] *intro*!: *setsum.cong*)

lemma *continuous-interval-bij*:
fixes *a b* :: $'a$::*euclidean-space*
shows *continuous* (*at x*) (*interval-bij* (*a*, *b*) (*u*, *v*))
by (*auto simp add*: *divide-inverse interval-bij-def intro*!: *continuous-setsum continuous-intros*)

lemma *continuous-on-interval-bij*: *continuous-on s* (*interval-bij* (*a*, *b*) (*u*, *v*))
apply(*rule continuous-at-imp-continuous-on*)
apply (*rule*, *rule continuous-interval-bij*)
done

lemma *in-interval-interval-bij*:
fixes *a b u v x* :: $'a$::*euclidean-space*
assumes $x \in cbox\ a\ b$
and $cbox\ u\ v \neq \{\}$
shows *interval-bij* (*a*, *b*) (*u*, *v*) $x \in cbox\ u\ v$
apply (*simp only*: *interval-bij-def split-conv mem-box inner-setsum-left-Basis cong*: *ball-cong*)
apply *safe*
proof −
fix *i* :: $'a$
assume *i*: $i \in Basis$
have $cbox\ a\ b \neq \{\}$
using *assms* **by** *auto*
with *i* **have** ∗: $a{\cdot}i \leq b{\cdot}i\ u{\cdot}i \leq v{\cdot}i$
using *assms*(*2*) **by** (*auto simp add*: *box-eq-empty*)
have *x*: $a{\cdot}i{\leq}x{\cdot}i\ x{\cdot}i{\leq}b{\cdot}i$

using *assms(1)*[*unfolded mem-box*] **using** *i* **by** *auto*
have $0 \leq (x \cdot i - a \cdot i) \,/\, (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i)$
using $* x$ **by** *auto*
then show $u \cdot i \leq u \cdot i + (x \cdot i - a \cdot i) \,/\, (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i)$
using $*$ **by** *auto*
have $((x \cdot i - a \cdot i) \,/\, (b \cdot i - a \cdot i)) * (v \cdot i - u \cdot i) \leq 1 * (v \cdot i - u \cdot i)$
apply (*rule mult-right-mono*)
unfolding *divide-le-eq-1*
using $* x$
apply *auto*
done
then show $u \cdot i + (x \cdot i - a \cdot i) \,/\, (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i) \leq v \cdot i$
using $*$ **by** *auto*
qed

lemma *interval-bij-bij*:
$\forall (i::'a::euclidean\text{-}space) \in Basis.\ a \cdot i < b \cdot i \wedge u \cdot i < v \cdot i \Longrightarrow$
interval-bij $(a,\ b)$ $(u,\ v)$ (*interval-bij* $(u,\ v)$ $(a,\ b)$ $x) = x$
by (*auto simp*: *interval-bij-def euclidean-eq-iff* [**where** $'a='a$])

lemma *interval-bij-bij-cart*: **fixes** $x::real\hat{}'n$ **assumes** $\forall\, i.\ a\$i < b\$i \wedge u\$i < v\$i$
shows *interval-bij* (a,b) (u,v) (*interval-bij* (u,v) (a,b) $x) = x$
using *assms* **by** (*intro interval-bij-bij*) (*auto simp*: *Basis-vec-def inner-axis*)

37.2 Fashoda meet theorem

lemma *infnorm-2*:
fixes x :: *real^2*
shows *infnorm* x = *max* $|x\$1|$ $|x\$2|$
unfolding *infnorm-cart UNIV-2* **by** (*rule cSup-eq*) *auto*

lemma *infnorm-eq-1-2*:
fixes x :: *real^2*
shows *infnorm* $x = 1 \longleftrightarrow$
$|x\$1| \leq 1 \wedge |x\$2| \leq 1 \wedge (x\$1 = -1 \vee x\$1 = 1 \vee x\$2 = -1 \vee x\$2 = 1)$
unfolding *infnorm-2* **by** *auto*

lemma *infnorm-eq-1-imp*:
fixes x :: *real^2*
assumes *infnorm* $x = 1$
shows $|x\$1| \leq 1$ **and** $|x\$2| \leq 1$
using *assms* **unfolding** *infnorm-eq-1-2* **by** *auto*

lemma *fashoda-unit*:
fixes $f\ g$ :: *real* $\Rightarrow$ *real^2*
assumes f ' $\{-1\ ..\ 1\} \subseteq cbox\ (-1)\ 1$
and g ' $\{-1\ ..\ 1\} \subseteq cbox\ (-1)\ 1$
and *continuous-on* $\{-1\ ..\ 1\}$ f
and *continuous-on* $\{-1\ ..\ 1\}$ g

```
    and f (− 1)$1 = − 1
    and f 1$1 = 1 g (− 1) $2 = −1
    and g 1 $2 = 1
  shows ∃ s∈{−1 .. 1}. ∃ t∈{−1 .. 1}. f s = g t
proof (rule ccontr)
  assume ¬ ?thesis
  note as = this[unfolded bex-simps,rule-format]
  def sqprojection ≡ λz::real^2. (inverse (infnorm z)) *R z
  def negatex ≡ λx::real^2. (vector [−(x$1), x$2])::real^2
  have lem1: ∀ z::real^2. infnorm (negatex z) = infnorm z
    unfolding negatex-def infnorm-2 vector-2 by auto
  have lem2: ∀ z. z ≠ 0 ⟶ infnorm (sqprojection z) = 1
    unfolding sqprojection-def
    unfolding infnorm-mul[unfolded scalar-mult-eq-scaleR]
    unfolding abs-inverse real-abs-infnorm
    apply (subst infnorm-eq-0[symmetric])
    apply auto
    done
  let ?F = λw::real^2. (f ∘ (λx. x$1)) w − (g ∘ (λx. x$2)) w
  have *: ⋀i. (λx::real^2. x $ i) ` cbox (− 1) 1 = {−1 .. 1}
    apply (rule set-eqI)
    unfolding image-iff Bex-def mem-interval-cart interval-cbox-cart
    apply rule
    defer
    apply (rule-tac x=vec x in exI)
    apply auto
    done
  {
    fix x
    assume x ∈ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w) ` (cbox (− 1)
(1::real^2))
    then obtain w :: real^2 where w:
        w ∈ cbox (− 1) 1
        x = (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w
      unfolding image-iff ..
    then have x ≠ 0
      using as[of w$1 w$2]
      unfolding mem-interval-cart atLeastAtMost-iff
      by auto
  } note x0 = this
  have 21: ⋀i::2. i ≠ 1 ⟹ i = 2
    using UNIV-2 by auto
  have 1: box (− 1) (1::real^2) ≠ {}
    unfolding interval-eq-empty-cart by auto
  have 2: continuous-on (cbox (− 1) 1) (negatex ∘ sqprojection ∘ ?F)
    apply (intro continuous-intros continuous-on-component)
    unfolding *
    apply (rule assms)+
    apply (subst sqprojection-def)
```

```
    apply (intro continuous-intros)
    apply (simp add: infnorm-eq-0 x0)
    apply (rule linear-continuous-on)
   proof −
    show bounded-linear negatex
     apply (rule bounded-linearI')
     unfolding vec-eq-iff
    proof (rule-tac[!] allI)
     fix i :: 2
     fix x y :: real^2
     fix c :: real
     show negatex (x + y) $ i =
       (negatex x + negatex y) $ i negatex (c *R x) $ i = (c *R negatex x) $ i
       apply −
       apply (case-tac[!] i≠1)
       prefer 3
       apply (drule-tac[1−2] 21)
       unfolding negatex-def
       apply (auto simp add:vector-2)
       done
    qed
   qed
   have 3: (negatex ∘ sqprojection ∘ ?F) ' cbox (−1) 1 ⊆ cbox (−1) 1
    unfolding subset-eq
   proof (rule, goal-cases)
    case (1 x)
    then obtain y :: real^2 where y:
        y ∈ cbox (− 1) 1
        x = (negatex ∘ sqprojection ∘ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2))
w)) y
      unfolding image-iff ..
    have ?F y ≠ 0
     apply (rule x0)
     using y(1)
     apply auto
     done
    then have ∗: infnorm (sqprojection (?F y)) = 1
     unfolding y o-def
     by − (rule lem2[rule-format])
    have infnorm x = 1
     unfolding ∗[symmetric] y o-def
     by (rule lem1[rule-format])
    then show x ∈ cbox (−1) 1
     unfolding mem-interval-cart interval-cbox-cart infnorm-2
     apply −
     apply rule
    proof −
     fix i
     assume max |x $ 1| |x $ 2| = 1
```

```
      then show (− 1) $ i ≤ x $ i ∧ x $ i ≤ 1 $ i
        apply (cases i = 1)
        defer
        apply (drule 21)
        apply auto
        done
    qed
  qed
  obtain x :: real^2 where x:
      x ∈ cbox (− 1) 1
      (negatex ∘ sqprojection ∘ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w)) x
= x
     apply (rule brouwer-weak[of cbox (− 1) (1::real^2) negatex ∘ sqprojection ∘
?F])
    apply (rule compact-cbox convex-box)+
    unfolding interior-cbox
    apply (rule 1 2 3)+
    apply blast
    done
  have ?F x ≠ 0
    apply (rule x0)
    using x(1)
    apply auto
    done
  then have ∗: infnorm (sqprojection (?F x)) = 1
    unfolding o-def
    by (rule lem2[rule-format])
  have nx: infnorm x = 1
    apply (subst x(2)[symmetric])
    unfolding ∗[symmetric] o-def
    apply (rule lem1[rule-format])
    done
  have ∀x i. x ≠ 0 ⟶ (0 < (sqprojection x)$i ⟷ 0 < x$i)
    and ∀x i. x ≠ 0 ⟶ ((sqprojection x)$i < 0 ⟷ x$i < 0)
    apply −
    apply (rule-tac[!] allI impI)+
  proof −
    fix x :: real^2
    fix i :: 2
    assume x: x ≠ 0
    have inverse (infnorm x) > 0
      using x[unfolded infnorm-pos-lt[symmetric]] by auto
    then show (0 < sqprojection x $ i) = (0 < x $ i)
      and (sqprojection x $ i < 0) = (x $ i < 0)
      unfolding sqprojection-def vector-component-simps vector-scaleR-component
real-scaleR-def
      unfolding zero-less-mult-iff mult-less-0-iff
      by (auto simp add: field-simps)
  qed
```

```
note lem3 = this[rule-format]
have x1: x $ 1 ∈ {− 1..1::real} x $ 2 ∈ {− 1..1::real}
  using x(1) unfolding mem-interval-cart by auto
then have nz: f (x $ 1) − g (x $ 2) ≠ 0
  unfolding right-minus-eq
  apply −
  apply (rule as)
  apply auto
  done
have x $ 1 = −1 ∨ x $ 1 = 1 ∨ x $ 2 = −1 ∨ x $ 2 = 1
  using nx unfolding infnorm-eq-1-2 by auto
then show False
proof −
  fix P Q R S
  presume P ∨ Q ∨ R ∨ S
    and P ⟹ False
    and Q ⟹ False
    and R ⟹ False
    and S ⟹ False
  then show False by auto
next
  assume as: x$1 = 1
  then have ∗: f (x $ 1) $ 1 = 1
    using assms(6) by auto
  have sqprojection (f (x$1) − g (x$2)) $ 1 < 0
    using x(2)[unfolded o-def vec-eq-iff,THEN spec[where x=1]]
    unfolding as negatex-def vector-2
    by auto
  moreover
  from x1 have g (x $ 2) ∈ cbox (−1) 1
    apply −
    apply (rule assms(2)[unfolded subset-eq,rule-format])
    apply auto
    done
  ultimately show False
    unfolding lem3[OF nz] vector-component-simps ∗ mem-interval-cart
    apply (erule-tac x=1 in allE)
    apply auto
    done
next
  assume as: x$1 = −1
  then have ∗: f (x $ 1) $ 1 = − 1
    using assms(5) by auto
  have sqprojection (f (x$1) − g (x$2)) $ 1 > 0
    using x(2)[unfolded o-def vec-eq-iff,THEN spec[where x=1]]
    unfolding as negatex-def vector-2
    by auto
  moreover
  from x1 have g (x $ 2) ∈ cbox (−1) 1
```

```
    apply −
    apply (rule assms(2)[unfolded subset-eq,rule-format])
    apply auto
    done
  ultimately show False
    unfolding lem3[OF nz] vector-component-simps * mem-interval-cart
    apply (erule-tac x=1 in allE)
    apply auto
    done
 next
  assume as: x$2 = 1
  then have *: g (x $ 2) $ 2 = 1
    using assms(8) by auto
  have sqprojection (f (x$1) − g (x$2)) $ 2 > 0
    using x(2)[unfolded o-def vec-eq-iff,THEN spec[where x=2]]
    unfolding as negatex-def vector-2
    by auto
  moreover
  from x1 have f (x $ 1) ∈ cbox (−1) 1
    apply −
    apply (rule assms(1)[unfolded subset-eq,rule-format])
    apply auto
    done
  ultimately show False
    unfolding lem3[OF nz] vector-component-simps * mem-interval-cart
    apply (erule-tac x=2 in allE)
    apply auto
    done
 next
  assume as: x$2 = −1
  then have *: g (x $ 2) $ 2 = − 1
    using assms(7) by auto
  have sqprojection (f (x$1) − g (x$2)) $ 2 < 0
    using x(2)[unfolded o-def vec-eq-iff,THEN spec[where x=2]]
    unfolding as negatex-def vector-2
    by auto
  moreover
  from x1 have f (x $ 1) ∈ cbox (−1) 1
    apply −
    apply (rule assms(1)[unfolded subset-eq,rule-format])
    apply auto
    done
  ultimately show False
    unfolding lem3[OF nz] vector-component-simps * mem-interval-cart
    apply (erule-tac x=2 in allE)
    apply auto
    done
 qed auto
qed
```

lemma *fashoda-unit-path*:
fixes *f g* :: *real* ⇒ *real^2*
assumes *path f*
and *path g*
and *path-image f* ⊆ *cbox* (−*1*) *1*
and *path-image g* ⊆ *cbox* (−*1*) *1*
and (*pathstart f*)$*1* = −*1*
and (*pathfinish f*)$*1* = *1*
and (*pathstart g*)$*2* = −*1*
and (*pathfinish g*)$*2* = *1*
obtains *z* **where** *z* ∈ *path-image f* **and** *z* ∈ *path-image g*
proof −
note *assms*=*assms*[*unfolded path-def pathstart-def pathfinish-def path-image-def*]
def *iscale* ≡ λ*z*::*real*. *inverse 2* ∗*R* (*z* + *1*)
have *isc*: *iscale* ‘ {− *1*..*1*} ⊆ {*0*..*1*}
unfolding *iscale-def* **by** *auto*
have ∃*s*∈{− *1*..*1*}. ∃*t*∈{− *1*..*1*}. (*f* ∘ *iscale*) *s* = (*g* ∘ *iscale*) *t*
proof (*rule fashoda-unit*)
show (*f* ∘ *iscale*) ‘ {− *1*..*1*} ⊆ *cbox* (− *1*) *1* (*g* ∘ *iscale*) ‘ {− *1*..*1*} ⊆ *cbox* (− *1*) *1*
using *isc* **and** *assms*(*3*−*4*) **by** (*auto simp add*: *image-comp* [*symmetric*])
have ∗: *continuous-on* {− *1*..*1*} *iscale*
unfolding *iscale-def* **by** (*rule continuous-intros*)+
show *continuous-on* {− *1*..*1*} (*f* ∘ *iscale*) *continuous-on* {− *1*..*1*} (*g* ∘ *iscale*)
apply −
apply (*rule-tac*[!] *continuous-on-compose*[*OF* ∗])
apply (*rule-tac*[!] *continuous-on-subset*[*OF* - *isc*])
apply (*rule assms*)+
done
have ∗: (*1* / *2*) ∗*R* (*1* + (*1*::*real^1*)) = *1*
unfolding *vec-eq-iff* **by** *auto*
show (*f* ∘ *iscale*) (− *1*) $ *1* = − *1*
and (*f* ∘ *iscale*) *1* $ *1* = *1*
and (*g* ∘ *iscale*) (− *1*) $ *2* = −*1*
and (*g* ∘ *iscale*) *1* $ *2* = *1*
unfolding *o-def iscale-def*
using *assms*
by (*auto simp add*: ∗)
qed
then obtain *s t* **where** *st*:
s ∈ {− *1*..*1*}
t ∈ {− *1*..*1*}
(*f* ∘ *iscale*) *s* = (*g* ∘ *iscale*) *t*
by *auto*
show *thesis*
apply (*rule-tac z* = *f* (*iscale s*) **in** *that*)
using *st*
unfolding *o-def path-image-def image-iff*

```
    apply −
    apply (rule-tac x=iscale s in bexI)
    prefer 3
    apply (rule-tac x=iscale t in bexI)
    using isc[unfolded subset-eq, rule-format]
    apply auto
    done
qed

lemma fashoda:
  fixes b :: real^2
  assumes path f
    and path g
    and path-image f ⊆ cbox a b
    and path-image g ⊆ cbox a b
    and (pathstart f)$1 = a$1
    and (pathfinish f)$1 = b$1
    and (pathstart g)$2 = a$2
    and (pathfinish g)$2 = b$2
  obtains z where z ∈ path-image f and z ∈ path-image g
proof −
  fix P Q S
  presume P ∨ Q ∨ S P ⟹ thesis and Q ⟹ thesis and S ⟹ thesis
  then show thesis
    by auto
next
  have cbox a b ≠ {}
    using assms(3) using path-image-nonempty[of f] by auto
  then have a ≤ b
    unfolding interval-eq-empty-cart less-eq-vec-def by (auto simp add: not-less)
  then show a$1 = b$1 ∨ a$2 = b$2 ∨ (a$1 < b$1 ∧ a$2 < b$2)
    unfolding less-eq-vec-def forall-2 by auto
next
  assume as: a$1 = b$1
  have ∃z∈path-image g. z$2 = (pathstart f)$2
    apply (rule connected-ivt-component-cart)
    apply (rule connected-path-image assms)+
    apply (rule pathstart-in-path-image)
    apply (rule pathfinish-in-path-image)
   unfolding assms using assms(3)[unfolded path-image-def subset-eq,rule-format,of
f 0]
    unfolding pathstart-def
    apply (auto simp add: less-eq-vec-def mem-interval-cart)
    done
  then obtain z :: real^2 where z: z ∈ path-image g z $ 2 = pathstart f $ 2 ..
  have z ∈ cbox a b
    using z(1) assms(4)
    unfolding path-image-def
    by blast
```

```
  then have z = f 0
    unfolding vec-eq-iff forall-2
    unfolding z(2) pathstart-def
   using assms(3)[unfolded path-image-def subset-eq mem-interval-cart,rule-format,of
f 0 1]
    unfolding mem-interval-cart
    apply (erule-tac x=1 in allE)
    using as
    apply auto
    done
  then show thesis
    apply -
    apply (rule that[OF - z(1)])
    unfolding path-image-def
    apply auto
    done
next
  assume as: a$2 = b$2
  have ∃z∈path-image f. z$1 = (pathstart g)$1
    apply (rule connected-ivt-component-cart)
    apply (rule connected-path-image assms)+
    apply (rule pathstart-in-path-image)
    apply (rule pathfinish-in-path-image)
    unfolding assms
    using assms(4)[unfolded path-image-def subset-eq,rule-format,of g 0]
    unfolding pathstart-def
    apply (auto simp add: less-eq-vec-def mem-interval-cart)
    done
  then obtain z where z: z ∈ path-image f z $ 1 = pathstart g $ 1 ..
  have z ∈ cbox a b
    using z(1) assms(3)
    unfolding path-image-def
    by blast
  then have z = g 0
    unfolding vec-eq-iff forall-2
    unfolding z(2) pathstart-def
   using assms(4)[unfolded path-image-def subset-eq mem-interval-cart,rule-format,of
g 0 2]
    unfolding mem-interval-cart
    apply (erule-tac x=2 in allE)
    using as
    apply auto
    done
  then show thesis
    apply -
    apply (rule that[OF z(1)])
    unfolding path-image-def
    apply auto
    done
```

```
next
  assume as: a $ 1 < b $ 1 ∧ a $ 2 < b $ 2
  have int-nem: cbox (−1) (1::real^2) ≠ {}
    unfolding interval-eq-empty-cart by auto
  obtain z :: real^2 where z:
      z ∈ (interval-bij (a, b) (− 1, 1) ∘ f) ‘ {0..1}
      z ∈ (interval-bij (a, b) (− 1, 1) ∘ g) ‘ {0..1}
    apply (rule fashoda-unit-path[of interval-bij (a,b) (− 1,1) ∘ f interval-bij (a,b)
(− 1,1) ∘ g])
    unfolding path-def path-image-def pathstart-def pathfinish-def
    apply (rule-tac[1−2] continuous-on-compose)
    apply (rule assms[unfolded path-def] continuous-on-interval-bij)+
    unfolding subset-eq
    apply(rule-tac[1−2] ballI)
  proof −
    fix x
    assume x ∈ (interval-bij (a, b) (− 1, 1) ∘ f) ‘ {0..1}
    then obtain y where y:
        y ∈ {0..1}
        x = (interval-bij (a, b) (− 1, 1) ∘ f) y
      unfolding image-iff ..
    show x ∈ cbox (− 1) 1
      unfolding y o-def
      apply (rule in-interval-interval-bij)
      using y(1)
      using assms(3)[unfolded path-image-def subset-eq] int-nem
      apply auto
      done
  next
    fix x
    assume x ∈ (interval-bij (a, b) (− 1, 1) ∘ g) ‘ {0..1}
    then obtain y where y:
        y ∈ {0..1}
        x = (interval-bij (a, b) (− 1, 1) ∘ g) y
      unfolding image-iff ..
    show x ∈ cbox (− 1) 1
      unfolding y o-def
      apply (rule in-interval-interval-bij)
      using y(1)
      using assms(4)[unfolded path-image-def subset-eq] int-nem
      apply auto
      done
  next
    show (interval-bij (a, b) (− 1, 1) ∘ f) 0 $ 1 = −1
      and (interval-bij (a, b) (− 1, 1) ∘ f) 1 $ 1 = 1
      and (interval-bij (a, b) (− 1, 1) ∘ g) 0 $ 2 = −1
      and (interval-bij (a, b) (− 1, 1) ∘ g) 1 $ 2 = 1
      using assms as
      by (simp-all add: axis-in-Basis cart-eq-inner-axis pathstart-def pathfinish-def
```

```
interval-bij-def)
        (simp-all add: inner-axis)
  qed
  from z(1) obtain zf where zf:
      zf ∈ {0..1}
      z = (interval-bij (a, b) (- 1, 1) ∘ f) zf
    unfolding image-iff ..
  from z(2) obtain zg where zg:
      zg ∈ {0..1}
      z = (interval-bij (a, b) (- 1, 1) ∘ g) zg
    unfolding image-iff ..
  have *: ∀ i. (- 1) $ i < (1::real^2) $ i ∧ a $ i < b $ i
    unfolding forall-2
    using as
    by auto
  show thesis
    apply (rule-tac z=interval-bij (- 1,1) (a,b) z in that)
    apply (subst zf)
    defer
    apply (subst zg)
    unfolding o-def interval-bij-bij-cart[OF *] path-image-def
    using zf(1) zg(1)
    apply auto
    done
qed
```

37.3 Some slightly ad hoc lemmas I use below

```
lemma segment-vertical:
  fixes a :: real^2
  assumes a$1 = b$1
  shows x ∈ closed-segment a b ⟷
    x$1 = a$1 ∧ x$1 = b$1 ∧ (a$2 ≤ x$2 ∧ x$2 ≤ b$2 ∨ b$2 ≤ x$2 ∧ x$2 ≤
a$2)
  (is - = ?R)
proof -
  let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
  {
    presume ?L ⟹ ?R and ?R ⟹ ?L
    then show ?thesis
      unfolding closed-segment-def mem-Collect-eq
    unfolding vec-eq-iff forall-2 scalar-mult-eq-scaleR[symmetric] vector-component-simps
      by blast
  }
  {
    assume ?L
    then obtain u where u:
        x $ 1 = (1 - u) * a $ 1 + u * b $ 1
```

```
    x $ 2 = (1 - u) * a $ 2 + u * b $ 2
    0 ≤ u
    u ≤ 1
   by blast
  { fix b a
   assume b + u * a > a + u * b
   then have (1 - u) * b > (1 - u) * a
    by (auto simp add:field-simps)
   then have b ≥ a
    apply (drule-tac mult-left-less-imp-less)
    using u
    apply auto
    done
   then have u * a ≤ u * b
    apply -
    apply (rule mult-left-mono[OF - u(3)])
    using u(3-4)
    apply (auto simp add: field-simps)
    done
  } note * = this
  {
   fix a b
   assume u * b > u * a
   then have (1 - u) * a ≤ (1 - u) * b
    apply -
    apply (rule mult-left-mono)
    apply (drule mult-left-less-imp-less)
    using u
    apply auto
    done
   then have a + u * b ≤ b + u * a
    by (auto simp add: field-simps)
  } note ** = this
  then show ?R
   unfolding u assms
   using u
   by (auto simp add:field-simps not-le intro: * **)
 }
 {
  assume ?R
  then show ?L
  proof (cases x$2 = b$2)
   case True
   then show ?L
    apply (rule-tac x=(x$2 - a$2) / (b$2 - a$2) in exI)
    unfolding assms True
    using ‹?R›
    apply (auto simp add: field-simps)
    done
```

```
    next
      case False
      then show ?L
        apply (rule-tac x=1 - (x$2 - b$2) / (a$2 - b$2) in exI)
        unfolding assms
        using ‹?R›
        apply (auto simp add: field-simps)
        done
    qed
  }
qed

lemma segment-horizontal:
  fixes a :: real^2
  assumes a$2 = b$2
  shows x ∈ closed-segment a b ⟷
    x$2 = a$2 ∧ x$2 = b$2 ∧ (a$1 ≤ x$1 ∧ x$1 ≤ b$1 ∨ b$1 ≤ x$1 ∧ x$1 ≤
a$1)
  (is - = ?R)
proof -
  let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
  {
    presume ?L ⟹ ?R and ?R ⟹ ?L
    then show ?thesis
      unfolding closed-segment-def mem-Collect-eq
    unfolding vec-eq-iff forall-2 scalar-mult-eq-scaleR[symmetric] vector-component-simps
      by blast
  }
  {
    assume ?L
    then obtain u where u:
        x $ 1 = (1 - u) * a $ 1 + u * b $ 1
        x $ 2 = (1 - u) * a $ 2 + u * b $ 2
        0 ≤ u
        u ≤ 1
      by blast
    {
      fix b a
      assume b + u * a > a + u * b
      then have (1 - u) * b > (1 - u) * a
        by (auto simp add: field-simps)
      then have b ≥ a
        apply (drule-tac mult-left-less-imp-less)
        using u
        apply auto
        done
      then have u * a ≤ u * b
        apply -
```

```
      apply (rule mult-left-mono[OF - u(3)])
      using u(3-4)
      apply (auto simp add: field-simps)
      done
  } note * = this
  {
    fix a b
    assume u * b > u * a
    then have (1 - u) * a ≤ (1 - u) * b
      apply -
      apply (rule mult-left-mono)
      apply (drule mult-left-less-imp-less)
      using u
      apply auto
      done
    then have a + u * b ≤ b + u * a
      by (auto simp add: field-simps)
  } note ** = this
  then show ?R
    unfolding u assms
    using u
    by (auto simp add: field-simps not-le intro: * **)
 }
 {
  assume ?R
  then show ?L
  proof (cases x$1 = b$1)
    case True
    then show ?L
      apply (rule-tac x=(x$1 - a$1) / (b$1 - a$1) in exI)
      unfolding assms True
      using ‹?R›
      apply (auto simp add: field-simps)
      done
  next
    case False
    then show ?L
      apply (rule-tac x=1 - (x$1 - b$1) / (a$1 - b$1) in exI)
      unfolding assms
      using ‹?R›
      apply (auto simp add: field-simps)
      done
  qed
 }
qed
```

37.4 Useful Fashoda corollary pointed out to me by Tom Hales

lemma *fashoda-interlace*:
fixes *a* :: *real^2*
assumes *path f*
and *path g*
and *path-image f* ⊆ *cbox a b*
and *path-image g* ⊆ *cbox a b*
and (*pathstart f*)$*2* = *a*$*2*
and (*pathfinish f*)$*2* = *a*$*2*
and (*pathstart g*)$*2* = *a*$*2*
and (*pathfinish g*)$*2* = *a*$*2*
and (*pathstart f*)$*1* < (*pathstart g*)$*1*
and (*pathstart g*)$*1* < (*pathfinish f*)$*1*
and (*pathfinish f*)$*1* < (*pathfinish g*)$*1*
obtains *z* **where** *z* ∈ *path-image f* **and** *z* ∈ *path-image g*
proof −
have *cbox a b* ≠ {}
using *path-image-nonempty*[*of f*] **using** *assms*(*3*) **by** *auto*
note *ab*=*this*[*unfolded interval-eq-empty-cart not-ex forall-2 not-less*]
have *pathstart f* ∈ *cbox a b*
and *pathfinish f* ∈ *cbox a b*
and *pathstart g* ∈ *cbox a b*
and *pathfinish g* ∈ *cbox a b*
using *pathstart-in-path-image pathfinish-in-path-image*
using *assms*(*3*−*4*)
by *auto*
note *startfin* = *this*[*unfolded mem-interval-cart forall-2*]
let *?P1* = *linepath* (*vector*[*a*$*1* − *2*, *a*$*2* − *2*]) (*vector*[(*pathstart f*)$*1*,*a*$*2* − *2*]) +++
linepath(*vector*[(*pathstart f*)$*1*,*a*$*2* − *2*])(*pathstart f*) +++ *f* +++
linepath(*pathfinish f*)(*vector*[(*pathfinish f*)$*1*,*a*$*2* − *2*]) +++
linepath(*vector*[(*pathfinish f*)$*1*,*a*$*2* − *2*])(*vector*[*b*$*1* + *2*,*a*$*2* − *2*])
let *?P2* = *linepath*(*vector*[(*pathstart g*)$*1*, (*pathstart g*)$*2* − *3*])(*pathstart g*) +++ *g* +++
linepath(*pathfinish g*)(*vector*[(*pathfinish g*)$*1*,*a*$*2* − *1*]) +++
linepath(*vector*[(*pathfinish g*)$*1*,*a*$*2* − *1*])(*vector*[*b*$*1* + *1*,*a*$*2* − *1*]) +++
linepath(*vector*[*b*$*1* + *1*,*a*$*2* − *1*])(*vector*[*b*$*1* + *1*,*b*$*2* + *3*])
let *?a* = *vector*[*a*$*1* − *2*, *a*$*2* − *3*]
let *?b* = *vector*[*b*$*1* + *2*, *b*$*2* + *3*]
have *P1P2*: *path-image ?P1* = *path-image* (*linepath* (*vector*[*a*$*1* − *2*, *a*$*2* − *2*]) (*vector*[(*pathstart f*)$*1*,*a*$*2* − *2*])) ∪
path-image (*linepath*(*vector*[(*pathstart f*)$*1*,*a*$*2* − *2*])(*pathstart f*)) ∪ *path-image f* ∪
path-image (*linepath*(*pathfinish f*)(*vector*[(*pathfinish f*)$*1*,*a*$*2* − *2*])) ∪
path-image (*linepath*(*vector*[(*pathfinish f*)$*1*,*a*$*2* − *2*])(*vector*[*b*$*1* + *2*,*a*$*2* − *2*]))
path-image ?P2 = *path-image*(*linepath*(*vector*[(*pathstart g*)$*1*, (*pathstart g*)$*2* − *3*])(*pathstart g*)) ∪ *path-image g* ∪

```
      path-image(linepath(pathfinish g)(vector[(pathfinish g)$1,a$2 - 1])) ∪
     path-image(linepath(vector[(pathfinish g)$1,a$2 - 1])(vector[b$1 + 1,a$2 -
1])) ∪
      path-image(linepath(vector[b$1 + 1,a$2 - 1])(vector[b$1 + 1,b$2 + 3]))
using assms(1-2)
     by(auto simp add: path-image-join path-linepath)
  have abab: cbox a b ⊆ cbox ?a ?b
    unfolding interval-cbox-cart[symmetric]
    by (auto simp add:less-eq-vec-def forall-2 vector-2)
  obtain z where
    z ∈ path-image
          (linepath (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f $ 1, a $ 2
- 2]) +++
           linepath (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f) +++
           f +++
           linepath (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) +++
           linepath (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2, a $ 2
- 2]))
    z ∈ path-image
         (linepath (vector [pathstart g $ 1, pathstart g $ 2 - 3]) (pathstart g) +++
           g +++
           linepath (pathfinish g) (vector [pathfinish g $ 1, a $ 2 - 1]) +++
           linepath (vector [pathfinish g $ 1, a $ 2 - 1]) (vector [b $ 1 + 1, a $ 2
- 1]) +++
          linepath (vector [b $ 1 + 1, a $ 2 - 1]) (vector [b $ 1 + 1, b $ 2 + 3]))
    apply (rule fashoda[of ?P1 ?P2 ?a ?b])
     unfolding pathstart-join pathfinish-join pathstart-linepath pathfinish-linepath
vector-2
  proof -
    show path ?P1 and path ?P2
      using assms by auto
    have path-image ?P1 ⊆ cbox ?a ?b
      unfolding P1P2 path-image-linepath
      apply (rule Un-least)+
      defer 3
    apply (rule-tac[1-4] convex-box(1)[unfolded convex-contains-segment,rule-format])
      unfolding mem-interval-cart forall-2 vector-2
      using ab startfin abab assms(3)
      using assms(9-)
      unfolding assms
      apply (auto simp add: field-simps box-def)
      done
    then show path-image ?P1 ⊆ cbox ?a ?b .
    have path-image ?P2 ⊆ cbox ?a ?b
      unfolding P1P2 path-image-linepath
      apply (rule Un-least)+
      defer 2
    apply (rule-tac[1-4] convex-box(1)[unfolded convex-contains-segment,rule-format])
      unfolding mem-interval-cart forall-2 vector-2
```

```
    using ab startfin abab assms(4)
    using assms(9-)
    unfolding assms
    apply (auto simp add: field-simps box-def)
    done
  then show path-image ?P2 ⊆ cbox ?a ?b .
  show a $ 1 - 2 = a $ 1 - 2
    and b $ 1 + 2 = b $ 1 + 2
    and pathstart g $ 2 - 3 = a $ 2 - 3
    and b $ 2 + 3 = b $ 2 + 3
    by (auto simp add: assms)
 qed
 note z=this[unfolded P1P2 path-image-linepath]
 show thesis
  apply (rule that[of z])
 proof -
  have (z ∈ closed-segment (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f
$ 1, a $ 2 - 2]) ∨
    z ∈ closed-segment (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f)) ∨
    z ∈ closed-segment (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) ∨
    z ∈ closed-segment (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2,
a $ 2 - 2]) ⟹
  (((z ∈ closed-segment (vector [pathstart g $ 1, pathstart g $ 2 - 3]) (pathstart
g)) ∨
    z ∈ closed-segment (pathfinish g) (vector [pathfinish g $ 1, a $ 2 - 1])) ∨
    z ∈ closed-segment (vector [pathfinish g $ 1, a $ 2 - 1]) (vector [b $ 1 + 1,
a $ 2 - 1])) ∨
    z ∈ closed-segment (vector [b $ 1 + 1, a $ 2 - 1]) (vector [b $ 1 + 1, b $
2 + 3]) ⟹ False
  proof (simp only: segment-vertical segment-horizontal vector-2, goal-cases)
    case prems: 1
    have pathfinish f ∈ cbox a b
      using assms(3) pathfinish-in-path-image[of f] by auto
    then have 1 + b $ 1 ≤ pathfinish f $ 1 ⟹ False
      unfolding mem-interval-cart forall-2 by auto
    then have z$1 ≠ pathfinish f$1
      using prems(2)
      using assms ab
      by (auto simp add: field-simps)
    moreover have pathstart f ∈ cbox a b
      using assms(3) pathstart-in-path-image[of f]
      by auto
    then have 1 + b $ 1 ≤ pathstart f $ 1 ⟹ False
      unfolding mem-interval-cart forall-2
      by auto
    then have z$1 ≠ pathstart f$1
      using prems(2) using assms ab
      by (auto simp add: field-simps)
    ultimately have *: z$2 = a$2 - 2
```

```
      using prems(1)
      by auto
    have z$1 ≠ pathfinish g$1
      using prems(2)
      using assms ab
      by (auto simp add: field-simps *)
    moreover have pathstart g ∈ cbox a b
      using assms(4) pathstart-in-path-image[of g]
      by auto
    note this[unfolded mem-interval-cart forall-2]
    then have z$1 ≠ pathstart g$1
      using prems(1)
      using assms ab
      by (auto simp add: field-simps *)
    ultimately have a $ 2 - 1 ≤ z $ 2 ∧ z $ 2 ≤ b $ 2 + 3 ∨ b $ 2 + 3 ≤ z
$ 2 ∧ z $ 2 ≤ a $ 2 - 1
      using prems(2)
      unfolding * assms
      by (auto simp add: field-simps)
    then show False
      unfolding * using ab by auto
  qed
  then have z ∈ path-image f ∨ z ∈ path-image g
    using z unfolding Un-iff by blast
  then have z': z ∈ cbox a b
    using assms(3-4)
    by auto
  have a $ 2 = z $ 2 ⟹ (z $ 1 = pathstart f $ 1 ∨ z $ 1 = pathfinish f $ 1)
⟹
    z = pathstart f ∨ z = pathfinish f
    unfolding vec-eq-iff forall-2 assms
    by auto
  with z' show z ∈ path-image f
    using z(1)
    unfolding Un-iff mem-interval-cart forall-2
    apply -
    apply (simp only: segment-vertical segment-horizontal vector-2)
    unfolding assms
    apply auto
    done
  have a $ 2 = z $ 2 ⟹ (z $ 1 = pathstart g $ 1 ∨ z $ 1 = pathfinish g $ 1)
⟹
    z = pathstart g ∨ z = pathfinish g
    unfolding vec-eq-iff forall-2 assms
    by auto
  with z' show z ∈ path-image g
    using z(2)
    unfolding Un-iff mem-interval-cart forall-2
    apply (simp only: segment-vertical segment-horizontal vector-2)
```

unfolding *assms*
apply *auto*
done
qed
qed

end

38 Continuity and iterations (of set transformers)

theory *Order-Continuity*
imports *Complex-Main*
begin

lemma *SUP-nat-binary*:
(*SUP n*::*nat*. *if n = 0 then A else B*) = (*sup A B*::'*a*::*complete-lattice*)
apply (*auto intro*!: *antisym SUP-least*)
apply (*rule SUP-upper2*[**where** *i=0*])
apply *simp-all*
apply (*rule SUP-upper2*[**where** *i=1*])
apply *simp-all*
done

lemma *INF-nat-binary*:
(*INF n*::*nat*. *if n = 0 then A else B*) = (*inf A B*::'*a*::*complete-lattice*)
apply (*auto intro*!: *antisym INF-greatest*)
apply (*rule INF-lower2*[**where** *i=0*])
apply *simp-all*
apply (*rule INF-lower2*[**where** *i=1*])
apply *simp-all*
done

The name *continuous* is already taken in *Complex-Main*, so we use *sup-continuous* and *inf-continuous*. These names appear sometimes in literature and have the advantage that these names are duals.

named-theorems *order-continuous-intros*

38.1 Continuity for complete lattices

definition
sup-continuous :: ('*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*) ⇒ *bool* **where**
sup-continuous F ⟷ (∀ *M*::*nat* ⇒ '*a*. *mono M* ⟶ *F* (*SUP i*. *M i*) = (*SUP i*. *F* (*M i*)))

lemma *sup-continuousD*: *sup-continuous F* $\Longrightarrow$ *mono M* $\Longrightarrow$ *F* (*SUP i*::*nat*. *M i*) = (*SUP i*. *F* (*M i*))
by (*auto simp*: *sup-continuous-def*)

lemma *sup-continuous-mono*:
assumes [*simp*]: *sup-continuous F* **shows** *mono F*
proof
fix *A B* :: *'a* **assume** [*simp*]: *A* $\leq$ *B*
have *F B* = *F* (*SUP n*::*nat*. *if n* = *0 then A else B*)
by (*simp add*: *sup-absorb2 SUP-nat-binary*)
also have ... = (*SUP n*::*nat*. *if n* = *0 then F A else F B*)
by (*auto simp*: *sup-continuousD mono-def intro*!: *SUP-cong*)
finally show *F A* $\leq$ *F B*
by (*simp add*: *SUP-nat-binary le-iff-sup*)
qed

lemma [*order-continuous-intros*]:
shows *sup-continuous-const*: *sup-continuous* (λx. *c*)
and *sup-continuous-id*: *sup-continuous* (λx. *x*)
and *sup-continuous-apply*: *sup-continuous* (λf. *f x*)
and *sup-continuous-fun*: ($\bigwedge s$. *sup-continuous* (λx. *P x s*)) $\Longrightarrow$ *sup-continuous P*
and *sup-continuous-If*: *sup-continuous F* $\Longrightarrow$ *sup-continuous G* $\Longrightarrow$ *sup-continuous* (λf. *if C then F f else G f*)
by (*auto simp*: *sup-continuous-def*)

lemma *sup-continuous-compose*:
assumes *f*: *sup-continuous f* **and** *g*: *sup-continuous g*
shows *sup-continuous* (λx. *f* (*g x*))
unfolding *sup-continuous-def*
proof *safe*
fix *M* :: *nat* $\Rightarrow$ *'c* **assume** *mono M*
moreover then have *mono* (λi. *g* (*M i*))
using *sup-continuous-mono*[*OF g*] **by** (*auto simp*: *mono-def*)
ultimately show *f* (*g* (*SUPREMUM UNIV M*)) = (*SUP i*. *f* (*g* (*M i*)))
by (*auto simp*: *sup-continuous-def g*[*THEN sup-continuousD*] *f*[*THEN sup-continuousD*])
qed

lemma *sup-continuous-sup*[*order-continuous-intros*]:
sup-continuous f $\Longrightarrow$ *sup-continuous g* $\Longrightarrow$ *sup-continuous* (λx. *sup* (*f x*) (*g x*))
by (*simp add*: *sup-continuous-def SUP-sup-distrib*)

lemma *sup-continuous-inf*[*order-continuous-intros*]:
fixes *P Q* :: *'a* :: *complete-lattice* $\Rightarrow$ *'b* :: *complete-distrib-lattice*
assumes *P*: *sup-continuous P* **and** *Q*: *sup-continuous Q*
shows *sup-continuous* (λx. *inf* (*P x*) (*Q x*))
unfolding *sup-continuous-def*
proof (*safe intro*!: *antisym*)
fix *M* :: *nat* $\Rightarrow$ *'a* **assume** *M*: *incseq M*

```
  have inf (P (SUP i. M i)) (Q (SUP i. M i)) ≤ (SUP j i. inf (P (M i)) (Q (M
j)))
    unfolding sup-continuousD[OF P M] sup-continuousD[OF Q M] inf-SUP
SUP-inf ..
  also have ... ≤ (SUP i. inf (P (M i)) (Q (M i)))
  proof (intro SUP-least)
    fix i j from M assms[THEN sup-continuous-mono] show inf (P (M i)) (Q
(M j)) ≤ (SUP i. inf (P (M i)) (Q (M i)))
      by (intro SUP-upper2[of sup i j] inf-mono) (auto simp: mono-def)
  qed
  finally show inf (P (SUP i. M i)) (Q (SUP i. M i)) ≤ (SUP i. inf (P (M i))
(Q (M i))) .

  show (SUP i. inf (P (M i)) (Q (M i))) ≤ inf (P (SUP i. M i)) (Q (SUP i. M
i))
    unfolding sup-continuousD[OF P M] sup-continuousD[OF Q M] by (intro
SUP-least inf-mono SUP-upper)
qed

lemma sup-continuous-and[order-continuous-intros]:
  sup-continuous P ⟹ sup-continuous Q ⟹ sup-continuous (λx. P x ∧ Q x)
  using sup-continuous-inf[of P Q] by simp

lemma sup-continuous-or[order-continuous-intros]:
  sup-continuous P ⟹ sup-continuous Q ⟹ sup-continuous (λx. P x ∨ Q x)
  by (auto simp: sup-continuous-def)

lemma sup-continuous-lfp:
  assumes sup-continuous F shows lfp F = (SUP i. (F ^^ i) bot) (is lfp F = ?U)
proof (rule antisym)
  note mono = sup-continuous-mono[OF ‹sup-continuous F›]
  show ?U ≤ lfp F
  proof (rule SUP-least)
    fix i show (F ^^ i) bot ≤ lfp F
    proof (induct i)
      case (Suc i)
      have (F ^^ Suc i) bot = F ((F ^^ i) bot) by simp
      also have ... ≤ F (lfp F) by (rule monoD[OF mono Suc])
      also have ... = lfp F by (simp add: lfp-unfold[OF mono, symmetric])
      finally show ?case .
    qed simp
  qed
  show lfp F ≤ ?U
  proof (rule lfp-lowerbound)
    have mono (λi::nat. (F ^^ i) bot)
    proof −
      { fix i::nat have (F ^^ i) bot ≤ (F ^^ (Suc i)) bot
        proof (induct i)
          case 0 show ?case by simp
```

```
      next
        case Suc thus ?case using monoD[OF mono Suc] by auto
      qed }
    thus ?thesis by (auto simp add: mono-iff-le-Suc)
  qed
  hence F ?U = (SUP i. (F ^^ Suc i) bot)
    using ⟨sup-continuous F⟩ by (simp add: sup-continuous-def)
  also have ... ≤ ?U
    by (fast intro: SUP-least SUP-upper)
  finally show F ?U ≤ ?U .
 qed
qed

lemma lfp-transfer-bounded:
 assumes P: P bot ⋀x. P x ⟹ P (f x) ⋀M. (⋀i. P (M i)) ⟹ P (SUP i::nat.
M i)
 assumes α: ⋀M. mono M ⟹ (⋀i::nat. P (M i)) ⟹ α (SUP i. M i) = (SUP
i. α (M i))
 assumes f: sup-continuous f and g: sup-continuous g
 assumes [simp]: ⋀x. P x ⟹ x ≤ lfp f ⟹ α (f x) = g (α x)
 assumes g-bound: ⋀x. α bot ≤ g x
 shows α (lfp f) = lfp g
proof (rule antisym)
 note mono-g = sup-continuous-mono[OF g]
 note mono-f = sup-continuous-mono[OF f]
 have lfp-bound: α bot ≤ lfp g
   by (subst lfp-unfold[OF mono-g]) (rule g-bound)

 have P-pow: P ((f ^^ i) bot) for i
   by (induction i) (auto intro!: P)
 have incseq-pow: mono (λi. (f ^^ i) bot)
   unfolding mono-iff-le-Suc
 proof
   fix i show (f ^^ i) bot ≤ (f ^^ (Suc i)) bot
   proof (induct i)
     case Suc thus ?case using monoD[OF sup-continuous-mono[OF f] Suc] by
auto
   qed (simp add: le-fun-def)
 qed
 have P-lfp: P (lfp f)
   using P-pow unfolding sup-continuous-lfp[OF f] by (auto intro!: P)

 have iter-le-lfp: (f ^^ n) bot ≤ lfp f for n
   apply (induction n)
   apply simp
   apply (subst lfp-unfold[OF mono-f])
   apply (auto intro!: monoD[OF mono-f])
   done
```

have $\alpha\ (lfp\ f) = (SUP\ i.\ \alpha\ ((f\hat{}\hat{}i)\ bot))$
unfolding *sup-continuous-lfp*[*OF f*] **using** *incseq-pow P-pow* **by** (*rule* α)
also have $\ldots \leq lfp\ g$
proof (*rule SUP-least*)
fix *i* **show** $\alpha\ ((f\hat{}\hat{}i)\ bot) \leq lfp\ g$
proof (*induction i*)
case (*Suc n*) **then show** *?case*
by (*subst lfp-unfold*[*OF mono-g*]) (*simp add*: *monoD*[*OF mono-g*] *P-pow iter-le-lfp*)
qed (*simp add*: *lfp-bound*)
qed
finally show $\alpha\ (lfp\ f) \leq lfp\ g$ **.**

show $lfp\ g \leq \alpha\ (lfp\ f)$
proof (*induction rule*: *lfp-ordinal-induct*[*OF mono-g*])
case (*1 S*) **then show** *?case*
by (*subst lfp-unfold*[*OF sup-continuous-mono*[*OF f*]])
(*simp add*: *monoD*[*OF mono-g*] *P-lfp*)
qed (*auto intro*: *Sup-least*)
qed

lemma *lfp-transfer*:
sup-continuous $\alpha \Longrightarrow$ *sup-continuous f* $\Longrightarrow$ *sup-continuous g* $\Longrightarrow$
$(\bigwedge x.\ \alpha\ bot \leq g\ x) \Longrightarrow (\bigwedge x.\ x \leq lfp\ f \Longrightarrow \alpha\ (f\ x) = g\ (\alpha\ x)) \Longrightarrow \alpha\ (lfp\ f) = lfp\ g$
by (*rule lfp-transfer-bounded*[**where** *P*=*top*]) (*auto dest*: *sup-continuousD*)

definition
inf-continuous :: ('*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*) ⇒ *bool* **where**
inf-continuous F $\longleftrightarrow$ (∀ *M*::*nat* ⇒ '*a*. *antimono M* $\longrightarrow$ *F* (*INF i*. *M i*) = (*INF i*. *F* (*M i*)))

lemma *inf-continuousD*: *inf-continuous F* $\Longrightarrow$ *antimono M* $\Longrightarrow$ *F* (*INF i*::*nat*. *M i*) = (*INF i*. *F* (*M i*))
by (*auto simp*: *inf-continuous-def*)

lemma *inf-continuous-mono*:
assumes [*simp*]: *inf-continuous F* **shows** *mono F*
proof
fix *A B* :: '*a* **assume** [*simp*]: $A \leq B$
have *F A* = *F* (*INF n*::*nat*. *if n* = *0 then B else A*)
by (*simp add*: *inf-absorb2 INF-nat-binary*)
also have $\ldots$ = (*INF n*::*nat*. *if n* = *0 then F B else F A*)
by (*auto simp*: *inf-continuousD antimono-def intro*!: *INF-cong*)
finally show *F A* ≤ *F B*
by (*simp add*: *INF-nat-binary le-iff-inf inf-commute*)
qed

lemma [*order-continuous-intros*]:

shows *inf-continuous-const*: *inf-continuous* ($\lambda x.\ c$)
and *inf-continuous-id*: *inf-continuous* ($\lambda x.\ x$)
and *inf-continuous-apply*: *inf-continuous* ($\lambda f.\ f\ x$)
and *inf-continuous-fun*: $(\bigwedge s.$ *inf-continuous* $(\lambda x.\ P\ x\ s)) \Longrightarrow$ *inf-continuous* P
and *inf-continuous-If*: *inf-continuous* $F \Longrightarrow$ *inf-continuous* $G \Longrightarrow$ *inf-continuous* $(\lambda f.$ *if* C *then* $F\ f$ *else* $G\ f)$
by (*auto simp*: *inf-continuous-def*)

lemma *inf-continuous-inf*[*order-continuous-intros*]:
inf-continuous $f \Longrightarrow$ *inf-continuous* $g \Longrightarrow$ *inf-continuous* $(\lambda x.$ *inf* $(f\ x)\ (g\ x))$
by (*simp add*: *inf-continuous-def INF-inf-distrib*)

lemma *inf-continuous-sup*[*order-continuous-intros*]:
fixes $P\ Q :: {'a} ::$ *complete-lattice* $\Rightarrow {'b} ::$ *complete-distrib-lattice*
assumes P: *inf-continuous* P **and** Q: *inf-continuous* Q
shows *inf-continuous* $(\lambda x.$ *sup* $(P\ x)\ (Q\ x))$
unfolding *inf-continuous-def*
proof (*safe intro*!: *antisym*)
fix $M ::$ *nat* $\Rightarrow {'a}$ **assume** M: *decseq* M
show *sup* $(P\ (INF\ i.\ M\ i))\ (Q\ (INF\ i.\ M\ i)) \leq (INF\ i.$ *sup* $(P\ (M\ i))\ (Q\ (M\ i)))$
unfolding *inf-continuousD*[*OF* $P\ M$] *inf-continuousD*[*OF* $Q\ M$] **by** (*intro INF-greatest sup-mono INF-lower*)

have $(INF\ i.$ *sup* $(P\ (M\ i))\ (Q\ (M\ i))) \leq (INF\ j\ i.$ *sup* $(P\ (M\ i))\ (Q\ (M\ j)))$
proof (*intro INF-greatest*)
fix $i\ j$ **from** M *assms*[*THEN inf-continuous-mono*] **show** *sup* $(P\ (M\ i))\ (Q\ (M\ j)) \geq (INF\ i.$ *sup* $(P\ (M\ i))\ (Q\ (M\ i)))$
by (*intro INF-lower2*[*of sup i j*] *sup-mono*) (*auto simp*: *mono-def antimono-def*)
qed
also have $\ldots \leq$ *sup* $(P\ (INF\ i.\ M\ i))\ (Q\ (INF\ i.\ M\ i))$
unfolding *inf-continuousD*[*OF P M*] *inf-continuousD*[*OF Q M*] *INF-sup sup-INF*
..
finally show *sup* $(P\ (INF\ i.\ M\ i))\ (Q\ (INF\ i.\ M\ i)) \geq (INF\ i.$ *sup* $(P\ (M\ i))\ (Q\ (M\ i)))$.
qed

lemma *inf-continuous-and*[*order-continuous-intros*]:
inf-continuous $P \Longrightarrow$ *inf-continuous* $Q \Longrightarrow$ *inf-continuous* $(\lambda x.\ P\ x \wedge Q\ x)$
using *inf-continuous-inf*[*of* $P\ Q$] **by** *simp*

lemma *inf-continuous-or*[*order-continuous-intros*]:
inf-continuous $P \Longrightarrow$ *inf-continuous* $Q \Longrightarrow$ *inf-continuous* $(\lambda x.\ P\ x \vee Q\ x)$
using *inf-continuous-sup*[*of* $P\ Q$] **by** *simp*

lemma *inf-continuous-compose*:
assumes f: *inf-continuous* f **and** g: *inf-continuous* g
shows *inf-continuous* $(\lambda x.\ f\ (g\ x))$
unfolding *inf-continuous-def*

proof *safe*
fix M :: $nat \Rightarrow 'c$ **assume** *antimono M*
moreover then have *antimono* ($\lambda i.\ g\ (M\ i)$)
using *inf-continuous-mono*[*OF g*] **by** (*auto simp*: *mono-def antimono-def*)
ultimately show $f\ (g\ (INFIMUM\ UNIV\ M)) = (INF\ i.\ f\ (g\ (M\ i)))$
by (*auto simp*: *inf-continuous-def g*[*THEN inf-continuousD*] *f*[*THEN inf-continuousD*])
qed

lemma *inf-continuous-gfp*:
assumes *inf-continuous F* **shows** *gfp F* = (*INF i.* (*F* ^^ *i*) *top*) (**is** *gfp F* = *?U*)
proof (*rule antisym*)
note *mono* = *inf-continuous-mono*[*OF* ‹*inf-continuous F*›]
show *gfp F* ≤ *?U*
proof (*rule INF-greatest*)
fix *i* **show** *gfp F* ≤ (*F* ^^ *i*) *top*
proof (*induct i*)
case (*Suc i*)
have *gfp F* = *F* (*gfp F*) **by** (*simp add*: *gfp-unfold*[*OF mono*, *symmetric*])
also have ... ≤ *F* ((*F* ^^ *i*) *top*) **by** (*rule monoD*[*OF mono Suc*])
also have ... = (*F* ^^ *Suc i*) *top* **by** *simp*
finally show *?case* **.**
qed *simp*
qed
show *?U* ≤ *gfp F*
proof (*rule gfp-upperbound*)
have ∗: *antimono* (λ*i*::*nat*. (*F* ^^ *i*) *top*)
proof −
{ fix *i*::*nat* **have** (*F* ^^ *Suc i*) *top* ≤ (*F* ^^ *i*) *top*
proof (*induct i*)
case *0* **show** *?case* **by** *simp*
next
case *Suc* **thus** *?case* **using** *monoD*[*OF mono Suc*] **by** *auto*
qed **}**
thus *?thesis* **by** (*auto simp add*: *antimono-iff-le-Suc*)
qed
have *?U* ≤ (*INF i.* (*F* ^^ *Suc i*) *top*)
by (*fast intro*: *INF-greatest INF-lower*)
also have ... ≤ *F ?U*
by (*simp add*: *inf-continuousD* ‹*inf-continuous F*› ∗)
finally show *?U* ≤ *F ?U* **.**
qed
qed

lemma *gfp-transfer*:
assumes α: *inf-continuous* α **and** *f*: *inf-continuous f* **and** *g*: *inf-continuous g*
assumes [*simp*]: α *top* = *top* ⋀*x*. α (*f x*) = *g* (α *x*)
shows α (*gfp f*) = *gfp g*
proof −
have α (*gfp f*) = (*INF i.* α ((*f*^^*i*) *top*))

unfolding *inf-continuous-gfp*[*OF f*] **by** (*intro f α inf-continuousD antimono-funpow inf-continuous-mono*)
moreover have *α* ((*f* ^^ *i*) *top*) = (*g* ^^ *i*) *top* **for** *i*
by (*induction i*; *simp*)
ultimately show *?thesis*
unfolding *inf-continuous-gfp*[*OF g*] **by** *simp*
qed

lemma *gfp-transfer-bounded*:
assumes *P*: *P* (*f top*) ⋀*x*. *P x* ⟹ *P* (*f x*) ⋀*M*. *antimono M* ⟹ (⋀*i*. *P* (*M i*)) ⟹ *P* (*INF i*::*nat*. *M i*)
assumes *α*: ⋀*M*. *antimono M* ⟹ (⋀*i*::*nat*. *P* (*M i*)) ⟹ *α* (*INF i*. *M i*) = (*INF i*. *α* (*M i*))
assumes *f*: *inf-continuous f* **and** *g*: *inf-continuous g*
assumes [*simp*]: ⋀*x*. *P x* ⟹ *α* (*f x*) = *g* (*α x*)
assumes *g-bound*: ⋀*x*. *g x* ≤ *α* (*f top*)
shows *α* (*gfp f*) = *gfp g*
proof (*rule antisym*)
note *mono-g* = *inf-continuous-mono*[*OF g*]

have *P-pow*: *P* ((*f* ^^ *i*) (*f top*)) **for** *i*
by (*induction i*) (*auto intro*!: *P*)

have *antimono-pow*: *antimono* (*λi*. (*f* ^^ *i*) *top*)
unfolding *antimono-iff-le-Suc*
proof
fix *i* **show** (*f* ^^ *Suc i*) *top* ≤ (*f* ^^ *i*) *top*
proof (*induct i*)
case *Suc* **thus** *?case* **using** *monoD*[*OF inf-continuous-mono*[*OF f*] *Suc*] **by** *auto*
qed (*simp add*: *le-fun-def*)
qed
have *antimono-pow2*: *antimono* (*λi*. (*f* ^^ *i*) (*f top*))
proof
show *x* ≤ *y* ⟹ (*f* ^^ *y*) (*f top*) ≤ (*f* ^^ *x*) (*f top*) **for** *x y*
using *antimono-pow*[*THEN antimonoD*, *of Suc x Suc y*]
unfolding *funpow-Suc-right* **by** *simp*
qed

have *gfp-f*: *gfp f* = (*INF i*. (*f* ^^ *i*) (*f top*))
unfolding *inf-continuous-gfp*[*OF f*]
proof (*rule INF-eq*)
show ∃*j*∈*UNIV*. (*f* ^^ *j*) (*f top*) ≤ (*f* ^^ *i*) *top* **for** *i*
by (*intro bexI*[*of - i − 1*]) (*auto simp*: *diff-Suc funpow-Suc-right simp del*: *funpow.simps*(*2*) *split*: *nat.split*)
show ∃*j*∈*UNIV*. (*f* ^^ *j*) *top* ≤ (*f* ^^ *i*) (*f top*) **for** *i*
by (*intro bexI*[*of - Suc i*]) (*auto simp*: *funpow-Suc-right simp del*: *funpow.simps*(*2*))
qed

```
  have P-lfp: P (gfp f)
    unfolding gfp-f by (auto intro!: P P-pow antimono-pow2)

  have α (gfp f) = (INF i. α ((f^^i) (f top)))
    unfolding gfp-f by (rule α) (auto intro!: P-pow antimono-pow2)
  also have ... ≥ gfp g
  proof (rule INF-greatest)
    fix i show gfp g ≤ α ((f^^i) (f top))
    proof (induction i)
      case (Suc n) then show ?case
        by (subst gfp-unfold[OF mono-g]) (simp add: monoD[OF mono-g] P-pow)
    next
      case 0
      have gfp g ≤ α (f top)
        by (subst gfp-unfold[OF mono-g]) (rule g-bound)
      then show ?case
        by simp
    qed
  qed
  finally show gfp g ≤ α (gfp f) .

  show α (gfp f) ≤ gfp g
  proof (induction rule: gfp-ordinal-induct[OF mono-g])
    case (1 S) then show ?case
      by (subst gfp-unfold[OF inf-continuous-mono[OF f]])
        (simp add: monoD[OF mono-g] P-lfp)
  qed (auto intro: Inf-greatest)
qed

end
```

39 Extended natural numbers (i.e. with infinity)

```
theory Extended-Nat
imports Main Countable Order-Continuity
begin

class infinity =
  fixes infinity :: 'a  (∞)
```

39.1 Type definition

We extend the standard natural numbers by a special value indicating infinity.

```
typedef enat = UNIV :: nat option set ..
```

TODO: introduce enat as coinductive datatype, enat is just *of-nat*

definition *enat* :: *nat* ⇒ *enat* **where**
enat n = *Abs-enat* (*Some n*)

instantiation *enat* :: *infinity*
begin

definition ∞ = *Abs-enat None*
instance ..

end

instance *enat* :: *countable*
proof
show ∃ *to-nat*::*enat* ⇒ *nat*. *inj to-nat*
by (*rule exI*[*of* - *to-nat* ∘ *Rep-enat*]) (*simp add*: *inj-on-def Rep-enat-inject*)
qed

old-rep-datatype *enat* ∞ :: *enat*
proof −
fix *P i* **assume** ⋀*j*. *P* (*enat j*) *P* ∞
then show *P i*
proof *induct*
case (*Abs-enat y*) **then show** *?case*
by (*cases y rule*: *option.exhaust*)
(*auto simp*: *enat-def infinity-enat-def*)
qed
qed (*auto simp add*: *enat-def infinity-enat-def Abs-enat-inject*)

declare [[*coercion enat*::*nat*⇒*enat*]]

lemmas *enat2-cases* = *enat.exhaust*[*case-product enat.exhaust*]
lemmas *enat3-cases* = *enat.exhaust*[*case-product enat.exhaust enat.exhaust*]

lemma *not-infinity-eq* [*iff*]: (*x* ≠ ∞) = (∃ *i*. *x* = *enat i*)
by (*cases x*) *auto*

lemma *not-enat-eq* [*iff*]: (∀ *y*. *x* ≠ *enat y*) = (*x* = ∞)
by (*cases x*) *auto*

primrec *the-enat* :: *enat* ⇒ *nat*
where *the-enat* (*enat n*) = *n*

39.2 Constructors and numbers

instantiation *enat* :: {*zero*, *one*}
begin

definition
0 = *enat 0*

definition
1 = enat 1

instance ..

end

definition *eSuc* :: *enat* ⇒ *enat* **where**
eSuc i = (*case i of enat n* ⇒ *enat* (*Suc n*) | ∞ ⇒ ∞)

lemma *enat-0* [*code-post*]: *enat 0 = 0*
by (*simp add*: *zero-enat-def*)

lemma *enat-1* [*code-post*]: *enat 1 = 1*
by (*simp add*: *one-enat-def*)

lemma *enat-0-iff*: *enat x = 0* ⟷ *x = 0 0 = enat x* ⟷ *x = 0*
by (*auto simp add*: *zero-enat-def*)

lemma *enat-1-iff*: *enat x = 1* ⟷ *x = 1 1 = enat x* ⟷ *x = 1*
by (*auto simp add*: *one-enat-def*)

lemma *one-eSuc*: *1 = eSuc 0*
by (*simp add*: *zero-enat-def one-enat-def eSuc-def*)

lemma *infinity-ne-i0* [*simp*]: (∞::*enat*) ≠ *0*
by (*simp add*: *zero-enat-def*)

lemma *i0-ne-infinity* [*simp*]: *0* ≠ (∞::*enat*)
by (*simp add*: *zero-enat-def*)

lemma *zero-one-enat-neq* [*simp*]:
¬ *0 =* (*1*::*enat*)
¬ *1 =* (*0*::*enat*)
unfolding *zero-enat-def one-enat-def* **by** *simp-all*

lemma *infinity-ne-i1* [*simp*]: (∞::*enat*) ≠ *1*
by (*simp add*: *one-enat-def*)

lemma *i1-ne-infinity* [*simp*]: *1* ≠ (∞::*enat*)
by (*simp add*: *one-enat-def*)

lemma *eSuc-enat*: *eSuc* (*enat n*) = *enat* (*Suc n*)
by (*simp add*: *eSuc-def*)

lemma *eSuc-infinity* [*simp*]: *eSuc* ∞ = ∞
by (*simp add*: *eSuc-def*)

lemma *eSuc-ne-0* [*simp*]: *eSuc n* ≠ *0*
by (*simp add*: *eSuc-def zero-enat-def split*: *enat.splits*)

lemma *zero-ne-eSuc* [*simp*]: *0* ≠ *eSuc n*
by (*rule eSuc-ne-0* [*symmetric*])

lemma *eSuc-inject* [*simp*]: *eSuc m* = *eSuc n* ⟷ *m* = *n*
by (*simp add*: *eSuc-def split*: *enat.splits*)

lemma *eSuc-enat-iff*: *eSuc x* = *enat y* ⟷ (∃ *n*. *y* = *Suc n* ∧ *x* = *enat n*)
by (*cases y*) (*auto simp*: *enat-0 eSuc-enat*[*symmetric*])

lemma *enat-eSuc-iff*: *enat y* = *eSuc x* ⟷ (∃ *n*. *y* = *Suc n* ∧ *enat n* = *x*)
by (*cases y*) (*auto simp*: *enat-0 eSuc-enat*[*symmetric*])

39.3 Addition

instantiation *enat* :: *comm-monoid-add*
begin

definition [*nitpick-simp*]:
m + *n* = (*case m of* ∞ ⇒ ∞ | *enat m* ⇒ (*case n of* ∞ ⇒ ∞ | *enat n* ⇒ *enat* (*m* + *n*)))

lemma *plus-enat-simps* [*simp*, *code*]:
fixes *q* :: *enat*
shows *enat m* + *enat n* = *enat* (*m* + *n*)
and ∞ + *q* = ∞
and *q* + ∞ = ∞
by (*simp-all add*: *plus-enat-def split*: *enat.splits*)

instance
proof
fix *n m q* :: *enat*
show *n* + *m* + *q* = *n* + (*m* + *q*)
by (*cases n m q rule*: *enat3-cases*) *auto*
show *n* + *m* = *m* + *n*
by (*cases n m rule*: *enat2-cases*) *auto*
show *0* + *n* = *n*
by (*cases n*) (*simp-all add*: *zero-enat-def*)
qed

end

lemma *eSuc-plus-1*:
eSuc n = *n* + *1*
by (*cases n*) (*simp-all add*: *eSuc-enat one-enat-def*)

lemma *plus-1-eSuc*:

1 + q = eSuc q
q + 1 = eSuc q
by (*simp-all add*: *eSuc-plus-1 ac-simps*)

lemma *iadd-Suc*: *eSuc m + n = eSuc (m + n)*
by (*simp-all add*: *eSuc-plus-1 ac-simps*)

lemma *iadd-Suc-right*: *m + eSuc n = eSuc (m + n)*
by (*simp only*: *add.commute*[*of m*] *iadd-Suc*)

lemma *iadd-is-0*: *(m + n = (0::enat)) = (m = 0 ∧ n = 0)*
by (*cases m*, *cases n*, *simp-all add*: *zero-enat-def*)

39.4 Multiplication

instantiation *enat* :: *comm-semiring-1*
begin

definition *times-enat-def* [*nitpick-simp*]:
*m * n = (case m of ∞ ⇒ if n = 0 then 0 else ∞ | enat m ⇒*
*(case n of ∞ ⇒ if m = 0 then 0 else ∞ | enat n ⇒ enat (m * n)))*

lemma *times-enat-simps* [*simp*, *code*]:
*enat m * enat n = enat (m * n)*
*∞ * ∞ = (∞::enat)*
*∞ * enat n = (if n = 0 then 0 else ∞)*
*enat m * ∞ = (if m = 0 then 0 else ∞)*
unfolding *times-enat-def zero-enat-def*
by (*simp-all split*: *enat.split*)

instance
proof
fix *a b c* :: *enat*
show *(a * b) * c = a * (b * c)*
unfolding *times-enat-def zero-enat-def*
by (*simp split*: *enat.split*)
show *a * b = b * a*
unfolding *times-enat-def zero-enat-def*
by (*simp split*: *enat.split*)
show *1 * a = a*
unfolding *times-enat-def zero-enat-def one-enat-def*
by (*simp split*: *enat.split*)
show *(a + b) * c = a * c + b * c*
unfolding *times-enat-def zero-enat-def*
by (*simp split*: *enat.split add*: *distrib-right*)
show *0 * a = 0*
unfolding *times-enat-def zero-enat-def*
by (*simp split*: *enat.split*)
show *a * 0 = 0*

unfolding *times-enat-def zero-enat-def*
by (*simp split*: *enat.split*)
show (*0*::*enat*) ≠ *1*
unfolding *zero-enat-def one-enat-def*
by *simp*
qed

end

lemma *mult-eSuc*: *eSuc m* * *n* = *n* + *m* * *n*
unfolding *eSuc-plus-1* **by** (*simp add*: *algebra-simps*)

lemma *mult-eSuc-right*: *m* * *eSuc n* = *m* + *m* * *n*
unfolding *eSuc-plus-1* **by** (*simp add*: *algebra-simps*)

lemma *of-nat-eq-enat*: *of-nat n* = *enat n*
apply (*induct n*)
apply (*simp add*: *enat-0*)
apply (*simp add*: *plus-1-eSuc eSuc-enat*)
done

instance *enat* :: *semiring-char-0*
proof
have *inj enat* **by** (*rule injI*) *simp*
then show *inj* (λ*n*. *of-nat n* :: *enat*) **by** (*simp add*: *of-nat-eq-enat*)
qed

lemma *imult-is-0* [*simp*]: ((*m*::*enat*) * *n* = *0*) = (*m* = *0* ∨ *n* = *0*)
by (*auto simp add*: *times-enat-def zero-enat-def split*: *enat.split*)

lemma *imult-is-infinity*: ((*a*::*enat*) * *b* = ∞) = (*a* = ∞ ∧ *b* ≠ *0* ∨ *b* = ∞ ∧ *a* ≠ *0*)
by (*auto simp add*: *times-enat-def zero-enat-def split*: *enat.split*)

39.5 Numerals

lemma *numeral-eq-enat*:
numeral k = *enat* (*numeral k*)
using *of-nat-eq-enat* [*of numeral k*] **by** *simp*

lemma *enat-numeral* [*code-abbrev*]:
enat (*numeral k*) = *numeral k*
using *numeral-eq-enat* **..**

lemma *infinity-ne-numeral* [*simp*]: (∞::*enat*) ≠ *numeral k*
by (*simp add*: *numeral-eq-enat*)

lemma *numeral-ne-infinity* [*simp*]: *numeral k* ≠ (∞::*enat*)
by (*simp add*: *numeral-eq-enat*)

lemma *eSuc-numeral* [*simp*]: *eSuc* (*numeral k*) = *numeral* (*k* + *Num.One*)
by (*simp only*: *eSuc-plus-1 numeral-plus-one*)

39.6 Subtraction

instantiation *enat* :: *minus*
begin

definition *diff-enat-def*:
a − *b* = (*case a of* (*enat x*) ⇒ (*case b of* (*enat y*) ⇒ *enat* (*x* − *y*) | ∞ ⇒ *0*)
| ∞ ⇒ ∞)

instance ..

end

lemma *idiff-enat-enat* [*simp*, *code*]: *enat a* − *enat b* = *enat* (*a* − *b*)
by (*simp add*: *diff-enat-def*)

lemma *idiff-infinity* [*simp*, *code*]: ∞ − *n* = (∞::*enat*)
by (*simp add*: *diff-enat-def*)

lemma *idiff-infinity-right* [*simp*, *code*]: *enat a* − ∞ = *0*
by (*simp add*: *diff-enat-def*)

lemma *idiff-0* [*simp*]: (*0*::*enat*) − *n* = *0*
by (*cases n*, *simp-all add*: *zero-enat-def*)

lemmas *idiff-enat-0* [*simp*] = *idiff-0* [*unfolded zero-enat-def*]

lemma *idiff-0-right* [*simp*]: (*n*::*enat*) − *0* = *n*
by (*cases n*) (*simp-all add*: *zero-enat-def*)

lemmas *idiff-enat-0-right* [*simp*] = *idiff-0-right* [*unfolded zero-enat-def*]

lemma *idiff-self* [*simp*]: *n* ≠ ∞ ⟹ (*n*::*enat*) − *n* = *0*
by (*auto simp*: *zero-enat-def*)

lemma *eSuc-minus-eSuc* [*simp*]: *eSuc n* − *eSuc m* = *n* − *m*
by (*simp add*: *eSuc-def split*: *enat.split*)

lemma *eSuc-minus-1* [*simp*]: *eSuc n* − *1* = *n*
by (*simp add*: *one-enat-def eSuc-enat*[*symmetric*] *zero-enat-def*[*symmetric*])

39.7 Ordering

instantiation *enat* :: *linordered-ab-semigroup-add*
begin

definition [*nitpick-simp*]:
$m \leq n = (\mathit{case}\ n\ \mathit{of}\ \mathit{enat}\ n1 \Rightarrow (\mathit{case}\ m\ \mathit{of}\ \mathit{enat}\ m1 \Rightarrow m1 \leq n1 \mid \infty \Rightarrow \mathit{False}) \mid \infty \Rightarrow \mathit{True})$

definition [*nitpick-simp*]:
$m < n = (\mathit{case}\ m\ \mathit{of}\ \mathit{enat}\ m1 \Rightarrow (\mathit{case}\ n\ \mathit{of}\ \mathit{enat}\ n1 \Rightarrow m1 < n1 \mid \infty \Rightarrow \mathit{True}) \mid \infty \Rightarrow \mathit{False})$

lemma *enat-ord-simps* [*simp*]:
$\mathit{enat}\ m \leq \mathit{enat}\ n \longleftrightarrow m \leq n$
$\mathit{enat}\ m < \mathit{enat}\ n \longleftrightarrow m < n$
$q \leq (\infty::\mathit{enat})$
$q < (\infty::\mathit{enat}) \longleftrightarrow q \neq \infty$
$(\infty::\mathit{enat}) \leq q \longleftrightarrow q = \infty$
$(\infty::\mathit{enat}) < q \longleftrightarrow \mathit{False}$
by (*simp-all add*: *less-eq-enat-def less-enat-def split*: *enat.splits*)

lemma *numeral-le-enat-iff* [*simp*]:
shows $\mathit{numeral}\ m \leq \mathit{enat}\ n \longleftrightarrow \mathit{numeral}\ m \leq n$
by (*auto simp*: *numeral-eq-enat*)

lemma *numeral-less-enat-iff* [*simp*]:
shows $\mathit{numeral}\ m < \mathit{enat}\ n \longleftrightarrow \mathit{numeral}\ m < n$
by (*auto simp*: *numeral-eq-enat*)

lemma *enat-ord-code* [*code*]:
$\mathit{enat}\ m \leq \mathit{enat}\ n \longleftrightarrow m \leq n$
$\mathit{enat}\ m < \mathit{enat}\ n \longleftrightarrow m < n$
$q \leq (\infty::\mathit{enat}) \longleftrightarrow \mathit{True}$
$\mathit{enat}\ m < \infty \longleftrightarrow \mathit{True}$
$\infty \leq \mathit{enat}\ n \longleftrightarrow \mathit{False}$
$(\infty::\mathit{enat}) < q \longleftrightarrow \mathit{False}$
by *simp-all*

instance
by *standard* (*auto simp add*: *less-eq-enat-def less-enat-def plus-enat-def split*: *enat.splits*)

end

instance *enat* :: *ordered-comm-semiring*
proof
fix $a\ b\ c$ :: *enat*
assume $a \leq b$ **and** $0 \leq c$
thus $c * a \leq c * b$
unfolding *times-enat-def less-eq-enat-def zero-enat-def*
by (*simp split*: *enat.splits*)
qed

lemma *enat-ord-number* [*simp*]:
(*numeral m* :: *enat*) ≤ *numeral n* ⟷ (*numeral m* :: *nat*) ≤ *numeral n*
(*numeral m* :: *enat*) < *numeral n* ⟷ (*numeral m* :: *nat*) < *numeral n*
by (*simp-all add*: *numeral-eq-enat*)

lemma *i0-lb* [*simp*]: (*0*::*enat*) ≤ *n*
by (*simp add*: *zero-enat-def less-eq-enat-def split*: *enat.splits*)

lemma *ile0-eq* [*simp*]: *n* ≤ (*0*::*enat*) ⟷ *n* = *0*
by (*simp add*: *zero-enat-def less-eq-enat-def split*: *enat.splits*)

lemma *infinity-ileE* [*elim*!]: ∞ ≤ *enat m* ⟹ *R*
by (*simp add*: *zero-enat-def less-eq-enat-def split*: *enat.splits*)

lemma *infinity-ilessE* [*elim*!]: ∞ < *enat m* ⟹ *R*
by *simp*

lemma *not-iless0* [*simp*]: ¬ *n* < (*0*::*enat*)
by (*simp add*: *zero-enat-def less-enat-def split*: *enat.splits*)

lemma *i0-less* [*simp*]: (*0*::*enat*) < *n* ⟷ *n* ≠ *0*
by (*simp add*: *zero-enat-def less-enat-def split*: *enat.splits*)

lemma *eSuc-ile-mono* [*simp*]: *eSuc n* ≤ *eSuc m* ⟷ *n* ≤ *m*
by (*simp add*: *eSuc-def less-eq-enat-def split*: *enat.splits*)

lemma *eSuc-mono* [*simp*]: *eSuc n* < *eSuc m* ⟷ *n* < *m*
by (*simp add*: *eSuc-def less-enat-def split*: *enat.splits*)

lemma *ile-eSuc* [*simp*]: *n* ≤ *eSuc n*
by (*simp add*: *eSuc-def less-eq-enat-def split*: *enat.splits*)

lemma *not-eSuc-ilei0* [*simp*]: ¬ *eSuc n* ≤ *0*
by (*simp add*: *zero-enat-def eSuc-def less-eq-enat-def split*: *enat.splits*)

lemma *i0-iless-eSuc* [*simp*]: *0* < *eSuc n*
by (*simp add*: *zero-enat-def eSuc-def less-enat-def split*: *enat.splits*)

lemma *iless-eSuc0*[*simp*]: (*n* < *eSuc 0*) = (*n* = *0*)
by (*simp add*: *zero-enat-def eSuc-def less-enat-def split*: *enat.split*)

lemma *ileI1*: *m* < *n* ⟹ *eSuc m* ≤ *n*
by (*simp add*: *eSuc-def less-eq-enat-def less-enat-def split*: *enat.splits*)

lemma *Suc-ile-eq*: *enat* (*Suc m*) ≤ *n* ⟷ *enat m* < *n*
by (*cases n*) *auto*

lemma *iless-Suc-eq* [*simp*]: *enat m* < *eSuc n* ⟷ *enat m* ≤ *n*
by (*auto simp add*: *eSuc-def less-enat-def split*: *enat.splits*)

lemma *imult-infinity*: (*0*::*enat*) < *n* ⟹ ∞ ∗ *n* = ∞
by (*simp add*: *zero-enat-def less-enat-def split*: *enat.splits*)

lemma *imult-infinity-right*: (*0*::*enat*) < *n* ⟹ *n* ∗ ∞ = ∞
by (*simp add*: *zero-enat-def less-enat-def split*: *enat.splits*)

lemma *enat-0-less-mult-iff*: (*0* < (*m*::*enat*) ∗ *n*) = (*0* < *m* ∧ *0* < *n*)
by (*simp only*: *i0-less imult-is-0*, *simp*)

lemma *mono-eSuc*: *mono eSuc*
by (*simp add*: *mono-def*)

lemma *min-enat-simps* [*simp*]:
min (*enat m*) (*enat n*) = *enat* (*min m n*)
min q 0 = *0*
min 0 q = *0*
min q (∞::*enat*) = *q*
min (∞::*enat*) *q* = *q*
by (*auto simp add*: *min-def*)

lemma *max-enat-simps* [*simp*]:
max (*enat m*) (*enat n*) = *enat* (*max m n*)
max q 0 = *q*
max 0 q = *q*
max q ∞ = (∞::*enat*)
max ∞ *q* = (∞::*enat*)
by (*simp-all add*: *max-def*)

lemma *enat-ile*: *n* ≤ *enat m* ⟹ ∃ *k*. *n* = *enat k*
by (*cases n*) *simp-all*

lemma *enat-iless*: *n* < *enat m* ⟹ ∃ *k*. *n* = *enat k*
by (*cases n*) *simp-all*

lemma *iadd-le-enat-iff*:
x + *y* ≤ *enat n* ⟷ (∃ *y' x'*. *x* = *enat x'* ∧ *y* = *enat y'* ∧ *x'* + *y'* ≤ *n*)
by(*cases x y rule*: *enat.exhaust*[*case-product enat.exhaust*]) *simp-all*

lemma *chain-incr*: ∀ *i*. ∃ *j*. *Y i* < *Y j* ==> ∃ *j*. *enat k* < *Y j*
apply (*induct-tac k*)
apply (*simp* (*no-asm*) *only*: *enat-0*)
apply (*fast intro*: *le-less-trans* [*OF i0-lb*])
apply (*erule exE*)
apply (*drule spec*)
apply (*erule exE*)

apply (*drule ileI1*)
apply (*rule eSuc-enat* [*THEN subst*])
apply (*rule exI*)
apply (*erule* (*1*) *le-less-trans*)
done

lemma *eSuc-max*: *eSuc* (*max x y*) = *max* (*eSuc x*) (*eSuc y*)
by (*simp add*: *eSuc-def split*: *enat.split*)

lemma *eSuc-Max*:
assumes *finite A A* ≠ {}
shows *eSuc* (*Max A*) = *Max* (*eSuc* ‘ *A*)
using *assms* **proof** *induction*
case (*insert x A*)
thus *?case* **by**(*cases A* = {})(*simp-all add*: *eSuc-max*)
qed *simp*

instantiation *enat* :: {*order-bot*, *order-top*}
begin

definition *bot-enat* :: *enat* **where** *bot-enat* = *0*
definition *top-enat* :: *enat* **where** *top-enat* = ∞

instance
by *standard* (*simp-all add*: *bot-enat-def top-enat-def*)

end

lemma *finite-enat-bounded*:
assumes *le-fin*: ⋀*y*. *y* ∈ *A* ⟹ *y* ≤ *enat n*
shows *finite A*
proof (*rule finite-subset*)
show *finite* (*enat* ‘ {..*n*}) **by** *blast*
have *A* ⊆ {..*enat n*} **using** *le-fin* **by** *fastforce*
also have ... ⊆ *enat* ‘ {..*n*}
apply (*rule subsetI*)
subgoal for *x* **by** (*cases x*) *auto*
done
finally show *A* ⊆ *enat* ‘ {..*n*} **.**
qed

39.8 Cancellation simprocs

lemma *enat-add-left-cancel*: *a* + *b* = *a* + *c* ⟷ *a* = (∞::*enat*) ∨ *b* = *c*
unfolding *plus-enat-def* **by** (*simp split*: *enat.split*)

lemma *enat-add-left-cancel-le*: *a* + *b* ≤ *a* + *c* ⟷ *a* = (∞::*enat*) ∨ *b* ≤ *c*
unfolding *plus-enat-def* **by** (*simp split*: *enat.split*)

lemma *enat-add-left-cancel-less*: $a + b < a + c \longleftrightarrow a \neq (\infty::enat) \wedge b < c$
unfolding *plus-enat-def* **by** (*simp split*: *enat.split*)

ML ‹
```
structure Cancel-Enat-Common =
struct
  (* copied from src/HOL/Tools/nat-numeral-simprocs.ML *)
  fun find-first-t -    - []         = raise TERM(find-first-t, [])
    | find-first-t past u (t::terms) =
          if u aconv t then (rev past @ terms)
          else find-first-t (t::past) u terms

  fun dest-summing (Const (@{const-name Groups.plus}, -) $ t $ u, ts) =
        dest-summing (t, dest-summing (u, ts))
    | dest-summing (t, ts) = t :: ts

  val mk-sum = Arith-Data.long-mk-sum
  fun dest-sum t = dest-summing (t, [])
  val find-first = find-first-t []
  val trans-tac = Numeral-Simprocs.trans-tac
  val norm-ss =
    simpset-of (put-simpset HOL-basic-ss @{context}
      addsimps @{thms ac-simps add-0-left add-0-right})
  fun norm-tac ctxt = ALLGOALS (simp-tac (put-simpset norm-ss ctxt))
  fun simplify-meta-eq ctxt cancel-th th =
    Arith-Data.simplify-meta-eq [] ctxt
      ([th, cancel-th] MRS trans)
  fun mk-eq (a, b) = HOLogic.mk-Trueprop (HOLogic.mk-eq (a, b))
end

structure Eq-Enat-Cancel = ExtractCommonTermFun
(open Cancel-Enat-Common
  val mk-bal = HOLogic.mk-eq
  val dest-bal = HOLogic.dest-bin @{const-name HOL.eq} @{typ enat}
  fun simp-conv - - = SOME @{thm enat-add-left-cancel}
)

structure Le-Enat-Cancel = ExtractCommonTermFun
(open Cancel-Enat-Common
  val mk-bal = HOLogic.mk-binrel @{const-name Orderings.less-eq}
  val dest-bal = HOLogic.dest-bin @{const-name Orderings.less-eq} @{typ enat}
  fun simp-conv - - = SOME @{thm enat-add-left-cancel-le}
)

structure Less-Enat-Cancel = ExtractCommonTermFun
(open Cancel-Enat-Common
  val mk-bal = HOLogic.mk-binrel @{const-name Orderings.less}
  val dest-bal = HOLogic.dest-bin @{const-name Orderings.less} @{typ enat}
  fun simp-conv - - = SOME @{thm enat-add-left-cancel-less}
```

```
)
›

simproc-setup enat-eq-cancel
  ((l::enat) + m = n | (l::enat) = m + n) =
  ‹fn phi => fn ctxt => fn ct => Eq-Enat-Cancel.proc ctxt (Thm.term-of ct)›

simproc-setup enat-le-cancel
  ((l::enat) + m ≤ n | (l::enat) ≤ m + n) =
  ‹fn phi => fn ctxt => fn ct => Le-Enat-Cancel.proc ctxt (Thm.term-of ct)›

simproc-setup enat-less-cancel
  ((l::enat) + m < n | (l::enat) < m + n) =
  ‹fn phi => fn ctxt => fn ct => Less-Enat-Cancel.proc ctxt (Thm.term-of ct)›
```

TODO: add regression tests for these simprocs

TODO: add simprocs for combining and cancelling numerals

39.9 Well-ordering

```
lemma less-enatE:
  [| n < enat m; !!k. n = enat k ==> k < m ==> P |] ==> P
by (induct n) auto

lemma less-infinityE:
  [| n < ∞; !!k. n = enat k ==> P |] ==> P
by (induct n) auto

lemma enat-less-induct:
  assumes prem: !!n. ∀ m::enat. m < n --> P m ==> P n shows P n
proof −
  have P-enat: !!k. P (enat k)
    apply (rule nat-less-induct)
    apply (rule prem, clarify)
    apply (erule less-enatE, simp)
    done
  show ?thesis
  proof (induct n)
    fix nat
    show P (enat nat) by (rule P-enat)
  next
    show P ∞
      apply (rule prem, clarify)
      apply (erule less-infinityE)
      apply (simp add: P-enat)
      done
  qed
qed
```

instance *enat* :: *wellorder*
proof
fix *P* **and** *n*
assume *hyp*: $(\bigwedge n{::}enat.\ (\bigwedge m{::}enat.\ m < n \Longrightarrow P\ m) \Longrightarrow P\ n)$
show *P n* **by** (*blast intro*: *enat-less-induct hyp*)
qed

39.10 Complete Lattice

instantiation *enat* :: *complete-lattice*
begin

definition *inf-enat* :: *enat* ⇒ *enat* ⇒ *enat* **where**
inf-enat = *min*

definition *sup-enat* :: *enat* ⇒ *enat* ⇒ *enat* **where**
sup-enat = *max*

definition *Inf-enat* :: *enat set* ⇒ *enat* **where**
Inf-enat A = (*if A* = {} *then* ∞ *else* (*LEAST x*. *x* ∈ *A*))

definition *Sup-enat* :: *enat set* ⇒ *enat* **where**
Sup-enat A = (*if A* = {} *then 0 else if finite A then Max A else* ∞)
instance
proof
fix *x* :: *enat* **and** *A* :: *enat set*
{ assume *x* ∈ *A* **then show** *Inf A* ≤ *x*
unfolding *Inf-enat-def* **by** (*auto intro*: *Least-le*) **}**
{ assume $\bigwedge y.\ y \in A \Longrightarrow x \leq y$ **then show** *x* ≤ *Inf A*
unfolding *Inf-enat-def*
by (*cases A* = {}) (*auto intro*: *LeastI2-ex*) **}**
{ assume *x* ∈ *A* **then show** *x* ≤ *Sup A*
unfolding *Sup-enat-def* **by** (*cases finite A*) *auto* **}**
{ assume $\bigwedge y.\ y \in A \Longrightarrow y \leq x$ **then show** *Sup A* ≤ *x*
unfolding *Sup-enat-def* **using** *finite-enat-bounded* **by** *auto* **}**
qed (*simp-all add*:
inf-enat-def sup-enat-def bot-enat-def top-enat-def Inf-enat-def Sup-enat-def)
end

instance *enat* :: *complete-linorder* **..**

lemma *eSuc-Sup*: *A* ≠ {} ⟹ *eSuc* (*Sup A*) = *Sup* (*eSuc* ‘ *A*)
by(*auto simp add*: *Sup-enat-def eSuc-Max inj-on-def dest*: *finite-imageD*)

lemma *sup-continuous-eSuc*: *sup-continuous f* ⟹ *sup-continuous* (λ*x*. *eSuc* (*f x*))
using *eSuc-Sup*[*of* - ‘ *UNIV*] **by** (*auto simp*: *sup-continuous-def*)

39.11 Traditional theorem names

lemmas *enat-defs = zero-enat-def one-enat-def eSuc-def*
plus-enat-def less-eq-enat-def less-enat-def

end

40 Liminf and Limsup on complete lattices

theory *Liminf-Limsup*
imports *Complex-Main*
begin

lemma *le-Sup-iff-less*:
fixes $x :: 'a :: \{complete\text{-}linorder, dense\text{-}linorder\}$
shows $x \leq (SUP\ i{:}A.\ f\ i) \longleftrightarrow (\forall y{<}x.\ \exists i{\in}A.\ y \leq f\ i)$ (**is** *?lhs = ?rhs*)
unfolding *le-SUP-iff*
by (*blast intro*: *less-imp-le less-trans less-le-trans dest*: *dense*)

lemma *Inf-le-iff-less*:
fixes $x :: 'a :: \{complete\text{-}linorder, dense\text{-}linorder\}$
shows $(INF\ i{:}A.\ f\ i) \leq x \longleftrightarrow (\forall y{>}x.\ \exists i{\in}A.\ f\ i \leq y)$
unfolding *INF-le-iff*
by (*blast intro*: *less-imp-le less-trans le-less-trans dest*: *dense*)

lemma *SUP-pair*:
fixes $f :: - \Rightarrow - \Rightarrow -$ :: *complete-lattice*
shows $(SUP\ i : A.\ SUP\ j : B.\ f\ i\ j) = (SUP\ p : A \times B.\ f\ (fst\ p)\ (snd\ p))$
by (*rule antisym*) (*auto intro*!: *SUP-least SUP-upper2*)

lemma *INF-pair*:
fixes $f :: - \Rightarrow - \Rightarrow -$ :: *complete-lattice*
shows $(INF\ i : A.\ INF\ j : B.\ f\ i\ j) = (INF\ p : A \times B.\ f\ (fst\ p)\ (snd\ p))$
by (*rule antisym*) (*auto intro*!: *INF-greatest INF-lower2*)

40.0.1 *Liminf* and *Limsup*

definition $Liminf :: 'a\ filter \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b$ :: *complete-lattice* **where**
$Liminf\ F\ f = (SUP\ P{:}\{P.\ eventually\ P\ F\}.\ INF\ x{:}\{x.\ P\ x\}.\ f\ x)$

definition $Limsup :: 'a\ filter \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b$ :: *complete-lattice* **where**
$Limsup\ F\ f = (INF\ P{:}\{P.\ eventually\ P\ F\}.\ SUP\ x{:}\{x.\ P\ x\}.\ f\ x)$

abbreviation $liminf \equiv Liminf\ sequentially$

abbreviation $limsup \equiv Limsup\ sequentially$

lemma *Liminf-eqI*:
$(\bigwedge P.\ eventually\ P\ F \Longrightarrow INFIMUM\ (Collect\ P)\ f \leq x) \Longrightarrow$

$(\bigwedge y.\ (\bigwedge P.\ \mathit{eventually}\ P\ F \Longrightarrow \mathit{INFIMUM}\ (\mathit{Collect}\ P)\ f \leq y) \Longrightarrow x \leq y) \Longrightarrow \mathit{Liminf}\ F\ f = x$
unfolding *Liminf-def* **by** (*auto intro*!: *SUP-eqI*)

lemma *Limsup-eqI*:
$(\bigwedge P.\ \mathit{eventually}\ P\ F \Longrightarrow x \leq \mathit{SUPREMUM}\ (\mathit{Collect}\ P)\ f) \Longrightarrow$
$(\bigwedge y.\ (\bigwedge P.\ \mathit{eventually}\ P\ F \Longrightarrow y \leq \mathit{SUPREMUM}\ (\mathit{Collect}\ P)\ f) \Longrightarrow y \leq x) \Longrightarrow \mathit{Limsup}\ F\ f = x$
unfolding *Limsup-def* **by** (*auto intro*!: *INF-eqI*)

lemma *liminf-SUP-INF*: $\mathit{liminf}\ f = (\mathit{SUP}\ n.\ \mathit{INF}\ m{:}\{n..\}.\ f\ m)$
unfolding *Liminf-def eventually-sequentially*
by (*rule SUP-eq*) (*auto simp*: *atLeast-def intro*!: *INF-mono*)

lemma *limsup-INF-SUP*: $\mathit{limsup}\ f = (\mathit{INF}\ n.\ \mathit{SUP}\ m{:}\{n..\}.\ f\ m)$
unfolding *Limsup-def eventually-sequentially*
by (*rule INF-eq*) (*auto simp*: *atLeast-def intro*!: *SUP-mono*)

lemma *Limsup-const*:
assumes *ntriv*: $\neg$ *trivial-limit F*
shows $\mathit{Limsup}\ F\ (\lambda x.\ c) = c$
proof −
have $*$: $\bigwedge P.\ \mathit{Ex}\ P \longleftrightarrow P \neq (\lambda x.\ \mathit{False})$ **by** *auto*
have $\bigwedge P.\ \mathit{eventually}\ P\ F \Longrightarrow (\mathit{SUP}\ x : \{x.\ P\ x\}.\ c) = c$
using *ntriv* **by** (*intro SUP-const*) (*auto simp*: *eventually-False* $*$)
then show *?thesis*
unfolding *Limsup-def* **using** *eventually-True*
by (*subst INF-cong*[**where** $D{=}\lambda x.\ c$])
(*auto intro*!: *INF-const simp del*: *eventually-True*)
qed

lemma *Liminf-const*:
assumes *ntriv*: $\neg$ *trivial-limit F*
shows $\mathit{Liminf}\ F\ (\lambda x.\ c) = c$
proof −
have $*$: $\bigwedge P.\ \mathit{Ex}\ P \longleftrightarrow P \neq (\lambda x.\ \mathit{False})$ **by** *auto*
have $\bigwedge P.\ \mathit{eventually}\ P\ F \Longrightarrow (\mathit{INF}\ x : \{x.\ P\ x\}.\ c) = c$
using *ntriv* **by** (*intro INF-const*) (*auto simp*: *eventually-False* $*$)
then show *?thesis*
unfolding *Liminf-def* **using** *eventually-True*
by (*subst SUP-cong*[**where** $D{=}\lambda x.\ c$])
(*auto intro*!: *SUP-const simp del*: *eventually-True*)
qed

lemma *Liminf-mono*:
assumes *ev*: *eventually* $(\lambda x.\ f\ x \leq g\ x)\ F$
shows $\mathit{Liminf}\ F\ f \leq \mathit{Liminf}\ F\ g$
unfolding *Liminf-def*
proof (*safe intro*!: *SUP-mono*)

fix P **assume** *eventually* P F
with *ev* **have** *eventually* $(\lambda x.\ f\ x \leq g\ x \wedge P\ x)$ F (**is** *eventually* *?Q* F) **by** (*rule* *eventually-conj*)
then show $\exists\, Q{\in}\{P.\ \textit{eventually}\ P\ F\}.\ \textit{INFIMUM}\ (\textit{Collect}\ P)\ f \leq \textit{INFIMUM}\ (\textit{Collect}\ Q)\ g$
by (*intro* *bexI*[*of* - *?Q*]) (*auto* *intro*!: *INF-mono*)
qed

lemma *Liminf-eq*:
assumes *eventually* $(\lambda x.\ f\ x = g\ x)$ F
shows *Liminf* F f = *Liminf* F g
by (*intro* *antisym* *Liminf-mono* *eventually-mono*[*OF* *assms*]) *auto*

lemma *Limsup-mono*:
assumes *ev*: *eventually* $(\lambda x.\ f\ x \leq g\ x)$ F
shows *Limsup* F $f \leq$ *Limsup* F g
unfolding *Limsup-def*
proof (*safe* *intro*!: *INF-mono*)
fix P **assume** *eventually* P F
with *ev* **have** *eventually* $(\lambda x.\ f\ x \leq g\ x \wedge P\ x)$ F (**is** *eventually* *?Q* F) **by** (*rule* *eventually-conj*)
then show $\exists\, Q{\in}\{P.\ \textit{eventually}\ P\ F\}.\ \textit{SUPREMUM}\ (\textit{Collect}\ Q)\ f \leq \textit{SUPREMUM}\ (\textit{Collect}\ P)\ g$
by (*intro* *bexI*[*of* - *?Q*]) (*auto* *intro*!: *SUP-mono*)
qed

lemma *Limsup-eq*:
assumes *eventually* $(\lambda x.\ f\ x = g\ x)$ *net*
shows *Limsup net* f = *Limsup net* g
by (*intro* *antisym* *Limsup-mono* *eventually-mono*[*OF* *assms*]) *auto*

lemma *Liminf-le-Limsup*:
assumes *ntriv*: $\neg$ *trivial-limit* F
shows *Liminf* F $f \leq$ *Limsup* F f
unfolding *Limsup-def* *Liminf-def*
apply (*rule* *SUP-least*)
apply (*rule* *INF-greatest*)
proof *safe*
fix P Q **assume** *eventually* P F *eventually* Q F
then have *eventually* $(\lambda x.\ P\ x \wedge Q\ x)$ F (**is** *eventually* *?C* F) **by** (*rule* *eventually-conj*)
then have *not-False*: $(\lambda x.\ P\ x \wedge Q\ x) \neq (\lambda x.\ \textit{False})$
using *ntriv* **by** (*auto* *simp* *add*: *eventually-False*)
have *INFIMUM* (*Collect* P) $f \leq$ *INFIMUM* (*Collect* *?C*) f
by (*rule* *INF-mono*) *auto*
also have $\ldots \leq$ *SUPREMUM* (*Collect* *?C*) f
using *not-False* **by** (*intro* *INF-le-SUP*) *auto*
also have $\ldots \leq$ *SUPREMUM* (*Collect* Q) f
by (*rule* *SUP-mono*) *auto*
finally show *INFIMUM* (*Collect* P) $f \leq$ *SUPREMUM* (*Collect* Q) f **.**

qed

lemma *Liminf-bounded*:
assumes *ntriv*: ¬ *trivial-limit F*
assumes *le*: *eventually* (λ*n*. *C* ≤ *X n*) *F*
shows *C* ≤ *Liminf F X*
using *Liminf-mono*[*OF le*] *Liminf-const*[*OF ntriv*, *of C*] **by** *simp*

lemma *Limsup-bounded*:
assumes *ntriv*: ¬ *trivial-limit F*
assumes *le*: *eventually* (λ*n*. *X n* ≤ *C*) *F*
shows *Limsup F X* ≤ *C*
using *Limsup-mono*[*OF le*] *Limsup-const*[*OF ntriv*, *of C*] **by** *simp*

lemma *le-Limsup*:
assumes *F*: *F* ≠ *bot* **and** *x*: ∀ *F x in F*. *l* ≤ *f x*
shows *l* ≤ *Limsup F f*
proof −
have *l* = *Limsup F* (λ*x*. *l*)
using *F* **by** (*simp add*: *Limsup-const*)
also have ... ≤ *Limsup F f*
by (*intro Limsup-mono x*)
finally show *?thesis* **.**
qed

lemma *le-Liminf-iff*:
fixes *X* :: - ⇒ - :: *complete-linorder*
shows *C* ≤ *Liminf F X* ⟷ (∀ *y*<*C*. *eventually* (λ*x*. *y* < *X x*) *F*)
proof −
have *eventually* (λ*x*. *y* < *X x*) *F*
if *eventually P F y* < *INFIMUM* (*Collect P*) *X* **for** *y P*
using *that* **by** (*auto elim*!: *eventually-mono dest*: *less-INF-D*)
moreover
have ∃ *P*. *eventually P F* ∧ *y* < *INFIMUM* (*Collect P*) *X*
if *y* < *C* **and** *y*: ∀ *y*<*C*. *eventually* (λ*x*. *y* < *X x*) *F* **for** *y P*
proof (*cases* ∃ *z*. *y* < *z* ∧ *z* < *C*)
case *True*
then obtain *z* **where** *z*: *y* < *z* ∧ *z* < *C* **..**
moreover from *z* **have** *z* ≤ *INFIMUM* {*x*. *z* < *X x*} *X*
by (*auto intro*!: *INF-greatest*)
ultimately show *?thesis*
using *y* **by** (*intro exI*[*of* - λ*x*. *z* < *X x*]) *auto*
next
case *False*
then have *C* ≤ *INFIMUM* {*x*. *y* < *X x*} *X*
by (*intro INF-greatest*) *auto*
with ‹*y* < *C*› **show** *?thesis*
using *y* **by** (*intro exI*[*of* - λ*x*. *y* < *X x*]) *auto*
qed

```
  ultimately show ?thesis
    unfolding Liminf-def le-SUP-iff by auto
qed

lemma Limsup-le-iff:
  fixes X :: - ⇒ - :: complete-linorder
  shows C ≥ Limsup F X ⟷ (∀ y>C. eventually (λx. y > X x) F)
proof −
  { fix y P assume eventually P F y > SUPREMUM (Collect P) X
    then have eventually (λx. y > X x) F
      by (auto elim!: eventually-mono dest: SUP-lessD) }
  moreover
  { fix y P assume y > C and y: ∀ y>C. eventually (λx. y > X x) F
    have ∃ P. eventually P F ∧ y > SUPREMUM (Collect P) X
    proof (cases ∃ z. C < z ∧ z < y)
      case True
      then obtain z where z: C < z ∧ z < y ..
      moreover from z have z ≥ SUPREMUM {x. z > X x} X
        by (auto intro!: SUP-least)
      ultimately show ?thesis
        using y by (intro exI[of - λx. z > X x]) auto
    next
      case False
      then have C ≥ SUPREMUM {x. y > X x} X
        by (intro SUP-least) (auto simp: not-less)
      with ‹y > C› show ?thesis
        using y by (intro exI[of - λx. y > X x]) auto
    qed }
  ultimately show ?thesis
    unfolding Limsup-def INF-le-iff by auto
qed

lemma less-LiminfD:
  y < Liminf F (f :: - ⇒ 'a :: complete-linorder) ⟹ eventually (λx. f x > y) F
  using le-Liminf-iff[of Liminf F f F f] by simp

lemma Limsup-lessD:
  y > Limsup F (f :: - ⇒ 'a :: complete-linorder) ⟹ eventually (λx. f x < y) F
  using Limsup-le-iff[of F f Limsup F f] by simp

lemma lim-imp-Liminf:
  fixes f :: 'a ⇒ - :: {complete-linorder,linorder-topology}
  assumes ntriv: ¬ trivial-limit F
  assumes lim: (f ⟶ f0) F
  shows Liminf F f = f0
proof (intro Liminf-eqI)
  fix P assume P: eventually P F
  then have eventually (λx. INFIMUM (Collect P) f ≤ f x) F
    by eventually-elim (auto intro!: INF-lower)
```

```
  then show INFIMUM (Collect P) f ≤ f0
    by (rule tendsto-le[OF ntriv lim tendsto-const])
next
  fix y assume upper: ⋀P. eventually P F ⟹ INFIMUM (Collect P) f ≤ y
  show f0 ≤ y
  proof cases
    assume ∃ z. y < z ∧ z < f0
    then obtain z where y < z ∧ z < f0 ..
    moreover have z ≤ INFIMUM {x. z < f x} f
      by (rule INF-greatest) simp
    ultimately show ?thesis
      using lim[THEN topological-tendstoD, THEN upper, of {z <..}] by auto
  next
    assume discrete: ¬ (∃ z. y < z ∧ z < f0)
    show ?thesis
    proof (rule classical)
      assume ¬ f0 ≤ y
      then have eventually (λx. y < f x) F
        using lim[THEN topological-tendstoD, of {y <..}] by auto
      then have eventually (λx. f0 ≤ f x) F
        using discrete by (auto elim!: eventually-mono)
      then have INFIMUM {x. f0 ≤ f x} f ≤ y
        by (rule upper)
      moreover have f0 ≤ INFIMUM {x. f0 ≤ f x} f
        by (intro INF-greatest) simp
      ultimately show f0 ≤ y by simp
    qed
  qed
qed

lemma lim-imp-Limsup:
  fixes f :: 'a ⇒ - :: {complete-linorder,linorder-topology}
  assumes ntriv: ¬ trivial-limit F
  assumes lim: (f ⟶ f0) F
  shows Limsup F f = f0
proof (intro Limsup-eqI)
  fix P assume P: eventually P F
  then have eventually (λx. f x ≤ SUPREMUM (Collect P) f) F
    by eventually-elim (auto intro!: SUP-upper)
  then show f0 ≤ SUPREMUM (Collect P) f
    by (rule tendsto-le[OF ntriv tendsto-const lim])
next
  fix y assume lower: ⋀P. eventually P F ⟹ y ≤ SUPREMUM (Collect P) f
  show y ≤ f0
  proof (cases ∃ z. f0 < z ∧ z < y)
    case True
    then obtain z where f0 < z ∧ z < y ..
    moreover have SUPREMUM {x. f x < z} f ≤ z
      by (rule SUP-least) simp
```

```
    ultimately show ?thesis
      using lim[THEN topological-tendstoD, THEN lower, of {..< z}] by auto
  next
    case False
    show ?thesis
    proof (rule classical)
      assume ¬ y ≤ f0
      then have eventually (λx. f x < y) F
        using lim[THEN topological-tendstoD, of {..< y}] by auto
      then have eventually (λx. f x ≤ f0) F
        using False by (auto elim!: eventually-mono simp: not-less)
      then have y ≤ SUPREMUM {x. f x ≤ f0} f
        by (rule lower)
      moreover have SUPREMUM {x. f x ≤ f0} f ≤ f0
        by (intro SUP-least) simp
      ultimately show y ≤ f0 by simp
    qed
  qed
qed

lemma Liminf-eq-Limsup:
  fixes f0 :: 'a :: {complete-linorder,linorder-topology}
  assumes ntriv: ¬ trivial-limit F
    and lim: Liminf F f = f0 Limsup F f = f0
  shows (f ⟶ f0) F
proof (rule order-tendstoI)
  fix a assume f0 < a
  with assms have Limsup F f < a by simp
  then obtain P where eventually P F SUPREMUM (Collect P) f < a
    unfolding Limsup-def INF-less-iff by auto
  then show eventually (λx. f x < a) F
    by (auto elim!: eventually-mono dest: SUP-lessD)
next
  fix a assume a < f0
  with assms have a < Liminf F f by simp
  then obtain P where eventually P F a < INFIMUM (Collect P) f
    unfolding Liminf-def less-SUP-iff by auto
  then show eventually (λx. a < f x) F
    by (auto elim!: eventually-mono dest: less-INF-D)
qed

lemma tendsto-iff-Liminf-eq-Limsup:
  fixes f0 :: 'a :: {complete-linorder,linorder-topology}
  shows ¬ trivial-limit F ⟹ (f ⟶ f0) F ⟷ (Liminf F f = f0 ∧ Limsup F
f = f0)
  by (metis Liminf-eq-Limsup lim-imp-Limsup lim-imp-Liminf)

lemma liminf-subseq-mono:
  fixes X :: nat ⇒ 'a :: complete-linorder
```

```
  assumes subseq r
  shows liminf X ≤ liminf (X ∘ r)
proof−
  have ⋀n. (INF m:{n..}. X m) ≤ (INF m:{n..}. (X ∘ r) m)
  proof (safe intro!: INF-mono)
    fix n m :: nat assume n ≤ m then show ∃ma∈{n..}. X ma ≤ (X ∘ r) m
      using seq-suble[OF ‹subseq r›, of m] by (intro bexI[of - r m]) auto
  qed
  then show ?thesis by (auto intro!: SUP-mono simp: liminf-SUP-INF comp-def)
qed

lemma limsup-subseq-mono:
  fixes X :: nat ⇒ 'a :: complete-linorder
  assumes subseq r
  shows limsup (X ∘ r) ≤ limsup X
proof−
  have (SUP m:{n..}. (X ∘ r) m) ≤ (SUP m:{n..}. X m) for n
  proof (safe intro!: SUP-mono)
    fix m :: nat
    assume n ≤ m
    then show ∃ma∈{n..}. (X ∘ r) m ≤ X ma
      using seq-suble[OF ‹subseq r›, of m] by (intro bexI[of - r m]) auto
  qed
  then show ?thesis
    by (auto intro!: INF-mono simp: limsup-INF-SUP comp-def)
qed

lemma continuous-on-imp-continuous-within:
  continuous-on s f ⟹ t ⊆ s ⟹ x ∈ s ⟹ continuous (at x within t) f
  unfolding continuous-on-eq-continuous-within
  by (auto simp: continuous-within intro: tendsto-within-subset)

lemma Liminf-compose-continuous-mono:
   fixes f :: 'a::{complete-linorder, linorder-topology} ⇒ 'b::{complete-linorder,
linorder-topology}
  assumes c: continuous-on UNIV f and am: mono f and F: F ≠ bot
  shows Liminf F (λn. f (g n)) = f (Liminf F g)
proof −
  { fix P assume eventually P F
    have ∃x. P x
    proof (rule ccontr)
      assume ¬ (∃x. P x) then have P = (λx. False)
        by auto
      with ‹eventually P F› F show False
        by auto
    qed }
  note ∗ = this

  have f (Liminf F g) = (SUP P : {P. eventually P F}. f (Inf (g ‘ Collect P)))
```

```
    unfolding Liminf-def SUP-def
   by (subst continuous-at-Sup-mono[OF am continuous-on-imp-continuous-within[OF
c]])
       (auto intro: eventually-True)
  also have ... = (SUP P : {P. eventually P F}. INFIMUM (g ` Collect P) f)
   by (intro SUP-cong refl continuous-at-Inf-mono[OF am continuous-on-imp-continuous-within[OF
c]])
       (auto dest!: eventually-happens simp: F)
  finally show ?thesis by (auto simp: Liminf-def)
qed

lemma Limsup-compose-continuous-mono:
   fixes f :: 'a::{complete-linorder, linorder-topology} ⇒ 'b::{complete-linorder,
linorder-topology}
  assumes c: continuous-on UNIV f and am: mono f and F: F ≠ bot
  shows Limsup F (λn. f (g n)) = f (Limsup F g)
proof −
  { fix P assume eventually P F
    have ∃ x. P x
    proof (rule ccontr)
      assume ¬ (∃ x. P x) then have P = (λx. False)
        by auto
      with ‹eventually P F› F show False
        by auto
    qed }
  note ∗ = this

  have f (Limsup F g) = (INF P : {P. eventually P F}. f (Sup (g ` Collect P)))
    unfolding Limsup-def INF-def
   by (subst continuous-at-Inf-mono[OF am continuous-on-imp-continuous-within[OF
c]])
       (auto intro: eventually-True)
  also have ... = (INF P : {P. eventually P F}. SUPREMUM (g ` Collect P) f)
   by (intro INF-cong refl continuous-at-Sup-mono[OF am continuous-on-imp-continuous-within[OF
c]])
       (auto dest!: eventually-happens simp: F)
  finally show ?thesis by (auto simp: Limsup-def)
qed

lemma Liminf-compose-continuous-antimono:
 fixes f :: 'a::{complete-linorder,linorder-topology} ⇒ 'b::{complete-linorder,linorder-topology}
  assumes c: continuous-on UNIV f
    and am: antimono f
    and F: F ≠ bot
  shows Liminf F (λn. f (g n)) = f (Limsup F g)
proof −
  have ∗: ∃ x. P x if eventually P F for P
  proof (rule ccontr)
    assume ¬ (∃ x. P x) then have P = (λx. False)
```

```
    by auto
  with ‹eventually P F› F show False
    by auto
 qed
 have f (Limsup F g) = (SUP P : {P. eventually P F}. f (Sup (g ‘ Collect P)))
  unfolding Limsup-def INF-def
  by (subst continuous-at-Inf-antimono[OF am continuous-on-imp-continuous-within[OF
c]])
     (auto intro: eventually-True)
 also have ... = (SUP P : {P. eventually P F}. INFIMUM (g ‘ Collect P) f)
  by (intro SUP-cong refl continuous-at-Sup-antimono[OF am continuous-on-imp-continuous-within[OF
c]])
     (auto dest!: eventually-happens simp: F)
 finally show ?thesis
  by (auto simp: Liminf-def)
qed

lemma Limsup-compose-continuous-antimono:
  fixes f :: 'a::{complete-linorder, linorder-topology} ⇒ 'b::{complete-linorder,
linorder-topology}
 assumes c: continuous-on UNIV f and am: antimono f and F: F ≠ bot
 shows Limsup F (λn. f (g n)) = f (Liminf F g)
proof −
 { fix P assume eventually P F
  have ∃ x. P x
  proof (rule ccontr)
   assume ¬ (∃ x. P x) then have P = (λx. False)
    by auto
   with ‹eventually P F› F show False
    by auto
  qed }
 note ∗ = this

 have f (Liminf F g) = (INF P : {P. eventually P F}. f (Inf (g ‘ Collect P)))
  unfolding Liminf-def SUP-def
  by (subst continuous-at-Sup-antimono[OF am continuous-on-imp-continuous-within[OF
c]])
     (auto intro: eventually-True)
 also have ... = (INF P : {P. eventually P F}. SUPREMUM (g ‘ Collect P) f)
  by (intro INF-cong refl continuous-at-Inf-antimono[OF am continuous-on-imp-continuous-within[OF
c]])
     (auto dest!: eventually-happens simp: F)
 finally show ?thesis
  by (auto simp: Limsup-def)
qed
```

40.1 More Limits

lemma *convergent-limsup-cl*:

fixes $X :: nat \Rightarrow 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
shows *convergent* $X \Longrightarrow limsup\ X = lim\ X$
by (*auto simp*: *convergent-def limI lim-imp-Limsup*)

lemma *convergent-liminf-cl*:
fixes $X :: nat \Rightarrow 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
shows *convergent* $X \Longrightarrow liminf\ X = lim\ X$
by (*auto simp*: *convergent-def limI lim-imp-Liminf*)

lemma *lim-increasing-cl*:
assumes $\bigwedge n\ m.\ n \geq m \Longrightarrow f\ n \geq f\ m$
obtains l **where** $f \longrightarrow (l::'a::\{complete\text{-}linorder,linorder\text{-}topology\})$
proof
show $f \longrightarrow (SUP\ n.\ f\ n)$
using *assms*
by (*intro increasing-tendsto*)
(*auto simp*: *SUP-upper eventually-sequentially less-SUP-iff intro*: *less-le-trans*)
qed

lemma *lim-decreasing-cl*:
assumes $\bigwedge n\ m.\ n \geq m \Longrightarrow f\ n \leq f\ m$
obtains l **where** $f \longrightarrow (l::'a::\{complete\text{-}linorder,linorder\text{-}topology\})$
proof
show $f \longrightarrow (INF\ n.\ f\ n)$
using *assms*
by (*intro decreasing-tendsto*)
(*auto simp*: *INF-lower eventually-sequentially INF-less-iff intro*: *le-less-trans*)
qed

lemma *compact-complete-linorder*:
fixes $X :: nat \Rightarrow 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
shows $\exists l\ r.\ subseq\ r \wedge (X \circ r) \longrightarrow l$
proof −
obtain r **where** *subseq* r **and** *mono*: *monoseq* $(X \circ r)$
using *seq-monosub*[*of X*]
unfolding *comp-def*
by *auto*
then have $(\forall n\ m.\ m \leq n \longrightarrow (X \circ r)\ m \leq (X \circ r)\ n) \vee (\forall n\ m.\ m \leq n \longrightarrow (X \circ r)\ n \leq (X \circ r)\ m)$
by (*auto simp add*: *monoseq-def*)
then obtain l **where** $(X \circ r) \longrightarrow l$
using *lim-increasing-cl*[*of* $X \circ r$] *lim-decreasing-cl*[*of* $X \circ r$]
by *auto*
then show *?thesis*
using ‹*subseq r*› **by** *auto*
qed

end

41 Extended real number line

theory *Extended-Real*
imports *Complex-Main Extended-Nat Liminf-Limsup*
begin

This should be part of *Extended-Nat* or *Order-Continuity*, but then the AFP-entry *Jinja-Thread* fails, as it does overload certain named from *Complex-Main*.

lemma *continuous-at-left-imp-sup-continuous*:
fixes $f :: 'a \Rightarrow 'a$::{*complete-linorder*, *linorder-topology*}
assumes *mono f* $\bigwedge x$. *continuous* (*at-left x*) *f*
shows *sup-continuous f*
unfolding *sup-continuous-def*
proof *safe*
fix *M* :: *nat* $\Rightarrow$ $'a$ **assume** *incseq M* **then show** *f* (*SUP i*. *M i*) = (*SUP i*. *f* (*M i*))
using *continuous-at-Sup-mono*[*OF assms*, *of range M*] **by** *simp*
qed

lemma *sup-continuous-at-left*:
fixes $f :: 'a \Rightarrow 'a$::{*complete-linorder*, *linorder-topology*, *first-countable-topology*}
assumes *f*: *sup-continuous f*
shows *continuous* (*at-left x*) *f*
proof *cases*
assume $x = bot$ **then show** *?thesis*
by (*simp add*: *trivial-limit-at-left-bot*)
next
assume *x*: $x \neq bot$
show *?thesis*
unfolding *continuous-within*
proof (*intro tendsto-at-left-sequentially*[*of bot*])
fix *S* :: *nat* $\Rightarrow$ $'a$ **assume** *S*: *incseq S* **and** *S-x*: $S \longrightarrow x$
from *S-x* **have** *x-eq*: $x = (SUP\ i.\ S\ i)$
by (*rule LIMSEQ-unique*) (*intro LIMSEQ-SUP S*)
show $(\lambda n.\ f\ (S\ n)) \longrightarrow f\ x$
unfolding *x-eq sup-continuousD*[*OF f S*]
using *S sup-continuous-mono*[*OF f*] **by** (*intro LIMSEQ-SUP*) (*auto simp*: *mono-def*)
qed (*insert x*, *auto simp*: *bot-less*)
qed

lemma *sup-continuous-iff-at-left*:
fixes $f :: 'a \Rightarrow 'a$::{*complete-linorder*, *linorder-topology*, *first-countable-topology*}
shows *sup-continuous f* $\longleftrightarrow$ $(\forall x.$ *continuous* (*at-left x*) *f*$) \wedge$ *mono f*
using *sup-continuous-at-left*[*of f*] *continuous-at-left-imp-sup-continuous*[*of f*]
sup-continuous-mono[*of f*] **by** *auto*

lemma *continuous-at-right-imp-inf-continuous*:
fixes $f :: 'a \Rightarrow 'a$::{*complete-linorder*, *linorder-topology*}

```
  assumes mono f ⋀x. continuous (at-right x) f
  shows inf-continuous f
  unfolding inf-continuous-def
proof safe
  fix M :: nat ⇒ 'a assume decseq M then show f (INF i. M i) = (INF i. f (M
i))
    using continuous-at-Inf-mono[OF assms, of range M] by simp
qed

lemma inf-continuous-at-right:
  fixes f :: 'a ⇒ 'a::{complete-linorder, linorder-topology, first-countable-topology}
  assumes f: inf-continuous f
  shows continuous (at-right x) f
proof cases
  assume x = top then show ?thesis
    by (simp add: trivial-limit-at-right-top)
next
  assume x: x ≠ top
  show ?thesis
    unfolding continuous-within
  proof (intro tendsto-at-right-sequentially[of - top])
    fix S :: nat ⇒ 'a assume S: decseq S and S-x: S ⟶ x
    from S-x have x-eq: x = (INF i. S i)
      by (rule LIMSEQ-unique) (intro LIMSEQ-INF S)
    show (λn. f (S n)) ⟶ f x
      unfolding x-eq inf-continuousD[OF f S]
        using S inf-continuous-mono[OF f] by (intro LIMSEQ-INF) (auto simp:
mono-def antimono-def)
  qed (insert x, auto simp: less-top)
qed

lemma inf-continuous-iff-at-right:
  fixes f :: 'a ⇒ 'a::{complete-linorder, linorder-topology, first-countable-topology}
  shows inf-continuous f ⟷ (∀ x. continuous (at-right x) f) ∧ mono f
  using inf-continuous-at-right[of f] continuous-at-right-imp-inf-continuous[of f]
    inf-continuous-mono[of f] by auto

instantiation enat :: linorder-topology
begin

definition open-enat :: enat set ⇒ bool where
  open-enat = generate-topology (range lessThan ∪ range greaterThan)

instance
  proof qed (rule open-enat-def)

end

lemma open-enat: open {enat n}
```

proof (*cases n*)
case *0*
then have {*enat n*} = {..< *eSuc 0*}
by (*auto simp*: *enat-0*)
then show *?thesis*
by *simp*
next
case (*Suc n'*)
then have {*enat n*} = {*enat n'* <..< *enat* (*Suc n*)}
apply *auto*
apply (*case-tac x*)
apply *auto*
done
then show *?thesis*
by *simp*
qed

lemma *open-enat-iff*:
fixes *A* :: *enat set*
shows *open A* ⟷ (∞ ∈ *A* ⟶ (∃ *n*::*nat*. {*n* <..} ⊆ *A*))
proof *safe*
assume ∞ ∉ *A*
then have *A* = (⋃ *n*∈{*n*. *enat n* ∈ *A*}. {*enat n*})
apply *auto*
apply (*case-tac x*)
apply *auto*
done
moreover have *open* ...
by (*auto intro*: *open-enat*)
ultimately show *open A*
by *simp*
next
fix *n* **assume** {*enat n* <..} ⊆ *A*
then have *A* = (⋃ *n*∈{*n*. *enat n* ∈ *A*}. {*enat n*}) ∪ {*enat n* <..}
apply *auto*
apply (*case-tac x*)
apply *auto*
done
moreover have *open* ...
by (*intro open-Un open-UN ballI open-enat open-greaterThan*)
ultimately show *open A*
by *simp*
next
assume *open A* ∞ ∈ *A*
then have *generate-topology* (*range lessThan* ∪ *range greaterThan*) *A* ∞ ∈ *A*
unfolding *open-enat-def* **by** *auto*
then show ∃ *n*::*nat*. {*n* <..} ⊆ *A*
proof *induction*
case (*Int A B*)

```
    then obtain n m where {enat n<..} ⊆ A {enat m<..} ⊆ B
      by auto
    then have {enat (max n m) <..} ⊆ A ∩ B
      by (auto simp add: subset-eq Ball-def max-def enat-ord-code(1)[symmetric]
simp del: enat-ord-code(1))
    then show ?case
      by auto
  next
    case (UN K)
    then obtain k where k ∈ K ∞ ∈ k
      by auto
    with UN.IH[OF this] show ?case
      by auto
  qed auto
qed
```

For more lemmas about the extended real numbers go to `~~/src/HOL/Multivariate_Analysis/Extended_Real_Limits.thy`

41.1 Definition and basic properties

```
datatype ereal = ereal real | PInfty | MInfty

instantiation ereal :: uminus
begin

fun uminus-ereal where
  − (ereal r) = ereal (− r)
| − PInfty = MInfty
| − MInfty = PInfty

instance ..

end

instantiation ereal :: infinity
begin

definition (∞::ereal) = PInfty
instance ..

end

declare [[coercion ereal :: real ⇒ ereal]]

lemma ereal-uminus-uminus[simp]:
  fixes a :: ereal
  shows − (− a) = a
  by (cases a) simp-all
```

lemma
shows *PInfty-eq-infinity*[*simp*]: $PInfty = \infty$
and *MInfty-eq-minfinity*[*simp*]: $MInfty = -\infty$
and *MInfty-neq-PInfty*[*simp*]: $\infty \neq -(\infty::ereal) - \infty \neq (\infty::ereal)$
and *MInfty-neq-ereal*[*simp*]: $ereal\ r \neq -\infty - \infty \neq ereal\ r$
and *PInfty-neq-ereal*[*simp*]: $ereal\ r \neq \infty\ \infty \neq ereal\ r$
and *PInfty-cases*[*simp*]: $(case\ \infty\ of\ ereal\ r \Rightarrow f\ r \mid PInfty \Rightarrow y \mid MInfty \Rightarrow z) = y$
and *MInfty-cases*[*simp*]: $(case\ -\infty\ of\ ereal\ r \Rightarrow f\ r \mid PInfty \Rightarrow y \mid MInfty \Rightarrow z) = z$
by (*simp-all add*: *infinity-ereal-def*)

declare
PInfty-eq-infinity[*code-post*]
MInfty-eq-minfinity[*code-post*]

lemma [*code-unfold*]:
$\infty = PInfty$
$-\ PInfty = MInfty$
by *simp-all*

lemma *inj-ereal*[*simp*]: *inj-on ereal A*
unfolding *inj-on-def* **by** *auto*

lemma *ereal-cases*[*cases type*: *ereal*]:
obtains (*real*) r **where** $x = ereal\ r$
| (*PInf*) $x = \infty$
| (*MInf*) $x = -\infty$
using *assms* **by** (*cases x*) *auto*

lemmas *ereal2-cases* = *ereal-cases*[*case-product ereal-cases*]
lemmas *ereal3-cases* = *ereal2-cases*[*case-product ereal-cases*]

lemma *ereal-all-split*: $\bigwedge P.\ (\forall x::ereal.\ P\ x) \longleftrightarrow P\ \infty \wedge (\forall x.\ P\ (ereal\ x)) \wedge P\ (-\infty)$
by (*metis ereal-cases*)

lemma *ereal-ex-split*: $\bigwedge P.\ (\exists x::ereal.\ P\ x) \longleftrightarrow P\ \infty \vee (\exists x.\ P\ (ereal\ x)) \vee P\ (-\infty)$
by (*metis ereal-cases*)

lemma *ereal-uminus-eq-iff*[*simp*]:
fixes $a\ b :: ereal$
shows $-a = -b \longleftrightarrow a = b$
by (*cases rule*: *ereal2-cases*[*of a b*]) *simp-all*

function *real-of-ereal* :: $ereal \Rightarrow real$ **where**
$real\text{-}of\text{-}ereal\ (ereal\ r) = r$

| *real-of-ereal* $\infty = 0$
| *real-of-ereal* $(-\infty) = 0$
by (*auto intro*: *ereal-cases*)
termination by *standard* (*rule wf-empty*)

lemma *real-of-ereal*[*simp*]:
real-of-ereal $(-x :: ereal) = -$ (*real-of-ereal* x)
by (*cases* x) *simp-all*

lemma *range-ereal*[*simp*]: *range ereal* $= UNIV - \{\infty, -\infty\}$
proof *safe*
fix x
assume $x \notin range\ ereal\ x \neq \infty$
then show $x = -\infty$
by (*cases* x) *auto*
qed *auto*

lemma *ereal-range-uminus*[*simp*]: *range uminus* $= (UNIV :: ereal\ set)$
proof *safe*
fix $x :: ereal$
show $x \in range\ uminus$
by (*intro image-eqI*[*of - - −x*]) *auto*
qed *auto*

instantiation *ereal* :: *abs*
begin

function *abs-ereal* **where**
$|ereal\ r| = ereal\ |r|$
| $|-\infty| = (\infty :: ereal)$
| $|\infty| = (\infty :: ereal)$
by (*auto intro*: *ereal-cases*)
termination proof qed (*rule wf-empty*)

instance ..

end

lemma *abs-eq-infinity-cases*[*elim*!]:
fixes $x :: ereal$
assumes $|x| = \infty$
obtains $x = \infty \mid x = -\infty$
using *assms* **by** (*cases* x) *auto*

lemma *abs-neq-infinity-cases*[*elim*!]:
fixes $x :: ereal$
assumes $|x| \neq \infty$
obtains r **where** $x = ereal\ r$
using *assms* **by** (*cases* x) *auto*

lemma *abs-ereal-uminus*[*simp*]:
fixes x :: *ereal*
shows $|-x| = |x|$
by (*cases x*) *auto*

lemma *ereal-infinity-cases*:
fixes a :: *ereal*
shows $a \neq \infty \Longrightarrow a \neq -\infty \Longrightarrow |a| \neq \infty$
by *auto*

41.1.1 Addition

instantiation *ereal* :: {*one*,*comm-monoid-add*,*zero-neq-one*}
begin

definition *0* = *ereal 0*
definition *1* = *ereal 1*

function *plus-ereal* **where**
ereal r + *ereal p* = *ereal* (*r* + *p*)
| ∞ + *a* = (∞::*ereal*)
| *a* + ∞ = (∞::*ereal*)
| *ereal r* + −∞ = − ∞
| −∞ + *ereal p* = −(∞::*ereal*)
| −∞ + −∞ = −(∞::*ereal*)
proof *goal-cases*
case *prems*: (*1 P x*)
then obtain *a b* **where** *x* = (*a*, *b*)
by (*cases x*) *auto*
with *prems* **show** *P*
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*
qed *auto*
termination by *standard* (*rule wf-empty*)

lemma *Infty-neq-0*[*simp*]:
(∞::*ereal*) ≠ *0 0* ≠ (∞::*ereal*)
−(∞::*ereal*) ≠ *0 0* ≠ −(∞::*ereal*)
by (*simp-all add*: *zero-ereal-def*)

lemma *ereal-eq-0*[*simp*]:
ereal r = *0* ⟷ *r* = *0*
0 = *ereal r* ⟷ *r* = *0*
unfolding *zero-ereal-def* **by** *simp-all*

lemma *ereal-eq-1*[*simp*]:
ereal r = *1* ⟷ *r* = *1*
1 = *ereal r* ⟷ *r* = *1*
unfolding *one-ereal-def* **by** *simp-all*

instance
proof
  fix *a b c* :: *ereal*
  show *0 + a = a*
    by (*cases a*) (*simp-all add*: *zero-ereal-def*)
  show *a + b = b + a*
    by (*cases rule*: *ereal2-cases*[*of a b*]) *simp-all*
  show *a + b + c = a + (b + c)*
    by (*cases rule*: *ereal3-cases*[*of a b c*]) *simp-all*
  show *0 ≠* (*1*::*ereal*)
    by (*simp add*: *one-ereal-def zero-ereal-def*)
qed

end

lemma *ereal-0-plus* [*simp*]: *ereal 0 + x = x*
  and *plus-ereal-0* [*simp*]: *x + ereal 0 = x*
by(*simp-all add*: *zero-ereal-def*[*symmetric*])

instance *ereal* :: *numeral* **..**

lemma *real-of-ereal-0*[*simp*]: *real-of-ereal* (*0*::*ereal*) = *0*
  unfolding *zero-ereal-def* **by** *simp*

lemma *abs-ereal-zero*[*simp*]: |*0*| = (*0*::*ereal*)
  unfolding *zero-ereal-def abs-ereal.simps* **by** *simp*

lemma *ereal-uminus-zero*[*simp*]: − *0* = (*0*::*ereal*)
  by (*simp add*: *zero-ereal-def*)

lemma *ereal-uminus-zero-iff*[*simp*]:
  fixes *a* :: *ereal*
  shows *−a = 0 ⟷ a = 0*
  by (*cases a*) *simp-all*

lemma *ereal-plus-eq-PInfty*[*simp*]:
  fixes *a b* :: *ereal*
  shows *a + b = ∞ ⟷ a = ∞ ∨ b = ∞*
  by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-plus-eq-MInfty*[*simp*]:
  fixes *a b* :: *ereal*
  shows *a + b = −∞ ⟷* (*a = −∞ ∨ b = −∞*) *∧ a ≠ ∞ ∧ b ≠ ∞*
  by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-add-cancel-left*:
  fixes *a b* :: *ereal*
  assumes *a ≠ −∞*

shows $a + b = a + c \longleftrightarrow a = \infty \vee b = c$
using *assms* **by** (*cases rule*: *ereal3-cases*[*of a b c*]) *auto*

lemma *ereal-add-cancel-right*:
fixes $a\ b$ :: *ereal*
assumes $a \neq -\infty$
shows $b + a = c + a \longleftrightarrow a = \infty \vee b = c$
using *assms* **by** (*cases rule*: *ereal3-cases*[*of a b c*]) *auto*

lemma *ereal-real*: *ereal* (*real-of-ereal x*) = (*if* $|x| = \infty$ *then 0 else x*)
by (*cases x*) *simp-all*

lemma *real-of-ereal-add*:
fixes $a\ b$ :: *ereal*
shows *real-of-ereal* $(a + b) =$
(*if* $(|a| = \infty) \wedge (|b| = \infty) \vee (|a| \neq \infty) \wedge (|b| \neq \infty)$ *then real-of-ereal a + real-of-ereal b else 0*)
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

41.1.2 Linear order on *ereal*

instantiation *ereal* :: *linorder*
begin

function *less-ereal*
where
ereal x < *ereal y* $\longleftrightarrow x < y$
| $(\infty$::*ereal*$) < a \longleftrightarrow$ *False*
| $a < -(\infty$::*ereal*$) \longleftrightarrow$ *False*
| *ereal x* $< \infty \longleftrightarrow$ *True*
| $-\infty <$ *ereal r* $\longleftrightarrow$ *True*
| $-\infty < (\infty$::*ereal*$) \longleftrightarrow$ *True*
proof *goal-cases*
case *prems*: (*1 P x*)
then obtain $a\ b$ **where** $x = (a,b)$ **by** (*cases x*) *auto*
with *prems* **show** *P* **by** (*cases rule*: *ereal2-cases*[*of a b*]) *auto*
qed *simp-all*
termination by (*relation* {}) *simp*

definition $x \leq (y$::*ereal*$) \longleftrightarrow x < y \vee x = y$

lemma *ereal-infty-less*[*simp*]:
fixes x :: *ereal*
shows $x < \infty \longleftrightarrow (x \neq \infty)$
$-\infty < x \longleftrightarrow (x \neq -\infty)$
by (*cases x*, *simp-all*) (*cases x*, *simp-all*)

lemma *ereal-infty-less-eq*[*simp*]:
fixes x :: *ereal*

shows $\infty \leq x \longleftrightarrow x = \infty$
and $x \leq -\infty \longleftrightarrow x = -\infty$
by (*auto simp add*: *less-eq-ereal-def*)

lemma *ereal-less*[*simp*]:
$ereal\ r < 0 \longleftrightarrow (r < 0)$
$0 < ereal\ r \longleftrightarrow (0 < r)$
$ereal\ r < 1 \longleftrightarrow (r < 1)$
$1 < ereal\ r \longleftrightarrow (1 < r)$
$0 < (\infty::ereal)$
$-(\infty::ereal) < 0$
by (*simp-all add*: *zero-ereal-def one-ereal-def*)

lemma *ereal-less-eq*[*simp*]:
$x \leq (\infty::ereal)$
$-(\infty::ereal) \leq x$
$ereal\ r \leq ereal\ p \longleftrightarrow r \leq p$
$ereal\ r \leq 0 \longleftrightarrow r \leq 0$
$0 \leq ereal\ r \longleftrightarrow 0 \leq r$
$ereal\ r \leq 1 \longleftrightarrow r \leq 1$
$1 \leq ereal\ r \longleftrightarrow 1 \leq r$
by (*auto simp add*: *less-eq-ereal-def zero-ereal-def one-ereal-def*)

lemma *ereal-infty-less-eq2*:
$a \leq b \Longrightarrow a = \infty \Longrightarrow b = (\infty::ereal)$
$a \leq b \Longrightarrow b = -\infty \Longrightarrow a = -(\infty::ereal)$
by *simp-all*

instance
proof
fix $x\ y\ z :: ereal$
show $x \leq x$
by (*cases x*) *simp-all*
show $x < y \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
by (*cases rule*: *ereal2-cases*[*of x y*]) *auto*
show $x \leq y \vee y \leq x$
by (*cases rule*: *ereal2-cases*[*of x y*]) *auto*
{
assume $x \leq y\ y \leq x$
then show $x = y$
by (*cases rule*: *ereal2-cases*[*of x y*]) *auto*
}
{
assume $x \leq y\ y \leq z$
then show $x \leq z$
by (*cases rule*: *ereal3-cases*[*of x y z*]) *auto*
}
qed

end

lemma *ereal-dense2*: $x < y \Longrightarrow \exists z.\ x < \mathit{ereal}\ z \wedge \mathit{ereal}\ z < y$
using *lt-ex gt-ex dense* **by** (*cases x y rule*: *ereal2-cases*) *auto*

instance *ereal* :: *dense-linorder*
by *standard* (*blast dest*: *ereal-dense2*)

instance *ereal* :: *ordered-ab-semigroup-add*
proof
fix *a b c* :: *ereal*
assume $a \leq b$
then show $c + a \leq c + b$
by (*cases rule*: *ereal3-cases*[*of a b c*]) *auto*
qed

lemma *real-of-ereal-positive-mono*:
fixes *x y* :: *ereal*
shows $0 \leq x \Longrightarrow x \leq y \Longrightarrow y \neq \infty \Longrightarrow \mathit{real\text{-}of\text{-}ereal}\ x \leq \mathit{real\text{-}of\text{-}ereal}\ y$
by (*cases rule*: *ereal2-cases*[*of x y*]) *auto*

lemma *ereal-MInfty-lessI*[*intro*, *simp*]:
fixes *a* :: *ereal*
shows $a \neq -\infty \Longrightarrow -\infty < a$
by (*cases a*) *auto*

lemma *ereal-less-PInfty*[*intro*, *simp*]:
fixes *a* :: *ereal*
shows $a \neq \infty \Longrightarrow a < \infty$
by (*cases a*) *auto*

lemma *ereal-less-ereal-Ex*:
fixes *a b* :: *ereal*
shows $x < \mathit{ereal}\ r \longleftrightarrow x = -\infty \vee (\exists p.\ p < r \wedge x = \mathit{ereal}\ p)$
by (*cases x*) *auto*

lemma *less-PInf-Ex-of-nat*: $x \neq \infty \longleftrightarrow (\exists n{::}nat.\ x < \mathit{ereal}\ (\mathit{real}\ n))$
proof (*cases x*)
case (*real r*)
then show *?thesis*
using *reals-Archimedean2*[*of r*] **by** *simp*
qed *simp-all*

lemma *ereal-add-mono*:
fixes *a b c d* :: *ereal*
assumes $a \leq b$
and $c \leq d$
shows $a + c \leq b + d$
using *assms*

apply (*cases a*)
apply (*cases rule*: *ereal3-cases*[*of b c d*], *auto*)
apply (*cases rule*: *ereal3-cases*[*of b c d*], *auto*)
done

lemma *ereal-minus-le-minus*[*simp*]:
fixes *a b* :: *ereal*
shows $-a \leq -b \longleftrightarrow b \leq a$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-minus-less-minus*[*simp*]:
fixes *a b* :: *ereal*
shows $-a < -b \longleftrightarrow b < a$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-le-real-iff*:
$x \leq$ *real-of-ereal y* $\longleftrightarrow (|y| \neq \infty \longrightarrow$ *ereal x* $\leq y) \wedge (|y| = \infty \longrightarrow x \leq 0)$
by (*cases y*) *auto*

lemma *real-le-ereal-iff*:
real-of-ereal y $\leq x \longleftrightarrow (|y| \neq \infty \longrightarrow y \leq$ *ereal x*$) \wedge (|y| = \infty \longrightarrow 0 \leq x)$
by (*cases y*) *auto*

lemma *ereal-less-real-iff*:
$x <$ *real-of-ereal y* $\longleftrightarrow (|y| \neq \infty \longrightarrow$ *ereal x* $< y) \wedge (|y| = \infty \longrightarrow x < 0)$
by (*cases y*) *auto*

lemma *real-less-ereal-iff*:
real-of-ereal y $< x \longleftrightarrow (|y| \neq \infty \longrightarrow y <$ *ereal x*$) \wedge (|y| = \infty \longrightarrow 0 < x)$
by (*cases y*) *auto*

lemma *real-of-ereal-pos*:
fixes *x* :: *ereal*
shows $0 \leq x \Longrightarrow 0 \leq$ *real-of-ereal x* **by** (*cases x*) *auto*

lemmas *real-of-ereal-ord-simps* =
ereal-le-real-iff real-le-ereal-iff ereal-less-real-iff real-less-ereal-iff

lemma *abs-ereal-ge0*[*simp*]: $0 \leq x \Longrightarrow |x :: ereal| = x$
by (*cases x*) *auto*

lemma *abs-ereal-less0*[*simp*]: $x < 0 \Longrightarrow |x :: ereal| = -x$
by (*cases x*) *auto*

lemma *abs-ereal-pos*[*simp*]: $0 \leq |x :: ereal|$
by (*cases x*) *auto*

lemma *ereal-abs-leI*:
fixes *x y* :: *ereal*

shows $[\![x \leq y; -x \leq y]\!] \Longrightarrow |x| \leq y$
by(*cases x y rule*: *ereal2-cases*)(*simp-all*)

lemma *real-of-ereal-le-0*[*simp*]: *real-of-ereal* $(x :: ereal) \leq 0 \longleftrightarrow x \leq 0 \vee x = \infty$
by (*cases x*) *auto*

lemma *abs-real-of-ereal*[*simp*]: $|$*real-of-ereal* $(x :: ereal)| =$ *real-of-ereal* $|x|$
by (*cases x*) *auto*

lemma *zero-less-real-of-ereal*:
fixes $x ::$ *ereal*
shows $0 <$ *real-of-ereal* $x \longleftrightarrow 0 < x \wedge x \neq \infty$
by (*cases x*) *auto*

lemma *ereal-0-le-uminus-iff*[*simp*]:
fixes $a ::$ *ereal*
shows $0 \leq - a \longleftrightarrow a \leq 0$
by (*cases rule*: *ereal2-cases*[*of a*]) *auto*

lemma *ereal-uminus-le-0-iff*[*simp*]:
fixes $a ::$ *ereal*
shows $- a \leq 0 \longleftrightarrow 0 \leq a$
by (*cases rule*: *ereal2-cases*[*of a*]) *auto*

lemma *ereal-add-strict-mono*:
fixes $a\ b\ c\ d ::$ *ereal*
assumes $a \leq b$
and $0 \leq a$
and $a \neq \infty$
and $c < d$
shows $a + c < b + d$
using *assms*
by (*cases rule*: *ereal3-cases*[*case-product ereal-cases*, *of a b c d*]) *auto*

lemma *ereal-less-add*:
fixes $a\ b\ c ::$ *ereal*
shows $|a| \neq \infty \Longrightarrow c < b \Longrightarrow a + c < a + b$
by (*cases rule*: *ereal2-cases*[*of b c*]) *auto*

lemma *ereal-add-nonneg-eq-0-iff*:
fixes $a\ b ::$ *ereal*
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow a + b = 0 \longleftrightarrow a = 0 \wedge b = 0$
by (*cases a b rule*: *ereal2-cases*) *auto*

lemma *ereal-uminus-eq-reorder*: $- a = b \longleftrightarrow a = (-b::ereal)$
by *auto*

lemma *ereal-uminus-less-reorder*: $- a < b \longleftrightarrow -b < (a::ereal)$
by (*subst* (*3*) *ereal-uminus-uminus*[*symmetric*]) (*simp only*: *ereal-minus-less-minus*)

lemma *ereal-less-uminus-reorder*: $a < - b \longleftrightarrow b < - (a::ereal)$
by (*subst* (*3*) *ereal-uminus-uminus*[*symmetric*]) (*simp only*: *ereal-minus-less-minus*)

lemma *ereal-uminus-le-reorder*: $- a \leq b \longleftrightarrow -b \leq (a::ereal)$
by (*subst* (*3*) *ereal-uminus-uminus*[*symmetric*]) (*simp only*: *ereal-minus-le-minus*)

lemmas *ereal-uminus-reorder* =
ereal-uminus-eq-reorder ereal-uminus-less-reorder ereal-uminus-le-reorder

lemma *ereal-bot*:
fixes x :: *ereal*
assumes $\bigwedge B.\ x \leq ereal\ B$
shows $x = - \infty$
proof (*cases x*)
case (*real r*)
with *assms*[*of r − 1*] **show** *?thesis*
by *auto*
next
case *PInf*
with *assms*[*of 0*] **show** *?thesis*
by *auto*
next
case *MInf*
then show *?thesis*
by *simp*
qed

lemma *ereal-top*:
fixes x :: *ereal*
assumes $\bigwedge B.\ x \geq ereal\ B$
shows $x = \infty$
proof (*cases x*)
case (*real r*)
with *assms*[*of r + 1*] **show** *?thesis*
by *auto*
next
case *MInf*
with *assms*[*of 0*] **show** *?thesis*
by *auto*
next
case *PInf*
then show *?thesis*
by *simp*
qed

lemma
shows *ereal-max*[*simp*]: *ereal* (*max x y*) = *max* (*ereal x*) (*ereal y*)
and *ereal-min*[*simp*]: *ereal* (*min x y*) = *min* (*ereal x*) (*ereal y*)

by (*simp-all add*: *min-def max-def*)

lemma *ereal-max-0*: *max 0* (*ereal r*) = *ereal* (*max 0 r*)
by (*auto simp*: *zero-ereal-def*)

lemma
fixes *f* :: *nat* ⇒ *ereal*
shows *ereal-incseq-uminus*[*simp*]: *incseq* (λ*x*. − *f x*) ⟷ *decseq f*
and *ereal-decseq-uminus*[*simp*]: *decseq* (λ*x*. − *f x*) ⟷ *incseq f*
unfolding *decseq-def incseq-def* **by** *auto*

lemma *incseq-ereal*: *incseq f* ⟹ *incseq* (λ*x*. *ereal* (*f x*))
unfolding *incseq-def* **by** *auto*

lemma *ereal-add-nonneg-nonneg*[*simp*]:
fixes *a b* :: *ereal*
shows *0* ≤ *a* ⟹ *0* ≤ *b* ⟹ *0* ≤ *a* + *b*
using *add-mono*[*of 0 a 0 b*] **by** *simp*

lemma *image-eqD*: *f* ‘ *A* = *B* ⟹ ∀ *x*∈*A*. *f x* ∈ *B*
by *auto*

lemma *incseq-setsumI*:
fixes *f* :: *nat* ⇒ *'a*::{*comm-monoid-add*,*ordered-ab-semigroup-add*}
assumes ⋀*i*. *0* ≤ *f i*
shows *incseq* (λ*i*. *setsum f* {..< *i*})
proof (*intro incseq-SucI*)
fix *n*
have *setsum f* {..< *n*} + *0* ≤ *setsum f* {..<*n*} + *f n*
using *assms* **by** (*rule add-left-mono*)
then show *setsum f* {..< *n*} ≤ *setsum f* {..< *Suc n*}
by *auto*
qed

lemma *incseq-setsumI2*:
fixes *f* :: *'i* ⇒ *nat* ⇒ *'a*::{*comm-monoid-add*,*ordered-ab-semigroup-add*}
assumes ⋀*n*. *n* ∈ *A* ⟹ *incseq* (*f n*)
shows *incseq* (λ*i*. ∑ *n*∈*A*. *f n i*)
using *assms*
unfolding *incseq-def* **by** (*auto intro*: *setsum-mono*)

lemma *setsum-ereal*[*simp*]: (∑ *x*∈*A*. *ereal* (*f x*)) = *ereal* (∑ *x*∈*A*. *f x*)
proof (*cases finite A*)
case *True*
then show *?thesis* **by** *induct auto*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *setsum-Pinfty*:
fixes $f :: 'a \Rightarrow ereal$
shows $(\sum x{\in}P.\ f\ x) = \infty \longleftrightarrow finite\ P \wedge (\exists\, i{\in}P.\ f\ i = \infty)$
proof *safe*
assume $*$: $setsum\ f\ P = \infty$
show *finite P*
proof (*rule ccontr*)
assume $\neg$ *finite P*
with $*$ **show** *False*
by *auto*
qed
show $\exists\, i{\in}P.\ f\ i = \infty$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then have $\bigwedge i.\ i \in P \Longrightarrow f\ i \neq \infty$
by *auto*
with ‹*finite P*› **have** $setsum\ f\ P \neq \infty$
by *induct auto*
with $*$ **show** *False*
by *auto*
qed
next
fix *i*
assume *finite P* **and** $i \in P$ **and** $f\ i = \infty$
then show $setsum\ f\ P = \infty$
proof *induct*
case (*insert x A*)
show *?case* **using** *insert* **by** (*cases* $x = i$) *auto*
qed *simp*
qed

lemma *setsum-Inf*:
fixes $f :: 'a \Rightarrow ereal$
shows $|setsum\ f\ A| = \infty \longleftrightarrow finite\ A \wedge (\exists\, i{\in}A.\ |f\ i| = \infty)$
proof
assume $*$: $|setsum\ f\ A| = \infty$
have *finite A*
by (*rule ccontr*) (*insert* $*$, *auto*)
moreover have $\exists\, i{\in}A.\ |f\ i| = \infty$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then have $\forall\, i{\in}A.\ \exists\, r.\ f\ i = ereal\ r$
by *auto*
from *bchoice*[*OF this*] **obtain** *r* **where** $\forall\, x{\in}A.\ f\ x = ereal\ (r\ x)$ **..**
with $*$ **show** *False*
by *auto*
qed
ultimately show $finite\ A \wedge (\exists\, i{\in}A.\ |f\ i| = \infty)$

```
    by auto
next
  assume finite A ∧ (∃ i∈A. |f i| = ∞)
  then obtain i where finite A i ∈ A and |f i| = ∞
    by auto
  then show |setsum f A| = ∞
  proof induct
    case (insert j A)
    then show ?case
      by (cases rule: ereal3-cases[of f i f j setsum f A]) auto
  qed simp
qed

lemma setsum-real-of-ereal:
  fixes f :: 'i ⇒ ereal
  assumes ⋀x. x ∈ S ⟹ |f x| ≠ ∞
  shows (∑ x∈S. real-of-ereal (f x)) = real-of-ereal (setsum f S)
proof −
  have ∀ x∈S. ∃ r. f x = ereal r
  proof
    fix x
    assume x ∈ S
    from assms[OF this] show ∃ r. f x = ereal r
      by (cases f x) auto
  qed
  from bchoice[OF this] obtain r where ∀ x∈S. f x = ereal (r x) ..
  then show ?thesis
    by simp
qed

lemma setsum-ereal-0:
  fixes f :: 'a ⇒ ereal
  assumes finite A
    and ⋀i. i ∈ A ⟹ 0 ≤ f i
  shows (∑ x∈A. f x) = 0 ⟷ (∀ i∈A. f i = 0)
proof
  assume setsum f A = 0 with assms show ∀ i∈A. f i = 0
  proof (induction A)
    case (insert a A)
    then have f a = 0 ∧ (∑ a∈A. f a) = 0
     by (subst ereal-add-nonneg-eq-0-iff[symmetric]) (simp-all add: setsum-nonneg)
    with insert show ?case
      by simp
  qed simp
qed auto
```

41.1.3 Multiplication

instantiation *ereal* :: {*comm-monoid-mult*,*sgn*}

begin

function *sgn-ereal* :: *ereal* ⇒ *ereal* **where**
  sgn (*ereal r*) = *ereal* (*sgn r*)
| *sgn* (∞::*ereal*) = *1*
| *sgn* (−∞::*ereal*) = *−1*
by (*auto intro*: *ereal-cases*)
termination by *standard* (*rule wf-empty*)

function *times-ereal* **where**
  ereal r ∗ *ereal p* = *ereal* (*r* ∗ *p*)
| *ereal r* ∗ ∞ = (*if r* = *0 then 0 else if r* > *0 then* ∞ *else* −∞)
| ∞ ∗ *ereal r* = (*if r* = *0 then 0 else if r* > *0 then* ∞ *else* −∞)
| *ereal r* ∗ −∞ = (*if r* = *0 then 0 else if r* > *0 then* −∞ *else* ∞)
| −∞ ∗ *ereal r* = (*if r* = *0 then 0 else if r* > *0 then* −∞ *else* ∞)
| (∞::*ereal*) ∗ ∞ = ∞
| −(∞::*ereal*) ∗ ∞ = −∞
| (∞::*ereal*) ∗ −∞ = −∞
| −(∞::*ereal*) ∗ −∞ = ∞
proof *goal-cases*
  case *prems*: (*1 P x*)
  then obtain *a b* **where** *x* = (*a*, *b*)
    by (*cases x*) *auto*
  with *prems* **show** *P*
    by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*
qed *simp-all*
termination by (*relation* {}) *simp*

instance
proof
  fix *a b c* :: *ereal*
  show *1* ∗ *a* = *a*
    by (*cases a*) (*simp-all add*: *one-ereal-def*)
  show *a* ∗ *b* = *b* ∗ *a*
    by (*cases rule*: *ereal2-cases*[*of a b*]) *simp-all*
  show *a* ∗ *b* ∗ *c* = *a* ∗ (*b* ∗ *c*)
    by (*cases rule*: *ereal3-cases*[*of a b c*])
      (*simp-all add*: *zero-ereal-def zero-less-mult-iff*)
qed

end

lemma [*simp*]:
  shows *ereal-1-times*: *ereal 1* ∗ *x* = *x*
  and *times-ereal-1*: *x* ∗ *ereal 1* = *x*
by(*simp-all add*: *one-ereal-def*[*symmetric*])

lemma *one-not-le-zero-ereal*[*simp*]: ¬ (*1* ≤ (*0*::*ereal*))
  by (*simp add*: *one-ereal-def zero-ereal-def*)

lemma *real-ereal-1*[*simp*]: *real-of-ereal* (*1*::*ereal*) = *1*
unfolding *one-ereal-def* **by** *simp*

lemma *real-of-ereal-le-1*:
fixes *a* :: *ereal*
shows *a* ≤ *1* ⟹ *real-of-ereal a* ≤ *1*
by (*cases a*) (*auto simp*: *one-ereal-def*)

lemma *abs-ereal-one*[*simp*]: |*1*| = (*1*::*ereal*)
unfolding *one-ereal-def* **by** *simp*

lemma *ereal-mult-zero*[*simp*]:
fixes *a* :: *ereal*
shows *a* ∗ *0* = *0*
by (*cases a*) (*simp-all add*: *zero-ereal-def*)

lemma *ereal-zero-mult*[*simp*]:
fixes *a* :: *ereal*
shows *0* ∗ *a* = *0*
by (*cases a*) (*simp-all add*: *zero-ereal-def*)

lemma *ereal-m1-less-0*[*simp*]: −(*1*::*ereal*) < *0*
by (*simp add*: *zero-ereal-def one-ereal-def*)

lemma *ereal-times*[*simp*]:
1 ≠ (∞::*ereal*) (∞::*ereal*) ≠ *1*
1 ≠ −(∞::*ereal*) −(∞::*ereal*) ≠ *1*
by (*auto simp*: *one-ereal-def*)

lemma *ereal-plus-1*[*simp*]:
1 + *ereal r* = *ereal* (*r* + *1*)
ereal r + *1* = *ereal* (*r* + *1*)
1 + −(∞::*ereal*) = −∞
−(∞::*ereal*) + *1* = −∞
unfolding *one-ereal-def* **by** *auto*

lemma *ereal-zero-times*[*simp*]:
fixes *a b* :: *ereal*
shows *a* ∗ *b* = *0* ⟷ *a* = *0* ∨ *b* = *0*
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-mult-eq-PInfty*[*simp*]:
a ∗ *b* = (∞::*ereal*) ⟷
(*a* = ∞ ∧ *b* > *0*) ∨ (*a* > *0* ∧ *b* = ∞) ∨ (*a* = −∞ ∧ *b* < *0*) ∨ (*a* < *0* ∧ *b* = −∞)
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-mult-eq-MInfty*[*simp*]:

$a * b = -(\infty::ereal) \longleftrightarrow$
$(a = \infty \wedge b < 0) \vee (a < 0 \wedge b = \infty) \vee (a = -\infty \wedge b > 0) \vee (a > 0 \wedge b = -\infty)$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-abs-mult*: $|x * y :: ereal| = |x| * |y|$
by (*cases x y rule*: *ereal2-cases*) (*auto simp*: *abs-mult*)

lemma *ereal-0-less-1*[*simp*]: $0 < (1::ereal)$
by (*simp-all add*: *zero-ereal-def one-ereal-def*)

lemma *ereal-mult-minus-left*[*simp*]:
fixes $a\ b :: ereal$
shows $-a * b = -(a * b)$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-mult-minus-right*[*simp*]:
fixes $a\ b :: ereal$
shows $a * -b = -(a * b)$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-mult-infty*[*simp*]:
$a * (\infty::ereal) = (if\ a = 0\ then\ 0\ else\ if\ 0 < a\ then\ \infty\ else\ -\infty)$
by (*cases a*) *auto*

lemma *ereal-infty-mult*[*simp*]:
$(\infty::ereal) * a = (if\ a = 0\ then\ 0\ else\ if\ 0 < a\ then\ \infty\ else\ -\infty)$
by (*cases a*) *auto*

lemma *ereal-mult-strict-right-mono*:
assumes $a < b$
and $0 < c$
and $c < (\infty::ereal)$
shows $a * c < b * c$
using *assms*
by (*cases rule*: *ereal3-cases*[*of a b c*]) (*auto simp*: *zero-le-mult-iff*)

lemma *ereal-mult-strict-left-mono*:
$a < b \Longrightarrow 0 < c \Longrightarrow c < (\infty::ereal) \Longrightarrow c * a < c * b$
using *ereal-mult-strict-right-mono*
by (*simp add*: *mult.commute*[*of c*])

lemma *ereal-mult-right-mono*:
fixes $a\ b\ c :: ereal$
shows $a \leq b \Longrightarrow 0 \leq c \Longrightarrow a * c \leq b * c$
using *assms*
apply (*cases c = 0*)
apply *simp*
apply (*cases rule*: *ereal3-cases*[*of a b c*])

apply (*auto simp*: *zero-le-mult-iff*)
done

lemma *ereal-mult-left-mono*:
fixes $a\ b\ c :: ereal$
shows $a \leq b \Longrightarrow 0 \leq c \Longrightarrow c * a \leq c * b$
using *ereal-mult-right-mono*
by (*simp add*: *mult.commute*[*of c*])

lemma *zero-less-one-ereal*[*simp*]: $0 \leq (1::ereal)$
by (*simp add*: *one-ereal-def zero-ereal-def*)

lemma *ereal-0-le-mult*[*simp*]: $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 \leq a * (b :: ereal)$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-right-distrib*:
fixes $r\ a\ b :: ereal$
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow r * (a + b) = r * a + r * b$
by (*cases rule*: *ereal3-cases*[*of r a b*]) (*simp-all add*: *field-simps*)

lemma *ereal-left-distrib*:
fixes $r\ a\ b :: ereal$
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow (a + b) * r = a * r + b * r$
by (*cases rule*: *ereal3-cases*[*of r a b*]) (*simp-all add*: *field-simps*)

lemma *ereal-mult-le-0-iff*:
fixes $a\ b :: ereal$
shows $a * b \leq 0 \longleftrightarrow (0 \leq a \wedge b \leq 0) \vee (a \leq 0 \wedge 0 \leq b)$
by (*cases rule*: *ereal2-cases*[*of a b*]) (*simp-all add*: *mult-le-0-iff*)

lemma *ereal-zero-le-0-iff*:
fixes $a\ b :: ereal$
shows $0 \leq a * b \longleftrightarrow (0 \leq a \wedge 0 \leq b) \vee (a \leq 0 \wedge b \leq 0)$
by (*cases rule*: *ereal2-cases*[*of a b*]) (*simp-all add*: *zero-le-mult-iff*)

lemma *ereal-mult-less-0-iff*:
fixes $a\ b :: ereal$
shows $a * b < 0 \longleftrightarrow (0 < a \wedge b < 0) \vee (a < 0 \wedge 0 < b)$
by (*cases rule*: *ereal2-cases*[*of a b*]) (*simp-all add*: *mult-less-0-iff*)

lemma *ereal-zero-less-0-iff*:
fixes $a\ b :: ereal$
shows $0 < a * b \longleftrightarrow (0 < a \wedge 0 < b) \vee (a < 0 \wedge b < 0)$
by (*cases rule*: *ereal2-cases*[*of a b*]) (*simp-all add*: *zero-less-mult-iff*)

lemma *ereal-left-mult-cong*:
fixes $a\ b\ c :: ereal$
shows $c = d \Longrightarrow (d \neq 0 \Longrightarrow a = b) \Longrightarrow a * c = b * d$
by (*cases* $c = 0$) *simp-all*

lemma *ereal-right-mult-cong*:
fixes $a\ b\ c :: ereal$
shows $c = d \Longrightarrow (d \neq 0 \Longrightarrow a = b) \Longrightarrow c * a = d * b$
by (*cases* $c = 0$) *simp-all*

lemma *ereal-distrib*:
fixes $a\ b\ c :: ereal$
assumes $a \neq \infty \vee b \neq -\infty$
and $a \neq -\infty \vee b \neq \infty$
and $|c| \neq \infty$
shows $(a + b) * c = a * c + b * c$
using *assms*
by (*cases rule*: *ereal3-cases*[*of a b c*]) (*simp-all add*: *field-simps*)

lemma *numeral-eq-ereal* [*simp*]: $numeral\ w = ereal\ (numeral\ w)$
apply (*induct w rule*: *num-induct*)
apply (*simp only*: *numeral-One one-ereal-def*)
apply (*simp only*: *numeral-inc ereal-plus-1*)
done

lemma *distrib-left-ereal-nn*:
$c \geq 0 \Longrightarrow (x + y) * ereal\ c = x * ereal\ c + y * ereal\ c$
by(*cases x y rule*: *ereal2-cases*)(*simp-all add*: *ring-distribs*)

lemma *setsum-ereal-right-distrib*:
fixes $f :: 'a \Rightarrow ereal$
shows $(\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i) \Longrightarrow r * setsum\ f\ A = (\sum n{\in}A.\ r * f\ n)$
by (*induct A rule*: *infinite-finite-induct*) (*auto simp*: *ereal-right-distrib setsum-nonneg*)

lemma *setsum-ereal-left-distrib*:
$(\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i) \Longrightarrow setsum\ f\ A * r = (\sum n{\in}A.\ f\ n * r :: ereal)$
using *setsum-ereal-right-distrib*[*of A f r*] **by** (*simp add*: *mult-ac*)

lemma *setsum-left-distrib-ereal*:
$c \geq 0 \Longrightarrow setsum\ f\ A * ereal\ c = (\sum x{\in}A.\ f\ x * c :: ereal)$
by(*subst setsum-comp-morphism*[**where** $h{=}\lambda x.\ x * ereal\ c$, *symmetric*])(*simp-all add*: *distrib-left-ereal-nn*)

lemma *ereal-le-epsilon*:
fixes $x\ y :: ereal$
assumes $\forall e.\ 0 < e \longrightarrow x \leq y + e$
shows $x \leq y$
proof −
{
assume *a*: $\exists r.\ y = ereal\ r$
then obtain *r* **where** *r-def*: $y = ereal\ r$
by *auto*
{

```
    assume x = −∞
    then have ?thesis by auto
  }
  moreover
  {
    assume x ≠ −∞
    then obtain p where p-def: x = ereal p
    using a assms[rule-format, of 1]
      by (cases x) auto
    {
      fix e
      have 0 < e ⟶ p ≤ r + e
        using assms[rule-format, of ereal e] p-def r-def by auto
    }
    then have p ≤ r
      apply (subst field-le-epsilon)
      apply auto
      done
    then have ?thesis
      using r-def p-def by auto
  }
  ultimately have ?thesis
    by blast
}
moreover
{
  assume y = −∞ | y = ∞
  then have ?thesis
    using assms[rule-format, of 1] by (cases x) auto
}
ultimately show ?thesis
  by (cases y) auto
qed

lemma ereal-le-epsilon2:
  fixes x y :: ereal
  assumes ∀ e. 0 < e ⟶ x ≤ y + ereal e
  shows x ≤ y
proof −
  {
    fix e :: ereal
    assume e > 0
    {
      assume e = ∞
      then have x ≤ y + e
        by auto
    }
    moreover
    {
```

```
    assume e ≠ ∞
    then obtain r where e = ereal r
      using ‹e > 0› by (cases e) auto
    then have x ≤ y + e
      using assms[rule-format, of r] ‹e>0› by auto
  }
  ultimately have x ≤ y + e
    by blast
}
then show ?thesis
  using ereal-le-epsilon by auto
qed

lemma ereal-le-real:
  fixes x y :: ereal
  assumes ∀ z. x ≤ ereal z ⟶ y ≤ ereal z
  shows y ≤ x
 by (metis assms ereal-bot ereal-cases ereal-infty-less-eq(2) ereal-less-eq(1) linorder-le-cases)

lemma setprod-ereal-0:
  fixes f :: 'a ⇒ ereal
  shows (∏ i∈A. f i) = 0 ⟷ finite A ∧ (∃ i∈A. f i = 0)
proof (cases finite A)
  case True
  then show ?thesis by (induct A) auto
next
  case False
  then show ?thesis by auto
qed

lemma setprod-ereal-pos:
  fixes f :: 'a ⇒ ereal
  assumes pos: ⋀i. i ∈ I ⟹ 0 ≤ f i
  shows 0 ≤ (∏ i∈I. f i)
proof (cases finite I)
  case True
  from this pos show ?thesis
    by induct auto
next
  case False
  then show ?thesis by simp
qed

lemma setprod-PInf:
  fixes f :: 'a ⇒ ereal
  assumes ⋀i. i ∈ I ⟹ 0 ≤ f i
  shows (∏ i∈I. f i) = ∞ ⟷ finite I ∧ (∃ i∈I. f i = ∞) ∧ (∀ i∈I. f i ≠ 0)
proof (cases finite I)
  case True
```

from *this assms* **show** *?thesis*
proof (*induct I*)
case (*insert i I*)
then have *pos*: $0 \leq f\ i\ 0 \leq setprod\ f\ I$
by (*auto intro*!: *setprod-ereal-pos*)
from *insert* **have** $(\prod j{\in}insert\ i\ I.\ f\ j) = \infty \longleftrightarrow setprod\ f\ I * f\ i = \infty$
by *auto*
also have $\ldots \longleftrightarrow (setprod\ f\ I = \infty \vee f\ i = \infty) \wedge f\ i \neq 0 \wedge setprod\ f\ I \neq 0$
using *setprod-ereal-pos*[*of I f*] *pos*
by (*cases rule*: *ereal2-cases*[*of f i setprod f I*]) *auto*
also have $\ldots \longleftrightarrow$ *finite* (*insert i I*) $\wedge$ $(\exists j{\in}insert\ i\ I.\ f\ j = \infty) \wedge (\forall j{\in}insert\ i\ I.\ f\ j \neq 0)$
using *insert* **by** (*auto simp*: *setprod-ereal-0*)
finally show *?case* **.**
qed *simp*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *setprod-ereal*: $(\prod i{\in}A.\ ereal\ (f\ i)) = ereal\ (setprod\ f\ A)$
proof (*cases finite A*)
case *True*
then show *?thesis*
by *induct* (*auto simp*: *one-ereal-def*)
next
case *False*
then show *?thesis*
by (*simp add*: *one-ereal-def*)
qed

41.1.4 Power

lemma *ereal-power*[*simp*]: $(ereal\ x)\ \hat{}\ n = ereal\ (x\hat{}n)$
by (*induct n*) (*auto simp*: *one-ereal-def*)

lemma *ereal-power-PInf*[*simp*]: $(\infty{::}ereal)\ \hat{}\ n = (if\ n = 0\ then\ 1\ else\ \infty)$
by (*induct n*) (*auto simp*: *one-ereal-def*)

lemma *ereal-power-uminus*[*simp*]:
fixes $x :: ereal$
shows $(-x)\ \hat{}\ n = (if\ even\ n\ then\ x\ \hat{}\ n\ else\ -(x\hat{}n))$
by (*induct n*) (*auto simp*: *one-ereal-def*)

lemma *ereal-power-numeral*[*simp*]:
$(numeral\ num :: ereal)\ \hat{}\ n = ereal\ (numeral\ num\ \hat{}\ n)$
by (*induct n*) (*auto simp*: *one-ereal-def*)

lemma *zero-le-power-ereal*[*simp*]:

fixes a :: *ereal*
assumes $0 \leq a$
shows $0 \leq a \hat{} n$
using *assms* **by** (*induct n*) (*auto simp*: *ereal-zero-le-0-iff*)

41.1.5 Subtraction

lemma *ereal-minus-minus-image*[*simp*]:
fixes S :: *ereal set*
shows *uminus* ‘ *uminus* ‘ $S = S$
by (*auto simp*: *image-iff*)

lemma *ereal-uminus-lessThan*[*simp*]:
fixes a :: *ereal*
shows *uminus* ‘ $\{..<a\} = \{-a<..\}$
proof −
{
fix x
assume $-a < x$
then have $-x < -(-a)$
by (*simp del*: *ereal-uminus-uminus*)
then have $-x < a$
by *simp*
}
then show *?thesis*
by *force*
qed

lemma *ereal-uminus-greaterThan*[*simp*]: *uminus* ‘ $\{(a::ereal)<..\} = \{..<-a\}$
by (*metis ereal-uminus-lessThan ereal-uminus-uminus ereal-minus-minus-image*)

instantiation *ereal* :: *minus*
begin

definition $x - y = x + -(y::ereal)$
instance ..

end

lemma *ereal-minus*[*simp*]:
ereal r − *ereal* p = *ereal* $(r - p)$
$-\infty$ − *ereal* $r = -\infty$
ereal $r - \infty = -\infty$
$(\infty::ereal) - x = \infty$
$-(\infty::ereal) - \infty = -\infty$
$x - -y = x + y$
$x - 0 = x$
$0 - x = -x$
by (*simp-all add*: *minus-ereal-def*)

lemma *ereal-x-minus-x*[*simp*]: $x - x = (\mathit{if}\ |x| = \infty\ \mathit{then}\ \infty\ \mathit{else}\ 0{::}\mathit{ereal})$
by (*cases x*) *simp-all*

lemma *ereal-eq-minus-iff*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $x = z - y \longleftrightarrow$
$(|y| \neq \infty \longrightarrow x + y = z) \wedge$
$(y = -\infty \longrightarrow x = \infty) \wedge$
$(y = \infty \longrightarrow z = \infty \longrightarrow x = \infty) \wedge$
$(y = \infty \longrightarrow z \neq \infty \longrightarrow x = -\infty)$
by (*cases rule*: *ereal3-cases*[*of x y z*]) *auto*

lemma *ereal-eq-minus*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $|y| \neq \infty \Longrightarrow x = z - y \longleftrightarrow x + y = z$
by (*auto simp*: *ereal-eq-minus-iff*)

lemma *ereal-less-minus-iff*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $x < z - y \longleftrightarrow$
$(y = \infty \longrightarrow z = \infty \wedge x \neq \infty) \wedge$
$(y = -\infty \longrightarrow x \neq \infty) \wedge$
$(|y| \neq \infty \longrightarrow x + y < z)$
by (*cases rule*: *ereal3-cases*[*of x y z*]) *auto*

lemma *ereal-less-minus*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $|y| \neq \infty \Longrightarrow x < z - y \longleftrightarrow x + y < z$
by (*auto simp*: *ereal-less-minus-iff*)

lemma *ereal-le-minus-iff*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $x \leq z - y \longleftrightarrow (y = \infty \longrightarrow z \neq \infty \longrightarrow x = -\infty) \wedge (|y| \neq \infty \longrightarrow x + y \leq z)$
by (*cases rule*: *ereal3-cases*[*of x y z*]) *auto*

lemma *ereal-le-minus*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $|y| \neq \infty \Longrightarrow x \leq z - y \longleftrightarrow x + y \leq z$
by (*auto simp*: *ereal-le-minus-iff*)

lemma *ereal-minus-less-iff*:
fixes $x\ y\ z :: \mathit{ereal}$
shows $x - y < z \longleftrightarrow y \neq -\infty \wedge (y = \infty \longrightarrow x \neq \infty \wedge z \neq -\infty) \wedge (y \neq \infty \longrightarrow x < z + y)$
by (*cases rule*: *ereal3-cases*[*of x y z*]) *auto*

lemma *ereal-minus-less*:

fixes $x\ y\ z$ :: *ereal*
shows $|y| \neq \infty \Longrightarrow x - y < z \longleftrightarrow x < z + y$
by (*auto simp*: *ereal-minus-less-iff*)

lemma *ereal-minus-le-iff*:
fixes $x\ y\ z$ :: *ereal*
shows $x - y \leq z \longleftrightarrow$
$(y = -\infty \longrightarrow z = \infty) \wedge$
$(y = \infty \longrightarrow x = \infty \longrightarrow z = \infty) \wedge$
$(|y| \neq \infty \longrightarrow x \leq z + y)$
by (*cases rule*: *ereal3-cases*[*of x y z*]) *auto*

lemma *ereal-minus-le*:
fixes $x\ y\ z$ :: *ereal*
shows $|y| \neq \infty \Longrightarrow x - y \leq z \longleftrightarrow x \leq z + y$
by (*auto simp*: *ereal-minus-le-iff*)

lemma *ereal-minus-eq-minus-iff*:
fixes $a\ b\ c$ :: *ereal*
shows $a - b = a - c \longleftrightarrow$
$b = c \vee a = \infty \vee (a = -\infty \wedge b \neq -\infty \wedge c \neq -\infty)$
by (*cases rule*: *ereal3-cases*[*of a b c*]) *auto*

lemma *ereal-add-le-add-iff*:
fixes $a\ b\ c$ :: *ereal*
shows $c + a \leq c + b \longleftrightarrow$
$a \leq b \vee c = \infty \vee (c = -\infty \wedge a \neq \infty \wedge b \neq \infty)$
by (*cases rule*: *ereal3-cases*[*of a b c*]) (*simp-all add*: *field-simps*)

lemma *ereal-add-le-add-iff2*:
fixes $a\ b\ c$ :: *ereal*
shows $a + c \leq b + c \longleftrightarrow a \leq b \vee c = \infty \vee (c = -\infty \wedge a \neq \infty \wedge b \neq \infty)$
by(*cases rule*: *ereal3-cases*[*of a b c*])(*simp-all add*: *field-simps*)

lemma *ereal-mult-le-mult-iff*:
fixes $a\ b\ c$ :: *ereal*
shows $|c| \neq \infty \Longrightarrow c * a \leq c * b \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
by (*cases rule*: *ereal3-cases*[*of a b c*]) (*simp-all add*: *mult-le-cancel-left*)

lemma *ereal-minus-mono*:
fixes $A\ B\ C\ D$ :: *ereal* **assumes** $A \leq B\ D \leq C$
shows $A - C \leq B - D$
using *assms*
by (*cases rule*: *ereal3-cases*[*case-product ereal-cases*, *of A B C D*]) *simp-all*

lemma *real-of-ereal-minus*:
fixes $a\ b$ :: *ereal*
shows *real-of-ereal* $(a - b) =$ (*if* $|a| = \infty \vee |b| = \infty$ *then 0 else real-of-ereal a* $-$ *real-of-ereal b*)

by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *real-of-ereal-minus'*: $|x| = \infty \longleftrightarrow |y| = \infty \Longrightarrow$ *real-of-ereal* $x -$ *real-of-ereal* $y =$ *real-of-ereal* $(x - y ::$ *ereal*$)$
by(*subst real-of-ereal-minus*) *auto*

lemma *ereal-diff-positive*:
fixes $a\ b ::$ *ereal* **shows** $a \leq b \Longrightarrow 0 \leq b - a$
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-between*:
fixes $x\ e ::$ *ereal*
assumes $|x| \neq \infty$
and $0 < e$
shows $x - e < x$
and $x < x + e$
using *assms*
apply (*cases x*, *cases e*)
apply *auto*
using *assms*
apply (*cases x*, *cases e*)
apply *auto*
done

lemma *ereal-minus-eq-PInfty-iff*:
fixes $x\ y ::$ *ereal*
shows $x - y = \infty \longleftrightarrow y = -\infty \vee x = \infty$
by (*cases x y rule*: *ereal2-cases*) *simp-all*

lemma *ereal-diff-add-eq-diff-diff-swap*:
fixes $x\ y\ z ::$ *ereal*
shows $|y| \neq \infty \Longrightarrow x - (y + z) = x - y - z$
by(*cases x y z rule*: *ereal3-cases*) *simp-all*

lemma *ereal-diff-add-assoc2*:
fixes $x\ y\ z ::$ *ereal*
shows $x + y - z = x - z + y$
by(*cases x y z rule*: *ereal3-cases*) *simp-all*

lemma *ereal-add-uminus-conv-diff*: **fixes** $x\ y\ z ::$ *ereal* **shows** $-x + y = y - x$
by(*cases x y rule*: *ereal2-cases*) *simp-all*

lemma *ereal-minus-diff-eq*:
fixes $x\ y ::$ *ereal*
shows $[\![\ x = \infty \longrightarrow y \neq \infty; x = -\infty \longrightarrow y \neq -\infty\ ]\!] \Longrightarrow -(x - y) = y - x$
by(*cases x y rule*: *ereal2-cases*) *simp-all*

lemma *ediff-le-self* [*simp*]: $x - y \leq (x ::$ *enat*$)$
by(*cases x y rule*: *enat.exhaust*[*case-product enat.exhaust*]) *simp-all*

41.1.6 Division

instantiation *ereal* :: *inverse*
begin

function *inverse-ereal* **where**
  inverse (*ereal r*) = (*if r* = *0 then* ∞ *else ereal* (*inverse r*))
| *inverse* (∞::*ereal*) = *0*
| *inverse* (−∞::*ereal*) = *0*
  by (*auto intro*: *ereal-cases*)
termination by (*relation* {}) *simp*

definition *x div y* = *x* ∗ *inverse* (*y* :: *ereal*)

instance ..

end

lemma *real-of-ereal-inverse*[*simp*]:
  fixes *a* :: *ereal*
  shows *real-of-ereal* (*inverse a*) = *1* / *real-of-ereal a*
  by (*cases a*) (*auto simp*: *inverse-eq-divide*)

lemma *ereal-inverse*[*simp*]:
  inverse (*0*::*ereal*) = ∞
  inverse (*1*::*ereal*) = *1*
  by (*simp-all add*: *one-ereal-def zero-ereal-def*)

lemma *ereal-divide*[*simp*]:
  ereal r / *ereal p* = (*if p* = *0 then ereal r* ∗ ∞ *else ereal* (*r* / *p*))
  unfolding *divide-ereal-def* **by** (*auto simp*: *divide-real-def*)

lemma *ereal-divide-same*[*simp*]:
  fixes *x* :: *ereal*
  shows *x* / *x* = (*if* |*x*| = ∞ ∨ *x* = *0 then 0 else 1*)
  by (*cases x*) (*simp-all add*: *divide-real-def divide-ereal-def one-ereal-def*)

lemma *ereal-inv-inv*[*simp*]:
  fixes *x* :: *ereal*
  shows *inverse* (*inverse x*) = (*if x* ≠ −∞ *then x else* ∞)
  by (*cases x*) *auto*

lemma *ereal-inverse-minus*[*simp*]:
  fixes *x* :: *ereal*
  shows *inverse* (− *x*) = (*if x* = *0 then* ∞ *else* −*inverse x*)
  by (*cases x*) *simp-all*

lemma *ereal-uminus-divide*[*simp*]:
  fixes *x y* :: *ereal*
  shows − *x* / *y* = − (*x* / *y*)

unfolding *divide-ereal-def* **by** *simp*

lemma *ereal-divide-Infty*[*simp*]:
fixes x :: *ereal*
shows x / ∞ = *0* x / $-\infty$ = *0*
unfolding *divide-ereal-def* **by** *simp-all*

lemma *ereal-divide-one*[*simp*]: x / *1* = (x::*ereal*)
unfolding *divide-ereal-def* **by** *simp*

lemma *ereal-divide-ereal*[*simp*]: ∞ / *ereal* r = (*if* *0* $\leq$ r *then* ∞ *else* $-\infty$)
unfolding *divide-ereal-def* **by** *simp*

lemma *ereal-inverse-nonneg-iff*: *0* $\leq$ *inverse* (x :: *ereal*) $\longleftrightarrow$ *0* $\leq$ x $\vee$ x = $-\infty$
by (*cases* x) *auto*

lemma *inverse-ereal-ge0I*: *0* $\leq$ (x :: *ereal*) $\Longrightarrow$ *0* $\leq$ *inverse* x
by(*cases* x) *simp-all*

lemma *zero-le-divide-ereal*[*simp*]:
fixes a :: *ereal*
assumes *0* $\leq$ a
and *0* $\leq$ b
shows *0* $\leq$ a / b
using *assms* **by** (*cases rule*: *ereal2-cases*[*of* a b]) (*auto simp*: *zero-le-divide-iff*)

lemma *ereal-le-divide-pos*:
fixes x y z :: *ereal*
shows x > *0* $\Longrightarrow$ x $\neq$ ∞ $\Longrightarrow$ y $\leq$ z / x $\longleftrightarrow$ x * y $\leq$ z
by (*cases rule*: *ereal3-cases*[*of* x y z]) (*auto simp*: *field-simps*)

lemma *ereal-divide-le-pos*:
fixes x y z :: *ereal*
shows x > *0* $\Longrightarrow$ x $\neq$ ∞ $\Longrightarrow$ z / x $\leq$ y $\longleftrightarrow$ z $\leq$ x * y
by (*cases rule*: *ereal3-cases*[*of* x y z]) (*auto simp*: *field-simps*)

lemma *ereal-le-divide-neg*:
fixes x y z :: *ereal*
shows x < *0* $\Longrightarrow$ x $\neq$ $-\infty$ $\Longrightarrow$ y $\leq$ z / x $\longleftrightarrow$ z $\leq$ x * y
by (*cases rule*: *ereal3-cases*[*of* x y z]) (*auto simp*: *field-simps*)

lemma *ereal-divide-le-neg*:
fixes x y z :: *ereal*
shows x < *0* $\Longrightarrow$ x $\neq$ $-\infty$ $\Longrightarrow$ z / x $\leq$ y $\longleftrightarrow$ x * y $\leq$ z
by (*cases rule*: *ereal3-cases*[*of* x y z]) (*auto simp*: *field-simps*)

lemma *ereal-inverse-antimono-strict*:
fixes x y :: *ereal*
shows *0* $\leq$ x $\Longrightarrow$ x < y $\Longrightarrow$ *inverse* y < *inverse* x

by (*cases rule*: *ereal2-cases*[*of x y*]) *auto*

lemma *ereal-inverse-antimono*:
fixes *x y* :: *ereal*
shows $0 \leq x \Longrightarrow x \leq y \Longrightarrow$ *inverse y* $\leq$ *inverse x*
by (*cases rule*: *ereal2-cases*[*of x y*]) *auto*

lemma *inverse-inverse-Pinfty-iff*[*simp*]:
fixes *x* :: *ereal*
shows *inverse x* $= \infty \longleftrightarrow x = 0$
by (*cases x*) *auto*

lemma *ereal-inverse-eq-0*:
fixes *x* :: *ereal*
shows *inverse x* $= 0 \longleftrightarrow x = \infty \vee x = -\infty$
by (*cases x*) *auto*

lemma *ereal-0-gt-inverse*:
fixes *x* :: *ereal*
shows $0 <$ *inverse x* $\longleftrightarrow x \neq \infty \wedge 0 \leq x$
by (*cases x*) *auto*

lemma *ereal-inverse-le-0-iff*:
fixes *x* :: *ereal*
shows *inverse x* $\leq 0 \longleftrightarrow x < 0 \vee x = \infty$
by(*cases x*) *auto*

lemma *ereal-divide-eq-0-iff*: *x / y* $= 0 \longleftrightarrow x = 0 \vee |y :: ereal| = \infty$
by(*cases x y rule*: *ereal2-cases*) *simp-all*

lemma *ereal-mult-less-right*:
fixes *a b c* :: *ereal*
assumes $b * a < c * a$
and $0 < a$
and $a < \infty$
shows $b < c$
using *assms*
by (*cases rule*: *ereal3-cases*[*of a b c*])
(*auto split*: *split-if-asm simp*: *zero-less-mult-iff zero-le-mult-iff*)

lemma *ereal-mult-divide*: **fixes** *a b* :: *ereal* **shows** $0 < b \Longrightarrow b < \infty \Longrightarrow b * (a$ / $b) = a$
by (*cases a b rule*: *ereal2-cases*) *auto*

lemma *ereal-power-divide*:
fixes *x y* :: *ereal*
shows $y \neq 0 \Longrightarrow (x$ / $y)$ ^ $n = x$^n / y^n
by (*cases rule*: *ereal2-cases* [*of x y*])
(*auto simp*: *one-ereal-def zero-ereal-def power-divide zero-le-power-eq*)

lemma *ereal-le-mult-one-interval*:
  fixes x y :: *ereal*
  assumes y: $y \neq -\infty$
  assumes z: $\bigwedge z.\ 0 < z \Longrightarrow z < 1 \Longrightarrow z * x \leq y$
  shows $x \leq y$
proof (*cases* x)
  case *PInf*
  with z[*of 1 / 2*] **show** $x \leq y$
    by (*simp add*: *one-ereal-def*)
next
  case (*real* r)
  note r = *this*
  show $x \leq y$
  proof (*cases* y)
    case (*real* p)
    note p = *this*
    have $r \leq p$
    proof (*rule field-le-mult-one-interval*)
      fix z :: *real*
      assume *0* $< z$ **and** $z <$ *1*
      with z[*of ereal z*] **show** $z * r \leq p$
        using p r **by** (*auto simp*: *zero-le-mult-iff one-ereal-def*)
    qed
    then show $x \leq y$
      using p r **by** *simp*
  qed (*insert* y, *simp-all*)
qed *simp*

lemma *ereal-divide-right-mono*[*simp*]:
  fixes x y z :: *ereal*
  assumes $x \leq y$
    and *0* $< z$
  shows x / $z \leq y$ / z
  using *assms* **by** (*cases x y z rule*: *ereal3-cases*) (*auto intro*: *divide-right-mono*)

lemma *ereal-divide-left-mono*[*simp*]:
  fixes x y z :: *ereal*
  assumes $y \leq x$
    and *0* $< z$
    and *0* $< x * y$
  shows z / $x \leq z$ / y
  using *assms*
  by (*cases x y z rule*: *ereal3-cases*)
    (*auto intro*: *divide-left-mono simp*: *field-simps zero-less-mult-iff mult-less-0-iff split*: *split-if-asm*)

lemma *ereal-divide-zero-left*[*simp*]:
  fixes a :: *ereal*

shows *0 / a = 0*
by (*cases a*) (*auto simp*: *zero-ereal-def*)

lemma *ereal-times-divide-eq-left*[*simp*]:
fixes *a b c* :: *ereal*
shows *b / c * a = b * a / c*
by (*cases a b c rule*: *ereal3-cases*) (*auto simp*: *field-simps zero-less-mult-iff mult-less-0-iff*)

lemma *ereal-times-divide-eq*: *a * (b / c* :: *ereal*) = *a * b / c*
by (*cases a b c rule*: *ereal3-cases*)
(*auto simp*: *field-simps zero-less-mult-iff*)

lemma *ereal-inverse-real*: $|z| \neq \infty \Longrightarrow z \neq 0 \Longrightarrow$ *ereal* (*inverse* (*real-of-ereal z*))
= *inverse z*
by (*cases z*) *simp-all*

lemma *ereal-inverse-mult*:
$a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow$ *inverse* (*a* * (*b*::*ereal*)) = *inverse a * inverse b*
by (*cases a*; *cases b*) *auto*

41.2 Complete lattice

instantiation *ereal* :: *lattice*
begin

definition [*simp*]: *sup x y* = (*max x y* :: *ereal*)
definition [*simp*]: *inf x y* = (*min x y* :: *ereal*)
instance by *standard simp-all*

end

instantiation *ereal* :: *complete-lattice*
begin

definition *bot* = (−∞::*ereal*)
definition *top* = (∞::*ereal*)

definition *Sup S* = (*SOME x* :: *ereal*. (∀ *y*∈*S*. *y* ≤ *x*) ∧ (∀ *z*. (∀ *y*∈*S*. *y* ≤ *z*) ⟶
x ≤ *z*))
definition *Inf S* = (*SOME x* :: *ereal*. (∀ *y*∈*S*. *x* ≤ *y*) ∧ (∀ *z*. (∀ *y*∈*S*. *z* ≤ *y*) ⟶
z ≤ *x*))

lemma *ereal-complete-Sup*:
fixes *S* :: *ereal set*
shows ∃ *x*. (∀ *y*∈*S*. *y* ≤ *x*) ∧ (∀ *z*. (∀ *y*∈*S*. *y* ≤ *z*) ⟶ *x* ≤ *z*)
proof (*cases* ∃ *x*. ∀ *a*∈*S*. *a* ≤ *ereal x*)
case *True*
then obtain *y* **where** *y*: ⋀*a*. *a*∈*S* ⟹ *a* ≤ *ereal y*
by *auto*

```
  then have ∞ ∉ S
    by force
  show ?thesis
  proof (cases S ≠ {-∞} ∧ S ≠ {})
    case True
    with ‹∞ ∉ S› obtain x where x: x ∈ S |x| ≠ ∞
      by auto
    obtain s where s: ∀x∈ereal -` S. x ≤ s ⋀z. (∀x∈ereal -` S. x ≤ z) ⟹ s
≤ z
    proof (atomize-elim, rule complete-real)
      show ∃x. x ∈ ereal -` S
        using x by auto
      show ∃z. ∀x∈ereal -` S. x ≤ z
        by (auto dest: y intro!: exI[of - y])
    qed
    show ?thesis
    proof (safe intro!: exI[of - ereal s])
      fix y
      assume y ∈ S
      with s ‹∞ ∉ S› show y ≤ ereal s
        by (cases y) auto
    next
      fix z
      assume ∀y∈S. y ≤ z
      with ‹S ≠ {-∞} ∧ S ≠ {}› show ereal s ≤ z
        by (cases z) (auto intro!: s)
    qed
  next
    case False
    then show ?thesis
      by (auto intro!: exI[of - -∞])
  qed
next
  case False
  then show ?thesis
    by (fastforce intro!: exI[of - ∞] ereal-top intro: order-trans dest: less-imp-le
simp: not-le)
qed

lemma ereal-complete-uminus-eq:
  fixes S :: ereal set
  shows (∀y∈uminus`S. y ≤ x) ∧ (∀z. (∀y∈uminus`S. y ≤ z) ⟶ x ≤ z)
    ⟷ (∀y∈S. -x ≤ y) ∧ (∀z. (∀y∈S. z ≤ y) ⟶ z ≤ -x)
  by simp (metis ereal-minus-le-minus ereal-uminus-uminus)

lemma ereal-complete-Inf:
  ∃x. (∀y∈S::ereal set. x ≤ y) ∧ (∀z. (∀y∈S. z ≤ y) ⟶ z ≤ x)
  using ereal-complete-Sup[of uminus ` S]
  unfolding ereal-complete-uminus-eq
```

by *auto*

instance
proof
show *Sup* {} = (*bot*::*ereal*)
apply (*auto simp*: *bot-ereal-def Sup-ereal-def*)
apply (*rule some1-equality*)
apply (*metis ereal-bot ereal-less-eq*(*2*))
apply (*metis ereal-less-eq*(*2*))
done
show *Inf* {} = (*top*::*ereal*)
apply (*auto simp*: *top-ereal-def Inf-ereal-def*)
apply (*rule some1-equality*)
apply (*metis ereal-top ereal-less-eq*(*1*))
apply (*metis ereal-less-eq*(*1*))
done
qed (*auto intro*: *someI2-ex ereal-complete-Sup ereal-complete-Inf*
simp: *Sup-ereal-def Inf-ereal-def bot-ereal-def top-ereal-def*)

end

instance *ereal* :: *complete-linorder* **..**

instance *ereal* :: *linear-continuum*
proof
show ∃ *a b*::*ereal*. *a* ≠ *b*
using *zero-neq-one* **by** *blast*
qed

41.2.1 Topological space

instantiation *ereal* :: *linear-continuum-topology*
begin

definition *open-ereal* :: *ereal set* ⇒ *bool* **where**
open-ereal-generated: *open-ereal* = *generate-topology* (*range lessThan* ∪ *range greaterThan*)

instance
by *standard* (*simp add*: *open-ereal-generated*)

end

lemma *continuous-on-ereal*[*continuous-intros*]:
assumes *f*: *continuous-on s f* **shows** *continuous-on s* (λ*x*. *ereal* (*f x*))
by (*rule continuous-on-compose2* [*OF continuous-onI-mono*[*of ereal UNIV*] *f*])
auto

lemma *tendsto-ereal*[*tendsto-intros*, *simp*, *intro*]: (*f* ⟶ *x*) *F* ⟹ ((λ*x*. *ereal* (*f*

$x)) \longrightarrow ereal\ x)\ F$
using *isCont-tendsto-compose*[*of x ereal f F*] *continuous-on-ereal*[*of UNIV* $\lambda x.$ *x*]
by (*simp add*: *continuous-on-eq-continuous-at*)

lemma *tendsto-uminus-ereal*[*tendsto-intros*, *simp*, *intro*]: $(f \longrightarrow x)\ F \Longrightarrow ((\lambda x.$ $- f\ x::ereal) \longrightarrow - x)\ F$
apply (*rule tendsto-compose*[**where** *g*=*uminus*])
apply (*auto intro*!: *order-tendstoI simp*: *eventually-at-topological*)
apply (*rule-tac x*=$\{..< -a\}$ **in** *exI*)
apply (*auto split*: *ereal.split simp*: *ereal-less-uminus-reorder*) []
apply (*rule-tac x*=$\{- a <..\}$ **in** *exI*)
apply (*auto split*: *ereal.split simp*: *ereal-uminus-reorder*) []
done

lemma *at-infty-ereal-eq-at-top*: *at* ∞ = *filtermap ereal at-top*
unfolding *filter-eq-iff eventually-at-filter eventually-at-top-linorder eventually-filtermap top-ereal-def*[*symmetric*]
apply (*subst eventually-nhds-top*[*of 0*])
apply (*auto simp*: *top-ereal-def less-le ereal-all-split ereal-ex-split*)
apply (*metis PInfty-neq-ereal*(*2*) *ereal-less-eq*(*3*) *ereal-top le-cases order-trans*)
done

lemma *ereal-Lim-uminus*: $(f \longrightarrow f0)\ net \longleftrightarrow ((\lambda x. - f\ x::ereal) \longrightarrow - f0)$ *net*
using *tendsto-uminus-ereal*[*of f f0 net*] *tendsto-uminus-ereal*[*of* $\lambda x. - f\ x$ *- f0 net*]
by *auto*

lemma *ereal-divide-less-iff*: $0 < (c::ereal) \Longrightarrow c < \infty \Longrightarrow a\ /\ c < b \longleftrightarrow a < b * c$
by (*cases a b c rule*: *ereal3-cases*) (*auto simp*: *field-simps*)

lemma *ereal-less-divide-iff*: $0 < (c::ereal) \Longrightarrow c < \infty \Longrightarrow a < b\ /\ c \longleftrightarrow a * c < b$
by (*cases a b c rule*: *ereal3-cases*) (*auto simp*: *field-simps*)

lemma *tendsto-cmult-ereal*[*tendsto-intros*, *simp*, *intro*]:
assumes *c*: $|c| \neq \infty$ **and** *f*: $(f \longrightarrow x)\ F$ **shows** $((\lambda x.\ c * f\ x::ereal) \longrightarrow c * x)\ F$
proof −
{ fix *c* :: *ereal* **assume** $0 < c\ c < \infty$
then have $((\lambda x.\ c * f\ x::ereal) \longrightarrow c * x)\ F$
apply (*intro tendsto-compose*[*OF - f*])
apply (*auto intro*!: *order-tendstoI simp*: *eventually-at-topological*)
apply (*rule-tac x*=$\{a/c <..\}$ **in** *exI*)
apply (*auto split*: *ereal.split simp*: *ereal-divide-less-iff mult.commute*) []
apply (*rule-tac x*=$\{..< a/c\}$ **in** *exI*)
apply (*auto split*: *ereal.split simp*: *ereal-less-divide-iff mult.commute*) []

```
    done }
  note ∗ = this

  have ((0 < c ∧ c < ∞) ∨ (−∞ < c ∧ c < 0) ∨ c = 0)
    using c by (cases c) auto
  then show ?thesis
  proof (elim disjE conjE)
    assume − ∞ < c c < 0
    then have 0 < − c − c < ∞
      by (auto simp: ereal-uminus-reorder ereal-less-uminus-reorder[of 0])
    then have ((λx. (− c) ∗ f x) ⟶ (− c) ∗ x) F
      by (rule ∗)
    from tendsto-uminus-ereal[OF this] show ?thesis
      by simp
  qed (auto intro!: ∗)
qed

lemma tendsto-cmult-ereal-not-0[tendsto-intros, simp, intro]:
  assumes x ≠ 0 and f: (f ⟶ x) F shows ((λx. c ∗ f x::ereal) ⟶ c ∗ x)
F
proof cases
  assume |c| = ∞
  show ?thesis
  proof (rule filterlim-cong[THEN iffD1, OF refl refl - tendsto-const])
    have 0 < x ∨ x < 0
      using ‹x ≠ 0› by (auto simp add: neq-iff)
    then show eventually (λx'. c ∗ x = c ∗ f x') F
    proof
      assume 0 < x from order-tendstoD(1)[OF f this] show ?thesis
        by eventually-elim (insert ‹0<x› ‹|c| = ∞›, auto)
    next
      assume x < 0 from order-tendstoD(2)[OF f this] show ?thesis
        by eventually-elim (insert ‹x<0› ‹|c| = ∞›, auto)
    qed
  qed
qed (rule tendsto-cmult-ereal[OF - f])

lemma tendsto-cadd-ereal[tendsto-intros, simp, intro]:
  assumes c: y ≠ − ∞ x ≠ − ∞ and f: (f ⟶ x) F shows ((λx. f x + y::ereal)
⟶ x + y) F
  apply (intro tendsto-compose[OF - f])
  apply (auto intro!: order-tendstoI simp: eventually-at-topological)
  apply (rule-tac x={a − y <..} in exI)
  apply (auto split: ereal.split simp: ereal-minus-less-iff c) []
  apply (rule-tac x={..< a − y} in exI)
  apply (auto split: ereal.split simp: ereal-less-minus-iff c) []
  done

lemma tendsto-add-left-ereal[tendsto-intros, simp, intro]:
```

assumes *c*: $|y| \neq \infty$ **and** *f*: $(f \longrightarrow x)$ *F* **shows** $((\lambda x.\ f\ x + y::ereal) \longrightarrow x + y)$ *F*
apply (*intro tendsto-compose*[*OF* - *f*])
apply (*auto intro*!: *order-tendstoI simp*: *eventually-at-topological*)
apply (*rule-tac x*=$\{a - y <..\}$ **in** *exI*)
apply (*insert c*, *auto split*: *ereal.split simp*: *ereal-minus-less-iff*) []
apply (*rule-tac x*=$\{..< a - y\}$ **in** *exI*)
apply (*auto split*: *ereal.split simp*: *ereal-less-minus-iff c*) []
done

lemma *continuous-at-ereal*[*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ ereal\ (f\ x))$
unfolding *continuous-def* **by** *auto*

lemma *ereal-Sup*:
assumes $*$: $|SUP\ a{:}A.\ ereal\ a| \neq \infty$
shows *ereal* $(Sup\ A) = (SUP\ a{:}A.\ ereal\ a)$
proof (*rule continuous-at-Sup-mono*)
obtain *r* **where** *r*: *ereal r* $= (SUP\ a{:}A.\ ereal\ a)$ $A \neq \{\}$
using $*$ **by** (*force simp*: *bot-ereal-def*)
then show *bdd-above A A* $\neq \{\}$
by (*auto intro*!: *SUP-upper bdd-aboveI*[*of* - *r*] *simp add*: *ereal-less-eq*(*3*)[*symmetric*] *simp del*: *ereal-less-eq*)
qed (*auto simp*: *mono-def continuous-at-imp-continuous-at-within continuous-at-ereal*)

lemma *ereal-SUP*: $|SUP\ a{:}A.\ ereal\ (f\ a)| \neq \infty \Longrightarrow ereal\ (SUP\ a{:}A.\ f\ a) = (SUP\ a{:}A.\ ereal\ (f\ a))$
using *ereal-Sup*[*of f*‘*A*] **by** *auto*

lemma *ereal-Inf*:
assumes $*$: $|INF\ a{:}A.\ ereal\ a| \neq \infty$
shows *ereal* $(Inf\ A) = (INF\ a{:}A.\ ereal\ a)$
proof (*rule continuous-at-Inf-mono*)
obtain *r* **where** *r*: *ereal r* $= (INF\ a{:}A.\ ereal\ a)$ $A \neq \{\}$
using $*$ **by** (*force simp*: *top-ereal-def*)
then show *bdd-below A A* $\neq \{\}$
by (*auto intro*!: *INF-lower bdd-belowI*[*of* - *r*] *simp add*: *ereal-less-eq*(*3*)[*symmetric*] *simp del*: *ereal-less-eq*)
qed (*auto simp*: *mono-def continuous-at-imp-continuous-at-within continuous-at-ereal*)

lemma *ereal-Inf'*:
assumes $*$: *bdd-below A A* $\neq \{\}$
shows *ereal* $(Inf\ A) = (INF\ a{:}A.\ ereal\ a)$
proof (*rule ereal-Inf*)
from $*$ **obtain** *l u* **where** $\bigwedge x.\ x \in A \Longrightarrow l \leq x\ u \in A$
by (*auto simp*: *bdd-below-def*)
then have $l \leq (INF\ x{:}A.\ ereal\ x)$ $(INF\ x{:}A.\ ereal\ x) \leq u$
by (*auto intro*!: *INF-greatest INF-lower*)
then show $|INF\ a{:}A.\ ereal\ a| \neq \infty$

by *auto*
qed

lemma *ereal-INF*: $|INF\ a{:}A.\ ereal\ (f\ a)| \neq \infty \Longrightarrow ereal\ (INF\ a{:}A.\ f\ a) = (INF\ a{:}A.\ ereal\ (f\ a))$
using *ereal-Inf*[*of f*‘*A*] **by** *auto*

lemma *ereal-Sup-uminus-image-eq*: $Sup\ (uminus\ `\ S{::}ereal\ set) = -\ Inf\ S$
by (*auto intro*!: *SUP-eqI*
simp: *Ball-def*[*symmetric*] *ereal-uminus-le-reorder le-Inf-iff*
intro!: *complete-lattice-class.Inf-lower2*)

lemma *ereal-SUP-uminus-eq*:
fixes $f :: 'a \Rightarrow ereal$
shows $(SUP\ x{:}S.\ uminus\ (f\ x)) = -\ (INF\ x{:}S.\ f\ x)$
using *ereal-Sup-uminus-image-eq* [*of f* ‘ *S*] **by** (*simp add*: *comp-def*)

lemma *ereal-inj-on-uminus*[*intro*, *simp*]: *inj-on uminus* ($A :: ereal\ set$)
by (*auto intro*!: *inj-onI*)

lemma *ereal-Inf-uminus-image-eq*: $Inf\ (uminus\ `\ S{::}ereal\ set) = -\ Sup\ S$
using *ereal-Sup-uminus-image-eq*[*of uminus* ‘ *S*] **by** *simp*

lemma *ereal-INF-uminus-eq*:
fixes $f :: 'a \Rightarrow ereal$
shows $(INF\ x{:}S.\ -\ f\ x) = -\ (SUP\ x{:}S.\ f\ x)$
using *ereal-Inf-uminus-image-eq* [*of f* ‘ *S*] **by** (*simp add*: *comp-def*)

lemma *ereal-SUP-uminus*:
fixes $f :: 'a \Rightarrow ereal$
shows $(SUP\ i : R.\ -\ f\ i) = -\ (INF\ i : R.\ f\ i)$
using *ereal-Sup-uminus-image-eq*[*of f*‘*R*]
by (*simp add*: *image-image*)

lemma *ereal-SUP-not-infty*:
fixes $f :: - \Rightarrow ereal$
shows $A \neq \{\} \Longrightarrow l \neq -\infty \Longrightarrow u \neq \infty \Longrightarrow \forall a{\in}A.\ l \leq f\ a \wedge f\ a \leq u \Longrightarrow |SUPREMUM\ A\ f| \neq \infty$
using *SUP-upper2*[*of* - *A l f*] *SUP-least*[*of A f u*]
by (*cases SUPREMUM A f*) *auto*

lemma *ereal-INF-not-infty*:
fixes $f :: - \Rightarrow ereal$
shows $A \neq \{\} \Longrightarrow l \neq -\infty \Longrightarrow u \neq \infty \Longrightarrow \forall a{\in}A.\ l \leq f\ a \wedge f\ a \leq u \Longrightarrow |INFIMUM\ A\ f| \neq \infty$
using *INF-lower2*[*of* - *A f u*] *INF-greatest*[*of A l f*]
by (*cases INFIMUM A f*) *auto*

lemma *ereal-image-uminus-shift*:

fixes X Y :: *ereal set*
shows *uminus* ' $X = Y \longleftrightarrow X =$ *uminus* ' Y
proof
assume *uminus* ' $X = Y$
then have *uminus* ' *uminus* ' $X =$ *uminus* ' Y
by (*simp add*: *inj-image-eq-iff*)
then show $X =$ *uminus* ' Y
by (*simp add*: *image-image*)
qed (*simp add*: *image-image*)

lemma *Sup-eq-MInfty*:
fixes S :: *ereal set*
shows *Sup* $S = -\infty \longleftrightarrow S = \{\} \vee S = \{-\infty\}$
unfolding *bot-ereal-def*[*symmetric*] **by** *auto*

lemma *Inf-eq-PInfty*:
fixes S :: *ereal set*
shows *Inf* $S = \infty \longleftrightarrow S = \{\} \vee S = \{\infty\}$
using *Sup-eq-MInfty*[*of uminus*'S]
unfolding *ereal-Sup-uminus-image-eq ereal-image-uminus-shift* **by** *simp*

lemma *Inf-eq-MInfty*:
fixes S :: *ereal set*
shows $-\infty \in S \Longrightarrow$ *Inf* $S = -\infty$
unfolding *bot-ereal-def*[*symmetric*] **by** *auto*

lemma *Sup-eq-PInfty*:
fixes S :: *ereal set*
shows $\infty \in S \Longrightarrow$ *Sup* $S = \infty$
unfolding *top-ereal-def*[*symmetric*] **by** *auto*

lemma *not-MInfty-nonneg*[*simp*]: $0 \leq (x::ereal) \Longrightarrow x \neq -\infty$
by *auto*

lemma *Sup-ereal-close*:
fixes e :: *ereal*
assumes $0 < e$
and S: $|Sup\ S| \neq \infty$ $S \neq \{\}$
shows $\exists x{\in}S.$ *Sup* $S - e < x$
using *assms* **by** (*cases e*) (*auto intro*!: *less-Sup-iff*[*THEN iffD1*])

lemma *Inf-ereal-close*:
fixes e :: *ereal*
assumes $|Inf\ X| \neq \infty$
and $0 < e$
shows $\exists x{\in}X.\ x <$ *Inf* $X + e$
proof (*rule Inf-less-iff*[*THEN iffD1*])
show *Inf* $X <$ *Inf* $X + e$
using *assms* **by** (*cases e*) *auto*

qed

lemma *SUP-PInfty*:
(⋀*n*::*nat*. ∃ *i*∈*A*. *ereal* (*real n*) ≤ *f i*) ⟹ (*SUP i*:*A*. *f i* :: *ereal*) = ∞
unfolding *top-ereal-def*[*symmetric*] *SUP-eq-top-iff*
by (*metis MInfty-neq-PInfty*(*2*) *PInfty-neq-ereal*(*2*) *less-PInf-Ex-of-nat less-ereal.elims*(*2*) *less-le-trans*)

lemma *SUP-nat-Infty*: (*SUP i*::*nat*. *ereal* (*real i*)) = ∞
by (*rule SUP-PInfty*) *auto*

lemma *SUP-ereal-add-left*:
assumes *I* ≠ {} *c* ≠ −∞
shows (*SUP i*:*I*. *f i* + *c* :: *ereal*) = (*SUP i*:*I*. *f i*) + *c*
proof *cases*
assume (*SUP i*:*I*. *f i*) = − ∞
moreover then have ⋀*i*. *i* ∈ *I* ⟹ *f i* = −∞
unfolding *Sup-eq-MInfty Sup-image-eq*[*symmetric*] **by** *auto*
ultimately show *?thesis*
by (*cases c*) (*auto simp*: ‹*I* ≠ {}›)
next
assume (*SUP i*:*I*. *f i*) ≠ − ∞ **then show** *?thesis*
unfolding *Sup-image-eq*[*symmetric*]
by (*subst continuous-at-Sup-mono*[**where** *f*=λ*x*. *x* + *c*])
(*auto simp*: *continuous-at-imp-continuous-at-within continuous-at mono-def ereal-add-mono* ‹*I* ≠ {}› ‹*c* ≠ −∞›)
qed

lemma *SUP-ereal-add-right*:
fixes *c* :: *ereal*
shows *I* ≠ {} ⟹ *c* ≠ −∞ ⟹ (*SUP i*:*I*. *c* + *f i*) = *c* + (*SUP i*:*I*. *f i*)
using *SUP-ereal-add-left*[*of I c f*] **by** (*simp add*: *add.commute*)

lemma *SUP-ereal-minus-right*:
assumes *I* ≠ {} *c* ≠ −∞
shows (*SUP i*:*I*. *c* − *f i* :: *ereal*) = *c* − (*INF i*:*I*. *f i*)
using *SUP-ereal-add-right*[*OF assms*, *of* λ*i*. − *f i*]
by (*simp add*: *ereal-SUP-uminus minus-ereal-def*)

lemma *SUP-ereal-minus-left*:
assumes *I* ≠ {} *c* ≠ ∞
shows (*SUP i*:*I*. *f i* − *c*:: *ereal*) = (*SUP i*:*I*. *f i*) − *c*
using *SUP-ereal-add-left*[*OF* ‹*I* ≠ {}›, *of* −*c f*] **by** (*simp add*: ‹*c* ≠ ∞› *minus-ereal-def*)

lemma *INF-ereal-minus-right*:
assumes *I* ≠ {} **and** |*c*| ≠ ∞
shows (*INF i*:*I*. *c* − *f i*) = *c* − (*SUP i*:*I*. *f i*::*ereal*)
proof −
{ fix *b* **have** (−*c*) + *b* = − (*c* − *b*)

```
    using ‹|c| ≠ ∞› by (cases c b rule: ereal2-cases) auto }
  note ∗ = this
  show ?thesis
    using SUP-ereal-add-right[OF ‹I ≠ {}›, of −c f] ‹|c| ≠ ∞›
    by (auto simp add: ∗ ereal-SUP-uminus-eq)
qed

lemma SUP-ereal-le-addI:
  fixes f :: 'i ⇒ ereal
  assumes ⋀i. f i + y ≤ z and y ≠ −∞
  shows SUPREMUM UNIV f + y ≤ z
  unfolding SUP-ereal-add-left[OF UNIV-not-empty ‹y ≠ −∞›, symmetric]
  by (rule SUP-least assms)+

lemma SUP-combine:
  fixes f :: 'a::semilattice-sup ⇒ 'a::semilattice-sup ⇒ 'b::complete-lattice
  assumes mono: ⋀a b c d. a ≤ b ⟹ c ≤ d ⟹ f a c ≤ f b d
  shows (SUP i:UNIV. SUP j:UNIV. f i j) = (SUP i. f i i)
proof (rule antisym)
  show (SUP i j. f i j) ≤ (SUP i. f i i)
    by (rule SUP-least SUP-upper2[where i=sup i j for i j] UNIV-I mono sup-ge1
sup-ge2)+
  show (SUP i. f i i) ≤ (SUP i j. f i j)
    by (rule SUP-least SUP-upper2 UNIV-I mono order-refl)+
qed

lemma SUP-ereal-add:
  fixes f g :: nat ⇒ ereal
  assumes inc: incseq f incseq g
    and pos: ⋀i. f i ≠ −∞ ⋀i. g i ≠ −∞
  shows (SUP i. f i + g i) = SUPREMUM UNIV f + SUPREMUM UNIV g
  apply (subst SUP-ereal-add-left[symmetric, OF UNIV-not-empty])
  apply (metis SUP-upper UNIV-I assms(4) ereal-infty-less-eq(2))
  apply (subst (2) add.commute)
  apply (subst SUP-ereal-add-left[symmetric, OF UNIV-not-empty assms(3)])
  apply (subst (2) add.commute)
  apply (rule SUP-combine[symmetric] ereal-add-mono inc[THEN monoD] | as-
sumption)+
  done

lemma INF-ereal-add:
  fixes f :: nat ⇒ ereal
  assumes decseq f decseq g
    and fin: ⋀i. f i ≠ ∞ ⋀i. g i ≠ ∞
  shows (INF i. f i + g i) = INFIMUM UNIV f + INFIMUM UNIV g
proof −
  have INF-less: (INF i. f i) < ∞ (INF i. g i) < ∞
    using assms unfolding INF-less-iff by auto
  { fix a b :: ereal assume a ≠ ∞ b ≠ ∞
```

```
    then have − ((− a) + (− b)) = a + b
      by (cases a b rule: ereal2-cases) auto }
  note ∗ = this
  have (INF i. f i + g i) = (INF i. − ((− f i) + (− g i)))
    by (simp add: fin ∗)
  also have ... = INFIMUM UNIV f + INFIMUM UNIV g
    unfolding ereal-INF-uminus-eq
    using assms INF-less
    by (subst SUP-ereal-add) (auto simp: ereal-SUP-uminus fin ∗)
  finally show ?thesis .
qed

lemma SUP-ereal-add-pos:
  fixes f g :: nat ⇒ ereal
  assumes inc: incseq f incseq g
    and pos: ⋀i. 0 ≤ f i ⋀i. 0 ≤ g i
  shows (SUP i. f i + g i) = SUPREMUM UNIV f + SUPREMUM UNIV g
proof (intro SUP-ereal-add inc)
  fix i
  show f i ≠ −∞ g i ≠ −∞
    using pos[of i] by auto
qed

lemma SUP-ereal-setsum:
  fixes f g :: 'a ⇒ nat ⇒ ereal
  assumes ⋀n. n ∈ A ⟹ incseq (f n)
    and pos: ⋀n i. n ∈ A ⟹ 0 ≤ f n i
  shows (SUP i. ∑ n∈A. f n i) = (∑ n∈A. SUPREMUM UNIV (f n))
proof (cases finite A)
  case True
  then show ?thesis using assms
    by induct (auto simp: incseq-setsumI2 setsum-nonneg SUP-ereal-add-pos)
next
  case False
  then show ?thesis by simp
qed

lemma SUP-ereal-mult-left:
  fixes f :: 'a ⇒ ereal
  assumes I ≠ {}
  assumes f: ⋀i. i ∈ I ⟹ 0 ≤ f i and c: 0 ≤ c
  shows (SUP i:I. c ∗ f i) = c ∗ (SUP i:I. f i)
proof cases
  assume (SUP i: I. f i) = 0
  moreover then have ⋀i. i ∈ I ⟹ f i = 0
    by (metis SUP-upper f antisym)
  ultimately show ?thesis
    by simp
next
```

```
  assume (SUP i:I. f i) ≠ 0 then show ?thesis
    unfolding SUP-def
    by (subst continuous-at-Sup-mono[where f=λx. c * x])
       (auto simp: mono-def continuous-at continuous-at-imp-continuous-at-within
‹I ≠ {}›
             intro!: ereal-mult-left-mono c)
qed

lemma countable-approach:
  fixes x :: ereal
  assumes x ≠ −∞
  shows ∃f. incseq f ∧ (∀ i::nat. f i < x) ∧ (f ⟶ x)
proof (cases x)
  case (real r)
  moreover have (λn. r − inverse (real (Suc n))) ⟶ r − 0
    by (intro tendsto-intros LIMSEQ-inverse-real-of-nat)
  ultimately show ?thesis
    by (intro exI[of - λn. x − inverse (Suc n)]) (auto simp: incseq-def)
next
  case PInf with LIMSEQ-SUP[of λn::nat. ereal (real n)] show ?thesis
    by (intro exI[of - λn. ereal (real n)]) (auto simp: incseq-def SUP-nat-Infty)
qed (simp add: assms)

lemma Sup-countable-SUP:
  assumes A ≠ {}
  shows ∃f::nat ⇒ ereal. incseq f ∧ range f ⊆ A ∧ Sup A = (SUP i. f i)
proof cases
  assume Sup A = −∞
  with ‹A ≠ {}› have A = {−∞}
    by (auto simp: Sup-eq-MInfty)
  then show ?thesis
    by (auto intro!: exI[of - λ-. −∞] simp: bot-ereal-def)
next
  assume Sup A ≠ −∞
  then obtain l where incseq l and l: ⋀i::nat. l i < Sup A and l-Sup: l ⟶
Sup A
    by (auto dest: countable-approach)

  have ∃f. ∀ n. (f n ∈ A ∧ l n ≤ f n) ∧ (f n ≤ f (Suc n))
  proof (rule dependent-nat-choice)
    show ∃ x. x ∈ A ∧ l 0 ≤ x
      using l[of 0] by (auto simp: less-Sup-iff)
  next
    fix x n assume x ∈ A ∧ l n ≤ x
    moreover from l[of Suc n] obtain y where y ∈ A l (Suc n) < y
      by (auto simp: less-Sup-iff)
    ultimately show ∃ y. (y ∈ A ∧ l (Suc n) ≤ y) ∧ x ≤ y
      by (auto intro!: exI[of - max x y] split: split-max)
  qed
```

then guess f **.. note** $f = this$
then have $range\ f \subseteq A\ incseq\ f$
by (*auto simp*: *incseq-Suc-iff*)
moreover
have $(SUP\ i.\ f\ i) = Sup\ A$
proof (*rule tendsto-unique*)
show $f \longrightarrow (SUP\ i.\ f\ i)$
by (*rule LIMSEQ-SUP* ‹*incseq f*›)+
show $f \longrightarrow Sup\ A$
using *l f*
by (*intro tendsto-sandwich*[*OF - - l-Sup tendsto-const*])
(*auto simp*: *Sup-upper*)
qed *simp*
ultimately show *?thesis*
by *auto*
qed

lemma *SUP-countable-SUP*:
$A \neq \{\} \Longrightarrow \exists f{::}nat \Rightarrow ereal.\ range\ f \subseteq g`A \wedge SUPREMUM\ A\ g = SUPREMUM\ UNIV\ f$
using *Sup-countable-SUP* [*of g`A*] **by** *auto*

41.3 Relation to *enat*

definition $ereal\text{-}of\text{-}enat\ n = (case\ n\ of\ enat\ n \Rightarrow ereal\ (real\ n) \mid \infty \Rightarrow \infty)$

declare [[*coercion ereal-of-enat* :: *enat* ⇒ *ereal*]]
declare [[*coercion* (λ*n*. *ereal* (*real n*)) :: *nat* ⇒ *ereal*]]

lemma *ereal-of-enat-simps*[*simp*]:
$ereal\text{-}of\text{-}enat\ (enat\ n) = ereal\ n$
$ereal\text{-}of\text{-}enat\ \infty = \infty$
by (*simp-all add*: *ereal-of-enat-def*)

lemma *ereal-of-enat-le-iff*[*simp*]: $ereal\text{-}of\text{-}enat\ m \leq ereal\text{-}of\text{-}enat\ n \longleftrightarrow m \leq n$
by (*cases m n rule*: *enat2-cases*) *auto*

lemma *ereal-of-enat-less-iff*[*simp*]: $ereal\text{-}of\text{-}enat\ m < ereal\text{-}of\text{-}enat\ n \longleftrightarrow m < n$
by (*cases m n rule*: *enat2-cases*) *auto*

lemma *numeral-le-ereal-of-enat-iff*[*simp*]: $numeral\ m \leq ereal\text{-}of\text{-}enat\ n \longleftrightarrow numeral\ m \leq n$
by (*cases n*) (*auto*)

lemma *numeral-less-ereal-of-enat-iff*[*simp*]: $numeral\ m < ereal\text{-}of\text{-}enat\ n \longleftrightarrow numeral\ m < n$
by (*cases n*) *auto*

lemma *ereal-of-enat-ge-zero-cancel-iff*[*simp*]: $0 \leq ereal\text{-}of\text{-}enat\ n \longleftrightarrow 0 \leq n$

by (*cases n*) (*auto simp*: *enat-0*[*symmetric*])

lemma *ereal-of-enat-gt-zero-cancel-iff*[*simp*]: $0 < \textit{ereal-of-enat}\ n \longleftrightarrow 0 < n$
by (*cases n*) (*auto simp*: *enat-0*[*symmetric*])

lemma *ereal-of-enat-zero*[*simp*]: *ereal-of-enat 0 = 0*
by (*auto simp*: *enat-0*[*symmetric*])

lemma *ereal-of-enat-inf*[*simp*]: $\textit{ereal-of-enat}\ n = \infty \longleftrightarrow n = \infty$
by (*cases n*) *auto*

lemma *ereal-of-enat-add*: $\textit{ereal-of-enat}\ (m + n) = \textit{ereal-of-enat}\ m + \textit{ereal-of-enat}\ n$
by (*cases m n rule*: *enat2-cases*) *auto*

lemma *ereal-of-enat-sub*:
assumes $n \leq m$
shows $\textit{ereal-of-enat}\ (m - n) = \textit{ereal-of-enat}\ m - \textit{ereal-of-enat}\ n$
using *assms* **by** (*cases m n rule*: *enat2-cases*) *auto*

lemma *ereal-of-enat-mult*:
$\textit{ereal-of-enat}\ (m * n) = \textit{ereal-of-enat}\ m * \textit{ereal-of-enat}\ n$
by (*cases m n rule*: *enat2-cases*) *auto*

lemmas *ereal-of-enat-pushin* = *ereal-of-enat-add ereal-of-enat-sub ereal-of-enat-mult*
lemmas *ereal-of-enat-pushout* = *ereal-of-enat-pushin*[*symmetric*]

lemma *ereal-of-enat-nonneg*: $\textit{ereal-of-enat}\ n \geq 0$
by(*cases n*) *simp-all*

lemma *ereal-of-enat-Sup*:
assumes $A \neq \{\}$ **shows** $\textit{ereal-of-enat}\ (\textit{Sup}\ A) = (\textit{SUP}\ a : A.\ \textit{ereal-of-enat}\ a)$
proof (*intro antisym mono-Sup*)
show $\textit{ereal-of-enat}\ (\textit{Sup}\ A) \leq (\textit{SUP}\ a : A.\ \textit{ereal-of-enat}\ a)$
proof *cases*
assume *finite A*
with ‹$A \neq \{\}$› **obtain** *a* **where** $a \in A\ \textit{ereal-of-enat}\ (\textit{Sup}\ A) = \textit{ereal-of-enat}\ a$
using *Max-in*[*of A*] **by** (*auto simp*: *Sup-enat-def simp del*: *Max-in*)
then show *?thesis*
by (*auto intro*: *SUP-upper*)
next
assume ¬ *finite A*
have [*simp*]: $(\textit{SUP}\ a : A.\ \textit{ereal-of-enat}\ a) = \textit{top}$
unfolding *SUP-eq-top-iff*
proof *safe*
fix *x* :: *ereal* **assume** $x < \textit{top}$
then obtain *n* :: *nat* **where** $x < n$
using *less-PInf-Ex-of-nat top-ereal-def* **by** *auto*
obtain *a* **where** $a \in A - \textit{enat}\ `\ \{..\ n\}$

by (*metis* ‹¬ *finite A*› *all-not-in-conv finite-Diff2 finite-atMost finite-imageI finite.emptyI*)
then have $a \in A$ *ereal* $n \leq$ *ereal-of-enat a*
by (*auto simp*: *image-iff Ball-def*)
(*metis enat-iless enat-ord-simps*(*1*) *ereal-of-enat-less-iff ereal-of-enat-simps*(*1*) *less-le not-less*)
with ‹$x < n$›
show $\exists i{\in}A.\ x <$ *ereal-of-enat i*
by (*auto intro*!: *bexI*[*of* - *a*])
qed
show *?thesis*
by *simp*
qed
qed (*simp add*: *mono-def*)

lemma *ereal-of-enat-SUP*:
$A \neq \{\} \Longrightarrow$ *ereal-of-enat* (*SUP a*:*A*. *f a*) = (*SUP a* : *A*. *ereal-of-enat* (*f a*))
using *ereal-of-enat-Sup*[*of f*‘*A*] **by** *auto*

41.4 Limits on *ereal*

lemma *open-PInfty*: *open* $A \Longrightarrow \infty \in A \Longrightarrow (\exists x.\ \{$*ereal* x<..$\} \subseteq A)$
unfolding *open-ereal-generated*
proof (*induct rule*: *generate-topology.induct*)
case (*Int A B*)
then obtain *x z* **where** $\infty \in A \Longrightarrow \{$*ereal* x <..$\} \subseteq A$ $\infty \in B \Longrightarrow \{$*ereal* z <..$\} \subseteq B$
by *auto*
with *Int* **show** *?case*
by (*intro exI*[*of* - *max x z*]) *fastforce*
next
case (*Basis S*)
{
fix *x*
have $x \neq \infty \Longrightarrow \exists t.\ x \leq$ *ereal t*
by (*cases x*) *auto*
}
moreover note *Basis*
ultimately show *?case*
by (*auto split*: *ereal.split*)
qed (*fastforce simp add*: *vimage-Union*)+

lemma *open-MInfty*: *open* $A \Longrightarrow -\infty \in A \Longrightarrow (\exists x.\ \{..<$*ereal* $x\} \subseteq A)$
unfolding *open-ereal-generated*
proof (*induct rule*: *generate-topology.induct*)
case (*Int A B*)
then obtain *x z* **where** $-\infty \in A \Longrightarrow \{..<$ *ereal* $x\} \subseteq A$ $-\infty \in B \Longrightarrow \{..<$ *ereal* $z\} \subseteq B$
by *auto*
with *Int* **show** *?case*

```
    by (intro exI[of - min x z]) fastforce
next
  case (Basis S)
  {
    fix x
    have x ≠ - ∞ ⟹ ∃ t. ereal t ≤ x
      by (cases x) auto
  }
  moreover note Basis
  ultimately show ?case
    by (auto split: ereal.split)
qed (fastforce simp add: vimage-Union)+

lemma open-ereal-vimage: open S ⟹ open (ereal −` S)
  by (intro open-vimage continuous-intros)

lemma open-ereal: open S ⟹ open (ereal ` S)
  unfolding open-generated-order[where 'a=real]
proof (induct rule: generate-topology.induct)
  case (Basis S)
  moreover {
    fix x
    have ereal ` {..< x} = { −∞ <..< ereal x }
      apply auto
      apply (case-tac xa)
      apply auto
      done
  }
  moreover {
    fix x
    have ereal ` {x <..} = { ereal x <..< ∞ }
      apply auto
      apply (case-tac xa)
      apply auto
      done
  }
  ultimately show ?case
    by auto
qed (auto simp add: image-Union image-Int)


lemma eventually-finite:
  fixes x :: ereal
  assumes |x| ≠ ∞ (f ⟶ x) F
  shows eventually (λx. |f x| ≠ ∞) F
proof −
  have (f ⟶ ereal (real-of-ereal x)) F
    using assms by (cases x) auto
  then have eventually (λx. f x ∈ ereal ` UNIV) F
```

by (*rule topological-tendstoD*) (*auto intro*: *open-ereal*)
also have $(\lambda x.\ f\ x \in ereal\ `\ UNIV) = (\lambda x.\ |f\ x| \neq \infty)$
by *auto*
finally show *?thesis* **.**
qed

lemma *open-ereal-def*:
$open\ A \longleftrightarrow open\ (ereal\ -`\ A) \wedge (\infty \in A \longrightarrow (\exists x.\ \{ereal\ x <..\} \subseteq A)) \wedge (-\infty \in A \longrightarrow (\exists x.\ \{..<ereal\ x\} \subseteq A))$
(**is** *open A* $\longleftrightarrow$ *?rhs*)
proof
assume *open A*
then show *?rhs*
using *open-PInfty open-MInfty open-ereal-vimage* **by** *auto*
next
assume *?rhs*
then obtain *x y* **where** *A*: $open\ (ereal\ -`\ A)\ \infty \in A \Longrightarrow \{ereal\ x<..\} \subseteq A\ -\infty \in A \Longrightarrow \{..<\ ereal\ y\} \subseteq A$
by *auto*
have $*$: $A = ereal\ `\ (ereal\ -`\ A) \cup (if\ \infty \in A\ then\ \{ereal\ x<..\}\ else\ \{\}) \cup (if\ -\infty \in A\ then\ \{..<\ ereal\ y\}\ else\ \{\})$
using *A(2,3)* **by** *auto*
from *open-ereal*[*OF A(1)*] **show** *open A*
by (*subst* $*$) (*auto simp*: *open-Un*)
qed

lemma *open-PInfty2*:
assumes *open A*
and $\infty \in A$
obtains *x* **where** $\{ereal\ x<..\} \subseteq A$
using *open-PInfty*[*OF assms*] **by** *auto*

lemma *open-MInfty2*:
assumes *open A*
and $-\infty \in A$
obtains *x* **where** $\{..<ereal\ x\} \subseteq A$
using *open-MInfty*[*OF assms*] **by** *auto*

lemma *ereal-openE*:
assumes *open A*
obtains *x y* **where** $open\ (ereal\ -`\ A)$
and $\infty \in A \Longrightarrow \{ereal\ x<..\} \subseteq A$
and $-\infty \in A \Longrightarrow \{..<ereal\ y\} \subseteq A$
using *assms open-ereal-def* **by** *auto*

lemmas *open-ereal-lessThan* = *open-lessThan*[**where** *'a=ereal*]
lemmas *open-ereal-greaterThan* = *open-greaterThan*[**where** *'a=ereal*]
lemmas *ereal-open-greaterThanLessThan* = *open-greaterThanLessThan*[**where** *'a=ereal*]

lemmas *closed-ereal-atLeast* = *closed-atLeast*[**where** *'a=ereal*]
lemmas *closed-ereal-atMost* = *closed-atMost*[**where** *'a=ereal*]
lemmas *closed-ereal-atLeastAtMost* = *closed-atLeastAtMost*[**where** *'a=ereal*]
lemmas *closed-ereal-singleton* = *closed-singleton*[**where** *'a=ereal*]

lemma *ereal-open-cont-interval*:
  fixes S :: *ereal set*
  assumes *open* S
    and $x \in S$
    and $|x| \neq \infty$
  obtains e **where** $e > 0$ **and** $\{x-e <..< x+e\} \subseteq S$
proof −
  from ‹*open S*›
  have *open* (*ereal* −‘ S)
    by (*rule ereal-openE*)
  then obtain e **where** $e > 0$ **and** e: $\bigwedge y$. *dist y* (*real-of-ereal x*) < $e \Longrightarrow$ *ereal* $y \in S$
    using *assms* **unfolding** *open-dist* **by** *force*
  show *thesis*
  proof (*intro that subsetI*)
    show $0 <$ *ereal e*
      using ‹$0 < e$› **by** *auto*
    fix y
    assume $y \in \{x -$ *ereal e*<..<$x +$ *ereal e*$\}$
    with *assms* **obtain** t **where** $y =$ *ereal t dist t* (*real-of-ereal x*) < e
      by (*cases y*) (*auto simp*: *dist-real-def*)
    then show $y \in S$
      using *e*[*of t*] **by** *auto*
  qed
qed

lemma *ereal-open-cont-interval2*:
  fixes S :: *ereal set*
  assumes *open* S
    and $x \in S$
    and x: $|x| \neq \infty$
  obtains a b **where** $a < x$ **and** $x < b$ **and** $\{a <..< b\} \subseteq S$
proof −
  obtain e **where** $0 < e$ $\{x -$ *e*<..<$x +$ *e*$\} \subseteq S$
    using *assms* **by** (*rule ereal-open-cont-interval*)
  with *that*[*of x − e x + e*] *ereal-between*[*OF x*, *of e*]
  show *thesis*
    by *auto*
qed

41.4.1 Convergent sequences

lemma *lim-real-of-ereal*[*simp*]:
  assumes *lim*: (*f* ⟶ *ereal x*) *net*

shows $((\lambda x.\ \mathit{real\text{-}of\text{-}ereal}\ (f\ x)) \longrightarrow x)\ \mathit{net}$
proof (*intro topological-tendstoI*)
fix S
assume *open* S **and** $x \in S$
then have S: *open* S *ereal* $x \in$ *ereal* ' S
by (*simp-all add*: *inj-image-mem-iff*)
show *eventually* $(\lambda x.\ \mathit{real\text{-}of\text{-}ereal}\ (f\ x) \in S)\ \mathit{net}$
by (*auto intro*: *eventually-mono* [*OF lim*[*THEN topological-tendstoD*, *OF open-ereal*, *OF S*]])
qed

lemma *lim-ereal*[*simp*]: $((\lambda n.\ \mathit{ereal}\ (f\ n)) \longrightarrow \mathit{ereal}\ x)\ \mathit{net} \longleftrightarrow (f \longrightarrow x)\ \mathit{net}$
by (*auto dest*!: *lim-real-of-ereal*)

lemma *convergent-real-imp-convergent-ereal*:
assumes *convergent a*
shows *convergent* $(\lambda n.\ \mathit{ereal}\ (a\ n))$ **and** $\mathit{lim}\ (\lambda n.\ \mathit{ereal}\ (a\ n)) = \mathit{ereal}\ (\mathit{lim}\ a)$
proof −
from *assms* **obtain** L **where** L: $a \longrightarrow L$ **unfolding** *convergent-def* **..**
hence *lim*: $(\lambda n.\ \mathit{ereal}\ (a\ n)) \longrightarrow \mathit{ereal}\ L$ **using** *lim-ereal* **by** *auto*
thus *convergent* $(\lambda n.\ \mathit{ereal}\ (a\ n))$ **unfolding** *convergent-def* **..**
thus $\mathit{lim}\ (\lambda n.\ \mathit{ereal}\ (a\ n)) = \mathit{ereal}\ (\mathit{lim}\ a)$ **using** *lim L limI* **by** *metis*
qed

lemma *tendsto-PInfty*: $(f \longrightarrow \infty)\ F \longleftrightarrow (\forall r.\ \mathit{eventually}\ (\lambda x.\ \mathit{ereal}\ r < f\ x)\ F)$
proof −
{
fix l :: *ereal*
assume $\forall r.\ \mathit{eventually}\ (\lambda x.\ \mathit{ereal}\ r < f\ x)\ F$
from *this*[*THEN spec*, *of real-of-ereal l*] **have** $l \neq \infty \Longrightarrow \mathit{eventually}\ (\lambda x.\ l < f\ x)\ F$
by (*cases l*) (*auto elim*: *eventually-mono*)
}
then show *?thesis*
by (*auto simp*: *order-tendsto-iff*)
qed

lemma *tendsto-PInfty'*: $(f \longrightarrow \infty)\ F = (\forall r{>}c.\ \mathit{eventually}\ (\lambda x.\ \mathit{ereal}\ r < f\ x)\ F)$
proof (*subst tendsto-PInfty*, *intro iffI allI impI*)
assume A: $\forall r{>}c.\ \mathit{eventually}\ (\lambda x.\ \mathit{ereal}\ r < f\ x)\ F$
fix r :: *real*
from A **have** A: *eventually* $(\lambda x.\ \mathit{ereal}\ r < f\ x)\ F$ **if** $r > c$ **for** r **using** *that* **by** *blast*
show *eventually* $(\lambda x.\ \mathit{ereal}\ r < f\ x)\ F$
proof (*cases* $r > c$)
case *False*
hence B: *ereal* $r \leq$ *ereal* $(c + 1)$ **by** *simp*
have $c < c + 1$ **by** *simp*

```
    from A[OF this] show eventually (λx. ereal r < f x) F
      by eventually-elim (rule le-less-trans[OF B])
  qed (simp add: A)
qed simp

lemma tendsto-PInfty-eq-at-top:
  ((λz. ereal (f z)) ⟶ ∞) F ⟷ (LIM z F. f z :> at-top)
  unfolding tendsto-PInfty filterlim-at-top-dense by simp

lemma tendsto-MInfty: (f ⟶ −∞) F ⟷ (∀ r. eventually (λx. f x < ereal r)
F)
  unfolding tendsto-def
proof safe
  fix S :: ereal set
  assume open S −∞ ∈ S
  from open-MInfty[OF this] obtain B where {..<ereal B} ⊆ S ..
  moreover
  assume ∀ r::real. eventually (λz. f z < r) F
  then have eventually (λz. f z ∈ {..< B}) F
    by auto
  ultimately show eventually (λz. f z ∈ S) F
    by (auto elim!: eventually-mono)
next
  fix x
  assume ∀ S. open S ⟶ −∞ ∈ S ⟶ eventually (λx. f x ∈ S) F
  from this[rule-format, of {..< ereal x}] show eventually (λy. f y < ereal x) F
    by auto
qed

lemma tendsto-MInfty': (f ⟶ −∞) F = (∀ r<c. eventually (λx. ereal r > f
x) F)
proof (subst tendsto-MInfty, intro iffI allI impI)
  assume A: ∀ r<c. eventually (λx. ereal r > f x) F
  fix r :: real
  from A have A: eventually (λx. ereal r > f x) F if r < c for r using that by
blast
  show eventually (λx. ereal r > f x) F
  proof (cases r < c)
    case False
    hence B: ereal r ≥ ereal (c − 1) by simp
    have c > c − 1 by simp
    from A[OF this] show eventually (λx. ereal r > f x) F
      by eventually-elim (erule less-le-trans[OF - B])
  qed (simp add: A)
qed simp

lemma Lim-PInfty: f ⟶ ∞ ⟷ (∀ B. ∃ N. ∀ n≥N. f n ≥ ereal B)
  unfolding tendsto-PInfty eventually-sequentially
proof safe
```

fix r
assume $\forall\, r.\ \exists\, N.\ \forall\, n{\geq}N.$ *ereal* $r \leq f\ n$
then obtain N **where** $\forall\, n{\geq}N.$ *ereal* $(r + 1) \leq f\ n$
by *blast*
moreover have *ereal* $r <$ *ereal* $(r + 1)$
by *auto*
ultimately show $\exists\, N.\ \forall\, n{\geq}N.$ *ereal* $r < f\ n$
by (*blast intro*: *less-le-trans*)
qed (*blast intro*: *less-imp-le*)

lemma *Lim-MInfty*: $f \longrightarrow -\infty \longleftrightarrow (\forall\, B.\ \exists\, N.\ \forall\, n{\geq}N.$ *ereal* $B \geq f\ n)$
unfolding *tendsto-MInfty eventually-sequentially*
proof *safe*
fix r
assume $\forall\, r.\ \exists\, N.\ \forall\, n{\geq}N.\ f\ n \leq$ *ereal* r
then obtain N **where** $\forall\, n{\geq}N.\ f\ n \leq$ *ereal* $(r - 1)$
by *blast*
moreover have *ereal* $(r - 1) <$ *ereal* r
by *auto*
ultimately show $\exists\, N.\ \forall\, n{\geq}N.\ f\ n <$ *ereal* r
by (*blast intro*: *le-less-trans*)
qed (*blast intro*: *less-imp-le*)

lemma *Lim-bounded-PInfty*: $f \longrightarrow l \Longrightarrow (\bigwedge n.\ f\ n \leq$ *ereal* $B) \Longrightarrow l \neq \infty$
using *LIMSEQ-le-const2*[*of f l ereal B*] **by** *auto*

lemma *Lim-bounded-MInfty*: $f \longrightarrow l \Longrightarrow (\bigwedge n.$ *ereal* $B \leq f\ n) \Longrightarrow l \neq -\infty$
using *LIMSEQ-le-const*[*of f l ereal B*] **by** *auto*

lemma *tendsto-zero-erealI*:
assumes $\bigwedge e.\ e > 0 \Longrightarrow$ *eventually* $(\lambda x.\ |f\ x| <$ *ereal* $e)\ F$
shows $(f \longrightarrow 0)\ F$
proof (*subst filterlim-cong*[*OF refl refl*])
from *assms*[*OF zero-less-one*] **show** *eventually* $(\lambda x.\ f\ x =$ *ereal* (*real-of-ereal* (*f x*))) *F*
by *eventually-elim* (*auto simp*: *ereal-real*)
hence *eventually* $(\lambda x.$ *abs* (*real-of-ereal* $(f\ x)) < e)\ F$ **if** $e > 0$ **for** e **using** *assms*[*OF that*]
by *eventually-elim* (*simp add*: *real-less-ereal-iff that*)
hence $((\lambda x.$ *real-of-ereal* $(f\ x)) \longrightarrow 0)\ F$ **unfolding** *tendsto-iff*
by (*auto simp*: *tendsto-iff dist-real-def*)
thus $((\lambda x.$ *ereal* (*real-of-ereal* $(f\ x))) \longrightarrow 0)\ F$ **by** (*simp add*: *zero-ereal-def*)
qed

lemma *tendsto-explicit*:
$f \longrightarrow f0 \longleftrightarrow (\forall\, S.$ *open* $S \longrightarrow f0 \in S \longrightarrow (\exists\, N.\ \forall\, n{\geq}N.\ f\ n \in S))$
unfolding *tendsto-def eventually-sequentially* **by** *auto*

lemma *Lim-bounded-PInfty2*: $f \longrightarrow l \Longrightarrow \forall\, n{\geq}N.\ f\ n \leq$ *ereal* $B \Longrightarrow l \neq \infty$

using *LIMSEQ-le-const2*[*of f l ereal B*] **by** *fastforce*

lemma *Lim-bounded-ereal*: $f \longrightarrow (l :: 'a::linorder\text{-}topology) \Longrightarrow \forall n{\geq}M.\ f\ n \leq C \Longrightarrow l \leq C$
by (*intro LIMSEQ-le-const2*) *auto*

lemma *Lim-bounded2-ereal*:
assumes *lim*:$f \longrightarrow (l :: 'a::linorder\text{-}topology)$
and *ge*: $\forall n{\geq}N.\ f\ n \geq C$
shows $l \geq C$
using *ge*
by (*intro tendsto-le*[*OF trivial-limit-sequentially lim tendsto-const*])
(*auto simp*: *eventually-sequentially*)

lemma *real-of-ereal-mult*[*simp*]:
fixes *a b* :: *ereal*
shows *real-of-ereal* $(a * b) =$ *real-of-ereal a* $*$ *real-of-ereal b*
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *real-of-ereal-eq-0*:
fixes $x ::$ *ereal*
shows *real-of-ereal* $x = 0 \longleftrightarrow x = \infty \vee x = -\infty \vee x = 0$
by (*cases x*) *auto*

lemma *tendsto-ereal-realD*:
fixes $f :: 'a \Rightarrow$ *ereal*
assumes $x \neq 0$
and *tendsto*: $((\lambda x.\ ereal\ (real\text{-}of\text{-}ereal\ (f\ x))) \longrightarrow x)$ *net*
shows $(f \longrightarrow x)$ *net*
proof (*intro topological-tendstoI*)
fix *S*
assume *S*: *open S x* $\in S$
with ‹$x \neq 0$› **have** *open* $(S - \{0\})\ x \in S - \{0\}$
by *auto*
from *tendsto*[*THEN topological-tendstoD*, *OF this*]
show *eventually* $(\lambda x.\ f\ x \in S)$ *net*
by (*rule eventually-rev-mp*) (*auto simp*: *ereal-real*)
qed

lemma *tendsto-ereal-realI*:
fixes $f :: 'a \Rightarrow$ *ereal*
assumes *x*: $|x| \neq \infty$ **and** *tendsto*: $(f \longrightarrow x)$ *net*
shows $((\lambda x.\ ereal\ (real\text{-}of\text{-}ereal\ (f\ x))) \longrightarrow x)$ *net*
proof (*intro topological-tendstoI*)
fix *S*
assume *open S* **and** $x \in S$
with *x* **have** *open* $(S - \{\infty, -\infty\})\ x \in S - \{\infty, -\infty\}$
by *auto*
from *tendsto*[*THEN topological-tendstoD*, *OF this*]

show *eventually* (λ*x*. *ereal* (*real-of-ereal* (*f x*)) ∈ *S*) *net*
by (*elim eventually-mono*) (*auto simp*: *ereal-real*)
qed

lemma *ereal-mult-cancel-left*:
fixes *a b c* :: *ereal*
shows *a* ∗ *b* = *a* ∗ *c* ⟷ (|*a*| = ∞ ∧ *0* < *b* ∗ *c*) ∨ *a* = *0* ∨ *b* = *c*
by (*cases rule*: *ereal3-cases*[*of a b c*]) (*simp-all add*: *zero-less-mult-iff*)

lemma *tendsto-add-ereal*:
fixes *x y* :: *ereal*
assumes *x*: |*x*| ≠ ∞ **and** *y*: |*y*| ≠ ∞
assumes *f*: (*f* ⟶ *x*) *F* **and** *g*: (*g* ⟶ *y*) *F*
shows ((λ*x*. *f x* + *g x*) ⟶ *x* + *y*) *F*
proof −
from *x* **obtain** *r* **where** *x'*: *x* = *ereal r* **by** (*cases x*) *auto*
with *f* **have** ((λ*i*. *real-of-ereal* (*f i*)) ⟶ *r*) *F* **by** *simp*
moreover
from *y* **obtain** *p* **where** *y'*: *y* = *ereal p* **by** (*cases y*) *auto*
with *g* **have** ((λ*i*. *real-of-ereal* (*g i*)) ⟶ *p*) *F* **by** *simp*
ultimately have ((λ*i*. *real-of-ereal* (*f i*) + *real-of-ereal* (*g i*)) ⟶ *r* + *p*) *F*
by (*rule tendsto-add*)
moreover
from *eventually-finite*[*OF x f*] *eventually-finite*[*OF y g*]
have *eventually* (λ*x*. *f x* + *g x* = *ereal* (*real-of-ereal* (*f x*) + *real-of-ereal* (*g x*)))
F
by *eventually-elim auto*
ultimately show *?thesis*
by (*simp add*: *x' y' cong*: *filterlim-cong*)
qed

lemma *ereal-inj-affinity*:
fixes *m t* :: *ereal*
assumes |*m*| ≠ ∞
and *m* ≠ *0*
and |*t*| ≠ ∞
shows *inj-on* (λ*x*. *m* ∗ *x* + *t*) *A*
using *assms*
by (*cases rule*: *ereal2-cases*[*of m t*])
(*auto intro*!: *inj-onI simp*: *ereal-add-cancel-right ereal-mult-cancel-left*)

lemma *ereal-PInfty-eq-plus*[*simp*]:
fixes *a b* :: *ereal*
shows ∞ = *a* + *b* ⟷ *a* = ∞ ∨ *b* = ∞
by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-MInfty-eq-plus*[*simp*]:
fixes *a b* :: *ereal*
shows −∞ = *a* + *b* ⟷ (*a* = −∞ ∧ *b* ≠ ∞) ∨ (*b* = −∞ ∧ *a* ≠ ∞)

by (*cases rule*: *ereal2-cases*[*of a b*]) *auto*

lemma *ereal-less-divide-pos*:
fixes *x y* :: *ereal*
shows $x > 0 \Longrightarrow x \neq \infty \Longrightarrow y < z \ / \ x \longleftrightarrow x * y < z$
by (*cases rule*: *ereal3-cases*[*of x y z*]) (*auto simp*: *field-simps*)

lemma *ereal-divide-less-pos*:
fixes *x y z* :: *ereal*
shows $x > 0 \Longrightarrow x \neq \infty \Longrightarrow y \ / \ x < z \longleftrightarrow y < x * z$
by (*cases rule*: *ereal3-cases*[*of x y z*]) (*auto simp*: *field-simps*)

lemma *ereal-divide-eq*:
fixes *a b c* :: *ereal*
shows $b \neq 0 \Longrightarrow |b| \neq \infty \Longrightarrow a \ / \ b = c \longleftrightarrow a = b * c$
by (*cases rule*: *ereal3-cases*[*of a b c*])
(*simp-all add*: *field-simps*)

lemma *ereal-inverse-not-MInfty*[*simp*]: *inverse* (*a*::*ereal*) $\neq -\infty$
by (*cases a*) *auto*

lemma *ereal-mult-m1*[*simp*]: $x * ereal\ (-1) = -x$
by (*cases x*) *auto*

lemma *ereal-real'*:
assumes $|x| \neq \infty$
shows *ereal* (*real-of-ereal x*) = *x*
using *assms* **by** *auto*

lemma *real-ereal-id*: *real-of-ereal* $\circ$ *ereal* = *id*
proof −
{
fix *x*
have (*real-of-ereal o ereal*) *x* = *id x*
by *auto*
}
then show *?thesis*
using *ext* **by** *blast*
qed

lemma *open-image-ereal*: *open*(*UNIV*−{ ∞ , ($-\infty$ :: *ereal*)})
by (*metis range-ereal open-ereal open-UNIV*)

lemma *ereal-le-distrib*:
fixes *a b c* :: *ereal*
shows $c * (a + b) \leq c * a + c * b$
by (*cases rule*: *ereal3-cases*[*of a b c*])
(*auto simp add*: *field-simps not-le mult-le-0-iff mult-less-0-iff*)

lemma *ereal-pos-distrib*:
fixes *a b c* :: *ereal*
assumes $0 \leq c$
and $c \neq \infty$
shows $c * (a + b) = c * a + c * b$
using *assms*
by (*cases rule*: *ereal3-cases*[*of a b c*])
(*auto simp add*: *field-simps not-le mult-le-0-iff mult-less-0-iff*)

lemma *ereal-max-mono*: (*a*::*ereal*) $\leq b \Longrightarrow c \leq d \Longrightarrow max\ a\ c \leq max\ b\ d$
by (*metis sup-ereal-def sup-mono*)

lemma *ereal-max-least*: (*a*::*ereal*) $\leq x \Longrightarrow c \leq x \Longrightarrow max\ a\ c \leq x$
by (*metis sup-ereal-def sup-least*)

lemma *ereal-LimI-finite*:
fixes *x* :: *ereal*
assumes $|x| \neq \infty$
and $\bigwedge r.\ 0 < r \Longrightarrow \exists N.\ \forall n \geq N.\ u\ n < x + r \wedge x < u\ n + r$
shows $u \longrightarrow x$
proof (*rule topological-tendstoI*, *unfold eventually-sequentially*)
obtain *rx* **where** *rx*: $x = ereal\ rx$
using *assms* **by** (*cases x*) *auto*
fix *S*
assume *open S* **and** $x \in S$
then have *open* (*ereal* $-$ ' *S*)
unfolding *open-ereal-def* **by** *auto*
with ‹$x \in S$› **obtain** *r* **where** $0 < r$ **and** *dist*: $\bigwedge y.\ dist\ y\ rx < r \Longrightarrow ereal\ y \in S$
unfolding *open-dist rx* **by** *auto*
then obtain *n* **where**
upper: $\bigwedge N.\ n \leq N \Longrightarrow u\ N < x + ereal\ r$ **and**
lower: $\bigwedge N.\ n \leq N \Longrightarrow x < u\ N + ereal\ r$
using *assms*(*2*)[*of ereal r*] **by** *auto*
show $\exists N.\ \forall n \geq N.\ u\ n \in S$
proof (*safe intro*!: *exI*[*of* - *n*])
fix *N*
assume $n \leq N$
from *upper*[*OF this*] *lower*[*OF this*] *assms* ‹$0 < r$›
have $u\ N \notin \{\infty,(-\infty)\}$
by *auto*
then obtain *ra* **where** *ra-def*: $(u\ N) = ereal\ ra$
by (*cases u N*) *auto*
then have $rx < ra + r$ **and** $ra < rx + r$
using *rx assms* ‹$0 < r$› *lower*[*OF* ‹$n \leq N$›] *upper*[*OF* ‹$n \leq N$›]
by *auto*
then have *dist* (*real-of-ereal* (*u N*)) *rx* $< r$
using *rx ra-def*
by (*auto simp*: *dist-real-def abs-diff-less-iff field-simps*)

```
    from dist[OF this] show u N ∈ S
      using ‹u N ∉ {∞, −∞}›
      by (auto simp: ereal-real split: split-if-asm)
  qed
qed

lemma tendsto-obtains-N:
  assumes f ⟶ f0
  assumes open S
    and f0 ∈ S
  obtains N where ∀ n≥N. f n ∈ S
  using assms using tendsto-def
  using tendsto-explicit[of f f0] assms by auto

lemma ereal-LimI-finite-iff:
  fixes x :: ereal
  assumes |x| ≠ ∞
  shows u ⟶ x ⟷ (∀ r. 0 < r ⟶ (∃ N. ∀ n≥N. u n < x + r ∧ x < u n
+ r))
  (is ?lhs ⟷ ?rhs)
proof
  assume lim: u ⟶ x
  {
    fix r :: ereal
    assume r > 0
    then obtain N where ∀ n≥N. u n ∈ {x − r <..< x + r}
      apply (subst tendsto-obtains-N[of u x {x − r <..< x + r}])
      using lim ereal-between[of x r] assms ‹r > 0›
      apply auto
      done
    then have ∃ N. ∀ n≥N. u n < x + r ∧ x < u n + r
      using ereal-minus-less[of r x]
      by (cases r) auto
  }
  then show ?rhs
    by auto
next
  assume ?rhs
  then show u ⟶ x
    using ereal-LimI-finite[of x] assms by auto
qed

lemma ereal-Limsup-uminus:
  fixes f :: 'a ⇒ ereal
  shows Limsup net (λx. − (f x)) = − Liminf net f
  unfolding Limsup-def Liminf-def ereal-SUP-uminus ereal-INF-uminus-eq ..

lemma liminf-bounded-iff:
  fixes x :: nat ⇒ ereal
```

shows $C \leq \mathit{liminf}\ x \longleftrightarrow (\forall B{<}C.\ \exists N.\ \forall n{\geq}N.\ B < x\ n)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
unfolding *le-Liminf-iff eventually-sequentially* **..**

lemma *Liminf-add-le*:
fixes $f\ g :: \text{-} \Rightarrow \mathit{ereal}$
assumes *F*: $F \neq \mathit{bot}$
assumes *ev*: *eventually* $(\lambda x.\ 0 \leq f\ x)\ F$ *eventually* $(\lambda x.\ 0 \leq g\ x)\ F$
shows $\mathit{Liminf}\ F\ f + \mathit{Liminf}\ F\ g \leq \mathit{Liminf}\ F\ (\lambda x.\ f\ x + g\ x)$
unfolding *Liminf-def*
proof (*subst SUP-ereal-add-left*[*symmetric*])
let *?F* $= \{P.\ \mathit{eventually}\ P\ F\}$
let *?INF* $= \lambda P\ g.\ \mathit{INFIMUM}\ (\mathit{Collect}\ P)\ g$
show *?F* $\neq \{\}$
by (*auto intro*: *eventually-True*)
show $(\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ g) \neq -\infty$
unfolding *bot-ereal-def*[*symmetric*] *SUP-bot-conv INF-eq-bot-iff*
by (*auto intro*!: *exI*[*of* - *0*] *ev simp*: *bot-ereal-def*)
have $(\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ f + (\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ g)) \leq (\mathit{SUP}\ P'{:}?F.\ (\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P'\ f + ?\mathit{INF}\ P\ g))$
proof (*safe intro*!: *SUP-mono bexI*[*of* - $\lambda x.\ P\ x \wedge 0 \leq f\ x$ **for** *P*])
fix *P* **let** $?P' = \lambda x.\ P\ x \wedge 0 \leq f\ x$
assume *eventually P F*
with *ev* **show** *eventually ?P' F*
by *eventually-elim auto*
have $?\mathit{INF}\ P\ f + (\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ g) \leq ?\mathit{INF}\ ?P'\ f + (\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ g)$
by (*intro ereal-add-mono INF-mono*) *auto*
also have $\ldots = (\mathit{SUP}\ P'{:}?F.\ ?\mathit{INF}\ ?P'\ f + ?\mathit{INF}\ P'\ g)$
proof (*rule SUP-ereal-add-right*[*symmetric*])
show *INFIMUM* $\{x.\ P\ x \wedge 0 \leq f\ x\}\ f \neq -\infty$
unfolding *bot-ereal-def*[*symmetric*] *INF-eq-bot-iff*
by (*auto intro*!: *exI*[*of* - *0*] *ev simp*: *bot-ereal-def*)
qed *fact*
finally show $?\mathit{INF}\ P\ f + (\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ g) \leq (\mathit{SUP}\ P'{:}?F.\ ?\mathit{INF}\ ?P'\ f + ?\mathit{INF}\ P'\ g)$ **.**
qed
also have $\ldots \leq (\mathit{SUP}\ P{:}?F.\ \mathit{INF}\ x{:}\mathit{Collect}\ P.\ f\ x + g\ x)$
proof (*safe intro*!: *SUP-least*)
fix *P Q* **assume** $*$: *eventually P F eventually Q F*
show $?\mathit{INF}\ P\ f + ?\mathit{INF}\ Q\ g \leq (\mathit{SUP}\ P{:}?F.\ \mathit{INF}\ x{:}\mathit{Collect}\ P.\ f\ x + g\ x)$
proof (*rule SUP-upper2*)
show $(\lambda x.\ P\ x \wedge Q\ x) \in ?F$
using $*$ **by** (*auto simp*: *eventually-conj*)
show $?\mathit{INF}\ P\ f + ?\mathit{INF}\ Q\ g \leq (\mathit{INF}\ x{:}\{x.\ P\ x \wedge Q\ x\}.\ f\ x + g\ x)$
by (*intro INF-greatest ereal-add-mono*) (*auto intro*: *INF-lower*)
qed
qed
finally show $(\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ f + (\mathit{SUP}\ P{:}?F.\ ?\mathit{INF}\ P\ g)) \leq (\mathit{SUP}\ P{:}?F.$

INF x:*Collect P*. *f x* + *g x*) .
qed

lemma *Sup-ereal-mult-right'*:
assumes *nonempty*: $Y \neq \{\}$
and *x*: $x \geq 0$
shows (*SUP i*:*Y*. *f i*) ∗ *ereal x* = (*SUP i*:*Y*. *f i* ∗ *ereal x*) (**is** *?lhs* = *?rhs*)
proof(*cases x* = *0*)
case *True* **thus** *?thesis* **by**(*auto simp add*: *nonempty zero-ereal-def*[*symmetric*])
next
case *False*
show *?thesis*
proof(*rule antisym*)
show *?rhs* ≤ *?lhs*
by(*rule SUP-least*)(*simp add*: *ereal-mult-right-mono SUP-upper x*)
next
have *?lhs* / *ereal x* = (*SUP i*:*Y*. *f i*) ∗ (*ereal x* / *ereal x*) **by**(*simp only*: *ereal-times-divide-eq*)
also have ... = (*SUP i*:*Y*. *f i*) **using** *False* **by** *simp*
also have ... ≤ *?rhs* / *x*
proof(*rule SUP-least*)
fix *i*
assume *i* ∈ *Y*
have *f i* = *f i* ∗ (*ereal x* / *ereal x*) **using** *False* **by** *simp*
also have ... = *f i* ∗ *x* / *x* **by**(*simp only*: *ereal-times-divide-eq*)
also from ‹*i* ∈ *Y*› **have** *f i* ∗ *x* ≤ *?rhs* **by**(*rule SUP-upper*)
hence *f i* ∗ *x* / *x* ≤ *?rhs* / *x* **using** *x False* **by** *simp*
finally show *f i* ≤ *?rhs* / *x* .
qed
finally have (*?lhs* / *x*) ∗ *x* ≤ (*?rhs* / *x*) ∗ *x*
by(*rule ereal-mult-right-mono*)(*simp add*: *x*)
also have ... = *?rhs* **using** *False ereal-divide-eq mult.commute* **by** *force*
also have (*?lhs* / *x*) ∗ *x* = *?lhs* **using** *False ereal-divide-eq mult.commute* **by** *force*
finally show *?lhs* ≤ *?rhs* .
qed
qed

lemma *Sup-ereal-mult-left'*:
⟦ *Y* ≠ {}; *x* ≥ *0* ⟧ ⟹ *ereal x* ∗ (*SUP i*:*Y*. *f i*) = (*SUP i*:*Y*. *ereal x* ∗ *f i*)
by(*subst* (*1 2*) *mult.commute*)(*rule Sup-ereal-mult-right'*)

lemma *sup-continuous-add*[*order-continuous-intros*]:
fixes *f g* :: *'a*::*complete-lattice* ⇒ *ereal*
assumes *nn*: ⋀*x*. *0* ≤ *f x* ⋀*x*. *0* ≤ *g x* **and** *cont*: *sup-continuous f sup-continuous g*
shows *sup-continuous* (λ*x*. *f x* + *g x*)
unfolding *sup-continuous-def*
proof *safe*

fix $M :: nat \Rightarrow 'a$ **assume** *incseq M*
then show $f\ (SUP\ i.\ M\ i) + g\ (SUP\ i.\ M\ i) = (SUP\ i.\ f\ (M\ i) + g\ (M\ i))$
using *SUP-ereal-add-pos*[*of* $\lambda i.\ f\ (M\ i)$ $\lambda i.\ g\ (M\ i)$] *nn*
cont[*THEN sup-continuous-mono*] *cont*[*THEN sup-continuousD*]
by (*auto simp*: *mono-def*)
qed

lemma *sup-continuous-mult-right*[*order-continuous-intros*]:
$0 \le c \Longrightarrow c < \infty \Longrightarrow$ *sup-continuous f* $\Longrightarrow$ *sup-continuous* $(\lambda x.\ f\ x * c :: ereal)$
by (*cases c*) (*auto simp*: *sup-continuous-def fun-eq-iff Sup-ereal-mult-right'*)

lemma *sup-continuous-mult-left*[*order-continuous-intros*]:
$0 \le c \Longrightarrow c < \infty \Longrightarrow$ *sup-continuous f* $\Longrightarrow$ *sup-continuous* $(\lambda x.\ c * f\ x :: ereal)$
using *sup-continuous-mult-right*[*of c f*] **by** (*simp add*: *mult-ac*)

lemma *sup-continuous-ereal-of-enat*[*order-continuous-intros*]:
assumes *f*: *sup-continuous f* **shows** *sup-continuous* $(\lambda x.\ ereal\text{-}of\text{-}enat\ (f\ x))$
by (*rule sup-continuous-compose*[*OF - f*])
(*auto simp*: *sup-continuous-def ereal-of-enat-SUP*)

41.4.2 Sums

lemma *sums-ereal-positive*:
fixes $f :: nat \Rightarrow ereal$
assumes $\bigwedge i.\ 0 \le f\ i$
shows *f sums* $(SUP\ n.\ \sum i{<}n.\ f\ i)$
proof −
have *incseq* $(\lambda i.\ \sum j{=}0..{<}i.\ f\ j)$
using *ereal-add-mono*[*OF - assms*]
by (*auto intro*!: *incseq-SucI*)
from *LIMSEQ-SUP*[*OF this*]
show *?thesis* **unfolding** *sums-def*
by (*simp add*: *atLeast0LessThan*)
qed

lemma *summable-ereal-pos*:
fixes $f :: nat \Rightarrow ereal$
assumes $\bigwedge i.\ 0 \le f\ i$
shows *summable f*
using *sums-ereal-positive*[*of f*, *OF assms*]
unfolding *summable-def*
by *auto*

lemma *sums-ereal*: $(\lambda x.\ ereal\ (f\ x))$ *sums ereal x* $\longleftrightarrow$ *f sums x*
unfolding *sums-def* **by** *simp*

lemma *suminf-ereal-eq-SUP*:
fixes $f :: nat \Rightarrow ereal$
assumes $\bigwedge i.\ 0 \le f\ i$

shows $(\sum x.\ f\ x) = (SUP\ n.\ \sum i{<}n.\ f\ i)$
using *sums-ereal-positive*[*of f*, *OF assms*, *THEN sums-unique*]
by *simp*

lemma *suminf-bound*:
fixes $f :: nat \Rightarrow ereal$
assumes $\forall N.\ (\sum n{<}N.\ f\ n) \leq x$
and *pos*: $\bigwedge n.\ 0 \leq f\ n$
shows $suminf\ f \leq x$
proof (*rule Lim-bounded-ereal*)
have *summable f* **using** *pos*[*THEN summable-ereal-pos*] **.**
then show $(\lambda N.\ \sum n{<}N.\ f\ n) \longrightarrow suminf\ f$
by (*auto dest*!: *summable-sums simp*: *sums-def atLeast0LessThan*)
show $\forall n{\geq}0.\ setsum\ f\ \{..{<}n\} \leq x$
using *assms* **by** *auto*
qed

lemma *suminf-bound-add*:
fixes $f :: nat \Rightarrow ereal$
assumes $\forall N.\ (\sum n{<}N.\ f\ n) + y \leq x$
and *pos*: $\bigwedge n.\ 0 \leq f\ n$
and $y \neq -\infty$
shows $suminf\ f + y \leq x$
proof (*cases y*)
case (*real r*)
then have $\forall N.\ (\sum n{<}N.\ f\ n) \leq x - y$
using *assms* **by** (*simp add*: *ereal-le-minus*)
then have $(\sum\ n.\ f\ n) \leq x - y$
using *pos* **by** (*rule suminf-bound*)
then show $(\sum\ n.\ f\ n) + y \leq x$
using *assms real* **by** (*simp add*: *ereal-le-minus*)
qed (*insert assms*, *auto*)

lemma *suminf-upper*:
fixes $f :: nat \Rightarrow ereal$
assumes $\bigwedge n.\ 0 \leq f\ n$
shows $(\sum n{<}N.\ f\ n) \leq (\sum n.\ f\ n)$
unfolding *suminf-ereal-eq-SUP* [*OF assms*]
by (*auto intro*: *complete-lattice-class.SUP-upper*)

lemma *suminf-0-le*:
fixes $f :: nat \Rightarrow ereal$
assumes $\bigwedge n.\ 0 \leq f\ n$
shows $0 \leq (\sum n.\ f\ n)$
using *suminf-upper*[*of f 0*, *OF assms*]
by *simp*

lemma *suminf-le-pos*:
fixes $f\ g :: nat \Rightarrow ereal$

```
  assumes ⋀N. f N ≤ g N
    and ⋀N. 0 ≤ f N
  shows suminf f ≤ suminf g
proof (safe intro!: suminf-bound)
  fix n
  {
    fix N
    have 0 ≤ g N
      using assms(2,1)[of N] by auto
  }
  have setsum f {..<n} ≤ setsum g {..<n}
    using assms by (auto intro: setsum-mono)
  also have ... ≤ suminf g
    using ‹⋀N. 0 ≤ g N›
    by (rule suminf-upper)
  finally show setsum f {..<n} ≤ suminf g .
qed (rule assms(2))

lemma suminf-half-series-ereal: (∑ n. (1/2 :: ereal) ^ Suc n) = 1
  using sums-ereal[THEN iffD2, OF power-half-series, THEN sums-unique, sym-
metric]
  by (simp add: one-ereal-def)

lemma suminf-add-ereal:
  fixes f g :: nat ⇒ ereal
  assumes ⋀i. 0 ≤ f i
    and ⋀i. 0 ≤ g i
  shows (∑ i. f i + g i) = suminf f + suminf g
  apply (subst (1 2 3) suminf-ereal-eq-SUP)
  unfolding setsum.distrib
   apply (intro assms ereal-add-nonneg-nonneg SUP-ereal-add-pos incseq-setsumI
setsum-nonneg ballI)+
  done

lemma suminf-cmult-ereal:
  fixes f g :: nat ⇒ ereal
  assumes ⋀i. 0 ≤ f i
    and 0 ≤ a
  shows (∑ i. a * f i) = a * suminf f
  by (auto simp: setsum-ereal-right-distrib[symmetric] assms
       ereal-zero-le-0-iff setsum-nonneg suminf-ereal-eq-SUP
       intro!: SUP-ereal-mult-left)

lemma suminf-PInfty:
  fixes f :: nat ⇒ ereal
  assumes ⋀i. 0 ≤ f i
    and suminf f ≠ ∞
  shows f i ≠ ∞
proof −
```

```
  from suminf-upper[of f Suc i, OF assms(1)] assms(2)
  have (∑ i<Suc i. f i) ≠ ∞
    by auto
  then show ?thesis
    unfolding setsum-Pinfty by simp
qed

lemma suminf-PInfty-fun:
  assumes ⋀i. 0 ≤ f i
    and suminf f ≠ ∞
  shows ∃f'. f = (λx. ereal (f' x))
proof −
  have ∀ i. ∃ r. f i = ereal r
  proof
    fix i
    show ∃ r. f i = ereal r
      using suminf-PInfty[OF assms] assms(1)[of i]
      by (cases f i) auto
  qed
  from choice[OF this] show ?thesis
    by auto
qed

lemma summable-ereal:
  assumes ⋀i. 0 ≤ f i
    and (∑ i. ereal (f i)) ≠ ∞
  shows summable f
proof −
  have 0 ≤ (∑ i. ereal (f i))
    using assms by (intro suminf-0-le) auto
  with assms obtain r where r: (∑ i. ereal (f i)) = ereal r
    by (cases ∑ i. ereal (f i)) auto
  from summable-ereal-pos[of λx. ereal (f x)]
  have summable (λx. ereal (f x))
    using assms by auto
  from summable-sums[OF this]
  have (λx. ereal (f x)) sums (∑ x. ereal (f x))
    by auto
  then show summable f
    unfolding r sums-ereal summable-def ..
qed

lemma suminf-ereal:
  assumes ⋀i. 0 ≤ f i
    and (∑ i. ereal (f i)) ≠ ∞
  shows (∑ i. ereal (f i)) = ereal (suminf f)
proof (rule sums-unique[symmetric])
  from summable-ereal[OF assms]
  show (λx. ereal (f x)) sums (ereal (suminf f))
```

```
    unfolding sums-ereal
    using assms
    by (intro summable-sums summable-ereal)
qed

lemma suminf-ereal-minus:
  fixes f g :: nat ⇒ ereal
  assumes ord: ⋀i. g i ≤ f i ⋀i. 0 ≤ g i
    and fin: suminf f ≠ ∞ suminf g ≠ ∞
  shows (∑ i. f i − g i) = suminf f − suminf g
proof −
  {
    fix i
    have 0 ≤ f i
      using ord[of i] by auto
  }
  moreover
  from suminf-PInfty-fun[OF ‹⋀i. 0 ≤ f i› fin(1)] obtain f' where [simp]: f =
(λx. ereal (f' x)) ..
  from suminf-PInfty-fun[OF ‹⋀i. 0 ≤ g i› fin(2)] obtain g' where [simp]: g =
(λx. ereal (g' x)) ..
  {
    fix i
    have 0 ≤ f i − g i
      using ord[of i] by (auto simp: ereal-le-minus-iff)
  }
  moreover
  have suminf (λi. f i − g i) ≤ suminf f
    using assms by (auto intro!: suminf-le-pos simp: field-simps)
  then have suminf (λi. f i − g i) ≠ ∞
    using fin by auto
  ultimately show ?thesis
    using assms ‹⋀i. 0 ≤ f i›
    apply simp
    apply (subst (1 2 3) suminf-ereal)
    apply (auto intro!: suminf-diff[symmetric] summable-ereal)
    done
qed

lemma suminf-ereal-PInf [simp]: (∑ x. ∞::ereal) = ∞
proof −
  have (∑ i<Suc 0. ∞) ≤ (∑ x. ∞::ereal)
    by (rule suminf-upper) auto
  then show ?thesis
    by simp
qed

lemma summable-real-of-ereal:
  fixes f :: nat ⇒ ereal
```

```
  assumes f: ⋀i. 0 ≤ f i
    and fin: (∑ i. f i) ≠ ∞
  shows summable (λi. real-of-ereal (f i))
proof (rule summable-def[THEN iffD2])
  have 0 ≤ (∑ i. f i)
    using assms by (auto intro: suminf-0-le)
  with fin obtain r where r: ereal r = (∑ i. f i)
    by (cases (∑ i. f i)) auto
  {
    fix i
    have f i ≠ ∞
      using f by (intro suminf-PInfty[OF - fin]) auto
    then have |f i| ≠ ∞
      using f[of i] by auto
  }
  note fin = this
  have (λi. ereal (real-of-ereal (f i))) sums (∑ i. ereal (real-of-ereal (f i)))
    using f
    by (auto intro!: summable-ereal-pos simp: ereal-le-real-iff zero-ereal-def)
  also have ... = ereal r
    using fin r by (auto simp: ereal-real)
  finally show ∃ r. (λi. real-of-ereal (f i)) sums r
    by (auto simp: sums-ereal)
qed

lemma suminf-SUP-eq:
  fixes f :: nat ⇒ nat ⇒ ereal
  assumes ⋀i. incseq (λn. f n i)
    and ⋀n i. 0 ≤ f n i
  shows (∑ i. SUP n. f n i) = (SUP n. ∑ i. f n i)
proof −
  {
    fix n :: nat
    have (∑ i<n. SUP k. f k i) = (SUP k. ∑ i<n. f k i)
      using assms
      by (auto intro!: SUP-ereal-setsum [symmetric])
  }
  note ∗ = this
  show ?thesis
    using assms
    apply (subst (1 2) suminf-ereal-eq-SUP)
    unfolding ∗
    apply (auto intro!: SUP-upper2)
    apply (subst SUP-commute)
    apply rule
    done
qed

lemma suminf-setsum-ereal:
```

fixes $f :: - \Rightarrow - \Rightarrow ereal$
assumes *nonneg*: $\bigwedge i\ a.\ a \in A \Longrightarrow 0 \le f\ i\ a$
shows $(\sum i.\ \sum a{\in}A.\ f\ i\ a) = (\sum a{\in}A.\ \sum i.\ f\ i\ a)$
proof (*cases finite A*)
case *True*
then show *?thesis*
using *nonneg*
by *induct* (*simp-all add*: *suminf-add-ereal setsum-nonneg*)
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *suminf-ereal-eq-0*:
fixes $f :: nat \Rightarrow ereal$
assumes *nneg*: $\bigwedge i.\ 0 \le f\ i$
shows $(\sum i.\ f\ i) = 0 \longleftrightarrow (\forall i.\ f\ i = 0)$
proof
assume $(\sum i.\ f\ i) = 0$
{
fix i
assume $f\ i \neq 0$
with *nneg* **have** $0 < f\ i$
by (*auto simp*: *less-le*)
also have $f\ i = (\sum j.\ if\ j = i\ then\ f\ i\ else\ 0)$
by (*subst suminf-finite*[**where** $N{=}\{i\}$]) *auto*
also have $\ldots \le (\sum i.\ f\ i)$
using *nneg*
by (*auto intro*!: *suminf-le-pos*)
finally have *False*
using ‹$(\sum i.\ f\ i) = 0$› **by** *auto*
}
then show $\forall i.\ f\ i = 0$
by *auto*
qed *simp*

lemma *suminf-ereal-offset-le*:
fixes $f :: nat \Rightarrow ereal$
assumes *f*: $\bigwedge i.\ 0 \le f\ i$
shows $(\sum i.\ f\ (i + k)) \le suminf\ f$
proof −
have $(\lambda n.\ \sum i{<}n.\ f\ (i + k)) \longrightarrow (\sum i.\ f\ (i + k))$
using *summable-sums*[*OF summable-ereal-pos*] **by** (*simp add*: *sums-def atLeast0LessThan f*)
moreover have $(\lambda n.\ \sum i{<}n.\ f\ i) \longrightarrow (\sum i.\ f\ i)$
using *summable-sums*[*OF summable-ereal-pos*] **by** (*simp add*: *sums-def atLeast0LessThan f*)
then have $(\lambda n.\ \sum i{<}n + k.\ f\ i) \longrightarrow (\sum i.\ f\ i)$
by (*rule LIMSEQ-ignore-initial-segment*)

```
  ultimately show ?thesis
  proof (rule LIMSEQ-le, safe intro!: exI[of - k])
    fix n assume k ≤ n
    have (∑ i<n. f (i + k)) = (∑ i<n. (f ∘ (λi. i + k)) i)
      by simp
    also have ... = (∑ i∈(λi. i + k) ` {..<n}. f i)
      by (subst setsum.reindex) auto
    also have ... ≤ setsum f {..<n + k}
      by (intro setsum-mono3) (auto simp: f)
    finally show (∑ i<n. f (i + k)) ≤ setsum f {..<n + k} .
  qed
qed

lemma sums-suminf-ereal: f sums x ⟹ (∑ i. ereal (f i)) = ereal x
  by (metis sums-ereal sums-unique)

lemma suminf-ereal': summable f ⟹ (∑ i. ereal (f i)) = ereal (∑ i. f i)
  by (metis sums-ereal sums-unique summable-def)

lemma suminf-ereal-finite: summable f ⟹ (∑ i. ereal (f i)) ≠ ∞
  by (auto simp: sums-ereal[symmetric] summable-def sums-unique[symmetric])

lemma suminf-ereal-finite-neg:
  assumes summable f
  shows (∑ x. ereal (f x)) ≠ −∞
proof−
  from assms obtain x where f sums x by blast
  hence (λx. ereal (f x)) sums ereal x by (simp add: sums-ereal)
  from sums-unique[OF this] have (∑ x. ereal (f x)) = ereal x ..
  thus (∑ x. ereal (f x)) ≠ −∞ by simp-all
qed

lemma SUP-ereal-add-directed:
  fixes f g :: 'a ⇒ ereal
  assumes nonneg: ⋀i. i ∈ I ⟹ 0 ≤ f i ⋀i. i ∈ I ⟹ 0 ≤ g i
  assumes directed: ⋀i j. i ∈ I ⟹ j ∈ I ⟹ ∃ k∈I. f i + g j ≤ f k + g k
  shows (SUP i:I. f i + g i) = (SUP i:I. f i) + (SUP i:I. g i)
proof cases
  assume I = {} then show ?thesis
    by (simp add: bot-ereal-def)
next
  assume I ≠ {}
  show ?thesis
  proof (rule antisym)
    show (SUP i:I. f i + g i) ≤ (SUP i:I. f i) + (SUP i:I. g i)
      by (rule SUP-least; intro ereal-add-mono SUP-upper)
  next
    have bot < (SUP i:I. g i)
      using ‹I ≠ {}› nonneg(2) by (auto simp: bot-ereal-def less-SUP-iff)
```

```
    then have (SUP i:I. f i) + (SUP i:I. g i) = (SUP i:I. f i + (SUP i:I. g i))
      by (intro SUP-ereal-add-left[symmetric] ‹I ≠ {}›) auto
    also have ... = (SUP i:I. (SUP j:I. f i + g j))
      using nonneg(1) by (intro SUP-cong refl SUP-ereal-add-right[symmetric] ‹I
≠ {}›) auto
    also have ... ≤ (SUP i:I. f i + g i)
      using directed by (intro SUP-least) (blast intro: SUP-upper2)
    finally show (SUP i:I. f i) + (SUP i:I. g i) ≤ (SUP i:I. f i + g i) .
  qed
qed

lemma SUP-ereal-setsum-directed:
  fixes f g :: 'a ⇒ 'b ⇒ ereal
  assumes I ≠ {}
  assumes directed: ⋀N i j. N ⊆ A ⟹ i ∈ I ⟹ j ∈ I ⟹ ∃k∈I. ∀n∈N. f n i
≤ f n k ∧ f n j ≤ f n k
  assumes nonneg: ⋀n i. i ∈ I ⟹ n ∈ A ⟹ 0 ≤ f n i
  shows (SUP i:I. ∑ n∈A. f n i) = (∑ n∈A. SUP i:I. f n i)
proof −
  have N ⊆ A ⟹ (SUP i:I. ∑ n∈N. f n i) = (∑ n∈N. SUP i:I. f n i) for N
  proof (induction N rule: infinite-finite-induct)
    case (insert n N)
    moreover have (SUP i:I. f n i + (∑ l∈N. f l i)) = (SUP i:I. f n i) + (SUP
i:I. ∑ l∈N. f l i)
    proof (rule SUP-ereal-add-directed)
      fix i assume i ∈ I then show 0 ≤ f n i 0 ≤ (∑ l∈N. f l i)
        using insert by (auto intro!: setsum-nonneg nonneg)
    next
      fix i j assume i ∈ I j ∈ I
      from directed[OF ‹insert n N ⊆ A› this] guess k ..
      then show ∃k∈I. f n i + (∑ l∈N. f l j) ≤ f n k + (∑ l∈N. f l k)
        by (intro bexI[of - k]) (auto intro!: ereal-add-mono setsum-mono)
    qed
    ultimately show ?case
      by simp
  qed (simp-all add: SUP-constant ‹I ≠ {}›)
  from this[of A] show ?thesis by simp
qed

lemma suminf-SUP-eq-directed:
  fixes f :: - ⇒ nat ⇒ ereal
  assumes I ≠ {}
  assumes directed: ⋀N i j. i ∈ I ⟹ j ∈ I ⟹ finite N ⟹ ∃k∈I. ∀n∈N. f i
n ≤ f k n ∧ f j n ≤ f k n
  assumes nonneg: ⋀n i. 0 ≤ f n i
  shows (∑ i. SUP n:I. f n i) = (SUP n:I. ∑ i. f n i)
proof (subst (1 2) suminf-ereal-eq-SUP)
  show ⋀n i. 0 ≤ f n i ⋀i. 0 ≤ (SUP n:I. f n i)
    using ‹I ≠ {}› nonneg by (auto intro: SUP-upper2)
```

show (*SUP n.* $\sum i{<}n$. *SUP n*:*I. f n i*) = (*SUP n*:*I. SUP j.* $\sum i{<}j$. *f n i*)
apply (*subst SUP-commute*)
apply (*subst SUP-ereal-setsum-directed*)
apply (*auto intro*!: *assms simp*: *finite-subset*)
done
qed

lemma *ereal-dense3*:
fixes *x y* :: *ereal*
shows $x < y \Longrightarrow \exists\, r{::}rat.\ x <$ *real-of-rat r* $\wedge$ *real-of-rat r* $< y$
proof (*cases x y rule*: *ereal2-cases*, *simp-all*)
fix *r q* :: *real*
assume $r < q$
from *Rats-dense-in-real*[*OF this*] **show** $\exists\, x.\ r <$ *real-of-rat x* $\wedge$ *real-of-rat x* $< q$
by (*fastforce simp*: *Rats-def*)
next
fix *r* :: *real*
show $\exists\, x.\ r <$ *real-of-rat x* $\exists\, x.$ *real-of-rat x* $< r$
using *gt-ex*[*of r*] *lt-ex*[*of r*] *Rats-dense-in-real*
by (*auto simp*: *Rats-def*)
qed

lemma *continuous-within-ereal*[*intro*, *simp*]: $x \in A \Longrightarrow$ *continuous* (*at x within A*) *ereal*
using *continuous-on-eq-continuous-within*[*of A ereal*]
by (*auto intro*: *continuous-on-ereal continuous-on-id*)

lemma *ereal-open-uminus*:
fixes *S* :: *ereal set*
assumes *open S*
shows *open* (*uminus ' S*)
using ‹*open S*›[*unfolded open-generated-order*]
proof *induct*
have *range uminus* = (*UNIV* :: *ereal set*)
by (*auto simp*: *image-iff ereal-uminus-eq-reorder*)
then show *open* (*range uminus* :: *ereal set*)
by *simp*
qed (*auto simp add*: *image-Union image-Int*)

lemma *ereal-uminus-complement*:
fixes *S* :: *ereal set*
shows *uminus ' (− S)* = *− uminus ' S*
by (*auto intro*!: *bij-image-Compl-eq surjI*[*of - uminus*] *simp*: *bij-betw-def*)

lemma *ereal-closed-uminus*:
fixes *S* :: *ereal set*
assumes *closed S*
shows *closed* (*uminus ' S*)
using *assms*

unfolding *closed-def ereal-uminus-complement*[*symmetric*]
by (*rule ereal-open-uminus*)

lemma *ereal-open-affinity-pos*:
fixes S :: *ereal set*
assumes *open* S
and m: $m \neq \infty$ $0 < m$
and t: $|t| \neq \infty$
shows *open* $((\lambda x.\ m * x + t)\ `\ S)$
proof −
have *open* $((\lambda x.\ \mathit{inverse}\ m * (x + -t)) -` S)$
using m t
apply (*intro open-vimage* ‹*open S*›)
apply (*intro continuous-at-imp-continuous-on ballI tendsto-cmult-ereal continuous-at*[*THEN iffD2*]
tendsto-ident-at tendsto-add-left-ereal)
apply *auto*
done
also have $(\lambda x.\ \mathit{inverse}\ m * (x + -t)) -` S = (\lambda x.\ (x - t)\ /\ m) -` S$
using m t **by** (*auto simp*: *divide-ereal-def mult.commute uminus-ereal.simps*[*symmetric*] *minus-ereal-def*
simp del: *uminus-ereal.simps*)
also have $(\lambda x.\ (x - t)\ /\ m) -` S = (\lambda x.\ m * x + t)\ `\ S$
using m t
by (*simp add*: *set-eq-iff image-iff*)
(*metis abs-ereal-less0 abs-ereal-uminus ereal-divide-eq ereal-eq-minus ereal-minus*(7,8)
ereal-minus-less-minus ereal-mult-eq-PInfty ereal-uminus-uminus ereal-zero-mult)
finally show *?thesis* **.**
qed

lemma *ereal-open-affinity*:
fixes S :: *ereal set*
assumes *open* S
and m: $|m| \neq \infty$ $m \neq 0$
and t: $|t| \neq \infty$
shows *open* $((\lambda x.\ m * x + t)\ `\ S)$
proof *cases*
assume $0 < m$
then show *?thesis*
using *ereal-open-affinity-pos*[*OF* ‹*open S*› - - t, *of* m] m
by *auto*
next
assume ¬ $0 < m$ **then**
have $0 < -m$
using ‹$m \neq 0$›
by (*cases* m) *auto*
then have m: $-m \neq \infty$ $0 < -m$
using ‹$|m| \neq \infty$›

```
    by (auto simp: ereal-uminus-eq-reorder)
  from ereal-open-affinity-pos[OF ereal-open-uminus[OF ‹open S›] m t] show ?thesis
    unfolding image-image by simp
qed

lemma open-uminus-iff:
  fixes S :: ereal set
  shows open (uminus ‘ S) ⟷ open S
  using ereal-open-uminus[of S] ereal-open-uminus[of uminus ‘ S]
  by auto

lemma ereal-Liminf-uminus:
  fixes f :: 'a ⇒ ereal
  shows Liminf net (λx. − (f x)) = − Limsup net f
  using ereal-Limsup-uminus[of - (λx. − (f x))] by auto

lemma Liminf-PInfty:
  fixes f :: 'a ⇒ ereal
  assumes ¬ trivial-limit net
  shows (f ⟶ ∞) net ⟷ Liminf net f = ∞
  unfolding tendsto-iff-Liminf-eq-Limsup[OF assms]
  using Liminf-le-Limsup[OF assms, of f]
  by auto

lemma Limsup-MInfty:
  fixes f :: 'a ⇒ ereal
  assumes ¬ trivial-limit net
  shows (f ⟶ −∞) net ⟷ Limsup net f = −∞
  unfolding tendsto-iff-Liminf-eq-Limsup[OF assms]
  using Liminf-le-Limsup[OF assms, of f]
  by auto

lemma convergent-ereal:
  fixes X :: nat ⇒ 'a :: {complete-linorder,linorder-topology}
  shows convergent X ⟷ limsup X = liminf X
  using tendsto-iff-Liminf-eq-Limsup[of sequentially]
  by (auto simp: convergent-def)

lemma limsup-le-liminf-real:
  fixes X :: nat ⇒ real and L :: real
  assumes 1: limsup X ≤ L and 2: L ≤ liminf X
  shows X ⟶ L
proof −
  from 1 2 have limsup X ≤ liminf X by auto
  hence 3: limsup X = liminf X
    apply (subst eq-iff, rule conjI)
    by (rule Liminf-le-Limsup, auto)
  hence 4: convergent (λn. ereal (X n))
    by (subst convergent-ereal)
```

hence *limsup X = lim (λn. ereal(X n))*
by (*rule convergent-limsup-cl*)
also from *1 2 3* **have** *limsup X = L* **by** *auto*
finally have *lim (λn. ereal(X n)) = L* **..**
hence *(λn. ereal (X n)) ⟶ L*
apply (*elim subst*)
by (*subst convergent-LIMSEQ-iff* [*symmetric*], *rule 4*)
thus *?thesis* **by** *simp*
qed

lemma *liminf-PInfty*:
fixes *X :: nat ⇒ ereal*
shows *X ⟶ ∞ ⟷ liminf X = ∞*
by (*metis Liminf-PInfty trivial-limit-sequentially*)

lemma *limsup-MInfty*:
fixes *X :: nat ⇒ ereal*
shows *X ⟶ −∞ ⟷ limsup X = −∞*
by (*metis Limsup-MInfty trivial-limit-sequentially*)

lemma *ereal-lim-mono*:
fixes *X Y :: nat ⇒ 'a::linorder-topology*
assumes *⋀n. N ≤ n ⟹ X n ≤ Y n*
and *X ⟶ x*
and *Y ⟶ y*
shows *x ≤ y*
using *assms(1)* **by** (*intro LIMSEQ-le*[*OF assms*(*2*,*3*)]) *auto*

lemma *incseq-le-ereal*:
fixes *X :: nat ⇒ 'a::linorder-topology*
assumes *inc*: *incseq X*
and *lim*: *X ⟶ L*
shows *X N ≤ L*
using *inc*
by (*intro ereal-lim-mono*[*of N*, *OF - tendsto-const lim*]) (*simp add*: *incseq-def*)

lemma *decseq-ge-ereal*:
assumes *dec*: *decseq X*
and *lim*: *X ⟶* (*L::'a::linorder-topology*)
shows *X N ≥ L*
using *dec* **by** (*intro ereal-lim-mono*[*of N*, *OF - lim tendsto-const*]) (*simp add*: *decseq-def*)

lemma *bounded-abs*:
fixes *a :: real*
assumes *a ≤ x*
and *x ≤ b*
shows *|x| ≤ max |a| |b|*
by (*metis abs-less-iff assms leI le-max-iff-disj*

less-eq-real-def less-le-not-le less-minus-iff minus-minus)

lemma *ereal-Sup-lim*:
fixes $a :: 'a::\{complete\text{-}linorder, linorder\text{-}topology\}$
assumes $\bigwedge n.\ b\ n \in s$
and $b \longrightarrow a$
shows $a \leq Sup\ s$
by (*metis Lim-bounded-ereal assms complete-lattice-class.Sup-upper*)

lemma *ereal-Inf-lim*:
fixes $a :: 'a::\{complete\text{-}linorder, linorder\text{-}topology\}$
assumes $\bigwedge n.\ b\ n \in s$
and $b \longrightarrow a$
shows $Inf\ s \leq a$
by (*metis Lim-bounded2-ereal assms complete-lattice-class.Inf-lower*)

lemma *SUP-Lim-ereal*:
fixes $X :: nat \Rightarrow 'a::\{complete\text{-}linorder, linorder\text{-}topology\}$
assumes *inc*: *incseq X*
and *l*: $X \longrightarrow l$
shows $(SUP\ n.\ X\ n) = l$
using *LIMSEQ-SUP*[*OF inc*] *tendsto-unique*[*OF trivial-limit-sequentially l*]
by *simp*

lemma *INF-Lim-ereal*:
fixes $X :: nat \Rightarrow 'a::\{complete\text{-}linorder, linorder\text{-}topology\}$
assumes *dec*: *decseq X*
and *l*: $X \longrightarrow l$
shows $(INF\ n.\ X\ n) = l$
using *LIMSEQ-INF*[*OF dec*] *tendsto-unique*[*OF trivial-limit-sequentially l*]
by *simp*

lemma *SUP-eq-LIMSEQ*:
assumes *mono f*
shows $(SUP\ n.\ ereal\ (f\ n)) = ereal\ x \longleftrightarrow f \longrightarrow x$
proof
have *inc*: *incseq* $(\lambda i.\ ereal\ (f\ i))$
using ‹*mono f*› **unfolding** *mono-def incseq-def* **by** *auto*
{
assume $f \longrightarrow x$
then have $(\lambda i.\ ereal\ (f\ i)) \longrightarrow ereal\ x$
by *auto*
from *SUP-Lim-ereal*[*OF inc this*] **show** $(SUP\ n.\ ereal\ (f\ n)) = ereal\ x$ **.**
next
assume $(SUP\ n.\ ereal\ (f\ n)) = ereal\ x$
with *LIMSEQ-SUP*[*OF inc*] **show** $f \longrightarrow x$ **by** *auto*
}
qed

lemma *liminf-ereal-cminus*:
fixes $f :: nat \Rightarrow ereal$
assumes $c \neq -\infty$
shows $liminf\ (\lambda x.\ c - f\ x) = c - limsup\ f$
proof (*cases c*)
case *PInf*
then show *?thesis*
by (*simp add*: *Liminf-const*)
next
case (*real r*)
then show *?thesis*
unfolding *liminf-SUP-INF limsup-INF-SUP*
apply (*subst INF-ereal-minus-right*)
apply *auto*
apply (*subst SUP-ereal-minus-right*)
apply *auto*
done
qed (*insert* ‹$c \neq -\infty$›, *simp*)

41.4.3 Continuity

lemma *continuous-at-of-ereal*:
$|x0 :: ereal| \neq \infty \Longrightarrow continuous\ (at\ x0)\ real\text{-}of\text{-}ereal$
unfolding *continuous-at*
by (*rule lim-real-of-ereal*) (*simp add*: *ereal-real*)

lemma *nhds-ereal*: $nhds\ (ereal\ r) = filtermap\ ereal\ (nhds\ r)$
by (*simp add*: *filtermap-nhds-open-map open-ereal continuous-at-of-ereal*)

lemma *at-ereal*: $at\ (ereal\ r) = filtermap\ ereal\ (at\ r)$
by (*simp add*: *filter-eq-iff eventually-at-filter nhds-ereal eventually-filtermap*)

lemma *at-left-ereal*: $at\text{-}left\ (ereal\ r) = filtermap\ ereal\ (at\text{-}left\ r)$
by (*simp add*: *filter-eq-iff eventually-at-filter nhds-ereal eventually-filtermap*)

lemma *at-right-ereal*: $at\text{-}right\ (ereal\ r) = filtermap\ ereal\ (at\text{-}right\ r)$
by (*simp add*: *filter-eq-iff eventually-at-filter nhds-ereal eventually-filtermap*)

lemma
shows *at-left-PInf*: $at\text{-}left\ \infty = filtermap\ ereal\ at\text{-}top$
and *at-right-MInf*: $at\text{-}right\ (-\infty) = filtermap\ ereal\ at\text{-}bot$
unfolding *filter-eq-iff eventually-filtermap eventually-at-top-dense eventually-at-bot-dense*
eventually-at-left[*OF ereal-less*(*5*)] *eventually-at-right*[*OF ereal-less*(*6*)]
by (*auto simp add*: *ereal-all-split ereal-ex-split*)

lemma *ereal-tendsto-simps1*:
$((f \circ real\text{-}of\text{-}ereal) \longrightarrow y)\ (at\text{-}left\ (ereal\ x)) \longleftrightarrow (f \longrightarrow y)\ (at\text{-}left\ x)$
$((f \circ real\text{-}of\text{-}ereal) \longrightarrow y)\ (at\text{-}right\ (ereal\ x)) \longleftrightarrow (f \longrightarrow y)\ (at\text{-}right\ x)$
$((f \circ real\text{-}of\text{-}ereal) \longrightarrow y)\ (at\text{-}left\ (\infty::ereal)) \longleftrightarrow (f \longrightarrow y)\ at\text{-}top$

```
  ((f ∘ real-of-ereal) ⟶ y) (at-right (−∞::ereal)) ⟷ (f ⟶ y) at-bot
 unfolding tendsto-compose-filtermap at-left-ereal at-right-ereal at-left-PInf at-right-MInf
  by (auto simp: filtermap-filtermap filtermap-ident)

lemma ereal-tendsto-simps2:
  ((ereal ∘ f) ⟶ ereal a) F ⟷ (f ⟶ a) F
  ((ereal ∘ f) ⟶ ∞) F ⟷ (LIM x F. f x :> at-top)
  ((ereal ∘ f) ⟶ −∞) F ⟷ (LIM x F. f x :> at-bot)
 unfolding tendsto-PInfty filterlim-at-top-dense tendsto-MInfty filterlim-at-bot-dense
  using lim-ereal by (simp-all add: comp-def)

lemma inverse-infty-ereal-tendsto-0: inverse −∞→ (0::ereal)
proof −
  have **: ((λx. ereal (inverse x)) ⟶ ereal 0) at-infinity
    by (intro tendsto-intros tendsto-inverse-0)

  show ?thesis
     by (simp add: at-infty-ereal-eq-at-top tendsto-compose-filtermap[symmetric]
comp-def)
    (auto simp: eventually-at-top-linorder exI[of - 1] zero-ereal-def at-top-le-at-infinity
          intro!: filterlim-mono-eventually[OF **])
qed

lemma inverse-ereal-tendsto-pos:
  fixes x :: ereal assumes 0 < x
  shows inverse −x→ inverse x
proof (cases x)
  case (real r)
  with ‹0 < x› have **: (λx. ereal (inverse x)) −r→ ereal (inverse r)
    by (auto intro!: tendsto-inverse)
  from real ‹0 < x› show ?thesis
   by (auto simp: at-ereal tendsto-compose-filtermap[symmetric] eventually-at-filter
          intro!: Lim-transform-eventually[OF - **] t1-space-nhds)
qed (insert ‹0 < x›, auto intro!: inverse-infty-ereal-tendsto-0)

lemma inverse-ereal-tendsto-at-right-0: (inverse ⟶ ∞) (at-right (0::ereal))
  unfolding tendsto-compose-filtermap[symmetric] at-right-ereal zero-ereal-def
  by (subst filterlim-cong[OF refl refl, where g=λx. ereal (inverse x)])
   (auto simp: eventually-at-filter tendsto-PInfty-eq-at-top filterlim-inverse-at-top-right)

lemmas ereal-tendsto-simps = ereal-tendsto-simps1 ereal-tendsto-simps2

lemma continuous-at-iff-ereal:
  fixes f :: 'a::t2-space ⇒ real
  shows continuous (at x0 within s) f ⟷ continuous (at x0 within s) (ereal ∘ f)
  unfolding continuous-within comp-def lim-ereal ..

lemma continuous-on-iff-ereal:
  fixes f :: 'a::t2-space => real
```

assumes *open A*
shows *continuous-on A f* $\longleftrightarrow$ *continuous-on A* (*ereal* $\circ$ *f*)
unfolding *continuous-on-def comp-def lim-ereal* **..**

lemma *continuous-on-real*: *continuous-on* (*UNIV* $-$ {∞, $-\infty$::*ereal*}) *real-of-ereal*
using *continuous-at-of-ereal continuous-on-eq-continuous-at open-image-ereal*
by *auto*

lemma *continuous-on-iff-real*:
fixes *f* :: *'a*::*t2-space* $\Rightarrow$ *ereal*
assumes $*$: $\bigwedge x.\ x \in A \Longrightarrow |f\ x| \neq \infty$
shows *continuous-on A f* $\longleftrightarrow$ *continuous-on A* (*real-of-ereal* $\circ$ *f*)
proof $-$
have *f* ' *A* $\subseteq$ *UNIV* $-$ {∞, $-\infty$}
using *assms* **by** *force*
then have $*$: *continuous-on* (*f* ' *A*) *real-of-ereal*
using *continuous-on-real* **by** (*simp add*: *continuous-on-subset*)
have $**$: *continuous-on* ((*real-of-ereal* $\circ$ *f*) ' *A*) *ereal*
by (*intro continuous-on-ereal continuous-on-id*)
{
assume *continuous-on A f*
then have *continuous-on A* (*real-of-ereal* $\circ$ *f*)
apply (*subst continuous-on-compose*)
using $*$
apply *auto*
done
}
moreover
{
assume *continuous-on A* (*real-of-ereal* $\circ$ *f*)
then have *continuous-on A* (*ereal* $\circ$ (*real-of-ereal* $\circ$ *f*))
apply (*subst continuous-on-compose*)
using $**$
apply *auto*
done
then have *continuous-on A f*
apply (*subst continuous-on-cong*[*of - A - ereal* $\circ$ (*real-of-ereal* $\circ$ *f*)])
using *assms ereal-real*
apply *auto*
done
}
ultimately show *?thesis*
by *auto*
qed

lemma *continuous-uminus-ereal* [*continuous-intros*]: *continuous-on* (*A* :: *ereal set*) *uminus*
unfolding *continuous-on-def*
by (*intro ballI tendsto-uminus-ereal*[*of* $\lambda x.\ x$::*ereal*]) *simp*

lemma *ereal-uminus-atMost* [*simp*]: *uminus ‘ {..(a::ereal)} = {−a..}*
proof (*intro equalityI subsetI*)
fix *x* :: *ereal* **assume** *x* ∈ *{−a..}*
hence *−(−x) ∈ uminus ‘ {..a}* **by** (*intro imageI*) (*simp add*: *ereal-uminus-le-reorder*)
thus *x ∈ uminus ‘ {..a}* **by** *simp*
qed *auto*

lemma *continuous-on-inverse-ereal* [*continuous-intros*]:
continuous-on {0::ereal ..} inverse
unfolding *continuous-on-def*
proof *clarsimp*
fix *x* :: *ereal* **assume** *0 ≤ x*
moreover have *at 0 within {0 ..} = at-right (0::ereal)*
by (*auto simp*: *filter-eq-iff eventually-at-filter le-less*)
moreover have *at x within {0 ..} = at x* **if** *0 < x*
using *that* **by** (*intro at-within-nhd*[*of - {0<..}*]) *auto*
ultimately show (*inverse ⟶ inverse x*) (*at x within {0..}*)
by (*auto simp*: *le-less inverse-ereal-tendsto-at-right-0 inverse-ereal-tendsto-pos*)
qed

lemma *continuous-inverse-ereal-nonpos*: *continuous-on* ({*..<0*} :: *ereal set*) *inverse*
proof (*subst continuous-on-cong*[*OF refl*])
have *continuous-on {(0::ereal)<..} inverse*
by (*rule continuous-on-subset*[*OF continuous-on-inverse-ereal*]) *auto*
thus *continuous-on {..<(0::ereal)}* (*uminus ∘ inverse ∘ uminus*)
by (*intro continuous-intros*) *simp-all*
qed *simp*

lemma *tendsto-inverse-ereal*:
assumes (*f ⟶* (*c* :: *ereal*)) *F*
assumes *eventually* (*λx. f x ≥ 0*) *F*
shows ((*λx. inverse* (*f x*)) *⟶ inverse c*) *F*
by (*cases F = bot*)
(*auto intro*!: *tendsto-le-const*[*of F*] *assms*
continuous-on-tendsto-compose[*OF continuous-on-inverse-ereal*])

41.4.4 liminf and limsup

lemma *Limsup-ereal-mult-right*:
assumes *F ≠ bot* (*c*::*real*) *≥ 0*
shows *Limsup F* (*λn. f n ∗ ereal c*) = *Limsup F f ∗ ereal c*
proof (*rule Limsup-compose-continuous-mono*)
from *assms* **show** *continuous-on UNIV* (*λa. a ∗ ereal c*)
using *tendsto-cmult-ereal*[*of ereal c λx. x*]
by (*force simp*: *continuous-on-def mult-ac*)
qed (*insert assms*, *auto simp*: *mono-def ereal-mult-right-mono*)

lemma *Liminf-ereal-mult-right*:

assumes $F \neq bot$ $(c::real) \geq 0$
shows $Liminf\ F\ (\lambda n.\ f\ n * ereal\ c) = Liminf\ F\ f * ereal\ c$
proof (*rule Liminf-compose-continuous-mono*)
from *assms* **show** *continuous-on UNIV* $(\lambda a.\ a * ereal\ c)$
using *tendsto-cmult-ereal*[*of ereal c* $\lambda x.\ x$]
by (*force simp*: *continuous-on-def mult-ac*)
qed (*insert assms*, *auto simp*: *mono-def ereal-mult-right-mono*)

lemma *Limsup-ereal-mult-left*:
assumes $F \neq bot$ $(c::real) \geq 0$
shows $Limsup\ F\ (\lambda n.\ ereal\ c * f\ n) = ereal\ c * Limsup\ F\ f$
using *Limsup-ereal-mult-right*[*OF assms*] **by** (*subst* (*1 2*) *mult.commute*)

lemma *limsup-ereal-mult-right*:
$(c::real) \geq 0 \Longrightarrow limsup\ (\lambda n.\ f\ n * ereal\ c) = limsup\ f * ereal\ c$
by (*rule Limsup-ereal-mult-right*) *simp-all*

lemma *limsup-ereal-mult-left*:
$(c::real) \geq 0 \Longrightarrow limsup\ (\lambda n.\ ereal\ c * f\ n) = ereal\ c * limsup\ f$
by (*subst* (*1 2*) *mult.commute*, *rule limsup-ereal-mult-right*) *simp-all*

lemma *Limsup-add-ereal-right*:
$F \neq bot \Longrightarrow abs\ c \neq \infty \Longrightarrow Limsup\ F\ (\lambda n.\ g\ n + (c :: ereal)) = Limsup\ F\ g + c$
by (*rule Limsup-compose-continuous-mono*) (*auto simp*: *mono-def ereal-add-mono continuous-on-def*)

lemma *Limsup-add-ereal-left*:
$F \neq bot \Longrightarrow abs\ c \neq \infty \Longrightarrow Limsup\ F\ (\lambda n.\ (c :: ereal) + g\ n) = c + Limsup\ F\ g$
by (*subst* (*1 2*) *add.commute*) (*rule Limsup-add-ereal-right*)

lemma *Liminf-add-ereal-right*:
$F \neq bot \Longrightarrow abs\ c \neq \infty \Longrightarrow Liminf\ F\ (\lambda n.\ g\ n + (c :: ereal)) = Liminf\ F\ g + c$
by (*rule Liminf-compose-continuous-mono*) (*auto simp*: *mono-def ereal-add-mono continuous-on-def*)

lemma *Liminf-add-ereal-left*:
$F \neq bot \Longrightarrow abs\ c \neq \infty \Longrightarrow Liminf\ F\ (\lambda n.\ (c :: ereal) + g\ n) = c + Liminf\ F\ g$
by (*subst* (*1 2*) *add.commute*) (*rule Liminf-add-ereal-right*)

lemma
assumes $F \neq bot$
assumes *nonneg*: *eventually* $(\lambda x.\ f\ x \geq (0::ereal))\ F$
shows *Liminf-inverse-ereal*: $Liminf\ F\ (\lambda x.\ inverse\ (f\ x)) = inverse\ (Limsup\ F\ f)$
and *Limsup-inverse-ereal*: $Limsup\ F\ (\lambda x.\ inverse\ (f\ x)) = inverse\ (Liminf\ F\ f)$
proof −

def *inv* ≡ λ*x*. *if x* ≤ *0 then* ∞ *else inverse x* :: *ereal*
have *continuous-on* ({..*0*} ∪ {*0*..}) *inv* **unfolding** *inv-def*
by (*intro continuous-on-If*) (*auto intro*!: *continuous-intros*)
also have {..*0*} ∪ {*0*..} = (*UNIV* :: *ereal set*) **by** *auto*
finally have *cont*: *continuous-on UNIV inv* **.**
have *antimono*: *antimono inv* **unfolding** *inv-def antimono-def*
by (*auto intro*!: *ereal-inverse-antimono*)

have *Liminf F* (λ*x*. *inverse* (*f x*)) = *Liminf F* (λ*x*. *inv* (*f x*)) **using** *nonneg*
by (*auto intro*!: *Liminf-eq elim*!: *eventually-mono simp*: *inv-def*)
also have ... = *inv* (*Limsup F f*)
by (*simp add*: *assms*(*1*) *Liminf-compose-continuous-antimono*[*OF cont anti-mono*])
also from *assms* **have** *Limsup F f* ≥ *0* **by** (*intro le-Limsup*) *simp-all*
hence *inv* (*Limsup F f*) = *inverse* (*Limsup F f*) **by** (*simp add*: *inv-def*)
finally show *Liminf F* (λ*x*. *inverse* (*f x*)) = *inverse* (*Limsup F f*) **.**

have *Limsup F* (λ*x*. *inverse* (*f x*)) = *Limsup F* (λ*x*. *inv* (*f x*)) **using** *nonneg*
by (*auto intro*!: *Limsup-eq elim*!: *eventually-mono simp*: *inv-def*)
also have ... = *inv* (*Liminf F f*)
by (*simp add*: *assms*(*1*) *Limsup-compose-continuous-antimono*[*OF cont anti-mono*])
also from *assms* **have** *Liminf F f* ≥ *0* **by** (*intro Liminf-bounded*) *simp-all*
hence *inv* (*Liminf F f*) = *inverse* (*Liminf F f*) **by** (*simp add*: *inv-def*)
finally show *Limsup F* (λ*x*. *inverse* (*f x*)) = *inverse* (*Liminf F f*) **.**
qed

41.4.5 Tests for code generator

value − ∞ :: *ereal*
value |−∞| :: *ereal*
value *4 + 5 / 4 − ereal 2* :: *ereal*
value *ereal 3* < ∞
value *real-of-ereal* (∞::*ereal*) = *0*

end

42 Limits on the Extended real number line

theory *Extended-Real-Limits*
imports
Topology-Euclidean-Space
~~/*src*/*HOL*/*Library*/*Extended-Real*
~~/*src*/*HOL*/*Library*/*Indicator-Function*
begin

lemma *compact-UNIV*:
compact (*UNIV* :: '*a*::{*complete-linorder*,*linorder-topology*,*second-countable-topology*} *set*)

using *compact-complete-linorder*
by (*auto simp*: *seq-compact-eq-compact*[*symmetric*] *seq-compact-def*)

lemma *compact-eq-closed*:
fixes S :: *'a*::{*complete-linorder*,*linorder-topology*,*second-countable-topology*} *set*
shows *compact* $S \longleftrightarrow$ *closed* S
using *closed-inter-compact*[*of* S, *OF - compact-UNIV*] *compact-imp-closed*
by *auto*

lemma *closed-contains-Sup-cl*:
fixes S :: *'a*::{*complete-linorder*,*linorder-topology*,*second-countable-topology*} *set*
assumes *closed* S
and $S \neq \{\}$
shows *Sup* $S \in S$
proof −
from *compact-eq-closed*[*of* S] *compact-attains-sup*[*of* S] *assms*
obtain s **where** S: $s \in S \; \forall\, t{\in}S.\; t \leq s$
by *auto*
then have *Sup* $S = s$
by (*auto intro*!: *Sup-eqI*)
with S **show** *?thesis*
by *simp*
qed

lemma *closed-contains-Inf-cl*:
fixes S :: *'a*::{*complete-linorder*,*linorder-topology*,*second-countable-topology*} *set*
assumes *closed* S
and $S \neq \{\}$
shows *Inf* $S \in S$
proof −
from *compact-eq-closed*[*of* S] *compact-attains-inf*[*of* S] *assms*
obtain s **where** S: $s \in S \; \forall\, t{\in}S.\; s \leq t$
by *auto*
then have *Inf* $S = s$
by (*auto intro*!: *Inf-eqI*)
with S **show** *?thesis*
by *simp*
qed

instance *ereal* :: *second-countable-topology*
proof (*standard*, *intro exI conjI*)
let *?B* = $(\bigcup r{\in}\mathbb{Q}.\; \{\{..< r\}, \{r <..\}\}$ :: *ereal set set*)
show *countable ?B*
by (*auto intro*: *countable-rat*)
show *open* = *generate-topology ?B*
proof (*intro ext iffI*)
fix S :: *ereal set*
assume *open* S
then show *generate-topology ?B S*

```
    unfolding open-generated-order
   proof induct
    case (Basis b)
    then obtain e where b = {..<e} ∨ b = {e<..}
     by auto
   moreover have {..<e} = ⋃{{..<x}|x. x ∈ ℚ ∧ x < e} {e<..} = ⋃{{x<..}|x.
x ∈ ℚ ∧ e < x}
     by (auto dest: ereal-dense3
           simp del: ex-simps
           simp add: ex-simps[symmetric] conj-commute Rats-def image-iff)
    ultimately show ?case
     by (auto intro: generate-topology.intros)
   qed (auto intro: generate-topology.intros)
  next
   fix S
   assume generate-topology ?B S
   then show open S
    by induct auto
  qed
qed

lemma ereal-open-closed-aux:
  fixes S :: ereal set
  assumes open S
    and closed S
    and S: (−∞) ∉ S
  shows S = {}
proof (rule ccontr)
  assume ¬ ?thesis
  then have ∗: Inf S ∈ S

    by (metis assms(2) closed-contains-Inf-cl)
  {
   assume Inf S = −∞
   then have False
    using ∗ assms(3) by auto
  }
  moreover
  {
   assume Inf S = ∞
   then have S = {∞}
    by (metis Inf-eq-PInfty ‹S ≠ {}›)
   then have False
    by (metis assms(1) not-open-singleton)
  }
  moreover
  {
   assume fin: |Inf S| ≠ ∞
   from ereal-open-cont-interval[OF assms(1) ∗ fin]
```

```
    obtain e where e: e > 0 {Inf S − e<..<Inf S + e} ⊆ S .
    then obtain b where b: Inf S − e < b b < Inf S
      using fin ereal-between[of Inf S e] dense[of Inf S − e]
      by auto
    then have b: {Inf S − e <..< Inf S + e}
      using e fin ereal-between[of Inf S e]
      by auto
    then have b ∈ S
      using e by auto
    then have False
      using b by (metis complete-lattice-class.Inf-lower leD)
  }
  ultimately show False
    by auto
qed

lemma ereal-open-closed:
  fixes S :: ereal set
  shows open S ∧ closed S ⟷ S = {} ∨ S = UNIV
proof −
  {
    assume lhs: open S ∧ closed S
    {
      assume −∞ ∉ S
      then have S = {}
        using lhs ereal-open-closed-aux by auto
    }
    moreover
    {
      assume −∞ ∈ S
      then have − S = {}
        using lhs ereal-open-closed-aux[of −S] by auto
    }
    ultimately have S = {} ∨ S = UNIV
      by auto
  }
  then show ?thesis
    by auto
qed

lemma ereal-open-atLeast:
  fixes x :: ereal
  shows open {x..} ⟷ x = −∞
proof
  assume x = −∞
  then have {x..} = UNIV
    by auto
  then show open {x..}
    by auto
```

next
assume *open* $\{x..\}$
then have *open* $\{x..\} \wedge$ *closed* $\{x..\}$
by *auto*
then have $\{x..\} = UNIV$
unfolding *ereal-open-closed* **by** *auto*
then show $x = -\infty$
by (*simp add*: *bot-ereal-def atLeast-eq-UNIV-iff*)
qed

lemma *mono-closed-real*:
fixes $S :: real\ set$
assumes *mono*: $\forall\, y\ z.\ y \in S \wedge y \leq z \longrightarrow z \in S$
and *closed* S
shows $S = \{\} \vee S = UNIV \vee (\exists\, a.\ S = \{a..\})$
proof −
{
assume $S \neq \{\}$
{ assume *ex*: $\exists\, B.\ \forall\, x{\in}S.\ B \leq x$
then have $*$: $\forall\, x{\in}S.\ Inf\ S \leq x$
using *cInf-lower*[*of* - *S*] *ex* **by** (*metis bdd-below-def*)
then have $Inf\ S \in S$
apply (*subst closed-contains-Inf*)
using *ex* ‹$S \neq \{\}$› ‹*closed S*›
apply *auto*
done
then have $\forall\, x.\ Inf\ S \leq x \longleftrightarrow x \in S$
using *mono*[*rule-format*, *of Inf S*] $*$
by *auto*
then have $S = \{Inf\ S\ ..\}$
by *auto*
then have $\exists\, a.\ S = \{a\ ..\}$
by *auto*
}
moreover
{
assume $\neg\ (\exists\, B.\ \forall\, x{\in}S.\ B \leq x)$
then have *nex*: $\forall\, B.\ \exists\, x{\in}S.\ x < B$
by (*simp add*: *not-le*)
{
fix y
obtain x **where** $x{\in}S$ **and** $x < y$
using *nex* **by** *auto*
then have $y \in S$
using *mono*[*rule-format*, *of x y*] **by** *auto*
}
then have $S = UNIV$
by *auto*
}

```
    ultimately have S = UNIV ∨ (∃ a. S = {a ..})
      by blast
  }
  then show ?thesis
    by blast
qed

lemma mono-closed-ereal:
  fixes S :: real set
  assumes mono: ∀ y z. y ∈ S ∧ y ≤ z ⟶ z ∈ S
    and closed S
  shows ∃ a. S = {x. a ≤ ereal x}
proof −
  {
    assume S = {}
    then have ?thesis
      apply (rule-tac x=PInfty in exI)
      apply auto
      done
  }
  moreover
  {
    assume S = UNIV
    then have ?thesis
      apply (rule-tac x=−∞ in exI)
      apply auto
      done
  }
  moreover
  {
    assume ∃ a. S = {a ..}
    then obtain a where S = {a ..}
      by auto
    then have ?thesis
      apply (rule-tac x=ereal a in exI)
      apply auto
      done
  }
  ultimately show ?thesis
    using mono-closed-real[of S] assms by auto
qed

lemma Liminf-within:
  fixes f :: 'a::metric-space ⇒ 'b::complete-lattice
  shows Liminf (at x within S) f = (SUP e:{0<..}. INF y:(S ∩ ball x e − {x}).
f y)
  unfolding Liminf-def eventually-at
proof (rule SUP-eq, simp-all add: Ball-def Bex-def, safe)
  fix P d
```

assume $0 < d$ **and** $\forall\, y.\ y \in S \longrightarrow y \neq x \wedge \mathit{dist}\ y\ x < d \longrightarrow P\ y$
then have $S \cap \mathit{ball}\ x\ d - \{x\} \subseteq \{x.\ P\ x\}$
by (*auto simp*: *zero-less-dist-iff dist-commute*)
then show $\exists\, r{>}0.\ \mathit{INFIMUM}\ (\mathit{Collect}\ P)\ f \leq \mathit{INFIMUM}\ (S \cap \mathit{ball}\ x\ r - \{x\})\ f$
by (*intro exI*[*of* - *d*] *INF-mono conjI* ‹$0 < d$›) *auto*
next
fix d :: *real*
assume $0 < d$
then show $\exists\, P.\ (\exists\, d{>}0.\ \forall\, xa.\ xa \in S \longrightarrow xa \neq x \wedge \mathit{dist}\ xa\ x < d \longrightarrow P\ xa) \wedge \mathit{INFIMUM}\ (S \cap \mathit{ball}\ x\ d - \{x\})\ f \leq \mathit{INFIMUM}\ (\mathit{Collect}\ P)\ f$
by (*intro exI*[*of* - $\lambda y.\ y \in S \cap \mathit{ball}\ x\ d - \{x\}$])
(*auto intro*!: *INF-mono exI*[*of* - *d*] *simp*: *dist-commute*)
qed

lemma *Limsup-within*:
fixes f :: $'a$::*metric-space* ⇒ $'b$::*complete-lattice*
shows *Limsup* (*at x within S*) f = (*INF e*:{*0*<..}. *SUP y*:($S \cap \mathit{ball}\ x\ e - \{x\}$). $f\ y$)
unfolding *Limsup-def eventually-at*
proof (*rule INF-eq*, *simp-all add*: *Ball-def Bex-def*, *safe*)
fix P d
assume $0 < d$ **and** $\forall\, y.\ y \in S \longrightarrow y \neq x \wedge \mathit{dist}\ y\ x < d \longrightarrow P\ y$
then have $S \cap \mathit{ball}\ x\ d - \{x\} \subseteq \{x.\ P\ x\}$
by (*auto simp*: *zero-less-dist-iff dist-commute*)
then show $\exists\, r{>}0.\ \mathit{SUPREMUM}\ (S \cap \mathit{ball}\ x\ r - \{x\})\ f \leq \mathit{SUPREMUM}\ (\mathit{Collect}\ P)\ f$
by (*intro exI*[*of* - *d*] *SUP-mono conjI* ‹$0 < d$›) *auto*
next
fix d :: *real*
assume $0 < d$
then show $\exists\, P.\ (\exists\, d{>}0.\ \forall\, xa.\ xa \in S \longrightarrow xa \neq x \wedge \mathit{dist}\ xa\ x < d \longrightarrow P\ xa) \wedge \mathit{SUPREMUM}\ (\mathit{Collect}\ P)\ f \leq \mathit{SUPREMUM}\ (S \cap \mathit{ball}\ x\ d - \{x\})\ f$
by (*intro exI*[*of* - $\lambda y.\ y \in S \cap \mathit{ball}\ x\ d - \{x\}$])
(*auto intro*!: *SUP-mono exI*[*of* - *d*] *simp*: *dist-commute*)
qed

lemma *Liminf-at*:
fixes f :: $'a$::*metric-space* ⇒ $'b$::*complete-lattice*
shows *Liminf* (*at x*) f = (*SUP e*:{*0*<..}. *INF y*:(*ball x e* − {*x*}). $f\ y$)
using *Liminf-within*[*of x UNIV f*] **by** *simp*

lemma *Limsup-at*:
fixes f :: $'a$::*metric-space* ⇒ $'b$::*complete-lattice*
shows *Limsup* (*at x*) f = (*INF e*:{*0*<..}. *SUP y*:(*ball x e* − {*x*}). $f\ y$)
using *Limsup-within*[*of x UNIV f*] **by** *simp*

lemma *min-Liminf-at*:
fixes f :: $'a$::*metric-space* ⇒ $'b$::*complete-linorder*

shows *min* (*f x*) (*Liminf* (*at x*) *f*) = (*SUP e*:{*0*<..}. *INF y*:*ball x e*. *f y*)
unfolding *inf-min*[*symmetric*] *Liminf-at*
apply (*subst inf-commute*)
apply (*subst SUP-inf*)
apply (*intro SUP-cong*[*OF refl*])
apply (*cut-tac A*=*ball x xa* − {*x*} **and** *B*={*x*} **and** *M*=*f* **in** *INF-union*)
apply (*drule sym*)
apply *auto*
apply (*metis INF-absorb centre-in-ball*)
done

42.1 monoset

definition (**in** *order*) *mono-set*:
mono-set S ⟷ (∀ *x y*. *x* ≤ *y* ⟶ *x* ∈ *S* ⟶ *y* ∈ *S*)

lemma (**in** *order*) *mono-greaterThan* [*intro*, *simp*]: *mono-set* {*B*<..} **unfolding** *mono-set* **by** *auto*
lemma (**in** *order*) *mono-atLeast* [*intro*, *simp*]: *mono-set* {*B*..} **unfolding** *mono-set* **by** *auto*
lemma (**in** *order*) *mono-UNIV* [*intro*, *simp*]: *mono-set UNIV* **unfolding** *mono-set* **by** *auto*
lemma (**in** *order*) *mono-empty* [*intro*, *simp*]: *mono-set* {} **unfolding** *mono-set* **by** *auto*

lemma (**in** *complete-linorder*) *mono-set-iff*:
fixes *S* :: *'a set*
defines *a* ≡ *Inf S*
shows *mono-set S* ⟷ *S* = {*a* <..} ∨ *S* = {*a*..} (**is** - = *?c*)
proof
assume *mono-set S*
then have *mono*: ⋀*x y*. *x* ≤ *y* ⟹ *x* ∈ *S* ⟹ *y* ∈ *S*
by (*auto simp*: *mono-set*)
show *?c*
proof *cases*
assume *a* ∈ *S*
show *?c*
using *mono*[*OF* - ‹*a* ∈ *S*›]
by (*auto intro*: *Inf-lower simp*: *a-def*)
next
assume *a* ∉ *S*
have *S* = {*a* <..}
proof *safe*
fix *x* **assume** *x* ∈ *S*
then have *a* ≤ *x*
unfolding *a-def* **by** (*rule Inf-lower*)
then show *a* < *x*
using ‹*x* ∈ *S*› ‹*a* ∉ *S*› **by** (*cases a* = *x*) *auto*
next

```
      fix x assume a < x
      then obtain y where y < x y ∈ S
        unfolding a-def Inf-less-iff ..
      with mono[of y x] show x ∈ S
        by auto
    qed
    then show ?c ..
  qed
qed auto

lemma ereal-open-mono-set:
  fixes S :: ereal set
  shows open S ∧ mono-set S ⟷ S = UNIV ∨ S = {Inf S <..}
  by (metis Inf-UNIV atLeast-eq-UNIV-iff ereal-open-atLeast
    ereal-open-closed mono-set-iff open-ereal-greaterThan)

lemma ereal-closed-mono-set:
  fixes S :: ereal set
  shows closed S ∧ mono-set S ⟷ S = {} ∨ S = {Inf S ..}
  by (metis Inf-UNIV atLeast-eq-UNIV-iff closed-ereal-atLeast
    ereal-open-closed mono-empty mono-set-iff open-ereal-greaterThan)

lemma ereal-Liminf-Sup-monoset:
  fixes f :: 'a ⇒ ereal
  shows Liminf net f =
     Sup {l. ∀ S. open S ⟶ mono-set S ⟶ l ∈ S ⟶ eventually (λx. f x ∈ S)
net}
    (is - = Sup ?A)
proof (safe intro!: Liminf-eqI complete-lattice-class.Sup-upper complete-lattice-class.Sup-least)
  fix P
  assume P: eventually P net
  fix S
  assume S: mono-set S INFIMUM (Collect P) f ∈ S
  {
    fix x
    assume P x
    then have INFIMUM (Collect P) f ≤ f x
      by (intro complete-lattice-class.INF-lower) simp
    with S have f x ∈ S
      by (simp add: mono-set)
  }
  with P show eventually (λx. f x ∈ S) net
    by (auto elim: eventually-mono)
next
  fix y l
  assume S: ∀ S. open S ⟶ mono-set S ⟶ l ∈ S ⟶ eventually (λx. f x ∈ S)
net
  assume P: ∀ P. eventually P net ⟶ INFIMUM (Collect P) f ≤ y
  show l ≤ y
```

```
  proof (rule dense-le)
    fix B
    assume B < l
    then have eventually (λx. f x ∈ {B <..}) net
      by (intro S[rule-format]) auto
    then have INFIMUM {x. B < f x} f ≤ y
      using P by auto
    moreover have B ≤ INFIMUM {x. B < f x} f
      by (intro INF-greatest) auto
    ultimately show B ≤ y
      by simp
  qed
qed

lemma ereal-Limsup-Inf-monoset:
  fixes f :: 'a ⇒ ereal
  shows Limsup net f =
    Inf {l. ∀ S. open S ⟶ mono-set (uminus ` S) ⟶ l ∈ S ⟶ eventually (λx.
f x ∈ S) net}
    (is - = Inf ?A)
proof (safe intro!: Limsup-eqI complete-lattice-class.Inf-lower complete-lattice-class.Inf-greatest)
  fix P
  assume P: eventually P net
  fix S
  assume S: mono-set (uminus`S) SUPREMUM (Collect P) f ∈ S
  {
    fix x
    assume P x
    then have f x ≤ SUPREMUM (Collect P) f
      by (intro complete-lattice-class.SUP-upper) simp
     with S(1)[unfolded mono-set, rule-format, of - SUPREMUM (Collect P) f
- f x] S(2)
    have f x ∈ S
      by (simp add: inj-image-mem-iff) }
  with P show eventually (λx. f x ∈ S) net
    by (auto elim: eventually-mono)
next
  fix y l
  assume S: ∀ S. open S ⟶ mono-set (uminus ` S) ⟶ l ∈ S ⟶ eventually
(λx. f x ∈ S) net
  assume P: ∀ P. eventually P net ⟶ y ≤ SUPREMUM (Collect P) f
  show y ≤ l
  proof (rule dense-ge)
    fix B
    assume l < B
    then have eventually (λx. f x ∈ {..< B}) net
      by (intro S[rule-format]) auto
    then have y ≤ SUPREMUM {x. f x < B} f
      using P by auto
```

```
    moreover have SUPREMUM {x. f x < B} f ≤ B
      by (intro SUP-least) auto
    ultimately show y ≤ B
      by simp
  qed
qed

lemma liminf-bounded-open:
  fixes x :: nat ⇒ ereal
  shows x0 ≤ liminf x ⟷ (∀ S. open S ⟶ mono-set S ⟶ x0 ∈ S ⟶ (∃ N.
∀ n≥N. x n ∈ S))
  (is - ⟷ ?P x0)
proof
  assume ?P x0
  then show x0 ≤ liminf x
    unfolding ereal-Liminf-Sup-monoset eventually-sequentially
    by (intro complete-lattice-class.Sup-upper) auto
next
  assume x0 ≤ liminf x
  {
    fix S :: ereal set
    assume om: open S mono-set S x0 ∈ S
    {
      assume S = UNIV
      then have ∃ N. ∀ n≥N. x n ∈ S
        by auto
    }
    moreover
    {
      assume S ≠ UNIV
      then obtain B where B: S = {B<..}
        using om ereal-open-mono-set by auto
      then have B < x0
        using om by auto
      then have ∃ N. ∀ n≥N. x n ∈ S
        unfolding B
        using ‹x0 ≤ liminf x› liminf-bounded-iff
        by auto
    }
    ultimately have ∃ N. ∀ n≥N. x n ∈ S
      by auto
  }
  then show ?P x0
    by auto
qed
```

42.2 Relate extended reals and the indicator function

```
lemma ereal-indicator-le-0: (indicator S x::ereal) ≤ 0 ⟷ x ∉ S
```

by (*auto split*: *split-indicator simp*: *one-ereal-def*)

lemma *ereal-indicator*: *ereal* (*indicator A x*) = *indicator A x*
by (*auto simp*: *indicator-def one-ereal-def*)

lemma *ereal-mult-indicator*: *ereal* (*x* ∗ *indicator A y*) = *ereal x* ∗ *indicator A y*
by (*simp split*: *split-indicator*)

lemma *ereal-indicator-mult*: *ereal* (*indicator A y* ∗ *x*) = *indicator A y* ∗ *ereal x*
by (*simp split*: *split-indicator*)

lemma *ereal-indicator-nonneg*[*simp*, *intro*]: *0* ≤ (*indicator A x* ::*ereal*)
unfolding *indicator-def* **by** *auto*

lemma *indicator-inter-arith-ereal*: *indicator A x* ∗ *indicator B x* = (*indicator* (*A* ∩ *B*) *x* :: *ereal*)
by (*simp split*: *split-indicator*)

end

43 Permutations, both general and specifically on finite sets.

theory *Permutations*
imports *Binomial*
begin

43.1 Transpositions

lemma *swap-id-idempotent* [*simp*]:
Fun.swap a b id ∘ *Fun.swap a b id* = *id*
by (*rule ext*, *auto simp add*: *Fun.swap-def*)

lemma *inv-swap-id*:
inv (*Fun.swap a b id*) = *Fun.swap a b id*
by (*rule inv-unique-comp*) *simp-all*

lemma *swap-id-eq*:
Fun.swap a b id x = (*if x* = *a then b else if x* = *b then a else x*)
by (*simp add*: *Fun.swap-def*)

43.2 Basic consequences of the definition

definition *permutes* (**infixr** *permutes 41*)
where (*p permutes S*) ⟷ (∀ *x*. *x* ∉ *S* ⟶ *p x* = *x*) ∧ (∀ *y*. ∃!*x*. *p x* = *y*)

lemma *permutes-in-image*: *p permutes S* ⟹ *p x* ∈ *S* ⟷ *x* ∈ *S*
unfolding *permutes-def* **by** *metis*

lemma *permutes-image*: *p permutes S* $\Longrightarrow$ *p ' S = S*
unfolding *permutes-def*
apply (*rule set-eqI*)
apply (*simp add*: *image-iff*)
apply *metis*
done

lemma *permutes-inj*: *p permutes S* $\Longrightarrow$ *inj p*
unfolding *permutes-def inj-on-def* **by** *blast*

lemma *permutes-surj*: *p permutes s* $\Longrightarrow$ *surj p*
unfolding *permutes-def surj-def* **by** *metis*

lemma *permutes-bij*: *p permutes s* $\Longrightarrow$ *bij p*
unfolding *bij-def* **by** (*metis permutes-inj permutes-surj*)

lemma *permutes-imp-bij*: *p permutes S* $\Longrightarrow$ *bij-betw p S S*
by (*metis UNIV-I bij-betw-subset permutes-bij permutes-image subsetI*)

lemma *bij-imp-permutes*: *bij-betw p S S* $\Longrightarrow$ ($\bigwedge x.\ x \notin S \Longrightarrow p\ x = x$) $\Longrightarrow$ *p permutes S*
unfolding *permutes-def bij-betw-def inj-on-def*
by *auto* (*metis image-iff*)+

lemma *permutes-inv-o*:
assumes *pS*: *p permutes S*
shows *p* $\circ$ *inv p = id*
and *inv p* $\circ$ *p = id*
using *permutes-inj*[*OF pS*] *permutes-surj*[*OF pS*]
unfolding *inj-iff*[*symmetric*] *surj-iff*[*symmetric*] **by** *blast*+

lemma *permutes-inverses*:
fixes *p* :: *'a* $\Rightarrow$ *'a*
assumes *pS*: *p permutes S*
shows *p* (*inv p x*) = *x*
and *inv p* (*p x*) = *x*
using *permutes-inv-o*[*OF pS*, *unfolded fun-eq-iff o-def*] **by** *auto*

lemma *permutes-subset*: *p permutes S* $\Longrightarrow$ *S* $\subseteq$ *T* $\Longrightarrow$ *p permutes T*
unfolding *permutes-def* **by** *blast*

lemma *permutes-empty*[*simp*]: *p permutes* {} $\longleftrightarrow$ *p = id*
unfolding *fun-eq-iff permutes-def* **by** *simp metis*

lemma *permutes-sing*[*simp*]: *p permutes* {*a*} $\longleftrightarrow$ *p = id*
unfolding *fun-eq-iff permutes-def* **by** *simp metis*

lemma *permutes-univ*: *p permutes UNIV* $\longleftrightarrow$ ($\forall y.\ \exists!x.\ p\ x = y$)
unfolding *permutes-def* **by** *simp*

lemma *permutes-inv-eq*: *p permutes S* $\Longrightarrow$ *inv p y* = *x* $\longleftrightarrow$ *p x* = *y*
unfolding *permutes-def inv-def*
apply *auto*
apply (*erule allE*[**where** *x*=*y*])
apply (*erule allE*[**where** *x*=*y*])
apply (*rule someI-ex*)
apply *blast*
apply (*rule some1-equality*)
apply *blast*
apply *blast*
done

lemma *permutes-swap-id*: *a* $\in$ *S* $\Longrightarrow$ *b* $\in$ *S* $\Longrightarrow$ *Fun.swap a b id permutes S*
unfolding *permutes-def Fun.swap-def fun-upd-def* **by** *auto metis*

lemma *permutes-superset*: *p permutes S* $\Longrightarrow$ ($\forall$ *x* $\in$ *S* $-$ *T*. *p x* = *x*) $\Longrightarrow$ *p permutes T*
by (*simp add*: *Ball-def permutes-def*) *metis*

43.3 Group properties

lemma *permutes-id*: *id permutes S*
unfolding *permutes-def* **by** *simp*

lemma *permutes-compose*: *p permutes S* $\Longrightarrow$ *q permutes S* $\Longrightarrow$ *q* $\circ$ *p permutes S*
unfolding *permutes-def o-def* **by** *metis*

lemma *permutes-inv*:
assumes *pS*: *p permutes S*
shows *inv p permutes S*
using *pS* **unfolding** *permutes-def permutes-inv-eq*[*OF pS*] **by** *metis*

lemma *permutes-inv-inv*:
assumes *pS*: *p permutes S*
shows *inv* (*inv p*) = *p*
unfolding *fun-eq-iff permutes-inv-eq*[*OF pS*] *permutes-inv-eq*[*OF permutes-inv*[*OF pS*]]
by *blast*

43.4 The number of permutations on a finite set

lemma *permutes-insert-lemma*:
assumes *pS*: *p permutes* (*insert a S*)
shows *Fun.swap a* (*p a*) *id* $\circ$ *p permutes S*
apply (*rule permutes-superset*[**where** *S* = *insert a S*])
apply (*rule permutes-compose*[*OF pS*])
apply (*rule permutes-swap-id*, *simp*)
using *permutes-in-image*[*OF pS*, *of a*]
apply *simp*

```
  apply (auto simp add: Ball-def Fun.swap-def)
  done

lemma permutes-insert: {p. p permutes (insert a S)} =
  (λ(b,p). Fun.swap a b id ∘ p) ' {(b,p). b ∈ insert a S ∧ p ∈ {p. p permutes S}}
proof −
  {
    fix p
    {
      assume pS: p permutes insert a S
      let ?b = p a
      let ?q = Fun.swap a (p a) id ∘ p
      have th0: p = Fun.swap a ?b id ∘ ?q
        unfolding fun-eq-iff o-assoc by simp
      have th1: ?b ∈ insert a S
        unfolding permutes-in-image[OF pS] by simp
      from permutes-insert-lemma[OF pS] th0 th1
       have ∃ b q. p = Fun.swap a b id ∘ q ∧ b ∈ insert a S ∧ q permutes S by
blast
    }
    moreover
    {
      fix b q
      assume bq: p = Fun.swap a b id ∘ q b ∈ insert a S q permutes S
      from permutes-subset[OF bq(3), of insert a S]
      have qS: q permutes insert a S
        by auto
      have aS: a ∈ insert a S
        by simp
      from bq(1) permutes-compose[OF qS permutes-swap-id[OF aS bq(2)]]
      have p permutes insert a S
        by simp
    }
    ultimately have p permutes insert a S ⟷
        (∃ b q. p = Fun.swap a b id ∘ q ∧ b ∈ insert a S ∧ q permutes S)
      by blast
  }
  then show ?thesis
    by auto
qed

lemma card-permutations:
  assumes Sn: card S = n
    and fS: finite S
  shows card {p. p permutes S} = fact n
  using fS Sn
proof (induct arbitrary: n)
  case empty
  then show ?case by simp
```

```
next
  case (insert x F)
  {
    fix n
    assume H0: card (insert x F) = n
    let ?xF = {p. p permutes insert x F}
    let ?pF = {p. p permutes F}
    let ?pF' = {(b, p). b ∈ insert x F ∧ p ∈ ?pF}
    let ?g = (λ(b, p). Fun.swap x b id ∘ p)
    from permutes-insert[of x F]
    have xfgpF': ?xF = ?g ` ?pF' .
    have Fs: card F = n − 1
      using ‹x ∉ F› H0 ‹finite F› by auto
    from insert.hyps Fs have pFs: card ?pF = fact (n − 1)
      using ‹finite F› by auto
    then have finite ?pF
      by (auto intro: card-ge-0-finite)
    then have pF'f: finite ?pF'
      using H0 ‹finite F›
      apply (simp only: Collect-case-prod Collect-mem-eq)
      apply (rule finite-cartesian-product)
      apply simp-all
      done

    have ginj: inj-on ?g ?pF'
    proof −
      {
        fix b p c q
        assume bp: (b,p) ∈ ?pF'
        assume cq: (c,q) ∈ ?pF'
        assume eq: ?g (b,p) = ?g (c,q)
        from bp cq have ths: b ∈ insert x F c ∈ insert x F x ∈ insert x F
          p permutes F q permutes F
          by auto
        from ths(4) ‹x ∉ F› eq have b = ?g (b,p) x
          unfolding permutes-def
          by (auto simp add: Fun.swap-def fun-upd-def fun-eq-iff)
        also have ... = ?g (c,q) x
          using ths(5) ‹x ∉ F› eq
          by (auto simp add: swap-def fun-upd-def fun-eq-iff)
        also have ... = c
          using ths(5) ‹x ∉ F›
          unfolding permutes-def
          by (auto simp add: Fun.swap-def fun-upd-def fun-eq-iff)
        finally have bc: b = c .
        then have Fun.swap x b id = Fun.swap x c id
          by simp
        with eq have Fun.swap x b id ∘ p = Fun.swap x b id ∘ q
          by simp
```

```
      then have Fun.swap x b id ∘ (Fun.swap x b id ∘ p) =
        Fun.swap x b id ∘ (Fun.swap x b id ∘ q)
        by simp
      then have p = q
        by (simp add: o-assoc)
      with bc have (b, p) = (c, q)
        by simp
    }
    then show ?thesis
      unfolding inj-on-def by blast
  qed
  from ‹x ∉ F› H0 have n0: n ≠ 0
    using ‹finite F› by auto
  then have ∃ m. n = Suc m
    by presburger
  then obtain m where n[simp]: n = Suc m
    by blast
  from pFs H0 have xFc: card ?xF = fact n
    unfolding xfgpF' card-image[OF ginj]
    using ‹finite F› ‹finite ?pF›
    apply (simp only: Collect-case-prod Collect-mem-eq card-cartesian-product)
    apply simp
    done
  from finite-imageI[OF pF'f, of ?g] have xFf: finite ?xF
    unfolding xfgpF' by simp
  have card ?xF = fact n
    using xFf xFc unfolding xFf by blast
 }
 then show ?case
  using insert by simp
qed

lemma finite-permutations:
 assumes fS: finite S
 shows finite {p. p permutes S}
 using card-permutations[OF refl fS]
 by (auto intro: card-ge-0-finite)
```

43.5 Permutations of index set for iterated operations

```
lemma (in comm-monoid-set) permute:
 assumes p permutes S
 shows F g S = F (g ∘ p) S
proof −
 from ‹p permutes S› have inj p
  by (rule permutes-inj)
 then have inj-on p S
  by (auto intro: subset-inj-on)
 then have F g (p ‘ S) = F (g ∘ p) S
```

by (*rule reindex*)
moreover from ‹*p permutes S*› **have** *p ' S = S*
by (*rule permutes-image*)
ultimately show *?thesis*
by *simp*
qed

43.6 Various combinations of transpositions with 2, 1 and 0 common elements

lemma *swap-id-common*: *a ≠ c ⟹ b ≠ c ⟹*
Fun.swap a b id ∘ Fun.swap a c id = Fun.swap b c id ∘ Fun.swap a b id
by (*simp add*: *fun-eq-iff Fun.swap-def*)

lemma *swap-id-common'*: *a ≠ b ⟹ a ≠ c ⟹*
Fun.swap a c id ∘ Fun.swap b c id = Fun.swap b c id ∘ Fun.swap a b id
by (*simp add*: *fun-eq-iff Fun.swap-def*)

lemma *swap-id-independent*: *a ≠ c ⟹ a ≠ d ⟹ b ≠ c ⟹ b ≠ d ⟹*
Fun.swap a b id ∘ Fun.swap c d id = Fun.swap c d id ∘ Fun.swap a b id
by (*simp add*: *fun-eq-iff Fun.swap-def*)

43.7 Permutations as transposition sequences

inductive *swapidseq* :: *nat ⇒* (*'a ⇒ 'a*) *⇒ bool*
where
id[*simp*]: *swapidseq 0 id*
| *comp-Suc*: *swapidseq n p ⟹ a ≠ b ⟹ swapidseq* (*Suc n*) (*Fun.swap a b id ∘ p*)

declare *id*[*unfolded id-def*, *simp*]

definition *permutation p ⟷* (*∃ n. swapidseq n p*)

43.8 Some closure properties of the set of permutations, with lengths

lemma *permutation-id*[*simp*]: *permutation id*
unfolding *permutation-def* **by** (*rule exI*[**where** *x=0*]) *simp*

declare *permutation-id*[*unfolded id-def*, *simp*]

lemma *swapidseq-swap*: *swapidseq* (*if a = b then 0 else 1*) (*Fun.swap a b id*)
apply *clarsimp*
using *comp-Suc*[*of 0 id a b*]
apply *simp*
done

lemma *permutation-swap-id*: *permutation* (*Fun.swap a b id*)

apply (*cases a = b*)
apply *simp-all*
unfolding *permutation-def*
using *swapidseq-swap*[*of a b*]
apply *blast*
done

lemma *swapidseq-comp-add*: *swapidseq n p* ⟹ *swapidseq m q* ⟹ *swapidseq* (*n* + *m*) (*p* ∘ *q*)
proof (*induct n p arbitrary*: *m q rule*: *swapidseq.induct*)
case (*id m q*)
then show *?case* **by** *simp*
next
case (*comp-Suc n p a b m q*)
have *th*: *Suc n* + *m* = *Suc* (*n* + *m*)
by *arith*
show *?case*
unfolding *th comp-assoc*
apply (*rule swapidseq.comp-Suc*)
using *comp-Suc.hyps*(*2*)[*OF comp-Suc.prems*] *comp-Suc.hyps*(*3*)
apply *blast*+
done
qed

lemma *permutation-compose*: *permutation p* ⟹ *permutation q* ⟹ *permutation* (*p* ∘ *q*)
unfolding *permutation-def* **using** *swapidseq-comp-add*[*of - p - q*] **by** *metis*

lemma *swapidseq-endswap*: *swapidseq n p* ⟹ *a* ≠ *b* ⟹ *swapidseq* (*Suc n*) (*p* ∘ *Fun.swap a b id*)
apply (*induct n p rule*: *swapidseq.induct*)
using *swapidseq-swap*[*of a b*]
apply (*auto simp add*: *comp-assoc intro*: *swapidseq.comp-Suc*)
done

lemma *swapidseq-inverse-exists*: *swapidseq n p* ⟹ ∃ *q*. *swapidseq n q* ∧ *p* ∘ *q* = *id* ∧ *q* ∘ *p* = *id*
proof (*induct n p rule*: *swapidseq.induct*)
case *id*
then show *?case*
by (*rule exI*[**where** *x*=*id*]) *simp*
next
case (*comp-Suc n p a b*)
from *comp-Suc.hyps* **obtain** *q* **where** *q*: *swapidseq n q p* ∘ *q* = *id q* ∘ *p* = *id*
by *blast*
let *?q* = *q* ∘ *Fun.swap a b id*
note *H* = *comp-Suc.hyps*
from *swapidseq-swap*[*of a b*] *H*(*3*) **have** *th0*: *swapidseq 1* (*Fun.swap a b id*)
by *simp*

from *swapidseq-comp-add*[*OF q*(*1*) *th0*] **have** *th1*: *swapidseq* (*Suc n*) *?q*
by *simp*
have *Fun.swap a b id* ∘ *p* ∘ *?q* = *Fun.swap a b id* ∘ (*p* ∘ *q*) ∘ *Fun.swap a b id*
by (*simp add*: *o-assoc*)
also have ... = *id*
by (*simp add*: *q*(*2*))
finally have *th2*: *Fun.swap a b id* ∘ *p* ∘ *?q* = *id* **.**
have *?q* ∘ (*Fun.swap a b id* ∘ *p*) = *q* ∘ (*Fun.swap a b id* ∘ *Fun.swap a b id*) ∘ *p*
by (*simp only*: *o-assoc*)
then have *?q* ∘ (*Fun.swap a b id* ∘ *p*) = *id*
by (*simp add*: *q*(*3*))
with *th1 th2* **show** *?case*
by *blast*
qed

lemma *swapidseq-inverse*:
assumes *H*: *swapidseq n p*
shows *swapidseq n* (*inv p*)
using *swapidseq-inverse-exists*[*OF H*] *inv-unique-comp*[*of p*] **by** *auto*

lemma *permutation-inverse*: *permutation p* ⟹ *permutation* (*inv p*)
using *permutation-def swapidseq-inverse* **by** *blast*

43.9 The identity map only has even transposition sequences

lemma *symmetry-lemma*:
assumes ⋀*a b c d*. *P a b c d* ⟹ *P a b d c*
and ⋀*a b c d*. *a* ≠ *b* ⟹ *c* ≠ *d* ⟹
a = *c* ∧ *b* = *d* ∨ *a* = *c* ∧ *b* ≠ *d* ∨ *a* ≠ *c* ∧ *b* = *d* ∨ *a* ≠ *c* ∧ *a* ≠ *d* ∧ *b* ≠ *c* ∧ *b* ≠ *d* ⟹
P a b c d
shows ⋀*a b c d*. *a* ≠ *b* ⟶ *c* ≠ *d* ⟶ *P a b c d*
using *assms* **by** *metis*

lemma *swap-general*: *a* ≠ *b* ⟹ *c* ≠ *d* ⟹
Fun.swap a b id ∘ *Fun.swap c d id* = *id* ∨
(∃*x y z*. *x* ≠ *a* ∧ *y* ≠ *a* ∧ *z* ≠ *a* ∧ *x* ≠ *y* ∧
Fun.swap a b id ∘ *Fun.swap c d id* = *Fun.swap x y id* ∘ *Fun.swap a z id*)
proof −
assume *H*: *a* ≠ *b c* ≠ *d*
have *a* ≠ *b* ⟶ *c* ≠ *d* ⟶
(*Fun.swap a b id* ∘ *Fun.swap c d id* = *id* ∨
(∃*x y z*. *x* ≠ *a* ∧ *y* ≠ *a* ∧ *z* ≠ *a* ∧ *x* ≠ *y* ∧
Fun.swap a b id ∘ *Fun.swap c d id* = *Fun.swap x y id* ∘ *Fun.swap a z id*))
apply (*rule symmetry-lemma*[**where** *a*=*a* **and** *b*=*b* **and** *c*=*c* **and** *d*=*d*])
apply (*simp-all only*: *swap-commute*)
apply (*case-tac a* = *c* ∧ *b* = *d*)
apply (*clarsimp simp only*: *swap-commute swap-id-idempotent*)
apply (*case-tac a* = *c* ∧ *b* ≠ *d*)

apply (*rule disjI2*)
apply (*rule-tac x=b* **in** *exI*)
apply (*rule-tac x=d* **in** *exI*)
apply (*rule-tac x=b* **in** *exI*)
apply (*clarsimp simp add*: *fun-eq-iff Fun.swap-def*)
apply (*case-tac a ≠ c ∧ b = d*)
apply (*rule disjI2*)
apply (*rule-tac x=c* **in** *exI*)
apply (*rule-tac x=d* **in** *exI*)
apply (*rule-tac x=c* **in** *exI*)
apply (*clarsimp simp add*: *fun-eq-iff Fun.swap-def*)
apply (*rule disjI2*)
apply (*rule-tac x=c* **in** *exI*)
apply (*rule-tac x=d* **in** *exI*)
apply (*rule-tac x=b* **in** *exI*)
apply (*clarsimp simp add*: *fun-eq-iff Fun.swap-def*)
done
with *H* **show** *?thesis* **by** *metis*
qed

lemma *swapidseq-id-iff*[*simp*]: *swapidseq 0 p* ⟷ *p = id*
using *swapidseq.cases*[*of 0 p p = id*]
by *auto*

lemma *swapidseq-cases*: *swapidseq n p* ⟷
n = 0 ∧ p = id ∨ (∃ *a b q m. n = Suc m ∧ p = Fun.swap a b id ∘ q ∧ swapidseq m q ∧ a ≠ b*)
apply (*rule iffI*)
apply (*erule swapidseq.cases*[*of n p*])
apply *simp*
apply (*rule disjI2*)
apply (*rule-tac x= a* **in** *exI*)
apply (*rule-tac x= b* **in** *exI*)
apply (*rule-tac x= pa* **in** *exI*)
apply (*rule-tac x= na* **in** *exI*)
apply *simp*
apply *auto*
apply (*rule comp-Suc*, *simp-all*)
done

lemma *fixing-swapidseq-decrease*:
assumes *spn*: *swapidseq n p*
and *ab*: *a ≠ b*
and *pa*: (*Fun.swap a b id ∘ p*) *a = a*
shows *n ≠ 0 ∧ swapidseq* (*n − 1*) (*Fun.swap a b id ∘ p*)
using *spn ab pa*
proof (*induct n arbitrary*: *p a b*)
case *0*
then show *?case*

```
    by (auto simp add: Fun.swap-def fun-upd-def)
next
  case (Suc n p a b)
  from Suc.prems(1) swapidseq-cases[of Suc n p]
  obtain c d q m where
    cdqm: Suc n = Suc m p = Fun.swap c d id ∘ q swapidseq m q c ≠ d n = m
    by auto
  {
    assume H: Fun.swap a b id ∘ Fun.swap c d id = id
    have ?case by (simp only: cdqm o-assoc H) (simp add: cdqm)
  }
  moreover
  {
    fix x y z
    assume H: x ≠ a y ≠ a z ≠ a x ≠ y
      Fun.swap a b id ∘ Fun.swap c d id = Fun.swap x y id ∘ Fun.swap a z id
    from H have az: a ≠ z
      by simp

    {
      fix h
      have (Fun.swap x y id ∘ h) a = a ⟷ h a = a
        using H by (simp add: Fun.swap-def)
    }
    note th3 = this
    from cdqm(2) have Fun.swap a b id ∘ p = Fun.swap a b id ∘ (Fun.swap c d
id ∘ q)
      by simp
    then have Fun.swap a b id ∘ p = Fun.swap x y id ∘ (Fun.swap a z id ∘ q)
      by (simp add: o-assoc H)
    then have (Fun.swap a b id ∘ p) a = (Fun.swap x y id ∘ (Fun.swap a z id ∘
q)) a
      by simp
    then have (Fun.swap x y id ∘ (Fun.swap a z id ∘ q)) a = a
      unfolding Suc by metis
    then have th1: (Fun.swap a z id ∘ q) a = a
      unfolding th3 .
    from Suc.hyps[OF cdqm(3)[ unfolded cdqm(5)[symmetric]] az th1]
    have th2: swapidseq (n − 1) (Fun.swap a z id ∘ q) n ≠ 0
      by blast+
    have th: Suc n − 1 = Suc (n − 1)
      using th2(2) by auto
    have ?case
      unfolding cdqm(2) H o-assoc th
      apply (simp only: Suc-not-Zero simp-thms comp-assoc)
      apply (rule comp-Suc)
      using th2 H
      apply blast+
      done
```

```
  }
  ultimately show ?case
    using swap-general[OF Suc.prems(2) cdqm(4)] by metis
qed

lemma swapidseq-identity-even:
  assumes swapidseq n (id :: 'a ⇒ 'a)
  shows even n
  using ‹swapidseq n id›
proof (induct n rule: nat-less-induct)
  fix n
  assume H: ∀ m<n. swapidseq m (id::'a ⇒ 'a) ⟶ even m swapidseq n (id :: 'a
⇒ 'a)
  {
    assume n = 0
    then have even n by presburger
  }
  moreover
  {
    fix a b :: 'a and q m
    assume h: n = Suc m (id :: 'a ⇒ 'a) = Fun.swap a b id ∘ q swapidseq m q a
≠ b
    from fixing-swapidseq-decrease[OF h(3,4), unfolded h(2)[symmetric]]
    have m: m ≠ 0 swapidseq (m − 1) (id :: 'a ⇒ 'a)
      by auto
    from h m have mn: m − 1 < n
      by arith
    from H(1)[rule-format, OF mn m(2)] h(1) m(1) have even n
      by presburger
  }
  ultimately show even n
    using H(2)[unfolded swapidseq-cases[of n id]] by auto
qed
```

43.10 Therefore we have a welldefined notion of parity

```
definition evenperm p = even (SOME n. swapidseq n p)

lemma swapidseq-even-even:
  assumes m: swapidseq m p
    and n: swapidseq n p
  shows even m ⟷ even n
proof −
  from swapidseq-inverse-exists[OF n]
  obtain q where q: swapidseq n q p ∘ q = id q ∘ p = id
    by blast
  from swapidseq-identity-even[OF swapidseq-comp-add[OF m q(1), unfolded q]]
  show ?thesis
    by arith
```

qed

lemma *evenperm-unique*:
assumes *p*: *swapidseq n p*
and *n*:*even n* = *b*
shows *evenperm p* = *b*
unfolding *n*[*symmetric*] *evenperm-def*
apply (*rule swapidseq-even-even*[**where** *p* = *p*])
apply (*rule someI*[**where** *x* = *n*])
using *p*
apply *blast*+
done

43.11 And it has the expected composition properties

lemma *evenperm-id*[*simp*]: *evenperm id* = *True*
by (*rule evenperm-unique*[**where** *n* = *0*]) *simp-all*

lemma *evenperm-swap*: *evenperm* (*Fun.swap a b id*) = (*a* = *b*)
by (*rule evenperm-unique*[**where** *n*=*if a* = *b then 0 else 1*]) (*simp-all add*: *swapidseq-swap*)

lemma *evenperm-comp*:
assumes *p*: *permutation p*
and *q*:*permutation q*
shows *evenperm* (*p* ∘ *q*) = (*evenperm p* = *evenperm q*)
proof −
from *p q* **obtain** *n m* **where** *n*: *swapidseq n p* **and** *m*: *swapidseq m q*
unfolding *permutation-def* **by** *blast*
note *nm* = *swapidseq-comp-add*[*OF n m*]
have *th*: *even* (*n* + *m*) = (*even n* ⟷ *even m*)
by *arith*
from *evenperm-unique*[*OF n refl*] *evenperm-unique*[*OF m refl*]
evenperm-unique[*OF nm th*]
show *?thesis*
by *blast*
qed

lemma *evenperm-inv*:
assumes *p*: *permutation p*
shows *evenperm* (*inv p*) = *evenperm p*
proof −
from *p* **obtain** *n* **where** *n*: *swapidseq n p*
unfolding *permutation-def* **by** *blast*
from *evenperm-unique*[*OF swapidseq-inverse*[*OF n*] *evenperm-unique*[*OF n refl*, *symmetric*]]
show *?thesis* **.**
qed

43.12 A more abstract characterization of permutations

lemma *bij-iff*: *bij f* $\longleftrightarrow$ $(\forall x.\ \exists!y.\ f\ y = x)$
unfolding *bij-def inj-on-def surj-def*
apply *auto*
apply *metis*
apply *metis*
done

lemma *permutation-bijective*:
assumes *p*: *permutation p*
shows *bij p*
proof −
from *p* **obtain** *n* **where** *n*: *swapidseq n p*
unfolding *permutation-def* **by** *blast*
from *swapidseq-inverse-exists*[*OF n*]
obtain *q* **where** *q*: *swapidseq n q p* ∘ *q* = *id q* ∘ *p* = *id*
by *blast*
then show *?thesis* **unfolding** *bij-iff*
apply (*auto simp add*: *fun-eq-iff*)
apply *metis*
done
qed

lemma *permutation-finite-support*:
assumes *p*: *permutation p*
shows *finite* {*x*. *p x* ≠ *x*}
proof −
from *p* **obtain** *n* **where** *n*: *swapidseq n p*
unfolding *permutation-def* **by** *blast*
from *n* **show** *?thesis*
proof (*induct n p rule*: *swapidseq.induct*)
case *id*
then show *?case* **by** *simp*
next
case (*comp-Suc n p a b*)
let *?S* = *insert a* (*insert b* {*x*. *p x* ≠ *x*})
from *comp-Suc.hyps*(*2*) **have** *fS*: *finite ?S*
by *simp*
from ‹*a* ≠ *b*› **have** *th*: {*x*. (*Fun.swap a b id* ∘ *p*) *x* ≠ *x*} ⊆ *?S*
by (*auto simp add*: *Fun.swap-def*)
from *finite-subset*[*OF th fS*] **show** *?case* **.**
qed
qed

lemma *bij-inv-eq-iff*: *bij p* ⟹ *x* = *inv p y* $\longleftrightarrow$ *p x* = *y*
using *surj-f-inv-f*[*of p*] **by** (*auto simp add*: *bij-def*)

lemma *bij-swap-comp*:
assumes *bp*: *bij p*

```
  shows Fun.swap a b id ∘ p = Fun.swap (inv p a) (inv p b) p
  using surj-f-inv-f[OF bij-is-surj[OF bp]]
  by (simp add: fun-eq-iff Fun.swap-def bij-inv-eq-iff[OF bp])

lemma bij-swap-ompose-bij: bij p ⟹ bij (Fun.swap a b id ∘ p)
proof −
  assume H: bij p
  show ?thesis
    unfolding bij-swap-comp[OF H] bij-swap-iff
    using H .
qed

lemma permutation-lemma:
  assumes fS: finite S
    and p: bij p
    and pS: ∀ x. x∉ S ⟶ p x = x
  shows permutation p
  using fS p pS
proof (induct S arbitrary: p rule: finite-induct)
  case (empty p)
  then show ?case by simp
next
  case (insert a F p)
  let ?r = Fun.swap a (p a) id ∘ p
  let ?q = Fun.swap a (p a) id ∘ ?r
  have raa: ?r a = a
    by (simp add: Fun.swap-def)
  from bij-swap-ompose-bij[OF insert(4)]
  have br: bij ?r  .

  from insert raa have th: ∀ x. x ∉ F ⟶ ?r x = x
    apply (clarsimp simp add: Fun.swap-def)
    apply (erule-tac x=x in allE)
    apply auto
    unfolding bij-iff
    apply metis
    done
  from insert(3)[OF br th]
  have rp: permutation ?r .
  have permutation ?q
    by (simp add: permutation-compose permutation-swap-id rp)
  then show ?case
    by (simp add: o-assoc)
qed

lemma permutation: permutation p ⟷ bij p ∧ finite {x. p x ≠ x}
  (is ?lhs ⟷ ?b ∧ ?f)
proof
  assume p: ?lhs
```

from *p permutation-bijective permutation-finite-support* **show** *?b ∧ ?f*
by *auto*
next
assume *?b ∧ ?f*
then have *?f ?b* **by** *blast+*
from *permutation-lemma*[*OF this*] **show** *?lhs*
by *blast*
qed

lemma *permutation-inverse-works*:
assumes *p*: *permutation p*
shows *inv p ∘ p = id*
and *p ∘ inv p = id*
using *permutation-bijective* [*OF p*]
unfolding *bij-def inj-iff surj-iff* **by** *auto*

lemma *permutation-inverse-compose*:
assumes *p*: *permutation p*
and *q*: *permutation q*
shows *inv* (*p ∘ q*) = *inv q ∘ inv p*
proof −
note *ps* = *permutation-inverse-works*[*OF p*]
note *qs* = *permutation-inverse-works*[*OF q*]
have *p ∘ q ∘* (*inv q ∘ inv p*) = *p ∘* (*q ∘ inv q*) *∘ inv p*
by (*simp add*: *o-assoc*)
also have *...* = *id*
by (*simp add*: *ps qs*)
finally have *th0*: *p ∘ q ∘* (*inv q ∘ inv p*) = *id* **.**
have *inv q ∘ inv p ∘* (*p ∘ q*) = *inv q ∘* (*inv p ∘ p*) *∘ q*
by (*simp add*: *o-assoc*)
also have *...* = *id*
by (*simp add*: *ps qs*)
finally have *th1*: *inv q ∘ inv p ∘* (*p ∘ q*) = *id* **.**
from *inv-unique-comp*[*OF th0 th1*] **show** *?thesis* **.**
qed

43.13 Relation to "permutes"

lemma *permutation-permutes*: *permutation p* ⟷ (∃ *S*. *finite S ∧ p permutes S*)
unfolding *permutation permutes-def bij-iff* [*symmetric*]
apply (*rule iffI*, *clarify*)
apply (*rule exI*[**where** *x*={*x*. *p x* ≠ *x*}])
apply *simp*
apply *clarsimp*
apply (*rule-tac B*=*S* **in** *finite-subset*)
apply *auto*
done

43.14 Hence a sort of induction principle composing by swaps

lemma *permutes-induct*: *finite S* $\Longrightarrow$ *P id* $\Longrightarrow$
(⋀ *a b p*. *a* ∈ *S* $\Longrightarrow$ *b* ∈ *S* $\Longrightarrow$ *P p* $\Longrightarrow$ *P p* $\Longrightarrow$ *permutation p* $\Longrightarrow$ *P* (*Fun.swap a b id* ∘ *p*)) $\Longrightarrow$
(⋀*p*. *p permutes S* $\Longrightarrow$ *P p*)
proof (*induct S rule*: *finite-induct*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x F p*)
let *?r* = *Fun.swap x* (*p x*) *id* ∘ *p*
let *?q* = *Fun.swap x* (*p x*) *id* ∘ *?r*
have *qp*: *?q* = *p*
by (*simp add*: *o-assoc*)
from *permutes-insert-lemma*[*OF insert.prems*(*3*)] *insert* **have** *Pr*: *P ?r*
by *blast*
from *permutes-in-image*[*OF insert.prems*(*3*), *of x*]
have *pxF*: *p x* ∈ *insert x F*
by *simp*
have *xF*: *x* ∈ *insert x F*
by *simp*
have *rp*: *permutation ?r*
unfolding *permutation-permutes* **using** *insert.hyps*(*1*)
permutes-insert-lemma[*OF insert.prems*(*3*)]
by *blast*
from *insert.prems*(*2*)[*OF xF pxF Pr Pr rp*]
show *?case*
unfolding *qp* **.**
qed

43.15 Sign of a permutation as a real number

definition *sign p* = (*if evenperm p then* (*1*::*int*) *else* −*1*)

lemma *sign-nz*: *sign p* ≠ *0*
by (*simp add*: *sign-def*)

lemma *sign-id*: *sign id* = *1*
by (*simp add*: *sign-def*)

lemma *sign-inverse*: *permutation p* $\Longrightarrow$ *sign* (*inv p*) = *sign p*
by (*simp add*: *sign-def evenperm-inv*)

lemma *sign-compose*: *permutation p* $\Longrightarrow$ *permutation q* $\Longrightarrow$ *sign* (*p* ∘ *q*) = *sign p* ∗ *sign q*
by (*simp add*: *sign-def evenperm-comp*)

lemma *sign-swap-id*: *sign* (*Fun.swap a b id*) = (*if a* = *b then 1 else* −*1*)
by (*simp add*: *sign-def evenperm-swap*)

lemma *sign-idempotent*: *sign p * sign p = 1*
by (*simp add*: *sign-def*)

43.16 More lemmas about permutations

lemma *permutes-natset-le*:
fixes *S* :: *'a*::*wellorder set*
assumes *p*: *p permutes S*
and *le*: $\forall i \in S.\ p\ i \leq i$
shows *p = id*
proof −
{
fix *n*
have *p n = n*
using *p le*
proof (*induct n arbitrary*: *S rule*: *less-induct*)
fix *n S*
assume *H*:
$\bigwedge m\ S.\ m < n \Longrightarrow p$ *permutes* $S \Longrightarrow \forall i{\in}S.\ p\ i \leq i \Longrightarrow p\ m = m$
p permutes S $\forall i \in S.\ p\ i \leq i$
{
assume $n \notin S$
with *H*(*2*) **have** *p n = n*
unfolding *permutes-def* **by** *metis*
}
moreover
{
assume *ns*: $n \in S$
from *H*(*3*) *ns* **have** $p\ n < n \vee p\ n = n$
by *auto*
moreover {
assume *h*: $p\ n < n$
from *H h* **have** *p* (*p n*) = *p n*
by *metis*
with *permutes-inj*[*OF H*(*2*)] **have** *p n = n*
unfolding *inj-on-def* **by** *blast*
with *h* **have** *False*
by *simp*
}
ultimately have *p n = n*
by *blast*
}
ultimately show *p n = n*
by *blast*
qed
}
then show *?thesis*
by (*auto simp add*: *fun-eq-iff*)

qed

lemma *permutes-natset-ge*:
fixes S :: *'a::wellorder set*
assumes p: p *permutes* S
and *le*: $\forall i \in S.\ p\ i \geq i$
shows $p = id$
proof −
{
fix i
assume i: $i \in S$
from i *permutes-in-image*[*OF permutes-inv*[*OF* p]] **have** *inv* $p\ i \in S$
by *simp*
with *le* **have** $p\ (inv\ p\ i) \geq inv\ p\ i$
by *blast*
with *permutes-inverses*[*OF* p] **have** $i \geq inv\ p\ i$
by *simp*
}
then have *th*: $\forall i{\in}S.\ inv\ p\ i \leq i$
by *blast*
from *permutes-natset-le*[*OF permutes-inv*[*OF* p] *th*]
have *inv* p = *inv id*
by *simp*
then show *?thesis*
apply (*subst permutes-inv-inv*[*OF* p, *symmetric*])
apply (*rule inv-unique-comp*)
apply *simp-all*
done
qed

lemma *image-inverse-permutations*: {*inv* p |p. p *permutes* S} = {p. p *permutes* S}
apply (*rule set-eqI*)
apply *auto*
using *permutes-inv-inv permutes-inv*
apply *auto*
apply (*rule-tac* x=*inv* x **in** *exI*)
apply *auto*
done

lemma *image-compose-permutations-left*:
assumes q: q *permutes* S
shows {$q \circ p$ | p. p *permutes* S} = {p . p *permutes* S}
apply (*rule set-eqI*)
apply *auto*
apply (*rule permutes-compose*)
using q
apply *auto*
apply (*rule-tac* x = *inv* $q \circ x$ **in** *exI*)

apply (*simp add*: *o-assoc permutes-inv permutes-compose permutes-inv-o*)
done

lemma *image-compose-permutations-right*:
assumes *q*: *q permutes S*
shows $\{p \circ q \mid p.\ p \text{ permutes } S\} = \{p\ .\ p \text{ permutes } S\}$
apply (*rule set-eqI*)
apply *auto*
apply (*rule permutes-compose*)
using *q*
apply *auto*
apply (*rule-tac x* = *x* ∘ *inv q* **in** *exI*)
apply (*simp add*: *o-assoc permutes-inv permutes-compose permutes-inv-o comp-assoc*)
done

lemma *permutes-in-seg*: *p permutes* $\{1\ ..n\} \Longrightarrow i \in \{1..n\} \Longrightarrow 1 \leq p\ i \wedge p\ i \leq n$
by (*simp add*: *permutes-def*) *metis*

lemma *setsum-permutations-inverse*:
setsum f $\{p.\ p \text{ permutes } S\}$ = *setsum* $(\lambda p.\ f(inv\ p))$ $\{p.\ p \text{ permutes } S\}$
(**is** *?lhs* = *?rhs*)
proof −
let *?S* = $\{p\ .\ p \text{ permutes } S\}$
have *th0*: *inj-on inv ?S*
proof (*auto simp add*: *inj-on-def*)
fix *q r*
assume *q*: *q permutes S*
and *r*: *r permutes S*
and *qr*: *inv q* = *inv r*
then have *inv* (*inv q*) = *inv* (*inv r*)
by *simp*
with *permutes-inv-inv*[*OF q*] *permutes-inv-inv*[*OF r*] **show** *q* = *r*
by *metis*
qed
have *th1*: *inv* ‘ *?S* = *?S*
using *image-inverse-permutations* **by** *blast*
have *th2*: *?rhs* = *setsum* (*f* ∘ *inv*) *?S*
by (*simp add*: *o-def*)
from *setsum.reindex*[*OF th0*, *of f*] **show** *?thesis* **unfolding** *th1 th2* **.**
qed

lemma *setum-permutations-compose-left*:
assumes *q*: *q permutes S*
shows *setsum f* $\{p.\ p \text{ permutes } S\}$ = *setsum* $(\lambda p.\ f(q \circ p))$ $\{p.\ p \text{ permutes } S\}$
(**is** *?lhs* = *?rhs*)
proof −
let *?S* = $\{p.\ p \text{ permutes } S\}$
have *th0*: *?rhs* = *setsum* (*f* ∘ (*op* ∘ *q*)) *?S*

```
    by (simp add: o-def)
  have th1: inj-on (op ∘ q) ?S
  proof (auto simp add: inj-on-def)
    fix p r
    assume p permutes S
      and r: r permutes S
      and rp: q ∘ p = q ∘ r
    then have inv q ∘ q ∘ p = inv q ∘ q ∘ r
      by (simp add: comp-assoc)
    with permutes-inj[OF q, unfolded inj-iff] show p = r
      by simp
  qed
  have th3: (op ∘ q) ' ?S = ?S
    using image-compose-permutations-left[OF q] by auto
  from setsum.reindex[OF th1, of f] show ?thesis unfolding th0 th1 th3 .
qed

lemma sum-permutations-compose-right:
  assumes q: q permutes S
  shows setsum f {p. p permutes S} = setsum (λp. f(p ∘ q)) {p. p permutes S}
  (is ?lhs = ?rhs)
proof −
  let ?S = {p. p permutes S}
  have th0: ?rhs = setsum (f ∘ (λp. p ∘ q)) ?S
    by (simp add: o-def)
  have th1: inj-on (λp. p ∘ q) ?S
  proof (auto simp add: inj-on-def)
    fix p r
    assume p permutes S
      and r: r permutes S
      and rp: p ∘ q = r ∘ q
    then have p ∘ (q ∘ inv q) = r ∘ (q ∘ inv q)
      by (simp add: o-assoc)
    with permutes-surj[OF q, unfolded surj-iff] show p = r
      by simp
  qed
  have th3: (λp. p ∘ q) ' ?S = ?S
    using image-compose-permutations-right[OF q] by auto
  from setsum.reindex[OF th1, of f]
  show ?thesis unfolding th0 th1 th3 .
qed
```

43.17 Sum over a set of permutations (could generalize to iteration)

```
lemma setsum-over-permutations-insert:
  assumes fS: finite S
    and aS: a ∉ S
  shows setsum f {p. p permutes (insert a S)} =
```

```
    setsum (λb. setsum (λq. f (Fun.swap a b id ∘ q)) {p. p permutes S}) (insert a
S)
proof −
  have th0: ⋀f a b. (λ(b,p). f (Fun.swap a b id ∘ p)) = f ∘ (λ(b,p). Fun.swap a
b id ∘ p)
    by (simp add: fun-eq-iff)
  have th1: ⋀P Q. P × Q = {(a,b). a ∈ P ∧ b ∈ Q}
    by blast
  have th2: ⋀P Q. P ⟹ (P ⟹ Q) ⟹ P ∧ Q
    by blast
  show ?thesis
    unfolding permutes-insert
    unfolding setsum.cartesian-product
    unfolding th1[symmetric]
    unfolding th0
  proof (rule setsum.reindex)
    let ?f = (λ(b, y). Fun.swap a b id ∘ y)
    let ?P = {p. p permutes S}
    {
      fix b c p q
      assume b: b ∈ insert a S
      assume c: c ∈ insert a S
      assume p: p permutes S
      assume q: q permutes S
      assume eq: Fun.swap a b id ∘ p = Fun.swap a c id ∘ q
      from p q aS have pa: p a = a and qa: q a = a
        unfolding permutes-def by metis+
      from eq have (Fun.swap a b id ∘ p) a  = (Fun.swap a c id ∘ q) a
        by simp
      then have bc: b = c
        by (simp add: permutes-def pa qa o-def fun-upd-def Fun.swap-def id-def
            cong del: if-weak-cong split: split-if-asm)
      from eq[unfolded bc] have (λp. Fun.swap a c id ∘ p) (Fun.swap a c id ∘ p)
=
        (λp. Fun.swap a c id ∘ p) (Fun.swap a c id ∘ q) by simp
      then have p = q
        unfolding o-assoc swap-id-idempotent
        by (simp add: o-def)
      with bc have b = c ∧ p = q
        by blast
    }
    then show inj-on ?f (insert a S × ?P)
      unfolding inj-on-def by clarify metis
  qed
qed

end
```

44 Traces, Determinant of square matrices and some properties

theory *Determinants*
imports
Cartesian-Euclidean-Space
~~/src/HOL/Library/Permutations
begin

44.1 Trace

definition *trace* :: *'a::semiring-1^'n^'n ⇒ 'a*
where *trace A = setsum (λi. ((A$i)$i)) (UNIV::'n set)*

lemma *trace-0*: *trace (mat 0) = 0*
by (*simp add*: *trace-def mat-def*)

lemma *trace-I*: *trace (mat 1 :: 'a::semiring-1^'n^'n) = of-nat(CARD('n))*
by (*simp add*: *trace-def mat-def*)

lemma *trace-add*: *trace ((A::'a::comm-semiring-1^'n^'n) + B) = trace A + trace B*
by (*simp add*: *trace-def setsum.distrib*)

lemma *trace-sub*: *trace ((A::'a::comm-ring-1^'n^'n) − B) = trace A − trace B*
by (*simp add*: *trace-def setsum-subtractf*)

lemma *trace-mul-sym*: *trace ((A::'a::comm-semiring-1^'n^'m) ** B) = trace (B**A)*
apply (*simp add*: *trace-def matrix-matrix-mult-def*)
apply (*subst setsum.commute*)
apply (*simp add*: *mult.commute*)
done

Definition of determinant.

definition *det*:: *'a::comm-ring-1^'n^'n ⇒ 'a* **where**
det A =
*setsum (λp. of-int (sign p) * setprod (λi. Aip i) (UNIV :: 'n set))*
{p. p permutes (UNIV :: 'n set)}

A few general lemmas we need below.

lemma *setprod-permute*:
assumes *p*: *p permutes S*
shows *setprod f S = setprod (f ∘ p) S*
using *assms* **by** (*fact setprod.permute*)

lemma *setproduct-permute-nat-interval*:
fixes *m n* :: *nat*
shows *p permutes {m..n} ⟹ setprod f {m..n} = setprod (f ∘ p) {m..n}*
by (*blast intro*!: *setprod-permute*)

Basic determinant properties.

lemma *det-transpose*: *det* (*transpose A*) = *det* (*A*::*'a*::*comm-ring-1* ^ *'n* ^ *'n*)
proof −
let *?di* = λ*A i j*. *A*$*i*$*j*
let *?U* = (*UNIV* :: *'n set*)
have *fU*: *finite ?U* **by** *simp*
{
fix *p*
assume *p*: *p* ∈ {*p*. *p permutes ?U*}
from *p* **have** *pU*: *p permutes ?U*
by *blast*
have *sth*: *sign* (*inv p*) = *sign p*
by (*metis sign-inverse fU p mem-Collect-eq permutation-permutes*)
from *permutes-inj*[*OF pU*]
have *pi*: *inj-on p ?U*
by (*blast intro*: *subset-inj-on*)
from *permutes-image*[*OF pU*]
have *setprod* (λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) *?U* =
setprod (λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) (*p* ' *?U*)
by *simp*
also have ... = *setprod* ((λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) ∘ *p*) *?U*
unfolding *setprod.reindex*[*OF pi*] **..**
also have ... = *setprod* (λ*i*. *?di A i* (*p i*)) *?U*
proof −
{
fix *i*
assume *i*: *i* ∈ *?U*
from *i permutes-inv-o*[*OF pU*] *permutes-in-image*[*OF pU*]
have ((λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) ∘ *p*) *i* = *?di A i* (*p i*)
unfolding *transpose-def* **by** (*simp add*: *fun-eq-iff*)
}
then show *setprod* ((λ*i*. *?di* (*transpose A*) *i* (*inv p i*)) ∘ *p*) *?U* =
setprod (λ*i*. *?di A i* (*p i*)) *?U*
by (*auto intro*: *setprod.cong*)
qed
finally have *of-int* (*sign* (*inv p*)) ∗ (*setprod* (λ*i*. *?di* (*transpose A*) *i* (*inv p i*))
?U) =
of-int (*sign p*) ∗ (*setprod* (λ*i*. *?di A i* (*p i*)) *?U*)
using *sth* **by** *simp*
}
then show *?thesis*
unfolding *det-def*
apply (*subst setsum-permutations-inverse*)
apply (*rule setsum.cong*)
apply (*rule refl*)
apply *blast*
done
qed

lemma *det-lowerdiagonal*:
fixes $A :: 'a::comm\text{-}ring\text{-}1\hat{}('n::\{finite,wellorder\})\hat{}('n::\{finite,wellorder\})$
assumes *ld*: $\bigwedge i\ j.\ i < j \Longrightarrow A\$i\$j = 0$
shows $det\ A = setprod\ (\lambda i.\ A\$i\$i)\ (UNIV :: 'n\ set)$
proof −
let $?U = UNIV :: 'n\ set$
let $?PU = \{p.\ p\ permutes\ ?U\}$
let $?pp = \lambda p.\ of\text{-}int\ (sign\ p) * setprod\ (\lambda i.\ A\$i\$p\ i)\ (UNIV :: 'n\ set)$
have *fU*: *finite ?U*
by *simp*
from *finite-permutations*[*OF fU*] **have** *fPU*: *finite ?PU* **.**
have *id0*: $\{id\} \subseteq ?PU$
by (*auto simp add*: *permutes-id*)
{
fix *p*
assume *p*: $p \in ?PU - \{id\}$
from *p* **have** *pU*: *p permutes ?U* **and** *pid*: $p \neq id$
by *blast*+
from *permutes-natset-le*[*OF pU*] *pid* **obtain** *i* **where** *i*: $p\ i > i$
by (*metis not-le*)
from *ld*[*OF i*] **have** *ex*:$\exists\, i \in ?U.\ A\$i\$p\ i = 0$
by *blast*
from *setprod-zero*[*OF fU ex*] **have** $?pp\ p = 0$
by *simp*
}
then have *p0*: $\forall\, p \in ?PU - \{id\}.\ ?pp\ p = 0$
by *blast*
from *setsum.mono-neutral-cong-left*[*OF fPU id0 p0*] **show** *?thesis*
unfolding *det-def* **by** (*simp add*: *sign-id*)
qed

lemma *det-upperdiagonal*:
fixes $A :: 'a::comm\text{-}ring\text{-}1\hat{}'n::\{finite,wellorder\}\hat{}'n::\{finite,wellorder\}$
assumes *ld*: $\bigwedge i\ j.\ i > j \Longrightarrow A\$i\$j = 0$
shows $det\ A = setprod\ (\lambda i.\ A\$i\$i)\ (UNIV :: 'n\ set)$
proof −
let $?U = UNIV :: 'n\ set$
let $?PU = \{p.\ p\ permutes\ ?U\}$
let $?pp = (\lambda p.\ of\text{-}int\ (sign\ p) * setprod\ (\lambda i.\ A\$i\$p\ i)\ (UNIV :: 'n\ set))$
have *fU*: *finite ?U*
by *simp*
from *finite-permutations*[*OF fU*] **have** *fPU*: *finite ?PU* **.**
have *id0*: $\{id\} \subseteq ?PU$
by (*auto simp add*: *permutes-id*)
{
fix *p*
assume *p*: $p \in ?PU - \{id\}$
from *p* **have** *pU*: *p permutes ?U* **and** *pid*: $p \neq id$
by *blast*+

```
    from permutes-natset-ge[OF pU] pid obtain i where i: p i < i
      by (metis not-le)
    from ld[OF i] have ex:∃ i ∈ ?U. A$i$p i = 0
      by blast
    from setprod-zero[OF fU ex] have ?pp p = 0
      by simp
  }
  then have p0: ∀ p ∈ ?PU −{id}. ?pp p = 0
    by blast
  from setsum.mono-neutral-cong-left[OF fPU id0 p0] show ?thesis
    unfolding det-def by (simp add: sign-id)
qed

lemma det-diagonal:
  fixes A :: 'a::comm-ring-1^'n^'n
  assumes ld: ⋀i j. i ≠ j ⟹ A$i$j = 0
  shows det A = setprod (λi. A$i$i) (UNIV::'n set)
proof −
  let ?U = UNIV:: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = λp. of-int (sign p) * setprod (λi. A$i$p i) (UNIV :: 'n set)
  have fU: finite ?U by simp
  from finite-permutations[OF fU] have fPU: finite ?PU .
  have id0: {id} ⊆ ?PU
    by (auto simp add: permutes-id)
  {
    fix p
    assume p: p ∈ ?PU − {id}
    then have p ≠ id
      by simp
    then obtain i where i: p i ≠ i
      unfolding fun-eq-iff by auto
    from ld [OF i [symmetric]] have ex:∃ i ∈ ?U. A$i$p i = 0
      by blast
    from setprod-zero [OF fU ex] have ?pp p = 0
      by simp
  }
  then have p0: ∀ p ∈ ?PU − {id}. ?pp p = 0
    by blast
  from setsum.mono-neutral-cong-left[OF fPU id0 p0] show ?thesis
    unfolding det-def by (simp add: sign-id)
qed

lemma det-I: det (mat 1 :: 'a::comm-ring-1^'n^'n) = 1
proof −
  let ?A = mat 1 :: 'a::comm-ring-1^'n^'n
  let ?U = UNIV :: 'n set
  let ?f = λi j. ?A$i$j
  {
```

```
    fix i
    assume i: i ∈ ?U
    have ?f i i = 1
      using i by (vector mat-def)
  }
  then have th: setprod (λi. ?f i i) ?U = setprod (λx. 1) ?U
    by (auto intro: setprod.cong)
  {
    fix i j
    assume i: i ∈ ?U and j: j ∈ ?U and ij: i ≠ j
    have ?f i j = 0 using i j ij
      by (vector mat-def)
  }
  then have det ?A = setprod (λi. ?f i i) ?U
    using det-diagonal by blast
  also have ... = 1
    unfolding th setprod.neutral-const ..
  finally show ?thesis .
qed

lemma det-0: det (mat 0 :: 'a::comm-ring-1^'n^'n) = 0
  by (simp add: det-def setprod-zero)

lemma det-permute-rows:
  fixes A :: 'a::comm-ring-1^'n^'n
  assumes p: p permutes (UNIV :: 'n::finite set)
  shows det (χ i. A$p i :: 'a^'n^'n) = of-int (sign p) * det A
  apply (simp add: det-def setsum-right-distrib mult.assoc[symmetric])
  apply (subst sum-permutations-compose-right[OF p])
proof (rule setsum.cong)
  let ?U = UNIV :: 'n set
  let ?PU = {p. p permutes ?U}
  fix q
  assume qPU: q ∈ ?PU
  have fU: finite ?U
    by simp
  from qPU have q: q permutes ?U
    by blast
  from p q have pp: permutation p and qp: permutation q
    by (metis fU permutation-permutes)+
  from permutes-inv[OF p] have ip: inv p permutes ?U .
  have setprod (λi. A$p i$ (q ∘ p) i) ?U = setprod ((λi. A$p i$(q ∘ p) i) ∘ inv
p) ?U
    by (simp only: setprod-permute[OF ip, symmetric])
  also have ... = setprod (λi. A $ (p ∘ inv p) i $ (q ∘ (p ∘ inv p)) i) ?U
    by (simp only: o-def)
  also have ... = setprod (λi. A$i$q i) ?U
    by (simp only: o-def permutes-inverses[OF p])
  finally have thp: setprod (λi. A$p i$ (q ∘ p) i) ?U = setprod (λi. A$i$q i) ?U
```

```
    by blast
  show of-int (sign (q ∘ p)) * setprod (λi. A$ p i$ (q ∘ p) i) ?U =
    of-int (sign p) * of-int (sign q) * setprod (λi. A$i$q i) ?U
    by (simp only: thp sign-compose[OF qp pp] mult.commute of-int-mult)
qed rule

lemma det-permute-columns:
  fixes A :: 'a::comm-ring-1^'n^'n
  assumes p: p permutes (UNIV :: 'n set)
  shows det(χ i j. A$i$ p j :: 'a^'n^'n) = of-int (sign p) * det A
proof −
  let ?Ap = χ i j. A$i$ p j :: 'a^'n^'n
  let ?At = transpose A
  have of-int (sign p) * det A = det (transpose (χ i. transpose A $ p i))
    unfolding det-permute-rows[OF p, of ?At] det-transpose ..
  moreover
  have ?Ap = transpose (χ i. transpose A $ p i)
    by (simp add: transpose-def vec-eq-iff)
  ultimately show ?thesis
    by simp
qed

lemma det-identical-rows:
  fixes A :: 'a::linordered-idom^'n^'n
  assumes ij: i ≠ j
    and r: row i A = row j A
  shows det A = 0
proof−
  have tha: ⋀(a::'a) b. a = b ⟹ b = − a ⟹ a = 0
    by simp
  have th1: of-int (−1) = − 1 by simp
  let ?p = Fun.swap i j id
  let ?A = χ i. A $ ?p i
  from r have A = ?A by (simp add: vec-eq-iff row-def Fun.swap-def)
  then have det A = det ?A by simp
  moreover have det A = − det ?A
    by (simp add: det-permute-rows[OF permutes-swap-id] sign-swap-id ij th1)
  ultimately show det A = 0 by (metis tha)
qed

lemma det-identical-columns:
  fixes A :: 'a::linordered-idom^'n^'n
  assumes ij: i ≠ j
    and r: column i A = column j A
  shows det A = 0
  apply (subst det-transpose[symmetric])
  apply (rule det-identical-rows[OF ij])
  apply (metis row-transpose r)
  done
```

lemma *det-zero-row*:
fixes *A* :: *'a*::{*idom*, *ring-char-0*}^*'n*^*'n*
assumes *r*: *row i A = 0*
shows *det A = 0*
using *r*
apply (*simp add*: *row-def det-def vec-eq-iff*)
apply (*rule setsum.neutral*)
apply (*auto simp*: *sign-nz*)
done

lemma *det-zero-column*:
fixes *A* :: *'a*::{*idom*,*ring-char-0*}^*'n*^*'n*
assumes *r*: *column i A = 0*
shows *det A = 0*
apply (*subst det-transpose*[*symmetric*])
apply (*rule det-zero-row* [*of i*])
apply (*metis row-transpose r*)
done

lemma *det-row-add*:
fixes *a b c* :: *'n*::*finite* ⇒ - ^ *'n*
shows *det*((χ *i*. *if i = k then a i + b i else c i*)::*'a*::*comm-ring-1*^*'n*^*'n*) =
det((χ *i*. *if i = k then a i else c i*)::*'a*::*comm-ring-1*^*'n*^*'n*) +
det((χ *i*. *if i = k then b i else c i*)::*'a*::*comm-ring-1*^*'n*^*'n*)
unfolding *det-def vec-lambda-beta setsum.distrib*[*symmetric*]
proof (*rule setsum.cong*)
let *?U = UNIV* :: *'n set*
let *?pU* = {*p*. *p permutes ?U*}
let *?f* = (λ*i*. *if i = k then a i + b i else c i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
let *?g* = (λ *i*. *if i = k then a i else c i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
let *?h* = (λ *i*. *if i = k then b i else c i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
fix *p*
assume *p*: *p* ∈ *?pU*
let *?Uk = ?U* − {*k*}
from *p* **have** *pU*: *p permutes ?U*
by *blast*
have *kU*: *?U = insert k ?Uk*
by *blast*
{
fix *j*
assume *j*: *j* ∈ *?Uk*
from *j* **have** *?f j* $ *p j = ?g j* $ *p j* **and** *?f j* $ *p j*= *?h j* $ *p j*
by *simp-all*
}
then have *th1*: *setprod* (λ*i*. *?f i* $ *p i*) *?Uk = setprod* (λ*i*. *?g i* $ *p i*) *?Uk*
and *th2*: *setprod* (λ*i*. *?f i* $ *p i*) *?Uk = setprod* (λ*i*. *?h i* $ *p i*) *?Uk*
apply −
apply (*rule setprod.cong*, *simp-all*)+

done
have *th3*: *finite ?Uk k* ∉ *?Uk*
by *auto*
have *setprod* (λ*i*. *?f i* $ *p i*) *?U* = *setprod* (λ*i*. *?f i* $ *p i*) (*insert k ?Uk*)
unfolding *kU*[*symmetric*] **..**
also have ... = *?f k* $ *p k* ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?Uk*
apply (*rule setprod.insert*)
apply *simp*
apply *blast*
done
also have ... = (*a k* $ *p k* ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?Uk*) + (*b k*$ *p k* ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?Uk*)
by (*simp add*: *field-simps*)
also have ... = (*a k* $ *p k* ∗ *setprod* (λ*i*. *?g i* $ *p i*) *?Uk*) + (*b k*$ *p k* ∗ *setprod* (λ*i*. *?h i* $ *p i*) *?Uk*)
by (*metis th1 th2*)
also have ... = *setprod* (λ*i*. *?g i* $ *p i*) (*insert k ?Uk*) + *setprod* (λ*i*. *?h i* $ *p i*) (*insert k ?Uk*)
unfolding *setprod.insert*[*OF th3*] **by** *simp*
finally have *setprod* (λ*i*. *?f i* $ *p i*) *?U* = *setprod* (λ*i*. *?g i* $ *p i*) *?U* + *setprod* (λ*i*. *?h i* $ *p i*) *?U*
unfolding *kU*[*symmetric*] **.**
then show *of-int* (*sign p*) ∗ *setprod* (λ*i*. *?f i* $ *p i*) *?U* =
of-int (*sign p*) ∗ *setprod* (λ*i*. *?g i* $ *p i*) *?U* + *of-int* (*sign p*) ∗ *setprod* (λ*i*. *?h i* $ *p i*) *?U*
by (*simp add*: *field-simps*)
qed *rule*

lemma *det-row-mul*:
fixes *a b* :: *'n*::*finite* ⇒ - ^ *'n*
shows *det*((χ *i*. *if i* = *k then c* ∗*s a i else b i*)::*'a*::*comm-ring-1*^*'n*^*'n*) =
c ∗ *det*((χ *i*. *if i* = *k then a i else b i*)::*'a*::*comm-ring-1*^*'n*^*'n*)
unfolding *det-def vec-lambda-beta setsum-right-distrib*
proof (*rule setsum.cong*)
let *?U* = *UNIV* :: *'n set*
let *?pU* = {*p*. *p permutes ?U*}
let *?f* = (λ*i*. *if i* = *k then c*∗*s a i else b i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
let *?g* = (λ *i*. *if i* = *k then a i else b i*)::*'n* ⇒ *'a*::*comm-ring-1*^*'n*
fix *p*
assume *p*: *p* ∈ *?pU*
let *?Uk* = *?U* − {*k*}
from *p* **have** *pU*: *p permutes ?U*
by *blast*
have *kU*: *?U* = *insert k ?Uk*
by *blast*
{
fix *j*
assume *j*: *j* ∈ *?Uk*
from *j* **have** *?f j* $ *p j* = *?g j* $ *p j*

```
    by simp
  }
  then have th1: setprod (λi. ?f i $ p i) ?Uk = setprod (λi. ?g i $ p i) ?Uk
    apply −
    apply (rule setprod.cong)
    apply simp-all
    done
  have th3: finite ?Uk k ∉ ?Uk
    by auto
  have setprod (λi. ?f i $ p i) ?U = setprod (λi. ?f i $ p i) (insert k ?Uk)
    unfolding kU[symmetric] ..
  also have ... = ?f k $ p k * setprod (λi. ?f i $ p i) ?Uk
    apply (rule setprod.insert)
    apply simp
    apply blast
    done
  also have ... = (c*s a k) $ p k * setprod (λi. ?f i $ p i) ?Uk
    by (simp add: field-simps)
  also have ... = c* (a k $ p k * setprod (λi. ?g i $ p i) ?Uk)
    unfolding th1 by (simp add: ac-simps)
  also have ... = c* (setprod (λi. ?g i $ p i) (insert k ?Uk))
    unfolding setprod.insert[OF th3] by simp
  finally have setprod (λi. ?f i $ p i) ?U = c* (setprod (λi. ?g i $ p i) ?U)
    unfolding kU[symmetric] .
  then show of-int (sign p) * setprod (λi. ?f i $ p i) ?U =
    c * (of-int (sign p) * setprod (λi. ?g i $ p i) ?U)
    by (simp add: field-simps)
qed rule

lemma det-row-0:
  fixes b :: 'n::finite ⇒ - ^ 'n
  shows det((χ i. if i = k then 0 else b i)::'a::comm-ring-1^'n^'n) = 0
  using det-row-mul[of k 0 λi. 1 b]
  apply simp
  apply (simp only: vector-smult-lzero)
  done

lemma det-row-operation:
  fixes A :: 'a::linordered-idom^'n^'n
  assumes ij: i ≠ j
  shows det (χ k. if k = i then row i A + c *s row j A else row k A) = det A
proof −
  let ?Z = (χ k. if k = i then row j A else row k A) :: 'a ^'n^'n
  have th: row i ?Z = row j ?Z by (vector row-def)
  have th2: ((χ k. if k = i then row i A else row k A) :: 'a^'n^'n) = A
    by (vector row-def)
  show ?thesis
    unfolding det-row-add [of i] det-row-mul[of i] det-identical-rows[OF ij th] th2
    by simp
```

qed

lemma *det-row-span*:
fixes A :: *real^'n^'n*
assumes x: $x \in span\ \{row\ j\ A\ |j.\ j \neq i\}$
shows $det\ (\chi\ k.\ if\ k = i\ then\ row\ i\ A + x\ else\ row\ k\ A) = det\ A$
proof −
let *?U* = *UNIV* :: *'n set*
let *?S* = $\{row\ j\ A\ |j.\ j \neq i\}$
let *?d* = $\lambda x.\ det\ (\chi\ k.\ if\ k = i\ then\ x\ else\ row\ k\ A)$
let *?P* = $\lambda x.\ ?d\ (row\ i\ A + x) = det\ A$
{
fix k
have $(if\ k = i\ then\ row\ i\ A + 0\ else\ row\ k\ A) = row\ k\ A$
by *simp*
}
then have *P0*: *?P 0*
apply −
apply (*rule cong*[*of det*, *OF refl*])
apply (*vector row-def*)
done
moreover
{
fix $c\ z\ y$
assume *zS*: $z \in ?S$ **and** *Py*: *?P y*
from *zS* **obtain** j **where** j: $z = row\ j\ A\ i \neq j$
by *blast*
let *?w* = $row\ i\ A + y$
have *th0*: $row\ i\ A + (c*s\ z + y) = ?w + c*s\ z$
by *vector*
have *thz*: *?d z = 0*
apply (*rule det-identical-rows*[*OF j*(*2*)])
using j
apply (*vector row-def*)
done
have $?d\ (row\ i\ A + (c*s\ z + y)) = ?d\ (?w + c*s\ z)$
unfolding *th0* **..**
then have $?P\ (c*s\ z + y)$
unfolding *thz Py det-row-mul*[*of i*] *det-row-add*[*of i*]
by *simp*
}
ultimately show *?thesis*
apply −
apply (*rule span-induct-alt*[*of ?P ?S*, *OF P0*, *folded scalar-mult-eq-scaleR*])
apply *blast*
apply (*rule x*)
done
qed

May as well do this, though it's a bit unsatisfactory since it ignores exact

duplicates by considering the rows/columns as a set.

lemma *det-dependent-rows*:
fixes *A*:: *real^'n^'n*
assumes *d*: *dependent* (*rows A*)
shows *det A = 0*
proof −
let *?U = UNIV* :: *'n set*
from *d* **obtain** *i* **where** *i*: *row i A* ∈ *span* (*rows A* − {*row i A*})
unfolding *dependent-def rows-def* **by** *blast*
{
fix *j k*
assume *jk*: *j* ≠ *k* **and** *c*: *row j A = row k A*
from *det-identical-rows*[*OF jk c*] **have** *?thesis* **.**
}
moreover
{
assume *H*: ⋀ *i j*. *i* ≠ *j* ⟹ *row i A* ≠ *row j A*
have *th0*: − *row i A* ∈ *span* {*row j A*|*j*. *j* ≠ *i*}
apply (*rule span-neg*)
apply (*rule set-rev-mp*)
apply (*rule i*)
apply (*rule span-mono*)
using *H i*
apply (*auto simp add*: *rows-def*)
done
from *det-row-span*[*OF th0*]
have *det A = det* (*χ k*. *if k = i then 0* ∗*s 1 else row k A*)
unfolding *right-minus vector-smult-lzero* **..**
with *det-row-mul*[*of i 0*::*real λi*. *1*]
have *det A = 0* **by** *simp*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *det-dependent-columns*:
assumes *d*: *dependent* (*columns* (*A*::*real^'n^'n*))
shows *det A = 0*
by (*metis d det-dependent-rows rows-transpose det-transpose*)

Multilinearity and the multiplication formula.

lemma *Cart-lambda-cong*: (⋀*x*. *f x = g x*) ⟹ (*vec-lambda f*::*'a^'n*) = (*vec-lambda g* :: *'a^'n*)
by (*rule iffD1*[*OF vec-lambda-unique*]) *vector*

lemma *det-linear-row-setsum*:
assumes *fS*: *finite S*
shows *det* ((*χ i*. *if i = k then setsum* (*a i*) *S else c i*)::*'a*::*comm-ring-1^'n^'n*)
=
setsum (*λj*. *det* ((*χ i*. *if i = k then a i j else c i*)::*'a^'n^'n*)) *S*

proof (*induct rule*: *finite-induct*[*OF fS*])
case *1*
then show *?case*
apply *simp*
unfolding *setsum.empty det-row-0*[*of k*]
apply *rule*
done
next
case (*2 x F*)
then show *?case*
by (*simp add*: *det-row-add cong del*: *if-weak-cong*)
qed

lemma *finite-bounded-functions*:
assumes *fS*: *finite S*
shows *finite* {*f*. (∀ *i* ∈ {*1*.. (*k*::*nat*)}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1* .. *k*} ⟶ *f i* = *i*)}
proof (*induct k*)
case *0*
have *th*: {*f*. ∀ *i*. *f i* = *i*} = {*id*}
by *auto*
show *?case*
by (*auto simp add*: *th*)
next
case (*Suc k*)
let *?f* = λ(*y*::*nat*,*g*) *i*. *if i* = *Suc k then y else g i*
let *?S* = *?f* ‘ (*S* × {*f*. (∀ *i*∈{*1*..*k*}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1*..*k*} ⟶ *f i* = *i*)})
have *?S* = {*f*. (∀ *i*∈{*1*.. *Suc k*}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1*.. *Suc k*} ⟶ *f i* = *i*)}
apply (*auto simp add*: *image-iff*)
apply (*rule-tac x*=*x* (*Suc k*) **in** *bexI*)
apply (*rule-tac x* = λ*i*. *if i* = *Suc k then i else x i* **in** *exI*)
apply *auto*
done
with *finite-imageI*[*OF finite-cartesian-product*[*OF fS Suc.hyps*(*1*)], *of ?f*]
show *?case*
by *metis*
qed

lemma *eq-id-iff*[*simp*]: (∀ *x*. *f x* = *x*) ⟷ *f* = *id*
by *auto*

lemma *det-linear-rows-setsum-lemma*:
assumes *fS*: *finite S*
and *fT*: *finite T*
shows *det* ((χ *i*. *if i* ∈ *T then setsum* (*a i*) *S else c i*):: *'a*::*comm-ring-1*^*'n*^*'n*)
=
setsum (λ*f*. *det*((χ *i*. *if i* ∈ *T then a i* (*f i*) *else c i*)::*'a*^*'n*^*'n*))
{*f*. (∀ *i* ∈ *T*. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ *T* ⟶ *f i* = *i*)}
using *fT*

proof (*induct T arbitrary*: *a c set*: *finite*)
case *empty*
have *th0*: $\bigwedge x\ y.\ (\chi\ i.\ \mathit{if}\ i \in \{\}\ \mathit{then}\ x\ i\ \mathit{else}\ y\ i) = (\chi\ i.\ y\ i)$
by *vector*
from *empty.prems* **show** *?case*
unfolding *th0* **by** *simp*
next
case (*insert z T a c*)
let *?F* $= \lambda T.\ \{f.\ (\forall i \in T.\ f\ i \in S) \wedge (\forall i.\ i \notin T \longrightarrow f\ i = i)\}$
let *?h* $= \lambda(y,g)\ i.\ \mathit{if}\ i = z\ \mathit{then}\ y\ \mathit{else}\ g\ i$
let *?k* $= \lambda h.\ (h(z),(\lambda i.\ \mathit{if}\ i = z\ \mathit{then}\ i\ \mathit{else}\ h\ i))$
let *?s* $= \lambda\ k\ a\ c\ f.\ \mathit{det}((\chi\ i.\ \mathit{if}\ i \in T\ \mathit{then}\ a\ i\ (f\ i)\ \mathit{else}\ c\ i)::'a\hat{}'n\hat{}'n)$
let *?c* $= \lambda j\ i.\ \mathit{if}\ i = z\ \mathit{then}\ a\ i\ j\ \mathit{else}\ c\ i$
have *thif*: $\bigwedge a\ b\ c\ d.\ (\mathit{if}\ a \vee b\ \mathit{then}\ c\ \mathit{else}\ d) = (\mathit{if}\ a\ \mathit{then}\ c\ \mathit{else}\ \mathit{if}\ b\ \mathit{then}\ c\ \mathit{else}\ d)$
by *simp*
have *thif2*: $\bigwedge a\ b\ c\ d\ e.\ (\mathit{if}\ a\ \mathit{then}\ b\ \mathit{else}\ \mathit{if}\ c\ \mathit{then}\ d\ \mathit{else}\ e) =$
$(\mathit{if}\ c\ \mathit{then}\ (\mathit{if}\ a\ \mathit{then}\ b\ \mathit{else}\ d)\ \mathit{else}\ (\mathit{if}\ a\ \mathit{then}\ b\ \mathit{else}\ e))$
by *simp*
from ‹$z \notin T$› **have** *nz*: $\bigwedge i.\ i \in T \Longrightarrow i = z \longleftrightarrow \mathit{False}$
by *auto*
have $\mathit{det}\ (\chi\ i.\ \mathit{if}\ i \in \mathit{insert}\ z\ T\ \mathit{then}\ \mathit{setsum}\ (a\ i)\ S\ \mathit{else}\ c\ i) =$
$\mathit{det}\ (\chi\ i.\ \mathit{if}\ i = z\ \mathit{then}\ \mathit{setsum}\ (a\ i)\ S\ \mathit{else}\ \mathit{if}\ i \in T\ \mathit{then}\ \mathit{setsum}\ (a\ i)\ S\ \mathit{else}\ c\ i)$
unfolding *insert-iff thif* **..**
also have $\ldots = (\sum j{\in}S.\ \mathit{det}\ (\chi\ i.\ \mathit{if}\ i \in T\ \mathit{then}\ \mathit{setsum}\ (a\ i)\ S\ \mathit{else}\ \mathit{if}\ i = z\ \mathit{then}\ a\ i\ j\ \mathit{else}\ c\ i))$
unfolding *det-linear-row-setsum*[*OF fS*]
apply (*subst thif2*)
using *nz*
apply (*simp cong del*: *if-weak-cong cong add*: *if-cong*)
done
finally have *tha*:
$\mathit{det}\ (\chi\ i.\ \mathit{if}\ i \in \mathit{insert}\ z\ T\ \mathit{then}\ \mathit{setsum}\ (a\ i)\ S\ \mathit{else}\ c\ i) =$
$(\sum (j, f){\in}S \times\ ?F\ T.\ \mathit{det}\ (\chi\ i.\ \mathit{if}\ i \in T\ \mathit{then}\ a\ i\ (f\ i)$
$\mathit{else}\ \mathit{if}\ i = z\ \mathit{then}\ a\ i\ j$
$\mathit{else}\ c\ i))$
unfolding *insert.hyps* **unfolding** *setsum.cartesian-product* **by** *blast*
show *?case* **unfolding** *tha*
using ‹$z \notin T$›
by (*intro setsum.reindex-bij-witness*[**where** $i{=}?k$ **and** $j{=}?h$])
(*auto intro*!: *cong*[*OF refl*[*of det*]] *simp*: *vec-eq-iff*)
qed

lemma *det-linear-rows-setsum*:
fixes *S* :: *'n*::*finite set*
assumes *fS*: *finite S*
shows $\mathit{det}\ (\chi\ i.\ \mathit{setsum}\ (a\ i)\ S) =$
$\mathit{setsum}\ (\lambda f.\ \mathit{det}\ (\chi\ i.\ a\ i\ (f\ i)) :: 'a{::}\mathit{comm\text{-}ring\text{-}1}\ \hat{}\ 'n\hat{}'n))\ \{f.\ \forall i.\ f\ i \in S\}$
proof −
have *th0*: $\bigwedge x\ y.\ ((\chi\ i.\ \mathit{if}\ i \in (\mathit{UNIV} :: 'n\ \mathit{set})\ \mathit{then}\ x\ i\ \mathit{else}\ y\ i) :: 'a\hat{}'n\hat{}'n) = (\chi$

```
i. x i)
    by vector
  from det-linear-rows-setsum-lemma[OF fS, of UNIV :: 'n set a, unfolded th0,
OF finite]
  show ?thesis by simp
qed

lemma matrix-mul-setsum-alt:
  fixes A B :: 'a::comm-ring-1^'n^'n
  shows A ** B = (χ i. setsum (λk. A$i$k *s B $ k) (UNIV :: 'n set))
  by (vector matrix-matrix-mult-def setsum-component)

lemma det-rows-mul:
  det((χ i. c i *s a i)::'a::comm-ring-1^'n^'n) =
    setprod (λi. c i) (UNIV:: 'n set) * det((χ i. a i)::'a^'n^'n)
proof (simp add: det-def setsum-right-distrib cong add: setprod.cong, rule set-
sum.cong)
  let ?U = UNIV :: 'n set
  let ?PU = {p. p permutes ?U}
  fix p
  assume pU: p ∈ ?PU
  let ?s = of-int (sign p)
  from pU have p: p permutes ?U
    by blast
  have setprod (λi. c i * a i $ p i) ?U = setprod c ?U * setprod (λi. a i $ p i) ?U
    unfolding setprod.distrib ..
  then show ?s * (∏ xa∈?U. c xa * a xa $ p xa) =
    setprod c ?U * (?s* (∏ xa∈?U. a xa $ p xa))
    by (simp add: field-simps)
qed rule

lemma det-mul:
  fixes A B :: 'a::linordered-idom^'n^'n
  shows det (A ** B) = det A * det B
proof −
  let ?U = UNIV :: 'n set
  let ?F = {f. (∀ i∈ ?U. f i ∈ ?U) ∧ (∀ i. i ∉ ?U ⟶ f i = i)}
  let ?PU = {p. p permutes ?U}
  have fU: finite ?U
    by simp
  have fF: finite ?F
    by (rule finite)
  {
    fix p
    assume p: p permutes ?U
    have p ∈ ?F unfolding mem-Collect-eq permutes-in-image[OF p]
      using p[unfolded permutes-def] by simp
  }
  then have PUF: ?PU ⊆ ?F by blast
```

```
 {
  fix f
  assume fPU: f ∈ ?F − ?PU
  have fUU: f ‘ ?U ⊆ ?U
   using fPU by auto
  from fPU have f: ∀ i ∈ ?U. f i ∈ ?U ∀ i. i ∉ ?U ⟶ f i = i ¬(∀ y. ∃!x. f x
= y)
   unfolding permutes-def by auto

  let ?A = (χ i. A$i$f i *s B$f i) :: 'a^'n^'n
  let ?B = (χ i. B$f i) :: 'a^'n^'n
  {
   assume fni: ¬ inj-on f ?U
   then obtain i j where ij: f i = f j i ≠ j
    unfolding inj-on-def by blast
   from ij
   have rth: row i ?B = row j ?B
    by (vector row-def)
   from det-identical-rows[OF ij(2) rth]
   have det (χ i. A$i$f i *s B$f i) = 0
    unfolding det-rows-mul by simp
  }
  moreover
  {
   assume fi: inj-on f ?U
   from f fi have fith: ⋀i j. f i = f j ⟹ i = j
    unfolding inj-on-def by metis
  note fs = fi[unfolded surjective-iff-injective-gen[OF fU fU refl fUU, symmetric]]
   {
    fix y
    from fs f have ∃ x. f x = y
     by blast
    then obtain x where x: f x = y
     by blast
    {
     fix z
     assume z: f z = y
     from fith x z have z = x
      by metis
    }
    with x have ∃!x. f x = y
     by blast
   }
   with f(3) have det (χ i. A$i$f i *s B$f i) = 0
    by blast
  }
  ultimately have det (χ i. A$i$f i *s B$f i) = 0
   by blast
 }
```

```
  then have zth: ∀ f∈ ?F - ?PU. det (χ i. A$i$f i *s B$f i) = 0
    by simp
  {
    fix p
    assume pU: p ∈ ?PU
    from pU have p: p permutes ?U
      by blast
    let ?s = λp. of-int (sign p)
    let ?f = λq. ?s p * (∏ i∈ ?U. A $ i $ p i) * (?s q * (∏ i∈ ?U. B $ i $ q i))
    have (setsum (λq. ?s q *
        (∏ i∈ ?U. (χ i. A $ i $ p i *s B $ p i :: 'a^'n^'n) $ i $ q i)) ?PU) =
      (setsum (λq. ?s p * (∏ i∈ ?U. A $ i $ p i) * (?s q * (∏ i∈ ?U. B $ i $ q
i))) ?PU)
      unfolding sum-permutations-compose-right[OF permutes-inv[OF p], of ?f]
    proof (rule setsum.cong)
      fix q
      assume qU: q ∈ ?PU
      then have q: q permutes ?U
        by blast
      from p q have pp: permutation p and pq: permutation q
        unfolding permutation-permutes by auto
      have th00: of-int (sign p) * of-int (sign p) = (1::'a)
        ⋀a. of-int (sign p) * (of-int (sign p) * a) = a
        unfolding mult.assoc[symmetric]
        unfolding of-int-mult[symmetric]
        by (simp-all add: sign-idempotent)
      have ths: ?s q = ?s p * ?s (q ∘ inv p)
        using pp pq permutation-inverse[OF pp] sign-inverse[OF pp]
        by (simp add:  th00 ac-simps sign-idempotent sign-compose)
      have th001: setprod (λi. B$i$ q (inv p i)) ?U = setprod ((λi. B$i$ q (inv p
i)) ∘ p) ?U
        by (rule setprod-permute[OF p])
      have thp: setprod (λi. (χ i. A$i$p i *s B$p i :: 'a^'n^'n) $i $ q i) ?U =
        setprod (λi. A$i$p i) ?U * setprod (λi. B$i$ q (inv p i)) ?U
        unfolding th001 setprod.distrib[symmetric] o-def permutes-inverses[OF p]
        apply (rule setprod.cong[OF refl])
        using permutes-in-image[OF q]
        apply vector
        done
      show ?s q * setprod (λi. (((χ i. A$i$p i *s B$p i) :: 'a^'n^'n)$i$q i)) ?U =
        ?s p * (setprod (λi. A$i$p i) ?U) * (?s (q ∘ inv p) * setprod (λi. B$i$(q ∘
inv p) i) ?U)
        using ths thp pp pq permutation-inverse[OF pp] sign-inverse[OF pp]
        by (simp add: sign-nz th00 field-simps sign-idempotent sign-compose)
    qed rule
  }
  then have th2: setsum (λf. det (χ i. A$i$f i *s B$f i)) ?PU = det A * det B
    unfolding det-def setsum-product
    by (rule setsum.cong [OF refl])
```

```
  have det (A**B) = setsum (λf.  det (χ i. A $ i $ f i *s B $ f i)) ?F
    unfolding matrix-mul-setsum-alt det-linear-rows-setsum[OF fU]
    by simp
  also have ... = setsum (λf. det (χ i. A$i$f i *s B$f i)) ?PU
    using setsum.mono-neutral-cong-left[OF fF PUF zth, symmetric]
    unfolding det-rows-mul by auto
  finally show ?thesis unfolding th2 .
qed
```

Relation to invertibility.

```
lemma invertible-left-inverse:
  fixes A :: real^'n^'n
  shows invertible A ⟷ (∃(B::real^'n^'n). B** A = mat 1)
  by (metis invertible-def matrix-left-right-inverse)

lemma invertible-righ-inverse:
  fixes A :: real^'n^'n
  shows invertible A ⟷ (∃(B::real^'n^'n). A** B = mat 1)
  by (metis invertible-def matrix-left-right-inverse)

lemma invertible-det-nz:
  fixes A::real ^'n^'n
  shows invertible A ⟷ det A ≠ 0
proof −
  {
    assume invertible A
    then obtain B :: real ^'n^'n where B: A ** B = mat 1
      unfolding invertible-righ-inverse by blast
    then have det (A ** B) = det (mat 1 :: real ^'n^'n)
      by simp
    then have det A ≠ 0
      by (simp add: det-mul det-I) algebra
  }
  moreover
  {
    assume H: ¬ invertible A
    let ?U = UNIV :: 'n set
    have fU: finite ?U
      by simp
    from H obtain c i where c: setsum (λi. c i *s row i A) ?U = 0
      and iU: i ∈ ?U
      and ci: c i ≠ 0
      unfolding invertible-righ-inverse
      unfolding matrix-right-invertible-independent-rows
      by blast
    have *: ⋀(a::real^'n) b. a + b = 0 ⟹ −a = b
      apply (drule-tac f=op + (− a) in cong[OF refl])
      apply (simp only: ab-left-minus add.assoc[symmetric])
      apply simp
```

```
    done
  from c ci
  have thr0: − row i A = setsum (λj. (1/ c i) *s (c j *s row j A)) (?U − {i})
    unfolding setsum.remove[OF fU iU] setsum-cmul
    apply −
    apply (rule vector-mul-lcancel-imp[OF ci])
    apply (auto simp add: field-simps)
    unfolding *
    apply rule
    done
  have thr: − row i A ∈ span {row j A| j. j ≠ i}
    unfolding thr0
    apply (rule span-setsum)
    apply simp
    apply (rule ballI)
    apply (rule span-mul [where 'a=real^'n, folded scalar-mult-eq-scaleR])+
    apply (rule span-superset)
    apply auto
    done
  let ?B = (χ k. if k = i then 0 else row k A) :: real ^'n^'n
  have thrb: row i ?B = 0 using iU by (vector row-def)
  have det A = 0
    unfolding det-row-span[OF thr, symmetric] right-minus
    unfolding det-zero-row[OF thrb] ..
 }
 ultimately show ?thesis
  by blast
qed
```

Cramer's rule.

```
lemma cramer-lemma-transpose:
 fixes A:: real^'n^'n
  and x :: real^'n
 shows det ((χ i. if i = k then setsum (λi. x$i *s row i A) (UNIV::'n set)
                          else row i A)::real^'n^'n) = x$k * det A
 (is ?lhs = ?rhs)
proof −
 let ?U = UNIV :: 'n set
 let ?Uk = ?U − {k}
 have U: ?U = insert k ?Uk
  by blast
 have fUk: finite ?Uk
  by simp
 have kUk: k ∉ ?Uk
  by simp
 have th00: ⋀k s. x$k *s row k A + s = (x$k − 1) *s row k A + row k A + s
  by (vector field-simps)
 have th001: ⋀f k . (λx. if x = k then f k else f x) = f
  by auto
```

```
  have (χ i. row i A) = A by (vector row-def)
  then have thd1: det (χ i. row i A) = det A
    by simp
  have thd0: det (χ i. if i = k then row k A + (∑ i ∈ ?Uk. x $ i *s row i A) else
row i A) = det A
    apply (rule det-row-span)
    apply (rule span-setsum)
    apply (rule ballI)
    apply (rule span-mul [where 'a=real^'n, folded scalar-mult-eq-scaleR])+
    apply (rule span-superset)
    apply auto
    done
  show ?lhs = x$k * det A
    apply (subst U)
    unfolding setsum.insert[OF fUk kUk]
    apply (subst th00)
    unfolding add.assoc
    apply (subst det-row-add)
    unfolding thd0
    unfolding det-row-mul
    unfolding th001[of k λi. row i A]
    unfolding thd1
    apply (simp add: field-simps)
    done
qed

lemma cramer-lemma:
  fixes A :: real^'n^'n
  shows det((χ i j. if j = k then (A *v x)$i else A$i$j):: real^'n^'n) = x$k * det
A
proof −
  let ?U = UNIV :: 'n set
  have *: ⋀c. setsum (λi. c i *s row i (transpose A)) ?U = setsum (λi. c i *s
column i A) ?U
    by (auto simp add: row-transpose intro: setsum.cong)
  show ?thesis
    unfolding matrix-mult-vsum
    unfolding cramer-lemma-transpose[of k x transpose A, unfolded det-transpose,
symmetric]
    unfolding *[of λi. x$i]
    apply (subst det-transpose[symmetric])
    apply (rule cong[OF refl[of det]])
    apply (vector transpose-def column-def row-def)
    done
qed

lemma cramer:
  fixes A ::real^'n^'n
  assumes d0: det A ≠ 0
```

shows $A *v x = b \longleftrightarrow x = (\chi\ k.\ det(\chi\ i\ j.\ if\ j{=}k\ then\ b\$i\ else\ A\$i\$j)\ /\ det\ A)$
proof −
from *d0* **obtain** B **where** B: $A ** B = mat\ 1\ B ** A = mat\ 1$
unfolding *invertible-det-nz*[*symmetric*] *invertible-def*
by *blast*
have $(A ** B) *v\ b = b$
by (*simp add*: B *matrix-vector-mul-lid*)
then have $A *v\ (B *v\ b) = b$
by (*simp add*: *matrix-vector-mul-assoc*)
then have *xe*: $\exists x.\ A *v\ x = b$
by *blast*
{
fix x
assume x: $A *v\ x = b$
have $x = (\chi\ k.\ det(\chi\ i\ j.\ if\ j{=}k\ then\ b\$i\ else\ A\$i\$j)\ /\ det\ A)$
unfolding x[*symmetric*]
using *d0* **by** (*simp add*: *vec-eq-iff cramer-lemma field-simps*)
}
with *xe* **show** *?thesis*
by *auto*
qed

Orthogonality of a transformation and matrix.

definition *orthogonal-transformation* $f \longleftrightarrow linear\ f \wedge (\forall\ v\ w.\ f\ v \cdot f\ w = v \cdot w)$

lemma *orthogonal-transformation*:
orthogonal-transformation $f \longleftrightarrow linear\ f \wedge (\forall\,(v{::}real\ \hat{}\text{-}).\ norm\ (f\ v) = norm\ v)$
unfolding *orthogonal-transformation-def*
apply *auto*
apply (*erule-tac* $x{=}v$ **in** *allE*)+
apply (*simp add*: *norm-eq-sqrt-inner*)
apply (*simp add*: *dot-norm linear-add*[*symmetric*])
done

definition *orthogonal-matrix* $(Q{::}'a{::}semiring\text{-}1\hat{}'n\hat{}'n) \longleftrightarrow$
transpose $Q ** Q = mat\ 1 \wedge Q ** transpose\ Q = mat\ 1$

lemma *orthogonal-matrix*: *orthogonal-matrix* $(Q{::}\ real\ \hat{}'n\hat{}'n) \longleftrightarrow$ *transpose* Q $** Q = mat\ 1$
by (*metis matrix-left-right-inverse orthogonal-matrix-def*)

lemma *orthogonal-matrix-id*: *orthogonal-matrix* $(mat\ 1 :: \text{-}\hat{}'n\hat{}'n)$
by (*simp add*: *orthogonal-matrix-def transpose-mat matrix-mul-lid*)

lemma *orthogonal-matrix-mul*:
fixes $A :: real\ \hat{}'n\hat{}'n$
assumes oA : *orthogonal-matrix* A
and oB: *orthogonal-matrix* B
shows *orthogonal-matrix*$(A ** B)$

```
  using oA oB
  unfolding orthogonal-matrix matrix-transpose-mul
  apply (subst matrix-mul-assoc)
  apply (subst matrix-mul-assoc[symmetric])
  apply (simp add: matrix-mul-rid)
  done

lemma orthogonal-transformation-matrix:
  fixes f:: real^'n ⇒ real^'n
  shows orthogonal-transformation f ⟷ linear f ∧ orthogonal-matrix(matrix f)
  (is ?lhs ⟷ ?rhs)
proof −
  let ?mf = matrix f
  let ?ot = orthogonal-transformation f
  let ?U = UNIV :: 'n set
  have fU: finite ?U by simp
  let ?m1 = mat 1 :: real ^'n^'n
  {
    assume ot: ?ot
    from ot have lf: linear f and fd: ∀ v w. f v · f w = v · w
      unfolding  orthogonal-transformation-def orthogonal-matrix by blast+
    {
      fix i j
      let ?A = transpose ?mf ** ?mf
      have th0: ⋀b (x::'a::comm-ring-1). (if b then 1 else 0)*x = (if b then x else
0)
        ⋀b (x::'a::comm-ring-1). x*(if b then 1 else 0) = (if b then x else 0)
        by simp-all
      from fd[rule-format, of axis i 1 axis j 1, unfolded matrix-works[OF lf, sym-
metric] dot-matrix-vector-mul]
      have ?A$i$j = ?m1 $ i $ j
      by (simp add: inner-vec-def matrix-matrix-mult-def columnvector-def rowvector-def
          th0 setsum.delta[OF fU] mat-def axis-def)
    }
    then have orthogonal-matrix ?mf
      unfolding orthogonal-matrix
      by vector
    with lf have ?rhs
      by blast
  }
  moreover
  {
    assume lf: linear f and om: orthogonal-matrix ?mf
    from lf om have ?lhs
      unfolding orthogonal-matrix-def norm-eq orthogonal-transformation
      unfolding matrix-works[OF lf, symmetric]
      apply (subst dot-matrix-vector-mul)
      apply (simp add: dot-matrix-product matrix-mul-lid)
      done
```

```
  }
  ultimately show ?thesis
    by blast
qed

lemma det-orthogonal-matrix:
  fixes Q:: 'a::linordered-idom^'n^'n
  assumes oQ: orthogonal-matrix Q
  shows det Q = 1 ∨ det Q = - 1
proof -
  have th: ⋀x::'a. x = 1 ∨ x = - 1 ⟷ x*x = 1 (is ⋀x::'a. ?ths x)
  proof -
    fix x:: 'a
    have th0: x * x - 1 = (x - 1) * (x + 1)
      by (simp add: field-simps)
    have th1: ⋀(x::'a) y. x = - y ⟷ x + y = 0
      apply (subst eq-iff-diff-eq-0)
      apply simp
      done
    have x * x = 1 ⟷ x * x - 1 = 0
      by simp
    also have ... ⟷ x = 1 ∨ x = - 1
      unfolding th0 th1 by simp
    finally show ?ths x ..
  qed
  from oQ have Q ** transpose Q = mat 1
    by (metis orthogonal-matrix-def)
  then have det (Q ** transpose Q) = det (mat 1:: 'a^'n^'n)
    by simp
  then have det Q * det Q = 1
    by (simp add: det-mul det-I det-transpose)
  then show ?thesis unfolding th .
qed
```

Linearity of scaling, and hence isometry, that preserves origin.

```
lemma scaling-linear:
  fixes f :: real ^'n ⇒ real ^'n
  assumes f0: f 0 = 0
    and fd: ∀ x y. dist (f x) (f y) = c * dist x y
  shows linear f
proof -
  {
    fix v w
    {
      fix x
      note fd[rule-format, of x 0, unfolded dist-norm f0 diff-0-right]
    }
    note th0 = this
    have f v · f w = c^2 * (v · w)
```

```
    unfolding dot-norm-neg dist-norm[symmetric]
    unfolding th0 fd[rule-format] by (simp add: power2-eq-square field-simps)}
  note fc = this
  show ?thesis
    unfolding linear-iff vector-eq[where 'a=real^'n] scalar-mult-eq-scaleR
    by (simp add: inner-add fc field-simps)
qed

lemma isometry-linear:
  f (0:: real^'n) = (0:: real^'n) ⟹ ∀ x y. dist(f x) (f y) = dist x y ⟹ linear f
  by (rule scaling-linear[where c=1]) simp-all
```

Hence another formulation of orthogonal transformation.

```
lemma orthogonal-transformation-isometry:
  orthogonal-transformation f ⟷ f(0::real^'n) = (0::real^'n) ∧ (∀ x y. dist(f x)
(f y) = dist x y)
  unfolding orthogonal-transformation
  apply (rule iffI)
  apply clarify
  apply (clarsimp simp add: linear-0 linear-sub[symmetric] dist-norm)
  apply (rule conjI)
  apply (rule isometry-linear)
  apply simp
  apply simp
  apply clarify
  apply (erule-tac x=v in allE)
  apply (erule-tac x=0 in allE)
  apply (simp add: dist-norm)
  done
```

Can extend an isometry from unit sphere.

```
lemma isometry-sphere-extend:
  fixes f:: real ^'n ⇒ real ^'n
  assumes f1: ∀ x. norm x = 1 ⟶ norm (f x) = 1
    and fd1: ∀ x y. norm x = 1 ⟶ norm y = 1 ⟶ dist (f x) (f y) = dist x y
  shows ∃ g. orthogonal-transformation g ∧ (∀ x. norm x = 1 ⟶ g x = f x)
proof −
  {
    fix x y x' y' x0 y0 x0' y0' :: real ^'n
    assume H:
      x = norm x *R x0
      y = norm y *R y0
      x' = norm x *R x0' y' = norm y *R y0'
      norm x0 = 1 norm x0' = 1 norm y0 = 1 norm y0' = 1
      norm(x0' − y0') = norm(x0 − y0)
    then have *: x0 · y0 = x0' · y0' + y0' · x0' − y0 · x0
      by (simp add: norm-eq norm-eq-1 inner-add inner-diff)
    have norm(x' − y') = norm(x − y)
      apply (subst H(1))
```

```
    apply (subst H(2))
    apply (subst H(3))
    apply (subst H(4))
    using H(5-9)
    apply (simp add: norm-eq norm-eq-1)
    apply (simp add: inner-diff scalar-mult-eq-scaleR)
    unfolding *
    apply (simp add: field-simps)
    done
}
note th0 = this
let ?g = λx. if x = 0 then 0 else norm x *R f (inverse (norm x) *R x)
{
  fix x:: real ^'n
  assume nx: norm x = 1
  have ?g x = f x
    using nx by auto
}
then have thfg: ∀ x. norm x = 1 ⟶ ?g x = f x
  by blast
have g0: ?g 0 = 0
  by simp
{
  fix x y :: real ^'n
  {
    assume x = 0 y = 0
    then have dist (?g x) (?g y) = dist x y
      by simp
  }
  moreover
  {
    assume x = 0 y ≠ 0
    then have dist (?g x) (?g y) = dist x y
      apply (simp add: dist-norm)
      apply (rule f1[rule-format])
      apply (simp add: field-simps)
      done
  }
  moreover
  {
    assume x ≠ 0 y = 0
    then have dist (?g x) (?g y) = dist x y
      apply (simp add: dist-norm)
      apply (rule f1[rule-format])
      apply (simp add: field-simps)
      done
  }
  moreover
  {
```

```
    assume z: x ≠ 0 y ≠ 0
    have th00:
      x = norm x *R (inverse (norm x) *R x)
      y = norm y *R (inverse (norm y) *R y)
      norm x *R f ((inverse (norm x) *R x)) = norm x *R f (inverse (norm x)
*R x)
      norm y *R f (inverse (norm y) *R y) = norm y *R f (inverse (norm y) *R
y)
      norm (inverse (norm x) *R x) = 1
      norm (f (inverse (norm x) *R x)) = 1
      norm (inverse (norm y) *R y) = 1
      norm (f (inverse (norm y) *R y)) = 1
      norm (f (inverse (norm x) *R x) − f (inverse (norm y) *R y)) =
        norm (inverse (norm x) *R x − inverse (norm y) *R y)
      using z
    by (auto simp add: field-simps intro: f1[rule-format] fd1[rule-format, unfolded
dist-norm])
    from z th0[OF th00] have dist (?g x) (?g y) = dist x y
      by (simp add: dist-norm)
  }
  ultimately have dist (?g x) (?g y) = dist x y
    by blast
 }
 note thd = this
  show ?thesis
  apply (rule exI[where x= ?g])
  unfolding orthogonal-transformation-isometry
  using g0 thfg thd
  apply metis
  done
qed
```

Rotation, reflection, rotoinversion.

```
definition rotation-matrix Q ⟷ orthogonal-matrix Q ∧ det Q = 1
definition rotoinversion-matrix Q ⟷ orthogonal-matrix Q ∧ det Q = − 1

lemma orthogonal-rotation-or-rotoinversion:
 fixes Q :: 'a::linordered-idom^'n^'n
 shows  orthogonal-matrix Q ⟷ rotation-matrix Q ∨ rotoinversion-matrix Q
 by (metis rotoinversion-matrix-def rotation-matrix-def det-orthogonal-matrix)
```

Explicit formulas for low dimensions.

```
lemma setprod-neutral-const: setprod f {(1::nat)..1} = f 1
 by simp

lemma setprod-2: setprod f {(1::nat)..2} = f 1 * f 2
 by (simp add: eval-nat-numeral atLeastAtMostSuc-conv mult.commute)

lemma setprod-3: setprod f {(1::nat)..3} = f 1 * f 2 * f 3
```

by (*simp add*: *eval-nat-numeral atLeastAtMostSuc-conv mult.commute*)

lemma *det-1*: *det* (*A*::*'a*::*comm-ring-1^1^1*) = *A$1$1*
by (*simp add*: *det-def of-nat-Suc sign-id*)

lemma *det-2*: *det* (*A*::*'a*::*comm-ring-1^2^2*) = *A$1$1* * *A$2$2* − *A$1$2* * *A$2$1*
proof −
have *f12*: *finite* {*2*::*2*} *1* ∉ {*2*::*2*} **by** *auto*
show *?thesis*
unfolding *det-def UNIV-2*
unfolding *setsum-over-permutations-insert*[*OF f12*]
unfolding *permutes-sing*
by (*simp add*: *sign-swap-id sign-id swap-id-eq*)
qed

lemma *det-3*:
det (*A*::*'a*::*comm-ring-1^3^3*) =
A$1$1 * *A$2$2* * *A$3$3* +
A$1$2 * *A$2$3* * *A$3$1* +
A$1$3 * *A$2$1* * *A$3$2* −
A$1$1 * *A$2$3* * *A$3$2* −
A$1$2 * *A$2$1* * *A$3$3* −
A$1$3 * *A$2$2* * *A$3$1*
proof −
have *f123*: *finite* {*2*::*3*, *3*} *1* ∉ {*2*::*3*, *3*}
by *auto*
have *f23*: *finite* {*3*::*3*} *2* ∉ {*3*::*3*}
by *auto*

show *?thesis*
unfolding *det-def UNIV-3*
unfolding *setsum-over-permutations-insert*[*OF f123*]
unfolding *setsum-over-permutations-insert*[*OF f23*]
unfolding *permutes-sing*
by (*simp add*: *sign-swap-id permutation-swap-id sign-compose sign-id swap-id-eq*)
qed

end

45 Pointwise order on product types

theory *Product-Order*
imports *Product-plus Conditionally-Complete-Lattices*
begin

45.1 Pointwise ordering

instantiation *prod* :: (*ord*, *ord*) *ord*
begin

definition
$x \leq y \longleftrightarrow fst\ x \leq fst\ y \wedge snd\ x \leq snd\ y$

definition
$(x::'a \times 'b) < y \longleftrightarrow x \leq y \wedge \neg\ y \leq x$

instance ..

end

lemma *fst-mono*: $x \leq y \Longrightarrow fst\ x \leq fst\ y$
unfolding *less-eq-prod-def* **by** *simp*

lemma *snd-mono*: $x \leq y \Longrightarrow snd\ x \leq snd\ y$
unfolding *less-eq-prod-def* **by** *simp*

lemma *Pair-mono*: $x \leq x' \Longrightarrow y \leq y' \Longrightarrow (x, y) \leq (x', y')$
unfolding *less-eq-prod-def* **by** *simp*

lemma *Pair-le* [*simp*]: $(a, b) \leq (c, d) \longleftrightarrow a \leq c \wedge b \leq d$
unfolding *less-eq-prod-def* **by** *simp*

instance *prod* :: (*preorder*, *preorder*) *preorder*
proof
fix $x\ y\ z :: 'a \times 'b$
show $x < y \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
by (*rule less-prod-def*)
show $x \leq x$
unfolding *less-eq-prod-def*
by *fast*
assume $x \leq y$ **and** $y \leq z$ **thus** $x \leq z$
unfolding *less-eq-prod-def*
by (*fast elim*: *order-trans*)
qed

instance *prod* :: (*order*, *order*) *order*
by *standard auto*

45.2 Binary infimum and supremum

instantiation *prod* :: (*inf*, *inf*) *inf*
begin

definition $inf\ x\ y = (inf\ (fst\ x)\ (fst\ y), inf\ (snd\ x)\ (snd\ y))$

lemma *inf-Pair-Pair* [*simp*]: $inf\ (a, b)\ (c, d) = (inf\ a\ c, inf\ b\ d)$
unfolding *inf-prod-def* **by** *simp*

lemma *fst-inf* [*simp*]: *fst* (*inf x y*) = *inf* (*fst x*) (*fst y*)
unfolding *inf-prod-def* **by** *simp*

lemma *snd-inf* [*simp*]: *snd* (*inf x y*) = *inf* (*snd x*) (*snd y*)
unfolding *inf-prod-def* **by** *simp*

instance ..

end

instance *prod* :: (*semilattice-inf*, *semilattice-inf*) *semilattice-inf*
by *standard auto*

instantiation *prod* :: (*sup*, *sup*) *sup*
begin

definition
sup x y = (*sup* (*fst x*) (*fst y*), *sup* (*snd x*) (*snd y*))

lemma *sup-Pair-Pair* [*simp*]: *sup* (*a*, *b*) (*c*, *d*) = (*sup a c*, *sup b d*)
unfolding *sup-prod-def* **by** *simp*

lemma *fst-sup* [*simp*]: *fst* (*sup x y*) = *sup* (*fst x*) (*fst y*)
unfolding *sup-prod-def* **by** *simp*

lemma *snd-sup* [*simp*]: *snd* (*sup x y*) = *sup* (*snd x*) (*snd y*)
unfolding *sup-prod-def* **by** *simp*

instance ..

end

instance *prod* :: (*semilattice-sup*, *semilattice-sup*) *semilattice-sup*
by *standard auto*

instance *prod* :: (*lattice*, *lattice*) *lattice* ..

instance *prod* :: (*distrib-lattice*, *distrib-lattice*) *distrib-lattice*
by *standard* (*auto simp add*: *sup-inf-distrib1*)

45.3 Top and bottom elements

instantiation *prod* :: (*top*, *top*) *top*
begin

definition
top = (*top*, *top*)

instance ..

end

lemma *fst-top* [*simp*]: *fst top* = *top*
unfolding *top-prod-def* **by** *simp*

lemma *snd-top* [*simp*]: *snd top* = *top*
unfolding *top-prod-def* **by** *simp*

lemma *Pair-top-top*: (*top*, *top*) = *top*
unfolding *top-prod-def* **by** *simp*

instance *prod* :: (*order-top*, *order-top*) *order-top*
by *standard* (*auto simp add*: *top-prod-def*)

instantiation *prod* :: (*bot*, *bot*) *bot*
begin

definition
bot = (*bot*, *bot*)

instance ..

end

lemma *fst-bot* [*simp*]: *fst bot* = *bot*
unfolding *bot-prod-def* **by** *simp*

lemma *snd-bot* [*simp*]: *snd bot* = *bot*
unfolding *bot-prod-def* **by** *simp*

lemma *Pair-bot-bot*: (*bot*, *bot*) = *bot*
unfolding *bot-prod-def* **by** *simp*

instance *prod* :: (*order-bot*, *order-bot*) *order-bot*
by *standard* (*auto simp add*: *bot-prod-def*)

instance *prod* :: (*bounded-lattice*, *bounded-lattice*) *bounded-lattice* **..**

instance *prod* :: (*boolean-algebra*, *boolean-algebra*) *boolean-algebra*
by *standard* (*auto simp add*: *prod-eqI diff-eq*)

45.4 Complete lattice operations

instantiation *prod* :: (*Inf*, *Inf*) *Inf*
begin

definition *Inf A* = (*INF x*:*A*. *fst x*, *INF x*:*A*. *snd x*)

instance ..

end

instantiation *prod* :: (*Sup*, *Sup*) *Sup*
begin

definition *Sup A* = (*SUP x*:*A*. *fst x*, *SUP x*:*A*. *snd x*)

instance ..

end

instance *prod* :: (*conditionally-complete-lattice*, *conditionally-complete-lattice*)
conditionally-complete-lattice
by *standard* (*force simp*: *less-eq-prod-def Inf-prod-def Sup-prod-def bdd-below-def bdd-above-def*
INF-def SUP-def simp del: *Inf-image-eq Sup-image-eq*
intro!: *cInf-lower cSup-upper cInf-greatest cSup-least*)+

instance *prod* :: (*complete-lattice*, *complete-lattice*) *complete-lattice*
by *standard* (*simp-all add*: *less-eq-prod-def Inf-prod-def Sup-prod-def*
INF-lower SUP-upper le-INF-iff SUP-le-iff bot-prod-def top-prod-def)

lemma *fst-Sup*: *fst* (*Sup A*) = (*SUP x*:*A*. *fst x*)
unfolding *Sup-prod-def* **by** *simp*

lemma *snd-Sup*: *snd* (*Sup A*) = (*SUP x*:*A*. *snd x*)
unfolding *Sup-prod-def* **by** *simp*

lemma *fst-Inf*: *fst* (*Inf A*) = (*INF x*:*A*. *fst x*)
unfolding *Inf-prod-def* **by** *simp*

lemma *snd-Inf*: *snd* (*Inf A*) = (*INF x*:*A*. *snd x*)
unfolding *Inf-prod-def* **by** *simp*

lemma *fst-SUP*: *fst* (*SUP x*:*A*. *f x*) = (*SUP x*:*A*. *fst* (*f x*))
using *fst-Sup* [*of f* ‘ *A*, *symmetric*] **by** (*simp add*: *comp-def*)

lemma *snd-SUP*: *snd* (*SUP x*:*A*. *f x*) = (*SUP x*:*A*. *snd* (*f x*))
using *snd-Sup* [*of f* ‘ *A*, *symmetric*] **by** (*simp add*: *comp-def*)

lemma *fst-INF*: *fst* (*INF x*:*A*. *f x*) = (*INF x*:*A*. *fst* (*f x*))
using *fst-Inf* [*of f* ‘ *A*, *symmetric*] **by** (*simp add*: *comp-def*)

lemma *snd-INF*: *snd* (*INF x*:*A*. *f x*) = (*INF x*:*A*. *snd* (*f x*))
using *snd-Inf* [*of f* ‘ *A*, *symmetric*] **by** (*simp add*: *comp-def*)

lemma *SUP-Pair*: $(SUP\ x{:}A.\ (f\ x,\ g\ x)) = (SUP\ x{:}A.\ f\ x,\ SUP\ x{:}A.\ g\ x)$
unfolding *SUP-def Sup-prod-def* **by** (*simp add*: *comp-def*)

lemma *INF-Pair*: $(INF\ x{:}A.\ (f\ x,\ g\ x)) = (INF\ x{:}A.\ f\ x,\ INF\ x{:}A.\ g\ x)$
unfolding *INF-def Inf-prod-def* **by** (*simp add*: *comp-def*)

Alternative formulations for set infima and suprema over the product of two complete lattices:

lemma *INF-prod-alt-def*:
$INFIMUM\ A\ f = (INFIMUM\ A\ (fst\ o\ f),\ INFIMUM\ A\ (snd\ o\ f))$
unfolding *INF-def Inf-prod-def* **by** *simp*

lemma *SUP-prod-alt-def*:
$SUPREMUM\ A\ f = (SUPREMUM\ A\ (fst\ o\ f),\ SUPREMUM\ A\ (snd\ o\ f))$
unfolding *SUP-def Sup-prod-def* **by** *simp*

45.5 Complete distributive lattices

instance *prod* :: (*complete-distrib-lattice*, *complete-distrib-lattice*) *complete-distrib-lattice*
proof (*standard*, *goal-cases*)
case *1*
then show *?case*
by (*auto simp*: *sup-prod-def Inf-prod-def INF-prod-alt-def sup-Inf sup-INF comp-def*)
next
case *2*
then show *?case*
by (*auto simp*: *inf-prod-def Sup-prod-def SUP-prod-alt-def inf-Sup inf-SUP comp-def*)
qed

end
theory *Ordered-Euclidean-Space*
imports
Cartesian-Euclidean-Space
~~/*src*/*HOL*/*Library*/*Product-Order*
begin

45.6 An ordering on euclidean spaces that will allow us to talk about intervals

class *ordered-euclidean-space* = *ord* + *inf* + *sup* + *abs* + *Inf* + *Sup* + *euclidean-space* +
assumes *eucl-le*: $x \le y \longleftrightarrow (\forall i{\in}Basis.\ x \cdot i \le y \cdot i)$
assumes *eucl-less-le-not-le*: $x < y \longleftrightarrow x \le y \wedge \neg\ y \le x$
assumes *eucl-inf*: $inf\ x\ y = (\sum i{\in}Basis.\ inf\ (x \cdot i)\ (y \cdot i) *_R i)$
assumes *eucl-sup*: $sup\ x\ y = (\sum i{\in}Basis.\ sup\ (x \cdot i)\ (y \cdot i) *_R i)$
assumes *eucl-Inf*: $Inf\ X = (\sum i{\in}Basis.\ (INF\ x{:}X.\ x \cdot i) *_R i)$
assumes *eucl-Sup*: $Sup\ X = (\sum i{\in}Basis.\ (SUP\ x{:}X.\ x \cdot i) *_R i)$

assumes *eucl-abs*: $|x| = (\sum i{\in}Basis.\ |x \cdot i| *_R i)$
begin

subclass *order*
by *standard*
(*auto simp*: *eucl-le eucl-less-le-not-le intro*!: *euclidean-eqI antisym intro*: *order.trans*)

subclass *ordered-ab-group-add-abs*
by *standard* (*auto simp*: *eucl-le inner-add-left eucl-abs abs-leI*)

subclass *ordered-real-vector*
by *standard* (*auto simp*: *eucl-le intro*!: *mult-left-mono mult-right-mono*)

subclass *lattice*
by *standard* (*auto simp*: *eucl-inf eucl-sup eucl-le*)

subclass *distrib-lattice*
by *standard* (*auto simp*: *eucl-inf eucl-sup sup-inf-distrib1 intro*!: *euclidean-eqI*)

subclass *conditionally-complete-lattice*
proof
fix $z{::}'a$ **and** $X{::}'a\ set$
assume $X \neq \{\}$
hence $\bigwedge i.\ (\lambda x.\ x \cdot i)\ `\ X \neq \{\}$ **by** *simp*
thus $(\bigwedge x.\ x \in X \Longrightarrow z \leq x) \Longrightarrow z \leq Inf\ X\ (\bigwedge x.\ x \in X \Longrightarrow x \leq z) \Longrightarrow Sup\ X \leq z$
by (*auto simp*: *eucl-Inf eucl-Sup eucl-le Inf-class.INF-def Sup-class.SUP-def*
simp del: *Inf-class.Inf-image-eq Sup-class.Sup-image-eq*
intro!: *cInf-greatest cSup-least*)
qed (*force intro*!: *cInf-lower cSup-upper*
simp: *bdd-below-def bdd-above-def preorder-class.bdd-below-def preorder-class.bdd-above-def*
eucl-Inf eucl-Sup eucl-le Inf-class.INF-def Sup-class.SUP-def
simp del: *Inf-class.Inf-image-eq Sup-class.Sup-image-eq*)+

lemma *inner-Basis-inf-left*: $i \in Basis \Longrightarrow inf\ x\ y \cdot i = inf\ (x \cdot i)\ (y \cdot i)$
and *inner-Basis-sup-left*: $i \in Basis \Longrightarrow sup\ x\ y \cdot i = sup\ (x \cdot i)\ (y \cdot i)$
by (*simp-all add*: *eucl-inf eucl-sup inner-setsum-left inner-Basis if-distrib comm-monoid-add-class.setsum.del*
cong: *if-cong*)

lemma *inner-Basis-INF-left*: $i \in Basis \Longrightarrow (INF\ x{:}X.\ f\ x) \cdot i = (INF\ x{:}X.\ f\ x \cdot i)$
and *inner-Basis-SUP-left*: $i \in Basis \Longrightarrow (SUP\ x{:}X.\ f\ x) \cdot i = (SUP\ x{:}X.\ f\ x \cdot i)$
using *eucl-Sup* [*of f ` X*] *eucl-Inf* [*of f ` X*] **by** (*simp-all add*: *comp-def*)

lemma *abs-inner*: $i \in Basis \Longrightarrow |x| \cdot i = |x \cdot i|$
by (*auto simp*: *eucl-abs*)

lemma
abs-scaleR: $|a *_R b| = |a| *_R |b|$
by (*auto simp*: *eucl-abs abs-mult intro*!: *euclidean-eqI*)

lemma *interval-inner-leI*:
assumes $x \in \{a .. b\}$ $0 \leq i$
shows $a \cdot i \leq x \cdot i$ $x \cdot i \leq b \cdot i$
using *assms*
unfolding *euclidean-inner*[*of a i*] *euclidean-inner*[*of x i*] *euclidean-inner*[*of b i*]
by (*auto intro*!: *setsum-mono mult-right-mono simp*: *eucl-le*)

lemma *inner-nonneg-nonneg*:
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 \leq a \cdot b$
using *interval-inner-leI*[*of a 0 a b*]
by *auto*

lemma *inner-Basis-mono*:
shows $a \leq b \Longrightarrow c \in Basis \Longrightarrow a \cdot c \leq b \cdot c$
by (*simp add*: *eucl-le*)

lemma *Basis-nonneg*[*intro*, *simp*]: $i \in Basis \Longrightarrow 0 \leq i$
by (*auto simp*: *eucl-le inner-Basis*)

lemma *Sup-eq-maximum-componentwise*:
fixes *s*::*'a set*
assumes *i*: $\bigwedge b.\ b \in Basis \Longrightarrow X \cdot b = i\ b \cdot b$
assumes *sup*: $\bigwedge b\ x.\ b \in Basis \Longrightarrow x \in s \Longrightarrow x \cdot b \leq X \cdot b$
assumes *i-s*: $\bigwedge b.\ b \in Basis \Longrightarrow (i\ b \cdot b) \in (\lambda x.\ x \cdot b)\ `\ s$
shows *Sup s* $= X$
using *assms*
unfolding *eucl-Sup euclidean-representation-setsum*
by (*auto simp*: *Sup-class.SUP-def simp del*: *Sup-class.Sup-image-eq intro*!: *conditionally-complete-lattice-class*

lemma *Inf-eq-minimum-componentwise*:
assumes *i*: $\bigwedge b.\ b \in Basis \Longrightarrow X \cdot b = i\ b \cdot b$
assumes *sup*: $\bigwedge b\ x.\ b \in Basis \Longrightarrow x \in s \Longrightarrow X \cdot b \leq x \cdot b$
assumes *i-s*: $\bigwedge b.\ b \in Basis \Longrightarrow (i\ b \cdot b) \in (\lambda x.\ x \cdot b)\ `\ s$
shows *Inf s* $= X$
using *assms*
unfolding *eucl-Inf euclidean-representation-setsum*
by (*auto simp*: *Inf-class.INF-def simp del*: *Inf-class.Inf-image-eq intro*!: *conditionally-complete-lattice-class.cI*

end

lemma
compact-attains-Inf-componentwise:
fixes *b*::*'a*::*ordered-euclidean-space*
assumes $b \in Basis$ **assumes** $X \neq \{\}$ *compact X*
obtains *x* **where** $x \in X$ $x \cdot b = Inf\ X \cdot b$ $\bigwedge y.\ y \in X \Longrightarrow x \cdot b \leq y \cdot b$

proof *atomize-elim*
let *?proj* = $(\lambda x.\ x \cdot b)$ ‘ X
from *assms* **have** *compact ?proj ?proj* $\neq \{\}$
by (*auto intro*!: *compact-continuous-image continuous-intros*)
from *compact-attains-inf*[*OF this*]
obtain *s x*
where *s*: $s \in (\lambda x.\ x \cdot b)$ ‘ X $\bigwedge t.\ t \in (\lambda x.\ x \cdot b)$ ‘ $X \Longrightarrow s \leq t$
and *x*: $x \in X$ $s = x \cdot b$ $\bigwedge y.\ y \in X \Longrightarrow x \cdot b \leq y \cdot b$
by *auto*
hence *Inf ?proj* = $x \cdot b$
by (*auto intro*!: *conditionally-complete-lattice-class.cInf-eq-minimum simp del*: *Inf-class.Inf-image-eq*)
hence $x \cdot b = Inf\ X \cdot b$
by (*auto simp*: *eucl-Inf Inf-class.INF-def inner-setsum-left inner-Basis if-distrib* ‹$b \in Basis$› *setsum.delta*
simp del: *Inf-class.Inf-image-eq*
cong: *if-cong*)
with *x* **show** $\exists x.\ x \in X \wedge x \cdot b = Inf\ X \cdot b \wedge (\forall y.\ y \in X \longrightarrow x \cdot b \leq y \cdot b)$
by *blast*
qed

lemma
compact-attains-Sup-componentwise:
fixes *b*::*'a*::*ordered-euclidean-space*
assumes $b \in Basis$ **assumes** $X \neq \{\}$ *compact X*
obtains *x* **where** $x \in X$ $x \cdot b = Sup\ X \cdot b$ $\bigwedge y.\ y \in X \Longrightarrow y \cdot b \leq x \cdot b$
proof *atomize-elim*
let *?proj* = $(\lambda x.\ x \cdot b)$ ‘ X
from *assms* **have** *compact ?proj ?proj* $\neq \{\}$
by (*auto intro*!: *compact-continuous-image continuous-intros*)
from *compact-attains-sup*[*OF this*]
obtain *s x*
where *s*: $s \in (\lambda x.\ x \cdot b)$ ‘ X $\bigwedge t.\ t \in (\lambda x.\ x \cdot b)$ ‘ $X \Longrightarrow t \leq s$
and *x*: $x \in X$ $s = x \cdot b$ $\bigwedge y.\ y \in X \Longrightarrow y \cdot b \leq x \cdot b$
by *auto*
hence *Sup ?proj* = $x \cdot b$
by (*auto intro*!: *cSup-eq-maximum simp del*: *Sup-image-eq*)
hence $x \cdot b = Sup\ X \cdot b$
by (*auto simp*: *eucl-Sup*[**where** *'a*=*'a*] *SUP-def inner-setsum-left inner-Basis if-distrib* ‹$b \in Basis$› *setsum.delta*
simp del: *Sup-image-eq cong*: *if-cong*)
with *x* **show** $\exists x.\ x \in X \wedge x \cdot b = Sup\ X \cdot b \wedge (\forall y.\ y \in X \longrightarrow y \cdot b \leq x \cdot b)$
by *blast*
qed

lemma (**in** *order*) *atLeastatMost-empty'*[*simp*]:
$(\sim\ a <= b) \Longrightarrow \{a..b\} = \{\}$
by (*auto*)

instance *real* :: *ordered-euclidean-space*
by *standard* (*auto simp*: *INF-def SUP-def*)

lemma *in-Basis-prod-iff*:
fixes *i*::'*a*::*euclidean-space*∗'*b*::*euclidean-space*
shows $i \in Basis \longleftrightarrow fst\ i = 0 \wedge snd\ i \in Basis \vee snd\ i = 0 \wedge fst\ i \in Basis$
by (*cases i*) (*auto simp*: *Basis-prod-def*)

instantiation *prod* :: (*abs*, *abs*) *abs*
begin

definition $|x| = (|fst\ x|, |snd\ x|)$

instance ..

end

instance *prod* :: (*ordered-euclidean-space*, *ordered-euclidean-space*) *ordered-euclidean-space*
by *standard*
(*auto intro*!: *add-mono simp add*: *euclidean-representation-setsum'* *Ball-def* *inner-prod-def*
in-Basis-prod-iff inner-Basis-inf-left inner-Basis-sup-left inner-Basis-INF-left *Inf-prod-def*
inner-Basis-SUP-left Sup-prod-def less-prod-def less-eq-prod-def eucl-le[**where** 'a='a]
eucl-le[**where** 'a='b] *abs-prod-def abs-inner*)

Instantiation for intervals on *ordered-euclidean-space*

lemma
fixes *a* :: '*a*::*ordered-euclidean-space*
shows *cbox-interval*: *cbox a b* = {*a*..*b*}
and *interval-cbox*: {*a*..*b*} = *cbox a b*
and *eucl-le-atMost*: {*x*. ∀ *i*∈*Basis*. *x* · *i* <= *a* · *i*} = {..*a*}
and *eucl-le-atLeast*: {*x*. ∀ *i*∈*Basis*. *a* · *i* <= *x* · *i*} = {*a*..}
by (*auto simp*: *eucl-le*[**where** 'a='a] *eucl-less-def box-def cbox-def*)

lemma *closed-eucl-atLeastAtMost*[*simp*, *intro*]:
fixes *a* :: '*a*::*ordered-euclidean-space*
shows *closed* {*a*..*b*}
by (*simp add*: *cbox-interval*[*symmetric*] *closed-cbox*)

lemma *closed-eucl-atMost*[*simp*, *intro*]:
fixes *a* :: '*a*::*ordered-euclidean-space*
shows *closed* {..*a*}
by (*simp add*: *eucl-le-atMost*[*symmetric*])

lemma *closed-eucl-atLeast*[*simp*, *intro*]:
fixes *a* :: '*a*::*ordered-euclidean-space*
shows *closed* {*a*..}

by (*simp add*: *eucl-le-atLeast*[*symmetric*])

lemma *bounded-closed-interval*:
fixes *a* :: *'a*::*ordered-euclidean-space*
shows *bounded* {*a* .. *b*}
using *bounded-cbox*[*of a b*]
by (*metis interval-cbox*)

lemma *convex-closed-interval*:
fixes *a* :: *'a*::*ordered-euclidean-space*
shows *convex* {*a* .. *b*}
using *convex-box*[*of a b*]
by (*metis interval-cbox*)

lemma *image-smult-interval*:(λ*x*. *m* $*_R$ (*x*::-::*ordered-euclidean-space*)) ' {*a* .. *b*}
=
(*if* {*a* .. *b*} = {} *then* {} *else if* 0 ≤ *m then* {*m* $*_R$ *a* .. *m* $*_R$ *b*} *else* {*m* $*_R$ *b* .. *m* $*_R$ *a*})
using *image-smult-cbox*[*of m a b*]
by (*simp add*: *cbox-interval*)

lemma *is-interval-closed-interval*:
is-interval {*a* .. (*b*::*'a*::*ordered-euclidean-space*)}
by (*metis cbox-interval is-interval-cbox*)

lemma *compact-interval*:
fixes *a b*::*'a*::*ordered-euclidean-space*
shows *compact* {*a* .. *b*}
by (*metis compact-cbox interval-cbox*)

no-notation
eucl-less (**infix** <*e* 50)

lemma *One-nonneg*: 0 ≤ (∑ *Basis*::*'a*::*ordered-euclidean-space*)
by (*auto intro*: *setsum-nonneg*)

lemma *content-closed-interval*:
fixes *a* :: *'a*::*ordered-euclidean-space*
assumes *a* ≤ *b*
shows *content* {*a* .. *b*} = (∏ *i*∈*Basis*. *b*•*i* − *a*•*i*)
using *content-cbox*[*of a b*] *assms*
by (*simp add*: *cbox-interval eucl-le*[**where** *'a*=*'a*])

lemma *integrable-const-ivl*[*intro*]:
fixes *a*::*'a*::*ordered-euclidean-space*
shows (λ*x*. *c*) *integrable-on* {*a* .. *b*}
unfolding *cbox-interval*[*symmetric*]
by (*rule integrable-const*)

lemma *integrable-on-subinterval*:
fixes *f* :: *'n*::*ordered-euclidean-space* ⇒ *'a*::*banach*
assumes *f integrable-on s*
and {*a* .. *b*} ⊆ *s*
shows *f integrable-on* {*a* .. *b*}
using *integrable-on-subcbox*[*of f s a b*] *assms*
by (*simp add*: *cbox-interval*)

lemma
fixes *a b*::*'a*::*ordered-euclidean-space*
shows *bdd-above-cbox*[*intro*, *simp*]: *bdd-above* (*cbox a b*)
and *bdd-below-cbox*[*intro*, *simp*]: *bdd-below* (*cbox a b*)
and *bdd-above-box*[*intro*, *simp*]: *bdd-above* (*box a b*)
and *bdd-below-box*[*intro*, *simp*]: *bdd-below* (*box a b*)
unfolding *atomize-conj*
by (*metis bdd-above-Icc bdd-above-mono bdd-below-Icc bdd-below-mono bounded-box bounded-subset-cbox interval-cbox*)

instantiation *vec* :: (*ordered-euclidean-space*, *finite*) *ordered-euclidean-space*
begin

definition *inf x y* = (*χ i*. *inf* (*x* $ *i*) (*y* $ *i*))
definition *sup x y* = (*χ i*. *sup* (*x* $ *i*) (*y* $ *i*))
definition *Inf X* = (*χ i*. (*INF x*:*X*. *x* $ *i*))
definition *Sup X* = (*χ i*. (*SUP x*:*X*. *x* $ *i*))
definition |*x*| = (*χ i*. |*x* $ *i*|)

instance
apply *standard*
unfolding *euclidean-representation-setsum'*
apply (*auto simp*: *less-eq-vec-def inf-vec-def sup-vec-def Inf-vec-def Sup-vec-def inner-axis*
Basis-vec-def inner-Basis-inf-left inner-Basis-sup-left inner-Basis-INF-left
inner-Basis-SUP-left eucl-le[**where** *'a*=*'a*] *less-le-not-le abs-vec-def abs-inner*)
done

end

end

46 Non-negative, non-positive integers and reals

theory *Nonpos-Ints*
imports *Complex-Main*
begin

46.1 Non-positive integers

The set of non-positive integers on a ring. (in analogy to the set of non-negative integers $\mathbb{N}$) This is useful e.g. for the Gamma function.

definition *nonpos-Ints* ($\mathbb{Z}_{\leq 0}$) **where** $\mathbb{Z}_{\leq 0} = \{of\text{-}int\ n\ |n.\ n \leq 0\}$

lemma *zero-in-nonpos-Ints* [*simp*,*intro*]: $0 \in \mathbb{Z}_{\leq 0}$
unfolding *nonpos-Ints-def* **by** (*auto intro*!: *exI*[*of - 0*::*int*])

lemma *neg-one-in-nonpos-Ints* [*simp*,*intro*]: $-1 \in \mathbb{Z}_{\leq 0}$
unfolding *nonpos-Ints-def* **by** (*auto intro*!: *exI*[*of - −1*::*int*])

lemma *neg-numeral-in-nonpos-Ints* [*simp*,*intro*]: $-numeral\ n \in \mathbb{Z}_{\leq 0}$
unfolding *nonpos-Ints-def* **by** (*auto intro*!: *exI*[*of - −numeral n*::*int*])

lemma *one-notin-nonpos-Ints* [*simp*]: $(1 :: 'a :: ring\text{-}char\text{-}0) \notin \mathbb{Z}_{\leq 0}$
by (*auto simp*: *nonpos-Ints-def*)

lemma *numeral-notin-nonpos-Ints* [*simp*]: $(numeral\ n :: 'a :: ring\text{-}char\text{-}0) \notin \mathbb{Z}_{\leq 0}$
by (*auto simp*: *nonpos-Ints-def*)

lemma *minus-of-nat-in-nonpos-Ints* [*simp*, *intro*]: $-\ of\text{-}nat\ n \in \mathbb{Z}_{\leq 0}$
proof −
have − *of-nat n* = *of-int* (−*int n*) **by** *simp*
also have −*int n* ≤ *0* **by** *simp*
hence *of-int* (−*int n*) ∈ $\mathbb{Z}_{\leq 0}$ **unfolding** *nonpos-Ints-def* **by** *blast*
finally show *?thesis* **.**
qed

lemma *of-nat-in-nonpos-Ints-iff*: $(of\text{-}nat\ n :: 'a :: \{ring\text{-}1, ring\text{-}char\text{-}0\}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n = 0$
proof
assume (*of-nat n* :: *'a*) ∈ $\mathbb{Z}_{\leq 0}$
then obtain *m* **where** *of-nat n* = (*of-int m* :: *'a*) *m* ≤ *0* **by** (*auto simp*: *nonpos-Ints-def*)
hence (*of-int m* :: *'a*) = *of-nat n* **by** *simp*
also have ... = *of-int* (*int n*) **by** *simp*
finally have *m* = *int n* **by** (*subst* (*asm*) *of-int-eq-iff*)
with ‹*m* ≤ *0*› **show** *n* = *0* **by** *auto*
qed *simp*

lemma *nonpos-Ints-of-int*: $n \leq 0 \Longrightarrow of\text{-}int\ n \in \mathbb{Z}_{\leq 0}$
unfolding *nonpos-Ints-def* **by** *blast*

lemma *nonpos-IntsI*:
$x \in \mathbb{Z} \Longrightarrow x \leq 0 \Longrightarrow (x :: 'a :: linordered\text{-}idom) \in \mathbb{Z}_{\leq 0}$
using *assms* **unfolding** *nonpos-Ints-def Ints-def* **by** *auto*

lemma *nonpos-Ints-subset-Ints*: $\mathbb{Z}_{\leq 0} \subseteq \mathbb{Z}$

unfolding *nonpos-Ints-def Ints-def* **by** *blast*

lemma *nonpos-Ints-nonpos* [*dest*]: $x \in \mathbb{Z}_{\leq 0} \Longrightarrow x \leq (0 :: 'a :: linordered\text{-}idom)$
unfolding *nonpos-Ints-def* **by** *auto*

lemma *nonpos-Ints-Int* [*dest*]: $x \in \mathbb{Z}_{\leq 0} \Longrightarrow x \in \mathbb{Z}$
unfolding *nonpos-Ints-def Ints-def* **by** *blast*

lemma *nonpos-Ints-cases*:
assumes $x \in \mathbb{Z}_{\leq 0}$
obtains n **where** $x = of\text{-}int\ n\ n \leq 0$
using *assms* **unfolding** *nonpos-Ints-def* **by** (*auto elim*!: *Ints-cases*)

lemma *nonpos-Ints-cases'*:
assumes $x \in \mathbb{Z}_{\leq 0}$
obtains n **where** $x = -of\text{-}nat\ n$
proof −
from *assms* **obtain** m **where** $x = of\text{-}int\ m$ **and** m: $m \leq 0$ **by** (*auto elim*!: *nonpos-Ints-cases*)
hence $x = -\ of\text{-}int\ (-m)$ **by** *auto*
also from m **have** $(of\text{-}int\ (-m) :: 'a) = of\text{-}nat\ (nat\ (-m))$ **by** *simp-all*
finally show *?thesis* **by** (*rule that*)
qed

lemma *of-real-in-nonpos-Ints-iff*: $(of\text{-}real\ x :: 'a :: real\text{-}algebra\text{-}1) \in \mathbb{Z}_{\leq 0} \longleftrightarrow x \in \mathbb{Z}_{\leq 0}$
proof
assume $of\text{-}real\ x \in (\mathbb{Z}_{\leq 0} :: 'a\ set)$
then obtain n **where** $(of\text{-}real\ x :: 'a) = of\text{-}int\ n\ n \leq 0$ **by** (*erule nonpos-Ints-cases*)
note ‹*of-real x = of-int n*›
also have $of\text{-}int\ n = of\text{-}real\ (of\text{-}int\ n)$ **by** (*rule of-real-of-int-eq* [*symmetric*])
finally have $x = of\text{-}int\ n$ **by** (*subst* (*asm*) *of-real-eq-iff*)
with ‹$n \leq 0$› **show** $x \in \mathbb{Z}_{\leq 0}$ **by** (*simp add*: *nonpos-Ints-of-int*)
qed (*auto elim*!: *nonpos-Ints-cases intro*!: *nonpos-Ints-of-int*)

lemma *nonpos-Ints-altdef*: $\mathbb{Z}_{\leq 0} = \{n \in \mathbb{Z}.\ (n :: 'a :: linordered\text{-}idom) \leq 0\}$
by (*auto intro*!: *nonpos-IntsI elim*!: *nonpos-Ints-cases*)

lemma *uminus-in-Nats-iff*: $-x \in \mathbb{N} \longleftrightarrow x \in \mathbb{Z}_{\leq 0}$
proof
assume $-x \in \mathbb{N}$
then obtain n **where** $n \geq 0\ -x = of\text{-}int\ n$ **by** (*auto simp*: *Nats-altdef1*)
hence $-n \leq 0\ x = of\text{-}int\ (-n)$ **by** (*simp-all add*: *eq-commute minus-equation-iff* [*of x*])
thus $x \in \mathbb{Z}_{\leq 0}$ **unfolding** *nonpos-Ints-def* **by** *blast*
next
assume $x \in \mathbb{Z}_{\leq 0}$
then obtain n **where** $n \leq 0\ x = of\text{-}int\ n$ **by** (*auto simp*: *nonpos-Ints-def*)
hence $-n \geq 0\ -x = of\text{-}int\ (-n)$ **by** (*simp-all add*: *eq-commute minus-equation-iff* [*of*

x])
thus $-x \in \mathbb{N}$ **unfolding** *Nats-altdef1* **by** *blast*
qed

lemma *uminus-in-nonpos-Ints-iff*: $-x \in \mathbb{Z}_{\leq 0} \longleftrightarrow x \in \mathbb{N}$
using *uminus-in-Nats-iff*[*of* $-x$] **by** *simp*

lemma *nonpos-Ints-mult*: $x \in \mathbb{Z}_{\leq 0} \Longrightarrow y \in \mathbb{Z}_{\leq 0} \Longrightarrow x * y \in \mathbb{N}$
using *Nats-mult*[*of* $-x$ $-y$] **by** (*simp add*: *uminus-in-Nats-iff*)

lemma *Nats-mult-nonpos-Ints*: $x \in \mathbb{N} \Longrightarrow y \in \mathbb{Z}_{\leq 0} \Longrightarrow x * y \in \mathbb{Z}_{\leq 0}$
using *Nats-mult*[*of* x $-y$] **by** (*simp add*: *uminus-in-Nats-iff*)

lemma *nonpos-Ints-mult-Nats*:
$x \in \mathbb{Z}_{\leq 0} \Longrightarrow y \in \mathbb{N} \Longrightarrow x * y \in \mathbb{Z}_{\leq 0}$
using *Nats-mult*[*of* $-x$ y] **by** (*simp add*: *uminus-in-Nats-iff*)

lemma *nonpos-Ints-add*:
$x \in \mathbb{Z}_{\leq 0} \Longrightarrow y \in \mathbb{Z}_{\leq 0} \Longrightarrow x + y \in \mathbb{Z}_{\leq 0}$
using *Nats-add*[*of* $-x$ $-y$] *uminus-in-Nats-iff*[*of* $y+x$, *simplified minus-add*]
by (*simp add*: *uminus-in-Nats-iff add.commute*)

lemma *nonpos-Ints-diff-Nats*:
$x \in \mathbb{Z}_{\leq 0} \Longrightarrow y \in \mathbb{N} \Longrightarrow x - y \in \mathbb{Z}_{\leq 0}$
using *Nats-add*[*of* $-x$ y] *uminus-in-Nats-iff*[*of* $x-y$, *simplified minus-add*]
by (*simp add*: *uminus-in-Nats-iff add.commute*)

lemma *Nats-diff-nonpos-Ints*:
$x \in \mathbb{N} \Longrightarrow y \in \mathbb{Z}_{\leq 0} \Longrightarrow x - y \in \mathbb{N}$
using *Nats-add*[*of* x $-y$] **by** (*simp add*: *uminus-in-Nats-iff add.commute*)

lemma *plus-of-nat-eq-0-imp*: $z + \mathit{of\text{-}nat}\ n = 0 \Longrightarrow z \in \mathbb{Z}_{\leq 0}$
proof −
assume $z + \mathit{of\text{-}nat}\ n = 0$
hence *A*: $z = -\ \mathit{of\text{-}nat}\ n$ **by** (*simp add*: *eq-neg-iff-add-eq-0*)
show $z \in \mathbb{Z}_{\leq 0}$ **by** (*subst A*) *simp*
qed

46.2 Non-negative reals

definition *nonneg-Reals* :: *'a*::*real-algebra-1 set* $(\mathbb{R}_{\geq 0})$
where $\mathbb{R}_{\geq 0} = \{\mathit{of\text{-}real}\ r \mid r.\ r \geq 0\}$

lemma *nonneg-Reals-of-real-iff* [*simp*]: *of-real* $r \in \mathbb{R}_{\geq 0} \longleftrightarrow r \geq 0$
by (*force simp add*: *nonneg-Reals-def*)

lemma *nonneg-Reals-subset-Reals*: $\mathbb{R}_{\geq 0} \subseteq \mathbb{R}$
unfolding *nonneg-Reals-def Reals-def* **by** *blast*

lemma *nonneg-Reals-Real* [*dest*]: $x \in \mathbb{R}_{\geq 0} \Longrightarrow x \in \mathbb{R}$
unfolding *nonneg-Reals-def Reals-def* **by** *blast*

lemma *nonneg-Reals-of-nat-I* [*simp*]: *of-nat n* $\in \mathbb{R}_{\geq 0}$
by (*metis nonneg-Reals-of-real-iff of-nat-0-le-iff of-real-of-nat-eq*)

lemma *nonneg-Reals-cases*:
assumes $x \in \mathbb{R}_{\geq 0}$
obtains *r* **where** $x =$ *of-real r* $r \geq 0$
using *assms* **unfolding** *nonneg-Reals-def* **by** (*auto elim*!: *Reals-cases*)

lemma *nonneg-Reals-zero-I* [*simp*]: *0* $\in \mathbb{R}_{\geq 0}$
unfolding *nonneg-Reals-def* **by** *auto*

lemma *nonneg-Reals-one-I* [*simp*]: *1* $\in \mathbb{R}_{\geq 0}$
by (*metis* (*mono-tags*, *lifting*) *nonneg-Reals-of-nat-I of-nat-1*)

lemma *nonneg-Reals-minus-one-I* [*simp*]: $-1 \notin \mathbb{R}_{\geq 0}$
by (*metis nonneg-Reals-of-real-iff le-minus-one-simps*(*3*) *of-real-1 of-real-def real-vector.scale-minus-left*)

lemma *nonneg-Reals-numeral-I* [*simp*]: *numeral w* $\in \mathbb{R}_{\geq 0}$
by (*metis* (*no-types*) *nonneg-Reals-of-nat-I of-nat-numeral*)

lemma *nonneg-Reals-minus-numeral-I* [*simp*]: $-$ *numeral w* $\notin \mathbb{R}_{\geq 0}$
using *nonneg-Reals-of-real-iff not-zero-le-neg-numeral* **by** *fastforce*

lemma *nonneg-Reals-add-I* [*simp*]: $[\![a \in \mathbb{R}_{\geq 0};\ b \in \mathbb{R}_{\geq 0}]\!] \Longrightarrow a + b \in \mathbb{R}_{\geq 0}$
apply (*simp add*: *nonneg-Reals-def*)
apply *clarify*
apply (*rename-tac r s*)
apply (*rule-tac x*=*r*+*s* **in** *exI*, *auto*)
done

lemma *nonneg-Reals-mult-I* [*simp*]: $[\![a \in \mathbb{R}_{\geq 0};\ b \in \mathbb{R}_{\geq 0}]\!] \Longrightarrow a * b \in \mathbb{R}_{\geq 0}$
unfolding *nonneg-Reals-def* **by** (*auto simp*: *of-real-def*)

lemma *nonneg-Reals-inverse-I* [*simp*]:
fixes *a* :: *'a*::*real-div-algebra*
shows $a \in \mathbb{R}_{\geq 0} \Longrightarrow$ *inverse a* $\in \mathbb{R}_{\geq 0}$
by (*simp add*: *nonneg-Reals-def image-iff*) (*metis inverse-nonnegative-iff-nonnegative of-real-inverse*)

lemma *nonneg-Reals-divide-I* [*simp*]:
fixes *a* :: *'a*::*real-div-algebra*
shows $[\![a \in \mathbb{R}_{\geq 0};\ b \in \mathbb{R}_{\geq 0}]\!] \Longrightarrow a\ /\ b \in \mathbb{R}_{\geq 0}$
by (*simp add*: *divide-inverse*)

lemma *nonneg-Reals-pow-I* [*simp*]: $a \in \mathbb{R}_{\geq 0} \Longrightarrow a\hat{}n \in \mathbb{R}_{\geq 0}$
by (*induction n*) *auto*

lemma *complex-nonneg-Reals-iff*: $z \in \mathbb{R}_{\geq 0} \longleftrightarrow Re\ z \geq 0 \wedge Im\ z = 0$
by (*auto simp*: *nonneg-Reals-def*) (*metis complex-of-real-def complex-surj*)

lemma *ii-not-nonneg-Reals* [*iff*]: $\mathrm{i} \notin \mathbb{R}_{\geq 0}$
by (*simp add*: *complex-nonneg-Reals-iff*)

46.3 Non-positive reals

definition *nonpos-Reals* :: *'a*::*real-algebra-1 set* ($\mathbb{R}_{\leq 0}$)
where $\mathbb{R}_{\leq 0} = \{of\text{-}real\ r \mid r.\ r \leq 0\}$

lemma *nonpos-Reals-of-real-iff* [*simp*]: *of-real* $r \in \mathbb{R}_{\leq 0} \longleftrightarrow r \leq 0$
by (*force simp add*: *nonpos-Reals-def*)

lemma *nonpos-Reals-subset-Reals*: $\mathbb{R}_{\leq 0} \subseteq \mathbb{R}$
unfolding *nonpos-Reals-def Reals-def* **by** *blast*

lemma *nonpos-Ints-subset-nonpos-Reals*: $\mathbb{Z}_{\leq 0} \subseteq \mathbb{R}_{\leq 0}$
by (*metis nonpos-Ints-cases nonpos-Ints-nonpos nonpos-Ints-of-int nonpos-Reals-of-real-iff of-real-of-int-eq subsetI*)

lemma *nonpos-Reals-of-nat-iff* [*simp*]: *of-nat* $n \in \mathbb{R}_{\leq 0} \longleftrightarrow n{=}0$
by (*metis nonpos-Reals-of-real-iff of-nat-le-0-iff of-real-of-nat-eq*)

lemma *nonpos-Reals-Real* [*dest*]: $x \in \mathbb{R}_{\leq 0} \Longrightarrow x \in \mathbb{R}$
unfolding *nonpos-Reals-def Reals-def* **by** *blast*

lemma *nonpos-Reals-cases*:
assumes $x \in \mathbb{R}_{\leq 0}$
obtains r **where** $x = of\text{-}real\ r\ r \leq 0$
using *assms* **unfolding** *nonpos-Reals-def* **by** (*auto elim*!: *Reals-cases*)

lemma *uminus-nonneg-Reals-iff* [*simp*]: $-x \in \mathbb{R}_{\geq 0} \longleftrightarrow x \in \mathbb{R}_{\leq 0}$
apply (*auto simp*: *nonpos-Reals-def nonneg-Reals-def*)
apply (*metis nonpos-Reals-of-real-iff minus-minus neg-le-0-iff-le of-real-minus*)
apply (*metis neg-0-le-iff-le of-real-minus*)
done

lemma *uminus-nonpos-Reals-iff* [*simp*]: $-x \in \mathbb{R}_{\leq 0} \longleftrightarrow x \in \mathbb{R}_{\geq 0}$
by (*metis* (*no-types*) *minus-minus uminus-nonneg-Reals-iff*)

lemma *nonpos-Reals-zero-I* [*simp*]: $0 \in \mathbb{R}_{\leq 0}$
unfolding *nonpos-Reals-def* **by** *force*

lemma *nonpos-Reals-one-I* [*simp*]: $1 \notin \mathbb{R}_{\leq 0}$
using *nonneg-Reals-minus-one-I uminus-nonneg-Reals-iff* **by** *blast*

lemma *nonpos-Reals-numeral-I* [*simp*]: *numeral* $w \notin \mathbb{R}_{\leq 0}$

using *nonneg-Reals-minus-numeral-I uminus-nonneg-Reals-iff* **by** *blast*

lemma *nonpos-Reals-add-I* [*simp*]: $[\![a \in \mathbb{R}_{\leq 0};\ b \in \mathbb{R}_{\leq 0}]\!] \Longrightarrow a + b \in \mathbb{R}_{\leq 0}$
by (*metis nonneg-Reals-add-I add-uminus-conv-diff minus-diff-eq minus-minus uminus-nonpos-Reals-iff*)

lemma *nonpos-Reals-mult-I1*: $[\![a \in \mathbb{R}_{\geq 0};\ b \in \mathbb{R}_{\leq 0}]\!] \Longrightarrow a * b \in \mathbb{R}_{\leq 0}$
by (*metis nonneg-Reals-mult-I mult-minus-right uminus-nonneg-Reals-iff*)

lemma *nonpos-Reals-mult-I2*: $[\![a \in \mathbb{R}_{\leq 0};\ b \in \mathbb{R}_{\geq 0}]\!] \Longrightarrow a * b \in \mathbb{R}_{\leq 0}$
by (*metis nonneg-Reals-mult-I mult-minus-left uminus-nonneg-Reals-iff*)

lemma *nonpos-Reals-mult-of-nat-iff*:
fixes *a*:: *'a* :: *real-div-algebra* **shows** *a* * *of-nat n* $\in \mathbb{R}_{\leq 0} \longleftrightarrow a \in \mathbb{R}_{\leq 0} \vee n=0$
apply (*auto intro*: *nonpos-Reals-mult-I2*)
apply (*auto simp*: *nonpos-Reals-def*)
apply (*rule-tac x=r/n* **in** *exI*)
apply (*auto simp*: *divide-simps*)
done

lemma *nonpos-Reals-inverse-I*:
fixes *a* :: *'a*::*real-div-algebra*
shows $a \in \mathbb{R}_{\leq 0} \Longrightarrow$ *inverse a* $\in \mathbb{R}_{\leq 0}$
using *nonneg-Reals-inverse-I uminus-nonneg-Reals-iff* **by** *fastforce*

lemma *nonpos-Reals-divide-I1*:
fixes *a* :: *'a*::*real-div-algebra*
shows $[\![a \in \mathbb{R}_{\geq 0};\ b \in \mathbb{R}_{\leq 0}]\!] \Longrightarrow a\ /\ b \in \mathbb{R}_{\leq 0}$
by (*simp add*: *nonpos-Reals-inverse-I nonpos-Reals-mult-I1 divide-inverse*)

lemma *nonpos-Reals-divide-I2*:
fixes *a* :: *'a*::*real-div-algebra*
shows $[\![a \in \mathbb{R}_{\leq 0};\ b \in \mathbb{R}_{\geq 0}]\!] \Longrightarrow a\ /\ b \in \mathbb{R}_{\leq 0}$
by (*metis nonneg-Reals-divide-I minus-divide-left uminus-nonneg-Reals-iff*)

lemma *nonpos-Reals-divide-of-nat-iff*:
fixes *a*:: *'a* :: *real-div-algebra* **shows** *a* / *of-nat n* $\in \mathbb{R}_{\leq 0} \longleftrightarrow a \in \mathbb{R}_{\leq 0} \vee n=0$
apply (*auto intro*: *nonpos-Reals-divide-I2*)
apply (*auto simp*: *nonpos-Reals-def*)
apply (*rule-tac x=r*n* **in** *exI*)
apply (*auto simp*: *divide-simps mult-le-0-iff*)
done

lemma *nonpos-Reals-pow-I*: $[\![a \in \mathbb{R}_{\leq 0};\ odd\ n]\!] \Longrightarrow a\hat{}n \in \mathbb{R}_{\leq 0}$
by (*metis nonneg-Reals-pow-I power-minus-odd uminus-nonneg-Reals-iff*)

lemma *complex-nonpos-Reals-iff*: $z \in \mathbb{R}_{\leq 0} \longleftrightarrow Re\ z \leq 0 \wedge Im\ z = 0$
using *complex-is-Real-iff* **by** (*force simp add*: *nonpos-Reals-def*)

lemma *ii-not-nonpos-Reals* [*iff*]: $\mathrm{i} \notin \mathbb{R}_{\leq 0}$
by (*simp add*: *complex-nonpos-Reals-iff*)

end

47 Complex Analysis Basics

theory *Complex-Analysis-Basics*
imports *Cartesian-Euclidean-Space* ~~/*src*/*HOL*/*Library*/*Nonpos-Ints*
begin

47.1 General lemmas

lemma *nonneg-Reals-cmod-eq-Re*: $z \in \mathbb{R}_{\geq 0} \Longrightarrow$ *norm* $z =$ *Re* z
by (*simp add*: *complex-nonneg-Reals-iff cmod-eq-Re*)

lemma *has-derivative-mult-right*:
fixes *c*:: *'a* :: *real-normed-algebra*
shows ((*op* ∗ *c*) *has-derivative* (*op* ∗ *c*)) *F*
by (*rule has-derivative-mult-right* [*OF has-derivative-id*])

lemma *has-derivative-of-real*[*derivative-intros*, *simp*]:
(*f has-derivative f'*) *F* ⟹ ((λ*x*. *of-real* (*f x*)) *has-derivative* (λ*x*. *of-real* (*f' x*)))
F
using *bounded-linear.has-derivative*[*OF bounded-linear-of-real*] .

lemma *has-vector-derivative-real-complex*:
DERIV f (*of-real a*) :> *f'* ⟹ ((λ*x*. *f* (*of-real x*)) *has-vector-derivative f'*) (*at a*
within s)
using *has-derivative-compose*[*of of-real of-real a - f op* ∗ *f'*]
by (*simp add*: *scaleR-conv-of-real ac-simps has-vector-derivative-def has-field-derivative-def*)

lemma *fact-cancel*:
fixes *c* :: *'a*::*real-field*
shows *of-nat* (*Suc n*) ∗ *c* / (*fact* (*Suc n*)) = *c* / (*fact n*)
by (*simp add*: *of-nat-mult del*: *of-nat-Suc times-nat.simps*)

lemma *bilinear-times*:
fixes *c*::*'a*::*real-algebra* **shows** *bilinear* (λ*x y*::*'a*. *x*∗*y*)
by (*auto simp*: *bilinear-def distrib-left distrib-right intro*!: *linearI*)

lemma *linear-cnj*: *linear cnj*
using *bounded-linear.linear*[*OF bounded-linear-cnj*] .

lemma *tendsto-Re-upper*:
assumes ~ (*trivial-limit F*)
(*f* ⟶ *l*) *F*
eventually (λ*x*. *Re*(*f x*) ≤ *b*) *F*
shows *Re*(*l*) ≤ *b*

by (*metis assms tendsto-le* [*OF - tendsto-const*] *tendsto-Re*)

lemma *tendsto-Re-lower*:
assumes ~ (*trivial-limit F*)
(*f* ⟶ *l*) *F*
eventually (λ*x*. *b* ≤ *Re*(*f x*)) *F*
shows *b* ≤ *Re*(*l*)
by (*metis assms tendsto-le* [*OF - - tendsto-const*] *tendsto-Re*)

lemma *tendsto-Im-upper*:
assumes ~ (*trivial-limit F*)
(*f* ⟶ *l*) *F*
eventually (λ*x*. *Im*(*f x*) ≤ *b*) *F*
shows *Im*(*l*) ≤ *b*
by (*metis assms tendsto-le* [*OF - tendsto-const*] *tendsto-Im*)

lemma *tendsto-Im-lower*:
assumes ~ (*trivial-limit F*)
(*f* ⟶ *l*) *F*
eventually (λ*x*. *b* ≤ *Im*(*f x*)) *F*
shows *b* ≤ *Im*(*l*)
by (*metis assms tendsto-le* [*OF - - tendsto-const*] *tendsto-Im*)

lemma *lambda-zero*: (λ*h*::'*a*::*mult-zero*. *0*) = *op* ∗ *0*
by *auto*

lemma *lambda-one*: (λ*x*::'*a*::*monoid-mult*. *x*) = *op* ∗ *1*
by *auto*

lemma *continuous-mult-left*:
fixes *c*::'*a*::*real-normed-algebra*
shows *continuous F f* ⟹ *continuous F* (λ*x*. *c* ∗ *f x*)
by (*rule continuous-mult* [*OF continuous-const*])

lemma *continuous-mult-right*:
fixes *c*::'*a*::*real-normed-algebra*
shows *continuous F f* ⟹ *continuous F* (λ*x*. *f x* ∗ *c*)
by (*rule continuous-mult* [*OF - continuous-const*])

lemma *continuous-on-mult-left*:
fixes *c*::'*a*::*real-normed-algebra*
shows *continuous-on s f* ⟹ *continuous-on s* (λ*x*. *c* ∗ *f x*)
by (*rule continuous-on-mult* [*OF continuous-on-const*])

lemma *continuous-on-mult-right*:
fixes *c*::'*a*::*real-normed-algebra*
shows *continuous-on s f* ⟹ *continuous-on s* (λ*x*. *f x* ∗ *c*)
by (*rule continuous-on-mult* [*OF - continuous-on-const*])

lemma *uniformly-continuous-on-cmul-right* [*continuous-intros*]:
fixes f :: *'a*::*real-normed-vector* ⇒ *'b*::*real-normed-algebra*
shows *uniformly-continuous-on s f* ⟹ *uniformly-continuous-on s* (λ*x*. *f x* ∗ *c*)
using *bounded-linear.uniformly-continuous-on*[*OF bounded-linear-mult-left*] .

lemma *uniformly-continuous-on-cmul-left*[*continuous-intros*]:
fixes f :: *'a*::*real-normed-vector* ⇒ *'b*::*real-normed-algebra*
assumes *uniformly-continuous-on s f*
shows *uniformly-continuous-on s* (λ*x*. *c* ∗ *f x*)
by (*metis assms bounded-linear.uniformly-continuous-on bounded-linear-mult-right*)

lemma *continuous-within-norm-id* [*continuous-intros*]: *continuous* (*at x within S*) *norm*
by (*rule continuous-norm* [*OF continuous-ident*])

lemma *continuous-on-norm-id* [*continuous-intros*]: *continuous-on S norm*
by (*intro continuous-on-id continuous-on-norm*)

47.2 DERIV stuff

lemma *DERIV-zero-connected-constant*:
fixes f :: *'a*::{*real-normed-field*,*euclidean-space*} ⇒ *'a*
assumes *connected s*
and *open s*
and *finite k*
and *continuous-on s f*
and ∀ *x*∈(*s* − *k*). *DERIV f x* :> *0*
obtains *c* **where** ⋀*x*. *x* ∈ *s* ⟹ *f*(*x*) = *c*
using *has-derivative-zero-connected-constant* [*OF assms*(*1*−*4*)] *assms*
by (*metis DERIV-const has-derivative-const Diff-iff at-within-open frechet-derivative-at has-field-derivative-def*)

lemma *DERIV-zero-constant*:
fixes f :: *'a*::{*real-normed-field*, *real-inner*} ⇒ *'a*
shows ⟦*convex s*;
⋀*x*. *x*∈*s* ⟹ (*f has-field-derivative 0*) (*at x within s*)⟧
⟹ ∃ *c*. ∀ *x* ∈ *s*. *f*(*x*) = *c*
by (*auto simp*: *has-field-derivative-def lambda-zero intro*: *has-derivative-zero-constant*)

lemma *DERIV-zero-unique*:
fixes f :: *'a*::{*real-normed-field*, *real-inner*} ⇒ *'a*
assumes *convex s*
and *d0*: ⋀*x*. *x*∈*s* ⟹ (*f has-field-derivative 0*) (*at x within s*)
and *a* ∈ *s*
and *x* ∈ *s*
shows *f x* = *f a*
by (*rule has-derivative-zero-unique* [*OF assms*(*1*) - *assms*(*4*,*3*)])
(*metis d0 has-field-derivative-imp-has-derivative lambda-zero*)

lemma *DERIV-zero-connected-unique*:
fixes $f :: 'a::\{real\text{-}normed\text{-}field, real\text{-}inner\} \Rightarrow 'a$
assumes *connected s*
and *open s*
and *d0*: $\bigwedge x.\ x \in s \Longrightarrow DERIV\ f\ x :> 0$
and $a \in s$
and $x \in s$
shows $f\ x = f\ a$
by (*rule has-derivative-zero-unique-connected* [*OF assms*(2,1) - *assms*(5,4)])
(*metis has-field-derivative-def lambda-zero d0*)

lemma *DERIV-transform-within*:
assumes (*f has-field-derivative f'*) (*at a within s*)
and $0 < d$ $a \in s$
and $\bigwedge x.\ x \in s \Longrightarrow dist\ x\ a < d \Longrightarrow f\ x = g\ x$
shows (*g has-field-derivative f'*) (*at a within s*)
using *assms* **unfolding** *has-field-derivative-def*
by (*blast intro*: *has-derivative-transform-within*)

lemma *DERIV-transform-within-open*:
assumes *DERIV f a :> f'*
and *open s* $a \in s$
and $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows *DERIV g a :> f'*
using *assms* **unfolding** *has-field-derivative-def*
by (*metis has-derivative-transform-within-open*)

lemma *DERIV-transform-at*:
assumes *DERIV f a :> f'*
and $0 < d$
and $\bigwedge x.\ dist\ x\ a < d \Longrightarrow f\ x = g\ x$
shows *DERIV g a :> f'*
by (*blast intro*: *assms DERIV-transform-within*)

lemma *DERIV-zero-UNIV-unique*:
fixes $f :: 'a::\{real\text{-}normed\text{-}field, real\text{-}inner\} \Rightarrow 'a$
shows $(\bigwedge x.\ DERIV\ f\ x :> 0) \Longrightarrow f\ x = f\ a$
by (*metis DERIV-zero-unique UNIV-I assms convex-UNIV*)

47.3 Some limit theorems about real part of real series etc.

lemma *sums-vec-nth* :
assumes *f sums a*
shows $(\lambda x.\ f\ x\ \$\ i)$ *sums a* $\$\ i$
using *assms* **unfolding** *sums-def*
by (*auto dest*: *tendsto-vec-nth* [**where** $i=i$])

lemma *summable-vec-nth* :

assumes *summable f*
shows *summable* ($\lambda x.\ f\ x\ \$\ i$)
using *assms* **unfolding** *summable-def*
by (*blast intro*: *sums-vec-nth*)

47.4 Complex number lemmas

lemma
shows *open-halfspace-Re-lt*: *open* $\{z.\ Re(z) < b\}$
and *open-halfspace-Re-gt*: *open* $\{z.\ Re(z) > b\}$
and *closed-halfspace-Re-ge*: *closed* $\{z.\ Re(z) \geq b\}$
and *closed-halfspace-Re-le*: *closed* $\{z.\ Re(z) \leq b\}$
and *closed-halfspace-Re-eq*: *closed* $\{z.\ Re(z) = b\}$
and *open-halfspace-Im-lt*: *open* $\{z.\ Im(z) < b\}$
and *open-halfspace-Im-gt*: *open* $\{z.\ Im(z) > b\}$
and *closed-halfspace-Im-ge*: *closed* $\{z.\ Im(z) \geq b\}$
and *closed-halfspace-Im-le*: *closed* $\{z.\ Im(z) \leq b\}$
and *closed-halfspace-Im-eq*: *closed* $\{z.\ Im(z) = b\}$
by (*intro open-Collect-less closed-Collect-le closed-Collect-eq isCont-Re isCont-Im continuous-ident continuous-const*)+

lemma *closed-complex-Reals*: *closed* ($\mathbb{R}$ :: *complex set*)
proof −
have ($\mathbb{R}$ :: *complex set*) = $\{z.\ Im\ z = 0\}$
by (*auto simp*: *complex-is-Real-iff*)
then show *?thesis*
by (*metis closed-halfspace-Im-eq*)
qed

lemma *closed-Real-halfspace-Re-le*: *closed* ($\mathbb{R} \cap \{w.\ Re\ w \leq x\}$)
by (*simp add*: *closed-Int closed-complex-Reals closed-halfspace-Re-le*)

corollary *closed-nonpos-Reals-complex* [*simp*]: *closed* ($\mathbb{R}_{\leq 0}$ :: *complex set*)
proof −
have $\mathbb{R}_{\leq 0} = \mathbb{R} \cap \{z.\ Re(z) \leq 0\}$
using *complex-nonpos-Reals-iff complex-is-Real-iff* **by** *auto*
then show *?thesis*
by (*metis closed-Real-halfspace-Re-le*)
qed

lemma *closed-Real-halfspace-Re-ge*: *closed* ($\mathbb{R} \cap \{w.\ x \leq Re(w)\}$)
using *closed-halfspace-Re-ge*
by (*simp add*: *closed-Int closed-complex-Reals*)

corollary *closed-nonneg-Reals-complex* [*simp*]: *closed* ($\mathbb{R}_{\geq 0}$ :: *complex set*)
proof −
have $\mathbb{R}_{\geq 0} = \mathbb{R} \cap \{z.\ Re(z) \geq 0\}$
using *complex-nonneg-Reals-iff complex-is-Real-iff* **by** *auto*
then show *?thesis*

by (*metis closed-Real-halfspace-Re-ge*)
qed

lemma *closed-real-abs-le*: *closed* $\{w \in \mathbb{R}.\ |Re\ w| \leq r\}$
proof −
have $\{w \in \mathbb{R}.\ |Re\ w| \leq r\} = (\mathbb{R} \cap \{w.\ Re\ w \leq r\}) \cap (\mathbb{R} \cap \{w.\ Re\ w \geq -r\})$
by *auto*
then show *closed* $\{w \in \mathbb{R}.\ |Re\ w| \leq r\}$
by (*simp add*: *closed-Int closed-Real-halfspace-Re-ge closed-Real-halfspace-Re-le*)
qed

lemma *real-lim*:
fixes *l*::*complex*
assumes $(f \longrightarrow l)\ F$ **and** $\sim$(*trivial-limit F*) **and** *eventually P F* **and** $\bigwedge a.\ P\ a \Longrightarrow f\ a \in \mathbb{R}$
shows $l \in \mathbb{R}$
proof (*rule Lim-in-closed-set*[*OF closed-complex-Reals - assms*(*2*,*1*)])
show *eventually* $(\lambda x.\ f\ x \in \mathbb{R})\ F$
using *assms*(*3*, *4*) **by** (*auto intro*: *eventually-mono*)
qed

lemma *real-lim-sequentially*:
fixes *l*::*complex*
shows $(f \longrightarrow l)$ *sequentially* $\Longrightarrow (\exists N.\ \forall n \geq N.\ f\ n \in \mathbb{R}) \Longrightarrow l \in \mathbb{R}$
by (*rule real-lim* [**where** *F*=*sequentially*]) (*auto simp*: *eventually-sequentially*)

lemma *real-series*:
fixes *l*::*complex*
shows *f sums l* $\Longrightarrow (\bigwedge n.\ f\ n \in \mathbb{R}) \Longrightarrow l \in \mathbb{R}$
unfolding *sums-def*
by (*metis real-lim-sequentially setsum-in-Reals*)

lemma *Lim-null-comparison-Re*:
assumes *eventually* $(\lambda x.\ norm(f\,x) \leq Re(g\,x))\ F$ $(g \longrightarrow 0)\ F$ **shows** $(f \longrightarrow 0)\ F$
by (*rule Lim-null-comparison*[*OF assms*(*1*)] *tendsto-eq-intros assms*(*2*))+ *simp*

47.5 Holomorphic functions

Could be generalized to real normed fields, but in practice that would only include the reals

definition *complex-differentiable* :: [*complex* ⇒ *complex*, *complex filter*] ⇒ *bool*
(**infixr** (*complex'-differentiable*) *50*)
where *f complex-differentiable F* ≡ ∃*f'*. (*f has-field-derivative f'*) *F*

lemma *complex-differentiable-imp-continuous-at*:
f complex-differentiable (*at x within s*) ⟹ *continuous* (*at x within s*) *f*
by (*metis DERIV-continuous complex-differentiable-def*)

lemma *complex-differentiable-within-subset*:
  ⟦*f complex-differentiable* (*at x within s*); *t* ⊆ *s*⟧
  ⟹ *f complex-differentiable* (*at x within t*)
 by (*metis DERIV-subset complex-differentiable-def*)

lemma *complex-differentiable-at-within*:
  ⟦*f complex-differentiable* (*at x*)⟧
  ⟹ *f complex-differentiable* (*at x within s*)
 unfolding *complex-differentiable-def*
 by (*metis DERIV-subset top-greatest*)

lemma *complex-differentiable-linear* [*derivative-intros*]: (*op* ∗ *c*) *complex-differentiable F*
proof −
 show *?thesis*
  unfolding *complex-differentiable-def has-field-derivative-def mult-commute-abs*
  by (*force intro*: *has-derivative-mult-right*)
qed

lemma *complex-differentiable-const* [*derivative-intros*]: (λ*z*. *c*) *complex-differentiable F*
 unfolding *complex-differentiable-def has-field-derivative-def*
 by (*rule exI* [**where** *x=0*])
  (*metis has-derivative-const lambda-zero*)

lemma *complex-differentiable-ident* [*derivative-intros*]: (λ*z*. *z*) *complex-differentiable F*
 unfolding *complex-differentiable-def has-field-derivative-def*
 by (*rule exI* [**where** *x=1*])
  (*simp add*: *lambda-one* [*symmetric*])

lemma *complex-differentiable-id* [*derivative-intros*]: *id complex-differentiable F*
 unfolding *id-def* **by** (*rule complex-differentiable-ident*)

lemma *complex-differentiable-minus* [*derivative-intros*]:
 f complex-differentiable F ⟹ (λ*z*. − (*f z*)) *complex-differentiable F*
 using *assms* **unfolding** *complex-differentiable-def*
 by (*metis field-differentiable-minus*)

lemma *complex-differentiable-add* [*derivative-intros*]:
 assumes *f complex-differentiable F g complex-differentiable F*
  shows (λ*z*. *f z* + *g z*) *complex-differentiable F*
 using *assms* **unfolding** *complex-differentiable-def*
 by (*metis field-differentiable-add*)

lemma *complex-differentiable-setsum* [*derivative-intros*]:
 (⋀*i*. *i* ∈ *I* ⟹ (*f i*) *complex-differentiable F*) ⟹ (λ*z*. ∑ *i*∈*I*. *f i z*) *complex-differentiable F*
 by (*induct I rule*: *infinite-finite-induct*)

(*auto intro*: *complex-differentiable-add complex-differentiable-const*)

lemma *complex-differentiable-diff* [*derivative-intros*]:
assumes *f complex-differentiable F g complex-differentiable F*
shows (λ*z*. *f z* − *g z*) *complex-differentiable F*
using *assms* **unfolding** *complex-differentiable-def*
by (*metis field-differentiable-diff*)

lemma *complex-differentiable-inverse* [*derivative-intros*]:
assumes *f complex-differentiable* (*at a within s*) *f a* ≠ *0*
shows (λ*z*. *inverse* (*f z*)) *complex-differentiable* (*at a within s*)
using *assms* **unfolding** *complex-differentiable-def*
by (*metis DERIV-inverse-fun*)

lemma *complex-differentiable-mult* [*derivative-intros*]:
assumes *f complex-differentiable* (*at a within s*)
g complex-differentiable (*at a within s*)
shows (λ*z*. *f z* ∗ *g z*) *complex-differentiable* (*at a within s*)
using *assms* **unfolding** *complex-differentiable-def*
by (*metis DERIV-mult* [*of f - a s g*])

lemma *complex-differentiable-divide* [*derivative-intros*]:
assumes *f complex-differentiable* (*at a within s*)
g complex-differentiable (*at a within s*)
g a ≠ *0*
shows (λ*z*. *f z* / *g z*) *complex-differentiable* (*at a within s*)
using *assms* **unfolding** *complex-differentiable-def*
by (*metis DERIV-divide* [*of f - a s g*])

lemma *complex-differentiable-power* [*derivative-intros*]:
assumes *f complex-differentiable* (*at a within s*)
shows (λ*z*. *f z* ^ *n*) *complex-differentiable* (*at a within s*)
using *assms* **unfolding** *complex-differentiable-def*
by (*metis DERIV-power*)

lemma *complex-differentiable-transform-within*:
0 < *d* ⟹
x ∈ *s* ⟹
(⋀*x'*. *x'* ∈ *s* ⟹ *dist x' x* < *d* ⟹ *f x'* = *g x'*) ⟹
f complex-differentiable (*at x within s*)
⟹ *g complex-differentiable* (*at x within s*)
unfolding *complex-differentiable-def has-field-derivative-def*
by (*blast intro*: *has-derivative-transform-within*)

lemma *complex-differentiable-compose-within*:
assumes *f complex-differentiable* (*at a within s*)
g complex-differentiable (*at* (*f a*) *within f*‘*s*)
shows (*g o f*) *complex-differentiable* (*at a within s*)
using *assms* **unfolding** *complex-differentiable-def*

by (*metis DERIV-image-chain*)

lemma *complex-differentiable-compose*:
f complex-differentiable at z $\Longrightarrow$ *g complex-differentiable at* (*f z*)
$\Longrightarrow$ (*g o f*) *complex-differentiable at z*
by (*metis complex-differentiable-at-within complex-differentiable-compose-within*)

lemma *complex-differentiable-within-open*:
$[\![a \in s;$ *open* $s]\!] \Longrightarrow$ *f complex-differentiable at a within s* $\longleftrightarrow$
f complex-differentiable at a
unfolding *complex-differentiable-def*
by (*metis at-within-open*)

47.6 Caratheodory characterization.

lemma *complex-differentiable-caratheodory-at*:
f complex-differentiable (*at z*) $\longleftrightarrow$
$(\exists g.\ (\forall w.\ f(w) - f(z) = g(w) * (w - z)) \wedge$ *continuous* (*at z*) *g*)
using *CARAT-DERIV* [*of f*]
by (*simp add*: *complex-differentiable-def has-field-derivative-def*)

lemma *complex-differentiable-caratheodory-within*:
f complex-differentiable (*at z within s*) $\longleftrightarrow$
$(\exists g.\ (\forall w.\ f(w) - f(z) = g(w) * (w - z)) \wedge$ *continuous* (*at z within s*) *g*)
using *DERIV-caratheodory-within* [*of f*]
by (*simp add*: *complex-differentiable-def has-field-derivative-def*)

47.7 Holomorphic

definition *holomorphic-on* :: [*complex* $\Rightarrow$ *complex*, *complex set*] $\Rightarrow$ *bool*
(**infixl** (*holomorphic′-on*) *50*)
where *f holomorphic-on s* $\equiv \forall x{\in}s.$ *f complex-differentiable* (*at x within s*)

named-theorems *holomorphic-intros structural introduction rules for holomorphic-on*

lemma *holomorphic-onI* [*intro?*]: ($\bigwedge x.\ x \in s \Longrightarrow$ *f complex-differentiable* (*at x within s*)) $\Longrightarrow$ *f holomorphic-on s*
by (*simp add*: *holomorphic-on-def*)

lemma *holomorphic-onD* [*dest?*]: $[\![$*f holomorphic-on s*; $x \in s]\!] \Longrightarrow$ *f complex-differentiable* (*at x within s*)
by (*simp add*: *holomorphic-on-def*)

lemma *holomorphic-on-imp-differentiable-at*:
$[\![$*f holomorphic-on s*; *open s*; $x \in s]\!] \Longrightarrow$ *f complex-differentiable* (*at x*)
using *at-within-open holomorphic-on-def* **by** *fastforce*

lemma *holomorphic-on-empty* [*holomorphic-intros*]: *f holomorphic-on* {}
by (*simp add*: *holomorphic-on-def*)

lemma *holomorphic-on-open*:
open s $\Longrightarrow$ *f holomorphic-on s* $\longleftrightarrow$ $(\forall x \in s.\ \exists f'.\ DERIV\ f\ x :> f')$
by (*auto simp*: *holomorphic-on-def complex-differentiable-def has-field-derivative-def at-within-open* [*of* - *s*])

lemma *holomorphic-on-imp-continuous-on*:
f holomorphic-on s $\Longrightarrow$ *continuous-on s f*
by (*metis complex-differentiable-imp-continuous-at continuous-on-eq-continuous-within holomorphic-on-def*)

lemma *holomorphic-on-subset*:
f holomorphic-on s $\Longrightarrow$ *t* $\subseteq$ *s* $\Longrightarrow$ *f holomorphic-on t*
unfolding *holomorphic-on-def*
by (*metis complex-differentiable-within-subset subsetD*)

lemma *holomorphic-transform*: $[\![f$ *holomorphic-on s*; $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x]\!]$ $\Longrightarrow$ *g holomorphic-on s*
by (*metis complex-differentiable-transform-within linordered-field-no-ub holomorphic-on-def*)

lemma *holomorphic-cong*: *s* = *t* ==> $(\bigwedge x.\ x \in s \Longrightarrow f\,x = g\,x) \Longrightarrow$ *f holomorphic-on s* $\longleftrightarrow$ *g holomorphic-on t*
by (*metis holomorphic-transform*)

lemma *holomorphic-on-linear* [*holomorphic-intros*]: (*op* $*$ *c*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-linear*)

lemma *holomorphic-on-const* [*holomorphic-intros*]: $(\lambda z.\ c)$ *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-const*)

lemma *holomorphic-on-ident* [*holomorphic-intros*]: $(\lambda x.\ x)$ *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-ident*)

lemma *holomorphic-on-id* [*holomorphic-intros*]: *id holomorphic-on s*
unfolding *id-def* **by** (*rule holomorphic-on-ident*)

lemma *holomorphic-on-compose*:
f holomorphic-on s $\Longrightarrow$ *g holomorphic-on* (*f* ‘ *s*) $\Longrightarrow$ (*g o f*) *holomorphic-on s*
using *complex-differentiable-compose-within*[*of f* - *s g*]
by (*auto simp*: *holomorphic-on-def*)

lemma *holomorphic-on-compose-gen*:
f holomorphic-on s $\Longrightarrow$ *g holomorphic-on t* $\Longrightarrow$ *f* ‘ *s* $\subseteq$ *t* $\Longrightarrow$ (*g o f*) *holomorphic-on s*
by (*metis holomorphic-on-compose holomorphic-on-subset*)

lemma *holomorphic-on-minus* [*holomorphic-intros*]: *f holomorphic-on s* $\Longrightarrow$ $(\lambda z.\ -(f\ z))$ *holomorphic-on s*
by (*metis complex-differentiable-minus holomorphic-on-def*)

lemma *holomorphic-on-add* [*holomorphic-intros*]:
⟦*f holomorphic-on s*; *g holomorphic-on s*⟧ ⟹ (λ*z*. *f z* + *g z*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-add*)

lemma *holomorphic-on-diff* [*holomorphic-intros*]:
⟦*f holomorphic-on s*; *g holomorphic-on s*⟧ ⟹ (λ*z*. *f z* − *g z*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-diff*)

lemma *holomorphic-on-mult* [*holomorphic-intros*]:
⟦*f holomorphic-on s*; *g holomorphic-on s*⟧ ⟹ (λ*z*. *f z* ∗ *g z*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-mult*)

lemma *holomorphic-on-inverse* [*holomorphic-intros*]:
⟦*f holomorphic-on s*; ⋀*z*. *z* ∈ *s* ⟹ *f z* ≠ *0*⟧ ⟹ (λ*z*. *inverse* (*f z*)) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-inverse*)

lemma *holomorphic-on-divide* [*holomorphic-intros*]:
⟦*f holomorphic-on s*; *g holomorphic-on s*; ⋀*z*. *z* ∈ *s* ⟹ *g z* ≠ *0*⟧ ⟹ (λ*z*. *f z* / *g z*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-divide*)

lemma *holomorphic-on-power* [*holomorphic-intros*]:
f holomorphic-on s ⟹ (λ*z*. (*f z*) ^ *n*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-power*)

lemma *holomorphic-on-setsum* [*holomorphic-intros*]:
(⋀*i*. *i* ∈ *I* ⟹ (*f i*) *holomorphic-on s*) ⟹ (λ*x*. *setsum* (λ*i*. *f i x*) *I*) *holomorphic-on s*
unfolding *holomorphic-on-def* **by** (*metis complex-differentiable-setsum*)

lemma *DERIV-deriv-iff-complex-differentiable*:
DERIV f x :> *deriv f x* ⟷ *f complex-differentiable at x*
unfolding *complex-differentiable-def* **by** (*metis DERIV-imp-deriv*)

lemma *complex-derivative-chain*:
f complex-differentiable at x ⟹ *g complex-differentiable at* (*f x*)
⟹ *deriv* (*g o f*) *x* = *deriv g* (*f x*) ∗ *deriv f x*
by (*metis DERIV-deriv-iff-complex-differentiable DERIV-chain DERIV-imp-deriv*)

lemma *deriv-linear*: *deriv* (λ*w*. *c* ∗ *w*) = (λ*z*. *c*)
by (*metis DERIV-imp-deriv DERIV-cmult-Id*)

lemma *deriv-ident*: *deriv* (λ*w*. *w*) = (λ*z*. *1*)
by (*metis DERIV-imp-deriv DERIV-ident*)

lemma *deriv-const*: *deriv* (λ*w*. *c*) = (λ*z*. *0*)
by (*metis DERIV-imp-deriv DERIV-const*)

lemma *complex-derivative-add*:
⟦*f complex-differentiable at z*; *g complex-differentiable at z*⟧
⟹ *deriv* (λ*w*. *f w* + *g w*) *z* = *deriv f z* + *deriv g z*
unfolding *DERIV-deriv-iff-complex-differentiable*[*symmetric*]
by (*auto intro*!: *DERIV-imp-deriv derivative-intros*)

lemma *complex-derivative-diff*:
⟦*f complex-differentiable at z*; *g complex-differentiable at z*⟧
⟹ *deriv* (λ*w*. *f w* − *g w*) *z* = *deriv f z* − *deriv g z*
unfolding *DERIV-deriv-iff-complex-differentiable*[*symmetric*]
by (*auto intro*!: *DERIV-imp-deriv derivative-intros*)

lemma *complex-derivative-mult*:
⟦*f complex-differentiable at z*; *g complex-differentiable at z*⟧
⟹ *deriv* (λ*w*. *f w* ∗ *g w*) *z* = *f z* ∗ *deriv g z* + *deriv f z* ∗ *g z*
unfolding *DERIV-deriv-iff-complex-differentiable*[*symmetric*]
by (*auto intro*!: *DERIV-imp-deriv derivative-eq-intros*)

lemma *complex-derivative-cmult*:
f complex-differentiable at z ⟹ *deriv* (λ*w*. *c* ∗ *f w*) *z* = *c* ∗ *deriv f z*
unfolding *DERIV-deriv-iff-complex-differentiable*[*symmetric*]
by (*auto intro*!: *DERIV-imp-deriv derivative-eq-intros*)

lemma *complex-derivative-cmult-right*:
f complex-differentiable at z ⟹ *deriv* (λ*w*. *f w* ∗ *c*) *z* = *deriv f z* ∗ *c*
unfolding *DERIV-deriv-iff-complex-differentiable*[*symmetric*]
by (*auto intro*!: *DERIV-imp-deriv derivative-eq-intros*)

lemma *complex-derivative-transform-within-open*:
⟦*f holomorphic-on s*; *g holomorphic-on s*; *open s*; *z* ∈ *s*; ⋀*w*. *w* ∈ *s* ⟹ *f w* = *g w*⟧
⟹ *deriv f z* = *deriv g z*
unfolding *holomorphic-on-def*
by (*rule DERIV-imp-deriv*)
(*metis DERIV-deriv-iff-complex-differentiable DERIV-transform-within-open at-within-open*)

lemma *complex-derivative-compose-linear*:
f complex-differentiable at (*c* ∗ *z*) ⟹ *deriv* (λ*w*. *f* (*c* ∗ *w*)) *z* = *c* ∗ *deriv f* (*c* ∗ *z*)
apply (*rule DERIV-imp-deriv*)
apply (*simp add*: *DERIV-deriv-iff-complex-differentiable* [*symmetric*])
apply (*drule DERIV-chain'* [*of times c c z UNIV f deriv f* (*c* ∗ *z*), *OF DERIV-cmult-Id*])
apply (*simp add*: *algebra-simps*)
done

47.8 Analyticity on a set

definition *analytic-on* (**infixl** (*analytic'-on*) *50*)

where
f analytic-on s $\equiv \forall x \in s.\ \exists e.\ 0 < e \wedge$ *f holomorphic-on* (*ball x e*)

lemma *analytic-imp-holomorphic*: *f analytic-on s* $\Longrightarrow$ *f holomorphic-on s*
by (*simp add*: *at-within-open* [*OF - open-ball*] *analytic-on-def holomorphic-on-def*)
(*metis centre-in-ball complex-differentiable-at-within*)

lemma *analytic-on-open*: *open s* $\Longrightarrow$ *f analytic-on s* $\longleftrightarrow$ *f holomorphic-on s*
apply (*auto simp*: *analytic-imp-holomorphic*)
apply (*auto simp*: *analytic-on-def holomorphic-on-def*)
by (*metis holomorphic-on-def holomorphic-on-subset open-contains-ball*)

lemma *analytic-on-imp-differentiable-at*:
f analytic-on s $\Longrightarrow$ *x* $\in$ *s* $\Longrightarrow$ *f complex-differentiable* (*at x*)
apply (*auto simp*: *analytic-on-def holomorphic-on-def*)
by (*metis Topology-Euclidean-Space.open-ball centre-in-ball complex-differentiable-within-open*)

lemma *analytic-on-subset*: *f analytic-on s* $\Longrightarrow$ *t* $\subseteq$ *s* $\Longrightarrow$ *f analytic-on t*
by (*auto simp*: *analytic-on-def*)

lemma *analytic-on-Un*: *f analytic-on* (*s* $\cup$ *t*) $\longleftrightarrow$ *f analytic-on s* $\wedge$ *f analytic-on t*
by (*auto simp*: *analytic-on-def*)

lemma *analytic-on-Union*: *f analytic-on* ($\bigcup$ *s*) $\longleftrightarrow$ ($\forall$ *t* $\in$ *s*. *f analytic-on t*)
by (*auto simp*: *analytic-on-def*)

lemma *analytic-on-UN*: *f analytic-on* ($\bigcup$ *i*$\in$*I*. *s i*) $\longleftrightarrow$ ($\forall$ *i*$\in$*I*. *f analytic-on* (*s i*))
by (*auto simp*: *analytic-on-def*)

lemma *analytic-on-holomorphic*:
f analytic-on s $\longleftrightarrow$ ($\exists$ *t*. *open t* $\wedge$ *s* $\subseteq$ *t* $\wedge$ *f holomorphic-on t*)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* $\longleftrightarrow$ ($\exists$ *t*. *open t* $\wedge$ *s* $\subseteq$ *t* $\wedge$ *f analytic-on t*)
proof *safe*
assume *f analytic-on s*
then show $\exists$ *t*. *open t* $\wedge$ *s* $\subseteq$ *t* $\wedge$ *f analytic-on t*
apply (*simp add*: *analytic-on-def*)
apply (*rule exI* [**where** *x*=$\bigcup${*u*. *open u* $\wedge$ *f analytic-on u*}], *auto*)
apply (*metis Topology-Euclidean-Space.open-ball analytic-on-open centre-in-ball*)
by (*metis analytic-on-def*)
next
fix *t*
assume *open t s* $\subseteq$ *t f analytic-on t*
then show *f analytic-on s*
by (*metis analytic-on-subset*)
qed
also have ... $\longleftrightarrow$ *?rhs*
by (*auto simp*: *analytic-on-open*)

finally show *?thesis* **.**
qed

lemma *analytic-on-linear*: (*op* ∗ *c*) *analytic-on s*
by (*auto simp add*: *analytic-on-holomorphic holomorphic-on-linear*)

lemma *analytic-on-const*: (λ*z*. *c*) *analytic-on s*
by (*metis analytic-on-def holomorphic-on-const zero-less-one*)

lemma *analytic-on-ident*: (λ*x*. *x*) *analytic-on s*
by (*simp add*: *analytic-on-def holomorphic-on-ident gt-ex*)

lemma *analytic-on-id*: *id analytic-on s*
unfolding *id-def* **by** (*rule analytic-on-ident*)

lemma *analytic-on-compose*:
assumes *f*: *f analytic-on s*
and *g*: *g analytic-on* (*f* ‘ *s*)
shows (*g o f*) *analytic-on s*
unfolding *analytic-on-def*
proof (*intro ballI*)
fix *x*
assume *x*: *x* ∈ *s*
then obtain *e* **where** *e*: *0* < *e* **and** *fh*: *f holomorphic-on ball x e* **using** *f*
by (*metis analytic-on-def*)
obtain *e′* **where** *e′*: *0* < *e′* **and** *gh*: *g holomorphic-on ball* (*f x*) *e′* **using** *g*
by (*metis analytic-on-def g image-eqI x*)
have *isCont f x*
by (*metis analytic-on-imp-differentiable-at complex-differentiable-imp-continuous-at f x*)
with *e′* **obtain** *d* **where** *d*: *0* < *d* **and** *fd*: *f* ‘ *ball x d* ⊆ *ball* (*f x*) *e′*
by (*auto simp*: *continuous-at-ball*)
have *g* ∘ *f holomorphic-on ball x* (*min d e*)
apply (*rule holomorphic-on-compose*)
apply (*metis fh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
by (*metis fd gh holomorphic-on-subset image-mono min.cobounded1 subset-ball*)
then show ∃ *e*>*0*. *g* ∘ *f holomorphic-on ball x e*
by (*metis d e min-less-iff-conj*)
qed

lemma *analytic-on-compose-gen*:
f analytic-on s ⟹ *g analytic-on t* ⟹ (⋀*z*. *z* ∈ *s* ⟹ *f z* ∈ *t*)
⟹ *g o f analytic-on s*
by (*metis analytic-on-compose analytic-on-subset image-subset-iff*)

lemma *analytic-on-neg*:
f analytic-on s ⟹ (λ*z*. −(*f z*)) *analytic-on s*
by (*metis analytic-on-holomorphic holomorphic-on-minus*)

lemma *analytic-on-add*:
assumes *f*: *f analytic-on s*
and *g*: *g analytic-on s*
shows (λ*z*. *f z* + *g z*) *analytic-on s*
unfolding *analytic-on-def*
proof (*intro ballI*)
fix *z*
assume *z*: *z* ∈ *s*
then obtain *e* **where** *e*: *0* < *e* **and** *fh*: *f holomorphic-on ball z e* **using** *f*
by (*metis analytic-on-def*)
obtain *e'* **where** *e'*: *0* < *e'* **and** *gh*: *g holomorphic-on ball z e'* **using** *g*
by (*metis analytic-on-def g z*)
have (λ*z*. *f z* + *g z*) *holomorphic-on ball z* (*min e e'*)
apply (*rule holomorphic-on-add*)
apply (*metis fh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
by (*metis gh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
then show ∃ *e*>*0*. (λ*z*. *f z* + *g z*) *holomorphic-on ball z e*
by (*metis e e' min-less-iff-conj*)
qed

lemma *analytic-on-diff*:
assumes *f*: *f analytic-on s*
and *g*: *g analytic-on s*
shows (λ*z*. *f z* − *g z*) *analytic-on s*
unfolding *analytic-on-def*
proof (*intro ballI*)
fix *z*
assume *z*: *z* ∈ *s*
then obtain *e* **where** *e*: *0* < *e* **and** *fh*: *f holomorphic-on ball z e* **using** *f*
by (*metis analytic-on-def*)
obtain *e'* **where** *e'*: *0* < *e'* **and** *gh*: *g holomorphic-on ball z e'* **using** *g*
by (*metis analytic-on-def g z*)
have (λ*z*. *f z* − *g z*) *holomorphic-on ball z* (*min e e'*)
apply (*rule holomorphic-on-diff*)
apply (*metis fh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
by (*metis gh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
then show ∃ *e*>*0*. (λ*z*. *f z* − *g z*) *holomorphic-on ball z e*
by (*metis e e' min-less-iff-conj*)
qed

lemma *analytic-on-mult*:
assumes *f*: *f analytic-on s*
and *g*: *g analytic-on s*
shows (λ*z*. *f z* ∗ *g z*) *analytic-on s*
unfolding *analytic-on-def*
proof (*intro ballI*)
fix *z*
assume *z*: *z* ∈ *s*
then obtain *e* **where** *e*: *0* < *e* **and** *fh*: *f holomorphic-on ball z e* **using** *f*

by (*metis analytic-on-def*)
obtain e' **where** e': $0 < e'$ **and** *gh*: *g holomorphic-on ball z e'* **using** *g*
by (*metis analytic-on-def g z*)
have $(\lambda z.\ f\ z * g\ z)$ *holomorphic-on ball z* (*min e e'*)
apply (*rule holomorphic-on-mult*)
apply (*metis fh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
by (*metis gh holomorphic-on-subset min.bounded-iff order-refl subset-ball*)
then show $\exists e{>}0.\ (\lambda z.\ f\ z * g\ z)$ *holomorphic-on ball z e*
by (*metis e e' min-less-iff-conj*)
qed

lemma *analytic-on-inverse*:
assumes *f*: *f analytic-on s*
and *nz*: $(\bigwedge z.\ z \in s \Longrightarrow f\ z \neq 0)$
shows $(\lambda z.\ inverse\ (f\ z))$ *analytic-on s*
unfolding *analytic-on-def*
proof (*intro ballI*)
fix *z*
assume *z*: $z \in s$
then obtain *e* **where** *e*: $0 < e$ **and** *fh*: *f holomorphic-on ball z e* **using** *f*
by (*metis analytic-on-def*)
have *continuous-on* (*ball z e*) *f*
by (*metis fh holomorphic-on-imp-continuous-on*)
then obtain e' **where** e': $0 < e'$ **and** nz': $\bigwedge y.\ dist\ z\ y < e' \Longrightarrow f\ y \neq 0$
by (*metis Topology-Euclidean-Space.open-ball centre-in-ball continuous-on-open-avoid e z nz*)
have $(\lambda z.\ inverse\ (f\ z))$ *holomorphic-on ball z* (*min e e'*)
apply (*rule holomorphic-on-inverse*)
apply (*metis fh holomorphic-on-subset min.cobounded2 min.commute subset-ball*)
by (*metis nz' mem-ball min-less-iff-conj*)
then show $\exists e{>}0.\ (\lambda z.\ inverse\ (f\ z))$ *holomorphic-on ball z e*
by (*metis e e' min-less-iff-conj*)
qed

lemma *analytic-on-divide*:
assumes *f*: *f analytic-on s*
and *g*: *g analytic-on s*
and *nz*: $(\bigwedge z.\ z \in s \Longrightarrow g\ z \neq 0)$
shows $(\lambda z.\ f\ z\ /\ g\ z)$ *analytic-on s*
unfolding *divide-inverse*
by (*metis analytic-on-inverse analytic-on-mult f g nz*)

lemma *analytic-on-power*:
f analytic-on s $\Longrightarrow$ $(\lambda z.\ (f\ z)\ \hat{}\ n)$ *analytic-on s*
by (*induct n*) (*auto simp*: *analytic-on-const analytic-on-mult*)

lemma *analytic-on-setsum*:
$(\bigwedge i.\ i \in I \Longrightarrow (f\ i)\ analytic\text{-}on\ s) \Longrightarrow (\lambda x.\ setsum\ (\lambda i.\ f\ i\ x)\ I)\ analytic\text{-}on\ s$

by (*induct I rule*: *infinite-finite-induct*) (*auto simp*: *analytic-on-const analytic-on-add*)

47.9 analyticity at a point.

lemma *analytic-at-ball*:
f analytic-on {*z*} ⟷ (∃ *e*. *0*<*e* ∧ *f holomorphic-on ball z e*)
by (*metis analytic-on-def singleton-iff*)

lemma *analytic-at*:
f analytic-on {*z*} ⟷ (∃ *s*. *open s* ∧ *z* ∈ *s* ∧ *f holomorphic-on s*)
by (*metis analytic-on-holomorphic empty-subsetI insert-subset*)

lemma *analytic-on-analytic-at*:
f analytic-on s ⟷ (∀ *z* ∈ *s*. *f analytic-on* {*z*})
by (*metis analytic-at-ball analytic-on-def*)

lemma *analytic-at-two*:
f analytic-on {*z*} ∧ *g analytic-on* {*z*} ⟷
(∃ *s*. *open s* ∧ *z* ∈ *s* ∧ *f holomorphic-on s* ∧ *g holomorphic-on s*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *s t*
where *st*: *open s z* ∈ *s f holomorphic-on s*
open t z ∈ *t g holomorphic-on t*
by (*auto simp*: *analytic-at*)
show *?rhs*
apply (*rule-tac x*=*s* ∩ *t* **in** *exI*)
using *st*
apply (*auto simp*: *Diff-subset holomorphic-on-subset*)
done
next
assume *?rhs*
then show *?lhs*
by (*force simp add*: *analytic-at*)
qed

47.10 Combining theorems for derivative with "analytic at" hypotheses

lemma
assumes *f analytic-on* {*z*} *g analytic-on* {*z*}
shows *complex-derivative-add-at*: *deriv* (λ*w*. *f w* + *g w*) *z* = *deriv f z* + *deriv g z*
and *complex-derivative-diff-at*: *deriv* (λ*w*. *f w* − *g w*) *z* = *deriv f z* − *deriv g z*
and *complex-derivative-mult-at*: *deriv* (λ*w*. *f w* ∗ *g w*) *z* =
f z ∗ *deriv g z* + *deriv f z* ∗ *g z*
proof −
obtain *s* **where** *s*: *open s z* ∈ *s f holomorphic-on s g holomorphic-on s*

using *assms* **by** (*metis analytic-at-two*)
show *deriv* (λ*w*. *f w* + *g w*) *z* = *deriv f z* + *deriv g z*
apply (*rule DERIV-imp-deriv* [*OF DERIV-add*])
using *s*
apply (*auto simp*: *holomorphic-on-open complex-differentiable-def DERIV-deriv-iff-complex-differentiable*)
done
show *deriv* (λ*w*. *f w* − *g w*) *z* = *deriv f z* − *deriv g z*
apply (*rule DERIV-imp-deriv* [*OF DERIV-diff*])
using *s*
apply (*auto simp*: *holomorphic-on-open complex-differentiable-def DERIV-deriv-iff-complex-differentiable*)
done
show *deriv* (λ*w*. *f w* ∗ *g w*) *z* = *f z* ∗ *deriv g z* + *deriv f z* ∗ *g z*
apply (*rule DERIV-imp-deriv* [*OF DERIV-mult'*])
using *s*
apply (*auto simp*: *holomorphic-on-open complex-differentiable-def DERIV-deriv-iff-complex-differentiable*)
done
qed

lemma *complex-derivative-cmult-at*:
f analytic-on {*z*} ⟹ *deriv* (λ*w*. *c* ∗ *f w*) *z* = *c* ∗ *deriv f z*
by (*auto simp*: *complex-derivative-mult-at deriv-const analytic-on-const*)

lemma *complex-derivative-cmult-right-at*:
f analytic-on {*z*} ⟹ *deriv* (λ*w*. *f w* ∗ *c*) *z* = *deriv f z* ∗ *c*
by (*auto simp*: *complex-derivative-mult-at deriv-const analytic-on-const*)

47.11 Complex differentiation of sequences and series

lemma *has-complex-derivative-sequence*:
fixes *s* :: *complex set*
assumes *cvs*: *convex s*
and *df*: ⋀*n x*. *x* ∈ *s* ⟹ (*f n has-field-derivative f' n x*) (*at x within s*)
and *conv*: ⋀*e*. 0 < *e* ⟹ ∃*N*. ∀*n x*. *n* ≥ *N* ⟶ *x* ∈ *s* ⟶ *norm* (*f' n x* − *g' x*) ≤ *e*
and ∃*x l*. *x* ∈ *s* ∧ ((λ*n*. *f n x*) ⟶ *l*) *sequentially*
shows ∃*g*. ∀*x* ∈ *s*. ((λ*n*. *f n x*) ⟶ *g x*) *sequentially* ∧
(*g has-field-derivative* (*g' x*)) (*at x within s*)
proof −
from *assms* **obtain** *x l* **where** *x*: *x* ∈ *s* **and** *tf*: ((λ*n*. *f n x*) ⟶ *l*) *sequentially*
by *blast*
{ fix *e*::*real* **assume** *e*: *e* > 0
then obtain *N* **where** *N*: ∀*n*≥*N*. ∀*x*. *x* ∈ *s* ⟶ *cmod* (*f' n x* − *g' x*) ≤ *e*
by (*metis conv*)
have ∃*N*. ∀*n*≥*N*. ∀*x*∈*s*. ∀*h*. *cmod* (*f' n x* ∗ *h* − *g' x* ∗ *h*) ≤ *e* ∗ *cmod h*
proof (*rule exI* [*of* - *N*], *clarify*)
fix *n y h*
assume *N* ≤ *n y* ∈ *s*
then have *cmod* (*f' n y* − *g' y*) ≤ *e*
by (*metis N*)

```
    then have cmod h * cmod (f' n y - g' y) ≤ cmod h * e
      by (auto simp: antisym-conv2 mult-le-cancel-left norm-triangle-ineq2)
    then show cmod (f' n y * h - g' y * h) ≤ e * cmod h
      by (simp add: norm-mult [symmetric] field-simps)
  qed
 } note ** = this
 show ?thesis
 unfolding has-field-derivative-def
 proof (rule has-derivative-sequence [OF cvs - - x])
  show ∀ n. ∀ x∈s. (f n has-derivative (op * (f' n x))) (at x within s)
    by (metis has-field-derivative-def df)
 next show (λn. f n x) ⟶⟶ l
  by (rule tf)
 next show ∀ e>0. ∃ N. ∀ n≥N. ∀ x∈s. ∀ h. cmod (f' n x * h - g' x * h) ≤ e *
cmod h
  by (blast intro: **)
 qed
qed


lemma has-complex-derivative-series:
 fixes s :: complex set
 assumes cvs: convex s
    and df: ⋀n x. x ∈ s ⟹ (f n has-field-derivative f' n x) (at x within s)
    and conv: ⋀e. 0 < e ⟹ ∃ N. ∀ n x. n ≥ N ⟶ x ∈ s
          ⟶ cmod ((∑ i<n. f' i x) - g' x) ≤ e
    and ∃ x l. x ∈ s ∧ ((λn. f n x) sums l)
  shows ∃ g. ∀ x ∈ s. ((λn. f n x) sums g x) ∧ ((g has-field-derivative g' x) (at x
within s))
proof -
 from assms obtain x l where x: x ∈ s and sf: ((λn. f n x) sums l)
  by blast
 { fix e::real assume e: e > 0
  then obtain N where N: ∀ n x. n ≥ N ⟶ x ∈ s
        ⟶ cmod ((∑ i<n. f' i x) - g' x) ≤ e
    by (metis conv)
  have ∃ N. ∀ n≥N. ∀ x∈s. ∀ h. cmod ((∑ i<n. h * f' i x) - g' x * h) ≤ e *
cmod h
  proof (rule exI [of - N], clarify)
    fix n y h
    assume N ≤ n y ∈ s
    then have cmod ((∑ i<n. f' i y) - g' y) ≤ e
      by (metis N)
    then have cmod h * cmod ((∑ i<n. f' i y) - g' y) ≤ cmod h * e
      by (auto simp: antisym-conv2 mult-le-cancel-left norm-triangle-ineq2)
    then show cmod ((∑ i<n. h * f' i y) - g' y * h) ≤ e * cmod h
      by (simp add: norm-mult [symmetric] field-simps setsum-right-distrib)
  qed
 } note ** = this
```

show *?thesis*
unfolding *has-field-derivative-def*
proof (*rule has-derivative-series* [*OF cvs - - x*])
fix *n x*
assume $x \in s$
then show $((f\ n)$ *has-derivative* $(\lambda z.\ z * f'\ n\ x))$ $(at\ x\ within\ s)$
by (*metis df has-field-derivative-def mult-commute-abs*)
next show $((\lambda n.\ f\ n\ x)$ *sums* $l)$
by (*rule sf*)
next show $\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x{\in}s.\ \forall h.\ cmod\ ((\sum i{<}n.\ h * f'\ i\ x) - g'\ x * h) \leq e * cmod\ h$
by (*blast intro*: ∗∗)
qed
qed

lemma *complex-differentiable-series*:
fixes $f :: nat \Rightarrow complex \Rightarrow complex$
assumes *convex s open s*
assumes $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n$ *has-field-derivative* $f'\ n\ x)$ $(at\ x)$
assumes *uniformly-convergent-on s* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$
assumes $x0 \in s$ *summable* $(\lambda n.\ f\ n\ x0)$ **and** x: $x \in s$
shows *summable* $(\lambda n.\ f\ n\ x)$ **and** $(\lambda x.\ \sum n.\ f\ n\ x)$ *complex-differentiable* $(at\ x)$
proof −
from *assms*(*4*) **obtain** g' **where** *A*: *uniform-limit s* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$ g' *sequentially*
unfolding *uniformly-convergent-on-def* **by** *blast*
from *x* **and** ‹*open s*› **have** *s*: *at x within s* = *at x* **by** (*rule at-within-open*)
have $\exists g.\ \forall x{\in}s.\ (\lambda n.\ f\ n\ x)$ *sums* $g\ x \wedge (g$ *has-field-derivative* $g'\ x)$ $(at\ x\ within\ s)$
by (*intro has-field-derivative-series*[*of s f f' g' x0*] *assms A has-field-derivative-at-within*)
then obtain *g* **where** *g*: $\bigwedge x.\ x \in s \Longrightarrow (\lambda n.\ f\ n\ x)$ *sums* $g\ x$
$\bigwedge x.\ x \in s \Longrightarrow (g$ *has-field-derivative* $g'\ x)$ $(at\ x\ within\ s)$ **by** *blast*
from *g*[*OF x*] **show** *summable* $(\lambda n.\ f\ n\ x)$ **by** (*auto simp*: *summable-def*)
from *g*(*2*)[*OF x*] **have** g': $(g$ *has-derivative op* $*$ $(g'\ x))$ $(at\ x)$
by (*simp add*: *has-field-derivative-def s*)
have $((\lambda x.\ \sum n.\ f\ n\ x)$ *has-derivative op* $*$ $(g'\ x))$ $(at\ x)$
by (*rule has-derivative-transform-within-open*[*OF g'* ‹*open s*› *x*])
(*insert g*, *auto simp*: *sums-iff*)
thus $(\lambda x.\ \sum n.\ f\ n\ x)$ *complex-differentiable* $(at\ x)$ **unfolding** *differentiable-def*
by (*auto simp*: *summable-def complex-differentiable-def has-field-derivative-def*)
qed

lemma *complex-differentiable-series'*:
fixes $f :: nat \Rightarrow complex \Rightarrow complex$
assumes *convex s open s*
assumes $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n$ *has-field-derivative* $f'\ n\ x)$ $(at\ x)$
assumes *uniformly-convergent-on s* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$

assumes *x0* ∈ *s summable* (λ*n*. *f n x0*)
shows (λ*x*. ∑ *n*. *f n x*) *complex-differentiable* (*at x0*)
using *complex-differentiable-series*[*OF assms*, *of x0*] ‹*x0* ∈ *s*› **by** *blast+*

47.12 Bound theorem

lemma *complex-differentiable-bound*:
fixes *s* :: *complex set*
assumes *cvs*: *convex s*
and *df*: ⋀*z*. *z* ∈ *s* ⟹ (*f has-field-derivative f' z*) (*at z within s*)
and *dn*: ⋀*z*. *z* ∈ *s* ⟹ *norm* (*f' z*) ≤ *B*
and *x* ∈ *s* *y* ∈ *s*
shows *norm*(*f x* − *f y*) ≤ *B* ∗ *norm*(*x* − *y*)
apply (*rule differentiable-bound* [*OF cvs*])
apply (*rule ballI*, *erule df* [*unfolded has-field-derivative-def*])
apply (*rule ballI*, *rule onorm-le*, *simp add*: *norm-mult mult-right-mono dn*)
apply *fact*
apply *fact*
done

47.13 Inverse function theorem for complex derivatives.

lemma *has-complex-derivative-inverse-basic*:
fixes *f* :: *complex* ⇒ *complex*
shows *DERIV f* (*g y*) :> *f'* ⟹
f' ≠ *0* ⟹
continuous (*at y*) *g* ⟹
open t ⟹
y ∈ *t* ⟹
(⋀*z*. *z* ∈ *t* ⟹ *f* (*g z*) = *z*)
⟹ *DERIV g y* :> *inverse* (*f'*)
unfolding *has-field-derivative-def*
apply (*rule has-derivative-inverse-basic*)
apply (*auto simp*: *bounded-linear-mult-right*)
done

lemma *has-complex-derivative-inverse-strong*:
fixes *f* :: *complex* ⇒ *complex*
shows *DERIV f x* :> *f'* ⟹
f' ≠ *0* ⟹
open s ⟹
x ∈ *s* ⟹
continuous-on s f ⟹
(⋀*z*. *z* ∈ *s* ⟹ *g* (*f z*) = *z*)
⟹ *DERIV g* (*f x*) :> *inverse* (*f'*)
unfolding *has-field-derivative-def*
apply (*rule has-derivative-inverse-strong* [*of s x f g*])
using *assms*
by *auto*

lemma *has-complex-derivative-inverse-strong-x*:
fixes f :: *complex* ⇒ *complex*
shows *DERIV f* (*g y*) :> $f' \Longrightarrow$
$f' \neq 0 \Longrightarrow$
open s ⟹
continuous-on s f ⟹
$g\ y \in s \Longrightarrow f(g\ y) = y \Longrightarrow$
$(\bigwedge z.\ z \in s \Longrightarrow g\ (f\ z) = z)$
⟹ *DERIV g y* :> *inverse* (f')
unfolding *has-field-derivative-def*
apply (*rule has-derivative-inverse-strong-x* [*of s g y f*])
using *assms*
by *auto*

47.14 Taylor on Complex Numbers

lemma *setsum-Suc-reindex*:
fixes f :: *nat* ⇒ 'a::*ab-group-add*
shows *setsum f* {0..*n*} = *f* 0 − *f* (*Suc n*) + *setsum* (λ*i*. *f* (*Suc i*)) {0..*n*}
by (*induct n*) *auto*

lemma *complex-taylor*:
assumes *s*: *convex s*
and *f*: $\bigwedge i\ x.\ x \in s \Longrightarrow i \leq n \Longrightarrow$ (*f i has-field-derivative f* (*Suc i*) *x*) (*at x within s*)
and *B*: $\bigwedge x.\ x \in s \Longrightarrow$ *cmod* (*f* (*Suc n*) *x*) ≤ *B*
and *w*: $w \in s$
and *z*: $z \in s$
shows *cmod*(*f* 0 *z* − (∑ *i*≤*n*. *f i w* ∗ (*z*−*w*) ^ *i* / (*fact i*)))
≤ *B* ∗ *cmod*(*z* − *w*) ^(*Suc n*) / *fact n*
proof −
have *wzs*: *closed-segment w z* ⊆ *s* **using** *assms*
by (*metis convex-contains-segment*)
{ fix *u*
assume *u* ∈ *closed-segment w z*
then have *u* ∈ *s*
by (*metis wzs subsetD*)
have (∑ *i*≤*n*. *f i u* ∗ (− *of-nat i* ∗ (*z*−*u*) ^(*i* − 1)) / (*fact i*) +
f (*Suc i*) *u* ∗ (*z*−*u*) ^ *i* / (*fact i*)) =
f (*Suc n*) *u* ∗ (*z*−*u*) ^ *n* / (*fact n*)
proof (*induction n*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc n*)
have (∑ *i*≤*Suc n*. *f i u* ∗ (− *of-nat i* ∗ (*z*−*u*) ^ (*i* − 1)) / (*fact i*) +
f (*Suc i*) *u* ∗ (*z*−*u*) ^ *i* / (*fact i*)) =
f (*Suc n*) *u* ∗ (*z*−*u*) ^ *n* / (*fact n*) +
f (*Suc* (*Suc n*)) *u* ∗ ((*z*−*u*) ∗ (*z*−*u*) ^ *n*) / (*fact* (*Suc n*)) −

```
        f (Suc n) u * ((1 + of-nat n) * (z−u) ^ n) / (fact (Suc n))
      using Suc by simp
    also have ... = f (Suc (Suc n)) u * (z−u) ^ Suc n / (fact (Suc n))
    proof −
      have (fact(Suc n)) *
          (f(Suc n) u *(z−u) ^ n / (fact n) +
           f(Suc(Suc n)) u *((z−u) *(z−u) ^ n) / (fact(Suc n)) −
           f(Suc n) u *((1 + of-nat n) *(z−u) ^ n) / (fact(Suc n))) =
         ((fact(Suc n)) *(f(Suc n) u *(z−u) ^ n)) / (fact n) +
        ((fact(Suc n)) *(f(Suc(Suc n)) u *((z−u) *(z−u) ^ n)) / (fact(Suc n)))
−
          ((fact(Suc n)) *(f(Suc n) u *(of-nat(Suc n) *(z−u) ^ n))) / (fact(Suc
n))
        by (simp add: algebra-simps del: fact.simps)
      also have ... = ((fact (Suc n)) * (f (Suc n) u * (z−u) ^ n)) / (fact n) +
                      (f (Suc (Suc n)) u * ((z−u) * (z−u) ^ n)) −
                      (f (Suc n) u * ((1 + of-nat n) * (z−u) ^ n))
        by (simp del: fact.simps)
      also have ... = (of-nat (Suc n) * (f (Suc n) u * (z−u) ^ n)) +
                      (f (Suc (Suc n)) u * ((z−u) * (z−u) ^ n)) −
                      (f (Suc n) u * ((1 + of-nat n) * (z−u) ^ n))
        by (simp only: fact.simps of-nat-mult ac-simps) simp
      also have ... = f (Suc (Suc n)) u * ((z−u) * (z−u) ^ n)
        by (simp add: algebra-simps)
      finally show ?thesis
       by (simp add: mult-left-cancel [where c = (fact (Suc n)), THEN iffD1]
del: fact.simps)
    qed
    finally show ?case .
  qed
  then have ((λv. (∑ i≤n. f i v * (z − v) ^i / (fact i)))
             has-field-derivative f (Suc n) u * (z−u) ^ n / (fact n))
            (at u within s)
    apply (intro derivative-eq-intros)
    apply (blast intro: assms ‹u ∈ s›)
    apply (rule refl)+
    apply (auto simp: field-simps)
    done
 } note sum-deriv = this
 { fix u
  assume u: u ∈ closed-segment w z
  then have us: u ∈ s
    by (metis wzs subsetD)
  have cmod (f (Suc n) u) * cmod (z − u) ^ n ≤ cmod (f (Suc n) u) * cmod
(u − z) ^ n
    by (metis norm-minus-commute order-refl)
  also have ... ≤ cmod (f (Suc n) u) * cmod (z − w) ^ n
    by (metis mult-left-mono norm-ge-zero power-mono segment-bound [OF u])
  also have ... ≤ B * cmod (z − w) ^ n
```

```
    by (metis norm-ge-zero zero-le-power mult-right-mono  B [OF us])
   finally have cmod (f (Suc n) u) * cmod (z - u) ^ n ≤ B * cmod (z - w) ^
n .
  } note cmod-bound = this
  have (∑ i≤n. f i z * (z - z) ^ i / (fact i)) = (∑ i≤n. (f i z / (fact i)) * 0 ^ i)
    by simp
  also have ... = f 0 z / (fact 0)
    by (subst setsum-zero-power) simp
  finally have cmod (f 0 z - (∑ i≤n. f i w * (z - w) ^ i / (fact i)))
                ≤ cmod ((∑ i≤n. f i w * (z - w) ^ i / (fact i)) -
                        (∑ i≤n. f i z * (z - z) ^ i / (fact i)))
    by (simp add: norm-minus-commute)
  also have ... ≤ B * cmod (z - w) ^ n / (fact n) * cmod (w - z)
    apply (rule complex-differentiable-bound
      [where f' = λw. f (Suc n) w * (z - w)^n / (fact n)
        and s = closed-segment w z, OF convex-closed-segment])
    apply (auto simp: ends-in-segment DERIV-subset [OF sum-deriv wzs]
                norm-divide norm-mult norm-power divide-le-cancel cmod-bound)
    done
  also have ... ≤ B * cmod (z - w) ^ Suc n / (fact n)
    by (simp add: algebra-simps norm-minus-commute)
  finally show ?thesis .
qed
```

Something more like the traditional MVT for real components.

```
lemma complex-mvt-line:
  assumes ⋀u. u ∈ closed-segment w z ⟹ (f has-field-derivative f'(u)) (at u)
   shows ∃ u. u ∈ closed-segment w z ∧ Re(f z) - Re(f w) = Re(f'(u) * (z - w))
proof -
  have twz: ⋀t. (1 - t) *R w + t *R z = w + t *R (z - w)
   by (simp add: real-vector.scale-left-diff-distrib real-vector.scale-right-diff-distrib)
  note assms[unfolded has-field-derivative-def, derivative-intros]
  show ?thesis
    apply (cut-tac mvt-simple
                [of 0 1 Re o f o (λt. (1 - t) *R w +  t *R z)
                 λu. Re o (λh. f'((1 - u) *R w + u *R z) * h) o (λt. t *R (z -
w))])
    apply auto
    apply (rule-tac x=(1 - x) *R w + x *R z in exI)
    apply (auto simp: closed-segment-def twz) []
    apply (intro derivative-eq-intros has-derivative-at-within, simp-all)
    apply (simp add: fun-eq-iff real-vector.scale-right-diff-distrib)
    apply (force simp: twz closed-segment-def)
    done
qed

lemma complex-taylor-mvt:
  assumes ⋀i x. ⟦x ∈ closed-segment w z; i ≤ n⟧ ⟹ ((f i) has-field-derivative f
(Suc i) x) (at x)
```

```
    shows ∃u. u ∈ closed-segment w z ∧
          Re (f 0 z) =
          Re ((∑ i = 0..n. f i w * (z − w) ^ i / (fact i)) +
             (f (Suc n) u * (z−u) ^n / (fact n)) * (z − w))
proof −
  { fix u
    assume u: u ∈ closed-segment w z
    have (∑ i = 0..n.
            (f (Suc i) u * (z−u) ^ i − of-nat i * (f i u * (z−u) ^ (i − Suc 0))) /
            (fact i)) =
         f (Suc 0) u −
            (f (Suc (Suc n)) u * ((z−u) ^ Suc n) − (of-nat (Suc n)) * (z−u) ^ n
* f (Suc n) u) /
            (fact (Suc n)) +
            (∑ i = 0..n.
               (f (Suc (Suc i)) u * ((z−u) ^ Suc i) − of-nat (Suc i) * (f (Suc i) u
* (z−u) ^ i)) /
               (fact (Suc i)))
      by (subst setsum-Suc-reindex) simp
    also have ... = f (Suc 0) u −
            (f (Suc (Suc n)) u * ((z−u) ^ Suc n) − (of-nat (Suc n)) * (z−u) ^ n
* f (Suc n) u) /
            (fact (Suc n)) +
            (∑ i = 0..n.
               f (Suc (Suc i)) u * ((z−u) ^ Suc i) / (fact (Suc i)) −
               f (Suc i) u * (z−u) ^ i / (fact i))
      by (simp only: diff-divide-distrib fact-cancel ac-simps)
    also have ... = f (Suc 0) u −
            (f (Suc (Suc n)) u * (z−u) ^ Suc n − of-nat (Suc n) * (z−u) ^ n * f
(Suc n) u) /
            (fact (Suc n)) +
            f (Suc (Suc n)) u * (z−u) ^ Suc n / (fact (Suc n)) − f (Suc 0) u
      by (subst setsum-Suc-diff) auto
    also have ... = f (Suc n) u * (z−u) ^ n / (fact n)
      by (simp only: algebra-simps diff-divide-distrib fact-cancel)
    finally have (∑ i = 0..n. (f (Suc i) u * (z − u) ^ i
                          − of-nat i * (f i u * (z−u) ^ (i − Suc 0))) / (fact i)) =
                 f (Suc n) u * (z − u) ^ n / (fact n) .
    then have ((λu. ∑ i = 0..n. f i u * (z − u) ^ i / (fact i)) has-field-derivative
               f (Suc n) u * (z − u) ^ n / (fact n)) (at u)
      apply (intro derivative-eq-intros)+
      apply (force intro: u assms)
      apply (rule refl)+
      apply (auto simp: ac-simps)
      done
  }
  then show ?thesis
    apply (cut-tac complex-mvt-line [of w z λu. ∑ i = 0..n. f i u * (z−u) ^ i /
(fact i)
```

```
            λu. (f (Suc n) u * (z−u) ^ n / (fact n))])
   apply (auto simp add: intro: open-closed-segment)
   done
qed
```

47.15 Polynomal function extremal theorem, from HOL Light

```
lemma polyfun-extremal-lemma:
   fixes c :: nat ⇒ 'a::real-normed-div-algebra
  assumes 0 < e
   shows ∃M. ∀z. M ≤ norm(z) ⟶ norm (∑i≤n. c(i) * z^i) ≤ e * norm(z)
^ (Suc n)
proof (induct n)
  case 0 with assms
  show ?case
   apply (rule-tac x=norm (c 0) / e in exI)
   apply (auto simp: field-simps)
   done
next
  case (Suc n)
  obtain M where M: ⋀z. M ≤ norm z ⟹ norm (∑i≤n. c i * z^i) ≤ e *
norm z ^ Suc n
   using Suc assms by blast
  show ?case
  proof (rule exI [where x= max M (1 + norm(c(Suc n)) / e)], clarsimp simp
del: power-Suc)
   fix z::'a
   assume z1: M ≤ norm z and 1 + norm (c (Suc n)) / e ≤ norm z
   then have z2: e + norm (c (Suc n)) ≤ e * norm z
     using assms by (simp add: field-simps)
   have norm (∑i≤n. c i * z^i) ≤ e * norm z ^ Suc n
     using M [OF z1] by simp
    then have norm (∑i≤n. c i * z^i) + norm (c (Suc n) * z ^ Suc n) ≤ e *
norm z ^ Suc n + norm (c (Suc n) * z ^ Suc n)
     by simp
   then have norm ((∑i≤n. c i * z^i) + c (Suc n) * z ^ Suc n) ≤ e * norm z
^ Suc n + norm (c (Suc n) * z ^ Suc n)
     by (blast intro: norm-triangle-le elim: )
   also have ... ≤ (e + norm (c (Suc n))) * norm z ^ Suc n
     by (simp add: norm-power norm-mult algebra-simps)
   also have ... ≤ (e * norm z) * norm z ^ Suc n
     by (metis z2 mult.commute mult-left-mono norm-ge-zero norm-power)
   finally show norm ((∑i≤n. c i * z^i) + c (Suc n) * z ^ Suc n) ≤ e * norm
z ^ Suc (Suc n)
     by simp
  qed
qed

lemma polyfun-extremal:
```

```
    fixes c :: nat ⇒ 'a::real-normed-div-algebra
  assumes k: c k ≠ 0 1≤k and kn: k≤n
    shows eventually (λz. norm (∑ i≤n. c(i) * z^i) ≥ B) at-infinity
using kn
proof (induction n)
  case 0
  then show ?case
    using k  by simp
next
  case (Suc m)
  let ?even = ?case
  show ?even
  proof (cases c (Suc m) = 0)
    case True
    then show ?even using Suc k
      by auto (metis antisym-conv less-eq-Suc-le not-le)
  next
    case False
    then obtain M where M:
          ⋀z. M ≤ norm z ⟹ norm (∑ i≤m. c i * z^i) ≤ norm (c (Suc m)) / 2
* norm z ^ Suc m
      using polyfun-extremal-lemma [of norm(c (Suc m)) / 2 c m] Suc
      by auto
    have ∃ b. ∀ z. b ≤ norm z ⟶ B ≤ norm (∑ i≤Suc m. c i * z^i)
     proof (rule exI [where x=max M (max 1 (|B| / (norm(c (Suc m)) / 2)))],
clarsimp simp del: power-Suc)
      fix z::'a
      assume z1: M ≤ norm z 1 ≤ norm z
        and |B| * 2 / norm (c (Suc m)) ≤ norm z
      then have z2: |B| ≤ norm (c (Suc m)) * norm z / 2
        using False by (simp add: field-simps)
      have nz: norm z ≤ norm z ^ Suc m
     by (metis ‹1 ≤ norm z› One-nat-def less-eq-Suc-le power-increasing power-one-right
zero-less-Suc)
      have *: ⋀y x. norm (c (Suc m)) * norm z / 2 ≤ norm y − norm x ⟹ B
≤ norm (x + y)
        by (metis abs-le-iff add.commute norm-diff-ineq order-trans z2)
      have norm z * norm (c (Suc m)) + 2 * norm (∑ i≤m. c i * z^i)
            ≤ norm (c (Suc m)) * norm z + norm (c (Suc m)) * norm z ^ Suc m
        using M [of z] Suc z1  by auto
      also have ... ≤ 2 * (norm (c (Suc m)) * norm z ^ Suc m)
        using nz by (simp add: mult-mono del: power-Suc)
      finally show B ≤ norm ((∑ i≤m. c i * z^i) + c (Suc m) * z ^ Suc m)
        using Suc.IH
        apply (auto simp: eventually-at-infinity)
        apply (rule *)
        apply (simp add: field-simps norm-mult norm-power)
        done
    qed
```

```
    then show ?even
      by (simp add: eventually-at-infinity)
  qed
qed

end
```

48 Bounded Continuous Functions

```
theory Bounded-Continuous-Function
imports Integration
begin
```

48.1 Definition

```
definition bcontfun :: ('a::topological-space ⇒ 'b::metric-space) set
  where bcontfun = {f. continuous-on UNIV f ∧ bounded (range f)}

typedef (overloaded) ('a, 'b) bcontfun =
    bcontfun :: ('a::topological-space ⇒ 'b::metric-space) set
  by (auto simp: bcontfun-def intro: continuous-intros simp: bounded-def)

lemma bcontfunE:
  assumes f ∈ bcontfun
  obtains y where continuous-on UNIV f ⋀x. dist (f x) u ≤ y
  using assms unfolding bcontfun-def
  by (metis (lifting) bounded-any-center dist-commute mem-Collect-eq rangeI)

lemma bcontfunE':
  assumes f ∈ bcontfun
  obtains y where continuous-on UNIV f ⋀x. dist (f x) undefined ≤ y
  using assms bcontfunE
  by metis

lemma bcontfunI: continuous-on UNIV f ⟹ (⋀x. dist (f x) u ≤ b) ⟹ f ∈
bcontfun
  unfolding bcontfun-def
  by (metis (lifting, no-types) bounded-def dist-commute mem-Collect-eq rangeE)

lemma bcontfunI': continuous-on UNIV f ⟹ (⋀x. dist (f x) undefined ≤ b) ⟹
f ∈ bcontfun
  using bcontfunI by metis

lemma continuous-on-Rep-bcontfun[intro, simp]: continuous-on T (Rep-bcontfun
x)
  using Rep-bcontfun[of x]
  by (auto simp: bcontfun-def intro: continuous-on-subset)
```

instantiation *bcontfun* :: (*topological-space*, *metric-space*) *metric-space*
begin

definition *dist-bcontfun* :: ('*a*, '*b*) *bcontfun* ⇒ ('*a*, '*b*) *bcontfun* ⇒ *real*
where *dist-bcontfun f g* = (*SUP x*. *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*))

definition *uniformity-bcontfun* :: (('*a*, '*b*) *bcontfun* × ('*a*, '*b*) *bcontfun*) *filter*
where *uniformity-bcontfun* = (*INF e*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-bcontfun* :: ('*a*, '*b*) *bcontfun set* ⇒ *bool*
where *open-bcontfun S* = (∀ *x*∈*S*. ∀$_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *S*)

lemma *dist-bounded*:
fixes *f* :: ('*a*, '*b*) *bcontfun*
shows *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) ≤ *dist f g*
proof −
have *Rep-bcontfun f* ∈ *bcontfun* **by** (*rule Rep-bcontfun*)
from *bcontfunE'*[*OF this*] **obtain** *y* **where** *y*:
continuous-on UNIV (*Rep-bcontfun f*)
⋀*x*. *dist* (*Rep-bcontfun f x*) *undefined* ≤ *y*
by *auto*
have *Rep-bcontfun g* ∈ *bcontfun* **by** (*rule Rep-bcontfun*)
from *bcontfunE'*[*OF this*] **obtain** *z* **where** *z*:
continuous-on UNIV (*Rep-bcontfun g*)
⋀*x*. *dist* (*Rep-bcontfun g x*) *undefined* ≤ *z*
by *auto*
show *?thesis*
unfolding *dist-bcontfun-def*
proof (*intro cSUP-upper bdd-aboveI2*)
fix *x*
have *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) ≤
dist (*Rep-bcontfun f x*) *undefined* + *dist* (*Rep-bcontfun g x*) *undefined*
by (*rule dist-triangle2*)
also have ... ≤ *y* + *z*
using *y*(*2*)[*of x*] *z*(*2*)[*of x*] **by** (*rule add-mono*)
finally show *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) ≤ *y* + *z* **.**
qed *simp*
qed

lemma *dist-bound*:
fixes *f* :: ('*a*, '*b*) *bcontfun*
assumes ⋀*x*. *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) ≤ *b*
shows *dist f g* ≤ *b*
using *assms* **by** (*auto simp*: *dist-bcontfun-def intro*: *cSUP-least*)

lemma *dist-bounded-Abs*:
fixes *f g* :: '*a* ⇒ '*b*
assumes *f* ∈ *bcontfun g* ∈ *bcontfun*
shows *dist* (*f x*) (*g x*) ≤ *dist* (*Abs-bcontfun f*) (*Abs-bcontfun g*)

by (*metis Abs-bcontfun-inverse assms dist-bounded*)

lemma *const-bcontfun*: $(\lambda x::'a.\ b::'b) \in$ *bcontfun*
by (*auto intro*: *bcontfunI continuous-on-const*)

lemma *dist-fun-lt-imp-dist-val-lt*:
assumes *dist f g* $<$ *e*
shows *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) $<$ *e*
using *dist-bounded assms* **by** (*rule le-less-trans*)

lemma *dist-val-lt-imp-dist-fun-le*:
assumes $\forall$ *x*. *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) $<$ *e*
shows *dist f g* $\leq$ *e*
unfolding *dist-bcontfun-def*
proof (*intro cSUP-least*)
fix *x*
show *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) $\leq$ *e*
using *assms*[*THEN spec*[**where** *x*=*x*]] **by** (*simp add*: *dist-norm*)
qed *simp*

instance
proof
fix *f g h* :: (*'a*, *'b*) *bcontfun*
show *dist f g* = *0* $\longleftrightarrow$ *f* = *g*
proof
have $\bigwedge$*x*. *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) $\leq$ *dist f g*
by (*rule dist-bounded*)
also assume *dist f g* = *0*
finally show *f* = *g*
by (*auto simp*: *Rep-bcontfun-inject*[*symmetric*] *Abs-bcontfun-inverse*)
qed (*auto simp*: *dist-bcontfun-def SUP-def simp del*: *Sup-image-eq intro*!: *cSup-eq*)
show *dist f g* $\leq$ *dist f h* + *dist g h*
proof (*subst dist-bcontfun-def*, *safe intro*!: *cSUP-least*)
fix *x*
have *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) $\leq$
dist (*Rep-bcontfun f x*) (*Rep-bcontfun h x*) + *dist* (*Rep-bcontfun g x*) (*Rep-bcontfun h x*)
by (*rule dist-triangle2*)
also have *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun h x*) $\leq$ *dist f h*
by (*rule dist-bounded*)
also have *dist* (*Rep-bcontfun g x*) (*Rep-bcontfun h x*) $\leq$ *dist g h*
by (*rule dist-bounded*)
finally show *dist* (*Rep-bcontfun f x*) (*Rep-bcontfun g x*) $\leq$ *dist f h* + *dist g h*
by *simp*
qed
qed (*rule open-bcontfun-def uniformity-bcontfun-def*)+

end

lemma *closed-Pi-bcontfun*:
fixes I :: $'a$::*metric-space set*
and X :: $'a \Rightarrow 'b$::*complete-space set*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *closed* $(X\ i)$
shows *closed* $(Abs\text{-}bcontfun\ `\ (Pi\ I\ X \cap bcontfun))$
unfolding *closed-sequential-limits*
proof *safe*
fix $f\ l$
assume *seq*: $\forall n.\ f\ n \in Abs\text{-}bcontfun\ `\ (Pi\ I\ X \cap bcontfun)$ **and** *lim*: $f \longrightarrow l$
have *lim-fun*: $\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x.\ dist\ (Rep\text{-}bcontfun\ (f\ n)\ x)\ (Rep\text{-}bcontfun\ l\ x) < e$
using *LIMSEQ-imp-Cauchy*[*OF lim*, *simplified Cauchy-def*] *metric-LIMSEQ-D*[*OF lim*]
by (*intro uniformly-cauchy-imp-uniformly-convergent*[**where** $P{=}\lambda$-. *True*, *simplified*])
(*metis dist-fun-lt-imp-dist-val-lt*)+
show $l \in Abs\text{-}bcontfun\ `\ (Pi\ I\ X \cap bcontfun)$
proof (*rule*, *safe*)
fix x **assume** $x \in I$
then have *closed* $(X\ x)$
using *assms* **by** *simp*
moreover have *eventually* $(\lambda xa.\ Rep\text{-}bcontfun\ (f\ xa)\ x \in X\ x)$ *sequentially*
proof (*rule always-eventually*, *safe*)
fix i
from *seq*[*THEN spec*, *of i*] ‹$x \in I$›
show *Rep-bcontfun* $(f\ i)\ x \in X\ x$
by (*auto simp*: *Abs-bcontfun-inverse*)
qed
moreover note *sequentially-bot*
moreover have $(\lambda n.\ Rep\text{-}bcontfun\ (f\ n)\ x) \longrightarrow Rep\text{-}bcontfun\ l\ x$
using *lim-fun* **by** (*blast intro*!: *metric-LIMSEQ-I*)
ultimately show *Rep-bcontfun* $l\ x \in X\ x$
by (*rule Lim-in-closed-set*)
qed (*auto simp*: *Rep-bcontfun Rep-bcontfun-inverse*)
qed

48.2 Complete Space

instance *bcontfun* :: (*metric-space*, *complete-space*) *complete-space*
proof
fix f :: *nat* $\Rightarrow$ ($'a$, $'b$) *bcontfun*
assume *Cauchy f* — Cauchy equals uniform convergence
then obtain g **where** *limit-function*:
$\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ \forall x.\ dist\ (Rep\text{-}bcontfun\ (f\ n)\ x)\ (g\ x) < e$
using *uniformly-convergent-eq-cauchy*[*of* λ-. *True*
$\lambda n.\ Rep\text{-}bcontfun\ (f\ n)$]
unfolding *Cauchy-def*
by (*metis dist-fun-lt-imp-dist-val-lt*)

```
  then obtain N where fg-dist: — for an upper bound on g
    ∀ n≥N. ∀ x. dist (g x) ( Rep-bcontfun (f n) x) < 1
    by (force simp add: dist-commute)
  from bcontfunE'[OF Rep-bcontfun, of f N] obtain b where
    f-bound: ∀ x. dist (Rep-bcontfun (f N) x) undefined ≤ b
    by force
  have g ∈ bcontfun — The limit function is bounded and continuous
  proof (intro bcontfunI)
    show continuous-on UNIV g
      using bcontfunE[OF Rep-bcontfun] limit-function
       by (intro continuous-uniform-limit[where f=λn. Rep-bcontfun (f n) and
F=sequentially])
        (auto simp add: eventually-sequentially trivial-limit-def dist-norm)
  next
    fix x
    from fg-dist have dist (g x) (Rep-bcontfun (f N) x) < 1
      by (simp add: dist-norm norm-minus-commute)
    with dist-triangle[of g x undefined Rep-bcontfun (f N) x]
    show dist (g x) undefined ≤ 1 + b using f-bound[THEN spec, of x]
      by simp
  qed
  show convergent f
  proof (rule convergentI, subst lim-sequentially, safe)
    — The limit function converges according to its norm
    fix e :: real
    assume e > 0
    then have e/2 > 0 by simp
    with limit-function[THEN spec, ofe/2]
    have ∃ N. ∀ n≥N. ∀ x. dist (Rep-bcontfun (f n) x) (g x) < e/2
      by simp
    then obtain N where N: ∀ n≥N. ∀ x. dist (Rep-bcontfun (f n) x) (g x) < e
/ 2 by auto
    show ∃ N. ∀ n≥N. dist (f n) (Abs-bcontfun g) < e
    proof (rule, safe)
      fix n
      assume N ≤ n
      with N show dist (f n) (Abs-bcontfun g) < e
        using dist-val-lt-imp-dist-fun-le[of
          f n Abs-bcontfun g e/2]
          Abs-bcontfun-inverse[OF ‹g ∈ bcontfun›] ‹e > 0› by simp
    qed
  qed
qed
```

48.3 Supremum norm for a normed vector space

```
instantiation bcontfun :: (topological-space, real-normed-vector) real-vector
begin
```

definition $-f = \mathit{Abs\text{-}bcontfun}\ (\lambda x.\ -(\mathit{Rep\text{-}bcontfun}\ f\ x))$

definition $f + g = \mathit{Abs\text{-}bcontfun}\ (\lambda x.\ \mathit{Rep\text{-}bcontfun}\ f\ x + \mathit{Rep\text{-}bcontfun}\ g\ x)$

definition $f - g = \mathit{Abs\text{-}bcontfun}\ (\lambda x.\ \mathit{Rep\text{-}bcontfun}\ f\ x - \mathit{Rep\text{-}bcontfun}\ g\ x)$

definition $0 = \mathit{Abs\text{-}bcontfun}\ (\lambda x.\ 0)$

definition $\mathit{scaleR}\ r\ f = \mathit{Abs\text{-}bcontfun}\ (\lambda x.\ r *_R \mathit{Rep\text{-}bcontfun}\ f\ x)$

lemma *plus-cont*:
fixes $f\ g :: 'a \Rightarrow 'b$
assumes f: $f \in \mathit{bcontfun}$
and g: $g \in \mathit{bcontfun}$
shows $(\lambda x.\ f\ x + g\ x) \in \mathit{bcontfun}$
proof −
from *bcontfunE'*[*OF f*] **obtain** y **where** *continuous-on UNIV f* $\bigwedge x.$ *dist* $(f\ x)$ *undefined* $\leq y$
by *auto*
moreover
from *bcontfunE'*[*OF g*] **obtain** z **where** *continuous-on UNIV g* $\bigwedge x.$ *dist* $(g\ x)$ *undefined* $\leq z$
by *auto*
ultimately show *?thesis*
proof (*intro bcontfunI*)
fix x
have *dist* $(f\ x + g\ x)\ 0 =$ *dist* $(f\ x + g\ x)\ (0 + 0)$
by *simp*
also have $\ldots \leq$ *dist* $(f\ x)\ 0 +$ *dist* $(g\ x)\ 0$
by (*rule dist-triangle-add*)
also have $\ldots \leq$ *dist* $(\mathit{Abs\text{-}bcontfun}\ f)\ 0 +$ *dist* $(\mathit{Abs\text{-}bcontfun}\ g)\ 0$
unfolding *zero-bcontfun-def* **using** *assms*
by (*auto intro*!: *add-mono dist-bounded-Abs const-bcontfun*)
finally show *dist* $(f\ x + g\ x)\ 0 \leq$ *dist* $(\mathit{Abs\text{-}bcontfun}\ f)\ 0 +$ *dist* $(\mathit{Abs\text{-}bcontfun}\ g)\ 0$ **.**
qed (*simp add*: *continuous-on-add*)
qed

lemma *Rep-bcontfun-plus*[*simp*]: *Rep-bcontfun* $(f + g)\ x =$ *Rep-bcontfun* $f\ x +$ *Rep-bcontfun* $g\ x$
by (*simp add*: *plus-bcontfun-def Abs-bcontfun-inverse plus-cont Rep-bcontfun*)

lemma *uminus-cont*:
fixes $f :: 'a \Rightarrow 'b$
assumes $f \in \mathit{bcontfun}$
shows $(\lambda x.\ -f\ x) \in \mathit{bcontfun}$
proof −
from *bcontfunE*[*OF assms*, *of 0*] **obtain** y
where *continuous-on UNIV f* $\bigwedge x.$ *dist* $(f\ x)\ 0 \leq y$

by *auto*
then show *?thesis*
proof (*intro bcontfunI*)
fix x
assume $\bigwedge x.\ dist\ (f\ x)\ 0 \leq y$
then show $dist\ (-\ f\ x)\ 0 \leq y$
by (*subst dist-minus*[*symmetric*]) *simp*
qed (*simp add*: *continuous-on-minus*)
qed

lemma *Rep-bcontfun-uminus*[*simp*]: *Rep-bcontfun* $(-\ f)\ x = -$ *Rep-bcontfun* $f\ x$
by (*simp add*: *uminus-bcontfun-def Abs-bcontfun-inverse uminus-cont Rep-bcontfun*)

lemma *minus-cont*:
fixes $f\ g :: 'a \Rightarrow 'b$
assumes f: $f \in bcontfun$
and g: $g \in bcontfun$
shows $(\lambda x.\ f\ x\ -\ g\ x) \in bcontfun$
using *plus-cont* [*of* $f\ -\ g$] *assms*
by (*simp add*: *uminus-cont fun-Compl-def*)

lemma *Rep-bcontfun-minus*[*simp*]: *Rep-bcontfun* $(f\ -\ g)\ x =$ *Rep-bcontfun* $f\ x\ -$ *Rep-bcontfun* $g\ x$
by (*simp add*: *minus-bcontfun-def Abs-bcontfun-inverse minus-cont Rep-bcontfun*)

lemma *scaleR-cont*:
fixes $a :: real$
and $f :: 'a \Rightarrow 'b$
assumes $f \in bcontfun$
shows $(\lambda x.\ a *_R f\ x) \in bcontfun$
proof −
from *bcontfunE*[*OF assms*, *of 0*] **obtain** y
where *continuous-on UNIV* f $\bigwedge x.\ dist\ (f\ x)\ 0 \leq y$
by *auto*
then show *?thesis*
proof (*intro bcontfunI*)
fix x
assume $\bigwedge x.\ dist\ (f\ x)\ 0 \leq y$
then show $dist\ (a *_R f\ x)\ 0 \leq |a| * y$
by (*metis norm-cmul-rule-thm norm-conv-dist*)
qed (*simp add*: *continuous-intros*)
qed

lemma *Rep-bcontfun-scaleR*[*simp*]: *Rep-bcontfun* $(a *_R g)\ x = a *_R$ *Rep-bcontfun* $g\ x$
by (*simp add*: *scaleR-bcontfun-def Abs-bcontfun-inverse scaleR-cont Rep-bcontfun*)

instance
by *standard*

(*simp-all add*: *plus-bcontfun-def zero-bcontfun-def minus-bcontfun-def scaleR-bcontfun-def Abs-bcontfun-inverse Rep-bcontfun-inverse Rep-bcontfun algebra-simps plus-cont const-bcontfun minus-cont scaleR-cont*)

end

instantiation *bcontfun* :: (*topological-space*, *real-normed-vector*) *real-normed-vector*
begin

definition *norm-bcontfun* :: ('*a*, '*b*) *bcontfun* ⇒ *real*
where *norm-bcontfun f* = *dist f 0*

definition *sgn* (*f*::('*a*,'*b*) *bcontfun*) = *f* /$_R$ *norm f*

instance
proof
fix *a* :: *real*
fix *f g* :: ('*a*, '*b*) *bcontfun*
show *dist f g* = *norm* (*f* − *g*)
by (*simp add*: *norm-bcontfun-def dist-bcontfun-def zero-bcontfun-def Abs-bcontfun-inverse const-bcontfun norm-conv-dist*[*symmetric*] *dist-norm*)
show *norm* (*f* + *g*) ≤ *norm f* + *norm g*
unfolding *norm-bcontfun-def*
proof (*subst dist-bcontfun-def*, *safe intro*!: *cSUP-least*)
fix *x*
have *dist* (*Rep-bcontfun* (*f* + *g*) *x*) (*Rep-bcontfun 0 x*) ≤
dist (*Rep-bcontfun f x*) *0* + *dist* (*Rep-bcontfun g x*) *0*
by (*metis* (*hide-lams*, *no-types*) *Rep-bcontfun-minus Rep-bcontfun-plus diff-0-right dist-norm*
le-less-linear less-irrefl norm-triangle-lt)
also have *dist* (*Rep-bcontfun f x*) *0* ≤ *dist f 0*
using *dist-bounded*[*of f x 0*]
by (*simp add*: *Abs-bcontfun-inverse const-bcontfun zero-bcontfun-def*)
also have *dist* (*Rep-bcontfun g x*) *0* ≤ *dist g 0* **using** *dist-bounded*[*of g x 0*]
by (*simp add*: *Abs-bcontfun-inverse const-bcontfun zero-bcontfun-def*)
finally show *dist* (*Rep-bcontfun* (*f* + *g*) *x*) (*Rep-bcontfun 0 x*) ≤ *dist f 0* + *dist g 0* **by** *simp*
qed
show *norm* (*a* *$_R$ *f*) = |*a*| * *norm f*
proof −
have |*a*| * *Sup* (*range* (λ*x*. *dist* (*Rep-bcontfun f x*) *0*)) =
(*SUP i*:*range* (λ*x*. *dist* (*Rep-bcontfun f x*) *0*). |*a*| * *i*)
proof (*intro continuous-at-Sup-mono bdd-aboveI2*)
fix *x*
show *dist* (*Rep-bcontfun f x*) *0* ≤ *norm f* **using** *dist-bounded*[*of f x 0*]
by (*simp add*: *norm-bcontfun-def norm-conv-dist Abs-bcontfun-inverse zero-bcontfun-def*
const-bcontfun)
qed (*auto intro*!: *monoI mult-left-mono continuous-intros*)

moreover
have *range* ($\lambda x.$ *dist* (*Rep-bcontfun* (a $*_R$ f) x) *0*) =
($\lambda x.$ $|a|$ $*$ x) ‘ (*range* ($\lambda x.$ *dist* (*Rep-bcontfun* f x) *0*))
by (*auto simp*: *norm-conv-dist*[*symmetric*])
ultimately
show *norm* (a $*_R$ f) = $|a|$ $*$ *norm* f
by (*simp add*: *norm-bcontfun-def dist-bcontfun-def norm-conv-dist Abs-bcontfun-inverse zero-bcontfun-def const-bcontfun SUP-def del*: *Sup-image-eq*)
qed
qed (*auto simp*: *norm-bcontfun-def sgn-bcontfun-def*)

end

lemma *bcontfun-normI*: *continuous-on UNIV* f $\Longrightarrow$ ($\bigwedge x.$ *norm* (f x) $\leq$ b) $\Longrightarrow$ f $\in$ *bcontfun*
unfolding *norm-conv-dist*
by (*auto intro*: *bcontfunI*)

lemma *norm-bounded*:
fixes f :: ('*a*::*topological-space*, '*b*::*real-normed-vector*) *bcontfun*
shows *norm* (*Rep-bcontfun* f x) $\leq$ *norm* f
using *dist-bounded*[*of f x 0*]
by (*simp add*: *norm-bcontfun-def norm-conv-dist Abs-bcontfun-inverse zero-bcontfun-def const-bcontfun*)

lemma *norm-bound*:
fixes f :: ('*a*::*topological-space*, '*b*::*real-normed-vector*) *bcontfun*
assumes $\bigwedge x.$ *norm* (*Rep-bcontfun* f x) $\leq$ b
shows *norm* f $\leq$ b
using *dist-bound*[*of f 0 b*] *assms*
by (*simp add*: *norm-bcontfun-def norm-conv-dist Abs-bcontfun-inverse zero-bcontfun-def const-bcontfun*)

48.4 Continuously Extended Functions

definition *clamp* :: '*a*::*euclidean-space* $\Rightarrow$ '*a* $\Rightarrow$ '*a* $\Rightarrow$ '*a* **where**
clamp a b x = ($\sum i{\in}Basis.$ (*if* $x{\cdot}i$ $<$ $a{\cdot}i$ *then* $a{\cdot}i$ *else if* $x{\cdot}i$ $\leq$ $b{\cdot}i$ *then* $x{\cdot}i$ *else* $b{\cdot}i$) $*_R$ i)

definition *ext-cont* :: ('*a*::*euclidean-space* $\Rightarrow$ '*b*::*real-normed-vector*) $\Rightarrow$ '*a* $\Rightarrow$ '*a* $\Rightarrow$ ('*a*, '*b*) *bcontfun*
where *ext-cont* f a b = *Abs-bcontfun* (($\lambda x.$ f (*clamp* a b x)))

lemma *ext-cont-def'*:
ext-cont f a b = *Abs-bcontfun* ($\lambda x.$
f ($\sum i{\in}Basis.$ (*if* $x{\cdot}i$ $<$ $a{\cdot}i$ *then* $a{\cdot}i$ *else if* $x{\cdot}i$ $\leq$ $b{\cdot}i$ *then* $x{\cdot}i$ *else* $b{\cdot}i$) $*_R$ i))
unfolding *ext-cont-def clamp-def* **..**

lemma *clamp-in-interval*:

assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \leq b \cdot i$
shows *clamp a b x* $\in$ *cbox a b*
unfolding *clamp-def*
using *box-ne-empty(1)*[*of a b*] *assms* **by** (*auto simp*: *cbox-def*)

lemma *dist-clamps-le-dist-args*:
fixes $x :: 'a$::*euclidean-space*
assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \leq b \cdot i$
shows *dist* (*clamp a b y*) (*clamp a b x*) $\leq$ *dist y x*
proof −
from *box-ne-empty(1)*[*of a b*] *assms* **have** ($\forall\, i{\in}Basis.\ a \cdot i \leq b \cdot i$)
by (*simp add*: *cbox-def*)
then have ($\sum i{\in}Basis.$ (*dist* (*clamp a b y* $\cdot$ *i*) (*clamp a b x* $\cdot$ *i*))^2) $\leq$
($\sum i{\in}Basis.$ (*dist* (*y* $\cdot$ *i*) (*x* $\cdot$ *i*))^2)
by (*auto intro*!: *setsum-mono simp*: *clamp-def dist-real-def abs-le-square-iff*[*symmetric*])
then show *?thesis*
by (*auto intro*: *real-sqrt-le-mono*
simp: *euclidean-dist-l2*[**where** *y*=*x*] *euclidean-dist-l2*[**where** *y*=*clamp a b x*]
setL2-def)
qed

lemma *clamp-continuous-at*:
fixes *f* :: $'a$::*euclidean-space* $\Rightarrow$ $'b$::*metric-space*
and *x* :: $'a$
assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \leq b \cdot i$
and *f-cont*: *continuous-on* (*cbox a b*) *f*
shows *continuous* (*at x*) ($\lambda x.$ *f* (*clamp a b x*))
unfolding *continuous-at-eps-delta*
proof *safe*
fix *x* :: $'a$
fix *e* :: *real*
assume *e* > *0*
moreover have *clamp a b x* $\in$ *cbox a b*
by (*simp add*: *clamp-in-interval assms*)
moreover note *f-cont*[*simplified continuous-on-iff*]
ultimately
obtain *d* **where** *d*: *0* < *d*
$\bigwedge x'.\ x' \in cbox\ a\ b \Longrightarrow dist\ x'\ (clamp\ a\ b\ x) < d \Longrightarrow dist\ (f\ x')\ (f\ (clamp\ a\ b\ x)) < e$
by *force*
show $\exists\, d{>}0.\ \forall x'.\ dist\ x'\ x < d \longrightarrow$
dist (*f* (*clamp a b x'*)) (*f* (*clamp a b x*)) < *e*
by (*auto intro*!: *d clamp-in-interval assms dist-clamps-le-dist-args*[*THEN le-less-trans*])
qed

lemma *clamp-continuous-on*:
fixes *f* :: $'a$::*euclidean-space* $\Rightarrow$ $'b$::*metric-space*
assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \leq b \cdot i$
and *f-cont*: *continuous-on* (*cbox a b*) *f*

shows *continuous-on UNIV* $(\lambda x.\ f\ (clamp\ a\ b\ x))$
using *assms*
by (*auto intro*: *continuous-at-imp-continuous-on clamp-continuous-at*)

lemma *clamp-bcontfun*:
fixes $f :: 'a{::}euclidean\text{-}space \Rightarrow 'b{::}real\text{-}normed\text{-}vector$
assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \leq b \cdot i$
and *continuous*: *continuous-on* $(cbox\ a\ b)\ f$
shows $(\lambda x.\ f\ (clamp\ a\ b\ x)) \in bcontfun$
proof −
have *bounded* $(f\ `\ (cbox\ a\ b))$
by (*rule compact-continuous-image*[*OF continuous compact-cbox*[*of a b*], *THEN compact-imp-bounded*])
then obtain *c* **where** *f-bound*: $\forall x{\in}f\ `\ cbox\ a\ b.\ norm\ x \leq c$
by (*auto simp add*: *bounded-pos*)
show $(\lambda x.\ f\ (clamp\ a\ b\ x)) \in bcontfun$
proof (*intro bcontfun-normI*)
fix *x*
show *norm* $(f\ (clamp\ a\ b\ x)) \leq c$
using *clamp-in-interval*[*OF assms*(*1*), *of x*] *f-bound*
by (*simp add*: *ext-cont-def*)
qed (*simp add*: *clamp-continuous-on assms*)
qed

lemma *integral-clamp*:
integral $\{t0{::}real..clamp\ t0\ t1\ x\}\ f =$
(*if* $x < t0$ *then 0 else if* $x \leq t1$ *then integral* $\{t0..x\}\ f$ *else integral* $\{t0..t1\}\ f$)
by (*auto simp*: *clamp-def*)

declare [[*coercion Rep-bcontfun*]]

lemma *ext-cont-cancel*[*simp*]:
fixes *x a b* :: $'a{::}euclidean\text{-}space$
assumes *x*: $x \in cbox\ a\ b$
and *continuous-on* $(cbox\ a\ b)\ f$
shows $ext\text{-}cont\ f\ a\ b\ x = f\ x$
using *assms*
unfolding *ext-cont-def*
proof (*subst Abs-bcontfun-inverse*[*OF clamp-bcontfun*])
show $f\ (clamp\ a\ b\ x) = f\ x$
using *x* **unfolding** *clamp-def mem-box*
by (*intro arg-cong*[**where** *f*=*f*] *euclidean-eqI*[**where** $'a{=}'a$]) (*simp add*: *not-less*)
qed (*auto simp*: *cbox-def*)

lemma *ext-cont-cong*:
assumes *t0* = *s0*
and *t1* = *s1*
and $\bigwedge t.\ t \in (cbox\ t0\ t1) \Longrightarrow f\ t = g\ t$

and *continuous-on* (*cbox t0 t1*) *f*
and *continuous-on* (*cbox s0 s1*) *g*
and *ord*: $\bigwedge i.\ i \in Basis \Longrightarrow t0 \cdot i \leq t1 \cdot i$
shows *ext-cont f t0 t1* = *ext-cont g s0 s1*
unfolding *assms ext-cont-def*
using *assms clamp-in-interval*[*OF ord*]
by (*subst Rep-bcontfun-inject*[*symmetric*]) *simp*

end

49 The Bernstein-Weierstrass and Stone-Weierstrass Theorems

By L C Paulson (2015)

theory *Weierstrass*
imports *Uniform-Limit Path-Connected*
begin

49.1 Bernstein polynomials

definition *Bernstein* :: [*nat*,*nat*,*real*] ⇒ *real* **where**
Bernstein n k x ≡ *of-nat* (*n choose k*) ∗ *x* ^ *k* ∗ (*1* − *x*) ^ (*n* − *k*)

lemma *Bernstein-nonneg*: ⟦*0* ≤ *x*; *x* ≤ *1*⟧ ⟹ *0* ≤ *Bernstein n k x*
by (*simp add*: *Bernstein-def*)

lemma *Bernstein-pos*: ⟦*0* < *x*; *x* < *1*; *k* ≤ *n*⟧ ⟹ *0* < *Bernstein n k x*
by (*simp add*: *Bernstein-def*)

lemma *sum-Bernstein* [*simp*]: (∑ *k* = *0*..*n*. *Bernstein n k x*) = *1*
using *binomial-ring* [*of x 1*−*x n*]
by (*simp add*: *Bernstein-def*)

lemma *binomial-deriv1*:
(∑*k*=*0*..*n*. (*of-nat k* ∗ *of-nat* (*n choose k*)) ∗ *a*^(*k*−*1*) ∗ *b*^(*n*−*k*)) = *real-of-nat n* ∗ (*a*+*b*) ^ (*n*−*1*)
apply (*rule DERIV-unique* [**where** *f* = λ*a*. (*a*+*b*)^*n* **and** *x*=*a*])
apply (*subst binomial-ring*)
apply (*rule derivative-eq-intros setsum.cong* | *simp*)+
done

lemma *binomial-deriv2*:
(∑*k*=*0*..*n*. (*of-nat k* ∗ *of-nat* (*k*−*1*) ∗ *of-nat* (*n choose k*)) ∗ *a*^(*k*−*2*) ∗ *b*^(*n*−*k*)) =
of-nat n ∗ *of-nat* (*n*−*1*) ∗ (*a*+*b*::*real*) ^ (*n*−*2*)
apply (*rule DERIV-unique* [**where** *f* = λ*a*. *of-nat n* ∗ (*a*+*b*::*real*) ^ (*n*−*1*) **and** *x*=*a*])
apply (*subst binomial-deriv1* [*symmetric*])

apply (*rule derivative-eq-intros setsum.cong* | *simp add*: *Num.numeral-2-eq-2*)+
done

lemma *sum-k-Bernstein* [*simp*]: $(\sum k = 0..n.\ real\ k * Bernstein\ n\ k\ x) = of\text{-}nat\ n * x$
apply (*subst binomial-deriv1* [*of n x 1−x*, *simplified*, *symmetric*])
apply (*simp add*: *setsum-left-distrib*)
apply (*auto simp*: *Bernstein-def algebra-simps realpow-num-eq-if intro*!: *setsum.cong*)
done

lemma *sum-kk-Bernstein* [*simp*]: $(\sum k = 0..n.\ real\ k * (real\ k - 1) * Bernstein\ n\ k\ x) = real\ n * (real\ n - 1) * x^2$
proof −
have $(\sum k = 0..n.\ real\ k * (real\ k - 1) * Bernstein\ n\ k\ x) = real\text{-}of\text{-}nat\ n * real\text{-}of\text{-}nat\ (n - Suc\ 0) * x^2$
apply (*subst binomial-deriv2* [*of n x 1−x*, *simplified*, *symmetric*])
apply (*simp add*: *setsum-left-distrib*)
apply (*rule setsum.cong* [*OF refl*])
apply (*simp add*: *Bernstein-def power2-eq-square algebra-simps*)
apply (*rename-tac k*)
apply (*subgoal-tac* $k = 0 \vee k = 1 \vee (\exists k'.\ k = Suc\ (Suc\ k'))$)
apply (*force simp add*: *field-simps of-nat-Suc power2-eq-square*)
by *presburger*
also have $... = n * (n - 1) * x^2$
by *auto*
finally show *?thesis*
by *auto*
qed

49.2 Explicit Bernstein version of the 1D Weierstrass approximation theorem

lemma *Bernstein-Weierstrass*:
fixes $f :: real \Rightarrow real$
assumes *contf*: *continuous-on* $\{0..1\}$ f **and** *e*: $0 < e$
shows $\exists N.\ \forall n\ x.\ N \leq n \wedge x \in \{0..1\} \longrightarrow |f\ x - (\sum k = 0..n.\ f(k/n) * Bernstein\ n\ k\ x)| < e$
proof −
have *bounded* $(f\ `\ \{0..1\})$
using *compact-continuous-image compact-imp-bounded contf* **by** *blast*
then obtain M **where** M: $\bigwedge x.\ 0 \leq x \Longrightarrow x \leq 1 \Longrightarrow |f\ x| \leq M$
by (*force simp add*: *bounded-iff*)
then have *Mge0*: $0 \leq M$ **by** *force*
have *ucontf*: *uniformly-continuous-on* $\{0..1\}$ f
using *compact-uniformly-continuous contf* **by** *blast*
then obtain d **where** d: $d{>}0$ $\bigwedge x\ x'.\ [\![x \in \{0..1\};\ x' \in \{0..1\};\ |x' - x| < d]\!] \Longrightarrow |f\ x' - f\ x| < e/2$
apply (*rule uniformly-continuous-onE* [**where** $e = e/2$])

```
    using e by (auto simp: dist-norm)
  { fix n::nat and x::real
    assume n: Suc (nat⌈4*M/(e*d^2)⌉) ≤ n and x: 0 ≤ x x ≤ 1
    have 0 < n using n by simp
    have ed0: - (e * d^2) < 0
      using e ‹0<d› by simp
    also have ... ≤ M * 4
      using ‹0≤M› by simp
    finally have [simp]: real-of-int (nat ⌈4 * M / (e * d^2)⌉) = real-of-int ⌈4 * M
/ (e * d^2)⌉
      using ‹0≤M› e ‹0<d›
      by (simp add: of-nat-Suc field-simps)
    have 4*M/(e*d^2) + 1 ≤ real (Suc (nat⌈4*M/(e*d^2)⌉))
      by (simp add: of-nat-Suc real-nat-ceiling-ge)
    also have ... ≤ real n
      using n by (simp add: of-nat-Suc field-simps)
    finally have nbig: 4*M/(e*d^2) + 1 ≤ real n .
    have sum-bern: (∑k = 0..n. (x - k/n)^2 * Bernstein n k x) = x * (1 - x) /
n
    proof -
      have *: ⋀a b x::real. (a - b)^2 * x = a * (a - 1) * x + (1 - 2 * b) * a * x
+ b * b * x
        by (simp add: algebra-simps power2-eq-square)
      have (∑ k = 0..n. (k - n * x)^2 * Bernstein n k x) = n * x * (1 - x)
        apply (simp add: * setsum.distrib)
        apply (simp add: setsum-right-distrib [symmetric] mult.assoc)
        apply (simp add: algebra-simps power2-eq-square)
        done
      then have (∑ k = 0..n. (k - n * x)^2 * Bernstein n k x)/n^2 = x * (1 -
x) / n
        by (simp add: power2-eq-square)
      then show ?thesis
        using n by (simp add: setsum-divide-distrib divide-simps mult.commute
power2-commute)
    qed
    { fix k
      assume k: k ≤ n
      then have kn: 0 ≤ k / n k / n ≤ 1
        by (auto simp: divide-simps)
      consider (lessd) |x - k / n| < d | (ged) d ≤ |x - k / n|
        by linarith
      then have |(f x - f (k/n))| ≤ e/2 + 2 * M / d^2 * (x - k/n)^2
      proof cases
        case lessd
        then have |(f x - f (k/n))| < e/2
          using d x kn by (simp add: abs-minus-commute)
        also have ... ≤ (e/2 + 2 * M / d^2 * (x - k/n)^2)
          using Mge0 d by simp
        finally show ?thesis by simp
```

```
    next
      case ged
      then have dle: d^2 ≤ (x - k/n)^2
        by (metis d(1) less-eq-real-def power2-abs power-mono)
      have |(f x - f (k/n))| ≤ |f x| + |f (k/n)|
        by (rule abs-triangle-ineq4)
      also have ... ≤ M+M
        by (meson M add-mono-thms-linordered-semiring(1) kn x)
      also have ... ≤ 2 * M * ((x - k/n)^2 / d^2)
        apply simp
       apply (rule Rings.ordered-semiring-class.mult-left-mono [of 1 ((x - k/n)^2
/ d^2), simplified])
        using dle ‹d>0› ‹M≥0› by auto
      also have ... ≤ e/2 + 2 * M / d^2 * (x - k/n)^2
        using e  by simp
      finally show ?thesis .
      qed
  } note * = this
  have |f x - (∑ k = 0..n. f(k / n) * Bernstein n k x)| ≤ |∑ k = 0..n. (f x
- f(k / n)) * Bernstein n k x|
   by (simp add: setsum-subtractf setsum-right-distrib [symmetric] algebra-simps)
  also have ... ≤ (∑ k = 0..n. (e/2 + (2 * M / d^2) * (x - k / n)^2) * Bernstein
n k x)
    apply (rule order-trans [OF setsum-abs setsum-mono])
    using *
    apply (simp add: abs-mult Bernstein-nonneg x mult-right-mono)
    done
  also have ... ≤ e/2 + (2 * M) / (d^2 * n)
   apply (simp only: setsum.distrib Rings.semiring-class.distrib-right setsum-right-distrib
[symmetric] mult.assoc sum-bern)
    using ‹d>0› x
    apply (simp add: divide-simps Mge0 mult-le-one mult-left-le)
    done
  also have ... < e
    apply (simp add: field-simps)
    using ‹d>0› nbig e ‹n>0›
    apply (simp add: divide-simps algebra-simps)
    using ed0 by linarith
  finally have |f x - (∑k = 0..n. f (real k / real n) * Bernstein n k x)| < e .
 }
 then show ?thesis
  by auto
qed
```

49.3 General Stone-Weierstrass theorem

Source: Bruno Brosowski and Frank Deutsch. An Elementary Proof of the Stone-Weierstrass Theorem. Proceedings of the American Mathematical Society Volume 81, Number 1, January 1981. DOI: 10.2307/2043993

http://www.jstor.org/stable/2043993

locale *function-ring-on* =
  fixes R :: ('a::*t2-space* ⇒ *real*) *set* **and** s :: 'a *set*
  assumes *compact*: *compact s*
  assumes *continuous*: $f \in R \Longrightarrow$ *continuous-on s f*
  assumes *add*: $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x.\ f\ x + g\ x) \in R$
  assumes *mult*: $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x.\ f\ x * g\ x) \in R$
  assumes *const*: $(\lambda\text{-}.\ c) \in R$
  assumes *separable*: $x \in s \Longrightarrow y \in s \Longrightarrow x \neq y \Longrightarrow \exists f{\in}R.\ f\ x \neq f\ y$

begin
  lemma *minus*: $f \in R \Longrightarrow (\lambda x.\ -\ f\ x) \in R$
    by (*frule mult* [*OF const* [*of* −*1*]]) *simp*

  lemma *diff*: $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x.\ f\ x - g\ x) \in R$
    unfolding *diff-conv-add-uminus* **by** (*metis add minus*)

  lemma *power*: $f \in R \Longrightarrow (\lambda x.\ f\ x\ \hat{}\ n) \in R$
    by (*induct n*) (*auto simp*: *const mult*)

  lemma *setsum*: $[\![$*finite I*; $\bigwedge i.\ i \in I \Longrightarrow f\ i \in R]\!] \Longrightarrow (\lambda x.\ \sum i \in I.\ f\ i\ x) \in R$
    by (*induct I rule*: *finite-induct*; *simp add*: *const add*)

  lemma *setprod*: $[\![$*finite I*; $\bigwedge i.\ i \in I \Longrightarrow f\ i \in R]\!] \Longrightarrow (\lambda x.\ \prod i \in I.\ f\ i\ x) \in R$
    by (*induct I rule*: *finite-induct*; *simp add*: *const mult*)

  definition *normf* :: ('a::*t2-space* ⇒ *real*) ⇒ *real*
    where *normf f* ≡ *SUP x*:*s*. |*f x*|

  lemma *normf-upper*: $[\![$*continuous-on s f*; $x \in s]\!] \Longrightarrow |f\ x| \leq$ *normf f*
    apply (*simp add*: *normf-def*)
    apply (*rule cSUP-upper*, *assumption*)
   by (*simp add*: *bounded-imp-bdd-above compact compact-continuous-image compact-imp-bounded continuous-on-rabs*)

  lemma *normf-least*: $s \neq \{\} \Longrightarrow (\bigwedge x.\ x \in s \Longrightarrow |f\ x| \leq M) \Longrightarrow$ *normf f* ≤ *M*
    by (*simp add*: *normf-def cSUP-least*)

end

lemma (**in** *function-ring-on*) *one*:
  assumes *U*: *open U* **and** *t0*: *t0* ∈ *s t0* ∈ *U* **and** *t1*: *t1* ∈ *s*−*U*
    shows ∃ *V*. *open V* ∧ *t0* ∈ *V* ∧ *s* ∩ *V* ⊆ *U* ∧
          (∀ *e*>*0*. ∃*f* ∈ *R*. *f* ' *s* ⊆ {*0*..*1*} ∧ (∀ *t* ∈ *s* ∩ *V*. *f t* < *e*) ∧ (∀ *t* ∈ *s* − *U*. *f t* > *1* − *e*))
proof −
  have ∃ *pt* ∈ *R*. *pt t0* = *0* ∧ *pt t* > *0* ∧ *pt* ' *s* ⊆ {*0*..*1*} **if** *t*: *t* ∈ *s* − *U* **for** *t*
  proof −
    have *t* ≠ *t0* **using** *t t0* **by** *auto*

then obtain *g* **where** *g*: $g \in R\ g\ t \neq g\ t0$
using *separable t0* **by** (*metis Diff-subset subset-eq t*)
def $h \equiv \lambda x.\ g\ x - g\ t0$
have $h \in R$
unfolding *h-def* **by** (*fast intro*: *g const diff*)
then have *hsq*: $(\lambda w.\ (h\ w)^2) \in R$
by (*simp add*: *power2-eq-square mult*)
have $h\ t \neq h\ t0$
by (*simp add*: *h-def g*)
then have $h\ t \neq 0$
by (*simp add*: *h-def*)
then have *ht2*: $0 < (h\ t)\ \hat{}\ 2$
by *simp*
also have $... \leq$ *normf* $(\lambda w.\ (h\ w)^2)$
using *t normf-upper* [**where** $x{=}t$] *continuous* [*OF hsq*] **by** *force*
finally have *nfp*: $0 <$ *normf* $(\lambda w.\ (h\ w)^2)$ **.**
def $p \equiv \lambda x.\ (1\ /$ *normf* $(\lambda w.\ (h\ w)^2)) * (h\ x)\ \hat{}\ 2$
have $p \in R$
unfolding *p-def* **by** (*fast intro*: *hsq const mult*)
moreover have $p\ t0 = 0$
by (*simp add*: *p-def h-def*)
moreover have $p\ t > 0$
using *nfp ht2* **by** (*simp add*: *p-def*)
moreover have $\bigwedge x.\ x \in s \Longrightarrow p\ x \in \{0..1\}$
using *nfp normf-upper* [*OF continuous* [*OF hsq*]] **by** (*auto simp*: *p-def*)
ultimately show $\exists pt \in R.\ pt\ t0 = 0 \wedge pt\ t > 0 \wedge pt\ `\ s \subseteq \{0..1\}$
by *auto*
qed
then obtain *pf* **where** *pf*: $\bigwedge t.\ t \in s - U \Longrightarrow pf\ t \in R \wedge pf\ t\ t0 = 0 \wedge pf\ t\ t > 0$
and *pf01*: $\bigwedge t.\ t \in s - U \Longrightarrow pf\ t\ `\ s \subseteq \{0..1\}$
by *metis*
have *com-sU*: *compact* $(s - U)$
using *compact closed-inter-compact U* **by** (*simp add*: *Diff-eq compact-inter-closed open-closed*)
have $\bigwedge t.\ t \in s - U \Longrightarrow \exists A.\ open\ A \wedge A \cap s = \{x{\in}s.\ 0 < pf\ t\ x\}$
apply (*rule open-Collect-positive*)
by (*metis pf continuous*)
then obtain *Uf* **where** *Uf*: $\bigwedge t.\ t \in s - U \Longrightarrow open\ (Uf\ t) \wedge (Uf\ t) \cap s = \{x{\in}s.\ 0 < pf\ t\ x\}$
by *metis*
then have *open-Uf*: $\bigwedge t.\ t \in s - U \Longrightarrow open\ (Uf\ t)$
by *blast*
have *tUft*: $\bigwedge t.\ t \in s - U \Longrightarrow t \in Uf\ t$
using *pf Uf* **by** *blast*
then have $*$: $s - U \subseteq (\bigcup x \in s - U.\ Uf\ x)$
by *blast*
obtain *subU* **where** *subU*: $subU \subseteq s - U$ *finite subU* $s - U \subseteq (\bigcup x \in subU.\ Uf\ x)$

```
    by (blast intro: that open-Uf compactE-image [OF com-sU - *])
  then have [simp]: subU ≠ {}
    using t1 by auto
  then have cardp: card subU > 0 using subU
    by (simp add: card-gt-0-iff)
  def p ≡ λx. (1 / card subU) * (∑ t ∈ subU. pf t x)
  have pR: p ∈ R
    unfolding p-def using subU pf by (fast intro: pf const mult setsum)
  have pt0 [simp]: p t0 = 0
    using subU pf by (auto simp: p-def intro: setsum.neutral)
  have pt-pos: p t > 0 if t: t ∈ s−U for t
  proof −
    obtain i where i: i ∈ subU t ∈ Uf i using subU t by blast
    show ?thesis
      using subU i t
      apply (clarsimp simp: p-def divide-simps)
      apply (rule setsum-pos2 [OF ‹finite subU›])
      using Uf t pf01 apply auto
      apply (force elim!: subsetCE)
      done
  qed
  have p01: p x ∈ {0..1} if t: x ∈ s for x
  proof −
    have 0 ≤ p x
      using subU cardp t
      apply (simp add: p-def divide-simps setsum-nonneg)
      apply (rule setsum-nonneg)
      using pf01 by force
    moreover have p x ≤ 1
      using subU cardp t
      apply (simp add: p-def divide-simps setsum-nonneg)
      apply (rule setsum-bounded-above [where 'a=real and K=1, simplified])
      using pf01 by force
    ultimately show ?thesis
      by auto
  qed
  have compact (p ‘ (s−U))
   by (meson Diff-subset com-sU compact-continuous-image continuous continuous-on-subset
pR)
  then have open (− (p ‘ (s−U)))
    by (simp add: compact-imp-closed open-Compl)
  moreover have 0 ∈ − (p ‘ (s−U))
    by (metis (no-types) ComplI image-iff not-less-iff-gr-or-eq pt-pos)
  ultimately obtain delta0 where delta0: delta0 > 0 ball 0 delta0 ⊆ − (p ‘
(s−U))
    by (auto simp: elim!: openE)
  then have pt-delta: ⋀x. x ∈ s−U ⟹ p x ≥ delta0
    by (force simp: ball-def dist-norm dest: p01)
  def δ ≡ delta0/2
```

have *delta0* $\leq$ *1* **using** *delta0 p01* [*of t1*] *t1*
  by (*force simp*: *ball-def dist-norm dest*: *p01*)
with *delta0* **have** *δ01*: *0* $<$ *δ δ* $<$ *1*
  by (*auto simp*: *δ-def*)
have *pt-δ*: $\bigwedge x.\ x \in s - U \Longrightarrow p\ x \geq \delta$
  using *pt-delta delta0* **by** (*force simp*: *δ-def*)
have $\exists A.$ *open* $A \wedge A \cap s = \{x{\in}s.\ p\ x < \delta/2\}$
  by (*rule open-Collect-less-Int* [*OF continuous* [*OF pR*] *continuous-on-const*])
then obtain *V* **where** *V*: *open V V* $\cap$ *s* $=$ $\{x{\in}s.\ p\ x < \delta/2\}$
  by *blast*
def *k* $\equiv$ *nat*$\lfloor 1/\delta \rfloor$ + *1*
have *k*$>$*0* **by** (*simp add*: *k-def*)
have *k*$-$*1* $\leq$ *1*$/\delta$
  using *δ01* **by** (*simp add*: *k-def*)
with *δ01* **have** *k* $\leq$ (*1*+*δ*)$/\delta$
  by (*auto simp*: *algebra-simps add-divide-distrib*)
also have ... $<$ *2*$/\delta$
  using *δ01* **by** (*auto simp*: *divide-simps*)
finally have *k2δ*: *k* $<$ *2*$/\delta$ **.**
have *1*$/\delta$ $<$ *k*
  using *δ01* **unfolding** *k-def* **by** *linarith*
with *δ01 k2δ* **have** *kδ*: *1* $<$ *k*$*\delta$ *k*$*\delta$ $<$ *2*
  by (*auto simp*: *divide-simps*)
def *q* $\equiv$ $\lambda n\ t.\ (1 - p\ t\ \hat{}\ n)\ \hat{}\ (k\hat{}n)$
have *qR*: *q n* $\in$ *R* **for** *n*
  by (*simp add*: *q-def const diff power pR*)
have *q01*: $\bigwedge n\ t.\ t \in s \Longrightarrow q\ n\ t \in \{0..1\}$
  using *p01* **by** (*simp add*: *q-def power-le-one algebra-simps*)
have *qt0* [*simp*]: $\bigwedge n.\ n{>}0 \Longrightarrow q\ n\ t0 = 1$
  using *t0 pf* **by** (*simp add*: *q-def power-0-left*)
{ fix *t* **and** *n*::*nat*
  assume *t*: *t* $\in$ *s* $\cap$ *V*
  with ‹*k*$>$*0*› *V* **have** *k* $*$ *p t* $<$ *k* $*$ *δ* / *2*
    by *force*
  then have *1* $-$ (*k* $*$ *δ* / *2*)$\hat{}n \leq 1 - (k * p\ t)\hat{}n$
    using ‹*k*$>$*0*› *p01 t* **by** (*simp add*: *power-mono*)
  also have ... $\leq$ *q n t*
    using *Bernoulli-inequality* [*of* $-$ ((*p t*)$\hat{}n$) *k*$\hat{}n$]
    apply (*simp add*: *q-def*)
    by (*metis IntE atLeastAtMost-iff p01 power-le-one power-mult-distrib t*)
  finally have *1* $-$ (*k* $*$ *δ* / *2*) $\hat{}$ *n* $\leq$ *q n t* **.**
} **note** *limitV* $=$ *this*
{ fix *t* **and** *n*::*nat*
  assume *t*: *t* $\in$ *s* $-$ *U*
  with ‹*k*$>$*0*› *U* **have** *k* $*$ *δ* $\leq$ *k* $*$ *p t*
    by (*simp add*: *pt-δ*)
  with *kδ* **have** *kpt*: *1* $<$ *k* $*$ *p t*
    by (*blast intro*: *less-le-trans*)
  have *ptn-pos*: *0* $<$ *p t* $\hat{}$ *n*

```
    using pt-pos [OF t] by simp
  have ptn-le: p t ^ n ≤ 1
    by (meson DiffE atLeastAtMost-iff p01 power-le-one t)
  have q n t = (1/(k^n * (p t)^n)) * (1 - p t ^ n) ^ (k^n) * k^n * (p t)^n
    using pt-pos [OF t] ‹k>0› by (simp add: q-def)
  also have ... ≤ (1/(k * (p t))^n) * (1 - p t ^ n) ^ (k^n) * (1 + k^n * (p
t)^n)
    using pt-pos [OF t] ‹k>0›
    apply simp
    apply (simp only: times-divide-eq-right [symmetric])
    apply (rule mult-left-mono [of 1::real, simplified])
    apply (simp-all add: power-mult-distrib)
    apply (rule zero-le-power)
    using ptn-le by linarith
  also have ... ≤ (1/(k * (p t))^n) * (1 - p t ^ n) ^ (k^n) * (1 + (p t)^n) ^
(k^n)
    apply (rule mult-left-mono [OF Bernoulli-inequality [of p t ^ n k^n]])
    using ‹k>0› ptn-pos ptn-le
    apply (auto simp: power-mult-distrib)
    done
  also have ... = (1/(k * (p t))^n) * (1 - p t ^ (2*n)) ^ (k^n)
    using pt-pos [OF t] ‹k>0›
    by (simp add: algebra-simps power-mult power2-eq-square power-mult-distrib
[symmetric])
  also have ... ≤ (1/(k * (p t))^n) * 1
    apply (rule mult-left-mono [OF power-le-one])
    using pt-pos ‹k>0› p01 power-le-one t apply auto
    done
  also have ... ≤ (1 / (k*δ))^n
    using ‹k>0› δ01  power-mono pt-δ t
    by (fastforce simp: field-simps)
  finally have q n t ≤ (1 / (real k * δ)) ^ n  .
 } note limitNonU = this
 def NN ≡ λe. 1 + nat ⌈max (ln e / ln (real k * δ / 2)) (- ln e / ln (real k *
δ))⌉
 have NN: of-nat (NN e) > ln e / ln (real k * δ / 2)  of-nat (NN e) > - ln e /
ln (real k * δ)
          if 0<e for e
    unfolding NN-def  by linarith+
 have NN1: ⋀e. e>0 ⟹ (k * δ / 2)^NN e < e
  apply (subst Transcendental.ln-less-cancel-iff [symmetric])
   prefer 3 apply (subst ln-realpow)
  using ‹k>0› ‹δ>0› NN  kδ
  apply (force simp add: field-simps)+
  done
 have NN0: ⋀e. e>0 ⟹ (1/(k*δ))^NN e < e
  apply (subst Transcendental.ln-less-cancel-iff [symmetric])
   prefer 3 apply (subst ln-realpow)
  using ‹k>0› ‹δ>0› NN kδ
```

```
    apply (force simp add: field-simps ln-div)+
    done
  { fix t and e::real
    assume e>0
    have t ∈ s ∩ V ⟹ 1 − q (NN e) t < e t ∈ s − U ⟹ q (NN e) t < e
    proof −
      assume t: t ∈ s ∩ V
      show 1 − q (NN e) t < e
       by (metis add.commute diff-le-eq not-le limitV [OF t] less-le-trans [OF NN1
[OF ‹e>0›]])
    next
      assume t: t ∈ s − U
      show q (NN e) t < e
      using limitNonU [OF t] less-le-trans [OF NN0 [OF ‹e>0›]] not-le by blast
    qed
  } then have ⋀e. e > 0 ⟹ ∃f∈R. f ‘ s ⊆ {0..1} ∧ (∀t ∈ s ∩ V. f t < e) ∧
(∀t ∈ s − U. 1 − e < f t)
    using q01
    by (rule-tac x=λx. 1 − q (NN e) x in bexI) (auto simp: algebra-simps intro:
diff const qR)
  moreover have t0V: t0 ∈ V  s ∩ V ⊆ U
    using pt-δ t0 U V δ01  by fastforce+
  ultimately show ?thesis using V t0V
    by blast
qed
```

Non-trivial case, with A and B both non-empty

```
lemma (in function-ring-on) two-special:
  assumes A: closed A A ⊆ s a ∈ A
      and B: closed B B ⊆ s b ∈ B
      and disj: A ∩ B = {}
      and e: 0 < e e < 1
    shows ∃f ∈ R. f ‘ s ⊆ {0..1} ∧ (∀x ∈ A. f x < e) ∧ (∀x ∈ B. f x > 1 − e)
proof −
  { fix w
    assume w ∈ A
    then have open ( − B) b ∈ s w ∉ B w ∈ s
      using assms by auto
    then have ∃V. open V ∧ w ∈ V ∧ s ∩ V ⊆ −B ∧
              (∀e>0. ∃f ∈ R. f ‘ s ⊆ {0..1} ∧ (∀x ∈ s ∩ V. f x < e) ∧ (∀x ∈ s
∩ B. f x > 1 − e))
      using one [of −B w b] assms ‹w ∈ A› by simp
  }
  then obtain Vf where Vf:
        ⋀w. w ∈ A ⟹ open (Vf w) ∧ w ∈ Vf w ∧ s ∩ Vf w ⊆ −B ∧
                    (∀e>0. ∃f ∈ R. f ‘ s ⊆ {0..1} ∧ (∀x ∈ s ∩ Vf w. f x < e) ∧
(∀x ∈ s ∩ B. f x > 1 − e))
    by metis
  then have open-Vf: ⋀w. w ∈ A ⟹ open (Vf w)
```

by *blast*
have *tVft*: $\bigwedge w.\ w \in A \Longrightarrow w \in Vf\ w$
using *Vf* **by** *blast*
then have *setsum-max-0*: $A \subseteq (\bigcup x \in A.\ Vf\ x)$
by *blast*
have *com-A*: *compact A* **using** *A*
by (*metis compact compact-inter-closed inf.absorb-iff2*)
obtain *subA* **where** *subA*: $subA \subseteq A$ *finite subA* $A \subseteq (\bigcup x \in subA.\ Vf\ x)$
by (*blast intro*: *that open-Vf compactE-image* [*OF com-A - setsum-max-0*])
then have [*simp*]: $subA \neq \{\}$
using ‹$a \in A$› **by** *auto*
then have *cardp*: *card subA* > *0* **using** *subA*
by (*simp add*: *card-gt-0-iff*)
have $\bigwedge w.\ w \in A \Longrightarrow \exists f \in R.\ f\ `\ s \subseteq \{0..1\} \wedge (\forall x \in s \cap Vf\ w.\ f\ x < e\ /\ card\ subA) \wedge (\forall x \in s \cap B.\ f\ x > 1 - e\ /\ card\ subA)$
using *Vf e cardp* **by** *simp*
then obtain *ff* **where** *ff*:
$\bigwedge w.\ w \in A \Longrightarrow ff\ w \in R \wedge ff\ w\ `\ s \subseteq \{0..1\} \wedge$
$(\forall x \in s \cap Vf\ w.\ ff\ w\ x < e\ /\ card\ subA) \wedge (\forall x \in s \cap B.\ ff\ w\ x > 1 - e\ /\ card\ subA)$
by *metis*
def *pff* $\equiv \lambda x.\ (\prod w \in subA.\ ff\ w\ x)$
have *pffR*: *pff* $\in R$
unfolding *pff-def* **using** *subA ff* **by** (*auto simp*: *intro*: *setprod*)
moreover
have *pff01*: *pff x* $\in \{0..1\}$ **if** *t*: $x \in s$ **for** *x*
proof −
have $0 \leq pff\ x$
using *subA cardp t*
apply (*simp add*: *pff-def divide-simps setsum-nonneg*)
apply (*rule Groups-Big.linordered-semidom-class.setprod-nonneg*)
using *ff* **by** *fastforce*
moreover have $pff\ x \leq 1$
using *subA cardp t*
apply (*simp add*: *pff-def divide-simps setsum-nonneg*)
apply (*rule setprod-mono* [**where** $g = \lambda x.\ 1$, *simplified*])
using *ff* **by** *fastforce*
ultimately show *?thesis*
by *auto*
qed
moreover
{ fix *v x*
assume *v*: $v \in subA$ **and** *x*: $x \in Vf\ v\ x \in s$
from *subA v* **have** $pff\ x = ff\ v\ x * (\prod w \in subA - \{v\}.\ ff\ w\ x)$
unfolding *pff-def* **by** (*metis setprod.remove*)
also have ... $\leq ff\ v\ x * 1$
apply (*rule Rings.ordered-semiring-class.mult-left-mono*)
apply (*rule setprod-mono* [**where** $g = \lambda x.\ 1$, *simplified*])
using *ff* [*THEN conjunct2*, *THEN conjunct1*] *v subA x*

```
    apply auto
    apply (meson atLeastAtMost-iff contra-subsetD imageI)
    apply (meson atLeastAtMost-iff contra-subsetD image-eqI)
    using atLeastAtMost-iff by blast
  also have ... < e / card subA
    using ff [THEN conjunct2, THEN conjunct2, THEN conjunct1] v subA x
    by auto
  also have ... ≤ e
    using cardp e by (simp add: divide-simps)
  finally have pff x < e .
 }
 then have ⋀x. x ∈ A ⟹ pff x < e
  using A Vf subA by (metis UN-E contra-subsetD)
 moreover
 { fix x
  assume x: x ∈ B
  then have x ∈ s
    using B by auto
  have 1 − e ≤ (1 − e / card subA) ^ card subA
    using Bernoulli-inequality [of −e / card subA card subA] e cardp
    by (auto simp: field-simps)
  also have ... = (∏ w ∈ subA. 1 − e / card subA)
    by (simp add: setprod-constant subA(2))
  also have ... < pff x
    apply (simp add: pff-def)
   apply (rule setprod-mono-strict [where f = λx. 1 − e / card subA, simplified])
    apply (simp-all add: subA(2))
    apply (intro ballI conjI)
    using e apply (force simp: divide-simps)
    using ff [THEN conjunct2, THEN conjunct2, THEN conjunct2] subA B x
    apply blast
    done
  finally have 1 − e < pff x .
 }
 ultimately
 show ?thesis by blast
qed

lemma (in function-ring-on) two:
 assumes A: closed A A ⊆ s
    and B: closed B B ⊆ s
    and disj: A ∩ B = {}
    and e: 0 < e e < 1
  shows ∃f ∈ R. f ' s ⊆ {0..1} ∧ (∀x ∈ A. f x < e) ∧ (∀x ∈ B. f x > 1 − e)
proof (cases A ≠ {} ∧ B ≠ {})
 case True then show ?thesis
  apply (simp add: ex-in-conv [symmetric])
  using assms
  apply safe
```

```
    apply (force simp add: intro!: two-special)
    done
next
  case False with e show ?thesis
    apply simp
    apply (erule disjE)
    apply (rule-tac [2] x=λx. 0 in bexI)
    apply (rule-tac x=λx. 1 in bexI)
    apply (auto simp: const)
    done
qed
```

The special case where f is non-negative and $e < (1::'a) \,/\, (3::'a)$

```
lemma (in function-ring-on) Stone-Weierstrass-special:
  assumes f: continuous-on s f and fpos: ⋀x. x ∈ s ⟹ f x ≥ 0
      and e: 0 < e e < 1/3
  shows ∃g ∈ R. ∀x∈s. |f x − g x| < 2∗e
proof −
  def n ≡ 1 + nat ⌈normf f / e⌉
  def A ≡ λj::nat. {x ∈ s. f x ≤ (j − 1/3)∗e}
  def B ≡ λj::nat. {x ∈ s. f x ≥ (j + 1/3)∗e}
  have ngt: (n−1) ∗ e ≥ normf f n≥1
    using e
    apply (simp-all add: n-def field-simps of-nat-Suc)
    by (metis real-nat-ceiling-ge mult.commute not-less pos-less-divide-eq)
  then have ge-fx: (n−1) ∗ e ≥ f x if x ∈ s for x
    using f normf-upper that by fastforce
  { fix j
    have A: closed (A j) A j ⊆ s
      apply (simp-all add: A-def Collect-restrict)
      apply (rule continuous-on-closed-Collect-le [OF f continuous-on-const])
      apply (simp add: compact compact-imp-closed)
      done
    have B: closed (B j) B j ⊆ s
      apply (simp-all add: B-def Collect-restrict)
      apply (rule continuous-on-closed-Collect-le [OF continuous-on-const f])
      apply (simp add: compact compact-imp-closed)
      done
    have disj: (A j) ∩ (B j) = {}
      using e by (auto simp: A-def B-def field-simps)
    have ∃f ∈ R. f ‘ s ⊆ {0..1} ∧ (∀x ∈ A j. f x < e/n) ∧ (∀x ∈ B j. f x > 1
− e/n)
      apply (rule two)
      using e A B disj ngt
      apply simp-all
      done
  }
  then obtain xf where xfR: ⋀j. xf j ∈ R and xf01: ⋀j. xf j ‘ s ⊆ {0..1}
                   and xfA: ⋀x j. x ∈ A j ⟹ xf j x < e/n
```

```
                    and xfB: ⋀x j. x ∈ B j ⟹ xf j x > 1 − e/n
  by metis
def g ≡ λx. e ∗ (∑ i≤n. xf i x)
have gR: g ∈ R
  unfolding g-def by (fast intro: mult const setsum xfR)
have gge0: ⋀x. x ∈ s ⟹ g x ≥ 0
 using e xf01 by (simp add: g-def zero-le-mult-iff image-subset-iff setsum-nonneg)
have A0: A 0 = {}
  using fpos e by (fastforce simp: A-def)
have An: A n = s
  using e ngt f normf-upper by (fastforce simp: A-def field-simps of-nat-diff)
have Asub: A j ⊆ A i if i≥j for i j
  using e that apply (clarsimp simp: A-def)
  apply (erule order-trans, simp)
  done
{ fix t
  assume t: t ∈ s
  def j ≡ LEAST j. t ∈ A j
  have jn: j ≤ n
    using t An by (simp add: Least-le j-def)
  have Aj: t ∈ A j
    using t An by (fastforce simp add: j-def intro: LeastI)
  then have Ai: t ∈ A i if i≥j for i
    using Asub [OF that] by blast
  then have fj1: f t ≤ (j − 1/3)∗e
    by (simp add: A-def)
  then have Anj: t ∉ A i if i<j for i
    using  Aj ‹i<j›
    apply (simp add: j-def)
    using not-less-Least by blast
  have j1: 1 ≤ j
    using A0 Aj j-def not-less-eq-eq by (fastforce simp add: j-def)
  then have Anj: t ∉ A (j−1)
    using Least-le by (fastforce simp add: j-def)
  then have fj2: (j − 4/3)∗e < f t
    using j1 t  by (simp add: A-def of-nat-diff)
  have ∗∗∗: xf i t ≤ e/n if i≥j for i
    using xfA [OF Ai] that by (simp add: less-eq-real-def)
  { fix i
    assume i+2 ≤ j
    then obtain d where i+2+d = j
      using le-Suc-ex that by blast
    then have t ∈ B i
      using Anj e ge-fx [OF t] ‹1 ≤ n› fpos [OF t] t
      apply (simp add: A-def B-def)
      apply (clarsimp simp add: field-simps of-nat-diff not-le of-nat-Suc)
      apply (rule order-trans [of - e ∗ 2 + (e ∗ (real d ∗ 3) + e ∗ (real i ∗ 3))])
      apply auto
      done
```

```
    then have xf i t > 1 - e/n
      by (rule xfB)
  } note **** = this
  have xf-le1: ⋀i. xf i t ≤ 1
    using xf01 t by force
  have g t = e * (∑ i<j. xf i t) + e * (∑ i=j..n. xf i t)
    using j1 jn e
    apply (simp add: g-def distrib-left [symmetric])
    apply (subst setsum.union-disjoint [symmetric])
    apply (auto simp: ivl-disj-un)
    done
  also have ... ≤ e*j + e * ((Suc n - j)*e/n)
    apply (rule add-mono)
    apply (simp-all only: mult-le-cancel-left-pos e)
     apply (rule setsum-bounded-above [OF xf-le1, where A = lessThan j, sim-
plified])
    using setsum-bounded-above [of {j..n} λi. xf i t, OF ***]
    apply simp
    done
  also have ... ≤ j*e + e*(n - j + 1)*e/n
    using ‹1 ≤ n› e by (simp add: field-simps del: of-nat-Suc)
  also have ... ≤ j*e + e*e
    using ‹1 ≤ n› e j1 by (simp add: field-simps del: of-nat-Suc)
  also have ... < (j + 1/3)*e
    using e by (auto simp: field-simps)
  finally have gj1: g t < (j + 1 / 3) * e .
  have gj2: (j - 4/3)*e < g t
  proof (cases 2 ≤ j)
    case False
    then have j=1 using j1 by simp
    with t gge0 e show ?thesis by force
  next
    case True
    then have (j - 4/3)*e < (j-1)*e - e^2
      using e by (auto simp: of-nat-diff algebra-simps power2-eq-square)
    also have ... < (j-1)*e - ((j - 1)/n) * e^2
      using e True jn by (simp add: power2-eq-square field-simps)
    also have ... = e * (j-1) * (1 - e/n)
      by (simp add: power2-eq-square field-simps)
    also have ... ≤ e * (∑ i≤j-2. xf i t)
      using e
      apply simp
       apply (rule order-trans [OF - setsum-bounded-below [OF less-imp-le [OF
****]]])
      using True
      apply (simp-all add: of-nat-Suc of-nat-diff)
      done
    also have ... ≤ g t
      using jn e
```

```
      using e xf01 t
      apply (simp add: g-def zero-le-mult-iff image-subset-iff setsum-nonneg)
      apply (rule Groups-Big.setsum-mono2, auto)
      done
    finally show ?thesis .
  qed
  have |f t − g t| < 2 ∗ e
    using fj1 fj2 gj1 gj2 by (simp add: abs-less-iff field-simps)
 }
 then show ?thesis
  by (rule-tac x=g in bexI) (auto intro: gR)
qed
```

The "unpretentious" formulation

```
lemma (in function-ring-on) Stone-Weierstrass-basic:
 assumes f: continuous-on s f and e: e > 0
 shows ∃ g ∈ R. ∀ x∈s. |f x − g x| < e
proof −
 have ∃ g ∈ R. ∀ x∈s. |(f x + normf f) − g x| < 2 ∗ min (e/2) (1/4)
  apply (rule Stone-Weierstrass-special)
  apply (rule Limits.continuous-on-add [OF f Topological-Spaces.continuous-on-const])
  using normf-upper [OF f] apply force
  apply (simp add: e, linarith)
  done
 then obtain g where g ∈ R ∀ x∈s. |g x − (f x + normf f)| < e
  by force
 then show ?thesis
  apply (rule-tac x=λx. g x − normf f in bexI)
  apply (auto simp: algebra-simps intro: diff const)
  done
qed


theorem (in function-ring-on) Stone-Weierstrass:
 assumes f: continuous-on s f
 shows ∃ F∈UNIV → R. LIM n sequentially. F n :> uniformly-on s f
proof −
 { fix e::real
  assume e: 0 < e
  then obtain N::nat where N: 0 < N 0 < inverse N inverse N < e
   by (auto simp: real-arch-inv [of e])
  { fix n :: nat and x :: 'a and g :: 'a ⇒ real
   assume n: N ≤ n ∀ x∈s. |f x − g x| < 1 / (1 + real n)
   assume x: x ∈ s
   have ¬ real (Suc n) < inverse e
    using ‹N ≤ n› N using less-imp-inverse-less by force
   then have 1 / (1 + real n) ≤ e
    using e by (simp add: field-simps of-nat-Suc)
   then have |f x − g x| < e
```

using *n*(2) *x* **by** *auto*
} **note** ∗ = *this*
have ∀$_F$ *n in sequentially*. ∀ *x*∈*s*. |*f x* − (*SOME g*. *g* ∈ *R* ∧ (∀ *x*∈*s*. |*f x* − *g x*| < *1* / (*1* + *real n*))) *x*| < *e*
apply (*rule eventually-sequentiallyI* [*of N*])
apply (*auto intro*: *someI2-bex* [*OF Stone-Weierstrass-basic* [*OF f*]] ∗)
done
} **then**
show *?thesis*
apply (*rule-tac x*=λ*n*::*nat*. *SOME g*. *g* ∈ *R* ∧ (∀ *x*∈*s*. |*f x* − *g x*| < *1* / (*1* + *n*)) **in** *bexI*)
prefer *2* **apply** (*force intro*: *someI2-bex* [*OF Stone-Weierstrass-basic* [*OF f*]])
unfolding *uniform-limit-iff*
apply (*auto simp*: *dist-norm abs-minus-commute*)
done
qed

A HOL Light formulation

corollary *Stone-Weierstrass-HOL*:
fixes *R* :: (′*a*::*t2-space* ⇒ *real*) *set* **and** *s* :: ′*a set*
assumes *compact s* ⋀*c*. *P*(λ*x*. *c*::*real*)
⋀*f*. *P f* ⟹ *continuous-on s f*
⋀*f g*. *P*(*f*) ∧ *P*(*g*) ⟹ *P*(λ*x*. *f x* + *g x*) ⋀*f g*. *P*(*f*) ∧ *P*(*g*) ⟹ *P*(λ*x*. *f x* ∗ *g x*)
⋀*x y*. *x* ∈ *s* ∧ *y* ∈ *s* ∧ ~(*x* = *y*) ⟹ ∃*f*. *P*(*f*) ∧ ~(*f x* = *f y*)
continuous-on s f
0 < *e*
shows ∃ *g*. *P*(*g*) ∧ (∀ *x* ∈ *s*. |*f x* − *g x*| < *e*)
proof −
interpret *PR*: *function-ring-on Collect P*
apply *unfold-locales*
using *assms*
by *auto*
show *?thesis*
using *PR.Stone-Weierstrass-basic* [*OF* ‹*continuous-on s f*› ‹*0* < *e*›]
by *blast*
qed

49.4 Polynomial functions

inductive *real-polynomial-function* :: (′*a*::*real-normed-vector* ⇒ *real*) ⇒ *bool* **where**
linear: *bounded-linear f* ⟹ *real-polynomial-function f*
| *const*: *real-polynomial-function* (λ*x*. *c*)
| *add*: ⟦*real-polynomial-function f*; *real-polynomial-function g*⟧ ⟹ *real-polynomial-function* (λ*x*. *f x* + *g x*)
| *mult*: ⟦*real-polynomial-function f*; *real-polynomial-function g*⟧ ⟹ *real-polynomial-function* (λ*x*. *f x* ∗ *g x*)

declare *real-polynomial-function.intros* [*intro*]

definition *polynomial-function* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ *bool*
where
polynomial-function p ≡ (∀*f*. *bounded-linear f* ⟶ *real-polynomial-function* (*f o p*))

lemma *real-polynomial-function-eq*: *real-polynomial-function p* = *polynomial-function p*
unfolding *polynomial-function-def*
proof
assume *real-polynomial-function p*
then show ∀*f*. *bounded-linear f* ⟶ *real-polynomial-function* (*f* ∘ *p*)
proof (*induction p rule*: *real-polynomial-function.induct*)
case (*linear h*) **then show** *?case*
by (*auto simp*: *bounded-linear-compose real-polynomial-function.linear*)
next
case (*const h*) **then show** *?case*
by (*simp add*: *real-polynomial-function.const*)
next
case (*add h*) **then show** *?case*
by (*force simp add*: *bounded-linear-def linear-add real-polynomial-function.add*)
next
case (*mult h*) **then show** *?case*
by (*force simp add*: *real-bounded-linear const real-polynomial-function.mult*)
qed
next
assume [*rule-format*, *OF bounded-linear-ident*]: ∀*f*. *bounded-linear f* ⟶ *real-polynomial-function* (*f* ∘ *p*)
then show *real-polynomial-function p*
by (*simp add*: *o-def*)
qed

lemma *polynomial-function-const* [*iff*]: *polynomial-function* (λ*x*. *c*)
by (*simp add*: *polynomial-function-def o-def const*)

lemma *polynomial-function-bounded-linear*:
bounded-linear f ⟹ *polynomial-function f*
by (*simp add*: *polynomial-function-def o-def bounded-linear-compose real-polynomial-function.linear*)

lemma *polynomial-function-id* [*iff*]: *polynomial-function*(λ*x*. *x*)
by (*simp add*: *polynomial-function-bounded-linear*)

lemma *polynomial-function-add* [*intro*]:
⟦*polynomial-function f*; *polynomial-function g*⟧ ⟹ *polynomial-function* (λ*x*. *f x* + *g x*)
by (*auto simp*: *polynomial-function-def bounded-linear-def linear-add real-polynomial-function.add o-def*)

lemma *polynomial-function-mult* [*intro*]:
assumes *f*: *polynomial-function f* **and** *g*: *polynomial-function g*
shows *polynomial-function* (λ*x*. *f x* ∗$_R$ *g x*)
using *g*
apply (*auto simp*: *polynomial-function-def bounded-linear-def Real-Vector-Spaces.linear.scaleR const real-polynomial-function.mult o-def*)
apply (*rule mult*)
using *f*
apply (*auto simp*: *real-polynomial-function-eq*)
done

lemma *polynomial-function-cmul* [*intro*]:
assumes *f*: *polynomial-function f*
shows *polynomial-function* (λ*x*. *c* ∗$_R$ *f x*)
by (*rule polynomial-function-mult* [*OF polynomial-function-const f*])

lemma *polynomial-function-minus* [*intro*]:
assumes *f*: *polynomial-function f*
shows *polynomial-function* (λ*x*. − *f x*)
using *polynomial-function-cmul* [*OF f*, *of* −*1*] **by** *simp*

lemma *polynomial-function-diff* [*intro*]:
⟦*polynomial-function f*; *polynomial-function g*⟧ ⟹ *polynomial-function* (λ*x*. *f x* − *g x*)
unfolding *add-uminus-conv-diff* [*symmetric*]
by (*metis polynomial-function-add polynomial-function-minus*)

lemma *polynomial-function-setsum* [*intro*]:
⟦*finite I*; ⋀*i*. *i* ∈ *I* ⟹ *polynomial-function* (λ*x*. *f x i*)⟧ ⟹ *polynomial-function* (λ*x*. *setsum* (*f x*) *I*)
by (*induct I rule*: *finite-induct*) *auto*

lemma *real-polynomial-function-minus* [*intro*]:
real-polynomial-function f ⟹ *real-polynomial-function* (λ*x*. − *f x*)
using *polynomial-function-minus* [*of f*]
by (*simp add*: *real-polynomial-function-eq*)

lemma *real-polynomial-function-diff* [*intro*]:
⟦*real-polynomial-function f*; *real-polynomial-function g*⟧ ⟹ *real-polynomial-function* (λ*x*. *f x* − *g x*)
using *polynomial-function-diff* [*of f*]
by (*simp add*: *real-polynomial-function-eq*)

lemma *real-polynomial-function-setsum* [*intro*]:
⟦*finite I*; ⋀*i*. *i* ∈ *I* ⟹ *real-polynomial-function* (λ*x*. *f x i*)⟧ ⟹ *real-polynomial-function* (λ*x*. *setsum* (*f x*) *I*)
using *polynomial-function-setsum* [*of I f*]
by (*simp add*: *real-polynomial-function-eq*)

lemma *real-polynomial-function-power* [*intro*]:
  real-polynomial-function f $\Longrightarrow$ *real-polynomial-function* ($\lambda x.$ *f x ^ n*)
 by (*induct n*) (*simp-all add*: *const mult*)

lemma *real-polynomial-function-compose* [*intro*]:
 assumes *f*: *polynomial-function f* **and** *g*: *real-polynomial-function g*
  shows *real-polynomial-function* (*g o f*)
 using *g*
 apply (*induction g rule*: *real-polynomial-function.induct*)
 using *f*
 apply (*simp-all add*: *polynomial-function-def o-def const add mult*)
 done

lemma *polynomial-function-compose* [*intro*]:
 assumes *f*: *polynomial-function f* **and** *g*: *polynomial-function g*
  shows *polynomial-function* (*g o f*)
 using *g real-polynomial-function-compose* [*OF f*]
 by (*auto simp*: *polynomial-function-def o-def*)

lemma *setsum-max-0*:
 fixes *x*::*real*
 shows ($\sum$ *i = 0..max m n. x^i * (if i* $\leq$ *m then a i else 0*)) = ($\sum$ *i = 0..m. x^i * a i*)
proof −
 have ($\sum$ *i = 0..max m n. x^i * (if i* $\leq$ *m then a i else 0*)) = ($\sum$ *i = 0..max m n. (if i* $\leq$ *m then x^i * a i else 0*))
  by (*auto simp*: *algebra-simps intro*: *setsum.cong*)
 also have ... = ($\sum$ *i = 0..m. (if i* $\leq$ *m then x^i * a i else 0*))
  by (*rule setsum.mono-neutral-right*) *auto*
 also have ... = ($\sum$ *i = 0..m. x^i * a i*)
  by (*auto simp*: *algebra-simps intro*: *setsum.cong*)
 finally show *?thesis* **.**
qed

lemma *real-polynomial-function-imp-setsum*:
 assumes *real-polynomial-function f*
  shows $\exists$ *a n*::*nat. f* = ($\lambda x.$ $\sum$ *i=0..n. a i * x ^ i*)
using *assms*
proof (*induct f*)
 case (*linear f*)
 then show *?case*
  apply (*clarsimp simp add*: *real-bounded-linear*)
  apply (*rule-tac x=*λ*i. if i=0 then 0 else c* **in** *exI*)
  apply (*rule-tac x=1* **in** *exI*)
  apply (*simp add*: *mult-ac*)
  done
next
 case (*const c*)
 show *?case*

apply (*rule-tac x=λi. c* **in** *exI*)
apply (*rule-tac x=0* **in** *exI*)
apply (*auto simp*: *mult-ac of-nat-Suc*)
done
case (*add f1 f2*)
then obtain *a1 n1 a2 n2* **where**
f1 = (λ*x*. ∑ *i* = *0..n1*. *a1 i* ∗ *x* ^ *i*) *f2* = (λ*x*. ∑ *i* = *0..n2*. *a2 i* ∗ *x* ^ *i*)
by *auto*
then show *?case*
apply (*rule-tac x=λi.* (*if i* ≤ *n1 then a1 i else 0*) + (*if i* ≤ *n2 then a2 i else 0*) **in** *exI*)
apply (*rule-tac x=max n1 n2* **in** *exI*)
using *setsum-max-0* [**where** *m=n1* **and** *n=n2*] *setsum-max-0* [**where** *m=n2* **and** *n=n1*]
apply (*simp add*: *setsum.distrib algebra-simps max.commute*)
done
case (*mult f1 f2*)
then obtain *a1 n1 a2 n2* **where**
f1 = (λ*x*. ∑ *i* = *0..n1*. *a1 i* ∗ *x* ^ *i*) *f2* = (λ*x*. ∑ *i* = *0..n2*. *a2 i* ∗ *x* ^ *i*)
by *auto*
then obtain *b1 b2* **where**
f1 = (λ*x*. ∑ *i* = *0..n1*. *b1 i* ∗ *x* ^ *i*) *f2* = (λ*x*. ∑ *i* = *0..n2*. *b2 i* ∗ *x* ^ *i*)
b1 = (λ*i*. *if i*≤*n1 then a1 i else 0*) *b2* = (λ*i*. *if i*≤*n2 then a2 i else 0*)
by *auto*
then show *?case*
apply (*rule-tac x=λi.* ∑ *k*≤*i*. *b1 k* ∗ *b2* (*i* − *k*) **in** *exI*)
apply (*rule-tac x=n1+n2* **in** *exI*)
using *polynomial-product* [*of n1 b1 n2 b2*]
apply (*simp add*: *Set-Interval.atLeast0AtMost*)
done
qed

lemma *real-polynomial-function-iff-setsum*:
real-polynomial-function f ⟷ (∃ *a n*::*nat*. *f* = (λ*x*. ∑ *i*=*0..n*. *a i* ∗ *x* ^ *i*))
apply (*rule iffI*)
apply (*erule real-polynomial-function-imp-setsum*)
apply (*auto simp*: *linear mult const real-polynomial-function-power real-polynomial-function-setsum*)
done

lemma *polynomial-function-iff-Basis-inner*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::*euclidean-space*
shows *polynomial-function f* ⟷ (∀ *b*∈*Basis*. *real-polynomial-function* (λ*x*. *inner* (*f x*) *b*))
(**is** *?lhs* = *?rhs*)
unfolding *polynomial-function-def*
proof (*intro iffI allI impI*)
assume ∀ *h*. *bounded-linear h* ⟶ *real-polynomial-function* (*h* ∘ *f*)
then show *?rhs*
by (*force simp add*: *bounded-linear-inner-left o-def*)

next
fix $h :: 'b \Rightarrow real$
assume *rp*: $\forall b{\in}Basis.$ *real-polynomial-function* $(\lambda x.\ f\ x \cdot b)$ **and** *h*: *bounded-linear h*
have *real-polynomial-function* $(h \circ (\lambda x.\ \sum b{\in}Basis.\ (f\ x \cdot b)\ *_R\ b))$
apply (*rule real-polynomial-function-compose* [*OF - linear* [*OF h*]])
using *rp*
apply (*auto simp*: *real-polynomial-function-eq polynomial-function-mult*)
done
then show *real-polynomial-function* $(h \circ f)$
by (*simp add*: *euclidean-representation-setsum-fun*)
qed

49.5 Stone-Weierstrass theorem for polynomial functions

First, we need to show that they are continous, differentiable and separable.

lemma *continuous-real-polymonial-function*:
assumes *real-polynomial-function f*
shows *continuous* (*at x*) *f*
using *assms*
by (*induct f*) (*auto simp*: *linear-continuous-at*)

lemma *continuous-polymonial-function*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::*euclidean-space*
assumes *polynomial-function f*
shows *continuous* (*at x*) *f*
apply (*rule euclidean-isCont*)
using *assms* **apply** (*simp add*: *polynomial-function-iff-Basis-inner*)
apply (*force dest*: *continuous-real-polymonial-function intro*: *isCont-scaleR*)
done

lemma *continuous-on-polymonial-function*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::*euclidean-space*
assumes *polynomial-function f*
shows *continuous-on s f*
using *continuous-polymonial-function* [*OF assms*] *continuous-at-imp-continuous-on*
by *blast*

lemma *has-real-derivative-polynomial-function*:
assumes *real-polynomial-function p*
shows $\exists p'.$ *real-polynomial-function* $p' \wedge$
$(\forall x.\ (p$ *has-real-derivative* $(p'\ x))\ (at\ x))$
using *assms*
proof (*induct p*)
case (*linear p*)
then show *?case*
by (*force simp*: *real-bounded-linear const intro*!: *derivative-eq-intros*)
next
case (*const c*)

```
  show ?case
    by (rule-tac x=λx. 0 in exI) auto
  case (add f1 f2)
  then obtain p1 p2 where
    real-polynomial-function p1 ⋀x. (f1 has-real-derivative p1 x) (at x)
    real-polynomial-function p2 ⋀x. (f2 has-real-derivative p2 x) (at x)
    by auto
  then show ?case
    apply (rule-tac x=λx. p1 x + p2 x in exI)
    apply (auto intro!: derivative-eq-intros)
    done
  case (mult f1 f2)
  then obtain p1 p2 where
    real-polynomial-function p1 ⋀x. (f1 has-real-derivative p1 x) (at x)
    real-polynomial-function p2 ⋀x. (f2 has-real-derivative p2 x) (at x)
    by auto
  then show ?case
    using mult
    apply (rule-tac x=λx. f1 x * p2 x + f2 x * p1 x in exI)
    apply (auto intro!: derivative-eq-intros)
    done
qed

lemma has-vector-derivative-polynomial-function:
  fixes p :: real ⇒ 'a::euclidean-space
  assumes polynomial-function p
    shows ∃p'. polynomial-function p' ∧
                (∀x. (p has-vector-derivative (p' x)) (at x))
proof −
  { fix b :: 'a
    assume b ∈ Basis
    then
     obtain p' where p': real-polynomial-function p' and pd: ⋀x. ((λx. p x • b)
has-real-derivative p' x) (at x)
      using assms [unfolded polynomial-function-iff-Basis-inner, rule-format] ‹b ∈
Basis›
      has-real-derivative-polynomial-function
      by blast
   have ∃q. polynomial-function q ∧ (∀x. ((λu. (p u • b) *R b) has-vector-derivative
q x) (at x))
      apply (rule-tac x=λx. p' x *R b in exI)
      using ‹b ∈ Basis› p'
      apply (simp add: polynomial-function-iff-Basis-inner inner-Basis)
      apply (auto intro: derivative-eq-intros pd)
      done
  }
  then obtain qf where qf:
      ⋀b. b ∈ Basis ⟹ polynomial-function (qf b)
      ⋀b x. b ∈ Basis ⟹ ((λu. (p u • b) *R b) has-vector-derivative qf b x) (at x)
```

```
    by metis
  show ?thesis
    apply (subst euclidean-representation-setsum-fun [of p, symmetric])
    apply (rule-tac x=λx. ∑ b∈Basis. qf b x in exI)
    apply (auto intro: has-vector-derivative-setsum qf)
    done
qed

lemma real-polynomial-function-separable:
  fixes x :: 'a::euclidean-space
  assumes x ≠ y shows ∃f. real-polynomial-function f ∧ f x ≠ f y
proof −
  have real-polynomial-function (λu. ∑ b∈Basis. (inner (x−u) b) ^2)
    apply (rule real-polynomial-function-setsum)
   apply (auto simp: algebra-simps real-polynomial-function-power real-polynomial-function-diff
               const linear bounded-linear-inner-left)
    done
  then show ?thesis
    apply (intro exI conjI, assumption)
    using assms
   apply (force simp add: euclidean-eq-iff [of x y] setsum-nonneg-eq-0-iff algebra-simps)
    done
qed

lemma Stone-Weierstrass-real-polynomial-function:
  fixes f :: 'a::euclidean-space ⇒ real
  assumes compact s continuous-on s f 0 < e
    shows ∃g. real-polynomial-function g ∧ (∀x ∈ s. |f x − g x| < e)
proof −
  interpret PR: function-ring-on Collect real-polynomial-function
    apply unfold-locales
    using assms continuous-on-polymonial-function real-polynomial-function-eq
    apply (auto intro: real-polynomial-function-separable)
    done
  show ?thesis
    using PR.Stone-Weierstrass-basic [OF ‹continuous-on s f› ‹0 < e›]
    by blast
qed

lemma Stone-Weierstrass-polynomial-function:
  fixes f :: 'a::euclidean-space ⇒ 'b::euclidean-space
  assumes s: compact s
      and f: continuous-on s f
      and e: 0 < e
    shows ∃g. polynomial-function g ∧ (∀x ∈ s. norm(f x − g x) < e)
proof −
  { fix b :: 'b
    assume b ∈ Basis
   have ∃p. real-polynomial-function p ∧ (∀x ∈ s. |f x • b − p x| < e / DIM('b))
```

apply (*rule exE* [*OF Stone-Weierstrass-real-polynomial-function* [*OF s -, of* $\lambda x.\ f\ x \cdot b\ e\ /\ card\ Basis$]])
using *e f*
apply (*auto simp*: *Euclidean-Space.DIM-positive intro*: *continuous-intros*)
done
}
then obtain *pf* **where** *pf*:
$\bigwedge b.\ b \in Basis \Longrightarrow$ *real-polynomial-function* (*pf b*) $\wedge$ ($\forall x \in s.\ |f\ x \cdot b - pf\ b\ x| < e\ /\ DIM('b)$)
apply (*rule bchoice* [*rule-format*, *THEN exE*])
apply *assumption*
apply (*force simp add*: *intro*: *that*)
done
have *polynomial-function* ($\lambda x.\ \sum b{\in}Basis.\ pf\ b\ x *_R b$)
using *pf*
by (*simp add*: *polynomial-function-setsum polynomial-function-mult real-polynomial-function-eq*)
moreover
{ fix *x*
assume $x \in s$
have *norm* ($\sum b{\in}Basis.\ (f\ x \cdot b) *_R b - pf\ b\ x *_R b$) $\leq$ ($\sum b{\in}Basis.$ *norm* ($(f\ x \cdot b) *_R b - pf\ b\ x *_R b$))
by (*rule norm-setsum*)
also have ... $<$ *of-nat* $DIM('b) * (e\ /\ DIM('b))$
apply (*rule setsum-bounded-above-strict*)
apply (*simp add*: *Real-Vector-Spaces.scaleR-diff-left* [*symmetric*] *pf* ‹$x \in s$›)
apply (*rule DIM-positive*)
done
also have ... = *e*
using *DIM-positive* **by** (*simp add*: *field-simps*)
finally have *norm* ($\sum b{\in}Basis.\ (f\ x \cdot b) *_R b - pf\ b\ x *_R b$) $< e$ **.**
}
ultimately
show *?thesis*
apply (*subst euclidean-representation-setsum-fun* [*of f*, *symmetric*])
apply (*rule-tac* $x{=}\lambda x.\ \sum b{\in}Basis.\ pf\ b\ x *_R b$ **in** *exI*)
apply (*auto simp*: *setsum-subtractf* [*symmetric*])
done
qed

49.6 Polynomial functions as paths

One application is to pick a smooth approximation to a path, or just pick a smooth path anyway in an open connected set

lemma *path-polynomial-function*:
fixes *g* :: *real* $\Rightarrow$ *'b*::*euclidean-space*
shows *polynomial-function g* $\Longrightarrow$ *path g*
by (*simp add*: *path-def continuous-on-polymonial-function*)

lemma *path-approx-polynomial-function*:

fixes $g :: real \Rightarrow 'b::euclidean\text{-}space$
assumes $path\ g\ 0 < e$
shows $\exists p.\ polynomial\text{-}function\ p \wedge$
$pathstart\ p = pathstart\ g \wedge$
$pathfinish\ p = pathfinish\ g \wedge$
$(\forall t \in \{0..1\}.\ norm(p\ t - g\ t) < e)$
proof −
obtain q **where** poq: $polynomial\text{-}function\ q$ **and** noq: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow norm\ (g\ x - q\ x) < e/4$
using $Stone\text{-}Weierstrass\text{-}polynomial\text{-}function\ [of\ \{0..1\}\ g\ e/4]\ assms$
by (*auto simp*: *path-def*)
have pf: $polynomial\text{-}function\ (\lambda t.\ q\ t + (g\ 0 - q\ 0) + t *_R (g\ 1 - q\ 1 - (g\ 0 - q\ 0)))$
by (*force simp add*: *poq*)
have $*$: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow norm\ (((q\ t - g\ t) + (g\ 0 - q\ 0)) + (t *_R (g\ 1 - q\ 1) + t *_R (q\ 0 - g\ 0))) < (e/4 + e/4) + (e/4+e/4)$
apply (*intro Real-Vector-Spaces.norm-add-less*)
using *noq*
apply (*auto simp*: *norm-minus-commute intro*: *le-less-trans* [*OF mult-left-le-one-le noq*] *simp del*: *less-divide-eq-numeral1*)
done
show *?thesis*
apply (*intro exI conjI*)
apply (*rule pf*)
using $*$
apply (*auto simp add*: *pathstart-def pathfinish-def algebra-simps*)
done
qed

lemma *connected-open-polynomial-connected*:
fixes $s :: 'a::euclidean\text{-}space\ set$
assumes s: *open s connected s*
and $x \in s\ y \in s$
shows $\exists g.\ polynomial\text{-}function\ g \wedge path\text{-}image\ g \subseteq s \wedge$
$pathstart\ g = x \wedge pathfinish\ g = y$
proof −
have *path-connected s* **using** *assms*
by (*simp add*: *connected-open-path-connected*)
with ‹$x \in s$› ‹$y \in s$› **obtain** p **where** p: *path p path-image p* $\subseteq$ *s pathstart p* $=$ *x pathfinish p* $=$ *y*
by (*force simp*: *path-connected-def*)
have $\exists e.\ 0 < e \wedge (\forall x \in path\text{-}image\ p.\ ball\ x\ e \subseteq s)$
proof (*cases s* $=$ *UNIV*)
case *True* **then show** *?thesis*
by (*simp add*: *gt-ex*)
next
case *False*
then have − $s \neq \{\}$ **by** *blast*
then show *?thesis*

apply (*rule-tac x=setdist* (*path-image p*) (−*s*) **in** *exI*)
using *s p*
apply (*simp add*: *setdist-gt-0-compact-closed compact-path-image open-closed*)
using *setdist-le-dist* [*of - path-image p - −s*]
by *fastforce*
qed
then obtain *e* **where** *0 < e***and** *eb*: ⋀*x*. *x* ∈ *path-image p* ⟹ *ball x e* ⊆ *s*
by *auto*
show *?thesis*
using *path-approx-polynomial-function* [*OF* ‹*path p*› ‹*0 < e*›]
apply *clarify*
apply (*intro exI conjI*, *assumption*)
using *p*
apply (*fastforce simp add*: *dist-norm path-image-def norm-minus-commute intro*: *eb* [*THEN subsetD*])+
done
qed

hide-fact *linear add mult const*

end

50 Radius of Convergence and Summation Tests

theory *Summation*
imports
Complex-Main
~~/*src*/*HOL*/*Library*/*Extended-Real*
~~/*src*/*HOL*/*Library*/*Liminf-Limsup*
begin

The definition of the radius of convergence of a power series, various summability tests, lemmas to compute the radius of convergence etc.

50.1 Rounded dual logarithm

definition *natlog2 n* = (*if n* = *0 then 0 else nat* ⌊*log 2* (*real-of-nat n*)⌋)

lemma *natlog2-0* [*simp*]: *natlog2 0* = *0* **by** (*simp add*: *natlog2-def*)
lemma *natlog2-1* [*simp*]: *natlog2 1* = *0* **by** (*simp add*: *natlog2-def*)
lemma *natlog2-eq-0-iff*: *natlog2 n* = *0* ⟷ *n* < *2* **by** (*simp add*: *natlog2-def*)

lemma *natlog2-power-of-two* [*simp*]: *natlog2* (*2* ^ *n*) = *n*
by (*simp add*: *natlog2-def log-nat-power*)

lemma *natlog2-mono*: *m* ≤ *n* ⟹ *natlog2 m* ≤ *natlog2 n*
unfolding *natlog2-def* **by** (*simp-all add*: *nat-mono floor-mono*)

lemma *pow-natlog2-le*: *n* > *0* ⟹ *2* ^ *natlog2 n* ≤ *n*

proof −
assume *n*: *n* > *0*
from *n* **have** *of-nat* (*2* ^ *natlog2 n*) = *2 powr real-of-nat* (*nat* ⌊*log 2* (*real-of-nat n*)⌋)
by (*subst powr-realpow*) (*simp-all add*: *natlog2-def*)
also have ... = *2 powr of-int* ⌊*log 2* (*real-of-nat n*)⌋ **using** *n* **by** *simp*
also have ... ≤ *2 powr log 2* (*real-of-nat n*) **by** (*intro powr-mono*) (*linarith*, *simp-all*)
also have ... = *of-nat n* **using** *n* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *pow-natlog2-gt*: *n* > *0* ⟹ *2* ∗ *2* ^ *natlog2 n* > *n*
and *pow-natlog2-ge*: *n* > *0* ⟹ *2* ∗ *2* ^ *natlog2 n* ≥ *n*
proof −
assume *n*: *n* > *0*
from *n* **have** *of-nat n* = *2 powr log 2* (*real-of-nat n*) **by** *simp*
also have ... < *2 powr* (*1* + *of-int* ⌊*log 2* (*real-of-nat n*)⌋)
by (*intro powr-less-mono*) (*linarith*, *simp-all*)
also from *n* **have** ... = *2 powr* (*1* + *real-of-nat* (*nat* ⌊*log 2* (*real-of-nat n*)⌋)) **by** *simp*
also from *n* **have** ... = *of-nat* (*2* ∗ *2* ^ *natlog2 n*)
by (*simp-all add*: *natlog2-def powr-real-of-int powr-add*)
finally show *2* ∗ *2* ^ *natlog2 n* > *n* **by** (*rule of-nat-less-imp-less*)
thus *2* ∗ *2* ^ *natlog2 n* ≥ *n* **by** *simp*
qed

lemma *natlog2-eqI*:
assumes *n* > *0* *2*^*k* ≤ *n n* < *2* ∗ *2*^*k*
shows *natlog2 n* = *k*
proof −
from *assms* **have** *of-nat* (*2* ^ *k*) ≤ *real-of-nat n* **by** (*subst of-nat-le-iff*) *simp-all*
hence *real-of-int* (*int k*) ≤ *log* (*of-nat 2*) (*real-of-nat n*)
by (*subst le-log-iff*) (*simp-all add*: *powr-realpow assms del*: *of-nat-le-iff*)
moreover from *assms* **have** *real-of-nat n* < *of-nat* (*2* ^ *Suc k*) **by** (*subst of-nat-less-iff*) *simp-all*
hence *log 2* (*real-of-nat n*) < *of-nat k* + *1*
by (*subst log-less-iff*) (*simp-all add*: *assms powr-realpow powr-add*)
ultimately have ⌊*log 2* (*real-of-nat n*)⌋ = *of-nat k* **by** (*intro floor-unique*) *simp-all*
with *assms* **show** *?thesis* **by** (*simp add*: *natlog2-def*)
qed

lemma *natlog2-rec*:
assumes *n* ≥ *2*
shows *natlog2 n* = *1* + *natlog2* (*n div 2*)
proof (*rule natlog2-eqI*)
from *assms* **have** *2* ^ (*1* + *natlog2* (*n div 2*)) ≤ *2* ∗ (*n div 2*)
by (*simp add*: *pow-natlog2-le*)

also have $\ldots \leq n$ **by** *simp*
finally show $2 \hat{} (1 + natlog2\ (n\ div\ 2)) \leq n$ **.**
next
from *assms* **have** $n < 2 * (n\ div\ 2 + 1)$ **by** *simp*
also from *assms* **have** $(n\ div\ 2) < 2 \hat{} (1 + natlog2\ (n\ div\ 2))$
by (*simp add*: *pow-natlog2-gt*)
hence $2 * (n\ div\ 2 + 1) \leq 2 * (2 \hat{} (1 + natlog2\ (n\ div\ 2)))$
by (*intro mult-left-mono*) *simp-all*
finally show $n < 2 * 2 \hat{} (1 + natlog2\ (n\ div\ 2))$ **.**
qed (*insert assms*, *simp-all*)

fun *natlog2-aux* **where**
natlog2-aux n acc = (*if* (*n*::*nat*) < *2 then acc else natlog2-aux* (*n div 2*) (*acc* + *1*))

lemma *natlog2-aux-correct*:
natlog2-aux n acc = *acc* + *natlog2 n*
by (*induction n acc rule*: *natlog2-aux.induct*) (*auto simp*: *natlog2-rec natlog2-eq-0-iff*)

lemma *natlog2-code* [*code*]: *natlog2 n* = *natlog2-aux n 0*
by (*subst natlog2-aux-correct*) *simp*

50.2 Convergence tests for infinite sums

50.2.1 Root test

lemma *limsup-root-powser*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real-normed-div-algebra*}
shows *limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n* ∗ *z* ^ *n*)))) =
limsup (λ*n*. *ereal* (*root n* (*norm* (*f n*)))) ∗ *ereal* (*norm z*)
proof −
have *A*: (λ*n*. *ereal* (*root n* (*norm* (*f n* ∗ *z* ^ *n*)))) =
(λ*n*. *ereal* (*root n* (*norm* (*f n*))) ∗ *ereal* (*norm z*)) (**is** *?g* = *?h*)
proof
fix *n* **show** *?g n* = *?h n*
by (*cases n* = *0*) (*simp-all add*: *norm-mult real-root-mult real-root-pos2 norm-power*)
qed
show *?thesis* **by** (*subst A*, *subst limsup-ereal-mult-right*) *simp-all*
qed

lemma *limsup-root-limit*:
assumes (λ*n*. *ereal* (*root n* (*norm* (*f n*)))) ⟶ *l* (**is** *?g* ⟶ -)
shows *limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n*)))) = *l*
proof −
from *assms* **have** *convergent ?g lim ?g* = *l*
unfolding *convergent-def* **by** (*blast intro*: *limI*)+
with *convergent-limsup-cl* **show** *?thesis* **by** *force*
qed

lemma *limsup-root-limit'*:

```
  assumes (λn. root n (norm (f n))) ⟶ l
  shows   limsup (λn. ereal (root n (norm (f n)))) = ereal l
  by (intro limsup-root-limit tendsto-ereal assms)

lemma root-test-convergence':
  fixes f :: nat ⇒ 'a :: banach
  defines l ≡ limsup (λn. ereal (root n (norm (f n))))
  assumes l: l < 1
  shows   summable f
proof −
  have 0 = limsup (λn. 0) by (simp add: Limsup-const)
   also have ... ≤ l unfolding l-def by (intro Limsup-mono) (simp-all add:
real-root-ge-zero)
  finally have l ≥ 0 by simp
  with l obtain l' where l': l = ereal l' by (cases l) simp-all

  def c ≡ (1 − l') / 2
  from l and ‹l ≥ 0› have c: l + c > l l' + c ≥ 0 l' + c < 1 unfolding c-def
    by (simp-all add: field-simps l')
  have ∀ C>l. eventually (λn. ereal (root n (norm (f n))) < C) sequentially
    by (subst Limsup-le-iff[symmetric]) (simp add: l-def)
 with c have eventually (λn. ereal (root n (norm (f n))) < l + ereal c) sequentially
by simp
  with eventually-gt-at-top[of 0::nat]
    have eventually (λn. norm (f n) ≤ (l' + c) ^ n) sequentially
  proof eventually-elim
    fix n :: nat assume n: n > 0
    assume ereal (root n (norm (f n))) < l + ereal c
    hence root n (norm (f n)) ≤ l' + c by (simp add: l')
    with c n have root n (norm (f n)) ^ n ≤ (l' + c) ^ n
      by (intro power-mono) (simp-all add: real-root-ge-zero)
    also from n have root n (norm (f n)) ^ n = norm (f n) by simp
    finally show norm (f n) ≤ (l' + c) ^ n by simp
  qed
  thus ?thesis
    by (rule summable-comparison-test-ev[OF - summable-geometric]) (simp add:
c)
qed

lemma root-test-divergence:
  fixes f :: nat ⇒ 'a :: banach
  defines l ≡ limsup (λn. ereal (root n (norm (f n))))
  assumes l: l > 1
  shows   ¬summable f
proof
  assume summable f
  hence bounded: Bseq f by (simp add: summable-imp-Bseq)

  have 0 = limsup (λn. 0) by (simp add: Limsup-const)
```

also have ... $\leq l$ **unfolding** *l-def* **by** (*intro Limsup-mono*) (*simp-all add*: *real-root-ge-zero*)
finally have *l-nonneg*: $l \geq 0$ **by** *simp*

def $c \equiv$ *if* $l = \infty$ *then 2 else 1 +* (*real-of-ereal* $l - 1$) / 2
from l *l-nonneg* **consider** $l = \infty \mid \exists l'.\ l =$ *ereal* l' **by** (*cases l*) *simp-all*
hence *c*: $c > 1 \wedge$ *ereal* $c < l$ **by** *cases* (*insert l*, *auto simp*: *c-def field-simps*)

have *unbounded*: $\neg$*bdd-above* $\{n.$ *root n* (*norm* (*f n*)) $> c\}$
proof
assume *bdd-above* $\{n.$ *root n* (*norm* (*f n*)) $> c\}$
then obtain *N* **where** $\forall n.$ *root n* (*norm* (*f n*)) $> c \longrightarrow n \leq N$ **unfolding** *bdd-above-def* **by** *blast*
hence $\exists N.\ \forall n{\geq}N.$ *root n* (*norm* (*f n*)) $\leq c$
by (*intro exI*[*of - N + 1*]) (*force simp*: *not-less-eq-eq*[*symmetric*])
hence *eventually* ($\lambda n.$ *root n* (*norm* (*f n*)) $\leq c$) *sequentially*
by (*auto simp*: *eventually-at-top-linorder*)
hence $l \leq c$ **unfolding** *l-def* **by** (*intro Limsup-bounded*) *simp-all*
with *c* **show** *False* **by** *auto*
qed

from *bounded* **obtain** *K* **where** *K*: $K > 0$ $\bigwedge n.$ *norm* (*f n*) $\leq K$ **using** *BseqE* **by** *blast*
def $n \equiv$ *nat* $\lceil$*log c K*$\rceil$
from *unbounded* **have** $\exists m{>}n.\ c <$ *root m* (*norm* (*f m*)) **unfolding** *bdd-above-def*
by (*auto simp*: *not-le*)
then guess *m* **by** (*elim exE conjE*) **note** $m =$ *this*
from *c K* **have** $K = c$ *powr log c K* **by** (*simp add*: *powr-def log-def*)
also from *c* **have** *c powr log c K* $\leq$ *c powr real n* **unfolding** *n-def*
by (*intro powr-mono*, *linarith*, *simp*)
finally have $K \leq c \hat{}\ n$ **using** *c* **by** (*simp add*: *powr-realpow*)
also from *c m* **have** $c \hat{}\ n < c \hat{}\ m$ **by** *simp*
also from *c m* **have** $c \hat{}\ m <$ *root m* (*norm* (*f m*)) $\hat{}\ m$ **by** (*intro power-strict-mono*) *simp-all*
also from *m* **have** ... $=$ *norm* (*f m*) **by** *simp*
finally show *False* **using** *K*(*2*)[*of m*] **by** *simp*
qed

50.2.2 Cauchy's condensation test

context
fixes *f* :: *nat* $\Rightarrow$ *real*
begin

private lemma *condensation-inequality*:
assumes *mono*: $\bigwedge m\ n.\ 0 < m \Longrightarrow m \leq n \Longrightarrow f\ n \leq f\ m$
shows ($\sum k{=}1..{<}n.\ f\ k$) $\geq$ ($\sum k{=}1..{<}n.\ f$ (*2* $*$ *2* $\hat{}$ *natlog2 k*)) (**is** *?thesis1*)
($\sum k{=}1..{<}n.\ f\ k$) $\leq$ ($\sum k{=}1..{<}n.\ f$ (*2* $\hat{}$ *natlog2 k*)) (**is** *?thesis2*)
by (*intro setsum-mono mono pow-natlog2-ge pow-natlog2-le*, *simp*, *simp*)+

private lemma *condensation-condense1*: $(\sum k{=}1..{<}2\hat{}n.\ f\ (2\ \hat{}\ natlog2\ k)) = (\sum k{<}n.\ 2\hat{}k * f\ (2\ \hat{}\ k))$
proof (*induction n*)
case (*Suc n*)
have $\{1..{<}2\hat{}Suc\ n\} = \{1..{<}2\hat{}n\} \cup \{2\hat{}n..{<}(2\hat{}Suc\ n :: nat)\}$ **by** *auto*
also have $(\sum k{\in}\ldots.\ f\ (2\ \hat{}\ natlog2\ k)) =$
$(\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)) + (\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}natlog2\ k))$
by (*subst setsum.union-disjoint*) (*insert Suc*, *auto*)
also have *natlog2 k* $= n$ **if** $k \in \{2\hat{}n..{<}2\hat{}Suc\ n\}$ **for** *k* **using** *that* **by** (*intro natlog2-eqI*) *simp-all*
hence $(\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}natlog2\ k)) = (\sum (\text{-}::nat) = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}n))$
by (*intro setsum.cong*) *simp-all*
also have $\ldots = 2\hat{}n * f\ (2\hat{}n)$ **by** (*simp add*: *of-nat-power*)
finally show *?case* **by** *simp*
qed *simp*

private lemma *condensation-condense2*: $(\sum k{=}1..{<}2\hat{}n.\ f\ (2 * 2\ \hat{}\ natlog2\ k)) = (\sum k{<}n.\ 2\hat{}k * f\ (2\ \hat{}\ Suc\ k))$
proof (*induction n*)
case (*Suc n*)
have $\{1..{<}2\hat{}Suc\ n\} = \{1..{<}2\hat{}n\} \cup \{2\hat{}n..{<}(2\hat{}Suc\ n :: nat)\}$ **by** *auto*
also have $(\sum k{\in}\ldots.\ f\ (2 * 2\ \hat{}\ natlog2\ k)) =$
$(\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)) + (\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2 * 2\hat{}natlog2\ k))$
by (*subst setsum.union-disjoint*) (*insert Suc*, *auto*)
also have *natlog2 k* $= n$ **if** $k \in \{2\hat{}n..{<}2\hat{}Suc\ n\}$ **for** *k* **using** *that* **by** (*intro natlog2-eqI*) *simp-all*
hence $(\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2{*}2\hat{}natlog2\ k)) = (\sum (\text{-}::nat) = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}Suc\ n))$
by (*intro setsum.cong*) *simp-all*
also have $\ldots = 2\hat{}n * f\ (2\hat{}Suc\ n)$ **by** (*simp add*: *of-nat-power*)
finally show *?case* **by** *simp*
qed *simp*

lemma *condensation-test*:
assumes *mono*: $\bigwedge m.\ 0 < m \Longrightarrow f\ (Suc\ m) \leq f\ m$
assumes *nonneg*: $\bigwedge n.\ f\ n \geq 0$
shows *summable f* $\longleftrightarrow$ *summable* $(\lambda n.\ 2\hat{}n * f\ (2\hat{}n))$
proof −
def $f' \equiv \lambda n.\ if\ n = 0\ then\ 0\ else\ f\ n$
from *mono* **have** *mono'*: *decseq* $(\lambda n.\ f\ (Suc\ n))$ **by** (*intro decseq-SucI*) *simp*
hence *mono'*: $f\ n \leq f\ m$ **if** $m \leq n\ m > 0$ **for** *m n*
using *that decseqD*[*OF mono'*, *of m − 1 n − 1*] **by** *simp*

have $(\lambda n.\ f\ (Suc\ n)) = (\lambda n.\ f'\ (Suc\ n))$ **by** (*intro ext*) (*simp add*: *f'-def*)
hence *summable f* $\longleftrightarrow$ *summable f'*
by (*subst* (*1 2*) *summable-Suc-iff* [*symmetric*]) (*simp only*:)

also have ... $\longleftrightarrow$ *convergent* ($\lambda n.\ \sum k{<}n.\ f'\ k$) **unfolding** *summable-iff-convergent*
..
also have *monoseq* ($\lambda n.\ \sum k{<}n.\ f'\ k$) **unfolding** *f'-def*
by (*intro mono-SucI1*) (*auto intro*!: *mult-nonneg-nonneg nonneg*)
hence *convergent* ($\lambda n.\ \sum k{<}n.\ f'\ k$) $\longleftrightarrow$ *Bseq* ($\lambda n.\ \sum k{<}n.\ f'\ k$)
by (*rule monoseq-imp-convergent-iff-Bseq*)
also have ... $\longleftrightarrow$ *Bseq* ($\lambda n.\ \sum k{=}1..{<}n.\ f'\ k$) **unfolding** *One-nat-def*
by (*subst setsum-shift-lb-Suc0-0-upt*) (*simp-all add*: *f'-def atLeast0LessThan*)
also have ... $\longleftrightarrow$ *Bseq* ($\lambda n.\ \sum k{=}1..{<}n.\ f\ k$) **unfolding** *f'-def* **by** *simp*
also have ... $\longleftrightarrow$ *Bseq* ($\lambda n.\ \sum k{=}1..{<}2\hat{}n.\ f\ k$)
by (*rule nonneg-incseq-Bseq-subseq-iff*[*symmetric*])
(*auto intro*!: *setsum-nonneg incseq-SucI nonneg simp*: *subseq-def*)
also have ... $\longleftrightarrow$ *Bseq* ($\lambda n.\ \sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$)
proof (*intro iffI*)
assume *A*: *Bseq* ($\lambda n.\ \sum k{=}1..{<}2\hat{}n.\ f\ k$)
have *eventually* ($\lambda n.\ norm\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)) \leq norm\ (\sum k{=}1..{<}2\hat{}n.\ f\ k)$) *sequentially*
proof (*intro always-eventually allI*)
fix *n* :: *nat*
have *norm* ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)$) = ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)$) **unfolding** *real-norm-def*
by (*intro abs-of-nonneg setsum-nonneg ballI mult-nonneg-nonneg nonneg*) *simp-all*
also have ... $\leq$ ($\sum k{=}1..{<}2\hat{}n.\ f\ k$)
by (*subst condensation-condense2* [*symmetric*]) (*intro condensation-inequality mono'*)
also have ... = *norm* ... **unfolding** *real-norm-def*
by (*intro abs-of-nonneg*[*symmetric*] *setsum-nonneg ballI mult-nonneg-nonneg nonneg*)
finally show *norm* ($\sum k{<}n.\ 2\ \hat{}\ k * f\ (2\ \hat{}\ Suc\ k)$) $\leq$ *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) **.**
qed
from *this* **and** *A* **have** *Bseq* ($\lambda n.\ \sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)$) **by** (*rule Bseq-eventually-mono*)
from *Bseq-mult*[*OF Bfun-const*[*of 2*] *this*] **have** *Bseq* ($\lambda n.\ \sum k{<}n.\ 2\hat{}Suc\ k * f\ (2\hat{}Suc\ k)$)
by (*simp add*: *setsum-right-distrib setsum-left-distrib mult-ac*)
hence *Bseq* ($\lambda n.\ (\sum k{=}Suc\ 0..{<}Suc\ n.\ 2\hat{}k * f\ (2\hat{}k)) + f\ 1$)
by (*intro Bseq-add*, *subst setsum-shift-bounds-Suc-ivl*) (*simp add*: *atLeast0LessThan*)
hence *Bseq* ($\lambda n.\ (\sum k{=}0..{<}Suc\ n.\ 2\hat{}k * f\ (2\hat{}k))$)
by (*subst setsum-head-upt-Suc*) (*simp-all add*: *add-ac*)
thus *Bseq* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$)
by (*subst* (*asm*) *Bseq-Suc-iff*) (*simp add*: *atLeast0LessThan*)
next
assume *A*: *Bseq* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$)
have *eventually* ($\lambda n.\ norm\ (\sum k{=}1..{<}2\hat{}n.\ f\ k) \leq norm\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$) *sequentially*
proof (*intro always-eventually allI*)
fix *n* :: *nat*

have *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) = ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) **unfolding** *real-norm-def*
by (*intro abs-of-nonneg setsum-nonneg ballI mult-nonneg-nonneg nonneg*)
also have ... ≤ ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$)
by (*subst condensation-condense1* [*symmetric*]) (*intro condensation-inequality mono'*)
also have ... = *norm* ... **unfolding** *real-norm-def*
by (*intro abs-of-nonneg* [*symmetric*] *setsum-nonneg ballI mult-nonneg-nonneg nonneg*) *simp-all*
finally show *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) ≤ *norm* ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$) **.**
qed
from *this* **and** *A* **show** *Bseq* ($\lambda n.\ \sum k{=}1..{<}2\hat{}n.\ f\ k$) **by** (*rule Bseq-eventually-mono*)
qed
also have *monoseq* ($\lambda n.$ ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$))
by (*intro mono-SucI1*) (*auto intro*!: *mult-nonneg-nonneg nonneg*)
hence *Bseq* ($\lambda n.$ ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$)) ⟷ *convergent* ($\lambda n.$ ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$))
by (*rule monoseq-imp-convergent-iff-Bseq* [*symmetric*])
also have ... ⟷ *summable* ($\lambda k.\ 2\hat{}k * f\ (2\hat{}k)$) **by** (*simp only*: *summable-iff-convergent*)
finally show *?thesis* **.**
qed

end

50.2.3 Summability of powers

lemma *abs-summable-complex-powr-iff*:
summable ($\lambda n.$ *norm* (*exp* (*of-real* (*ln* (*of-nat n*)) * *s*))) ⟷ *Re s* < −*1*
proof (*cases Re s* ≤ *0*)
let *?l* = $\lambda n.$ *complex-of-real* (*ln* (*of-nat n*))
case *False*
with *eventually-gt-at-top*[*of 0*::*nat*]
have *eventually* ($\lambda n.$ *norm* (*1* :: *real*) ≤ *norm* (*exp* (*?l n* * *s*))) *sequentially*
by (*auto intro*!: *ge-one-powr-ge-zero elim*!: *eventually-mono*)
from *summable-comparison-test-ev*[*OF this*] *False* **show** *?thesis* **by** (*auto simp*: *summable-const-iff*)
next
let *?l* = $\lambda n.$ *complex-of-real* (*ln* (*of-nat n*))
case *True*
hence *summable* ($\lambda n.$ *norm* (*exp* (*?l n* * *s*))) ⟷ *summable* ($\lambda n.$ *2*^*n* * *norm* (*exp* (*?l* (*2*^*n*) * *s*)))
by (*intro condensation-test*) (*auto intro*!: *mult-right-mono-neg*)
also have ($\lambda n.$ *2*^*n* * *norm* (*exp* (*?l* (*2*^*n*) * *s*))) = ($\lambda n.$ (*2 powr* (*Re s* + *1*)) ^ *n*)
proof
fix *n* :: *nat*
have *2*^*n* * *norm* (*exp* (*?l* (*2*^*n*) * *s*)) = *exp* (*real n* * *ln 2*) * *exp* (*real n* * *ln 2* * *Re s*)
using *True* **by** (*subst exp-of-nat-mult*) (*simp add*: *ln-realpow algebra-simps*)
also have ... = *exp* (*real n* * (*ln 2* * (*Re s* + *1*)))

by (*simp add*: *algebra-simps exp-add*)
also have ... = *exp* (*ln 2* ∗ (*Re s* + *1*)) ^ *n* **by** (*subst exp-of-nat-mult*) *simp*
also have *exp* (*ln 2* ∗ (*Re s* + *1*)) = *2 powr* (*Re s* + *1*) **by** (*simp add*: *powr-def*)
finally show *2^n* ∗ *norm* (*exp* (*?l* (*2^n*) ∗ *s*)) = (*2 powr* (*Re s* + *1*)) ^ *n* **.**
qed
also have *summable* ... ⟷ *2 powr* (*Re s* + *1*) < *2 powr 0*
by (*subst summable-geometric-iff*) *simp*
also have ... ⟷ *Re s* < −*1* **by** (*subst powr-less-cancel-iff*) (*simp*, *linarith*)
finally show *?thesis* **.**
qed

lemma *summable-complex-powr-iff*:
assumes *Re s* < −*1*
shows *summable* (λ*n*. *exp* (*of-real* (*ln* (*of-nat n*)) ∗ *s*))
by (*rule summable-norm-cancel*, *subst abs-summable-complex-powr-iff*) *fact*

lemma *summable-real-powr-iff*: *summable* (λ*n*. *of-nat n powr s* :: *real*) ⟷ *s* < −*1*
proof −
from *eventually-gt-at-top*[*of 0*::*nat*]
have *summable* (λ*n*. *of-nat n powr s*) ⟷ *summable* (λ*n*. *exp* (*ln* (*of-nat n*) ∗ *s*))
by (*intro summable-cong*) (*auto elim*!: *eventually-mono simp*: *powr-def*)
also have ... ⟷ *s* < −*1* **using** *abs-summable-complex-powr-iff*[*of of-real s*]
by *simp*
finally show *?thesis* **.**
qed

lemma *inverse-power-summable*:
assumes *s*: *s* ≥ *2*
shows *summable* (λ*n*. *inverse* (*of-nat n* ^ *s* :: *'a* :: {*real-normed-div-algebra*,*banach*}))
proof (*rule summable-norm-cancel*, *subst summable-cong*)
from *eventually-gt-at-top*[*of 0*::*nat*]
show *eventually* (λ*n*. *norm* (*inverse* (*of-nat n* ^ *s*:: *'a*)) = *real-of-nat n powr* (−*real s*)) *at-top*
by *eventually-elim* (*simp add*: *norm-inverse norm-power powr-minus powr-realpow*)
qed (*insert s summable-real-powr-iff*[*of* −*s*], *simp-all*)

lemma *not-summable-harmonic*: ¬*summable* (λ*n*. *inverse* (*of-nat n*) :: *'a* :: *real-normed-field*)
proof
assume *summable* (λ*n*. *inverse* (*of-nat n*) :: *'a*)
hence *convergent* (λ*n*. *norm* (*of-real* (∑ *k*<*n*. *inverse* (*of-nat k*)) :: *'a*))
by (*simp add*: *summable-iff-convergent convergent-norm*)
hence *convergent* (λ*n*. *abs* (∑ *k*<*n*. *inverse* (*of-nat k*)) :: *real*) **by** (*simp only*: *norm-of-real*)
also have (λ*n*. *abs* (∑ *k*<*n*. *inverse* (*of-nat k*)) :: *real*) = (λ*n*. ∑ *k*<*n*. *inverse* (*of-nat k*))
by (*intro ext abs-of-nonneg setsum-nonneg*) *auto*

also have *convergent ...* $\longleftrightarrow$ *summable* ($\lambda k.$ *inverse* (*of-nat k*) :: *real*)
by (*simp add*: *summable-iff-convergent*)
finally show *False* **using** *summable-real-powr-iff*[*of* -1] **by** (*simp add*: *powr-minus*)
qed

50.2.4 Kummer's test

lemma *kummers-test-convergence*:
fixes *f p* :: *nat* $\Rightarrow$ *real*
assumes *pos-f*: *eventually* ($\lambda n.$ *f n* $>$ *0*) *sequentially*
assumes *nonneg-p*: *eventually* ($\lambda n.$ *p n* $\geq$ *0*) *sequentially*
defines *l* $\equiv$ *liminf* ($\lambda n.$ *ereal* (*p n* $*$ *f n* / *f* (*Suc n*) $-$ *p* (*Suc n*)))
assumes *l*: *l* $>$ *0*
shows *summable f*
unfolding *summable-iff-convergent'*
proof $-$
def *r* $\equiv$ (*if l* $=$ ∞ *then 1 else real-of-ereal l* / *2*)
from *l* **have** *r* $>$ *0* $\wedge$ *of-real r* $<$ *l* **by** (*cases l*) (*simp-all add*: *r-def*)
hence *r*: *r* $>$ *0 of-real r* $<$ *l* **by** *simp-all*
hence *eventually* ($\lambda n.$ *p n* $*$ *f n* / *f* (*Suc n*) $-$ *p* (*Suc n*) $>$ *r*) *sequentially*
unfolding *l-def* **by** (*force dest*: *less-LiminfD*)
moreover from *pos-f* **have** *eventually* ($\lambda n.$ *f* (*Suc n*) $>$ *0*) *sequentially*
by (*subst eventually-sequentially-Suc*)
ultimately have *eventually* ($\lambda n.$ *p n* $*$ *f n* $-$ *p* (*Suc n*) $*$ *f* (*Suc n*) $>$ *r* $*$ *f* (*Suc n*)) *sequentially*
by *eventually-elim* (*simp add*: *field-simps*)
from *eventually-conj*[*OF pos-f eventually-conj*[*OF nonneg-p this*]]
obtain *m* **where** *m*: $\bigwedge n.$ *n* $\geq$ *m* $\Longrightarrow$ *f n* $>$ *0* $\bigwedge n.$ *n* $\geq$ *m* $\Longrightarrow$ *p n* $\geq$ *0*
$\bigwedge n.$ *n* $\geq$ *m* $\Longrightarrow$ *p n* $*$ *f n* $-$ *p* (*Suc n*) $*$ *f* (*Suc n*) $>$ *r* $*$ *f* (*Suc n*)
unfolding *eventually-at-top-linorder* **by** *blast*

let *?c* = (*norm* ($\sum k{\leq}m.$ *r* $*$ *f k*) + *p m* $*$ *f m*) / *r*
have *Bseq* ($\lambda n.$ ($\sum k{\leq}n$ + *Suc m.* *f k*))
proof (*rule BseqI'*)
fix *k* :: *nat*
def *n* $\equiv$ *k* + *Suc m*
have *n*: *n* $>$ *m* **by** (*simp add*: *n-def*)

from *r* **have** *r* $*$ *norm* ($\sum k{\leq}n.$ *f k*) = *norm* ($\sum k{\leq}n.$ *r* $*$ *f k*)
by (*simp add*: *setsum-right-distrib*[*symmetric*] *abs-mult*)
also from *n* **have** {*..n*} = {*..m*} $\cup$ {*Suc m..n*} **by** *auto*
hence ($\sum k{\leq}n.$ *r* $*$ *f k*) = ($\sum k{\in}${*..m*} $\cup$ {*Suc m..n*}*.* *r* $*$ *f k*) **by** (*simp only*:)
also have ... = ($\sum k{\leq}m.$ *r* $*$ *f k*) + ($\sum k{=}$*Suc m..n.* *r* $*$ *f k*)
by (*subst setsum.union-disjoint*) *auto*
also have *norm* ... $\leq$ *norm* ($\sum k{\leq}m.$ *r* $*$ *f k*) + *norm* ($\sum k{=}$*Suc m..n.* *r* $*$ *f k*)
by (*rule norm-triangle-ineq*)
also from *r less-imp-le*[*OF m*(*1*)] **have** ($\sum k{=}$*Suc m..n.* *r* $*$ *f k*) $\geq$ *0*
by (*intro setsum-nonneg*) *auto*

hence *norm* $(\sum k{=}Suc\ m..n.\ r * f\ k) = (\sum k{=}Suc\ m..n.\ r * f\ k)$ **by** *simp*
also have $(\sum k{=}Suc\ m..n.\ r * f\ k) = (\sum k{=}m..{<}n.\ r * f\ (Suc\ k))$
by (*subst setsum-shift-bounds-Suc-ivl* [*symmetric*])
(*simp only*: *atLeastLessThanSuc-atLeastAtMost*)
also from *m* **have** $\ldots \leq (\sum k{=}m..{<}n.\ p\ k * f\ k - p\ (Suc\ k) * f\ (Suc\ k))$
by (*intro setsum-mono*[*OF less-imp-le*]) *simp-all*
also have $\ldots = -(\sum k{=}m..{<}n.\ p\ (Suc\ k) * f\ (Suc\ k) - p\ k * f\ k)$
by (*simp add*: *setsum-negf* [*symmetric*] *algebra-simps*)
also from *n* **have** $\ldots = p\ m * f\ m - p\ n * f\ n$
by (*cases n*, *simp*, *simp only*: *atLeastLessThanSuc-atLeastAtMost*, *subst setsum-Suc-diff*) *simp-all*
also from *less-imp-le*[*OF m*(*1*)] *m*(*2*) *n* **have** $\ldots \leq p\ m * f\ m$ **by** *simp*
finally show *norm* $(\sum k{\leq}n.\ f\ k) \leq (norm\ (\sum k{\leq}m.\ r * f\ k) + p\ m * f\ m)\ /$ *r* **using** *r*
by (*subst pos-le-divide-eq*[*OF r*(*1*)]) (*simp only*: *mult-ac*)
qed
moreover have $(\sum k{\leq}n.\ f\ k) \leq (\sum k{\leq}n'.\ f\ k)$ **if** $Suc\ m \leq n\ n \leq n'$ **for** *n n'*
using *less-imp-le*[*OF m*(*1*)] *that* **by** (*intro setsum-mono2*) *auto*
ultimately show *convergent* $(\lambda n.\ \sum k{\leq}n.\ f\ k)$ **by** (*rule Bseq-monoseq-convergent'-inc*)
qed

lemma *kummers-test-divergence*:
fixes *f p* :: $nat \Rightarrow real$
assumes *pos-f*: *eventually* $(\lambda n.\ f\ n > 0)$ *sequentially*
assumes *pos-p*: *eventually* $(\lambda n.\ p\ n > 0)$ *sequentially*
assumes *divergent-p*: $\neg$*summable* $(\lambda n.\ inverse\ (p\ n))$
defines $l \equiv limsup\ (\lambda n.\ ereal\ (p\ n * f\ n\ /\ f\ (Suc\ n) - p\ (Suc\ n)))$
assumes *l*: $l < 0$
shows $\neg summable\ f$
proof
assume *summable f*
from *eventually-conj*[*OF pos-f eventually-conj*[*OF pos-p Limsup-lessD*[*OF l*[*unfolded l-def*]]]]
obtain *N* **where** *N*: $\bigwedge n.\ n \geq N \Longrightarrow p\ n > 0\ \bigwedge n.\ n \geq N \Longrightarrow f\ n > 0$
$\bigwedge n.\ n \geq N \Longrightarrow p\ n * f\ n\ /\ f\ (Suc\ n) - p\ (Suc\ n) < 0$
by (*auto simp*: *eventually-at-top-linorder*)
hence *A*: $p\ n * f\ n < p\ (Suc\ n) * f\ (Suc\ n)$ **if** $n \geq N$ **for** *n* **using** *that N*[*of n*] *N*[*of Suc n*]
by (*simp add*: *field-simps*)
have $p\ n * f\ n \geq p\ N * f\ N$ **if** $n \geq N$ **for** *n* **using** *that* **and** *A*
by (*induction n rule*: *dec-induct*) (*auto intro*!: *less-imp-le elim*!: *order.trans*)
from *eventually-ge-at-top*[*of N*] *N this*
have *eventually* $(\lambda n.\ norm\ (p\ N * f\ N * inverse\ (p\ n)) \leq f\ n)$ *sequentially*
by (*auto elim*!: *eventually-mono simp*: *field-simps abs-of-pos*)
from *this* **and** ‹*summable f*› **have** *summable* $(\lambda n.\ p\ N * f\ N * inverse\ (p\ n))$
by (*rule summable-comparison-test-ev*)
from *summable-mult*[*OF this*, *of inverse* $(p\ N * f\ N)$] *N*[*OF le-refl*]
have *summable* $(\lambda n.\ inverse\ (p\ n))$ **by** (*simp add*: *divide-simps*)

with *divergent-p* **show** *False* **by** *contradiction*
qed

50.2.5 Ratio test

lemma *ratio-test-convergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos-f*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *liminf* (λ*n*. *ereal* (*f n* / *f* (*Suc n*)))
assumes *l*: *l* > *1*
shows *summable f*
proof (*rule kummers-test-convergence*[*OF pos-f*])
note *l*
also have *l* = *liminf* (λ*n*. *ereal* (*f n* / *f* (*Suc n*) − *1*)) + *1*
by (*subst Liminf-add-ereal-right*[*symmetric*]) (*simp-all add*: *minus-ereal-def l-def one-ereal-def*)
finally show *liminf* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*)) > *0*
by (*cases liminf* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*))) *simp-all*
qed *simp*

lemma *ratio-test-divergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos-f*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *limsup* (λ*n*. *ereal* (*f n* / *f* (*Suc n*)))
assumes *l*: *l* < *1*
shows ¬*summable f*
proof (*rule kummers-test-divergence*[*OF pos-f*])
have *limsup* (λ*n*. *ereal* (*f n* / *f* (*Suc n*) − *1*)) + *1* = *l*
by (*subst Limsup-add-ereal-right*[*symmetric*]) (*simp-all add*: *minus-ereal-def l-def one-ereal-def*)
also note *l*
finally show *limsup* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*)) < *0*
by (*cases limsup* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*))) *simp-all*
qed (*simp-all add*: *summable-const-iff*)

50.2.6 Raabe's test

lemma *raabes-test-convergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *liminf* (λ*n*. *ereal* (*of-nat n* ∗ (*f n* / *f* (*Suc n*) − *1*)))
assumes *l*: *l* > *1*
shows *summable f*
proof (*rule kummers-test-convergence*)
let *?l′* = *liminf* (λ*n*. *ereal* (*of-nat n* ∗ *f n* / *f* (*Suc n*) − *of-nat* (*Suc n*)))
have *1* < *l* **by** *fact*
also have *l* = *liminf* (λ*n*. *ereal* (*of-nat n* ∗ *f n* / *f* (*Suc n*) − *of-nat* (*Suc n*)) + *1*)
by (*simp add*: *l-def algebra-simps*)
also have ... = *?l′* + *1* **by** (*subst Liminf-add-ereal-right*) *simp-all*

finally show *?l′ > 0* **by** (*cases ?l′*) (*simp-all add*: *algebra-simps*)
qed (*simp-all add*: *pos*)

lemma *raabes-test-divergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *limsup* (λ*n*. *ereal* (*of-nat n* ∗ (*f n* / *f* (*Suc n*) − *1*)))
assumes *l*: *l* < *1*
shows ¬*summable f*
proof (*rule kummers-test-divergence*)
let *?l′* = *limsup* (λ*n*. *ereal* (*of-nat n* ∗ *f n* / *f* (*Suc n*) − *of-nat* (*Suc n*)))
note *l*
also have *l* = *limsup* (λ*n*. *ereal* (*of-nat n* ∗ *f n* / *f* (*Suc n*) − *of-nat* (*Suc n*)) + *1*)
by (*simp add*: *l-def algebra-simps*)
also have ... = *?l′* + *1* **by** (*subst Limsup-add-ereal-right*) *simp-all*
finally show *?l′* < *0* **by** (*cases ?l′*) (*simp-all add*: *algebra-simps*)
qed (*insert pos eventually-gt-at-top*[*of 0*::*nat*] *not-summable-harmonic*, *simp-all*)

50.3 Radius of convergence

The radius of convergence of a power series. This value always exists, ranges from *0* to ∞, and the power series is guaranteed to converge for all inputs with a norm that is smaller than that radius and to diverge for all inputs with a norm that is greater.

definition *conv-radius* :: (*nat* ⇒ *′a* :: *banach*) ⇒ *ereal* **where**
conv-radius f = *inverse* (*limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n*)))))

lemma *conv-radius-nonneg*: *conv-radius f* ≥ *0*
proof −
have *0* = *limsup* (λ*n*. *0*) **by** (*subst Limsup-const*) *simp-all*
also have ... ≤ *limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n*))))
by (*intro Limsup-mono*) (*simp-all add*: *real-root-ge-zero*)
finally show *?thesis*
unfolding *conv-radius-def* **by** (*auto simp*: *ereal-inverse-nonneg-iff*)
qed

lemma *conv-radius-zero* [*simp*]: *conv-radius* (λ-. *0*) = ∞
by (*auto simp*: *conv-radius-def zero-ereal-def* [*symmetric*] *Limsup-const*)

lemma *conv-radius-cong*:
assumes *eventually* (λ*x*. *f x* = *g x*) *sequentially*
shows *conv-radius f* = *conv-radius g*
proof −
have *eventually* (λ*n*. *ereal* (*root n* (*norm* (*f n*))) = *ereal* (*root n* (*norm* (*g n*)))) *sequentially*
using *assms* **by** *eventually-elim simp*
from *Limsup-eq*[*OF this*] **show** *?thesis* **unfolding** *conv-radius-def* **by** *simp*

qed

lemma *conv-radius-altdef*:
conv-radius f = liminf (λn. inverse (ereal (root n (norm (f n)))))
by (*subst Liminf-inverse-ereal*) (*simp-all add*: *real-root-ge-zero conv-radius-def*)

lemma *abs-summable-in-conv-radius*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real-normed-div-algebra*}
assumes *ereal (norm z) < conv-radius f*
shows *summable (λn. norm (f n * z ^ n))*
proof (*rule root-test-convergence'*)
def *l* ≡ *limsup (λn. ereal (root n (norm (f n))))*
have *0 = limsup (λn. 0)* **by** (*simp add*: *Limsup-const*)
also have ... ≤ *l* **unfolding** *l-def* **by** (*intro Limsup-mono*) (*simp-all add*: *real-root-ge-zero*)
finally have *l-nonneg*: *l ≥ 0* **.**

have *limsup (λn. root n (norm (f n * z^n))) = l * ereal (norm z)* **unfolding** *l-def*
by (*rule limsup-root-powser*)
also from *l-nonneg* **consider** *l = 0 | l = ∞ | ∃ l'. l = ereal l' ∧ l' > 0*
by (*cases l*) (*auto simp*: *less-le*)
hence *l * ereal (norm z) < 1*
proof *cases*
assume *l = ∞*
hence *conv-radius f = 0* **unfolding** *conv-radius-def l-def* **by** *simp*
with *assms* **show** *?thesis* **by** *simp*
next
assume *∃ l'. l = ereal l' ∧ l' > 0*
then guess *l'* **by** (*elim exE conjE*) **note** *l' = this*
hence *l ≠ ∞* **by** *auto*
have *l * ereal (norm z) < l * conv-radius f*
by (*intro ereal-mult-strict-left-mono*) (*simp-all add*: *l' assms*)
also have *conv-radius f = inverse l* **by** (*simp add*: *conv-radius-def l-def*)
also from *l'* **have** *l * inverse l = 1* **by** *simp*
finally show *?thesis* **.**
qed *simp-all*
finally show *limsup (λn. ereal (root n (norm (norm (f n * z ^ n))))) < 1* **by** *simp*
qed

lemma *summable-in-conv-radius*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real-normed-div-algebra*}
assumes *ereal (norm z) < conv-radius f*
shows *summable (λn. f n * z ^ n)*
by (*rule summable-norm-cancel*, *rule abs-summable-in-conv-radius*) *fact+*

lemma *not-summable-outside-conv-radius*:

fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes *ereal* $(norm\ z) > conv\text{-}radius\ f$
shows $\neg summable\ (\lambda n.\ f\ n * z\ \hat{}\ n)$
proof (*rule root-test-divergence*)
def $l \equiv limsup\ (\lambda n.\ ereal\ (root\ n\ (norm\ (f\ n))))$
have $0 = limsup\ (\lambda n.\ 0)$ **by** (*simp add*: *Limsup-const*)
also have $\ldots \leq l$ **unfolding** *l-def* **by** (*intro Limsup-mono*) (*simp-all add*: *real-root-ge-zero*)
finally have *l-nonneg*: $l \geq 0$ **.**
from *assms* **have** *l-nz*: $l \neq 0$ **unfolding** *conv-radius-def l-def* **by** *auto*

have $limsup\ (\lambda n.\ ereal\ (root\ n\ (norm\ (f\ n * z\hat{}n)))) = l * ereal\ (norm\ z)$
unfolding *l-def* **by** (*rule limsup-root-powser*)
also have $\ldots > 1$
proof (*cases l*)
assume $l = \infty$
with *assms conv-radius-nonneg*[*of f*] **show** *?thesis*
by (*auto simp*: *zero-ereal-def*[*symmetric*])
next
fix l' **assume** l': $l = ereal\ l'$
from *l-nonneg l-nz* **have** $1 = l * inverse\ l$ **by** (*auto simp*: l' *field-simps*)
also from *l-nz* **have** $inverse\ l = conv\text{-}radius\ f$
unfolding *l-def conv-radius-def* **by** *auto*
also from l' *l-nz l-nonneg assms* **have** $l * \ldots < l * ereal\ (norm\ z)$
by (*intro ereal-mult-strict-left-mono*) (*auto simp*: l')
finally show *?thesis* **.**
qed (*insert l-nonneg*, *simp-all*)
finally show $limsup\ (\lambda n.\ ereal\ (root\ n\ (norm\ (f\ n * z\hat{}n)))) > 1$ **.**
qed

lemma *conv-radius-geI*:
assumes $summable\ (\lambda n.\ f\ n * z\ \hat{}\ n :: 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\})$
shows $conv\text{-}radius\ f \geq norm\ z$
using *not-summable-outside-conv-radius*[*of f z*] *assms* **by** (*force simp*: *not-le*[*symmetric*])

lemma *conv-radius-leI*:
assumes $\neg summable\ (\lambda n.\ norm\ (f\ n * z\ \hat{}\ n :: 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}))$
shows $conv\text{-}radius\ f \leq norm\ z$
using *abs-summable-in-conv-radius*[*of z f*] *assms* **by** (*force simp*: *not-le*[*symmetric*])

lemma *conv-radius-leI'*:
assumes $\neg summable\ (\lambda n.\ f\ n * z\ \hat{}\ n :: 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\})$
shows $conv\text{-}radius\ f \leq norm\ z$
using *summable-in-conv-radius*[*of z f*] *assms* **by** (*force simp*: *not-le*[*symmetric*])

lemma *conv-radius-geI-ex*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r < R \Longrightarrow \exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n$

```
* z^n)
  shows   conv-radius f ≥ R
proof (rule linorder-cases[of conv-radius f R])
  assume R: conv-radius f < R
  with conv-radius-nonneg[of f] obtain conv-radius'
    where [simp]: conv-radius f = ereal conv-radius'
    by (cases conv-radius f) simp-all
  def r ≡ if R = ∞ then conv-radius' + 1 else (real-of-ereal R + conv-radius') / 2
   from R conv-radius-nonneg[of f] have 0 < r ∧ ereal r < R ∧ ereal r >
conv-radius f
    unfolding r-def by (cases R) (auto simp: r-def field-simps)
  with assms(1)[of r] obtain z where norm z > conv-radius f summable (λn. f
n * z^n) by auto
  with not-summable-outside-conv-radius[of f z] show ?thesis by simp
qed simp-all

lemma conv-radius-geI-ex':
  fixes f :: nat ⇒ 'a :: {banach, real-normed-div-algebra}
  assumes ⋀r. 0 < r ⟹ ereal r < R ⟹ summable (λn. f n * of-real r^n)
  shows   conv-radius f ≥ R
proof (rule conv-radius-geI-ex)
  fix r assume 0 < r ereal r < R
  with assms[of r] show ∃z. norm z = r ∧ summable (λn. f n * z ^ n)
    by (intro exI[of - of-real r :: 'a]) auto
qed

lemma conv-radius-leI-ex:
  fixes f :: nat ⇒ 'a :: {banach, real-normed-div-algebra}
  assumes R ≥ 0
  assumes ⋀r. 0 < r ⟹ ereal r > R ⟹ ∃z. norm z = r ∧ ¬summable (λn.
norm (f n * z^n))
  shows   conv-radius f ≤ R
proof (rule linorder-cases[of conv-radius f R])
  assume R: conv-radius f > R
  from R assms(1) obtain R' where R': R = ereal R' by (cases R) simp-all
  def r ≡ if conv-radius f = ∞ then R' + 1 else (R' + real-of-ereal (conv-radius
f)) / 2
  from R conv-radius-nonneg[of f] have r > R ∧ r < conv-radius f unfolding
r-def
    by (cases conv-radius f) (auto simp: r-def field-simps R')
  with assms(1) assms(2)[of r] R'
    obtain z where norm z < conv-radius f ¬summable (λn. norm (f n * z^n))
by auto
  with abs-summable-in-conv-radius[of z f] show ?thesis by auto
qed simp-all

lemma conv-radius-leI-ex':
  fixes f :: nat ⇒ 'a :: {banach, real-normed-div-algebra}
  assumes R ≥ 0
```

assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r > R \Longrightarrow \neg summable\ (\lambda n.\ f\ n * of\text{-}real\ r\ \hat{}\ n)$
shows *conv-radius f* $\leq R$
proof (*rule conv-radius-leI-ex*)
fix *r* **assume** $0 < r$ *ereal r* $> R$
with *assms*(*2*)[*of r*] **show** $\exists z.\ norm\ z = r \wedge \neg summable\ (\lambda n.\ norm\ (f\ n * z\ \hat{}\ n))$
by (*intro exI*[*of* - *of-real r* :: *'a*]) (*auto dest*: *summable-norm-cancel*)
qed *fact+*

lemma *conv-radius-eqI*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real-normed-div-algebra*}
assumes $R \geq 0$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r < R \Longrightarrow \exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z\hat{}n)$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r > R \Longrightarrow \exists z.\ norm\ z = r \wedge \neg summable\ (\lambda n.\ norm\ (f\ n * z\hat{}n))$
shows *conv-radius f* $= R$
by (*intro antisym conv-radius-geI-ex conv-radius-leI-ex assms*)

lemma *conv-radius-eqI'*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real-normed-div-algebra*}
assumes $R \geq 0$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r < R \Longrightarrow summable\ (\lambda n.\ f\ n * (of\text{-}real\ r)\ \hat{}\ n)$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r > R \Longrightarrow \neg summable\ (\lambda n.\ norm\ (f\ n * (of\text{-}real\ r)\ \hat{}\ n))$
shows *conv-radius f* $= R$
proof (*intro conv-radius-eqI*[*OF assms*(*1*)])
fix *r* **assume** $0 < r$ *ereal r* $< R$ **with** *assms*(*2*)[*OF this*]
show $\exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z\ \hat{}\ n)$ **by** *force*
next
fix *r* **assume** $0 < r$ *ereal r* $> R$ **with** *assms*(*3*)[*OF this*]
show $\exists z.\ norm\ z = r \wedge \neg summable\ (\lambda n.\ norm\ (f\ n * z\ \hat{}\ n))$ **by** *force*
qed

lemma *conv-radius-zeroI*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*,*real-normed-div-algebra*}
assumes $\bigwedge z.\ z \neq 0 \Longrightarrow \neg summable\ (\lambda n.\ f\ n * z\hat{}n)$
shows *conv-radius f* $= 0$
proof (*rule ccontr*)
assume *conv-radius f* $\neq 0$
with *conv-radius-nonneg*[*of f*] **have** *pos*: *conv-radius f* > 0 **by** *simp*
def *r* $\equiv$ *if conv-radius f* $= \infty$ *then 1 else real-of-ereal* (*conv-radius f*) / *2*
from *pos* **have** *r*: *ereal r* $> 0 \wedge$ *ereal r* $<$ *conv-radius f*
by (*cases conv-radius f*) (*simp-all add*: *r-def*)
hence *summable* $(\lambda n.\ f\ n * of\text{-}real\ r\ \hat{}\ n)$ **by** (*intro summable-in-conv-radius*) *simp*
moreover from *r* **and** *assms*[*of of-real r*] **have** $\neg summable\ (\lambda n.\ f\ n * of\text{-}real\ r\ \hat{}\ n)$ **by** *simp*
ultimately show *False* **by** *contradiction*

qed

lemma *conv-radius-inftyI'*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes $\bigwedge r.\ r > c \Longrightarrow \exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z\hat{}n)$
shows *conv-radius f* $= \infty$
proof −
{
fix *r* :: *real*
have *max r* (*c* + *1*) > *c* **by** (*auto simp*: *max-def*)
from *assms*[*OF this*] **obtain** *z* **where** *norm z* = *max r* (*c* + *1*) *summable* (λ*n*. *f n* ∗ *z*^*n*) **by** *blast*
from *conv-radius-geI*[*OF this*(*2*)] *this*(*1*) **have** *conv-radius f* ≥ *r* **by** *simp*
}
from *this*[*of real-of-ereal* (*conv-radius f* + *1*)] **show** *conv-radius f* = ∞
by (*cases conv-radius f*) *simp-all*
qed

lemma *conv-radius-inftyI*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes $\bigwedge r.\ \exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z\hat{}n)$
shows *conv-radius f* $= \infty$
using *assms* **by** (*rule conv-radius-inftyI'*)

lemma *conv-radius-inftyI''*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes $\bigwedge z.\ summable\ (\lambda n.\ f\ n * z\hat{}n)$
shows *conv-radius f* $= \infty$
proof (*rule conv-radius-inftyI'*)
fix *r* :: *real* **assume** *r* > *0*
with *assms* **show** ∃*z*. *norm z* = *r* ∧ *summable* (λ*n*. *f n* ∗ *z*^*n*)
by (*intro exI*[*of* - *of-real r*]) *simp*
qed

lemma *conv-radius-ratio-limit-ereal*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes *nz*: *eventually* (λ*n*. *f n* ≠ *0*) *sequentially*
assumes *lim*: (λ*n*. *ereal* (*norm* (*f n*) / *norm* (*f* (*Suc n*)))) ⟶ *c*
shows *conv-radius f* = *c*
proof (*rule conv-radius-eqI'*)
show *c* ≥ *0* **by** (*intro Lim-bounded2-ereal*[*OF lim*]) *simp-all*
next
fix *r* **assume** *r*: *0* < *r ereal r* < *c*
let *?l* = *liminf* (λ*n*. *ereal* (*norm* (*f n* ∗ *of-real r* ^ *n*) / *norm* (*f* (*Suc n*) ∗ *of-real r* ^ *Suc n*)))
have *?l* = *liminf* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*)))) ∗ *ereal* (*inverse r*))
using *r* **by** (*simp add*: *norm-mult norm-power divide-simps*)
also from *r* **have** ... = *liminf* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*)))))

```
* ereal (inverse r)
    by (intro Liminf-ereal-mult-right) simp-all
  also have liminf (λn. ereal (norm (f n) / (norm (f (Suc n))))) = c
    by (intro lim-imp-Liminf lim) simp
  finally have l: ?l = c * ereal (inverse r) by simp
  from r have  l': c * ereal (inverse r) > 1 by (cases c) (simp-all add: field-simps)
  show summable (λn. f n * of-real r^n)
    by (rule summable-norm-cancel, rule ratio-test-convergence)
       (insert r nz l l', auto elim!: eventually-mono)
next
  fix r assume r: 0 < r ereal r > c
  let ?l = limsup (λn. ereal (norm (f n * of-real r ^ n) / norm (f (Suc n) * of-real
r ^ Suc n)))
  have ?l = limsup (λn. ereal (norm (f n) / (norm (f (Suc n)))) * ereal (inverse
r))
    using r by (simp add: norm-mult norm-power divide-simps)
  also from r have ... = limsup (λn. ereal (norm (f n) / (norm (f (Suc n)))))
* ereal (inverse r)
    by (intro Limsup-ereal-mult-right) simp-all
  also have limsup (λn. ereal (norm (f n) / (norm (f (Suc n))))) = c
    by (intro lim-imp-Limsup lim) simp
  finally have l: ?l = c * ereal (inverse r) by simp
  from r have  l': c * ereal (inverse r) < 1 by (cases c) (simp-all add: field-simps)
  show ¬summable (λn. norm (f n * of-real r^n))
    by (rule ratio-test-divergence) (insert r nz l l', auto elim!: eventually-mono)
qed

lemma conv-radius-ratio-limit-ereal-nonzero:
  fixes f :: nat ⇒ 'a :: {banach,real-normed-div-algebra}
  assumes nz:  c ≠ 0
  assumes lim: (λn. ereal (norm (f n) / norm (f (Suc n)))) ⟶ c
  shows   conv-radius f = c
proof (rule conv-radius-ratio-limit-ereal[OF - lim], rule ccontr)
  assume ¬eventually (λn. f n ≠ 0) sequentially
  hence frequently (λn. f n = 0) sequentially by (simp add: frequently-def)
  hence frequently (λn. ereal (norm (f n) / norm (f (Suc n))) = 0) sequentially
    by (force elim!: frequently-elim1)
  hence c = 0 by (intro limit-frequently-eq[OF - - lim]) auto
  with nz show False by contradiction
qed

lemma conv-radius-ratio-limit:
  fixes f :: nat ⇒ 'a :: {banach,real-normed-div-algebra}
  assumes c' = ereal c
  assumes nz:  eventually (λn. f n ≠ 0) sequentially
  assumes lim: (λn. norm (f n) / norm (f (Suc n))) ⟶ c
  shows   conv-radius f = c'
  using assms by (intro conv-radius-ratio-limit-ereal) simp-all
```

lemma *conv-radius-ratio-limit-nonzero*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\}$
assumes $c' = ereal\ c$
assumes nz: $c \neq 0$
assumes lim: $(\lambda n.\ norm\ (f\ n)\ /\ norm\ (f\ (Suc\ n))) \longrightarrow c$
shows *conv-radius* $f = c'$
using *assms* **by** (*intro conv-radius-ratio-limit-ereal-nonzero*) *simp-all*

lemma *conv-radius-mult-power*:
assumes $c \neq (0 :: 'a :: \{real\text{-}normed\text{-}div\text{-}algebra, banach\})$
shows *conv-radius* $(\lambda n.\ c\ \hat{}\ n * f\ n) =$ *conv-radius* $f\ /$ *ereal* $(norm\ c)$
proof −
have *limsup* $(\lambda n.\ ereal\ (root\ n\ (norm\ (c\ \hat{}\ n * f\ n)))) =$
limsup $(\lambda n.\ ereal\ (norm\ c) * ereal\ (root\ n\ (norm\ (f\ n))))$
using *eventually-gt-at-top*[*of* 0::*nat*]
by (*intro Limsup-eq*)
(*auto elim*!: *eventually-mono simp*: *norm-mult norm-power real-root-mult real-root-power*)
also have $\ldots =$ *ereal* $(norm\ c) *$ *limsup* $(\lambda n.\ ereal\ (root\ n\ (norm\ (f\ n))))$
using *assms* **by** (*subst Limsup-ereal-mult-left*[*symmetric*]) *simp-all*
finally have *A*: *limsup* $(\lambda n.\ ereal\ (root\ n\ (norm\ (c\ \hat{}\ n * f\ n)))) =$
ereal $(norm\ c) *$ *limsup* $(\lambda n.\ ereal\ (root\ n\ (norm\ (f\ n))))$ **.**
show *?thesis* **using** *assms*
apply (*cases limsup* $(\lambda n.\ ereal\ (root\ n\ (norm\ (f\ n)))) = 0$)
apply (*simp add*: *A conv-radius-def*)
apply (*unfold conv-radius-def A divide-ereal-def*, *simp add*: *mult.commute ereal-inverse-mult*)
done
qed

lemma *conv-radius-mult-power-right*:
assumes $c \neq (0 :: 'a :: \{real\text{-}normed\text{-}div\text{-}algebra, banach\})$
shows *conv-radius* $(\lambda n.\ f\ n * c\ \hat{}\ n) =$ *conv-radius* $f\ /$ *ereal* $(norm\ c)$
using *conv-radius-mult-power*[*OF assms*, *of f*]
unfolding *conv-radius-def* **by** (*simp add*: *mult.commute norm-mult*)

lemma *conv-radius-divide-power*:
assumes $c \neq (0 :: 'a :: \{real\text{-}normed\text{-}div\text{-}algebra, banach\})$
shows *conv-radius* $(\lambda n.\ f\ n\ /\ c\hat{}n) =$ *conv-radius* $f *$ *ereal* $(norm\ c)$
proof −
from *assms* **have** *inverse* $c \neq 0$ **by** *simp*
from *conv-radius-mult-power-right*[*OF this*, *of f*] **show** *?thesis*
by (*simp add*: *divide-inverse divide-ereal-def assms norm-inverse power-inverse*)
qed

lemma *conv-radius-add-ge*:
min (*conv-radius f*) (*conv-radius g*) ≤
conv-radius $(\lambda x.\ f\ x + g\ x :: 'a :: \{banach, real\text{-}normed\text{-}div\text{-}algebra\})$

by (*rule conv-radius-geI-ex'*)
(*auto simp*: *algebra-simps intro*!: *summable-add summable-in-conv-radius*)

lemma *conv-radius-mult-ge*:
fixes *f g* :: *nat* ⇒ ('*a* :: {*banach*,*real-normed-div-algebra*})
shows *conv-radius* (λ*x*. ∑ *i*≤*x*. *f i* ∗ *g* (*x* − *i*)) ≥ *min* (*conv-radius f*) (*conv-radius g*)
proof (*rule conv-radius-geI-ex'*)
fix *r* **assume** *r*: *r* > *0 ereal r* < *min* (*conv-radius f*) (*conv-radius g*)
from *r* **have** *summable* (λ*n*. (∑ *i*≤*n*. (*f i* ∗ *of-real r* ^ *i*) ∗ (*g* (*n* − *i*) ∗ *of-real r* ^ (*n* − *i*))))
by (*intro summable-Cauchy-product abs-summable-in-conv-radius*) *simp-all*
thus *summable* (λ*n*. (∑ *i*≤*n*. *f i* ∗ *g* (*n* − *i*)) ∗ *of-real r* ^ *n*)
by (*simp add*: *algebra-simps of-real-def scaleR-power power-add* [*symmetric*] *scaleR-setsum-right*)
qed

end

51 Complex Transcendental Functions

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

theory *Complex-Transcendental*
imports
Complex-Analysis-Basics
Summation
begin

definition *moebius a b c d* = (λ*z*. (*a*∗*z*+*b*) / (*c*∗*z*+*d* :: '*a* :: *field*))

lemma *moebius-inverse*:
assumes *a* ∗ *d* ≠ *b* ∗ *c c* ∗ *z* + *d* ≠ *0*
shows *moebius d* (−*b*) (−*c*) *a* (*moebius a b c d z*) = *z*
proof −
from *assms* **have** (−*c*) ∗ *moebius a b c d z* + *a* ≠ *0* **unfolding** *moebius-def*
by (*simp add*: *field-simps*)
with *assms* **show** *?thesis*
unfolding *moebius-def* **by** (*simp add*: *moebius-def divide-simps*) (*simp add*: *algebra-simps*)*?*
qed

lemma *moebius-inverse'*:
assumes *a* ∗ *d* ≠ *b* ∗ *c c* ∗ *z* − *a* ≠ *0*
shows *moebius a b c d* (*moebius d* (−*b*) (−*c*) *a z*) = *z*
using *assms moebius-inverse*[*of d a* −*b* −*c z*]
by (*auto simp*: *algebra-simps*)

lemma *cmod-add-real-less*:
assumes *Im z ≠ 0 r≠0*
shows *cmod (z + r) < cmod z + |r|*
proof (*cases z*)
case (*Complex x y*)
have *r * x / |r| < sqrt (x*x + y*y)*
apply (*rule real-less-rsqrt*)
using *assms*
apply (*simp add*: *Complex power2-eq-square*)
using *not-real-square-gt-zero* **by** *blast*
then show *?thesis* **using** *assms Complex*
apply (*auto simp*: *cmod-def*)
apply (*rule power2-less-imp-less*, *auto*)
apply (*simp add*: *power2-eq-square field-simps*)
done
qed

lemma *cmod-diff-real-less*: *Im z ≠ 0 ⟹ x≠0 ⟹ cmod (z − x) < cmod z + |x|*
using *cmod-add-real-less* [*of z −x*]
by *simp*

lemma *cmod-square-less-1-plus*:
assumes *Im z = 0 ⟹ |Re z| < 1*
shows *(cmod z)² < 1 + cmod (1 − z²)*
using *assms*
apply (*cases Im z = 0 ∨ Re z = 0*)
using *abs-square-less-1*
apply (*force simp add*: *Re-power2 Im-power2 cmod-def*)
using *cmod-diff-real-less* [*of 1 − z² 1*]
apply (*simp add*: *norm-power Im-power2*)
done

51.1 The Exponential Function is Differentiable and Continuous

lemma *complex-differentiable-within-exp*: *exp complex-differentiable* (*at z within s*)
using *DERIV-exp complex-differentiable-at-within complex-differentiable-def* **by** *blast*

lemma *continuous-within-exp*:
fixes *z*::*'a*::{*real-normed-field*,*banach*}
shows *continuous* (*at z within s*) *exp*
by (*simp add*: *continuous-at-imp-continuous-within*)

lemma *continuous-on-exp*:
fixes *s*::*'a*::{*real-normed-field*,*banach*} *set*
shows *continuous-on s exp*
by (*simp add*: *continuous-on-exp continuous-on-id*)

lemma *holomorphic-on-exp*: *exp holomorphic-on s*
by (*simp add*: *complex-differentiable-within-exp holomorphic-on-def*)

51.2 Euler and de Moivre formulas.

The sine series times i

lemma *sin-ii-eq*: $(\lambda n.\ (ii * \mathit{sin\text{-}coeff}\ n) * z\hat{}n)$ *sums* $(ii * \mathit{sin}\ z)$
proof −
have $(\lambda n.\ ii * \mathit{sin\text{-}coeff}\ n *_R z\hat{}n)$ *sums* $(ii * \mathit{sin}\ z)$
using *sin-converges sums-mult* **by** *blast*
then show *?thesis*
by (*simp add*: *scaleR-conv-of-real field-simps*)
qed

theorem *exp-Euler*: $\mathit{exp}(ii * z) = \mathit{cos}(z) + ii * \mathit{sin}(z)$
proof −
have $(\lambda n.\ (\mathit{cos\text{-}coeff}\ n + ii * \mathit{sin\text{-}coeff}\ n) * z\hat{}n)$
$= (\lambda n.\ (ii * z)\ \hat{}\ n\ /_R\ (\mathit{fact}\ n))$
proof
fix n
show $(\mathit{cos\text{-}coeff}\ n + ii * \mathit{sin\text{-}coeff}\ n) * z\hat{}n = (ii * z)\ \hat{}\ n\ /_R\ (\mathit{fact}\ n)$
by (*auto simp*: *cos-coeff-def sin-coeff-def scaleR-conv-of-real field-simps elim*!: *evenE oddE*)
qed
also have ... *sums* $(\mathit{exp}\ (ii * z))$
by (*rule exp-converges*)
finally have $(\lambda n.\ (\mathit{cos\text{-}coeff}\ n + ii * \mathit{sin\text{-}coeff}\ n) * z\hat{}n)$ *sums* $(\mathit{exp}\ (ii * z))$ **.**
moreover have $(\lambda n.\ (\mathit{cos\text{-}coeff}\ n + ii * \mathit{sin\text{-}coeff}\ n) * z\hat{}n)$ *sums* $(\mathit{cos}\ z + ii * \mathit{sin}\ z)$
using *sums-add* [*OF cos-converges* [*of z*] *sin-ii-eq* [*of z*]]
by (*simp add*: *field-simps scaleR-conv-of-real*)
ultimately show *?thesis*
using *sums-unique2* **by** *blast*
qed

corollary *exp-minus-Euler*: $\mathit{exp}(-(ii * z)) = \mathit{cos}(z) - ii * \mathit{sin}(z)$
using *exp-Euler* [*of* $-z$]
by *simp*

lemma *sin-exp-eq*: $\mathit{sin}\ z = (\mathit{exp}(ii * z) - \mathit{exp}(-(ii * z))) / (2{*}ii)$
by (*simp add*: *exp-Euler exp-minus-Euler*)

lemma *sin-exp-eq'*: $\mathit{sin}\ z = ii * (\mathit{exp}(-(ii * z)) - \mathit{exp}(ii * z)) / 2$
by (*simp add*: *exp-Euler exp-minus-Euler*)

lemma *cos-exp-eq*: $\mathit{cos}\ z = (\mathit{exp}(ii * z) + \mathit{exp}(-(ii * z))) / 2$
by (*simp add*: *exp-Euler exp-minus-Euler*)

51.3 Relationships between real and complex trig functions

lemma *real-sin-eq* [*simp*]:
fixes *x*::*real*
shows *Re*(*sin*(*of-real x*)) = *sin x*
by (*simp add*: *sin-of-real*)

lemma *real-cos-eq* [*simp*]:
fixes *x*::*real*
shows *Re*(*cos*(*of-real x*)) = *cos x*
by (*simp add*: *cos-of-real*)

lemma *DeMoivre*: (*cos z* + *ii* ∗ *sin z*) ^ *n* = *cos*(*n* ∗ *z*) + *ii* ∗ *sin*(*n* ∗ *z*)
apply (*simp add*: *exp-Euler* [*symmetric*])
by (*metis exp-of-nat-mult mult.left-commute*)

lemma *exp-cnj*:
fixes *z*::*complex*
shows *cnj* (*exp z*) = *exp* (*cnj z*)
proof −
have (λ*n*. *cnj* (*z* ^ *n* /$_R$ (*fact n*))) = (λ*n*. (*cnj z*) ^*n* /$_R$ (*fact n*))
by *auto*
also have ... *sums* (*exp* (*cnj z*))
by (*rule exp-converges*)
finally have (λ*n*. *cnj* (*z* ^ *n* /$_R$ (*fact n*))) *sums* (*exp* (*cnj z*)) **.**
moreover have (λ*n*. *cnj* (*z* ^ *n* /$_R$ (*fact n*))) *sums* (*cnj* (*exp z*))
by (*metis exp-converges sums-cnj*)
ultimately show *?thesis*
using *sums-unique2*
by *blast*
qed

lemma *cnj-sin*: *cnj*(*sin z*) = *sin*(*cnj z*)
by (*simp add*: *sin-exp-eq exp-cnj field-simps*)

lemma *cnj-cos*: *cnj*(*cos z*) = *cos*(*cnj z*)
by (*simp add*: *cos-exp-eq exp-cnj field-simps*)

lemma *complex-differentiable-at-sin*: *sin complex-differentiable at z*
using *DERIV-sin complex-differentiable-def* **by** *blast*

lemma *complex-differentiable-within-sin*: *sin complex-differentiable* (*at z within s*)
by (*simp add*: *complex-differentiable-at-sin complex-differentiable-at-within*)

lemma *complex-differentiable-at-cos*: *cos complex-differentiable at z*
using *DERIV-cos complex-differentiable-def* **by** *blast*

lemma *complex-differentiable-within-cos*: *cos complex-differentiable* (*at z within s*)
by (*simp add*: *complex-differentiable-at-cos complex-differentiable-at-within*)

lemma *holomorphic-on-sin*: *sin holomorphic-on s*
by (*simp add*: *complex-differentiable-within-sin holomorphic-on-def*)

lemma *holomorphic-on-cos*: *cos holomorphic-on s*
by (*simp add*: *complex-differentiable-within-cos holomorphic-on-def*)

51.4 Get a nice real/imaginary separation in Euler's formula.

lemma *Euler*: *exp*(*z*) = *of-real*(*exp*(*Re z*)) ∗
(*of-real*(*cos*(*Im z*)) + *ii* ∗ *of-real*(*sin*(*Im z*)))
by (*cases z*) (*simp add*: *exp-add exp-Euler cos-of-real exp-of-real sin-of-real*)

lemma *Re-sin*: *Re*(*sin z*) = *sin*(*Re z*) ∗ (*exp*(*Im z*) + *exp*(−(*Im z*))) / *2*
by (*simp add*: *sin-exp-eq field-simps Re-divide Im-exp*)

lemma *Im-sin*: *Im*(*sin z*) = *cos*(*Re z*) ∗ (*exp*(*Im z*) − *exp*(−(*Im z*))) / *2*
by (*simp add*: *sin-exp-eq field-simps Im-divide Re-exp*)

lemma *Re-cos*: *Re*(*cos z*) = *cos*(*Re z*) ∗ (*exp*(*Im z*) + *exp*(−(*Im z*))) / *2*
by (*simp add*: *cos-exp-eq field-simps Re-divide Re-exp*)

lemma *Im-cos*: *Im*(*cos z*) = *sin*(*Re z*) ∗ (*exp*(−(*Im z*)) − *exp*(*Im z*)) / *2*
by (*simp add*: *cos-exp-eq field-simps Im-divide Im-exp*)

lemma *Re-sin-pos*: *0* < *Re z* ⟹ *Re z* < *pi* ⟹ *Re* (*sin z*) > *0*
by (*auto simp*: *Re-sin Im-sin add-pos-pos sin-gt-zero*)

lemma *Im-sin-nonneg*: *Re z* = *0* ⟹ *0* ≤ *Im z* ⟹ *0* ≤ *Im* (*sin z*)
by (*simp add*: *Re-sin Im-sin algebra-simps*)

lemma *Im-sin-nonneg2*: *Re z* = *pi* ⟹ *Im z* ≤ *0* ⟹ *0* ≤ *Im* (*sin z*)
by (*simp add*: *Re-sin Im-sin algebra-simps*)

51.5 More on the Polar Representation of Complex Numbers

lemma *exp-Complex*: *exp*(*Complex r t*) = *of-real*(*exp r*) ∗ *Complex* (*cos t*) (*sin t*)
by (*simp add*: *exp-add exp-Euler exp-of-real sin-of-real cos-of-real*)

lemma *exp-eq-1*: *exp z* = *1* ⟷ *Re*(*z*) = *0* ∧ (∃ *n*::*int*. *Im*(*z*) = *of-int* (*2* ∗ *n*) ∗ *pi*)
apply *auto*
apply (*metis exp-eq-one-iff norm-exp-eq-Re norm-one*)
apply (*metis Re-exp cos-one-2pi-int mult.commute mult.left-neutral norm-exp-eq-Re norm-one one-complex.simps*(*1*))
by (*metis Im-exp Re-exp complex-Re-Im-cancel-iff cos-one-2pi-int sin-double Re-complex-of-real complex-Re-numeral exp-zero mult.assoc mult.left-commute mult-eq-0-iff mult-numeral-1 numeral-One of-real-0 sin-zero-iff-int2*)

lemma *exp-eq*: *exp w = exp z ⟷ (∃ n::int. w = z + (of-int (2 * n) * pi) * ii)*
(**is** *?lhs = ?rhs*)
proof −
have *exp w = exp z ⟷ exp (w−z) = 1*
by (*simp add*: *exp-diff*)
also have *... ⟷ (Re w = Re z ∧ (∃ n::int. Im w − Im z = of-int (2 * n) * pi))*
by (*simp add*: *exp-eq-1*)
also have *... ⟷ ?rhs*
by (*auto simp*: *algebra-simps intro*!: *complex-eqI*)
finally show *?thesis* **.**
qed

lemma *exp-complex-eqI*: *|Im w − Im z| < 2∗pi ⟹ exp w = exp z ⟹ w = z*
by (*auto simp*: *exp-eq abs-mult*)

lemma *exp-integer-2pi*:
assumes *n ∈ ℤ*
shows *exp((2 * n * pi) * ii) = 1*
proof −
have *exp((2 * n * pi) * ii) = exp 0*
using *assms*
by (*simp only*: *Ints-def exp-eq*) *auto*
also have *... = 1*
by *simp*
finally show *?thesis* **.**
qed

lemma *sin-cos-eq-iff*: *sin y = sin x ∧ cos y = cos x ⟷ (∃ n::int. y = x + 2 * n * pi)*
proof −
{ assume *sin y = sin x cos y = cos x*
then have *cos (y−x) = 1*
using *cos-add* [*of y −x*] **by** *simp*
then have *∃ n::int. y−x = n * 2 * pi*
using *cos-one-2pi-int* **by** *blast* **}**
then show *?thesis*
apply (*auto simp*: *sin-add cos-add*)
apply (*metis add.commute diff-add-cancel mult.commute*)
done
qed

lemma *exp-i-ne-1*:
assumes *0 < x x < 2∗pi*
shows *exp*(i * *of-real x*) ≠ *1*
proof
assume *exp* (i * *of-real x*) = *1*
then have *exp* (i * *of-real x*) = *exp 0*
by *simp*

then obtain *n* **where** i ∗ *of-real x* = (*of-int* (*2* ∗ *n*) ∗ *pi*) ∗ i
by (*simp only*: *Ints-def exp-eq*) *auto*
then have *of-real x* = (*of-int* (*2* ∗ *n*) ∗ *pi*)
by (*metis complex-i-not-zero mult.commute mult-cancel-left of-real-eq-iff real-scaleR-def scaleR-conv-of-real*)
then have *x* = (*of-int* (*2* ∗ *n*) ∗ *pi*)
by *simp*
then show *False* **using** *assms*
by (*cases n*) (*auto simp*: *zero-less-mult-iff mult-less-0-iff*)
qed

lemma *sin-eq-0*:
fixes *z*::*complex*
shows *sin z* = *0* ⟷ (∃ *n*::*int*. *z* = *of-real*(*n* ∗ *pi*))
by (*simp add*: *sin-exp-eq exp-eq of-real-numeral*)

lemma *cos-eq-0*:
fixes *z*::*complex*
shows *cos z* = *0* ⟷ (∃ *n*::*int*. *z* = *of-real*(*n* ∗ *pi*) + *of-real pi*/*2*)
using *sin-eq-0* [*of z* − *of-real pi*/*2*]
by (*simp add*: *sin-diff algebra-simps*)

lemma *cos-eq-1*:
fixes *z*::*complex*
shows *cos z* = *1* ⟷ (∃ *n*::*int*. *z* = *of-real*(*2* ∗ *n* ∗ *pi*))
proof −
have *cos z* = *cos* (*2*∗(*z*/*2*))
by *simp*
also have ... = *1* − *2* ∗ *sin* (*z*/*2*) ^ *2*
by (*simp only*: *cos-double-sin*)
finally have [*simp*]: *cos z* = *1* ⟷ *sin* (*z*/*2*) = *0*
by *simp*
show *?thesis*
by (*auto simp*: *sin-eq-0 of-real-numeral*)
qed

lemma *csin-eq-1*:
fixes *z*::*complex*
shows *sin z* = *1* ⟷ (∃ *n*::*int*. *z* = *of-real*(*2* ∗ *n* ∗ *pi*) + *of-real pi*/*2*)
using *cos-eq-1* [*of z* − *of-real pi*/*2*]
by (*simp add*: *cos-diff algebra-simps*)

lemma *csin-eq-minus1*:
fixes *z*::*complex*
shows *sin z* = −*1* ⟷ (∃ *n*::*int*. *z* = *of-real*(*2* ∗ *n* ∗ *pi*) + *3*/*2*∗*pi*)
(**is** - = *?rhs*)
proof −
have *sin z* = −*1* ⟷ *sin* (−*z*) = *1*
by (*simp add*: *equation-minus-iff*)

also have ... $\longleftrightarrow$ ($\exists$ *n*::*int*. $-z$ = *of-real*(2 * *n* * *pi*) + *of-real pi*/2)
by (*simp only*: *csin-eq-1*)
also have ... $\longleftrightarrow$ ($\exists$ *n*::*int*. *z* = $-$ *of-real*(2 * *n* * *pi*) $-$ *of-real pi*/2)
apply (*rule iff-exI*)
by (*metis* (*no-types*) *is-num-normalize*(8) *minus-minus of-real-def real-vector.scale-minus-left uminus-add-conv-diff*)
also have ... = *?rhs*
apply (*auto simp*: *of-real-numeral*)
apply (*rule-tac* [2] *x*=$-$(*x*+1) **in** *exI*)
apply (*rule-tac x*=$-$(*x*+1) **in** *exI*)
apply (*simp-all add*: *algebra-simps*)
done
finally show *?thesis* **.**
qed

lemma *ccos-eq-minus1*:
fixes *z*::*complex*
shows *cos z* = -1 $\longleftrightarrow$ ($\exists$ *n*::*int*. *z* = *of-real*(2 * *n* * *pi*) + *pi*)
using *csin-eq-1* [*of z* $-$ *of-real pi*/2]
apply (*simp add*: *sin-diff*)
apply (*simp add*: *algebra-simps of-real-numeral equation-minus-iff*)
done

lemma *sin-eq-1*: *sin x* = 1 $\longleftrightarrow$ ($\exists$ *n*::*int*. *x* = (2 * *n* + 1 / 2) * *pi*)
(**is** - = *?rhs*)
proof $-$
have *sin x* = 1 $\longleftrightarrow$ *sin* (*complex-of-real x*) = 1
by (*metis of-real-1 one-complex.simps*(1) *real-sin-eq sin-of-real*)
also have ... $\longleftrightarrow$ ($\exists$ *n*::*int*. *complex-of-real x* = *of-real*(2 * *n* * *pi*) + *of-real pi*/2)
by (*simp only*: *csin-eq-1*)
also have ... $\longleftrightarrow$ ($\exists$ *n*::*int*. *x* = *of-real*(2 * *n* * *pi*) + *of-real pi*/2)
apply (*rule iff-exI*)
apply (*auto simp*: *algebra-simps of-real-numeral*)
apply (*rule injD* [*OF inj-of-real* [**where** *'a* = *complex*]])
apply (*auto simp*: *of-real-numeral*)
done
also have ... = *?rhs*
by (*auto simp*: *algebra-simps*)
finally show *?thesis* **.**
qed

lemma *sin-eq-minus1*: *sin x* = -1 $\longleftrightarrow$ ($\exists$ *n*::*int*. *x* = (2*n* + 3/2) * *pi*) (**is** - = *?rhs*)
proof $-$
have *sin x* = -1 $\longleftrightarrow$ *sin* (*complex-of-real x*) = -1
by (*metis Re-complex-of-real of-real-def scaleR-minus1-left sin-of-real*)
also have ... $\longleftrightarrow$ ($\exists$ *n*::*int*. *complex-of-real x* = *of-real*(2 * *n* * *pi*) + 3/2*pi*)
by (*simp only*: *csin-eq-minus1*)

```
  also have ... ⟷ (∃ n::int. x = of-real(2 * n * pi) + 3/2*pi)
    apply (rule iff-exI)
    apply (auto simp: algebra-simps)
    apply (rule injD [OF inj-of-real [where 'a = complex]], auto)
    done
  also have ... = ?rhs
    by (auto simp: algebra-simps)
  finally show ?thesis .
qed

lemma cos-eq-minus1: cos x = -1 ⟷ (∃ n::int. x = (2*n + 1) * pi)
                  (is - = ?rhs)
proof -
  have cos x = -1 ⟷ cos (complex-of-real x) = -1
    by (metis Re-complex-of-real of-real-def scaleR-minus1-left cos-of-real)
  also have ... ⟷ (∃ n::int. complex-of-real x = of-real(2 * n * pi) + pi)
    by (simp only: ccos-eq-minus1)
  also have ... ⟷ (∃ n::int. x = of-real(2 * n * pi) + pi)
    apply (rule iff-exI)
    apply (auto simp: algebra-simps)
    apply (rule injD [OF inj-of-real [where 'a = complex]], auto)
    done
  also have ... = ?rhs
    by (auto simp: algebra-simps)
  finally show ?thesis .
qed

lemma dist-exp-ii-1: norm(exp(ii * of-real t) - 1) = 2 * |sin(t / 2)|
 apply (simp add: exp-Euler cmod-def power2-diff sin-of-real cos-of-real algebra-simps)
  using cos-double-sin [of t/2]
  apply (simp add: real-sqrt-mult)
  done

lemma sinh-complex:
  fixes z :: complex
  shows (exp z - inverse (exp z)) / 2 = -ii * sin(ii * z)
  by (simp add: sin-exp-eq divide-simps exp-minus of-real-numeral)

lemma sin-ii-times:
  fixes z :: complex
  shows sin(ii * z) = ii * ((exp z - inverse (exp z)) / 2)
  using sinh-complex by auto

lemma sinh-real:
  fixes x :: real
  shows of-real((exp x - inverse (exp x)) / 2) = -ii * sin(ii * of-real x)
  by (simp add: exp-of-real sin-ii-times of-real-numeral)

lemma cosh-complex:
```

fixes z :: *complex*
shows (*exp z + inverse* (*exp z*)) / *2* = *cos*(*ii* ∗ *z*)
by (*simp add*: *cos-exp-eq divide-simps exp-minus of-real-numeral exp-of-real*)

lemma *cosh-real*:
fixes x :: *real*
shows *of-real*((*exp x + inverse* (*exp x*)) / *2*) = *cos*(*ii* ∗ *of-real x*)
by (*simp add*: *cos-exp-eq divide-simps exp-minus of-real-numeral exp-of-real*)

lemmas *cos-ii-times* = *cosh-complex* [*symmetric*]

lemma *norm-cos-squared*:
norm(*cos z*) ^ *2* = *cos*(*Re z*) ^ *2* + (*exp*(*Im z*) − *inverse*(*exp*(*Im z*))) ^ *2* / *4*
apply (*cases z*)
apply (*simp add*: *cos-add cmod-power2 cos-of-real sin-of-real*)
apply (*simp add*: *cos-exp-eq sin-exp-eq exp-minus exp-of-real Re-divide Im-divide power-divide*)
apply (*simp only*: *left-diff-distrib* [*symmetric*] *power-mult-distrib*)
apply (*simp add*: *sin-squared-eq*)
apply (*simp add*: *power2-eq-square algebra-simps divide-simps*)
done

lemma *norm-sin-squared*:
norm(*sin z*) ^ *2* = (*exp*(*2* ∗ *Im z*) + *inverse*(*exp*(*2* ∗ *Im z*)) − *2* ∗ *cos*(*2* ∗ *Re z*)) / *4*
apply (*cases z*)
apply (*simp add*: *sin-add cmod-power2 cos-of-real sin-of-real cos-double-cos exp-double*)
apply (*simp add*: *cos-exp-eq sin-exp-eq exp-minus exp-of-real Re-divide Im-divide power-divide*)
apply (*simp only*: *left-diff-distrib* [*symmetric*] *power-mult-distrib*)
apply (*simp add*: *cos-squared-eq*)
apply (*simp add*: *power2-eq-square algebra-simps divide-simps*)
done

lemma *exp-uminus-Im*: *exp* (− *Im z*) ≤ *exp* (*cmod z*)
using *abs-Im-le-cmod linear order-trans* **by** *fastforce*

lemma *norm-cos-le*:
fixes *z*::*complex*
shows *norm*(*cos z*) ≤ *exp*(*norm z*)
proof −
have *Im z* ≤ *cmod z*
using *abs-Im-le-cmod abs-le-D1* **by** *auto*
with *exp-uminus-Im* **show** *?thesis*
apply (*simp add*: *cos-exp-eq norm-divide*)
apply (*rule order-trans* [*OF norm-triangle-ineq*], *simp*)
apply (*metis add-mono exp-le-cancel-iff mult-2-right*)
done
qed

lemma *norm-cos-plus1-le*:
fixes *z*::*complex*
shows *norm*(*1* + *cos z*) ≤ *2* ∗ *exp*(*norm z*)
proof −
have *mono*: ⋀*u w z*::*real*. (*1* ≤ *w* | *1* ≤ *z*) ⟹ (*w* ≤ *u* & *z* ≤ *u*) ⟹ *2* + *w* + *z* ≤ *4* ∗ *u*
by *arith*
have ∗: *Im z* ≤ *cmod z*
using *abs-Im-le-cmod abs-le-D1* **by** *auto*
have *triangle3*: ⋀*x y z*. *norm*(*x* + *y* + *z*) ≤ *norm*(*x*) + *norm*(*y*) + *norm*(*z*)
by (*simp add*: *norm-add-rule-thm*)
have *norm*(*1* + *cos z*) = *cmod* (*1* + (*exp* (i ∗ *z*) + *exp* (− (i ∗ *z*))) / *2*)
by (*simp add*: *cos-exp-eq*)
also have ... = *cmod* ((*2* + *exp* (i ∗ *z*) + *exp* (− (i ∗ *z*))) / *2*)
by (*simp add*: *field-simps*)
also have ... = *cmod* (*2* + *exp* (i ∗ *z*) + *exp* (− (i ∗ *z*))) / *2*
by (*simp add*: *norm-divide*)
finally show *?thesis*
apply (*rule ssubst*, *simp*)
apply (*rule order-trans* [*OF triangle3*], *simp*)
using *exp-uminus-Im* ∗
apply (*auto intro*: *mono*)
done
qed

51.6 Taylor series for complex exponential, sine and cosine.

declare *power-Suc* [*simp del*]

lemma *Taylor-exp*:
norm(*exp z* − (∑ *k*≤*n*. *z* ^ *k* / (*fact k*))) ≤ *exp*|*Re z*| ∗ (*norm z*) ^ (*Suc n*) / (*fact n*)
proof (*rule complex-taylor* [*of* - *n* λ*k*. *exp exp*|*Re z*| *0 z*, *simplified*])
show *convex* (*closed-segment 0 z*)
by (*rule convex-closed-segment* [*of 0 z*])
next
fix *k x*
assume *x* ∈ *closed-segment 0 z k* ≤ *n*
show (*exp has-field-derivative exp x*) (*at x within closed-segment 0 z*)
using *DERIV-exp DERIV-subset* **by** *blast*
next
fix *x*
assume *x* ∈ *closed-segment 0 z*
then show *Re x* ≤ |*Re z*|
apply (*auto simp*: *closed-segment-def scaleR-conv-of-real*)
by (*meson abs-ge-self abs-ge-zero linear mult-left-le-one-le mult-nonneg-nonpos order-trans*)
next

```
  show 0 ∈ closed-segment 0 z
    by (auto simp: closed-segment-def)
next
  show z ∈ closed-segment 0 z
    apply (simp add: closed-segment-def scaleR-conv-of-real)
    using of-real-1 zero-le-one by blast
qed

lemma
  assumes 0 ≤ u u ≤ 1
  shows cmod-sin-le-exp: cmod (sin (u *R z)) ≤ exp |Im z|
    and cmod-cos-le-exp: cmod (cos (u *R z)) ≤ exp |Im z|
proof −
  have mono: ⋀u w z::real. w ≤ u ⟹ z ≤ u ⟹ w + z ≤ u*2
    by arith
  show cmod (sin (u *R z)) ≤ exp |Im z| using assms
   apply (auto simp: scaleR-conv-of-real norm-mult norm-power sin-exp-eq norm-divide)
    apply (rule order-trans [OF norm-triangle-ineq4])
    apply (rule mono)
    apply (auto simp: abs-if mult-left-le-one-le)
    apply (meson mult-nonneg-nonneg neg-le-0-iff-le not-le order-trans)
    apply (meson less-eq-real-def mult-nonneg-nonpos neg-0-le-iff-le order-trans)
    done
  show cmod (cos (u *R z)) ≤ exp |Im z| using assms
   apply (auto simp: scaleR-conv-of-real norm-mult norm-power cos-exp-eq norm-divide)
    apply (rule order-trans [OF norm-triangle-ineq])
    apply (rule mono)
    apply (auto simp: abs-if mult-left-le-one-le)
    apply (meson mult-nonneg-nonneg neg-le-0-iff-le not-le order-trans)
    apply (meson less-eq-real-def mult-nonneg-nonpos neg-0-le-iff-le order-trans)
    done
qed

lemma Taylor-sin:
  norm(sin z − (∑ k≤n. complex-of-real (sin-coeff k) * z ^ k))
   ≤ exp|Im z| * (norm z) ^ (Suc n) / (fact n)
proof −
  have mono: ⋀u w z::real. w ≤ u ⟹ z ≤ u ⟹ w + z ≤ u*2
     by arith
  have *: cmod (sin z −
              (∑ i≤n. (−1) ^ (i div 2) * (if even i then sin 0 else cos 0) * z ^ i /
(fact i)))
          ≤ exp |Im z| * cmod z ^ Suc n / (fact n)
  proof (rule complex-taylor [of closed-segment 0 z n
                                λk x. (−1) ^(k div 2) * (if even k then sin x else cos x)
                                exp|Im z| 0 z,  simplified])
    fix k x
   show ((λx. (− 1) ^ (k div 2) * (if even k then sin x else cos x)) has-field-derivative
          (− 1) ^ (Suc k div 2) * (if odd k then sin x else cos x))
```

```
          (at x within closed-segment 0 z)
      apply (auto simp: power-Suc)
      apply (intro derivative-eq-intros | simp)+
      done
  next
    fix x
    assume x ∈ closed-segment 0 z
    then show cmod ((− 1) ^ (Suc n div 2) * (if odd n then sin x else cos x)) ≤
exp |Im z|
      by (auto simp: closed-segment-def norm-mult norm-power cmod-sin-le-exp
cmod-cos-le-exp)
  qed
  have **: ⋀k. complex-of-real (sin-coeff k) * z ^ k
          = (−1) ^(k div 2) * (if even k then sin 0 else cos 0) * z^k / of-nat (fact
k)
    by (auto simp: sin-coeff-def elim!: oddE)
  show ?thesis
    apply (rule order-trans [OF - *])
    apply (simp add: **)
    done
qed

lemma Taylor-cos:
  norm(cos z − (∑ k≤n. complex-of-real (cos-coeff k) * z ^ k))
   ≤ exp|Im z| * (norm z) ^ Suc n / (fact n)
proof −
  have mono: ⋀u w z::real. w ≤ u ⟹ z ≤ u ⟹ w + z ≤ u*2
     by arith
  have *: cmod (cos z −
                (∑ i≤n. (−1) ^ (Suc i div 2) * (if even i then cos 0 else sin 0) * z
^ i / (fact i)))
         ≤ exp |Im z| * cmod z ^ Suc n / (fact n)
  proof (rule complex-taylor [of closed-segment 0 z n λk x. (−1) ^(Suc k div 2) *
(if even k then cos x else sin x) exp|Im z| 0 z,
simplified])
    fix k x
    assume x ∈ closed-segment 0 z k ≤ n
     show ((λx. (− 1) ^ (Suc k div 2) * (if even k then cos x else sin x))
has-field-derivative
          (− 1) ^ Suc (k div 2) * (if odd k then cos x else sin x))
           (at x within closed-segment 0 z)
     apply (auto simp: power-Suc)
     apply (intro derivative-eq-intros | simp)+
     done
  next
    fix x
    assume x ∈ closed-segment 0 z
    then show cmod ((− 1) ^ Suc (n div 2) * (if odd n then cos x else sin x)) ≤
exp |Im z|
```

by (*auto simp*: *closed-segment-def norm-mult norm-power cmod-sin-le-exp cmod-cos-le-exp*)
qed
have $**$: $\bigwedge k$. *complex-of-real* (*cos-coeff k*) $*$ *z* ^ *k*
$= (-1)\hat{}(Suc\ k\ div\ 2) * (if\ even\ k\ then\ cos\ 0\ else\ sin\ 0) * z\hat{}k\ /\ of\text{-}nat\ (fact\ k)$
by (*auto simp*: *cos-coeff-def elim*!: *evenE*)
show *?thesis*
apply (*rule order-trans* [*OF* - $*$])
apply (*simp add*: $**$)
done
qed

declare *power-Suc* [*simp*]

32-bit Approximation to e

lemma *e-approx-32*: $|exp(1) - 5837465777 / 2147483648| \leq (inverse(2\ \hat{}\ 32)::real)$
using *Taylor-exp* [*of 1 14*] *exp-le*
apply (*simp add*: *setsum-left-distrib in-Reals-norm Re-exp atMost-nat-numeral fact-numeral*)
apply (*simp only*: *pos-le-divide-eq* [*symmetric*], *linarith*)
done

lemma *e-less-3*: $exp\ 1 < (3::real)$
using *e-approx-32*
by (*simp add*: *abs-if split*: *split-if-asm*)

lemma *ln3-gt-1*: $ln\ 3 > (1::real)$
by (*metis e-less-3 exp-less-cancel-iff exp-ln-iff less-trans ln-exp*)

51.7 The argument of a complex number

definition *Arg* :: *complex* $\Rightarrow$ *real* **where**
$Arg\ z \equiv if\ z = 0\ then\ 0$
$else\ THE\ t.\ 0 \leq t \wedge t < 2*pi \wedge$
$z = of\text{-}real(norm\ z) * exp(ii * of\text{-}real\ t)$

lemma *Arg-0* [*simp*]: $Arg(0) = 0$
by (*simp add*: *Arg-def*)

lemma *Arg-unique-lemma*:
assumes *z*: $z = of\text{-}real(norm\ z) * exp(ii * of\text{-}real\ t)$
and *z'*: $z = of\text{-}real(norm\ z) * exp(ii * of\text{-}real\ t')$
and *t*: $0 \leq t\ \ t < 2*pi$
and *t'*: $0 \leq t'\ t' < 2*pi$
and *nz*: $z \neq 0$
shows $t' = t$
proof −
have [*dest*]: $\bigwedge x\ y\ z::real.\ x \geq 0 \Longrightarrow x+y < z \Longrightarrow y<z$

```
    by arith
  have of-real (cmod z) * exp (i * of-real t') = of-real (cmod z) * exp (i * of-real
t)
    by (metis z z')
  then have exp (i * of-real t') = exp (i * of-real t)
    by (metis nz mult-left-cancel mult-zero-left z)
  then have sin t' = sin t ∧ cos t' = cos t
    apply (simp add: exp-Euler sin-of-real cos-of-real)
    by (metis Complex-eq complex.sel)
  then obtain n::int where n: t' = t + 2 * n * pi
    by (auto simp: sin-cos-eq-iff)
  then have n=0
    apply (rule-tac z=n in int-cases)
    using t t'
    apply (auto simp: mult-less-0-iff algebra-simps)
    done
  then show t' = t
     by (simp add: n)
qed

lemma Arg: 0 ≤ Arg z & Arg z < 2*pi & z = of-real(norm z) * exp(ii *
of-real(Arg z))
proof (cases z=0)
  case True then show ?thesis
    by (simp add: Arg-def)
next
  case False
  obtain t where t: 0 ≤ t t < 2*pi
            and ReIm: Re z / cmod z = cos t Im z / cmod z = sin t
    using sincos-total-2pi [OF complex-unit-circle [OF False]]
    by blast
  have z: z = of-real(norm z) * exp(ii * of-real t)
    apply (rule complex-eqI)
    using t False ReIm
    apply (auto simp: exp-Euler sin-of-real cos-of-real divide-simps)
    done
  show ?thesis
    apply (simp add: Arg-def False)
    apply (rule theI [where a=t])
    using t z False
    apply (auto intro: Arg-unique-lemma)
    done
qed

corollary
  shows Arg-ge-0: 0 ≤ Arg z
    and Arg-lt-2pi: Arg z < 2*pi
    and Arg-eq: z = of-real(norm z) * exp(ii * of-real(Arg z))
  using Arg by auto
```

lemma *complex-norm-eq-1-exp*: *norm z = 1 ⟷ (∃ t. z = exp(ii * of-real t))*
using *Arg [of z]* **by** *auto*

lemma *Arg-unique*: *⟦of-real r * exp(ii * of-real a) = z; 0 < r; 0 ≤ a; a < 2*pi⟧ ⟹ Arg z = a*
apply (*rule Arg-unique-lemma [OF - Arg-eq]*)
using *Arg [of z]*
apply (*auto simp*: *norm-mult*)
done

lemma *Arg-minus*: *z ≠ 0 ⟹ Arg (−z) = (if Arg z < pi then Arg z + pi else Arg z − pi)*
apply (*rule Arg-unique [of norm z]*)
apply (*rule complex-eqI*)
using *Arg-ge-0 [of z] Arg-eq [of z] Arg-lt-2pi [of z] Arg-eq [of z]*
apply *auto*
apply (*auto simp*: *Re-exp Im-exp cos-diff sin-diff cis-conv-exp [symmetric]*)
apply (*metis Re-rcis Im-rcis rcis-def*)+
done

lemma *Arg-times-of-real [simp]*: *0 < r ⟹ Arg (of-real r * z) = Arg z*
apply (*cases z=0, simp*)
apply (*rule Arg-unique [of r * norm z]*)
using *Arg*
apply *auto*
done

lemma *Arg-times-of-real2 [simp]*: *0 < r ⟹ Arg (z * of-real r) = Arg z*
by (*metis Arg-times-of-real mult.commute*)

lemma *Arg-divide-of-real [simp]*: *0 < r ⟹ Arg (z / of-real r) = Arg z*
by (*metis Arg-times-of-real2 less-numeral-extra(3) nonzero-eq-divide-eq of-real-eq-0-iff*)

lemma *Arg-le-pi*: *Arg z ≤ pi ⟷ 0 ≤ Im z*
proof (*cases z=0*)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
have *0 ≤ Im z ⟷ 0 ≤ Im (of-real (cmod z) * exp (i * complex-of-real (Arg z)))*
by (*metis Arg-eq*)
also have *... = (0 ≤ Im (exp (i * complex-of-real (Arg z))))*
using *False*
by (*simp add*: *zero-le-mult-iff*)
also have *... ⟷ Arg z ≤ pi*
by (*simp add*: *Im-exp*) (*metis Arg-ge-0 Arg-lt-2pi sin-lt-zero sin-ge-zero not-le*)
finally show *?thesis*

by *blast*
qed

lemma *Arg-lt-pi*: $0 < Arg\ z \wedge Arg\ z < pi \longleftrightarrow 0 < Im\ z$
proof (*cases z=0*)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
have $0 < Im\ z \longleftrightarrow 0 < Im\ (of\text{-}real\ (cmod\ z) * exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ (Arg\ z)))$
by (*metis Arg-eq*)
also have ... = $(0 < Im\ (exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ (Arg\ z))))$
using *False*
by (*simp add*: *zero-less-mult-iff*)
also have ... $\longleftrightarrow 0 < Arg\ z \wedge Arg\ z < pi$
using *Arg-ge-0 Arg-lt-2pi sin-le-zero sin-gt-zero*
apply (*auto simp*: *Im-exp*)
using *le-less* **apply** *fastforce*
using *not-le* **by** *blast*
finally show *?thesis*
by *blast*
qed

lemma *Arg-eq-0*: $Arg\ z = 0 \longleftrightarrow z \in \mathbb{R} \wedge 0 \le Re\ z$
proof (*cases z=0*)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
have $z \in \mathbb{R} \wedge 0 \le Re\ z \longleftrightarrow z \in \mathbb{R} \wedge 0 \le Re\ (of\text{-}real\ (cmod\ z) * exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ (Arg\ z)))$
by (*metis Arg-eq*)
also have ... $\longleftrightarrow z \in \mathbb{R} \wedge 0 \le Re\ (exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ (Arg\ z)))$
using *False*
by (*simp add*: *zero-le-mult-iff*)
also have ... $\longleftrightarrow Arg\ z = 0$
apply (*auto simp*: *Re-exp*)
apply (*metis Arg-lt-pi Arg-ge-0 Arg-le-pi cos-pi complex-is-Real-iff leD less-linear less-minus-one-simps*(*2*) *minus-minus neg-less-eq-nonneg order-refl*)
using *Arg-eq* [*of z*]
apply (*auto simp*: *Reals-def*)
done
finally show *?thesis*
by *blast*
qed

corollary *Arg-gt-0*:
assumes $z \in \mathbb{R} \Longrightarrow Re\ z < 0$

shows *Arg z > 0*
using *Arg-eq-0 Arg-ge-0 assms dual-order.strict-iff-order* **by** *fastforce*

lemma *Arg-of-real*: *Arg(of-real x) = 0 ⟷ 0 ≤ x*
by (*simp add*: *Arg-eq-0*)

lemma *Arg-eq-pi*: *Arg z = pi ⟷ z ∈ ℝ ∧ Re z < 0*
apply (*cases z=0*, *simp*)
using *Arg-eq-0* [*of* −*z*]
apply (*auto simp*: *complex-is-Real-iff Arg-minus*)
apply (*simp add*: *complex-Re-Im-cancel-iff*)
apply (*metis Arg-minus pi-gt-zero add.left-neutral minus-minus minus-zero*)
done

lemma *Arg-eq-0-pi*: *Arg z = 0 ∨ Arg z = pi ⟷ z ∈ ℝ*
using *Arg-eq-0 Arg-eq-pi not-le* **by** *auto*

lemma *Arg-inverse*: *Arg(inverse z) = (if z ∈ ℝ ∧ 0 ≤ Re z then Arg z else 2∗pi − Arg z)*
apply (*cases z=0*, *simp*)
apply (*rule Arg-unique* [*of inverse* (*norm z*)])
using *Arg-ge-0* [*of z*] *Arg-lt-2pi* [*of z*] *Arg-eq* [*of z*] *Arg-eq-0* [*of z*] *exp-two-pi-i*
apply (*auto simp*: *of-real-numeral algebra-simps exp-diff divide-simps*)
done

lemma *Arg-eq-iff*:
assumes *w ≠ 0 z ≠ 0*
shows *Arg w = Arg z ⟷ (∃ x. 0 < x & w = of-real x ∗ z)*
using *assms Arg-eq* [*of z*] *Arg-eq* [*of w*]
apply *auto*
apply (*rule-tac x=norm w / norm z* **in** *exI*)
apply (*simp add*: *divide-simps*)
by (*metis mult.commute mult.left-commute*)

lemma *Arg-inverse-eq-0*: *Arg(inverse z) = 0 ⟷ Arg z = 0*
using *complex-is-Real-iff*
apply (*simp add*: *Arg-eq-0*)
apply (*auto simp*: *divide-simps not-sum-power2-lt-zero*)
done

lemma *Arg-divide*:
assumes *w ≠ 0 z ≠ 0 Arg w ≤ Arg z*
shows *Arg(z / w) = Arg z − Arg w*
apply (*rule Arg-unique* [*of norm(z / w)*])
using *assms Arg-eq* [*of z*] *Arg-eq* [*of w*] *Arg-ge-0* [*of w*] *Arg-lt-2pi* [*of z*]
apply (*auto simp*: *exp-diff norm-divide algebra-simps divide-simps*)
done

lemma *Arg-le-div-sum*:

assumes $w \neq 0$ $z \neq 0$ $Arg\ w \leq Arg\ z$
shows $Arg\ z = Arg\ w + Arg(z\ /\ w)$
by (*simp add*: *Arg-divide assms*)

lemma *Arg-le-div-sum-eq*:
assumes $w \neq 0$ $z \neq 0$
shows $Arg\ w \leq Arg\ z \longleftrightarrow Arg\ z = Arg\ w + Arg(z\ /\ w)$
using *assms*
by (*auto simp*: *Arg-ge-0 intro*: *Arg-le-div-sum*)

lemma *Arg-diff*:
assumes $w \neq 0$ $z \neq 0$
shows $Arg\ w - Arg\ z = (if\ Arg\ z \leq Arg\ w\ then\ Arg(w\ /\ z)\ else\ Arg(w/z) - 2*pi)$
using *assms*
apply (*auto simp*: *Arg-ge-0 Arg-divide not-le*)
using *Arg-divide* [*of w z*] *Arg-inverse* [*of w/z*]
apply *auto*
by (*metis Arg-eq-0 less-irrefl minus-diff-eq right-minus-eq*)

lemma *Arg-add*:
assumes $w \neq 0$ $z \neq 0$
shows $Arg\ w + Arg\ z = (if\ Arg\ w + Arg\ z < 2*pi\ then\ Arg(w * z)\ else\ Arg(w * z) + 2*pi)$
using *assms*
using *Arg-diff* [*of w*∗*z z*] *Arg-le-div-sum-eq* [*of z w*∗*z*]
apply (*auto simp*: *Arg-ge-0 Arg-divide not-le*)
apply (*metis Arg-lt-2pi add.commute*)
apply (*metis* (*no-types*) *Arg add.commute diff-0 diff-add-cancel diff-less-eq diff-minus-eq-add not-less*)
done

lemma *Arg-times*:
assumes $w \neq 0$ $z \neq 0$
shows $Arg\ (w * z) = (if\ Arg\ w + Arg\ z < 2*pi\ then\ Arg\ w + Arg\ z\ else\ (Arg\ w + Arg\ z) - 2*pi)$
using *Arg-add* [*OF assms*]
by *auto*

lemma *Arg-cnj*: $Arg(cnj\ z) = (if\ z \in \mathbb{R} \wedge 0 \leq Re\ z\ then\ Arg\ z\ else\ 2*pi - Arg\ z)$
apply (*cases z=0*, *simp*)
apply (*rule trans* [*of* - *Arg*(*inverse z*)])
apply (*simp add*: *Arg-eq-iff divide-simps complex-norm-square* [*symmetric*] *mult.commute*)
apply (*metis norm-eq-zero of-real-power zero-less-power2*)
apply (*auto simp*: *of-real-numeral Arg-inverse*)
done

lemma *Arg-real*: $z \in \mathbb{R} \Longrightarrow Arg\ z = (if\ 0 \leq Re\ z\ then\ 0\ else\ pi)$

using *Arg-eq-0 Arg-eq-0-pi*
by *auto*

lemma *Arg-exp*: $0 \leq Im\ z \Longrightarrow Im\ z < 2{*}pi \Longrightarrow Arg(exp\ z) = Im\ z$
by (*rule Arg-unique* [*of* $exp(Re\ z)$]) (*auto simp*: *exp-eq-polar*)

lemma *complex-split-polar*:
obtains *r a*::*real* **where** $z = complex\text{-}of\text{-}real\ r * (cos\ a + \mathrm{i} * sin\ a)\ 0 \leq r\ 0 \leq a\ a < 2{*}pi$
using *Arg cis.ctr cis-conv-exp* **by** *fastforce*

lemma *Re-Im-le-cmod*: $Im\ w * sin\ \varphi + Re\ w * cos\ \varphi \leq cmod\ w$
proof (*cases w rule*: *complex-split-polar*)
case (*1 r a*) **with** *sin-cos-le1* [*of a* φ] **show** *?thesis*
apply (*simp add*: *norm-mult cmod-unit-one*)
by (*metis* (*no-types*, *hide-lams*) *abs-le-D1 distrib-left mult.commute mult.left-commute mult-left-le*)
qed

51.8 Analytic properties of tangent function

lemma *cnj-tan*: $cnj(tan\ z) = tan(cnj\ z)$
by (*simp add*: *cnj-cos cnj-sin tan-def*)

lemma *complex-differentiable-at-tan*: $\sim(cos\ z = 0) \Longrightarrow tan\ complex\text{-}differentiable\ at\ z$
unfolding *complex-differentiable-def*
using *DERIV-tan* **by** *blast*

lemma *complex-differentiable-within-tan*: $\sim(cos\ z = 0)$
$\Longrightarrow tan\ complex\text{-}differentiable\ (at\ z\ within\ s)$
using *complex-differentiable-at-tan complex-differentiable-at-within* **by** *blast*

lemma *continuous-within-tan*: $\sim(cos\ z = 0) \Longrightarrow continuous\ (at\ z\ within\ s)\ tan$
using *continuous-at-imp-continuous-within isCont-tan* **by** *blast*

lemma *continuous-on-tan* [*continuous-intros*]: $(\bigwedge z.\ z \in s \Longrightarrow \sim(cos\ z = 0)) \Longrightarrow continuous\text{-}on\ s\ tan$
by (*simp add*: *continuous-at-imp-continuous-on*)

lemma *holomorphic-on-tan*: $(\bigwedge z.\ z \in s \Longrightarrow \sim(cos\ z = 0)) \Longrightarrow tan\ holomorphic\text{-}on\ s$
by (*simp add*: *complex-differentiable-within-tan holomorphic-on-def*)

51.9 Complex logarithms (the conventional principal value)

instantiation *complex* :: *ln*
begin

definition *ln-complex* :: *complex* ⇒ *complex*

where *ln-complex* $\equiv \lambda z.$ *THE* $w.$ $exp\ w = z\ \&\ -pi < Im(w)\ \&\ Im(w) \leq pi$

lemma
assumes $z \neq 0$
shows *exp-Ln* [*simp*]: $exp(ln\ z) = z$
and *mpi-less-Im-Ln*: $-pi < Im(ln\ z)$
and *Im-Ln-le-pi*: $Im(ln\ z) \leq pi$
proof −
obtain ψ **where** z: $z\ /\ (cmod\ z) = Complex\ (cos\ \psi)\ (sin\ \psi)$
using *complex-unimodular-polar* [*of* $z\ /\ (norm\ z)$] *assms*
by (*auto simp*: *norm-divide divide-simps*)
obtain φ **where** φ: $-\ pi < \varphi\ \varphi \leq pi\ sin\ \varphi = sin\ \psi\ cos\ \varphi = cos\ \psi$
using *sincos-principal-value* [*of* ψ] *assms*
by (*auto simp*: *norm-divide divide-simps*)
have $exp(ln\ z) = z\ \&\ -pi < Im(ln\ z)\ \&\ Im(ln\ z) \leq pi$ **unfolding** *ln-complex-def*
apply (*rule theI* [**where** $a = Complex\ (ln(norm\ z))\ \varphi$])
using z *assms* φ
apply (*auto simp*: *field-simps exp-complex-eqI exp-eq-polar cis.code*)
done
then show $exp(ln\ z) = z\ -pi < Im(ln\ z)\ Im(ln\ z) \leq pi$
by *auto*
qed

lemma *Ln-exp* [*simp*]:
assumes $-pi < Im(z)\ Im(z) \leq pi$
shows $ln(exp\ z) = z$
apply (*rule exp-complex-eqI*)
using *assms mpi-less-Im-Ln* [*of exp z*] *Im-Ln-le-pi* [*of exp z*]
apply *auto*
done

51.10 Relation to Real Logarithm

lemma *Ln-of-real*:
assumes $0 < z$
shows $ln(of\text{-}real\ z{::}complex) = of\text{-}real(ln\ z)$
proof −
have $ln(of\text{-}real\ (exp\ (ln\ z)){::}complex) = ln\ (exp\ (of\text{-}real\ (ln\ z)))$
by (*simp add*: *exp-of-real*)
also have $\ldots = of\text{-}real(ln\ z)$
using *assms*
by (*subst Ln-exp*) *auto*
finally show *?thesis*
using *assms* **by** *simp*
qed

corollary *Ln-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow Re\ z > 0 \Longrightarrow ln\ z \in \mathbb{R}$
by (*auto simp*: *Ln-of-real elim*: *Reals-cases*)

corollary *Im-Ln-of-real* [*simp*]: $r > 0 \Longrightarrow Im\ (ln\ (of\text{-}real\ r)) = 0$
by (*simp add*: *Ln-of-real*)

lemma *cmod-Ln-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow 0 < Re\ z \Longrightarrow cmod\ (ln\ z) = norm\ (ln\ (Re\ z))$
using *Ln-of-real* **by** *force*

lemma *Ln-1*: $ln\ 1 = (0::complex)$
proof −
have $ln\ (exp\ 0) = (0::complex)$
by (*metis* (*mono-tags*, *hide-lams*) *Ln-of-real exp-zero ln-one of-real-0 of-real-1 zero-less-one*)
then show *?thesis*
by *simp*
qed

instance
by *intro-classes* (*rule ln-complex-def Ln-1*)

end

abbreviation *Ln* :: *complex* ⇒ *complex*
where $Ln \equiv ln$

lemma *Ln-eq-iff*: $w \neq 0 \Longrightarrow z \neq 0 \Longrightarrow (Ln\ w = Ln\ z \longleftrightarrow w = z)$
by (*metis exp-Ln*)

lemma *Ln-unique*: $exp(z) = w \Longrightarrow -pi < Im(z) \Longrightarrow Im(z) \leq pi \Longrightarrow Ln\ w = z$
using *Ln-exp* **by** *blast*

lemma *Re-Ln* [*simp*]: $z \neq 0 \Longrightarrow Re(Ln\ z) = ln(norm\ z)$
by (*metis exp-Ln assms ln-exp norm-exp-eq-Re*)

corollary *ln-cmod-le*:
assumes *z*: $z \neq 0$
shows $ln\ (cmod\ z) \leq cmod\ (Ln\ z)$
using *norm-exp* [*of Ln z*, *simplified exp-Ln* [*OF z*]]
by (*metis Re-Ln complex-Re-le-cmod z*)

lemma *exists-complex-root*:
fixes *a* :: *complex*
shows $n \neq 0 \Longrightarrow \exists z.\ z\ \hat{}\ n = a$
apply (*cases a=0*, *simp*)
apply (*rule-tac x= exp*(*Ln*(*a*) / *n*) **in** *exI*)
apply (*auto simp*: *exp-of-nat-mult* [*symmetric*])
done

51.11 The Unwinding Number and the Ln-product Formula

Note that in this special case the unwinding number is -1, 0 or 1.

definition *unwinding* :: *complex* ⇒ *complex* **where**
unwinding(*z*) = (*z* − *Ln*(*exp z*)) / (*of-real*(*2*∗*pi*) ∗ *ii*)

lemma *unwinding-2pi*: (*2*∗*pi*) ∗ *ii* ∗ *unwinding*(*z*) = *z* − *Ln*(*exp z*)
by (*simp add*: *unwinding-def*)

lemma *Ln-times-unwinding*:
w ≠ *0* ⟹ *z* ≠ *0* ⟹ *Ln*(*w* ∗ *z*) = *Ln*(*w*) + *Ln*(*z*) − (*2*∗*pi*) ∗ *ii* ∗ *unwinding*(*Ln w* + *Ln z*)
using *unwinding-2pi* **by** (*simp add*: *exp-add*)

51.12 Derivative of Ln away from the branch cut

lemma
assumes *z* ∉ ℝ≤0
shows *has-field-derivative-Ln*: (*Ln has-field-derivative inverse*(*z*)) (*at z*)
and *Im-Ln-less-pi*: *Im* (*Ln z*) < *pi*
proof −
have *znz*: *z* ≠ *0*
using *assms* **by** *auto*
then have *Im* (*Ln z*) ≠ *pi*
by (*metis* (*no-types*) *Im-exp Ln-in-Reals assms complex-nonpos-Reals-iff complex-is-Real-iff exp-Ln mult-zero-right not-less pi-neq-zero sin-pi znz*)
then show ∗: *Im* (*Ln z*) < *pi* **using** *assms Im-Ln-le-pi*
by (*simp add*: *le-neq-trans znz*)
show (*Ln has-field-derivative inverse*(*z*)) (*at z*)
apply (*rule has-complex-derivative-inverse-strong-x*
[**where** *f* = *exp* **and** *s* = {*w*. −*pi* < *Im*(*w*) & *Im*(*w*) < *pi*}])
using *znz* ∗
apply (*auto simp*: *continuous-on-exp open-Collect-conj open-halfspace-Im-gt open-halfspace-Im-lt*)
apply (*metis DERIV-exp exp-Ln*)
apply (*metis mpi-less-Im-Ln*)
done
qed

declare *has-field-derivative-Ln* [*derivative-intros*]
declare *has-field-derivative-Ln* [*THEN DERIV-chain2*, *derivative-intros*]

lemma *complex-differentiable-at-Ln*: *z* ∉ ℝ≤0 ⟹ *Ln complex-differentiable at z*
using *complex-differentiable-def has-field-derivative-Ln* **by** *blast*

lemma *complex-differentiable-within-Ln*: *z* ∉ ℝ≤0
⟹ *Ln complex-differentiable* (*at z within s*)
using *complex-differentiable-at-Ln complex-differentiable-within-subset* **by** *blast*

lemma *continuous-at-Ln*: $z \notin \mathbb{R}_{\leq 0} \Longrightarrow$ *continuous* (*at z*) *Ln*
by (*simp add*: *complex-differentiable-imp-continuous-at complex-differentiable-within-Ln*)

lemma *isCont-Ln'* [*simp*]:
$[\![$*isCont f z*; $f\ z \notin \mathbb{R}_{\leq 0}]\!] \Longrightarrow$ *isCont* ($\lambda x.$ *Ln* (*f x*)) *z*
by (*blast intro*: *isCont-o2* [*OF - continuous-at-Ln*])

lemma *continuous-within-Ln*: $z \notin \mathbb{R}_{\leq 0} \Longrightarrow$ *continuous* (*at z within s*) *Ln*
using *continuous-at-Ln continuous-at-imp-continuous-within* **by** *blast*

lemma *continuous-on-Ln* [*continuous-intros*]: $(\bigwedge z.\ z \in s \Longrightarrow z \notin \mathbb{R}_{\leq 0}) \Longrightarrow$
continuous-on s Ln
by (*simp add*: *continuous-at-imp-continuous-on continuous-within-Ln*)

lemma *holomorphic-on-Ln*: $(\bigwedge z.\ z \in s \Longrightarrow z \notin \mathbb{R}_{\leq 0}) \Longrightarrow$ *Ln holomorphic-on s*
by (*simp add*: *complex-differentiable-within-Ln holomorphic-on-def*)

51.13 Quadrant-type results for Ln

lemma *cos-lt-zero-pi*: $pi/2 < x \Longrightarrow x < 3*pi/2 \Longrightarrow cos\ x < 0$
using *cos-minus-pi cos-gt-zero-pi* [*of x−pi*]
by *simp*

lemma *Re-Ln-pos-lt*:
assumes $z \neq 0$
shows $|Im(Ln\ z)| < pi/2 \longleftrightarrow 0 < Re(z)$
proof −
{ fix *w*
assume $w = Ln\ z$
then have *w*: $Im\ w \leq pi - pi < Im\ w$
using *Im-Ln-le-pi* [*of z*] *mpi-less-Im-Ln* [*of z*] *assms*
by *auto*
then have $|Im\ w| < pi/2 \longleftrightarrow 0 < Re(exp\ w)$
apply (*auto simp*: *Re-exp zero-less-mult-iff cos-gt-zero-pi*)
using *cos-lt-zero-pi* [*of* $-(Im\ w)$] *cos-lt-zero-pi* [*of* (*Im w*)]
apply (*simp add*: *abs-if split*: *split-if-asm*)
apply (*metis* (*no-types*) *cos-minus cos-pi-half eq-divide-eq-numeral1*(*1*) *eq-numeral-simps*(*4*)
less-numeral-extra(*3*) *linorder-neqE-linordered-idom minus-mult-minus*
minus-mult-right
mult-numeral-1-right)
done
}
then show *?thesis* **using** *assms*
by *auto*
qed

lemma *Re-Ln-pos-le*:
assumes $z \neq 0$
shows $|Im(Ln\ z)| \leq pi/2 \longleftrightarrow 0 \leq Re(z)$

proof −
{ fix *w*
assume *w* = *Ln z*
then have *w*: *Im w* ≤ *pi* − *pi* < *Im w*
using *Im-Ln-le-pi* [*of z*] *mpi-less-Im-Ln* [*of z*] *assms*
by *auto*
then have |*Im w*| ≤ *pi*/*2* ⟷ *0* ≤ *Re*(*exp w*)
apply (*auto simp*: *Re-exp zero-le-mult-iff cos-ge-zero*)
using *cos-lt-zero-pi* [*of* − (*Im w*)] *cos-lt-zero-pi* [*of* (*Im w*)] *not-le*
apply (*auto simp*: *abs-if split*: *split-if-asm*)
done
}
then show *?thesis* **using** *assms*
by *auto*
qed

lemma *Im-Ln-pos-lt*:
assumes *z* ≠ *0*
shows *0* < *Im*(*Ln z*) ∧ *Im*(*Ln z*) < *pi* ⟷ *0* < *Im*(*z*)
proof −
{ fix *w*
assume *w* = *Ln z*
then have *w*: *Im w* ≤ *pi* − *pi* < *Im w*
using *Im-Ln-le-pi* [*of z*] *mpi-less-Im-Ln* [*of z*] *assms*
by *auto*
then have *0* < *Im w* ∧ *Im w* < *pi* ⟷ *0* < *Im*(*exp w*)
using *sin-gt-zero* [*of* − (*Im w*)] *sin-gt-zero* [*of* (*Im w*)]
apply (*auto simp*: *Im-exp zero-less-mult-iff*)
using *less-linear* **apply** *fastforce*
using *less-linear* **apply** *fastforce*
done
}
then show *?thesis* **using** *assms*
by *auto*
qed

lemma *Im-Ln-pos-le*:
assumes *z* ≠ *0*
shows *0* ≤ *Im*(*Ln z*) ∧ *Im*(*Ln z*) ≤ *pi* ⟷ *0* ≤ *Im*(*z*)
proof −
{ fix *w*
assume *w* = *Ln z*
then have *w*: *Im w* ≤ *pi* − *pi* < *Im w*
using *Im-Ln-le-pi* [*of z*] *mpi-less-Im-Ln* [*of z*] *assms*
by *auto*
then have *0* ≤ *Im w* ∧ *Im w* ≤ *pi* ⟷ *0* ≤ *Im*(*exp w*)
using *sin-ge-zero* [*of* − (*Im w*)] *sin-ge-zero* [*of* (*Im w*)]
apply (*auto simp*: *Im-exp zero-le-mult-iff sin-ge-zero*)
apply (*metis not-le not-less-iff-gr-or-eq pi-not-less-zero sin-eq-0-pi*)

done }
then show *?thesis* **using** *assms*
by *auto*
qed

lemma *Re-Ln-pos-lt-imp*: $0 < Re(z) \Longrightarrow |Im(Ln\ z)| < pi/2$
by (*metis Re-Ln-pos-lt less-irrefl zero-complex.simps*(*1*))

lemma *Im-Ln-pos-lt-imp*: $0 < Im(z) \Longrightarrow 0 < Im(Ln\ z) \wedge Im(Ln\ z) < pi$
by (*metis Im-Ln-pos-lt not-le order-refl zero-complex.simps*(*2*))

A reference to the set of positive real numbers

lemma *Im-Ln-eq-0*: $z \neq 0 \Longrightarrow (Im(Ln\ z) = 0 \longleftrightarrow 0 < Re(z) \wedge Im(z) = 0)$
by (*metis Im-complex-of-real Im-exp Ln-in-Reals Re-Ln-pos-lt Re-Ln-pos-lt-imp*
Re-complex-of-real complex-is-Real-iff exp-Ln exp-of-real pi-gt-zero)

lemma *Im-Ln-eq-pi*: $z \neq 0 \Longrightarrow (Im(Ln\ z) = pi \longleftrightarrow Re(z) < 0 \wedge Im(z) = 0)$
by (*metis Im-Ln-eq-0 Im-Ln-pos-le Im-Ln-pos-lt add.left-neutral complex-eq less-eq-real-def*

mult-zero-right not-less-iff-gr-or-eq pi-ge-zero pi-neq-zero rcis-zero-arg rcis-zero-mod)

51.14 More Properties of Ln

lemma *cnj-Ln*: $z \notin \mathbb{R}_{\leq 0} \Longrightarrow cnj(Ln\ z) = Ln(cnj\ z)$
apply (*cases z=0*, *auto*)
apply (*rule exp-complex-eqI*)
apply (*auto simp*: *abs-if split*: *split-if-asm*)
using *Im-Ln-less-pi Im-Ln-le-pi* **apply** *force*
apply (*metis complex-cnj-zero-iff diff-minus-eq-add diff-strict-mono minus-less-iff*

mpi-less-Im-Ln mult.commute mult-2-right)
by (*metis exp-Ln exp-cnj*)

lemma *Ln-inverse*: $z \notin \mathbb{R}_{\leq 0} \Longrightarrow Ln(inverse\ z) = -(Ln\ z)$
apply (*cases z=0*, *auto*)
apply (*rule exp-complex-eqI*)
using *mpi-less-Im-Ln* [*of z*] *mpi-less-Im-Ln* [*of inverse z*]
apply (*auto simp*: *abs-if exp-minus split*: *split-if-asm*)
apply (*metis Im-Ln-less-pi Im-Ln-le-pi add.commute add-mono-thms-linordered-field*(*3*)
inverse-nonzero-iff-nonzero mult-2)
done

lemma *Ln-minus1* [*simp*]: $Ln(-1) = ii * pi$
apply (*rule exp-complex-eqI*)
using *Im-Ln-le-pi* [*of −1*] *mpi-less-Im-Ln* [*of −1*] *cis-conv-exp cis-pi*
apply (*auto simp*: *abs-if*)
done

lemma *Ln-ii* [*simp*]: $Ln\ ii = ii * of\text{-}real\ pi/2$

using *Ln-exp* [*of ii* ∗ (*of-real pi*/*2*)]
unfolding *exp-Euler*
by *simp*

lemma *Ln-minus-ii* [*simp*]: *Ln*(−*ii*) = − (*ii* ∗ *pi*/*2*)
proof −
have *Ln*(−*ii*) = *Ln*(*inverse ii*) **by** *simp*
also have ... = − (*Ln ii*) **using** *Ln-inverse* **by** *blast*
also have ... = − (*ii* ∗ *pi*/*2*) **by** *simp*
finally show *?thesis* **.**
qed

lemma *Ln-times*:
assumes *w* ≠ *0 z* ≠ *0*
shows *Ln*(*w* ∗ *z*) =
(*if Im*(*Ln w* + *Ln z*) ≤ −*pi then*
(*Ln*(*w*) + *Ln*(*z*)) + *ii* ∗ *of-real*(*2*∗*pi*)
else if Im(*Ln w* + *Ln z*) > *pi then*
(*Ln*(*w*) + *Ln*(*z*)) − *ii* ∗ *of-real*(*2*∗*pi*)
else Ln(*w*) + *Ln*(*z*))
using *pi-ge-zero Im-Ln-le-pi* [*of w*] *Im-Ln-le-pi* [*of z*]
using *assms mpi-less-Im-Ln* [*of w*] *mpi-less-Im-Ln* [*of z*]
by (*auto simp*: *exp-add exp-diff sin-double cos-double exp-Euler intro*!: *Ln-unique*)

corollary *Ln-times-simple*:
⟦*w* ≠ *0*; *z* ≠ *0*; −*pi* < *Im*(*Ln w*) + *Im*(*Ln z*); *Im*(*Ln w*) + *Im*(*Ln z*) ≤ *pi*⟧
⟹ *Ln*(*w* ∗ *z*) = *Ln*(*w*) + *Ln*(*z*)
by (*simp add*: *Ln-times*)

corollary *Ln-times-of-real*:
⟦*r* > *0*; *z* ≠ *0*⟧ ⟹ *Ln*(*of-real r* ∗ *z*) = *ln r* + *Ln*(*z*)
using *mpi-less-Im-Ln Im-Ln-le-pi*
by (*force simp*: *Ln-times*)

corollary *Ln-divide-of-real*:
⟦*r* > *0*; *z* ≠ *0*⟧ ⟹ *Ln*(*z* / *of-real r*) = *Ln*(*z*) − *ln r*
using *Ln-times-of-real* [*of inverse r z*]
by (*simp add*: *ln-inverse Ln-of-real mult.commute divide-inverse of-real-inverse* [*symmetric*]
del: *of-real-inverse*)

lemma *Ln-minus*:
assumes *z* ≠ *0*
shows *Ln*(−*z*) = (*if Im*(*z*) ≤ *0* ∧ ~(*Re*(*z*) < *0* ∧ *Im*(*z*) = *0*)
then Ln(*z*) + *ii* ∗ *pi*
else Ln(*z*) − *ii* ∗ *pi*) (**is** - = *?rhs*)
using *Im-Ln-le-pi* [*of z*] *mpi-less-Im-Ln* [*of z*] *assms*
Im-Ln-eq-pi [*of z*] *Im-Ln-pos-lt* [*of z*]
by (*fastforce simp*: *exp-add exp-diff exp-Euler intro*!: *Ln-unique*)

lemma *Ln-inverse-if*:
assumes $z \neq 0$
shows $Ln\ (inverse\ z) = (if\ z \in \mathbb{R}_{\leq 0}\ then\ -(Ln\ z) + \mathrm{i} * 2 * complex\text{-}of\text{-}real\ pi\ else\ -(Ln\ z))$
proof (*cases* $z \in \mathbb{R}_{\leq 0}$)
case *False* **then show** *?thesis*
by (*simp add*: *Ln-inverse*)
next
case *True*
then have *z*: $Im\ z = 0\ Re\ z < 0$
using *assms*
apply (*auto simp*: *complex-nonpos-Reals-iff*)
by (*metis complex-is-Real-iff le-imp-less-or-eq of-real-0 of-real-Re*)
have $Ln(inverse\ z) = Ln(-\ (inverse\ (-z)))$
by *simp*
also have $... = Ln\ (inverse\ (-z)) + \mathrm{i} * complex\text{-}of\text{-}real\ pi$
using *assms z*
apply (*simp add*: *Ln-minus*)
apply (*simp add*: *field-simps*)
done
also have $... = -\ Ln\ (-\ z) + \mathrm{i} * complex\text{-}of\text{-}real\ pi$
apply (*subst Ln-inverse*)
using *z* **by** (*auto simp add*: *complex-nonneg-Reals-iff*)
also have $... = -\ (Ln\ z) + \mathrm{i} * 2 * complex\text{-}of\text{-}real\ pi$
apply (*subst Ln-minus* [*OF assms*])
using *assms z*
apply *simp*
done
finally show *?thesis* **by** (*simp add*: *True*)
qed

lemma *Ln-times-ii*:
assumes $z \neq 0$
shows $Ln(ii * z) = (if\ 0 \leq Re(z) \mid Im(z) < 0$
$then\ Ln(z) + ii * of\text{-}real\ pi/2$
$else\ Ln(z) - ii * of\text{-}real(3 * pi/2))$
using *Im-Ln-le-pi* [*of z*] *mpi-less-Im-Ln* [*of z*] *assms*
Im-Ln-eq-pi [*of z*] *Im-Ln-pos-lt* [*of z*] *Re-Ln-pos-le* [*of z*]
by (*auto simp*: *Ln-times*)

lemma *Ln-of-nat*: $0 < n \Longrightarrow Ln\ (of\text{-}nat\ n) = of\text{-}real\ (ln\ (of\text{-}nat\ n))$
by (*subst of-real-of-nat-eq*[*symmetric*], *subst Ln-of-real*[*symmetric*]) *simp-all*

lemma *Ln-of-nat-over-of-nat*:
assumes $m > 0\ n > 0$
shows $Ln\ (of\text{-}nat\ m\ /\ of\text{-}nat\ n) = of\text{-}real\ (ln\ (of\text{-}nat\ m) - ln\ (of\text{-}nat\ n))$
proof −
have $of\text{-}nat\ m\ /\ of\text{-}nat\ n = (of\text{-}real\ (of\text{-}nat\ m\ /\ of\text{-}nat\ n) :: complex)$ **by** *simp*

also from *assms* **have** *Ln ... = of-real (ln (of-nat m / of-nat n))*
by (*simp add*: *Ln-of-real*[*symmetric*])
also from *assms* **have** *... = of-real (ln (of-nat m) − ln (of-nat n))*
by (*simp add*: *ln-div*)
finally show *?thesis* **.**
qed

51.15 Relation between Ln and Arg, and hence continuity of Arg

lemma *Arg-Ln*:
assumes *0 < Arg z* **shows** *Arg z = Im(Ln(−z)) + pi*
proof (*cases z = 0*)
case *True*
with *assms* **show** *?thesis*
by *simp*
next
case *False*
then have *z / of-real(norm z) = exp(ii ∗ of-real(Arg z))*
using *Arg* [*of z*]
by (*metis abs-norm-cancel nonzero-mult-divide-cancel-left norm-of-real zero-less-norm-iff*)
then have *− z / of-real(norm z) = exp* (i ∗ (*of-real* (*Arg z*) − *pi*))
using *cis-conv-exp cis-pi*
by (*auto simp*: *exp-diff algebra-simps*)
then have *ln* (*− z / of-real(norm z)*) = *ln* (*exp* (i ∗ (*of-real* (*Arg z*) − *pi*)))
by *simp*
also have *... =* i ∗ (*of-real(Arg z)* − *pi*)
using *Arg* [*of z*] *assms pi-not-less-zero*
by *auto*
finally have *Arg z = Im* (*Ln* (*− z / of-real* (*cmod z*))) + *pi*
by *simp*
also have *... = Im* (*Ln* (*−z*) − *ln* (*cmod z*)) + *pi*
by (*metis diff-0-right minus-diff-eq zero-less-norm-iff Ln-divide-of-real False*)
also have *... = Im* (*Ln* (*−z*)) + *pi*
by *simp*
finally show *?thesis* **.**
qed

lemma *continuous-at-Arg*:
assumes $z \notin \mathbb{R}_{\geq 0}$
shows *continuous* (*at z*) *Arg*
proof −
have ∗: *isCont* (λ*z*. *Im* (*Ln* (*− z*)) + *pi*) *z*
by (*rule Complex.isCont-Im isCont-Ln′ continuous-intros* | *simp add*: *assms complex-is-Real-iff*)+
have [*simp*]: ⋀*x*. ⟦*Im x ≠ 0*⟧ ⟹ *Im* (*Ln* (*− x*)) + *pi* = *Arg x*
using *Arg-Ln Arg-gt-0 complex-is-Real-iff* **by** *auto*
consider *Re z < 0* | *Im z ≠ 0* **using** *assms*
using *complex-nonneg-Reals-iff not-le* **by** *blast*

then have [*simp*]: $(\lambda z.\ Im\ (Ln\ (-\ z))\ +\ pi)\ -z\to\ Arg\ z$
using $*$ **by** (*simp add*: *isCont-def*) (*metis Arg-Ln Arg-gt-0 complex-is-Real-iff*)
show *?thesis*
apply (*simp add*: *continuous-at*)
apply (*rule Lim-transform-within-open* [**where** $s= -\mathbb{R}_{\geq 0}$ **and** $f\ =\ \lambda z.\ Im(Ln(-z))\ +\ pi$])
apply (*auto simp add*: *not-le Arg-Ln* [*OF Arg-gt-0*] *complex-nonneg-Reals-iff closed-def* [*symmetric*])
using *assms* **apply** (*force simp add*: *complex-nonneg-Reals-iff*)
done
qed

lemma *Ln-series*:
fixes z :: *complex*
assumes *norm z < 1*
shows $(\lambda n.\ (-1)\ \hat{}Suc\ n\ /\ of\text{-}nat\ n\ *\ z\hat{}n)$ *sums ln* $(1\ +\ z)$ (**is** $(\lambda n.\ ?f\ n\ *\ z\hat{}n)$ *sums* -)
proof −
let *?F* $= \lambda z.\ \sum n.\ ?f\ n\ *\ z\hat{}n$ **and** *?F′* $= \lambda z.\ \sum n.\ diffs\ ?f\ n\ *\ z\hat{}n$
have *r*: *conv-radius ?f = 1*
by (*intro conv-radius-ratio-limit-nonzero*[*of - 1*])
(*simp-all add*: *norm-divide LIMSEQ-Suc-n-over-n del*: *of-nat-Suc*)

have $\exists\, c.\ \forall\, z{\in}ball\ 0\ 1.\ ln\ (1\ +\ z)\ -\ ?F\ z\ =\ c$
proof (*rule has-field-derivative-zero-constant*)
fix z :: *complex* **assume** *z′*: $z \in ball\ 0\ 1$
hence *z*: *norm z < 1* **by** (*simp add*: *dist-0-norm*)
def $t \equiv of\text{-}real\ (1\ +\ norm\ z)\ /\ 2$ :: *complex*
from *z* **have** *t*: *norm z < norm t norm t < 1* **unfolding** *t-def*
by (*simp-all add*: *field-simps norm-divide del*: *of-real-add*)

have *Re* $(-z) \leq$ *norm* $(-z)$ **by** (*rule complex-Re-le-cmod*)
also from *z* **have** *... < 1* **by** *simp*
finally have $((\lambda z.\ ln\ (1\ +\ z))$ *has-field-derivative inverse* $(1+z))$ $(at\ z)$
by (*auto intro*!: *derivative-eq-intros simp*: *complex-nonpos-Reals-iff*)
moreover have (*?F has-field-derivative ?F′ z*) (*at z*) **using** *t r*
by (*intro termdiffs-strong*[*of - t*] *summable-in-conv-radius*) *simp-all*
ultimately have $((\lambda z.\ ln\ (1\ +\ z)\ -\ ?F\ z)$ *has-field-derivative* $(inverse\ (1\ +\ z)\ -\ ?F'\ z))$
$(at\ z\ within\ ball\ 0\ 1)$
by (*intro derivative-intros*) (*simp-all add*: *at-within-open*[*OF z′*])
also have $(\lambda n.\ of\text{-}nat\ n\ *\ ?f\ n\ *\ z\ \hat{}\ (n\ -\ Suc\ 0))$ *sums ?F′ z* **using** *t r*
by (*intro diffs-equiv termdiff-converges*[*OF t*(*1*)] *summable-in-conv-radius*) *simp-all*
from *sums-split-initial-segment*[*OF this, of 1*]
have $(\lambda i.\ (-z)\ \hat{}\ i)$ *sums ?F′ z* **by** (*simp add*: *power-minus*[*of z*] *del*: *of-nat-Suc*)
hence *?F′ z* $=$ *inverse* $(1\ +\ z)$ **using** *z* **by** (*simp add*: *sums-iff suminf-geometric divide-inverse*)

```
    also have inverse (1 + z) − inverse (1 + z) = 0 by simp
    finally show ((λz. ln (1 + z) − ?F z) has-field-derivative 0) (at z within ball
0 1) .
  qed simp-all
  then obtain c where c: ⋀z. z ∈ ball 0 1 ⟹ ln (1 + z) − ?F z = c by blast
  from c[of 0] have c = 0 by (simp only: powser-zero) simp
  with c[of z] assms have ln (1 + z) = ?F z by (simp add: dist-0-norm)
  moreover have summable (λn. ?f n * z^n) using assms r
    by (intro summable-in-conv-radius) simp-all
  ultimately show ?thesis by (simp add: sums-iff)
qed

lemma Ln-approx-linear:
  fixes z :: complex
  assumes norm z < 1
  shows   norm (ln (1 + z) − z) ≤ norm z^2 / (1 − norm z)
proof −
  let ?f = λn. (−1) ^Suc n / of-nat n
  from assms have (λn. ?f n * z^n) sums ln (1 + z) using Ln-series by simp
 moreover have (λn. (if n = 1 then 1 else 0) * z^n) sums z using powser-sums-if[of
1] by simp
  ultimately have (λn. (?f n − (if n = 1 then 1 else 0)) * z^n) sums (ln (1 +
z) − z)
    by (subst left-diff-distrib, intro sums-diff) simp-all
  from sums-split-initial-segment[OF this, of Suc 1]
   have (λi. (−(z^2)) * inverse (2 + of-nat i) * (− z) ^i) sums (Ln (1 + z) − z)
    by (simp add: power2-eq-square mult-ac power-minus[of z] divide-inverse)
  hence (Ln (1 + z) − z) = (∑ i. (−(z^2)) * inverse (of-nat (i+2)) * (−z) ^i)
    by (simp add: sums-iff)
  also have A: summable (λn. norm z^2 * (inverse (real-of-nat (Suc (Suc n))) *
cmod z ^ n))
   by (rule summable-mult, rule summable-comparison-test-ev[OF - summable-geometric[of
norm z]])
       (auto simp: assms field-simps intro!: always-eventually)
  hence norm (∑ i. (−(z^2)) * inverse (of-nat (i+2)) * (−z) ^i) ≤
           (∑ i. norm (−(z^2) * inverse (of-nat (i+2)) * (−z) ^i))
    by (intro summable-norm)
      (auto simp: norm-power norm-inverse norm-mult mult-ac simp del: of-nat-add
of-nat-Suc)
  also have norm ((−z) ^2 * (−z) ^i) * inverse (of-nat (i+2)) ≤ norm ((−z) ^2
* (−z) ^i) * 1 for i
    by (intro mult-left-mono) (simp-all add: divide-simps)
  hence (∑ i. norm (−(z^2) * inverse (of-nat (i+2)) * (−z) ^i)) ≤
          (∑ i. norm (−(z^2) * (−z) ^i)) using A assms
    apply (simp-all only: norm-power norm-inverse norm-divide norm-mult)
    apply (intro suminf-le summable-mult summable-geometric)
    apply (auto simp: norm-power field-simps simp del: of-nat-add of-nat-Suc)
    done
  also have ... = norm z^2 * (∑ i. norm z^i) using assms
```

by (*subst suminf-mult* [*symmetric*]) (*auto intro*!: *summable-geometric simp*: *norm-mult norm-power*)
also have $(\sum i.\ norm\ z\hat{}i) = inverse\ (1 - norm\ z)$ **using** *assms*
by (*subst suminf-geometric*) (*simp-all add*: *divide-inverse*)
also have $norm\ z\hat{}2 * ... = norm\ z\hat{}2\ /\ (1 - norm\ z)$ **by** (*simp add*: *divide-inverse*)
finally show *?thesis* **.**
qed

Relation between Arg and arctangent in upper halfplane

lemma *Arg-arctan-upperhalf*:
assumes $0 < Im\ z$
shows $Arg\ z = pi/2 - arctan(Re\ z\ /\ Im\ z)$
proof (*cases* $z = 0$)
case *True* **with** *assms* **show** *?thesis*
by *simp*
next
case *False*
show *?thesis*
apply (*rule Arg-unique* [*of norm z*])
using *False assms arctan* [*of Re z / Im z*] *pi-ge-two pi-half-less-two*
apply (*auto simp*: *exp-Euler cos-diff sin-diff*)
using *norm-complex-def* [*of z*, *symmetric*]
apply (*simp add*: *sin-of-real cos-of-real sin-arctan cos-arctan field-simps real-sqrt-divide*)
apply (*metis complex-eq mult.assoc ring-class.ring-distribs*(*2*))
done
qed

lemma *Arg-eq-Im-Ln*:
assumes $0 \le Im\ z\ 0 < Re\ z$
shows $Arg\ z = Im\ (Ln\ z)$
proof (*cases* $z = 0 \vee Im\ z = 0$)
case *True* **then show** *?thesis*
using *assms Arg-eq-0 complex-is-Real-iff*
apply *auto*
by (*metis Arg-eq-0-pi Arg-eq-pi Im-Ln-eq-0 Im-Ln-eq-pi less-numeral-extra*(*3*) *zero-complex.simps*(*1*))
next
case *False*
then have $Arg\ z > 0$
using *Arg-gt-0 complex-is-Real-iff* **by** *blast*
then show *?thesis*
using *assms False*
by (*subst Arg-Ln*) (*auto simp*: *Ln-minus*)
qed

lemma *continuous-within-upperhalf-Arg*:
assumes $z \neq 0$
shows *continuous* (*at z within* $\{z.\ 0 \le Im\ z\}$) *Arg*
proof (*cases* $z \in \mathbb{R}_{\ge 0}$)

case *False* **then show** *?thesis*
using *continuous-at-Arg continuous-at-imp-continuous-within* **by** *auto*
next
case *True*
then have *z*: $z \in \mathbb{R}$ $0 < Re\ z$
using *assms* **by** (*auto simp*: *complex-nonneg-Reals-iff complex-is-Real-iff complex-neq-0*)
then have [*simp*]: $Arg\ z = 0$ $Im\ (Ln\ z) = 0$
by (*auto simp*: *Arg-eq-0 Im-Ln-eq-0 assms complex-is-Real-iff*)
show *?thesis*
proof (*clarsimp simp add*: *continuous-within Lim-within dist-norm*)
fix *e*::*real*
assume $0 < e$
moreover have *continuous* (*at z*) ($\lambda x.\ Im\ (Ln\ x)$)
using *z* **by** (*simp add*: *continuous-at-Ln complex-nonpos-Reals-iff*)
ultimately
obtain *d* **where** *d*: $d>0$ $\bigwedge x.\ x \neq z \Longrightarrow cmod\ (x - z) < d \Longrightarrow |Im\ (Ln\ x)| < e$
by (*auto simp*: *continuous-within Lim-within dist-norm*)
{ fix *x*
assume $cmod\ (x - z) < Re\ z\ /\ 2$
then have $|Re\ x - Re\ z| < Re\ z\ /\ 2$
by (*metis le-less-trans abs-Re-le-cmod minus-complex.simps*(*1*))
then have $0 < Re\ x$
using *z* **by** *linarith*
}
then show $\exists\, d>0.\ \forall x.\ 0 \leq Im\ x \longrightarrow x \neq z \wedge cmod\ (x - z) < d \longrightarrow |Arg\ x| < e$
apply (*rule-tac x*=*min d* (*Re z* / *2*) **in** *exI*)
using *z d*
apply (*auto simp*: *Arg-eq-Im-Ln*)
done
qed
qed

lemma *continuous-on-upperhalf-Arg*: *continuous-on* ($\{z.\ 0 \leq Im\ z\} - \{0\}$) *Arg*
apply (*auto simp*: *continuous-on-eq-continuous-within*)
by (*metis Diff-subset continuous-within-subset continuous-within-upperhalf-Arg*)

lemma *open-Arg-less-Int*:
assumes $0 \leq s$ $t \leq 2*pi$
shows *open* ($\{y.\ s < Arg\ y\} \cap \{y.\ Arg\ y < t\}$)
proof −
have *1*: *continuous-on* (*UNIV* − $\mathbb{R}_{\geq 0}$) *Arg*
using *continuous-at-Arg continuous-at-imp-continuous-within*
by (*auto simp*: *continuous-on-eq-continuous-within*)
have *2*: *open* (*UNIV* − $\mathbb{R}_{\geq 0}$ :: *complex set*) **by** (*simp add*: *open-Diff*)
have *open* ($\{z.\ s < z\} \cap \{z.\ z < t\}$)
using *open-lessThan* [*of t*] *open-greaterThan* [*of s*]
by (*metis greaterThan-def lessThan-def open-Int*)

moreover have $\{y.\ s < Arg\ y\} \cap \{y.\ Arg\ y < t\} \subseteq -\ \mathbb{R}_{\geq 0}$
using *assms* **by** (*auto simp*: *Arg-real complex-nonneg-Reals-iff complex-is-Real-iff*)
ultimately show *?thesis*
using *continuous-imp-open-vimage* [*OF 1 2*, *of* $\{z.\ Re\ z > s\} \cap \{z.\ Re\ z < t\}$]
by *auto*
qed

lemma *open-Arg-gt*: *open* $\{z.\ t < Arg\ z\}$
proof (*cases* $t < 0$)
case *True* **then have** $\{z.\ t < Arg\ z\} = UNIV$
using *Arg-ge-0 less-le-trans* **by** *auto*
then show *?thesis*
by *simp*
next
case *False* **then show** *?thesis*
using *open-Arg-less-Int* [*of t 2∗pi*] *Arg-lt-2pi*
by *auto*
qed

lemma *closed-Arg-le*: *closed* $\{z.\ Arg\ z \leq t\}$
using *open-Arg-gt* [*of t*]
by (*simp add*: *closed-def Set.Collect-neg-eq* [*symmetric*] *not-le*)

51.16 Complex Powers

lemma *powr-to-1* [*simp*]: *z powr 1* = (*z*::*complex*)
by (*simp add*: *powr-def*)

lemma *powr-nat*:
fixes *n*::*nat* **and** *z*::*complex* **shows** *z powr n* = (*if z* = *0 then 0 else z^n*)
by (*simp add*: *exp-of-nat-mult powr-def*)

lemma *powr-add-complex*:
fixes *w*::*complex* **shows** *w powr* (*z1* + *z2*) = *w powr z1* ∗ *w powr z2*
by (*simp add*: *powr-def algebra-simps exp-add*)

lemma *powr-minus-complex*:
fixes *w*::*complex* **shows** *w powr* (−*z*) = *inverse*(*w powr z*)
by (*simp add*: *powr-def exp-minus*)

lemma *powr-diff-complex*:
fixes *w*::*complex* **shows** *w powr* (*z1* − *z2*) = *w powr z1* / *w powr z2*
by (*simp add*: *powr-def algebra-simps exp-diff*)

lemma *norm-powr-real*: $w \in \mathbb{R} \Longrightarrow 0 < Re\ w \Longrightarrow norm(w\ powr\ z) = exp(Re\ z * ln(Re\ w))$
apply (*simp add*: *powr-def*)
using *Im-Ln-eq-0 complex-is-Real-iff norm-complex-def*

by *auto*

lemma *cnj-powr*:
assumes *Im a = 0 ⟹ Re a ≥ 0*
shows *cnj (a powr b) = cnj a powr cnj b*
proof (*cases a = 0*)
case *False*
with *assms* **have** *a ∉ ℝ≤0* **by** (*auto simp*: *complex-eq-iff complex-nonpos-Reals-iff*)
with *False* **show** *?thesis* **by** (*simp add*: *powr-def exp-cnj cnj-Ln*)
qed *simp*

lemma *powr-real-real*:
⟦*w ∈ ℝ*; *z ∈ ℝ*; *0 < Re w*⟧ *⟹ w powr z = exp(Re z * ln(Re w))*
apply (*simp add*: *powr-def*)
by (*metis complex-eq complex-is-Real-iff diff-0 diff-0-right diff-minus-eq-add exp-ln exp-not-eq-zero*
exp-of-real Ln-of-real mult-zero-right of-real-0 of-real-mult)

lemma *powr-of-real*:
fixes *x*::*real* **and** *y*::*real*
shows *0 < x ⟹ of-real x powr (of-real y*::*complex) = of-real (x powr y)*
by (*simp add*: *powr-def*) (*metis exp-of-real of-real-mult Ln-of-real*)

lemma *norm-powr-real-mono*:
⟦*w ∈ ℝ*; *1 < Re w*⟧
⟹ cmod(w powr z1) ≤ cmod(w powr z2) ⟷ Re z1 ≤ Re z2
by (*auto simp*: *powr-def algebra-simps Reals-def Ln-of-real*)

lemma *powr-times-real*:
⟦*x ∈ ℝ*; *y ∈ ℝ*; *0 ≤ Re x*; *0 ≤ Re y*⟧
*⟹ (x * y) powr z = x powr z * y powr z*
by (*auto simp*: *Reals-def powr-def Ln-times exp-add algebra-simps less-eq-real-def Ln-of-real*)

lemma *powr-neg-real-complex*:
shows *(− of-real x) powr a = (−1) powr (of-real (sgn x) * a) * of-real x powr (a :: complex)*
proof (*cases x = 0*)
assume *x*: *x ≠ 0*
hence *(−x) powr a = exp (a * ln (−of-real x))* **by** (*simp add*: *powr-def*)
also from *x* **have** *ln (−of-real x) = Ln (of-real x) + of-real (sgn x) * pi * ∗*
by (*simp add*: *Ln-minus Ln-of-real*)
also from *x assms* **have** *exp (a * ...) = cis pi powr (of-real (sgn x) * a) * of-real x powr a*
by (*simp add*: *powr-def exp-add algebra-simps Ln-of-real cis-conv-exp*)
also note *cis-pi*
finally show *?thesis* **by** *simp*
qed *simp-all*

lemma *has-field-derivative-powr*:
fixes z :: *complex*
shows $z \notin \mathbb{R}_{\leq 0} \Longrightarrow ((\lambda z.\ z\ powr\ s)\ has\text{-}field\text{-}derivative\ (s * z\ powr\ (s - 1)))$ $(at\ z)$
apply (*cases z=0*, *auto*)
apply (*simp add*: *powr-def*)
apply (*rule DERIV-transform-at* [**where** $d = norm\ z$ **and** $f = \lambda z.\ exp\ (s * Ln\ z)$])
apply (*auto simp*: *dist-complex-def*)
apply (*intro derivative-eq-intros* | *simp add*: *assms*)+
apply (*simp add*: *field-simps exp-diff*)
done

declare *has-field-derivative-powr*[*THEN DERIV-chain2*, *derivative-intros*]

lemma *has-field-derivative-powr-right*:
$w \neq 0 \Longrightarrow ((\lambda z.\ w\ powr\ z)\ has\text{-}field\text{-}derivative\ Ln\ w * w\ powr\ z)\ (at\ z)$
apply (*simp add*: *powr-def*)
apply (*intro derivative-eq-intros* | *simp add*: *assms*)+
done

lemma *complex-differentiable-powr-right*:
$w \neq 0 \Longrightarrow (\lambda z.\ w\ powr\ z)\ complex\text{-}differentiable\ (at\ z)$
using *complex-differentiable-def has-field-derivative-powr-right* **by** *blast*

lemma *holomorphic-on-powr-right*:
$f\ holomorphic\text{-}on\ s \Longrightarrow w \neq 0 \Longrightarrow (\lambda z.\ w\ powr\ (f\ z))\ holomorphic\text{-}on\ s$
unfolding *holomorphic-on-def*
using *DERIV-chain'* *complex-differentiable-def has-field-derivative-powr-right*
by *fastforce*

lemma *norm-powr-real-powr*:
$w \in \mathbb{R} \Longrightarrow 0 < Re\ w \Longrightarrow norm(w\ powr\ z) = Re\ w\ powr\ Re\ z$
by (*auto simp add*: *norm-powr-real powr-def Im-Ln-eq-0 complex-is-Real-iff in-Reals-norm*)

51.17 Some Limits involving Logarithms

lemma *lim-Ln-over-power*:
fixes *s*::*complex*
assumes $0 < Re\ s$
shows $((\lambda n.\ Ln\ n\ /\ (n\ powr\ s)) \longrightarrow 0)\ sequentially$
proof (*simp add*: *lim-sequentially dist-norm*, *clarify*)
fix *e*::*real*
assume *e*: $0 < e$
have $\exists xo>0.\ \forall x \geq xo.\ 0 < e * 2 + (e * Re\ s * 2 - 2) * x + e * (Re\ s)^2 * x^2$
proof (*rule-tac x=2/(e * (Re s)^2)* **in** *exI*, *safe*)
show $0 < 2\ /\ (e * (Re\ s)^2)$
using *e assms* **by** (*simp add*: *field-simps*)

next
fix x::*real*
assume x: $2 \,/\, (e * (Re\ s)^2) \leq x$
then have $x>0$
using *e assms*
by (*metis less-le-trans mult-eq-0-iff mult-pos-pos pos-less-divide-eq power2-eq-square zero-less-numeral*)
then show $0 < e * 2 + (e * Re\ s * 2 - 2) * x + e * (Re\ s)^2 * x^2$
using *e assms x*
apply (*auto simp*: *field-simps*)
apply (*rule-tac* $y = e * (x^2 * (Re\ s)^2)$ **in** *le-less-trans*)
apply (*auto simp*: *power2-eq-square field-simps add-pos-pos*)
done
qed
then have $\exists xo>0.\ \forall x{\geq}xo.\ x \,/\, e < 1 + (Re\ s * x) + (1/2) * (Re\ s * x)\ \hat{}\ 2$
using *e* **by** (*simp add*: *field-simps*)
then have $\exists xo>0.\ \forall x{\geq}xo.\ x \,/\, e < exp\ (Re\ s * x)$
using *assms*
by (*force intro*: *less-le-trans* [*OF - exp-lower-taylor-quadratic*])
then have $\exists xo>0.\ \forall x{\geq}xo.\ x < e * exp\ (Re\ s * x)$
using *e* **by** (*auto simp*: *field-simps*)
with *e* **show** $\exists no.\ \forall n{\geq}no.\ norm\ (Ln\ (of\text{-}nat\ n) \,/\, of\text{-}nat\ n\ powr\ s) < e$
apply (*auto simp*: *norm-divide norm-powr-real divide-simps*)
apply (*rule-tac* $x{=}nat\ \lceil exp\ xo \rceil$ **in** *exI*)
apply *clarify*
apply (*drule-tac* $x{=}ln\ n$ **in** *spec*)
apply (*auto simp*: *exp-less-mono nat-ceiling-le-eq not-le*)
apply (*metis exp-less-mono exp-ln not-le of-nat-0-less-iff*)
done
qed

lemma *lim-Ln-over-n*: $((\lambda n.\ Ln(of\text{-}nat\ n) \,/\, of\text{-}nat\ n) \longrightarrow 0)\ sequentially$
using *lim-Ln-over-power* [*of 1*]
by *simp*

lemma *Ln-Reals-eq*: $x \in \mathbb{R} \Longrightarrow Re\ x > 0 \Longrightarrow Ln\ x = of\text{-}real\ (ln\ (Re\ x))$
using *Ln-of-real* **by** *force*

lemma *powr-Reals-eq*: $x \in \mathbb{R} \Longrightarrow Re\ x > 0 \Longrightarrow x\ powr\ complex\text{-}of\text{-}real\ y = of\text{-}real\ (x\ powr\ y)$
by (*simp add*: *powr-of-real*)

lemma *lim-ln-over-power*:
fixes s :: *real*
assumes $0 < s$
shows $((\lambda n.\ ln\ n \,/\, (n\ powr\ s)) \longrightarrow 0)\ sequentially$
using *lim-Ln-over-power* [*of of-real s*, *THEN filterlim-sequentially-Suc* [*THEN iffD2*]] *assms*
apply (*subst filterlim-sequentially-Suc* [*symmetric*])

apply (*simp add*: *lim-sequentially dist-norm*
Ln-Reals-eq norm-powr-real-powr norm-divide)
done

lemma *lim-ln-over-n*: ((λ*n*. *ln*(*real-of-nat n*) / *of-nat n*) ⟶ *0*) *sequentially*
using *lim-ln-over-power* [*of 1*, *THEN filterlim-sequentially-Suc* [*THEN iffD2*]]
apply (*subst filterlim-sequentially-Suc* [*symmetric*])
apply (*simp add*: *lim-sequentially dist-norm*)
done

lemma *lim-1-over-complex-power*:
assumes *0* < *Re s*
shows ((λ*n*. *1* / (*of-nat n powr s*)) ⟶ *0*) *sequentially*
proof −
have ∀ *n*>*0*. *3* ≤ *n* ⟶ *1* ≤ *ln* (*real-of-nat n*)
using *ln3-gt-1*
by (*force intro*: *order-trans* [*of - ln 3*] *ln3-gt-1*)
moreover have (λ*n*. *cmod* (*Ln* (*of-nat n*) / *of-nat n powr s*)) ⟶ *0*
using *lim-Ln-over-power* [*OF assms*]
by (*metis tendsto-norm-zero-iff*)
ultimately show *?thesis*
apply (*auto intro*!: *Lim-null-comparison* [**where** *g* = λ*n*. *norm* (*Ln*(*of-nat n*) / *of-nat n powr s*)])
apply (*auto simp*: *norm-divide divide-simps eventually-sequentially*)
done
qed

lemma *lim-1-over-real-power*:
fixes *s* :: *real*
assumes *0* < *s*
shows ((λ*n*. *1* / (*of-nat n powr s*)) ⟶ *0*) *sequentially*
using *lim-1-over-complex-power* [*of of-real s*, *THEN filterlim-sequentially-Suc* [*THEN iffD2*]] *assms*
apply (*subst filterlim-sequentially-Suc* [*symmetric*])
apply (*simp add*: *lim-sequentially dist-norm*)
apply (*simp add*: *Ln-Reals-eq norm-powr-real-powr norm-divide*)
done

lemma *lim-1-over-Ln*: ((λ*n*. *1* / *Ln*(*of-nat n*)) ⟶ *0*) *sequentially*
proof (*clarsimp simp add*: *lim-sequentially dist-norm norm-divide divide-simps*)
fix *r*::*real*
assume *0* < *r*
have *ir*: *inverse* (*exp* (*inverse r*)) > *0*
by *simp*
obtain *n* **where** *n*: *1* < *of-nat n* ∗ *inverse* (*exp* (*inverse r*))
using *ex-less-of-nat-mult* [*of - 1*, *OF ir*]
by *auto*
then have *exp* (*inverse r*) < *of-nat n*
by (*simp add*: *divide-simps*)

then have *ln (exp (inverse r)) < ln (of-nat n)*
by (*metis exp-gt-zero less-trans ln-exp ln-less-cancel-iff*)
with ‹*0 < r*› **have** *1 < r * ln (real-of-nat n)*
by (*simp add*: *field-simps*)
moreover have *n > 0* **using** *n*
using *neq0-conv* **by** *fastforce*
ultimately show $\exists no.\ \forall n.\ Ln\ (of\text{-}nat\ n) \neq 0 \longrightarrow no \leq n \longrightarrow 1 < r * cmod\ (Ln\ (of\text{-}nat\ n))$
using *n* ‹*0 < r*›
apply (*rule-tac x=n* **in** *exI*)
apply (*auto simp*: *divide-simps*)
apply (*erule less-le-trans*, *auto*)
done
qed

lemma *lim-1-over-ln*: $((\lambda n.\ 1\ /\ ln(real\text{-}of\text{-}nat\ n)) \longrightarrow 0)\ sequentially$
using *lim-1-over-Ln* [*THEN filterlim-sequentially-Suc* [*THEN iffD2*]] *assms*
apply (*subst filterlim-sequentially-Suc* [*symmetric*])
apply (*simp add*: *lim-sequentially dist-norm*)
apply (*simp add*: *Ln-Reals-eq norm-powr-real-powr norm-divide*)
done

51.18 Relation between Square Root and exp/ln, hence its derivative

lemma *csqrt-exp-Ln*:
assumes *z ≠ 0*
shows *csqrt z = exp(Ln(z) / 2)*
proof −
have $(exp\ (Ln\ z\ /\ 2))^2 = (exp\ (Ln\ z))$
by (*metis exp-double nonzero-mult-divide-cancel-left times-divide-eq-right zero-neq-numeral*)
also have *... = z*
using *assms exp-Ln* **by** *blast*
finally have $csqrt\ z = csqrt\ ((exp\ (Ln\ z\ /\ 2))^2)$
by *simp*
also have *... = exp (Ln z / 2)*
apply (*subst csqrt-square*)
using *cos-gt-zero-pi* [*of* (*Im* (*Ln z*) / 2)] *Im-Ln-le-pi mpi-less-Im-Ln assms*
apply (*auto simp*: *Re-exp Im-exp zero-less-mult-iff zero-le-mult-iff*, *fastforce+*)
done
finally show *?thesis* **using** *assms csqrt-square*
by *simp*
qed

lemma *csqrt-inverse*:
assumes $z \notin \mathbb{R}_{\leq 0}$
shows *csqrt (inverse z) = inverse (csqrt z)*
proof (*cases z=0*, *simp*)
assume *z ≠ 0*

then show *?thesis*
using *assms csqrt-exp-Ln Ln-inverse exp-minus*
by (*simp add*: *csqrt-exp-Ln Ln-inverse exp-minus*)
qed

lemma *cnj-csqrt*:
assumes $z \notin \mathbb{R}_{\leq 0}$
shows *cnj*(*csqrt z*) = *csqrt*(*cnj z*)
proof (*cases z=0*, *simp*)
assume $z \neq 0$
then show *?thesis*
by (*simp add*: *assms cnj-Ln csqrt-exp-Ln exp-cnj*)
qed

lemma *has-field-derivative-csqrt*:
assumes $z \notin \mathbb{R}_{\leq 0}$
shows (*csqrt has-field-derivative inverse*(*2* ∗ *csqrt z*)) (*at z*)
proof −
have *z*: $z \neq 0$
using *assms* **by** *auto*
then have ∗: *inverse z* = *inverse* (*2*∗*z*) ∗ *2*
by (*simp add*: *divide-simps*)
have [*simp*]: *exp* (*Ln z* / *2*) ∗ *inverse z* = *inverse* (*csqrt z*)
by (*simp add*: *z field-simps csqrt-exp-Ln* [*symmetric*]) (*metis power2-csqrt power2-eq-square*)
have *Im z* = *0* ⟹ *0* < *Re z*
using *assms complex-nonpos-Reals-iff not-less* **by** *blast*
with *z* **have** ((λ*z*. *exp* (*Ln z* / *2*)) *has-field-derivative inverse* (*2* ∗ *csqrt z*)) (*at z*)
by (*force intro*: *derivative-eq-intros* ∗ *simp add*: *assms*)
then show *?thesis*
apply (*rule DERIV-transform-at*[**where** *d* = *norm z*])
apply (*intro z derivative-eq-intros* | *simp add*: *assms*)+
using *z*
apply (*metis csqrt-exp-Ln dist-0-norm less-irrefl*)
done
qed

lemma *complex-differentiable-at-csqrt*:
$z \notin \mathbb{R}_{\leq 0}$ ⟹ *csqrt complex-differentiable at z*
using *complex-differentiable-def has-field-derivative-csqrt* **by** *blast*

lemma *complex-differentiable-within-csqrt*:
$z \notin \mathbb{R}_{\leq 0}$ ⟹ *csqrt complex-differentiable* (*at z within s*)
using *complex-differentiable-at-csqrt complex-differentiable-within-subset* **by** *blast*

lemma *continuous-at-csqrt*:
$z \notin \mathbb{R}_{\leq 0}$ ⟹ *continuous* (*at z*) *csqrt*
by (*simp add*: *complex-differentiable-within-csqrt complex-differentiable-imp-continuous-at*)

corollary *isCont-csqrt'* [*simp*]:
  ⟦*isCont f z*; *f z* ∉ ℝ≤0⟧ ⟹ *isCont* (λ*x*. *csqrt* (*f x*)) *z*
 by (*blast intro*: *isCont-o2* [*OF* - *continuous-at-csqrt*])

lemma *continuous-within-csqrt*:
  z ∉ ℝ≤0 ⟹ *continuous* (*at z within s*) *csqrt*
 by (*simp add*: *complex-differentiable-imp-continuous-at complex-differentiable-within-csqrt*)

lemma *continuous-on-csqrt* [*continuous-intros*]:
  (⋀*z*. *z* ∈ *s* ⟹ *z* ∉ ℝ≤0) ⟹ *continuous-on s csqrt*
 by (*simp add*: *continuous-at-imp-continuous-on continuous-within-csqrt*)

lemma *holomorphic-on-csqrt*:
  (⋀*z*. *z* ∈ *s* ⟹ *z* ∉ ℝ≤0) ⟹ *csqrt holomorphic-on s*
 by (*simp add*: *complex-differentiable-within-csqrt holomorphic-on-def*)

lemma *continuous-within-closed-nontrivial*:
  closed s ⟹ *a* ∉ *s* ==> *continuous* (*at a within s*) *f*
 using *open-Compl*
 by (*force simp add*: *continuous-def eventually-at-topological filterlim-iff open-Collect-neg*)

lemma *continuous-within-csqrt-posreal*:
  continuous (*at z within* (ℝ ∩ {*w*. *0* ≤ *Re*(*w*)})) *csqrt*
proof (*cases z* ∈ ℝ≤0)
 case *True*
 then have *Im z* = *0 Re z* < *0* ∨ *z* = *0*
  using *cnj.code complex-cnj-zero-iff* **by** (*auto simp*: *complex-nonpos-Reals-iff*)
fastforce
 then show *?thesis*
  apply (*auto simp*: *continuous-within-closed-nontrivial* [*OF closed-Real-halfspace-Re-ge*])
  apply (*auto simp*: *continuous-within-eps-delta norm-conv-dist* [*symmetric*])
  apply (*rule-tac x*=*e*^*2* **in** *exI*)
  apply (*auto simp*: *Reals-def*)
  by (*metis linear not-less real-sqrt-less-iff real-sqrt-pow2-iff real-sqrt-power*)
next
 case *False*
  then show *?thesis* **by** (*blast intro*: *continuous-within-csqrt*)
qed

51.19 Complex arctangent

The branch cut gives standard bounds in the real case.

definition *Arctan* :: *complex* ⇒ *complex* **where**
  Arctan ≡ λ*z*. (i/*2*) ∗ *Ln*((*1* − i∗*z*) / (*1* + i∗*z*))

lemma *Arctan-def-moebius*: *Arctan z* = i/*2* ∗ *Ln* (*moebius* (−i) *1* i *1 z*)
 by (*simp add*: *Arctan-def moebius-def add-ac*)

lemma *Ln-conv-Arctan*:
assumes $z \neq -1$
shows $Ln\ z = -2*\mathrm{i} * Arctan\ (moebius\ 1\ (-1)\ (-\mathrm{i})\ (-\mathrm{i})\ z)$
proof −
have $Arctan\ (moebius\ 1\ (-1)\ (-\mathrm{i})\ (-\mathrm{i})\ z) =$
$\mathrm{i}/2 * Ln\ (moebius\ (-\mathrm{i})\ 1\ \mathrm{i}\ 1\ (moebius\ 1\ (-1)\ (-\mathrm{i})\ (-\mathrm{i})\ z))$
by (*simp add*: *Arctan-def-moebius*)
also from *assms* **have** $\mathrm{i} * z \neq \mathrm{i} * (-1)$ **by** (*subst mult-left-cancel*) *simp*
hence $\mathrm{i} * z - -\mathrm{i} \neq 0$ **by** (*simp add*: *eq-neg-iff-add-eq-0*)
from *moebius-inverse'*[*OF - this*, *of 1 1*]
have $moebius\ (-\mathrm{i})\ 1\ \mathrm{i}\ 1\ (moebius\ 1\ (-1)\ (-\mathrm{i})\ (-\mathrm{i})\ z) = z$ **by** *simp*
finally show *?thesis* **by** (*simp add*: *field-simps*)
qed

lemma *Arctan-0* [*simp*]: $Arctan\ 0 = 0$
by (*simp add*: *Arctan-def*)

lemma *Im-complex-div-lemma*: $Im((1 - \mathrm{i}*z) / (1 + \mathrm{i}*z)) = 0 \longleftrightarrow Re\ z = 0$
by (*auto simp*: *Im-complex-div-eq-0 algebra-simps*)

lemma *Re-complex-div-lemma*: $0 < Re((1 - \mathrm{i}*z) / (1 + \mathrm{i}*z)) \longleftrightarrow norm\ z < 1$
by (*simp add*: *Re-complex-div-gt-0 algebra-simps cmod-def power2-eq-square*)

lemma *tan-Arctan*:
assumes $z^2 \neq -1$
shows [*simp*]:$tan(Arctan\ z) = z$
proof −
have $1 + \mathrm{i}*z \neq 0$
by (*metis assms complex-i-mult-minus i-squared minus-unique power2-eq-square power2-minus*)
moreover
have $1 - \mathrm{i}*z \neq 0$
by (*metis assms complex-i-mult-minus i-squared power2-eq-square power2-minus right-minus-eq*)
ultimately
show *?thesis*
by (*simp add*: *Arctan-def tan-def sin-exp-eq cos-exp-eq exp-minus csqrt-exp-Ln* [*symmetric*]
divide-simps power2-eq-square [*symmetric*])
qed

lemma *Arctan-tan* [*simp*]:
assumes $|Re\ z| < pi/2$
shows $Arctan(tan\ z) = z$
proof −
have *ge-pi2*: $\bigwedge n::int.\ |of\text{-}int\ (2*n + 1) * pi/2| \geq pi/2$
by (*case-tac n rule*: *int-cases*) (*auto simp*: *abs-mult*)
have $exp\ (\mathrm{i}*z)*exp\ (\mathrm{i}*z) = -1 \longleftrightarrow exp\ (2*\mathrm{i}*z) = -1$
by (*metis distrib-right exp-add mult-2*)

```
  also have ... ⟷ exp (2*i*z) = exp (i*pi)
    using cis-conv-exp cis-pi by auto
  also have ... ⟷ exp (2*i*z - i*pi) = 1
   by (metis (no-types) diff-add-cancel diff-minus-eq-add exp-add exp-minus-inverse
mult.commute)
  also have ... ⟷ Re(i*2*z - i*pi) = 0 ∧ (∃ n::int. Im(i*2*z - i*pi) = of-int
(2 * n) * pi)
    by (simp add: exp-eq-1)
  also have ... ⟷ Im z = 0 ∧ (∃ n::int. 2 * Re z = of-int (2*n + 1) * pi)
    by (simp add: algebra-simps)
  also have ... ⟷ False
    using assms ge-pi2
    apply (auto simp: algebra-simps)
    by (metis abs-mult-pos not-less of-nat-less-0-iff of-nat-numeral)
  finally have *: exp (i*z)*exp (i*z) + 1 ≠ 0
    by (auto simp: add.commute minus-unique)
  show ?thesis
    using assms *
   apply (simp add: Arctan-def tan-def sin-exp-eq cos-exp-eq exp-minus divide-simps
                    ii-times-eq-iff power2-eq-square [symmetric])
    apply (rule Ln-unique)
    apply (auto simp: divide-simps exp-minus)
    apply (simp add: algebra-simps exp-double [symmetric])
    done
qed

lemma
  assumes Re z = 0 ⟹ |Im z| < 1
  shows Re-Arctan-bounds: |Re(Arctan z)| < pi/2
    and has-field-derivative-Arctan: (Arctan has-field-derivative inverse(1 + z^2))
(at z)
proof −
  have nz0: 1 + i*z ≠ 0
    using assms
   by (metis abs-one complex-i-mult-minus diff-0-right diff-minus-eq-add ii.simps(1)
ii.simps(2)
            less-irrefl minus-diff-eq mult.right-neutral right-minus-eq)
  have z ≠ −i using assms
    by auto
  then have zz: 1 + z * z ≠ 0
   by (metis abs-one assms i-squared ii.simps less-irrefl minus-unique square-eq-iff)
  have nz1: 1 − i*z ≠ 0
    using assms by (force simp add: ii-times-eq-iff)
  have nz2: inverse (1 + i*z) ≠ 0
    using assms
   by (metis Im-complex-div-lemma Re-complex-div-lemma cmod-eq-Im divide-complex-def
          less-irrefl mult-zero-right zero-complex.simps(1) zero-complex.simps(2))
  have nzi: ((1 − i*z) * inverse (1 + i*z)) ≠ 0
    using nz1 nz2 by auto
```

have *Im* ((*1* − i∗*z*) / (*1* + i∗*z*)) = *0* ⟹ *0* < *Re* ((*1* − i∗*z*) / (*1* + i∗*z*))
apply (*simp add*: *divide-complex-def*)
apply (*simp add*: *divide-simps split*: *split-if-asm*)
using *assms*
apply (*auto simp*: *algebra-simps abs-square-less-1* [*unfolded power2-eq-square*])
done
then have ∗: ((*1* − i∗*z*) / (*1* + i∗*z*)) ∉ $\mathbb{R}_{\leq 0}$
by (*auto simp add*: *complex-nonpos-Reals-iff*)
show |*Re*(*Arctan z*)| < *pi*/*2*
unfolding *Arctan-def divide-complex-def*
using *mpi-less-Im-Ln* [*OF nzi*]
apply (*auto simp*: *abs-if intro*!: *Im-Ln-less-pi* ∗ [*unfolded divide-complex-def*])
done
show (*Arctan has-field-derivative inverse*(*1* + *z*^{2})) (*at z*)
unfolding *Arctan-def scaleR-conv-of-real*
apply (*rule DERIV-cong*)
apply (*intro derivative-eq-intros* | *simp add*: *nz0* ∗)+
using *nz0 nz1 zz*
apply (*simp add*: *divide-simps power2-eq-square*)
apply (*auto simp*: *algebra-simps*)
done
qed

lemma *complex-differentiable-at-Arctan*: (*Re z* = *0* ⟹ |*Im z*| < *1*) ⟹ *Arctan complex-differentiable at z*
using *has-field-derivative-Arctan*
by (*auto simp*: *complex-differentiable-def*)

lemma *complex-differentiable-within-Arctan*:
(*Re z* = *0* ⟹ |*Im z*| < *1*) ⟹ *Arctan complex-differentiable* (*at z within s*)
using *complex-differentiable-at-Arctan complex-differentiable-at-within* **by** *blast*

declare *has-field-derivative-Arctan* [*derivative-intros*]
declare *has-field-derivative-Arctan* [*THEN DERIV-chain2*, *derivative-intros*]

lemma *continuous-at-Arctan*:
(*Re z* = *0* ⟹ |*Im z*| < *1*) ⟹ *continuous* (*at z*) *Arctan*
by (*simp add*: *complex-differentiable-imp-continuous-at complex-differentiable-within-Arctan*)

lemma *continuous-within-Arctan*:
(*Re z* = *0* ⟹ |*Im z*| < *1*) ⟹ *continuous* (*at z within s*) *Arctan*
using *continuous-at-Arctan continuous-at-imp-continuous-within* **by** *blast*

lemma *continuous-on-Arctan* [*continuous-intros*]:
(⋀*z*. *z* ∈ *s* ⟹ *Re z* = *0* ⟹ |*Im z*| < *1*) ⟹ *continuous-on s Arctan*
by (*auto simp*: *continuous-at-imp-continuous-on continuous-within-Arctan*)

lemma *holomorphic-on-Arctan*:
(⋀*z*. *z* ∈ *s* ⟹ *Re z* = *0* ⟹ |*Im z*| < *1*) ⟹ *Arctan holomorphic-on s*

by (*simp add*: *complex-differentiable-within-Arctan holomorphic-on-def*)

lemma *Arctan-series*:
assumes *z*: *norm* (*z* :: *complex*) < *1*
defines *g* ≡ λ*n*. *if odd n then* −i∗i ^ *n* / *n else 0*
defines *h* ≡ λ*z n*. (−*1*) ^ *n* / *of-nat* (*2*∗*n*+*1*) ∗ (*z*::*complex*) ^ (*2*∗*n*+*1*)
shows (λ*n*. *g n* ∗ *z* ^ *n*) *sums Arctan z*
and *h z sums Arctan z*
proof −
def *G* ≡ λ*z*. (∑ *n*. *g n* ∗ *z* ^ *n*)
have *summable*: *summable* (λ*n*. *g n* ∗ *u* ^ *n*) **if** *norm u* < *1* **for** *u*
proof (*cases u* = *0*)
assume *u*: *u* ≠ *0*
have (λ*n*. *ereal* (*norm* (*h u n*) / *norm* (*h u* (*Suc n*)))) = (λ*n*. *ereal* (*inverse* (*norm u*) ^ *2*) ∗
ereal ((*2* + *inverse* (*real* (*Suc n*))) / (*2* − *inverse* (*real* (*Suc n*)))))
proof
fix *n*
have *ereal* (*norm* (*h u n*) / *norm* (*h u* (*Suc n*))) =
ereal (*inverse* (*norm u*) ^ *2*) ∗ *ereal* ((*of-nat* (*2*∗*Suc n*+*1*) / *of-nat* (*Suc n*)) /
(*of-nat* (*2*∗*Suc n*−*1*) / *of-nat* (*Suc n*)))
by (*simp add*: *h-def norm-mult norm-power norm-divide divide-simps*
power2-eq-square eval-nat-numeral del: *of-nat-add of-nat-Suc*)
also have *of-nat* (*2*∗*Suc n*+*1*) / *of-nat* (*Suc n*) = (*2*::*real*) + *inverse* (*real* (*Suc n*))
by (*auto simp*: *divide-simps simp del*: *of-nat-Suc*) *simp-all?*
also have *of-nat* (*2*∗*Suc n*−*1*) / *of-nat* (*Suc n*) = (*2*::*real*) − *inverse* (*real* (*Suc n*))
by (*auto simp*: *divide-simps simp del*: *of-nat-Suc*) *simp-all?*
finally show *ereal* (*norm* (*h u n*) / *norm* (*h u* (*Suc n*))) = *ereal* (*inverse* (*norm u*) ^ *2*) ∗
ereal ((*2* + *inverse* (*real* (*Suc n*))) / (*2* − *inverse* (*real* (*Suc n*)))) **.**
qed
also have ... ⟶ *ereal* (*inverse* (*norm u*) ^ *2*) ∗ *ereal* ((*2* + *0*) / (*2* − *0*))
by (*intro tendsto-intros LIMSEQ-inverse-real-of-nat*) *simp-all*
finally have *liminf* (λ*n*. *ereal* (*cmod* (*h u n*) / *cmod* (*h u* (*Suc n*)))) = *inverse* (*norm u*) ^ *2*
by (*intro lim-imp-Liminf*) *simp-all*
moreover from *power-strict-mono*[*OF that*, *of 2*] *u* **have** *inverse* (*norm u*) ^ *2* > *1*
by (*simp add*: *divide-simps*)
ultimately have *A*: *liminf* (λ*n*. *ereal* (*cmod* (*h u n*) / *cmod* (*h u* (*Suc n*)))) > *1* **by** *simp*
from *u* **have** *summable* (*h u*)
by (*intro summable-norm-cancel*[*OF ratio-test-convergence*[*OF - A*]])
(*auto simp*: *h-def norm-divide norm-mult norm-power simp del*: *of-nat-Suc*
intro!: *mult-pos-pos divide-pos-pos always-eventually*)
thus *summable* (λ*n*. *g n* ∗ *u* ^ *n*)

by (*subst summable-mono-reindex*[*of* $\lambda n.\ 2*n+1$, *symmetric*])
(*auto simp*: *power-mult subseq-def g-def h-def elim*!: *oddE*)
qed (*simp add*: *h-def*)

have $\exists c.\ \forall u{\in}$*ball 0 1*. *Arctan* $u - G\ u = c$
proof (*rule has-field-derivative-zero-constant*)
fix u :: *complex* **assume** $u \in$ *ball 0 1*
hence u: *norm* $u < 1$ **by** (*simp add*: *dist-0-norm*)
def $K \equiv$ (*norm* u + *1*) / *2*
from u **and** *abs-Im-le-cmod*[*of* u] **have** *Im-u*: $|Im\ u| < 1$ **by** *linarith*
from u **have** K: $0 \le K$ *norm* $u < K$ $K < 1$ **by** (*simp-all add*: *K-def*)
hence (*G has-field-derivative* ($\sum n.$ *diffs g n* $*$ $u\hat{}n$)) (*at u*) **unfolding** *G-def*
by (*intro termdiffs-strong*[*of - of-real K*] *summable*) *simp-all*
also have ($\lambda n.$ *diffs g n* $*$ $u\hat{}n$) = ($\lambda n.$ *if even n then* (i$*u$)$\hat{}n$ *else 0*)
by (*intro ext*) (*simp-all del*: *of-nat-Suc add*: *g-def diffs-def power-mult-distrib*)
also have *suminf* ... = ($\sum n.$ ($-(u\hat{}2)$)$\hat{}n$)
by (*subst suminf-mono-reindex*[*of* $\lambda n.\ 2*n$, *symmetric*])
(*auto elim*!: *evenE simp*: *subseq-def power-mult power-mult-distrib*)
also from u **have** *norm* $u\hat{}2 < 1\hat{}2$ **by** (*intro power-strict-mono*) *simp-all*
hence ($\sum n.$ ($-(u\hat{}2)$)$\hat{}n$) = *inverse* (*1* + $u\hat{}2$)
by (*subst suminf-geometric*) (*simp-all add*: *norm-power inverse-eq-divide*)
finally have (*G has-field-derivative inverse* (*1* + u^2)) (*at u*) **.**
from *DERIV-diff*[*OF has-field-derivative-Arctan this*] *Im-u u*
show (($\lambda u.$ *Arctan u* $-$ *G u*) *has-field-derivative 0*) (*at u within ball 0 1*)
by (*simp-all add*: *dist-0-norm at-within-open*[*OF - open-ball*])
qed *simp-all*
then obtain c **where** c: $\bigwedge u.$ *norm* $u < 1 \Longrightarrow$ *Arctan* $u - G\ u = c$ **by** (*auto simp*: *dist-0-norm*)
from *this*[*of 0*] **have** $c = 0$ **by** (*simp add*: *G-def g-def powser-zero*)
with c z **have** *Arctan* $z = G\ z$ **by** *simp*
with *summable*[*OF z*] **show** ($\lambda n.$ *g n* $*$ $z\hat{}n$) *sums Arctan z* **unfolding** *G-def*
by (*simp add*: *sums-iff*)
thus *h z sums Arctan z* **by** (*subst* (*asm*) *sums-mono-reindex*[*of* $\lambda n.\ 2*n+1$, *symmetric*])
(*auto elim*!: *oddE simp*: *subseq-def power-mult g-def h-def*)
qed

A quickly-converging series for the logarithm, based on the arctangent.

lemma *ln-series-quadratic*:
assumes x: $x > (0::real)$
shows ($\lambda n.$ (*2*$*$((*x* $-$ *1*) / (*x* + *1*)) $\hat{}$ (*2*$*$*n*+*1*) / *of-nat* (*2*$*$*n*+*1*))) *sums ln x*
proof $-$
def $y \equiv$ *of-real* ((*x*$-$*1*)/(*x*+*1*)) :: *complex*
from x **have** x': *complex-of-real* $x \neq$ *of-real* (-1) **by** (*subst of-real-eq-iff*) *auto*
from x **have** $|x - 1| < |x + 1|$ **by** *linarith*
hence *norm* (*complex-of-real* (*x* $-$ *1*) / *complex-of-real* (*x* + *1*)) < *1*
by (*simp add*: *norm-divide del*: *of-real-add of-real-diff*)
hence *norm* (i $*$ *y*) < *1* **unfolding** *y-def* **by** (*subst norm-mult*) *simp*
hence ($\lambda n.$ (-2*i) $*$ ((-1) $\hat{}n$ / *of-nat* (*2*$*$*n*+*1*) $*$ (i$*y$) $\hat{}$(*2*$*$*n*+*1*))) *sums* ((-2*i)

$*$ *Arctan* (i$*y$))
by (*intro Arctan-series sums-mult*) *simp-all*
also have ($\lambda n.$ (-2*i) $*$ ((-1) ^ *n* / *of-nat* (2$*n+1) $*$ (i$*y$) ^ (2$*n+1))) =
($\lambda n.$ (-2*i) $*$ ((-1) ^ *n* $*$ (i$*y$*($-y^2$) ^ *n*)/*of-nat* (2$*n+1)))
by (*intro ext*) (*simp-all add*: *power-mult power-mult-distrib*)
also have ... = ($\lambda n.$ 2$*y$*$ ((-1) $*$ ($-y^2$)) ^ *n*/*of-nat* (2$*n+1))
by (*intro ext*, *subst power-mult-distrib*) (*simp add*: *algebra-simps power-mult*)
also have ... = ($\lambda n.$ 2$*y$^$(2$*n+1) / *of-nat* (2$*n+1))
by (*subst power-add*, *subst power-mult*) (*simp add*: *mult-ac*)
also have ... = ($\lambda n.$ *of-real* (2$*((x$-1)/(x+1)) ^ (2$*n+1) / *of-nat* (2$*n+1)))
by (*intro ext*) (*simp add*: *y-def*)
also have i $*$ *y* = (*of-real x* $-$ *1*) / ($-$i $*$ (*of-real x* + *1*))
by (*subst divide-divide-eq-left* [*symmetric*]) (*simp add*: *y-def*)
also have ... = *moebius 1* (-1) ($-$i) ($-$i) (*of-real x*) **by** (*simp add*: *moebius-def algebra-simps*)
also from *x′* **have** -2*i$*Arctan ... = *Ln* (*of-real x*) **by** (*intro Ln-conv-Arctan* [*symmetric*]) *simp-all*
also from *x* **have** ... = *ln x* **by** (*rule Ln-of-real*)
finally show *?thesis* **by** (*subst* (*asm*) *sums-of-real-iff*)
qed

51.20 Real arctangent

lemma *norm-exp-ii-times* [*simp*]: *norm* (*exp*(i $*$ *of-real y*)) = *1*
by *simp*

lemma *norm-exp-imaginary*: *norm*(*exp z*) = *1* $\Longrightarrow$ *Re z* = *0*
by (*simp add*: *complex-norm-eq-1-exp*)

lemma *Im-Arctan-of-real* [*simp*]: *Im* (*Arctan* (*of-real x*)) = *0*
unfolding *Arctan-def divide-complex-def*
apply (*simp add*: *complex-eq-iff*)
apply (*rule norm-exp-imaginary*)
apply (*subst exp-Ln*, *auto*)
apply (*simp-all add*: *cmod-def complex-eq-iff*)
apply (*auto simp*: *divide-simps*)
apply (*metis power-one sum-power2-eq-zero-iff zero-neq-one*, *algebra*)
done

lemma *arctan-eq-Re-Arctan*: *arctan x* = *Re* (*Arctan* (*of-real x*))
proof (*rule arctan-unique*)
show $-$ (*pi* / *2*) < *Re* (*Arctan* (*complex-of-real x*))
apply (*simp add*: *Arctan-def*)
apply (*rule Im-Ln-less-pi*)
apply (*auto simp*: *Im-complex-div-lemma complex-nonpos-Reals-iff*)
done
next
have $*$: (*1* $-$ i$*x) / (*1* + i$*x) $\neq$ *0*
by (*simp add*: *divide-simps*) (*simp add*: *complex-eq-iff*)

show *Re* (*Arctan* (*complex-of-real x*)) < *pi* / *2*
using *mpi-less-Im-Ln* [*OF* ∗]
by (*simp add*: *Arctan-def*)
next
have *tan* (*Re* (*Arctan* (*of-real x*))) = *Re* (*tan* (*Arctan* (*of-real x*)))
apply (*auto simp*: *tan-def Complex.Re-divide Re-sin Re-cos Im-sin Im-cos*)
apply (*simp add*: *field-simps*)
by (*simp add*: *power2-eq-square*)
also have ... = *x*
apply (*subst tan-Arctan*, *auto*)
by (*metis diff-0-right minus-diff-eq mult-zero-left not-le of-real-1 of-real-eq-iff of-real-minus of-real-power power2-eq-square real-minus-mult-self-le zero-less-one*)
finally show *tan* (*Re* (*Arctan* (*complex-of-real x*))) = *x* .
qed

lemma *Arctan-of-real*: *Arctan* (*of-real x*) = *of-real* (*arctan x*)
unfolding *arctan-eq-Re-Arctan divide-complex-def*
by (*simp add*: *complex-eq-iff*)

lemma *Arctan-in-Reals* [*simp*]: *z* ∈ ℝ ⟹ *Arctan z* ∈ ℝ
by (*metis Reals-cases Reals-of-real Arctan-of-real*)

declare *arctan-one* [*simp*]

lemma *arctan-less-pi4-pos*: *x* < *1* ⟹ *arctan x* < *pi*/*4*
by (*metis arctan-less-iff arctan-one*)

lemma *arctan-less-pi4-neg*: −*1* < *x* ⟹ −(*pi*/*4*) < *arctan x*
by (*metis arctan-less-iff arctan-minus arctan-one*)

lemma *arctan-less-pi4*: |*x*| < *1* ⟹ |*arctan x*| < *pi*/*4*
by (*metis abs-less-iff arctan-less-pi4-pos arctan-minus*)

lemma *arctan-le-pi4*: |*x*| ≤ *1* ⟹ |*arctan x*| ≤ *pi*/*4*
by (*metis abs-le-iff arctan-le-iff arctan-minus arctan-one*)

lemma *abs-arctan*: |*arctan x*| = *arctan* |*x*|
by (*simp add*: *abs-if arctan-minus*)

lemma *arctan-add-raw*:
assumes |*arctan x* + *arctan y*| < *pi*/*2*
shows *arctan x* + *arctan y* = *arctan*((*x* + *y*) / (*1* − *x* ∗ *y*))
proof (*rule arctan-unique* [*symmetric*])
show *12*: − (*pi* / *2*) < *arctan x* + *arctan y arctan x* + *arctan y* < *pi* / *2*
using *assms* **by** *linarith*+
show *tan* (*arctan x* + *arctan y*) = (*x* + *y*) / (*1* − *x* ∗ *y*)
using *cos-gt-zero-pi* [*OF 12*]
by (*simp add*: *arctan tan-add*)
qed

lemma *arctan-inverse*:
assumes *0 < x*
shows *arctan(inverse x) = pi/2 − arctan x*
proof −
have *arctan(inverse x) = arctan(inverse(tan(arctan x)))*
by (*simp add*: *arctan*)
also have ... = *arctan* (*tan* (*pi / 2 − arctan x*))
by (*simp add*: *tan-cot*)
also have ... = *pi/2 − arctan x*
proof −
have *0 < pi − arctan x*
using *arctan-ubound* [*of x*] *pi-gt-zero* **by** *linarith*
with *assms* **show** *?thesis*
by (*simp add*: *Transcendental.arctan-tan*)
qed
finally show *?thesis* **.**
qed

lemma *arctan-add-small*:
assumes |*x ∗ y*| < *1*
shows (*arctan x + arctan y = arctan*((*x + y*) / (*1 − x ∗ y*)))
proof (*cases x = 0 ∨ y = 0*)
case *True* **then show** *?thesis*
by *auto*
next
case *False*
then have ∗: |*arctan x*| < *pi / 2 −* |*arctan y*| **using** *assms*
apply (*auto simp add*: *abs-arctan arctan-inverse* [*symmetric*] *arctan-less-iff*)
apply (*simp add*: *divide-simps abs-mult*)
done
show *?thesis*
apply (*rule arctan-add-raw*)
using ∗ **by** *linarith*
qed

lemma *abs-arctan-le*:
fixes *x*::*real* **shows** |*arctan x*| ≤ |*x*|
proof −
{ fix *w*::*complex* **and** *z*::*complex*
assume ∗: *w* ∈ ℝ *z* ∈ ℝ
have *cmod* (*Arctan w − Arctan z*) ≤ *1 ∗ cmod* (*w−z*)
apply (*rule complex-differentiable-bound* [*OF convex-Reals*, *of Arctan - 1*])
apply (*rule has-field-derivative-at-within* [*OF has-field-derivative-Arctan*])
apply (*force simp add*: *Reals-def*)
apply (*simp add*: *norm-divide divide-simps in-Reals-norm complex-is-Real-iff power2-eq-square*)
using ∗ **by** *auto*
}

then have *cmod* (*Arctan* (*of-real x*) − *Arctan 0*) ≤ *1* ∗ *cmod* (*of-real x* −*0*)
using *Reals-0 Reals-of-real* **by** *blast*
then show *?thesis*
by (*simp add*: *Arctan-of-real*)
qed

lemma *arctan-le-self*: *0* ≤ *x* ⟹ *arctan x* ≤ *x*
by (*metis abs-arctan-le abs-of-nonneg zero-le-arctan-iff*)

lemma *abs-tan-ge*: |*x*| < *pi*/*2* ⟹ |*x*| ≤ |*tan x*|
by (*metis abs-arctan-le abs-less-iff arctan-tan minus-less-iff*)

51.21 Inverse Sine

definition *Arcsin* :: *complex* ⇒ *complex* **where**
Arcsin ≡ λ*z*. −i ∗ *Ln*(i ∗ *z* + *csqrt*(*1* − *z*^{2}))

lemma *Arcsin-body-lemma*: i ∗ *z* + *csqrt*(*1* − *z*^{2}) ≠ *0*
using *power2-csqrt* [*of 1* − *z*^{2}]
apply *auto*
by (*metis complex-i-mult-minus diff-add-cancel diff-minus-eq-add diff-self mult.assoc mult.left-commute numeral-One power2-csqrt power2-eq-square zero-neq-numeral*)

lemma *Arcsin-range-lemma*: |*Re z*| < *1* ⟹ *0* < *Re*(i ∗ *z* + *csqrt*(*1* − *z*^{2}))
using *Complex.cmod-power2* [*of z*, *symmetric*]
by (*simp add*: *real-less-rsqrt algebra-simps Re-power2 cmod-square-less-1-plus*)

lemma *Re-Arcsin*: *Re*(*Arcsin z*) = *Im* (*Ln* (i ∗ *z* + *csqrt*(*1* − *z*^{2})))
by (*simp add*: *Arcsin-def*)

lemma *Im-Arcsin*: *Im*(*Arcsin z*) = − *ln* (*cmod* (i ∗ *z* + *csqrt* (*1* − *z*^{2})))
by (*simp add*: *Arcsin-def Arcsin-body-lemma*)

lemma *one-minus-z2-notin-nonpos-Reals*:
assumes (*Im z* = *0* ⟹ |*Re z*| < *1*)
shows *1* − *z*^{2} ∉ $\mathbb{R}_{\leq 0}$
using *assms*
apply (*auto simp*: *complex-nonpos-Reals-iff Re-power2 Im-power2*)
using *power2-less-0* [*of Im z*] **apply** *force*
using *abs-square-less-1 not-le* **by** *blast*

lemma *isCont-Arcsin-lemma*:
assumes *le0*: *Re* (i ∗ *z* + *csqrt* (*1* − *z*^{2})) ≤ *0* **and** (*Im z* = *0* ⟹ |*Re z*| < *1*)
shows *False*
proof (*cases Im z* = *0*)
case *True*
then show *?thesis*
using *assms* **by** (*fastforce simp*: *cmod-def abs-square-less-1* [*symmetric*])
next

case *False*
have *neq*: $(cmod\ z)^2 \neq 1 + cmod\ (1 - z^2)$
proof (*clarsimp simp add*: *cmod-def*)
assume $(Re\ z)^2 + (Im\ z)^2 = 1 + sqrt\ ((1 - Re\ (z^2))^2 + (Im\ (z^2))^2)$
then have $((Re\ z)^2 + (Im\ z)^2 - 1)^2 = ((1 - Re\ (z^2))^2 + (Im\ (z^2))^2)$
by *simp*
then show *False* **using** *False*
by (*simp add*: *power2-eq-square algebra-simps*)
qed
moreover have *2*: $(Im\ z)^2 = (1 + ((Im\ z)^2 + cmod\ (1 - z^2)) - (Re\ z)^2)\ /\ 2$
using *le0*
apply *simp*
apply (*drule sqrt-le-D*)
using *cmod-power2* [*of z*] *norm-triangle-ineq2* [*of z^2 1*]
apply (*simp add*: *norm-power Re-power2 norm-minus-commute* [*of 1*])
done
ultimately show *False*
by (*simp add*: *Re-power2 Im-power2 cmod-power2*)
qed

lemma *isCont-Arcsin*:
assumes $(Im\ z = 0 \Longrightarrow |Re\ z| < 1)$
shows *isCont Arcsin z*
proof −
have $*$: $\mathrm{i} * z + csqrt\ (1 - z^2) \notin \mathbb{R}_{\leq 0}$
by (*metis isCont-Arcsin-lemma assms complex-nonpos-Reals-iff*)
show *?thesis*
using *assms*
apply (*simp add*: *Arcsin-def*)
apply (*rule isCont-Ln' isCont-csqrt' continuous-intros*)+
apply (*simp add*: *one-minus-z2-notin-nonpos-Reals assms*)
apply (*rule* $*$)
done
qed

lemma *isCont-Arcsin'* [*simp*]:
shows $isCont\ f\ z \Longrightarrow (Im\ (f\ z) = 0 \Longrightarrow |Re\ (f\ z)| < 1) \Longrightarrow isCont\ (\lambda x.\ Arcsin\ (f\ x))\ z$
by (*blast intro*: *isCont-o2* [*OF - isCont-Arcsin*])

lemma *sin-Arcsin* [*simp*]: $sin(Arcsin\ z) = z$
proof −
have $\mathrm{i}*z*2 + csqrt\ (1 - z^2)*2 = 0 \longleftrightarrow (\mathrm{i}*z)*2 + csqrt\ (1 - z^2)*2 = 0$
by (*simp add*: *algebra-simps*) — Cancelling a factor of 2
moreover have $\ldots \longleftrightarrow (\mathrm{i}*z) + csqrt\ (1 - z^2) = 0$
by (*metis Arcsin-body-lemma distrib-right no-zero-divisors zero-neq-numeral*)
ultimately show *?thesis*
apply (*simp add*: *sin-exp-eq Arcsin-def Arcsin-body-lemma exp-minus divide-simps*)
apply (*simp add*: *algebra-simps*)

apply (*simp add*: *power2-eq-square* [*symmetric*] *algebra-simps*)
done
qed

lemma *Re-eq-pihalf-lemma*:
$|Re\ z| = pi/2 \Longrightarrow Im\ z = 0 \Longrightarrow$
$Re\ ((exp\ (\mathrm{i}*z) + inverse\ (exp\ (\mathrm{i}*z)))\ /\ 2) = 0 \wedge 0 \leq Im\ ((exp\ (\mathrm{i}*z) + inverse\ (exp\ (\mathrm{i}*z)))\ /\ 2)$
apply (*simp add*: *cos-ii-times* [*symmetric*] *Re-cos Im-cos abs-if del*: *eq-divide-eq-numeral1*)
by (*metis cos-minus cos-pi-half*)

lemma *Re-less-pihalf-lemma*:
assumes $|Re\ z| < pi\ /\ 2$
shows $0 < Re\ ((exp\ (\mathrm{i}*z) + inverse\ (exp\ (\mathrm{i}*z)))\ /\ 2)$
proof −
have $0 < cos\ (Re\ z)$ **using** *assms*
using *cos-gt-zero-pi* **by** *auto*
then show *?thesis*
by (*simp add*: *cos-ii-times* [*symmetric*] *Re-cos Im-cos add-pos-pos*)
qed

lemma *Arcsin-sin*:
assumes $|Re\ z| < pi/2 \vee (|Re\ z| = pi/2 \wedge Im\ z = 0)$
shows $Arcsin(sin\ z) = z$
proof −
have $Arcsin(sin\ z) = -\ (\mathrm{i} * Ln\ (csqrt\ (1 - (\mathrm{i} * (exp\ (\mathrm{i}*z) - inverse\ (exp\ (\mathrm{i}*z))))^2\ /\ 4) - (inverse\ (exp\ (\mathrm{i}*z)) - exp\ (\mathrm{i}*z))\ /\ 2))$
by (*simp add*: *sin-exp-eq Arcsin-def exp-minus power-divide*)
also have $... = -\ (\mathrm{i} * Ln\ (csqrt\ (((exp\ (\mathrm{i}*z) + inverse\ (exp\ (\mathrm{i}*z)))/2)^2) - (inverse\ (exp\ (\mathrm{i}*z)) - exp\ (\mathrm{i}*z))\ /\ 2))$
by (*simp add*: *field-simps power2-eq-square*)
also have $... = -\ (\mathrm{i} * Ln\ (((exp\ (\mathrm{i}*z) + inverse\ (exp\ (\mathrm{i}*z)))/2) - (inverse\ (exp\ (\mathrm{i}*z)) - exp\ (\mathrm{i}*z))\ /\ 2))$
apply (*subst csqrt-square*)
using *assms Re-eq-pihalf-lemma Re-less-pihalf-lemma*
apply *auto*
done
also have $... = -\ (\mathrm{i} * Ln\ (exp\ (\mathrm{i}*z)))$
by (*simp add*: *field-simps power2-eq-square*)
also have $... = z$
apply (*subst Complex-Transcendental.Ln-exp*)
using *assms*
apply (*auto simp*: *abs-if simp del*: *eq-divide-eq-numeral1 split*: *split-if-asm*)
done
finally show *?thesis* **.**
qed

lemma *Arcsin-unique*:
$[\![sin\ z = w;\ |Re\ z| < pi/2 \vee (|Re\ z| = pi/2 \wedge Im\ z = 0)]\!] \Longrightarrow Arcsin\ w = z$

by (*metis Arcsin-sin*)

lemma *Arcsin-0* [*simp*]: *Arcsin 0 = 0*
by (*metis Arcsin-sin norm-zero pi-half-gt-zero real-norm-def sin-zero zero-complex.simps*(*1*))

lemma *Arcsin-1* [*simp*]: *Arcsin 1 = pi/2*
by (*metis Arcsin-sin Im-complex-of-real Re-complex-of-real numeral-One of-real-numeral pi-half-ge-zero real-sqrt-abs real-sqrt-pow2 real-sqrt-power sin-of-real sin-pi-half*)

lemma *Arcsin-minus-1* [*simp*]: *Arcsin*(*−1*) *= − (pi/2*)
by (*metis Arcsin-1 Arcsin-sin Im-complex-of-real Re-complex-of-real abs-of-nonneg of-real-minus pi-half-ge-zero power2-minus real-sqrt-abs sin-Arcsin sin-minus*)

lemma *has-field-derivative-Arcsin*:
assumes (*Im z = 0 ⟹ |Re z| < 1*)
shows (*Arcsin has-field-derivative inverse*(*cos*(*Arcsin z*))) (*at z*)
proof −
have (*sin* (*Arcsin z*))^2 *≠ 1*
using *assms*
apply *atomize*
apply (*auto simp*: *complex-eq-iff Re-power2 Im-power2 abs-square-eq-1*)
apply (*metis abs-minus-cancel abs-one abs-power2 numeral-One numeral-neq-neg-one*)
by (*metis abs-minus-cancel abs-one abs-power2 one-neq-neg-one*)
then have *cos* (*Arcsin z*) *≠ 0*
by (*metis diff-0-right power-zero-numeral sin-squared-eq*)
then show *?thesis*
apply (*rule has-complex-derivative-inverse-basic* [*OF DERIV-sin*])
apply (*auto intro*: *isCont-Arcsin open-ball* [*of z 1*] *assms*)
done
qed

declare *has-field-derivative-Arcsin* [*derivative-intros*]
declare *has-field-derivative-Arcsin* [*THEN DERIV-chain2*, *derivative-intros*]

lemma *complex-differentiable-at-Arcsin*:
(*Im z = 0 ⟹ |Re z| < 1*) *⟹ Arcsin complex-differentiable at z*
using *complex-differentiable-def has-field-derivative-Arcsin* **by** *blast*

lemma *complex-differentiable-within-Arcsin*:
(*Im z = 0 ⟹ |Re z| < 1*) *⟹ Arcsin complex-differentiable* (*at z within s*)
using *complex-differentiable-at-Arcsin complex-differentiable-within-subset* **by** *blast*

lemma *continuous-within-Arcsin*:
(*Im z = 0 ⟹ |Re z| < 1*) *⟹ continuous* (*at z within s*) *Arcsin*
using *continuous-at-imp-continuous-within isCont-Arcsin* **by** *blast*

lemma *continuous-on-Arcsin* [*continuous-intros*]:
(⋀*z*. *z ∈ s ⟹ Im z = 0 ⟹ |Re z| < 1*) *⟹ continuous-on s Arcsin*
by (*simp add*: *continuous-at-imp-continuous-on*)

lemma *holomorphic-on-Arcsin*: $(\bigwedge z.\ z \in s \Longrightarrow Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow$ *Arcsin holomorphic-on s*
by (*simp add*: *complex-differentiable-within-Arcsin holomorphic-on-def*)

51.22 Inverse Cosine

definition *Arccos* :: *complex* ⇒ *complex* **where**
Arccos ≡ $\lambda z.\ -\mathrm{i} * Ln(z + \mathrm{i} * csqrt(1 - z^2))$

lemma *Arccos-range-lemma*: $|Re\ z| < 1 \Longrightarrow 0 < Im(z + \mathrm{i} * csqrt(1 - z^2))$
using *Arcsin-range-lemma* [*of* $-z$]
by *simp*

lemma *Arccos-body-lemma*: $z + \mathrm{i} * csqrt(1 - z^2) \neq 0$
using *Arcsin-body-lemma* [*of z*]
by (*metis complex-i-mult-minus diff-add-cancel minus-diff-eq minus-unique mult.assoc mult.left-commute*
power2-csqrt power2-eq-square zero-neq-one)

lemma *Re-Arccos*: $Re(Arccos\ z) = Im\ (Ln\ (z + \mathrm{i} * csqrt(1 - z^2)))$
by (*simp add*: *Arccos-def*)

lemma *Im-Arccos*: $Im(Arccos\ z) = -\ ln\ (cmod\ (z + \mathrm{i} * csqrt\ (1 - z^2)))$
by (*simp add*: *Arccos-def Arccos-body-lemma*)

A very tricky argument to find!

lemma *isCont-Arccos-lemma*:
assumes *eq0*: $Im\ (z + \mathrm{i} * csqrt\ (1 - z^2)) = 0$ **and** $(Im\ z = 0 \Longrightarrow |Re\ z| < 1)$
shows *False*
proof (*cases* $Im\ z = 0$)
case *True*
then show *?thesis*
using *assms* **by** (*fastforce simp add*: *cmod-def abs-square-less-1* [*symmetric*])
next
case *False*
have *Imz*: $Im\ z = -\ sqrt\ ((1 + ((Im\ z)^2 + cmod\ (1 - z^2)) - (Re\ z)^2)\ /\ 2)$
using *eq0 abs-Re-le-cmod* [*of* $1 - z^2$]
by (*simp add*: *Re-power2 algebra-simps*)
have $(cmod\ z)^2 - 1 \neq cmod\ (1 - z^2)$
proof (*clarsimp simp add*: *cmod-def*)
assume $(Re\ z)^2 + (Im\ z)^2 - 1 = sqrt\ ((1 - Re\ (z^2))^2 + (Im\ (z^2))^2)$
then have $((Re\ z)^2 + (Im\ z)^2 - 1)^2 = ((1 - Re\ (z^2))^2 + (Im\ (z^2))^2)$
by *simp*
then show *False* **using** *False*
by (*simp add*: *power2-eq-square algebra-simps*)
qed
moreover have $(Im\ z)^2 = ((1 + ((Im\ z)^2 + cmod\ (1 - z^2)) - (Re\ z)^2)\ /\ 2)$
apply (*subst Imz*)

using *abs-Re-le-cmod* [*of* $1-z^2$]
apply (*simp add*: *Re-power2*)
done
ultimately show *False*
by (*simp add*: *cmod-power2*)
qed

lemma *isCont-Arccos*:
assumes ($Im\ z = 0 \Longrightarrow |Re\ z| < 1$)
shows *isCont Arccos z*
proof −
have $z + \mathrm{i} * csqrt\ (1 - z^2) \notin \mathbb{R}_{\leq 0}$
by (*metis complex-nonpos-Reals-iff isCont-Arccos-lemma assms*)
with *assms* **show** *?thesis*
apply (*simp add*: *Arccos-def*)
apply (*rule isCont-Ln' isCont-csqrt' continuous-intros*)+
apply (*simp-all add*: *one-minus-z2-notin-nonpos-Reals assms*)
done
qed

lemma *isCont-Arccos'* [*simp*]:
shows $isCont\ f\ z \Longrightarrow (Im\ (f\ z) = 0 \Longrightarrow |Re\ (f\ z)| < 1) \Longrightarrow isCont\ (\lambda x.\ Arccos\ (f\ x))\ z$
by (*blast intro*: *isCont-o2* [*OF* - *isCont-Arccos*])

lemma *cos-Arccos* [*simp*]: $cos(Arccos\ z) = z$
proof −
have $z*2 + \mathrm{i} * (2 * csqrt\ (1 - z^2)) = 0 \longleftrightarrow z*2 + \mathrm{i} * csqrt\ (1 - z^2)*2 = 0$
by (*simp add*: *algebra-simps*) — Cancelling a factor of 2
moreover have $\ldots \longleftrightarrow z + \mathrm{i} * csqrt\ (1 - z^2) = 0$
by (*metis distrib-right mult-eq-0-iff zero-neq-numeral*)
ultimately show *?thesis*
apply (*simp add*: *cos-exp-eq Arccos-def Arccos-body-lemma exp-minus field-simps*)
apply (*simp add*: *power2-eq-square* [*symmetric*])
done
qed

lemma *Arccos-cos*:
assumes $0 < Re\ z\ \&\ Re\ z < pi \vee$
$Re\ z = 0\ \&\ 0 \leq Im\ z \vee$
$Re\ z = pi\ \&\ Im\ z \leq 0$
shows $Arccos(cos\ z) = z$
proof −
have $*$: $((\mathrm{i} - (exp\ (\mathrm{i} * z))^2 * \mathrm{i})\ /\ (2 * exp\ (\mathrm{i} * z))) = sin\ z$
by (*simp add*: *sin-exp-eq exp-minus field-simps power2-eq-square*)
have $1 - (exp\ (\mathrm{i} * z) + inverse\ (exp\ (\mathrm{i} * z)))^2\ /\ 4 = ((\mathrm{i} - (exp\ (\mathrm{i} * z))^2 * \mathrm{i})\ /\ (2 * exp\ (\mathrm{i} * z)))^2$
by (*simp add*: *field-simps power2-eq-square*)
then have $Arccos(cos\ z) = -\ (\mathrm{i} * Ln\ ((exp\ (\mathrm{i} * z) + inverse\ (exp\ (\mathrm{i} * z)))\ /\ 2$

```
+
                    i * csqrt (((i - (exp (i * z))^2 * i) / (2 * exp (i * z)))^2)))
   by (simp add: cos-exp-eq Arccos-def exp-minus power-divide)
  also have ... = - (i * Ln ((exp (i * z) + inverse (exp (i * z))) / 2 +
                      i * ((i - (exp (i * z))^2 * i) / (2 * exp (i * z)))))
   apply (subst csqrt-square)
   using assms Re-sin-pos [of z] Im-sin-nonneg [of z] Im-sin-nonneg2 [of z]
   apply (auto simp: * Re-sin Im-sin)
   done
  also have ... =  - (i * Ln (exp (i*z)))
   by (simp add: field-simps power2-eq-square)
  also have ... = z
   using assms
   apply (subst Complex-Transcendental.Ln-exp, auto)
   done
  finally show ?thesis .
qed

lemma Arccos-unique:
   ⟦cos z = w;
     0 < Re z ∧ Re z < pi ∨
     Re z = 0 ∧ 0 ≤ Im z ∨
     Re z = pi ∧ Im z ≤ 0⟧ ⟹ Arccos w = z
  using Arccos-cos by blast

lemma Arccos-0 [simp]: Arccos 0 = pi/2
  by (rule Arccos-unique) (auto simp: of-real-numeral)

lemma Arccos-1 [simp]: Arccos 1 = 0
  by (rule Arccos-unique) auto

lemma Arccos-minus1: Arccos(-1) = pi
  by (rule Arccos-unique) auto

lemma has-field-derivative-Arccos:
  assumes (Im z = 0 ⟹ |Re z| < 1)
    shows (Arccos has-field-derivative - inverse(sin(Arccos z))) (at z)
proof -
  have (cos (Arccos z))^2 ≠ 1
   using assms
   apply atomize
   apply (auto simp: complex-eq-iff Re-power2 Im-power2 abs-square-eq-1)
   apply (metis abs-minus-cancel abs-one abs-power2 numeral-One numeral-neq-neg-one)
   apply (metis left-minus less-irrefl power-one sum-power2-gt-zero-iff zero-neq-neg-one)
   done
  then have - sin (Arccos z) ≠ 0
   by (metis cos-squared-eq diff-0-right mult-zero-left neg-0-equal-iff-equal power2-eq-square)
  then have (Arccos has-field-derivative inverse(- sin(Arccos z))) (at z)
   apply (rule has-complex-derivative-inverse-basic [OF DERIV-cos])
```

apply (*auto intro*: *isCont-Arccos open-ball* [*of z 1*] *assms*)
done
then show *?thesis*
by *simp*
qed

declare *has-field-derivative-Arcsin* [*derivative-intros*]
declare *has-field-derivative-Arcsin* [*THEN DERIV-chain2*, *derivative-intros*]

lemma *complex-differentiable-at-Arccos*:
(*Im z = 0* $\Longrightarrow$ |*Re z*| < *1*) $\Longrightarrow$ *Arccos complex-differentiable at z*
using *complex-differentiable-def has-field-derivative-Arccos* **by** *blast*

lemma *complex-differentiable-within-Arccos*:
(*Im z = 0* $\Longrightarrow$ |*Re z*| < *1*) $\Longrightarrow$ *Arccos complex-differentiable* (*at z within s*)
using *complex-differentiable-at-Arccos complex-differentiable-within-subset* **by** *blast*

lemma *continuous-within-Arccos*:
(*Im z = 0* $\Longrightarrow$ |*Re z*| < *1*) $\Longrightarrow$ *continuous* (*at z within s*) *Arccos*
using *continuous-at-imp-continuous-within isCont-Arccos* **by** *blast*

lemma *continuous-on-Arccos* [*continuous-intros*]:
($\bigwedge$*z*. *z* $\in$ *s* $\Longrightarrow$ *Im z = 0* $\Longrightarrow$ |*Re z*| < *1*) $\Longrightarrow$ *continuous-on s Arccos*
by (*simp add*: *continuous-at-imp-continuous-on*)

lemma *holomorphic-on-Arccos*: ($\bigwedge$*z*. *z* $\in$ *s* $\Longrightarrow$ *Im z = 0* $\Longrightarrow$ |*Re z*| < *1*) $\Longrightarrow$ *Arccos holomorphic-on s*
by (*simp add*: *complex-differentiable-within-Arccos holomorphic-on-def*)

51.23 Upper and Lower Bounds for Inverse Sine and Cosine

lemma *Arcsin-bounds*: |*Re z*| < *1* $\Longrightarrow$ |*Re*(*Arcsin z*)| < *pi/2*
unfolding *Re-Arcsin*
by (*blast intro*: *Re-Ln-pos-lt-imp Arcsin-range-lemma*)

lemma *Arccos-bounds*: |*Re z*| < *1* $\Longrightarrow$ *0* < *Re*(*Arccos z*) $\wedge$ *Re*(*Arccos z*) < *pi*
unfolding *Re-Arccos*
by (*blast intro*!: *Im-Ln-pos-lt-imp Arccos-range-lemma*)

lemma *Re-Arccos-bounds*: $-$*pi* < *Re*(*Arccos z*) $\wedge$ *Re*(*Arccos z*) $\leq$ *pi*
unfolding *Re-Arccos*
by (*blast intro*!: *mpi-less-Im-Ln Im-Ln-le-pi Arccos-body-lemma*)

lemma *Re-Arccos-bound*: |*Re*(*Arccos z*)| $\leq$ *pi*
by (*meson Re-Arccos-bounds abs-le-iff less-eq-real-def minus-less-iff*)

lemma *Re-Arcsin-bounds*: $-$*pi* < *Re*(*Arcsin z*) & *Re*(*Arcsin z*) $\leq$ *pi*
unfolding *Re-Arcsin*
by (*blast intro*!: *mpi-less-Im-Ln Im-Ln-le-pi Arcsin-body-lemma*)

lemma *Re-Arcsin-bound*: $|Re(Arcsin\ z)| \leq pi$
by (*meson Re-Arcsin-bounds abs-le-iff less-eq-real-def minus-less-iff*)

51.24 Interrelations between Arcsin and Arccos

lemma *cos-Arcsin-nonzero*:
assumes $z^2 \neq 1$ **shows** $cos(Arcsin\ z) \neq 0$
proof −
have *eq*: $(\mathrm{i} * z * (csqrt\ (1 - z^2)))^2 = z^2 * (z^2 - 1)$
by (*simp add*: *power-mult-distrib algebra-simps*)
have $\mathrm{i} * z * (csqrt\ (1 - z^2)) \neq z^2 - 1$
proof
assume $\mathrm{i} * z * (csqrt\ (1 - z^2)) = z^2 - 1$
then have $(\mathrm{i} * z * (csqrt\ (1 - z^2)))^2 = (z^2 - 1)^2$
by *simp*
then have $z^2 * (z^2 - 1) = (z^2 - 1)*(z^2 - 1)$
using *eq power2-eq-square* **by** *auto*
then show *False*
using *assms* **by** *simp*
qed
then have $1 + \mathrm{i} * z * (csqrt\ (1 - z * z)) \neq z^2$
by (*metis add-minus-cancel power2-eq-square uminus-add-conv-diff*)
then have $2*(1 + \mathrm{i} * z * (csqrt\ (1 - z * z))) \neq 2*z^2$
by (*metis mult-cancel-left zero-neq-numeral*)
then have $(\mathrm{i} * z + csqrt\ (1 - z^2))^2 \neq -1$
using *assms*
apply (*auto simp*: *power2-sum*)
apply (*simp add*: *power2-eq-square algebra-simps*)
done
then show *?thesis*
apply (*simp add*: *cos-exp-eq Arcsin-def exp-minus*)
apply (*simp add*: *divide-simps Arcsin-body-lemma*)
apply (*metis add.commute minus-unique power2-eq-square*)
done
qed

lemma *sin-Arccos-nonzero*:
assumes $z^2 \neq 1$ **shows** $sin(Arccos\ z) \neq 0$
proof −
have *eq*: $(\mathrm{i} * z * (csqrt\ (1 - z^2)))^2 = -(z^2) * (1 - z^2)$
by (*simp add*: *power-mult-distrib algebra-simps*)
have $\mathrm{i} * z * (csqrt\ (1 - z^2)) \neq 1 - z^2$
proof
assume $\mathrm{i} * z * (csqrt\ (1 - z^2)) = 1 - z^2$
then have $(\mathrm{i} * z * (csqrt\ (1 - z^2)))^2 = (1 - z^2)^2$
by *simp*
then have $-(z^2) * (1 - z^2) = (1 - z^2)*(1 - z^2)$
using *eq power2-eq-square* **by** *auto*

```
    then have −(z²) = (1 − z²)
      using assms
      by (metis add.commute add.right-neutral diff-add-cancel mult-right-cancel)
    then show False
      using assms by simp
  qed
  then have z² + i ∗ z ∗ (csqrt (1 − z²)) ≠ 1
    by (simp add: algebra-simps)
  then have 2∗(z² + i ∗ z ∗ (csqrt (1 − z²))) ≠ 2∗1
    by (metis mult-cancel-left2 zero-neq-numeral)
  then have (z + i ∗ csqrt (1 − z²))² ≠ 1
    using assms
   apply (auto simp: Power.comm-semiring-1-class.power2-sum power-mult-distrib)
    apply (simp add: power2-eq-square algebra-simps)
    done
  then show ?thesis
    apply (simp add: sin-exp-eq Arccos-def exp-minus)
    apply (simp add: divide-simps Arccos-body-lemma)
    apply (simp add: power2-eq-square)
    done
qed

lemma cos-sin-csqrt:
  assumes 0 < cos(Re z) ∨ cos(Re z) = 0 ∧ Im z ∗ sin(Re z) ≤ 0
    shows cos z = csqrt(1 − (sin z)²)
  apply (rule csqrt-unique [THEN sym])
  apply (simp add: cos-squared-eq)
  using assms
  apply (auto simp: Re-cos Im-cos add-pos-pos mult-le-0-iff zero-le-mult-iff)
  done

lemma sin-cos-csqrt:
  assumes 0 < sin(Re z) ∨ sin(Re z) = 0 ∧ 0 ≤ Im z ∗ cos(Re z)
    shows sin z = csqrt(1 − (cos z)²)
  apply (rule csqrt-unique [THEN sym])
  apply (simp add: sin-squared-eq)
  using assms
  apply (auto simp: Re-sin Im-sin add-pos-pos mult-le-0-iff zero-le-mult-iff)
  done

lemma Arcsin-Arccos-csqrt-pos:
    (0 < Re z | Re z = 0 & 0 ≤ Im z) ⟹ Arcsin z = Arccos(csqrt(1 − z²))
  by (simp add: Arcsin-def Arccos-def Complex.csqrt-square add.commute)

lemma Arccos-Arcsin-csqrt-pos:
    (0 < Re z | Re z = 0 & 0 ≤ Im z) ⟹ Arccos z = Arcsin(csqrt(1 − z²))
  by (simp add: Arcsin-def Arccos-def Complex.csqrt-square add.commute)

lemma sin-Arccos:
```

0 < Re z | Re z = 0 & 0 ≤ Im z ⟹ sin(Arccos z) = csqrt(1 − z²)
by (*simp add*: *Arccos-Arcsin-csqrt-pos*)

lemma *cos-Arcsin*:
0 < Re z | Re z = 0 & 0 ≤ Im z ⟹ cos(Arcsin z) = csqrt(1 − z²)
by (*simp add*: *Arcsin-Arccos-csqrt-pos*)

51.25 Relationship with Arcsin on the Real Numbers

lemma *Im-Arcsin-of-real*:
assumes *|x| ≤ 1*
shows *Im (Arcsin (of-real x)) = 0*
proof −
have *csqrt (1 − (of-real x)²) = (if x^2 ≤ 1 then sqrt (1 − x^2) else* i ∗ *sqrt (x^2 − 1))*
by (*simp add*: *of-real-sqrt del*: *csqrt-of-real-nonneg*)
then have *cmod* (i ∗ *of-real x + csqrt (1 − (of-real x)²)) ^2 = 1*
using *assms abs-square-le-1*
by (*force simp add*: *Complex.cmod-power2*)
then have *cmod* (i ∗ *of-real x + csqrt (1 − (of-real x)²)) = 1*
by (*simp add*: *norm-complex-def*)
then show *?thesis*
by (*simp add*: *Im-Arcsin exp-minus*)
qed

corollary *Arcsin-in-Reals* [*simp*]: *z ∈ ℝ ⟹ |Re z| ≤ 1 ⟹ Arcsin z ∈ ℝ*
by (*metis Im-Arcsin-of-real Re-complex-of-real Reals-cases complex-is-Real-iff*)

lemma *arcsin-eq-Re-Arcsin*:
assumes *|x| ≤ 1*
shows *arcsin x = Re (Arcsin (of-real x))*
unfolding *arcsin-def*
proof (*rule the-equality*, *safe*)
show *− (pi / 2) ≤ Re (Arcsin (complex-of-real x))*
using *Re-Ln-pos-le* [*OF Arcsin-body-lemma*, *of of-real x*]
by (*auto simp*: *Complex.in-Reals-norm Re-Arcsin*)
next
show *Re (Arcsin (complex-of-real x)) ≤ pi / 2*
using *Re-Ln-pos-le* [*OF Arcsin-body-lemma*, *of of-real x*]
by (*auto simp*: *Complex.in-Reals-norm Re-Arcsin*)
next
show *sin (Re (Arcsin (complex-of-real x))) = x*
using *Re-sin* [*of Arcsin (of-real x)*] *Arcsin-body-lemma* [*of of-real x*]
by (*simp add*: *Im-Arcsin-of-real assms*)
next
fix *x′*
assume *− (pi / 2) ≤ x′ x′ ≤ pi / 2 x = sin x′*
then show *x′ = Re (Arcsin (complex-of-real (sin x′)))*
apply (*simp add*: *sin-of-real* [*symmetric*])

apply (*subst Arcsin-sin*)
apply (*auto simp*:)
done
qed

lemma *of-real-arcsin*: $|x| \leq 1 \Longrightarrow$ *of-real*(*arcsin x*) = *Arcsin*(*of-real x*)
by (*metis Im-Arcsin-of-real add.right-neutral arcsin-eq-Re-Arcsin complex-eq mult-zero-right of-real-0*)

51.26 Relationship with Arccos on the Real Numbers

lemma *Im-Arccos-of-real*:
assumes $|x| \leq 1$
shows *Im* (*Arccos* (*of-real x*)) = 0
proof −
have *csqrt* (1 − (*of-real x*)^2) = (*if x^2 ≤ 1 then sqrt* (1 − *x^2*) *else* i ∗ *sqrt* (*x^2* − 1))
by (*simp add*: *of-real-sqrt del*: *csqrt-of-real-nonneg*)
then have *cmod* (*of-real x* + i ∗ *csqrt* (1 − (*of-real x*)^2)) ^2 = 1
using *assms abs-square-le-1*
by (*force simp add*: *Complex.cmod-power2*)
then have *cmod* (*of-real x* + i ∗ *csqrt* (1 − (*of-real x*)^2)) = 1
by (*simp add*: *norm-complex-def*)
then show *?thesis*
by (*simp add*: *Im-Arccos exp-minus*)
qed

corollary *Arccos-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow |Re\ z| \leq 1 \Longrightarrow Arccos\ z \in \mathbb{R}$
by (*metis Im-Arccos-of-real Re-complex-of-real Reals-cases complex-is-Real-iff*)

lemma *arccos-eq-Re-Arccos*:
assumes $|x| \leq 1$
shows *arccos x* = *Re* (*Arccos* (*of-real x*))
unfolding *arccos-def*
proof (*rule the-equality*, *safe*)
show 0 ≤ *Re* (*Arccos* (*complex-of-real x*))
using *Im-Ln-pos-le* [*OF Arccos-body-lemma*, *of of-real x*]
by (*auto simp*: *Complex.in-Reals-norm Re-Arccos*)
next
show *Re* (*Arccos* (*complex-of-real x*)) ≤ *pi*
using *Im-Ln-pos-le* [*OF Arccos-body-lemma*, *of of-real x*]
by (*auto simp*: *Complex.in-Reals-norm Re-Arccos*)
next
show *cos* (*Re* (*Arccos* (*complex-of-real x*))) = *x*
using *Re-cos* [*of Arccos* (*of-real x*)] *Arccos-body-lemma* [*of of-real x*]
by (*simp add*: *Im-Arccos-of-real assms*)
next
fix *x′*
assume 0 ≤ *x′ x′* ≤ *pi x* = *cos x′*

then show $x' = Re\ (Arccos\ (complex\text{-}of\text{-}real\ (cos\ x')))$
apply (*simp add*: *cos-of-real* [*symmetric*])
apply (*subst Arccos-cos*)
apply (*auto simp*:)
done
qed

lemma *of-real-arccos*: $|x| \leq 1 \Longrightarrow of\text{-}real(arccos\ x) = Arccos(of\text{-}real\ x)$
by (*metis Im-Arccos-of-real add.right-neutral arccos-eq-Re-Arccos complex-eq mult-zero-right of-real-0*)

51.27 Some interrelationships among the real inverse trig functions.

lemma *arccos-arctan*:
assumes $-1 < x\ x < 1$
shows $arccos\ x = pi/2 - arctan(x\ /\ sqrt(1 - x^2))$
proof −
have $arctan(x\ /\ sqrt(1 - x^2)) - (pi/2 - arccos\ x) = 0$
proof (*rule sin-eq-0-pi*)
show $-\ pi < arctan\ (x\ /\ sqrt\ (1 - x^2)) - (pi\ /\ 2 - arccos\ x)$
using *arctan-lbound* [*of x / sqrt*$(1 - x^2)$] *arccos-bounded* [*of x*] *assms*
by (*simp add*: *algebra-simps*)
next
show $arctan\ (x\ /\ sqrt\ (1 - x^2)) - (pi\ /\ 2 - arccos\ x) < pi$
using *arctan-ubound* [*of x / sqrt*$(1 - x^2)$] *arccos-bounded* [*of x*] *assms*
by (*simp add*: *algebra-simps*)
next
show $sin\ (arctan\ (x\ /\ sqrt\ (1 - x^2)) - (pi\ /\ 2 - arccos\ x)) = 0$
using *assms*
by (*simp add*: *algebra-simps sin-diff cos-add sin-arccos sin-arctan cos-arctan power2-eq-square square-eq-1-iff*)
qed
then show *?thesis*
by *simp*
qed

lemma *arcsin-plus-arccos*:
assumes $-1 \leq x\ x \leq 1$
shows $arcsin\ x + arccos\ x = pi/2$
proof −
have $arcsin\ x = pi/2 - arccos\ x$
apply (*rule sin-inj-pi*)
using *assms arcsin* [*OF assms*] *arccos* [*OF assms*]
apply (*auto simp*: *algebra-simps sin-diff*)
done
then show *?thesis*
by (*simp add*: *algebra-simps*)
qed

lemma *arcsin-arccos-eq*: $-1 \leq x \Longrightarrow x \leq 1 \Longrightarrow \arcsin\ x = pi/2 - \arccos\ x$
using *arcsin-plus-arccos* **by** *force*

lemma *arccos-arcsin-eq*: $-1 \leq x \Longrightarrow x \leq 1 \Longrightarrow \arccos\ x = pi/2 - \arcsin\ x$
using *arcsin-plus-arccos* **by** *force*

lemma *arcsin-arctan*: $-1 < x \Longrightarrow x < 1 \Longrightarrow \arcsin\ x = \arctan(x\ /\ sqrt(1 - x^2))$
by (*simp add*: *arccos-arctan arcsin-arccos-eq*)

lemma *csqrt-1-diff-eq*: $csqrt\ (1 - (of\text{-}real\ x)^2) = (if\ x\hat{}2 \leq 1\ then\ sqrt\ (1 - x\hat{}2)\ else\ \mathrm{i} * sqrt\ (x\hat{}2 - 1))$
by (*simp add*: *of-real-sqrt del*: *csqrt-of-real-nonneg*)

lemma *arcsin-arccos-sqrt-pos*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow \arcsin\ x = \arccos(sqrt(1 - x^2))$
apply (*simp add*: *abs-square-le-1 arcsin-eq-Re-Arcsin arccos-eq-Re-Arccos*)
apply (*subst Arcsin-Arccos-csqrt-pos*)
apply (*auto simp*: *power-le-one csqrt-1-diff-eq*)
done

lemma *arcsin-arccos-sqrt-neg*: $-1 \leq x \Longrightarrow x \leq 0 \Longrightarrow \arcsin\ x = -\arccos(sqrt(1 - x^2))$
using *arcsin-arccos-sqrt-pos* [*of* $-x$]
by (*simp add*: *arcsin-minus*)

lemma *arccos-arcsin-sqrt-pos*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow \arccos\ x = \arcsin(sqrt(1 - x^2))$
apply (*simp add*: *abs-square-le-1 arcsin-eq-Re-Arcsin arccos-eq-Re-Arccos*)
apply (*subst Arccos-Arcsin-csqrt-pos*)
apply (*auto simp*: *power-le-one csqrt-1-diff-eq*)
done

lemma *arccos-arcsin-sqrt-neg*: $-1 \leq x \Longrightarrow x \leq 0 \Longrightarrow \arccos\ x = pi - \arcsin(sqrt(1 - x^2))$
using *arccos-arcsin-sqrt-pos* [*of* $-x$]
by (*simp add*: *arccos-minus*)

51.28 continuity results for arcsin and arccos.

lemma *continuous-on-Arcsin-real* [*continuous-intros*]:
continuous-on $\{w \in \mathbb{R}.\ |Re\ w| \leq 1\}$ *Arcsin*
proof −
have *continuous-on* $\{w \in \mathbb{R}.\ |Re\ w| \leq 1\}$ $(\lambda x.\ complex\text{-}of\text{-}real\ (\arcsin\ (Re\ x)))$ =
continuous-on $\{w \in \mathbb{R}.\ |Re\ w| \leq 1\}$ $(\lambda x.\ complex\text{-}of\text{-}real\ (Re\ (Arcsin\ (of\text{-}real\ (Re\ x)))))$
by (*rule continuous-on-cong* [*OF refl*]) (*simp add*: *arcsin-eq-Re-Arcsin*)

also have ... = *?thesis*
by (*rule continuous-on-cong* [*OF refl*]) *simp*
finally show *?thesis*
using *continuous-on-arcsin* [*OF continuous-on-Re* [*OF continuous-on-id*], *of* {$w \in \mathbb{R}$. |*Re w*| ≤ *1*}]
continuous-on-of-real
by *fastforce*
qed

lemma *continuous-within-Arcsin-real*:
continuous (*at z within* {$w \in \mathbb{R}$. |*Re w*| ≤ *1*}) *Arcsin*
proof (*cases z* ∈ {$w \in \mathbb{R}$. |*Re w*| ≤ *1*})
case *True* **then show** *?thesis*
using *continuous-on-Arcsin-real continuous-on-eq-continuous-within*
by *blast*
next
case *False*
with *closed-real-abs-le* [*of 1*] **show** *?thesis*
by (*rule continuous-within-closed-nontrivial*)
qed

lemma *continuous-on-Arccos-real*:
continuous-on {$w \in \mathbb{R}$. |*Re w*| ≤ *1*} *Arccos*
proof −
have *continuous-on* {$w \in \mathbb{R}$. |*Re w*| ≤ *1*} (λ*x*. *complex-of-real* (*arccos* (*Re x*)))
=
continuous-on {$w \in \mathbb{R}$. |*Re w*| ≤ *1*} (λ*x*. *complex-of-real* (*Re* (*Arccos* (*of-real* (*Re x*)))))
by (*rule continuous-on-cong* [*OF refl*]) (*simp add*: *arccos-eq-Re-Arccos*)
also have ... = *?thesis*
by (*rule continuous-on-cong* [*OF refl*]) *simp*
finally show *?thesis*
using *continuous-on-arccos* [*OF continuous-on-Re* [*OF continuous-on-id*], *of* {$w \in \mathbb{R}$. |*Re w*| ≤ *1*}]
continuous-on-of-real
by *fastforce*
qed

lemma *continuous-within-Arccos-real*:
continuous (*at z within* {$w \in \mathbb{R}$. |*Re w*| ≤ *1*}) *Arccos*
proof (*cases z* ∈ {$w \in \mathbb{R}$. |*Re w*| ≤ *1*})
case *True* **then show** *?thesis*
using *continuous-on-Arccos-real continuous-on-eq-continuous-within*
by *blast*
next
case *False*
with *closed-real-abs-le* [*of 1*] **show** *?thesis*
by (*rule continuous-within-closed-nontrivial*)
qed

51.29 Roots of unity

lemma *complex-root-unity*:
fixes *j*::*nat*
assumes *n* ≠ *0*
shows *exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) ^ *n* = *1*
proof −
have ∗: *of-nat j* ∗ (*complex-of-real pi* ∗ *2*) = *complex-of-real* (*2* ∗ *real j* ∗ *pi*)
by (*simp add*: *of-real-numeral*)
then show *?thesis*
apply (*simp add*: *exp-of-nat-mult* [*symmetric*] *mult-ac exp-Euler*)
apply (*simp only*: ∗ *cos-of-real sin-of-real*)
apply (*simp add*:)
done
qed

lemma *complex-root-unity-eq*:
fixes *j*::*nat* **and** *k*::*nat*
assumes *1* ≤ *n*
shows (*exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) = *exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat k* / *of-nat n*)
⟷ *j mod n* = *k mod n*)
proof −
have (∃ *z*::*int*. i ∗ (*of-nat j* ∗ (*of-real pi* ∗ *2*)) =
i ∗ (*of-nat k* ∗ (*of-real pi* ∗ *2*)) + i ∗ (*of-int z* ∗ (*of-nat n* ∗ (*of-real pi* ∗ *2*)))) ⟷
(∃ *z*::*int*. *of-nat j* ∗ (i ∗ (*of-real pi* ∗ *2*)) =
(*of-nat k* + *of-nat n* ∗ *of-int z*) ∗ (i ∗ (*of-real pi* ∗ *2*)))
by (*simp add*: *algebra-simps*)
also have ... ⟷ (∃ *z*::*int*. *of-nat j* = *of-nat k* + *of-nat n* ∗ (*of-int z* :: *complex*))
by *simp*
also have ... ⟷ (∃ *z*::*int*. *of-nat j* = *of-nat k* + *of-nat n* ∗ *z*)
apply (*rule HOL.iff-exI*)
apply (*auto simp*:)
using *of-int-eq-iff* **apply** *fastforce*
by (*metis of-int-add of-int-mult of-int-of-nat-eq*)
also have ... ⟷ *int j mod int n* = *int k mod int n*
by (*auto simp*: *zmod-eq-dvd-iff dvd-def algebra-simps*)
also have ... ⟷ *j mod n* = *k mod n*
by (*metis of-nat-eq-iff zmod-int*)
finally have (∃ *z*. i ∗ (*of-nat j* ∗ (*of-real pi* ∗ *2*)) =
i ∗ (*of-nat k* ∗ (*of-real pi* ∗ *2*)) + i ∗ (*of-int z* ∗ (*of-nat n* ∗ (*of-real pi* ∗ *2*)))) ⟷ *j mod n* = *k mod n* **.**
note ∗ = *this*
show *?thesis*
using *assms*
by (*simp add*: *exp-eq divide-simps mult-ac of-real-numeral* ∗)
qed

corollary *bij-betw-roots-unity*:

bij-betw (λ*j*. *exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*))
{..<*n*} {*exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) | *j*. *j* < *n*}
by (*auto simp*: *bij-betw-def inj-on-def complex-root-unity-eq*)

lemma *complex-root-unity-eq-1*:
fixes *j*::*nat* **and** *k*::*nat*
assumes *1* ≤ *n*
shows *exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) = *1* ⟷ *n dvd j*
proof −
have *1* = *exp*(*2* ∗ *of-real pi* ∗ i ∗ (*of-nat n* / *of-nat n*))
using *assms* **by** *simp*
then have *exp*(*2* ∗ *of-real pi* ∗ i ∗ (*of-nat j* / *of-nat n*)) = *1* ⟷ *j mod n* = *n mod n*
using *complex-root-unity-eq* [*of n j n*] *assms*
by *simp*
then show *?thesis*
by *auto*
qed

lemma *finite-complex-roots-unity-explicit*:
finite {*exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) | *j*::*nat*. *j* < *n*}
by *simp*

lemma *card-complex-roots-unity-explicit*:
card {*exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) | *j*::*nat*. *j* < *n*} = *n*
by (*simp add*: *Finite-Set.bij-betw-same-card* [*OF bij-betw-roots-unity*, *symmetric*])

lemma *complex-roots-unity*:
assumes *1* ≤ *n*
shows {*z*::*complex*. *z*^*n* = *1*} = {*exp*(*2* ∗ *of-real pi* ∗ i ∗ *of-nat j* / *of-nat n*) | *j*::*nat*. *j* < *n*}
apply (*rule Finite-Set.card-seteq* [*symmetric*])
using *assms*
apply (*auto simp*: *card-complex-roots-unity-explicit finite-roots-unity complex-root-unity card-roots-unity*)
done

lemma *card-complex-roots-unity*: *1* ≤ *n* ⟹ *card* {*z*::*complex*. *z*^*n* = *1*} = *n*
by (*simp add*: *card-complex-roots-unity-explicit complex-roots-unity*)

lemma *complex-not-root-unity*:
1 ≤ *n* ⟹ ∃ *u*::*complex*. *norm u* = *1* ∧ *u*^*n* ≠ *1*
apply (*rule-tac x*=*exp* (*of-real pi* ∗ i ∗ *of-real* (*1* / *n*)) **in** *exI*)
apply (*auto simp*: *Re-complex-div-eq-0 exp-of-nat-mult* [*symmetric*] *mult-ac exp-Euler*)
done

end

52 Complex path integrals and Cauchy's integral theorem

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

theory *Cauchy-Integral-Thm*
imports *Complex-Transcendental Weierstrass Ordered-Euclidean-Space*
begin

52.1 Piecewise differentiable functions

definition *piecewise-differentiable-on*
(**infixr** *piecewise'-differentiable'-on 50*)
where *f piecewise-differentiable-on i* ≡
continuous-on i f ∧
(∃ *s*. *finite s* ∧ (∀ *x* ∈ *i* − *s*. *f differentiable* (*at x within i*)))

lemma *piecewise-differentiable-on-imp-continuous-on*:
f piecewise-differentiable-on s ⟹ *continuous-on s f*
by (*simp add*: *piecewise-differentiable-on-def*)

lemma *piecewise-differentiable-on-subset*:
f piecewise-differentiable-on s ⟹ *t* ≤ *s* ⟹ *f piecewise-differentiable-on t*
using *continuous-on-subset*
unfolding *piecewise-differentiable-on-def*
apply *safe*
apply (*blast intro*: *elim*: *continuous-on-subset*)
by (*meson Diff-iff differentiable-within-subset subsetCE*)

lemma *differentiable-on-imp-piecewise-differentiable*:
fixes *a*:: *'a*::{*linorder-topology*,*real-normed-vector*}
shows *f differentiable-on* {*a*..*b*} ⟹ *f piecewise-differentiable-on* {*a*..*b*}
apply (*simp add*: *piecewise-differentiable-on-def differentiable-imp-continuous-on*)
apply (*rule-tac x*={*a*,*b*} **in** *exI*, *simp add*: *differentiable-on-def*)
done

lemma *differentiable-imp-piecewise-differentiable*:
(⋀*x*. *x* ∈ *s* ⟹ *f differentiable* (*at x within s*))
⟹ *f piecewise-differentiable-on s*
by (*auto simp*: *piecewise-differentiable-on-def differentiable-imp-continuous-on differentiable-on-def*
intro: *differentiable-within-subset*)

lemma *piecewise-differentiable-const* [*iff*]: (λ*x*. *z*) *piecewise-differentiable-on s*
by (*simp add*: *differentiable-imp-piecewise-differentiable*)

lemma *piecewise-differentiable-compose*:
⟦*f piecewise-differentiable-on s*; *g piecewise-differentiable-on* (*f* ‘ *s*);
⋀*x*. *finite* (*s* ∩ *f*−‘{*x*})⟧
⟹ (*g o f*) *piecewise-differentiable-on s*

apply (*simp add*: *piecewise-differentiable-on-def*, *safe*)
apply (*blast intro*: *continuous-on-compose2*)
apply (*rename-tac A B*)
apply (*rule-tac x*=$A \cup (\bigcup x{\in}B.\ s \cap f-`\{x\})$ **in** *exI*)
apply (*blast intro*: *differentiable-chain-within*)
done

lemma *piecewise-differentiable-affine*:
fixes *m*::*real*
assumes *f piecewise-differentiable-on* $((\lambda x.\ m *_R x + c)\ `\ s)$
shows $(f\ o\ (\lambda x.\ m *_R x + c))$ *piecewise-differentiable-on s*
proof (*cases m* = *0*)
case *True*
then show *?thesis*
unfolding *o-def*
by (*force intro*: *differentiable-imp-piecewise-differentiable differentiable-const*)
next
case *False*
show *?thesis*
apply (*rule piecewise-differentiable-compose* [*OF differentiable-imp-piecewise-differentiable*])
apply (*rule assms derivative-intros* | *simp add*: *False vimage-def real-vector-affinity-eq*)+
done
qed

lemma *piecewise-differentiable-cases*:
fixes *c*::*real*
assumes *f piecewise-differentiable-on* $\{a..c\}$
g piecewise-differentiable-on $\{c..b\}$
$a \le c\ c \le b\ f\ c = g\ c$
shows $(\lambda x.\ if\ x \le c\ then\ f\ x\ else\ g\ x)$ *piecewise-differentiable-on* $\{a..b\}$
proof −
obtain *s t* **where** *st*: *finite s finite t*
$\forall x{\in}\{a..c\} - s.\ f$ *differentiable at x within* $\{a..c\}$
$\forall x{\in}\{c..b\} - t.\ g$ *differentiable at x within* $\{c..b\}$
using *assms*
by (*auto simp*: *piecewise-differentiable-on-def*)
have *finabc*: *finite* $(\{a,b,c\} \cup (s \cup t))$
by (*metis* ‹*finite s*› ‹*finite t*› *finite-Un finite-insert finite.emptyI*)
have *continuous-on* $\{a..c\}$ *f continuous-on* $\{c..b\}$ *g*
using *assms piecewise-differentiable-on-def* **by** *auto*
then have *continuous-on* $\{a..b\}$ $(\lambda x.\ if\ x \le c\ then\ f\ x\ else\ g\ x)$
using *continuous-on-cases* [*OF closed-real-atLeastAtMost* [*of a c*],
OF closed-real-atLeastAtMost [*of c b*],
of f g $\lambda x.\ x{\le}c$] *assms*
by (*force simp*: *ivl-disj-un-two-touch*)
moreover
{ fix *x*
assume *x*: $x \in \{a..b\} - (\{a,b,c\} \cup (s \cup t))$
have $(\lambda x.\ if\ x \le c\ then\ f\ x\ else\ g\ x)$ *differentiable at x within* $\{a..b\}$ (**is** *?diff-fg*)

```
    proof (cases x c rule: le-cases)
      case le show ?diff-fg
        apply (rule differentiable-transform-within [where d = dist x c and f =
f])
        using x le st
        apply (simp-all add: dist-real-def)
        apply (rule differentiable-at-withinI)
        apply (rule differentiable-within-open [where s = {a<..<c} - s, THEN
iffD1], simp-all)
        apply (blast intro: open-greaterThanLessThan finite-imp-closed)
        apply (force elim!: differentiable-subset)+
        done
    next
      case ge show ?diff-fg
        apply (rule differentiable-transform-within [where d = dist x c and f =
g])
        using x ge st
        apply (simp-all add: dist-real-def)
        apply (rule differentiable-at-withinI)
        apply (rule differentiable-within-open [where s = {c<..<b} - t, THEN
iffD1], simp-all)
        apply (blast intro: open-greaterThanLessThan finite-imp-closed)
        apply (force elim!: differentiable-subset)+
        done
    qed
  }
  then have ∃ s. finite s ∧
                (∀ x∈{a..b} - s. (λx. if x ≤ c then f x else g x) differentiable at x
within {a..b})
    by (meson finabc)
  ultimately show ?thesis
    by (simp add: piecewise-differentiable-on-def)
qed

lemma piecewise-differentiable-neg:
    f piecewise-differentiable-on s ⟹ (λx. -(f x)) piecewise-differentiable-on s
  by (auto simp: piecewise-differentiable-on-def continuous-on-minus)

lemma piecewise-differentiable-add:
  assumes f piecewise-differentiable-on i
          g piecewise-differentiable-on i
    shows (λx. f x + g x) piecewise-differentiable-on i
proof -
  obtain s t where st: finite s finite t
                   ∀ x∈i - s. f differentiable at x within i
                   ∀ x∈i - t. g differentiable at x within i
    using assms by (auto simp: piecewise-differentiable-on-def)
  then have finite (s ∪ t) ∧ (∀ x∈i - (s ∪ t). (λx. f x + g x) differentiable at x
within i)
```

by *auto*
moreover have *continuous-on i f continuous-on i g*
using *assms piecewise-differentiable-on-def* **by** *auto*
ultimately show *?thesis*
by (*auto simp*: *piecewise-differentiable-on-def continuous-on-add*)
qed

lemma *piecewise-differentiable-diff*:
⟦*f piecewise-differentiable-on s*; *g piecewise-differentiable-on s*⟧
⟹ (λ*x*. *f x* − *g x*) *piecewise-differentiable-on s*
unfolding *diff-conv-add-uminus*
by (*metis piecewise-differentiable-add piecewise-differentiable-neg*)

lemma *continuous-on-joinpaths-D1*:
continuous-on {*0..1*} (*g1* +++ *g2*) ⟹ *continuous-on* {*0..1*} *g1*
apply (*rule continuous-on-eq* [*of* - (*g1* +++ *g2*) *o* (*op*∗(*inverse 2*))])
apply (*rule continuous-intros* | *simp*)+
apply (*auto elim*!: *continuous-on-subset simp*: *joinpaths-def*)
done

lemma *continuous-on-joinpaths-D2*:
⟦*continuous-on* {*0..1*} (*g1* +++ *g2*); *pathfinish g1* = *pathstart g2*⟧ ⟹ *continuous-on* {*0..1*} *g2*
apply (*rule continuous-on-eq* [*of* - (*g1* +++ *g2*) *o* (λ*x*. *inverse 2*∗*x* + *1*/*2*)])
apply (*rule continuous-intros* | *simp*)+
apply (*auto elim*!: *continuous-on-subset simp add*: *joinpaths-def pathfinish-def pathstart-def Ball-def*)
done

lemma *piecewise-differentiable-D1*:
(*g1* +++ *g2*) *piecewise-differentiable-on* {*0..1*} ⟹ *g1 piecewise-differentiable-on* {*0..1*}
apply (*clarsimp simp add*: *piecewise-differentiable-on-def dest*!: *continuous-on-joinpaths-D1*)
apply (*rule-tac x*=*insert 1* ((*op*∗*2*)‘*s*) **in** *exI*)
apply *simp*
apply (*intro ballI*)
apply (*rule-tac d*=*dist* (*x*/*2*) (*1*/*2*) **and** *f* = (*g1* +++ *g2*) *o* (*op*∗(*inverse 2*))
in *differentiable-transform-within*)
apply (*auto simp*: *dist-real-def joinpaths-def*)
apply (*rule differentiable-chain-within derivative-intros* | *simp*)+
apply (*rule differentiable-subset*)
apply (*force simp*:)+
done

lemma *piecewise-differentiable-D2*:
⟦(*g1* +++ *g2*) *piecewise-differentiable-on* {*0..1*}; *pathfinish g1* = *pathstart g2*⟧
⟹ *g2 piecewise-differentiable-on* {*0..1*}
apply (*clarsimp simp add*: *piecewise-differentiable-on-def dest*!: *continuous-on-joinpaths-D2*)
apply (*rule-tac x*=*insert 0* ((λ*x*. *2*∗*x*−*1*)‘*s*) **in** *exI*)

apply *simp*
apply (*intro ballI*)
apply (*rule-tac d=dist ((x+1)/2) (1/2)* **and** $f = (g1 \mathbin{+\!\!+\!\!+} g2) \circ (\lambda x.\ (x+1)/2)$
in *differentiable-transform-within*)
apply (*auto simp*: *dist-real-def joinpaths-def abs-if field-simps split*: *split-if-asm*)
apply (*rule differentiable-chain-within derivative-intros* | *simp*)+
apply (*rule differentiable-subset*)
apply (*force simp*: *divide-simps*)+
done

52.1.1 The concept of continuously differentiable

definition *C1-differentiable-on* :: (*real* ⇒ *'a*::*real-normed-vector*) ⇒ *real set* ⇒ *bool*
(**infix** *C1'-differentiable'-on 50*)
where
f C1-differentiable-on s ⟷
(∃ *D*. (∀ *x* ∈ *s*. (*f has-vector-derivative* (*D x*)) (*at x*)) ∧ *continuous-on s D*)

lemma *C1-differentiable-on-eq*:
f C1-differentiable-on s ⟷
(∀ *x* ∈ *s*. *f differentiable at x*) ∧ *continuous-on s* (λ*x*. *vector-derivative f* (*at x*))
unfolding *C1-differentiable-on-def*
apply *safe*
using *differentiable-def has-vector-derivative-def* **apply** *blast*
apply (*erule continuous-on-eq*)
using *vector-derivative-at* **apply** *fastforce*
using *vector-derivative-works* **apply** *fastforce*
done

lemma *C1-differentiable-on-subset*:
f C1-differentiable-on t ⟹ *s* ⊆ *t* ⟹ *f C1-differentiable-on s*
unfolding *C1-differentiable-on-def continuous-on-eq-continuous-within*
by (*blast intro*: *continuous-within-subset*)

lemma *C1-differentiable-compose*:
⟦*f C1-differentiable-on s*; *g C1-differentiable-on* (*f* ‘ *s*);
⋀*x*. *finite* (*s* ∩ *f* −‘{*x*})⟧
⟹ (*g* ∘ *f*) *C1-differentiable-on s*
apply (*simp add*: *C1-differentiable-on-eq*, *safe*)
using *differentiable-chain-at* **apply** *blast*
apply (*rule continuous-on-eq* [*of* - λ*x*. *vector-derivative f* (*at x*) $*_R$ *vector-derivative g* (*at* (*f x*))])
apply (*rule Limits.continuous-on-scaleR*, *assumption*)
apply (*metis* (*mono-tags*, *lifting*) *continuous-on-eq continuous-at-imp-continuous-on continuous-on-compose differentiable-imp-continuous-within o-def*)
by (*simp add*: *vector-derivative-chain-at*)

lemma *C1-diff-imp-diff*: *f C1-differentiable-on s* $\Longrightarrow$ *f differentiable-on s*
by (*simp add*: *C1-differentiable-on-eq differentiable-at-imp-differentiable-on*)

lemma *C1-differentiable-on-ident* [*simp*, *derivative-intros*]: ($\lambda x.\ x$) *C1-differentiable-on s*
by (*auto simp*: *C1-differentiable-on-eq continuous-on-const*)

lemma *C1-differentiable-on-const* [*simp*, *derivative-intros*]: ($\lambda z.\ a$) *C1-differentiable-on s*
by (*auto simp*: *C1-differentiable-on-eq continuous-on-const*)

lemma *C1-differentiable-on-add* [*simp*, *derivative-intros*]:
f C1-differentiable-on s $\Longrightarrow$ *g C1-differentiable-on s* $\Longrightarrow$ ($\lambda x.\ f\ x + g\ x$) *C1-differentiable-on s*
unfolding *C1-differentiable-on-eq* **by** (*auto intro*: *continuous-intros*)

lemma *C1-differentiable-on-minus* [*simp*, *derivative-intros*]:
f C1-differentiable-on s $\Longrightarrow$ ($\lambda x.\ -\ f\ x$) *C1-differentiable-on s*
unfolding *C1-differentiable-on-eq* **by** (*auto intro*: *continuous-intros*)

lemma *C1-differentiable-on-diff* [*simp*, *derivative-intros*]:
f C1-differentiable-on s $\Longrightarrow$ *g C1-differentiable-on s* $\Longrightarrow$ ($\lambda x.\ f\ x - g\ x$) *C1-differentiable-on s*
unfolding *C1-differentiable-on-eq* **by** (*auto intro*: *continuous-intros*)

lemma *C1-differentiable-on-mult* [*simp*, *derivative-intros*]:
fixes *f g* :: *real* $\Rightarrow$ *'a* :: *real-normed-algebra*
shows *f C1-differentiable-on s* $\Longrightarrow$ *g C1-differentiable-on s* $\Longrightarrow$ ($\lambda x.\ f\ x * g\ x$) *C1-differentiable-on s*
unfolding *C1-differentiable-on-eq*
by (*auto simp*: *continuous-on-add continuous-on-mult continuous-at-imp-continuous-on differentiable-imp-continuous-within*)

lemma *C1-differentiable-on-scaleR* [*simp*, *derivative-intros*]:
f C1-differentiable-on s $\Longrightarrow$ *g C1-differentiable-on s* $\Longrightarrow$ ($\lambda x.\ f\ x *_R g\ x$) *C1-differentiable-on s*
unfolding *C1-differentiable-on-eq*
by (*rule continuous-intros* | *simp add*: *continuous-at-imp-continuous-on differentiable-imp-continuous-within*)

definition *piecewise-C1-differentiable-on*
(**infixr** *piecewise'-C1'-differentiable'-on 50*)
where *f piecewise-C1-differentiable-on i* $\equiv$
continuous-on i f $\wedge$
($\exists s.$ *finite s* $\wedge$ (*f C1-differentiable-on* (*i* $-$ *s*)))

lemma *C1-differentiable-imp-piecewise*:
f C1-differentiable-on s $\Longrightarrow$ *f piecewise-C1-differentiable-on s*
by (*auto simp*: *piecewise-C1-differentiable-on-def C1-differentiable-on-eq continuous-at-imp-continuous-on*

differentiable-imp-continuous-within)

lemma *piecewise-C1-imp-differentiable*:
  f piecewise-C1-differentiable-on i $\Longrightarrow$ *f piecewise-differentiable-on i*
 by (*auto simp*: *piecewise-C1-differentiable-on-def piecewise-differentiable-on-def*
      C1-differentiable-on-def differentiable-def has-vector-derivative-def
      intro: *has-derivative-at-within*)

lemma *piecewise-C1-differentiable-compose*:
  ⟦*f piecewise-C1-differentiable-on s*; *g piecewise-C1-differentiable-on* (*f* ‘ *s*);
    $\bigwedge x$. *finite* (*s* ∩ *f* −‘{*x*})⟧
    $\Longrightarrow$ (*g o f*) *piecewise-C1-differentiable-on s*
 apply (*simp add*: *piecewise-C1-differentiable-on-def*, *safe*)
 apply (*blast intro*: *continuous-on-compose2*)
 apply (*rename-tac A B*)
 apply (*rule-tac x*=*A* ∪ (⋃*x*∈*B*. *s* ∩ *f* −‘{*x*}) **in** *exI*)
 apply (*rule conjI*, *blast*)
 apply (*rule C1-differentiable-compose*)
 apply (*blast intro*: *C1-differentiable-on-subset*)
 apply (*blast intro*: *C1-differentiable-on-subset*)
 by (*simp add*: *Diff-Int-distrib2*)

lemma *piecewise-C1-differentiable-on-subset*:
  f piecewise-C1-differentiable-on s $\Longrightarrow$ *t* ≤ *s* $\Longrightarrow$ *f piecewise-C1-differentiable-on*
t
 by (*auto simp*: *piecewise-C1-differentiable-on-def elim*!: *continuous-on-subset C1-differentiable-on-subset*)

lemma *C1-differentiable-imp-continuous-on*:
 f C1-differentiable-on s $\Longrightarrow$ *continuous-on s f*
 unfolding *C1-differentiable-on-eq continuous-on-eq-continuous-within*
 using *differentiable-at-withinI differentiable-imp-continuous-within* **by** *blast*

lemma *C1-differentiable-on-empty* [*iff*]: *f C1-differentiable-on* {}
 unfolding *C1-differentiable-on-def*
 by *auto*

lemma *piecewise-C1-differentiable-affine*:
 fixes *m*::*real*
 assumes *f piecewise-C1-differentiable-on* ((λ*x*. *m* ∗ *x* + *c*) ‘ *s*)
 shows (*f o* (λ*x*. *m* ∗$_R$ *x* + *c*)) *piecewise-C1-differentiable-on s*
proof (*cases m* = *0*)
 case *True*
 then show *?thesis*
  unfolding *o-def* **by** (*auto simp*: *piecewise-C1-differentiable-on-def continuous-on-const*)
next
 case *False*
 show *?thesis*
  apply (*rule piecewise-C1-differentiable-compose* [*OF C1-differentiable-imp-piecewise*])
   apply (*rule assms derivative-intros* | *simp add*: *False vimage-def*)+

```
  using real-vector-affinity-eq [OF False, where c=c, unfolded scaleR-conv-of-real]
   apply simp
   done
qed

lemma piecewise-C1-differentiable-cases:
  fixes c::real
  assumes f piecewise-C1-differentiable-on {a..c}
          g piecewise-C1-differentiable-on {c..b}
          a ≤ c c ≤ b f c = g c
  shows (λx. if x ≤ c then f x else g x) piecewise-C1-differentiable-on {a..b}
proof −
  obtain s t where st: f C1-differentiable-on ({a..c} − s)
                      g C1-differentiable-on ({c..b} − t)
                      finite s finite t
    using assms
    by (force simp: piecewise-C1-differentiable-on-def)
  then have f-diff: f differentiable-on {a..<c} − s
       and g-diff: g differentiable-on {c<..b} − t
   by (simp-all add: C1-differentiable-on-eq differentiable-at-withinI differentiable-on-def)
  have continuous-on {a..c} f continuous-on {c..b} g
    using assms piecewise-C1-differentiable-on-def by auto
  then have cab: continuous-on {a..b} (λx. if x ≤ c then f x else g x)
    using continuous-on-cases [OF closed-real-atLeastAtMost [of a c],
                               OF closed-real-atLeastAtMost [of c b],
                               of f g λx. x≤c]  assms
    by (force simp: ivl-disj-un-two-touch)
  { fix x
    assume x: x ∈ {a..b} − insert c (s ∪ t)
    have (λx. if x ≤ c then f x else g x) differentiable at x (is ?diff-fg)
    proof (cases x c rule: le-cases)
      case le show ?diff-fg
        apply (rule differentiable-transform-within [where f=f and d = dist x c])
        using x dist-real-def le st by (auto simp: C1-differentiable-on-eq)
    next
      case ge show ?diff-fg
        apply (rule differentiable-transform-within [where f=g and d = dist x c])
        using dist-nz x dist-real-def ge st x by (auto simp: C1-differentiable-on-eq)
    qed
  }
  then have (∀x ∈ {a..b} − insert c (s ∪ t). (λx. if x ≤ c then f x else g x)
differentiable at x)
    by auto
  moreover
  { assume fcon: continuous-on ({a<..<c} − s) (λx. vector-derivative f (at x))
       and gcon: continuous-on ({c<..<b} − t) (λx. vector-derivative g (at x))
    have open ({a<..<c} − s)  open ({c<..<b} − t)
      using st by (simp-all add: open-Diff finite-imp-closed)
    moreover have continuous-on ({a<..<c} − s) (λx. vector-derivative (λx. if
```

$x \leq c$ *then* $f\ x$ *else* $g\ x)$ $(at\ x))$
apply (*rule continuous-on-eq* [*OF fcon*])
apply (*simp add*:)
apply (*rule vector-derivative-at* [*symmetric*])
apply (*rule-tac* f=f **and** d=*dist* $x\ c$ **in** *has-vector-derivative-transform-within*)
apply (*simp-all add*: *dist-norm vector-derivative-works* [*symmetric*])
apply (*metis* (*full-types*) *C1-differentiable-on-eq Diff-iff Groups.add-ac*(2) *add-mono-thms-linordered-field*(5) *atLeastAtMost-iff linorder-not-le order-less-irrefl st*(1))
apply *auto*
done
moreover have *continuous-on* ({c<..<b} − t) (λx. *vector-derivative* (λx. *if* $x \leq c$ *then* $f\ x$ *else* $g\ x$) $(at\ x)$)
apply (*rule continuous-on-eq* [*OF gcon*])
apply (*simp add*:)
apply (*rule vector-derivative-at* [*symmetric*])
apply (*rule-tac* f=g **and** d=*dist* $x\ c$ **in** *has-vector-derivative-transform-within*)
apply (*simp-all add*: *dist-norm vector-derivative-works* [*symmetric*])
apply (*metis* (*full-types*) *C1-differentiable-on-eq Diff-iff Groups.add-ac*(2) *add-mono-thms-linordered-field*(5) *atLeastAtMost-iff less-irrefl not-le st*(2))
apply *auto*
done
ultimately have *continuous-on* ({a<..<b} − *insert* c ($s \cup t$)) (λx. *vector-derivative* (λx. *if* $x \leq c$ *then* $f\ x$ *else* $g\ x$) $(at\ x)$)
apply (*rule continuous-on-subset* [*OF continuous-on-open-Un*], *auto*)
done
} note $* = this$
have *continuous-on* ({a<..<b} − *insert* c ($s \cup t$)) (λx. *vector-derivative* (λx. *if* $x \leq c$ *then* $f\ x$ *else* $g\ x$) $(at\ x)$)
using *st*
by (*auto simp*: *C1-differentiable-on-eq elim*!: *continuous-on-subset intro*: $*$)
ultimately have ∃ s. *finite* $s \wedge$ ((λx. *if* $x \leq c$ *then* $f\ x$ *else* $g\ x$) *C1-differentiable-on* {a..b} − s)
apply (*rule-tac* x={a,b,c} ∪ $s \cup t$ **in** *exI*)
using *st* **by** (*auto simp*: *C1-differentiable-on-eq elim*!: *continuous-on-subset*)
with *cab* **show** *?thesis*
by (*simp add*: *piecewise-C1-differentiable-on-def*)
qed

lemma *piecewise-C1-differentiable-neg*:
f *piecewise-C1-differentiable-on* $s \Longrightarrow$ (λx. −($f\ x$)) *piecewise-C1-differentiable-on* s
unfolding *piecewise-C1-differentiable-on-def*
by (*auto intro*!: *continuous-on-minus C1-differentiable-on-minus*)

lemma *piecewise-C1-differentiable-add*:
assumes f *piecewise-C1-differentiable-on* i
g *piecewise-C1-differentiable-on* i
shows (λx. $f\ x + g\ x$) *piecewise-C1-differentiable-on* i

proof −
obtain *s t* **where** *st*: *finite s finite t*
f C1-differentiable-on (*i*−*s*)
g C1-differentiable-on (*i*−*t*)
using *assms* **by** (*auto simp*: *piecewise-C1-differentiable-on-def*)
then have *finite* (*s* ∪ *t*) ∧ (λ*x*. *f x* + *g x*) *C1-differentiable-on i* − (*s* ∪ *t*)
by (*auto intro*: *C1-differentiable-on-add elim*!: *C1-differentiable-on-subset*)
moreover have *continuous-on i f continuous-on i g*
using *assms piecewise-C1-differentiable-on-def* **by** *auto*
ultimately show *?thesis*
by (*auto simp*: *piecewise-C1-differentiable-on-def continuous-on-add*)
qed

lemma *piecewise-C1-differentiable-diff*:
⟦*f piecewise-C1-differentiable-on s*; *g piecewise-C1-differentiable-on s*⟧
⟹ (λ*x*. *f x* − *g x*) *piecewise-C1-differentiable-on s*
unfolding *diff-conv-add-uminus*
by (*metis piecewise-C1-differentiable-add piecewise-C1-differentiable-neg*)

lemma *piecewise-C1-differentiable-D1*:
fixes *g1* :: *real* ⇒ *'a*::*real-normed-field*
assumes (*g1* +++ *g2*) *piecewise-C1-differentiable-on* {*0..1*}
shows *g1 piecewise-C1-differentiable-on* {*0..1*}
proof −
obtain *s* **where** *finite s*
and *co12*: *continuous-on* ({*0..1*} − *s*) (λ*x*. *vector-derivative* (*g1* +++ *g2*) (*at x*))
and *g12D*: ∀ *x*∈{*0..1*} − *s*. *g1* +++ *g2 differentiable at x*
using *assms* **by** (*auto simp*: *piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
then have *g1D*: *g1 differentiable at x* **if** *x* ∈ {*0..1*} − *insert 1* (*op* ∗ *2* ‘ *s*) **for** *x*
apply (*rule-tac d*=*dist* (*x*/*2*) (*1*/*2*) **and** *f* = (*g1* +++ *g2*) *o* (*op*∗(*inverse 2*))
in *differentiable-transform-within*)
using *that*
apply (*simp-all add*: *dist-real-def joinpaths-def*)
apply (*rule differentiable-chain-at derivative-intros* | *force*)+
done
have [*simp*]: *vector-derivative* (*g1* ∘ *op* ∗ *2*) (*at* (*x*/*2*)) = *2* ∗*R vector-derivative* *g1* (*at x*)
if *x* ∈ {*0..1*} − *insert 1* (*op* ∗ *2* ‘ *s*) **for** *x*
apply (*subst vector-derivative-chain-at*)
using *that*
apply (*rule derivative-eq-intros g1D* | *simp*)+
done
have *continuous-on* ({*0..1*/*2*} − *insert* (*1*/*2*) *s*) (λ*x*. *vector-derivative* (*g1* +++ *g2*) (*at x*))
using *co12* **by** (*rule continuous-on-subset*) *force*
then have *coDhalf*: *continuous-on* ({*0..1*/*2*} − *insert* (*1*/*2*) *s*) (λ*x*. *vector-derivative* (*g1 o op*∗*2*) (*at x*))
apply (*rule continuous-on-eq* [*OF - vector-derivative-at*])

```
   apply (rule-tac f=g1 o op*2 and d=dist x (1/2) in has-vector-derivative-transform-within)
   apply (simp-all add: dist-norm joinpaths-def vector-derivative-works [symmetric])
    apply (force intro: g1D differentiable-chain-at)
    apply auto
    done
  have continuous-on ({0..1} - insert 1 (op * 2 ` s))
                     ((λx. 1/2 * vector-derivative (g1 o op*2) (at x)) o op*(1/2))
    apply (rule continuous-intros)+
    using coDhalf
    apply (simp add: scaleR-conv-of-real image-set-diff image-image)
    done
 then have con-g1: continuous-on ({0..1} - insert 1 (op * 2 ` s)) (λx. vector-derivative
g1 (at x))
    by (rule continuous-on-eq) (simp add: scaleR-conv-of-real)
  have continuous-on {0..1} g1
    using continuous-on-joinpaths-D1 assms piecewise-C1-differentiable-on-def by
blast
  with ⟨finite s⟩ show ?thesis
   apply (clarsimp simp add: piecewise-C1-differentiable-on-def C1-differentiable-on-eq)
    apply (rule-tac x=insert 1 ((op*2)`s) in exI)
    apply (simp add: g1D con-g1)
  done
qed

lemma piecewise-C1-differentiable-D2:
  fixes g2 :: real ⇒ 'a::real-normed-field
   assumes (g1 +++ g2) piecewise-C1-differentiable-on {0..1} pathfinish g1 =
pathstart g2
    shows g2 piecewise-C1-differentiable-on {0..1}
proof -
  obtain s where finite s
          and co12: continuous-on ({0..1} - s) (λx. vector-derivative (g1 +++
g2) (at x))
          and g12D: ∀x∈{0..1} - s. g1 +++ g2 differentiable at x
   using assms by (auto simp: piecewise-C1-differentiable-on-def C1-differentiable-on-eq)
  then have g2D: g2 differentiable at x if x ∈ {0..1} - insert 0 ((λx. 2*x-1) `
s) for x
     apply (rule-tac d=dist ((x+1)/2) (1/2) and f = (g1 +++ g2) o (λx.
(x+1)/2) in differentiable-transform-within)
    using that
    apply (simp-all add: dist-real-def joinpaths-def)
    apply (auto simp: dist-real-def joinpaths-def field-simps)
    apply (rule differentiable-chain-at derivative-intros | force)+
    apply (drule-tac x= (x + 1) / 2 in bspec, force simp: divide-simps)
    apply assumption
    done
   have [simp]: vector-derivative (g2 ∘ (λx. 2*x-1)) (at ((x+1)/2)) = 2 *R
vector-derivative g2 (at x)
            if x ∈ {0..1} - insert 0 ((λx. 2*x-1) ` s) for x
```

using *that* **by** (*auto simp*: *vector-derivative-chain-at divide-simps g2D*)
have *continuous-on* ({*1/2..1*} − *insert* (*1/2*) *s*) (λ*x*. *vector-derivative* (*g1* +++ *g2*) (*at x*))
using *co12* **by** (*rule continuous-on-subset*) *force*
then have *coDhalf*: *continuous-on* ({*1/2..1*} − *insert* (*1/2*) *s*) (λ*x*. *vector-derivative* (*g2 o* (λ*x*. *2*∗*x*−*1*)) (*at x*))
apply (*rule continuous-on-eq* [*OF - vector-derivative-at*])
apply (*rule-tac f*=*g2 o* (λ*x*. *2*∗*x*−*1*) **and** *d*=*dist* (*3/4*) ((*x*+*1*)*/2*) **in** *has-vector-derivative-transform-within*
apply (*auto simp*: *dist-real-def field-simps joinpaths-def vector-derivative-works* [*symmetric*]
intro!: *g2D differentiable-chain-at*)
done
have [*simp*]: ((λ*x*. (*x* + *1*) / *2*) ‘ ({*0..1*} − *insert 0* ((λ*x*. *2* ∗ *x* − *1*) ‘ *s*))) = ({*1/2..1*} − *insert* (*1/2*) *s*)
apply (*simp add*: *image-set-diff inj-on-def image-image*)
apply (*auto simp*: *image-affinity-atLeastAtMost-div add-divide-distrib*)
done
have *continuous-on* ({*0..1*} − *insert 0* ((λ*x*. *2*∗*x*−*1*) ‘ *s*))
((λ*x*. *1/2* ∗ *vector-derivative* (*g2* ∘ (λ*x*. *2*∗*x*−*1*)) (*at x*)) *o* (λ*x*. (*x*+*1*)*/2*))
by (*rule continuous-intros* | *simp add*: *coDhalf*)+
then have *con-g2*: *continuous-on* ({*0..1*} − *insert 0* ((λ*x*. *2*∗*x*−*1*) ‘ *s*)) (λ*x*. *vector-derivative g2* (*at x*))
by (*rule continuous-on-eq*) (*simp add*: *scaleR-conv-of-real*)
have *continuous-on* {*0..1*} *g2*
using *continuous-on-joinpaths-D2 assms piecewise-C1-differentiable-on-def* **by** *blast*
with ‹*finite s*› **show** *?thesis*
apply (*clarsimp simp add*: *piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
apply (*rule-tac x*=*insert 0* ((λ*x*. *2* ∗ *x* − *1*) ‘ *s*) **in** *exI*)
apply (*simp add*: *g2D con-g2*)
done
qed

52.2 Valid paths, and their start and finish

lemma *Diff-Un-eq*: *A* − (*B* ∪ *C*) = *A* − *B* − *C*
by *blast*

definition *valid-path* :: (*real* ⇒ *'a* :: *real-normed-vector*) ⇒ *bool*
where *valid-path f* ≡ *f piecewise-C1-differentiable-on* {*0..1*::*real*}

definition *closed-path* :: (*real* ⇒ *'a* :: *real-normed-vector*) ⇒ *bool*
where *closed-path g* ≡ *g 0* = *g 1*

52.2.1 In particular, all results for paths apply

lemma *valid-path-imp-path*: *valid-path g* ⟹ *path g*
by (*simp add*: *path-def piecewise-C1-differentiable-on-def valid-path-def*)

lemma *connected-valid-path-image*: *valid-path g* $\Longrightarrow$ *connected*(*path-image g*)
by (*metis connected-path-image valid-path-imp-path*)

lemma *compact-valid-path-image*: *valid-path g* $\Longrightarrow$ *compact*(*path-image g*)
by (*metis compact-path-image valid-path-imp-path*)

lemma *bounded-valid-path-image*: *valid-path g* $\Longrightarrow$ *bounded*(*path-image g*)
by (*metis bounded-path-image valid-path-imp-path*)

lemma *closed-valid-path-image*: *valid-path g* $\Longrightarrow$ *closed*(*path-image g*)
by (*metis closed-path-image valid-path-imp-path*)

52.3 Contour Integrals along a path

This definition is for complex numbers only, and does not generalise to line integrals in a vector field

piecewise differentiable function on [0,1]

definition *has-contour-integral* :: (*complex* $\Rightarrow$ *complex*) $\Rightarrow$ *complex* $\Rightarrow$ (*real* $\Rightarrow$ *complex*) $\Rightarrow$ *bool*
(**infixr** *has′-contour′-integral 50*)
where (*f has-contour-integral i*) *g* $\equiv$
((*λx*. *f*(*g x*) $*$ *vector-derivative g* (*at x within* {*0..1*}))
has-integral i) {*0..1*}

definition *contour-integrable-on*
(**infixr** *contour′-integrable′-on 50*)
where *f contour-integrable-on g* $\equiv$ $\exists$ *i*. (*f has-contour-integral i*) *g*

definition *contour-integral*
where *contour-integral g f* $\equiv$ @*i*. (*f has-contour-integral i*) *g*

lemma *contour-integral-unique*: (*f has-contour-integral i*) *g* $\Longrightarrow$ *contour-integral g f* $=$ *i*
by (*auto simp*: *contour-integral-def has-contour-integral-def integral-def* [*symmetric*])

corollary *has-contour-integral-eqpath*:
⟦(*f has-contour-integral y*) *p*; *f contour-integrable-on* γ;
contour-integral p f $=$ *contour-integral* γ *f*⟧
$\Longrightarrow$ (*f has-contour-integral y*) γ
using *contour-integrable-on-def contour-integral-unique* **by** *auto*

lemma *has-contour-integral-integral*:
f contour-integrable-on i $\Longrightarrow$ (*f has-contour-integral* (*contour-integral i f*)) *i*
by (*metis contour-integral-unique contour-integrable-on-def*)

lemma *has-contour-integral-unique*:
(*f has-contour-integral i*) *g* $\Longrightarrow$ (*f has-contour-integral j*) *g* $\Longrightarrow$ *i* $=$ *j*
using *has-integral-unique*

by (*auto simp*: *has-contour-integral-def*)

lemma *has-contour-integral-integrable*: (*f has-contour-integral i*) *g* ⟹ *f contour-integrable-on g*
using *contour-integrable-on-def* **by** *blast*

lemma *vector-derivative-within-interior*:
⟦*x* ∈ *interior s*; *NO-MATCH UNIV s*⟧
⟹ *vector-derivative f* (*at x within s*) = *vector-derivative f* (*at x*)
apply (*simp add*: *vector-derivative-def has-vector-derivative-def has-derivative-def netlimit-within-interior*)
apply (*subst lim-within-interior*, *auto*)
done

lemma *has-integral-localized-vector-derivative*:
((λ*x*. *f* (*g x*) ∗ *vector-derivative g* (*at x within* {*a*..*b*})) *has-integral i*) {*a*..*b*}
⟷
((λ*x*. *f* (*g x*) ∗ *vector-derivative g* (*at x*)) *has-integral i*) {*a*..*b*}
proof −
have {*a*..*b*} − {*a*,*b*} = *interior* {*a*..*b*}
by (*simp add*: *atLeastAtMost-diff-ends*)
show *?thesis*
apply (*rule has-integral-spike-eq* [*of* {*a*,*b*}])
apply (*auto simp*: *vector-derivative-within-interior*)
done
qed

lemma *integrable-on-localized-vector-derivative*:
(λ*x*. *f* (*g x*) ∗ *vector-derivative g* (*at x within* {*a*..*b*})) *integrable-on* {*a*..*b*} ⟷
(λ*x*. *f* (*g x*) ∗ *vector-derivative g* (*at x*)) *integrable-on* {*a*..*b*}
by (*simp add*: *integrable-on-def has-integral-localized-vector-derivative*)

lemma *has-contour-integral*:
(*f has-contour-integral i*) *g* ⟷
((λ*x*. *f* (*g x*) ∗ *vector-derivative g* (*at x*)) *has-integral i*) {*0*..*1*}
by (*simp add*: *has-integral-localized-vector-derivative has-contour-integral-def*)

lemma *contour-integrable-on*:
f contour-integrable-on g ⟷
(λ*t*. *f*(*g t*) ∗ *vector-derivative g* (*at t*)) *integrable-on* {*0*..*1*}
by (*simp add*: *has-contour-integral integrable-on-def contour-integrable-on-def*)

52.4 Reversing a path

lemma *valid-path-imp-reverse*:
assumes *valid-path g*
shows *valid-path*(*reversepath g*)

```
proof −
  obtain s where finite s g C1-differentiable-on ({0..1} − s)
    using assms by (auto simp: valid-path-def piecewise-C1-differentiable-on-def)
  then have finite (op − 1 ‘ s) (reversepath g C1-differentiable-on ({0..1} − op
− 1 ‘ s))
    apply (auto simp: reversepath-def)
    apply (rule C1-differentiable-compose [of λx::real. 1−x - g, unfolded o-def])
    apply (auto simp: C1-differentiable-on-eq)
    apply (rule continuous-intros, force)
    apply (force elim!: continuous-on-subset)
    apply (simp add: finite-vimageI inj-on-def)
    done
  then show ?thesis using assms
   by (auto simp: valid-path-def piecewise-C1-differentiable-on-def path-def [symmetric])
qed

lemma valid-path-reversepath: valid-path(reversepath g) ⟷ valid-path g
  using valid-path-imp-reverse by force

lemma has-contour-integral-reversepath:
  assumes valid-path g (f has-contour-integral i) g
    shows (f has-contour-integral (−i)) (reversepath g)
proof −
  { fix s x
    assume xs: g C1-differentiable-on ({0..1} − s) x ∉ op − 1 ‘ s 0 ≤ x x ≤ 1
      have vector-derivative (λx. g (1 − x)) (at x within {0..1}) =
            − vector-derivative g (at (1 − x) within {0..1})
      proof −
        obtain f′ where f′: (g has-vector-derivative f′) (at (1 − x))
          using xs
          by (force simp: has-vector-derivative-def C1-differentiable-on-def)
        have (g o (λx. 1 − x) has-vector-derivative −1 ∗R f′) (at x)
          apply (rule vector-diff-chain-within)
          apply (intro vector-diff-chain-within derivative-eq-intros | simp)+
          apply (rule has-vector-derivative-at-within [OF f′])
          done
        then have mf′: ((λx. g (1 − x)) has-vector-derivative −f′) (at x)
          by (simp add: o-def)
        show ?thesis
          using xs
       by (auto simp: vector-derivative-at-within-ivl [OF mf′] vector-derivative-at-within-ivl
[OF f′])
      qed
  } note ∗ = this
  have 01: {0..1::real} = cbox 0 1
    by simp
  show ?thesis using assms
    apply (auto simp: has-contour-integral-def)
    apply (drule has-integral-affinity01 [where m= −1 and c=1])
```

apply (*auto simp*: *reversepath-def valid-path-def piecewise-C1-differentiable-on-def*)
apply (*drule has-integral-neg*)
apply (*rule-tac s* = (λ*x*. *1* − *x*) ‘ *s* **in** *has-integral-spike-finite*)
apply (*auto simp*: ∗)
done
qed

lemma *contour-integrable-reversepath*:
valid-path g ⟹ *f contour-integrable-on g* ⟹ *f contour-integrable-on* (*reversepath g*)
using *has-contour-integral-reversepath contour-integrable-on-def* **by** *blast*

lemma *contour-integrable-reversepath-eq*:
valid-path g ⟹ (*f contour-integrable-on* (*reversepath g*) ⟷ *f contour-integrable-on g*)
using *contour-integrable-reversepath valid-path-reversepath* **by** *fastforce*

lemma *contour-integral-reversepath*:
⟦*valid-path g*; *f contour-integrable-on g*⟧ ⟹ *contour-integral* (*reversepath g*) *f* = −(*contour-integral g f*)
using *has-contour-integral-reversepath contour-integrable-on-def contour-integral-unique* **by** *blast*

52.5 Joining two paths together

lemma *valid-path-join*:
assumes *valid-path g1 valid-path g2 pathfinish g1* = *pathstart g2*
shows *valid-path*(*g1* +++ *g2*)
proof −
have *g1 1* = *g2 0*
using *assms* **by** (*auto simp*: *pathfinish-def pathstart-def*)
moreover have (*g1 o* (λ*x*. *2*∗*x*)) *piecewise-C1-differentiable-on* {*0..1/2*}
apply (*rule piecewise-C1-differentiable-compose*)
using *assms*
apply (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def continuous-on-joinpaths*)
apply (*rule continuous-intros* | *simp*)+
apply (*force intro*: *finite-vimageI* [**where** *h* = *op*∗*2*] *inj-onI*)
done
moreover have (*g2 o* (λ*x*. *2*∗*x*−*1*)) *piecewise-C1-differentiable-on* {*1/2..1*}
apply (*rule piecewise-C1-differentiable-compose*)
using *assms* **unfolding** *valid-path-def piecewise-C1-differentiable-on-def*
by (*auto intro*!: *continuous-intros finite-vimageI* [**where** *h* = (λ*x*. *2*∗*x* − *1*)] *inj-onI*
simp: *image-affinity-atLeastAtMost-diff continuous-on-joinpaths*)
ultimately show *?thesis*
apply (*simp only*: *valid-path-def continuous-on-joinpaths joinpaths-def*)
apply (*rule piecewise-C1-differentiable-cases*)
apply (*auto simp*: *o-def*)
done

qed

lemma *valid-path-join-D1*:
fixes *g1* :: *real* ⇒ *'a*::*real-normed-field*
shows *valid-path* (*g1* +++ *g2*) ⟹ *valid-path g1*
unfolding *valid-path-def*
by (*rule piecewise-C1-differentiable-D1*)

lemma *valid-path-join-D2*:
fixes *g2* :: *real* ⇒ *'a*::*real-normed-field*
shows ⟦*valid-path* (*g1* +++ *g2*); *pathfinish g1* = *pathstart g2*⟧ ⟹ *valid-path g2*
unfolding *valid-path-def*
by (*rule piecewise-C1-differentiable-D2*)

lemma *valid-path-join-eq* [*simp*]:
fixes *g2* :: *real* ⇒ *'a*::*real-normed-field*
shows *pathfinish g1* = *pathstart g2* ⟹ (*valid-path*(*g1* +++ *g2*) ⟷ *valid-path g1* ∧ *valid-path g2*)
using *valid-path-join-D1 valid-path-join-D2 valid-path-join* **by** *blast*

lemma *has-contour-integral-join*:
assumes (*f has-contour-integral i1*) *g1* (*f has-contour-integral i2*) *g2*
valid-path g1 valid-path g2
shows (*f has-contour-integral* (*i1* + *i2*)) (*g1* +++ *g2*)
proof −
obtain *s1 s2*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
and *s2*: *finite s2* ∀ *x*∈{*0..1*} − *s2*. *g2 differentiable at x*
using *assms*
by (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
have *1*: ((λ*x*. *f* (*g1 x*) ∗ *vector-derivative g1* (*at x*)) *has-integral i1*) {*0..1*}
and *2*: ((λ*x*. *f* (*g2 x*) ∗ *vector-derivative g2* (*at x*)) *has-integral i2*) {*0..1*}
using *assms*
by (*auto simp*: *has-contour-integral*)
have *i1*: ((λ*x*. (*2*∗*f* (*g1* (*2*∗*x*))) ∗ *vector-derivative g1* (*at* (*2*∗*x*))) *has-integral i1*) {*0..1/2*}
and *i2*: ((λ*x*. (*2*∗*f* (*g2* (*2*∗*x* − *1*))) ∗ *vector-derivative g2* (*at* (*2*∗*x* − *1*))) *has-integral i2*) {*1/2..1*}
using *has-integral-affinity01* [*OF 1*, **where** *m*= *2* **and** *c*=*0*, *THEN has-integral-cmul* [**where** *c*=*2*]]
has-integral-affinity01 [*OF 2*, **where** *m*= *2* **and** *c*=−*1*, *THEN has-integral-cmul* [**where** *c*=*2*]]
by (*simp-all only*: *image-affinity-atLeastAtMost-div-diff*, *simp-all add*: *scaleR-conv-of-real mult-ac*)
have *g1*: ⟦*0* ≤ *z*; *z*∗*2* < *1*; *z*∗*2* ∉ *s1*⟧ ⟹
vector-derivative (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z*) =
2 ∗*R* *vector-derivative g1* (*at* (*z*∗*2*)) **for** *z*
apply (*rule vector-derivative-at* [*OF has-vector-derivative-transform-within*

[**where** $f = (\lambda x.\ g1(2{*}x))$ **and** $d = |z - 1/2|]]$)
apply (*simp-all add*: *dist-real-def abs-if split*: *split-if-asm*)
apply (*rule vector-diff-chain-at* [*of* $\lambda x.\ 2{*}x$ *2 - g1*, *simplified o-def*])
apply (*simp add*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
using *s1*
apply (*auto simp*: *algebra-simps vector-derivative-works*)
done
have *g2*: $[\![1 < z{*}2;\ z \le 1;\ z{*}2 - 1 \notin s2]\!] \Longrightarrow$
vector-derivative $(\lambda x.\ \textit{if } x{*}2 \le 1 \textit{ then } g1\ (2{*}x) \textit{ else } g2\ (2{*}x - 1))\ (\textit{at}$
$z) =$
$2 *_R$ *vector-derivative g2* $(\textit{at}\ (z{*}2 - 1))$ **for** *z*
apply (*rule vector-derivative-at* [*OF has-vector-derivative-transform-within*
[**where** $f = (\lambda x.\ g2\ (2{*}x - 1))$ **and** $d = |z - 1/2|]]$)
apply (*simp-all add*: *dist-real-def abs-if split*: *split-if-asm*)
apply (*rule vector-diff-chain-at* [*of* $\lambda x.\ 2{*}x - 1$ *2 - g2*, *simplified o-def*])
apply (*simp add*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
using *s2*
apply (*auto simp*: *algebra-simps vector-derivative-works*)
done
have $((\lambda x.\ f\ ((g1 +\!\!+\!\!+ g2)\ x) * \textit{vector-derivative}\ (g1 +\!\!+\!\!+ g2)\ (\textit{at}\ x))\ \textit{has-integral}$
i1) $\{0..1/2\}$
apply (*rule has-integral-spike-finite* [*OF - - i1*, *of insert* $(1/2)$ $(op{*}2\ -`\ s1)$])
using *s1*
apply (*force intro*: *finite-vimageI* [**where** $h = op{*}2$] *inj-onI*)
apply (*clarsimp simp add*: *joinpaths-def scaleR-conv-of-real mult-ac g1*)
done
moreover have $((\lambda x.\ f\ ((g1 +\!\!+\!\!+ g2)\ x) * \textit{vector-derivative}\ (g1 +\!\!+\!\!+ g2)\ (\textit{at}$
$x))\ \textit{has-integral i2})\ \{1/2..1\}$
apply (*rule has-integral-spike-finite* [*OF - - i2*, *of insert* $(1/2)$ $((\lambda x.\ 2{*}x{-}1)$
$-`\ s2)$])
using *s2*
apply (*force intro*: *finite-vimageI* [**where** $h = \lambda x.\ 2{*}x{-}1$] *inj-onI*)
apply (*clarsimp simp add*: *joinpaths-def scaleR-conv-of-real mult-ac g2*)
done
ultimately
show *?thesis*
apply (*simp add*: *has-contour-integral*)
apply (*rule has-integral-combine* [**where** $c = 1/2$], *auto*)
done
qed

lemma *contour-integrable-joinI*:
assumes *f contour-integrable-on g1 f contour-integrable-on g2*
valid-path g1 valid-path g2
shows *f contour-integrable-on* $(g1 +\!\!+\!\!+ g2)$
using *assms*
by (*meson has-contour-integral-join contour-integrable-on-def*)

lemma *contour-integrable-joinD1*:

assumes *f contour-integrable-on (g1 +++ g2) valid-path g1*
shows *f contour-integrable-on g1*
proof −
obtain *s1*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
have (λ*x*. *f* ((*g1 +++ g2*) (*x*/*2*)) ∗ *vector-derivative* (*g1 +++ g2*) (*at* (*x*/*2*))) *integrable-on* {*0..1*}
using *assms*
apply (*auto simp*: *contour-integrable-on*)
apply (*drule integrable-on-subcbox* [**where** *a*=*0* **and** *b*=*1*/*2*])
apply (*auto intro*: *integrable-affinity* [*of - 0 1*/*2*::*real 1*/*2 0*, *simplified*])
done
then have ∗: (λ*x*. (*f* ((*g1 +++ g2*) (*x*/*2*))/*2*) ∗ *vector-derivative* (*g1 +++ g2*) (*at* (*x*/*2*))) *integrable-on* {*0..1*}
by (*auto dest*: *integrable-cmul* [**where** *c*=*1*/*2*] *simp*: *scaleR-conv-of-real*)
have *g1*: ⟦*0* < *z*; *z* < *1*; *z* ∉ *s1*⟧ ⟹
vector-derivative (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at* (*z*/*2*)) =
2 ∗$_R$ *vector-derivative g1* (*at z*) **for** *z*
apply (*rule vector-derivative-at* [*OF has-vector-derivative-transform-within* [**where** *f* = (λ*x*. *g1*(*2*∗*x*)) **and** *d* = |(*z*−*1*)/*2*|]])
apply (*simp-all add*: *field-simps dist-real-def abs-if split*: *split-if-asm*)
apply (*rule vector-diff-chain-at* [*of* λ*x*. *x*∗*2 2 - g1*, *simplified o-def*])
using *s1*
apply (*auto simp*: *vector-derivative-works has-vector-derivative-def has-derivative-def bounded-linear-mult-left*)
done
show *?thesis*
using *s1*
apply (*auto simp*: *contour-integrable-on*)
apply (*rule integrable-spike-finite* [*of* {*0*,*1*} ∪ *s1*, *OF - - ∗*])
apply (*auto simp*: *joinpaths-def scaleR-conv-of-real g1*)
done
qed

lemma *contour-integrable-joinD2*:
assumes *f contour-integrable-on (g1 +++ g2) valid-path g2*
shows *f contour-integrable-on g2*
proof −
obtain *s2*
where *s2*: *finite s2* ∀ *x*∈{*0..1*} − *s2*. *g2 differentiable at x*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
have (λ*x*. *f* ((*g1 +++ g2*) (*x*/*2* + *1*/*2*)) ∗ *vector-derivative* (*g1 +++ g2*) (*at* (*x*/*2* + *1*/*2*))) *integrable-on* {*0..1*}
using *assms*
apply (*auto simp*: *contour-integrable-on*)

apply (*drule integrable-on-subcbox* [**where** *a=1/2* **and** *b=1*], *auto*)
apply (*drule integrable-affinity* [*of - 1/2::real 1 1/2 1/2*, *simplified*])
apply (*simp add*: *image-affinity-atLeastAtMost-diff*)
done
then have ∗: (λ*x*. (*f* ((*g1* +++ *g2*) (*x*/2 + 1/2))/2) ∗ *vector-derivative* (*g1* +++ *g2*) (*at* (*x*/2 + 1/2)))
integrable-on {0..1}
by (*auto dest*: *integrable-cmul* [**where** *c=1/2*] *simp*: *scaleR-conv-of-real*)
have *g2*: ⟦0 < *z*; *z* < 1; *z* ∉ *s2*⟧ ⟹
vector-derivative (λ*x*. *if x*∗2 ≤ 1 *then g1* (2∗*x*) *else g2* (2∗*x* − 1)) (*at* (*z*/2+1/2)) =
2 ∗$_R$ *vector-derivative g2* (*at z*) **for** *z*
apply (*rule vector-derivative-at* [*OF has-vector-derivative-transform-within* [**where** *f* = (λ*x*. *g2*(2∗*x*−1)) **and** *d* = |*z*/2|]])
apply (*simp-all add*: *field-simps dist-real-def abs-if split*: *split-if-asm*)
apply (*rule vector-diff-chain-at* [*of* λ*x*. *x*∗2−1 2 - *g2*, *simplified o-def*])
using *s2*
apply (*auto simp*: *has-vector-derivative-def has-derivative-def bounded-linear-mult-left*
vector-derivative-works add-divide-distrib)
done
show *?thesis*
using *s2*
apply (*auto simp*: *contour-integrable-on*)
apply (*rule integrable-spike-finite* [*of* {0,1} ∪ *s2*, *OF - - ∗*])
apply (*auto simp*: *joinpaths-def scaleR-conv-of-real g2*)
done
qed

lemma *contour-integrable-join* [*simp*]:
shows
⟦*valid-path g1*; *valid-path g2*⟧
⟹ *f contour-integrable-on* (*g1* +++ *g2*) ⟷ *f contour-integrable-on g1* ∧ *f contour-integrable-on g2*
using *contour-integrable-joinD1 contour-integrable-joinD2 contour-integrable-joinI*
by *blast*

lemma *contour-integral-join* [*simp*]:
shows
⟦*f contour-integrable-on g1*; *f contour-integrable-on g2*; *valid-path g1*; *valid-path g2*⟧
⟹ *contour-integral* (*g1* +++ *g2*) *f* = *contour-integral g1 f* + *contour-integral g2 f*
by (*simp add*: *has-contour-integral-integral has-contour-integral-join contour-integral-unique*)

52.6 Shifting the starting point of a (closed) path

lemma *shiftpath-alt-def*: *shiftpath a f* = (λ*x*. *if x* ≤ 1−*a then f* (*a* + *x*) *else f* (*a* + *x* − 1))
by (*auto simp*: *shiftpath-def*)

lemma *valid-path-shiftpath* [*intro*]:
assumes *valid-path g pathfinish g = pathstart g a ∈ {0..1}*
shows *valid-path*(*shiftpath a g*)
using *assms*
apply (*auto simp*: *valid-path-def shiftpath-alt-def*)
apply (*rule piecewise-C1-differentiable-cases*)
apply (*auto simp*: *algebra-simps*)
apply (*rule piecewise-C1-differentiable-affine* [*of g 1 a*, *simplified o-def scaleR-one*])
apply (*auto simp*: *pathfinish-def pathstart-def elim*: *piecewise-C1-differentiable-on-subset*)
apply (*rule piecewise-C1-differentiable-affine* [*of g 1 a−1*, *simplified o-def scaleR-one algebra-simps*])
apply (*auto simp*: *pathfinish-def pathstart-def elim*: *piecewise-C1-differentiable-on-subset*)
done

lemma *has-contour-integral-shiftpath*:
assumes *f*: (*f has-contour-integral i*) *g valid-path g*
and *a*: *a ∈ {0..1}*
shows (*f has-contour-integral i*) (*shiftpath a g*)
proof −
obtain *s*
where *s*: *finite s* **and** *g*: *∀ x∈{0..1} − s. g differentiable at x*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
have ∗: ((*λx. f* (*g x*) ∗ *vector-derivative g* (*at x*)) *has-integral i*) {*0..1*}
using *assms* **by** (*auto simp*: *has-contour-integral*)
then have *i*: *i* = *integral* {*a..1*} (*λx. f* (*g x*) ∗ *vector-derivative g* (*at x*)) +
integral {*0..a*} (*λx. f* (*g x*) ∗ *vector-derivative g* (*at x*))
apply (*rule has-integral-unique*)
apply (*subst add.commute*)
apply (*subst Integration.integral-combine*)
using *assms* ∗ *integral-unique* **by** *auto*
{ fix *x*
have *0 ≤ x ⟹ x + a < 1 ⟹ x ∉* (*λx. x − a*) ‘ *s ⟹*
vector-derivative (*shiftpath a g*) (*at x*) = *vector-derivative g* (*at* (*x + a*))
unfolding *shiftpath-def*
apply (*rule vector-derivative-at* [*OF has-vector-derivative-transform-within* [**where** *f* = (*λx. g*(*a*+*x*)) **and** *d* = *dist*(1−*a*) *x*]])
apply (*auto simp*: *field-simps dist-real-def abs-if split*: *split-if-asm*)
apply (*rule vector-diff-chain-at* [*of λx. x*+*a 1 - g*, *simplified o-def scaleR-one*])
apply (*intro derivative-eq-intros* | *simp*)+
using *g*
apply (*drule-tac x*=*x*+*a* **in** *bspec*)
using *a* **apply** (*auto simp*: *has-vector-derivative-def vector-derivative-works image-def add.commute*)
done
} note *vd1* = *this*
{ fix *x*
have *1 < x + a ⟹ x ≤ 1 ⟹ x ∉* (*λx. x − a* + *1*) ‘ *s ⟹*

```
        vector-derivative (shiftpath a g) (at x) = vector-derivative g (at (x + a −
1))
      unfolding shiftpath-def
       apply (rule vector-derivative-at [OF has-vector-derivative-transform-within
[where f = (λx. g(a+x−1)) and d = dist (1−a) x]])
        apply (auto simp: field-simps dist-real-def abs-if split: split-if-asm)
     apply (rule vector-diff-chain-at [of λx. x+a−1 1 - g, simplified o-def scaleR-one])
       apply (intro derivative-eq-intros | simp)+
      using g
      apply (drule-tac x=x+a−1 in bspec)
       using a apply (auto simp: has-vector-derivative-def vector-derivative-works
image-def add.commute)
      done
  } note vd2 = this
  have va1: (λx. f (g x) ∗ vector-derivative g (at x)) integrable-on ({a..1})
    using ∗ a   by (fastforce intro: integrable-subinterval-real)
  have v0a: (λx. f (g x) ∗ vector-derivative g (at x)) integrable-on ({0..a})
    apply (rule integrable-subinterval-real)
    using ∗ a by auto
  have ((λx. f (shiftpath a g x) ∗ vector-derivative (shiftpath a g) (at x))
       has-integral  integral {a..1} (λx. f (g x) ∗ vector-derivative g (at x)))  {0..1
− a}
    apply (rule has-integral-spike-finite
                [where s = {1−a} ∪ (λx. x−a) ‘ s and f = λx. f(g(a+x)) ∗
vector-derivative g (at(a+x))])
      using s apply blast
     using a apply (auto simp: algebra-simps vd1)
     apply (force simp: shiftpath-def add.commute)
   using has-integral-affinity [where m=1 and c=a, simplified, OF integrable-integral
[OF va1]]
    apply (simp add: image-affinity-atLeastAtMost-diff [where m=1 and c=a,
simplified] add.commute)
    done
  moreover
  have ((λx. f (shiftpath a g x) ∗ vector-derivative (shiftpath a g) (at x))
       has-integral  integral {0..a} (λx. f (g x) ∗ vector-derivative g (at x)))  {1
− a..1}
    apply (rule has-integral-spike-finite
              [where s = {1−a} ∪ (λx. x−a+1) ‘ s and f = λx. f(g(a+x−1)) ∗
vector-derivative g (at(a+x−1))])
      using s apply blast
     using a apply (auto simp: algebra-simps vd2)
     apply (force simp: shiftpath-def add.commute)
   using has-integral-affinity [where m=1 and c=a−1, simplified, OF integrable-integral
[OF v0a]]
     apply (simp add: image-affinity-atLeastAtMost [where m=1 and c=1−a,
simplified])
    apply (simp add: algebra-simps)
    done
```

ultimately show *?thesis*
using *a*
by (*auto simp*: *i has-contour-integral intro*: *has-integral-combine* [**where** *c* = *1−a*])
qed

lemma *has-contour-integral-shiftpath-D*:
assumes (*f has-contour-integral i*) (*shiftpath a g*)
valid-path g pathfinish g = *pathstart g a* ∈ {*0..1*}
shows (*f has-contour-integral i*) *g*
proof −
obtain *s*
where *s*: *finite s* **and** *g*: ∀ *x*∈{*0..1*} − *s*. *g differentiable at x*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq*)
{ fix *x*
assume *x*: *0* < *x x* < *1 x* ∉ *s*
then have *gx*: *g differentiable at x*
using *g* **by** *auto*
have *vector-derivative g* (*at x within* {*0..1*}) =
vector-derivative (*shiftpath* (*1* − *a*) (*shiftpath a g*)) (*at x within* {*0..1*})
apply (*rule vector-derivative-at-within-ivl*
[*OF has-vector-derivative-transform-within-open*
[**where** *f* = (*shiftpath* (*1* − *a*) (*shiftpath a g*)) **and** *s* = {*0<..<1*}−*s*]])
using *s g assms x*
apply (*auto simp*: *finite-imp-closed open-Diff shiftpath-shiftpath*
vector-derivative-within-interior vector-derivative-works [*symmetric*])
apply (*rule differentiable-transform-within* [*OF gx*, *of min x* (*1−x*)])
apply (*auto simp*: *dist-real-def shiftpath-shiftpath abs-if split*: *split-if-asm*)
done
} note *vd* = *this*
have *fi*: (*f has-contour-integral i*) (*shiftpath* (*1* − *a*) (*shiftpath a g*))
using *assms* **by** (*auto intro*!: *has-contour-integral-shiftpath*)
show *?thesis*
apply (*simp add*: *has-contour-integral-def*)
apply (*rule has-integral-spike-finite* [*of* {*0*,*1*} ∪ *s*, *OF - - fi* [*unfolded has-contour-integral-def*]])
using *s assms vd*
apply (*auto simp*: *Path-Connected.shiftpath-shiftpath*)
done
qed

lemma *has-contour-integral-shiftpath-eq*:
assumes *valid-path g pathfinish g* = *pathstart g a* ∈ {*0..1*}
shows (*f has-contour-integral i*) (*shiftpath a g*) ⟷ (*f has-contour-integral i*) *g*
using *assms has-contour-integral-shiftpath has-contour-integral-shiftpath-D* **by** *blast*

lemma *contour-integral-shiftpath*:
assumes *valid-path g pathfinish g = pathstart g a* ∈ {*0..1*}
shows *contour-integral* (*shiftpath a g*) *f* = *contour-integral g f*
using *assms* **by** (*simp add*: *contour-integral-def has-contour-integral-shiftpath-eq*)

52.7 More about straight-line paths

lemma *has-vector-derivative-linepath-within*:
(*linepath a b has-vector-derivative* (*b* − *a*)) (*at x within s*)
apply (*simp add*: *linepath-def has-vector-derivative-def algebra-simps*)
apply (*rule derivative-eq-intros* | *simp*)+
done

lemma *vector-derivative-linepath-within*:
x ∈ {*0..1*} ⟹ *vector-derivative* (*linepath a b*) (*at x within* {*0..1*}) = *b* − *a*
apply (*rule vector-derivative-within-closed-interval* [*of 0 1*::*real*, *simplified*])
apply (*auto simp*: *has-vector-derivative-linepath-within*)
done

lemma *vector-derivative-linepath-at* [*simp*]: *vector-derivative* (*linepath a b*) (*at x*)
= *b* − *a*
by (*simp add*: *has-vector-derivative-linepath-within vector-derivative-at*)

lemma *valid-path-linepath* [*iff*]: *valid-path* (*linepath a b*)
apply (*simp add*: *valid-path-def piecewise-C1-differentiable-on-def C1-differentiable-on-eq continuous-on-linepath*)
apply (*rule-tac x*={} **in** *exI*)
apply (*simp add*: *differentiable-on-def differentiable-def*)
using *has-vector-derivative-def has-vector-derivative-linepath-within*
apply (*fastforce simp add*: *continuous-on-eq-continuous-within*)
done

lemma *has-contour-integral-linepath*:
shows (*f has-contour-integral i*) (*linepath a b*) ⟷
((λ*x*. *f*(*linepath a b x*) ∗ (*b* − *a*)) *has-integral i*) {*0..1*}
by (*simp add*: *has-contour-integral vector-derivative-linepath-at*)

lemma *linepath-in-path*:
shows *x* ∈ {*0..1*} ⟹ *linepath a b x* ∈ *closed-segment a b*
by (*auto simp*: *segment linepath-def*)

lemma *linepath-image-01*: *linepath a b* ‘ {*0..1*} = *closed-segment a b*
by (*auto simp*: *segment linepath-def*)

lemma *linepath-in-convex-hull*:
fixes *x*::*real*
assumes *a*: *a* ∈ *convex hull s*
and *b*: *b* ∈ *convex hull s*
and *x*: *0*≤*x x*≤*1*

shows *linepath a b x ∈ convex hull s*
apply (*rule closed-segment-subset-convex-hull* [*OF a b*, *THEN subsetD*])
using *x*
apply (*auto simp*: *linepath-image-01* [*symmetric*])
done

lemma *Re-linepath*: *Re*(*linepath* (*of-real a*) (*of-real b*) *x*) = (*1* − *x*)∗*a* + *x*∗*b*
by (*simp add*: *linepath-def*)

lemma *Im-linepath*: *Im*(*linepath* (*of-real a*) (*of-real b*) *x*) = *0*
by (*simp add*: *linepath-def*)

lemma *linepath-of-real*: (*linepath* (*of-real a*) (*of-real b*) *x*) = *of-real* ((*1* − *x*)∗*a* + *x*∗*b*)
by (*simp add*: *scaleR-conv-of-real linepath-def*)

lemma *of-real-linepath*: *of-real* (*linepath a b x*) = *linepath* (*of-real a*) (*of-real b*) *x*
by (*metis linepath-of-real mult.right-neutral of-real-def real-scaleR-def*)

lemma *has-contour-integral-trivial* [*iff*]: (*f has-contour-integral 0*) (*linepath a a*)
by (*simp add*: *has-contour-integral-linepath*)

lemma *contour-integral-trivial* [*simp*]: *contour-integral* (*linepath a a*) *f* = *0*
using *has-contour-integral-trivial contour-integral-unique* **by** *blast*

52.8 Relation to subpath construction

lemma *valid-path-subpath*:
fixes *g* :: *real* ⇒ *'a* :: *real-normed-vector*
assumes *valid-path g u* ∈ {*0..1*} *v* ∈ {*0..1*}
shows *valid-path*(*subpath u v g*)
proof (*cases v*=*u*)
case *True*
then show *?thesis*
unfolding *valid-path-def subpath-def*
by (*force intro*: *C1-differentiable-on-const C1-differentiable-imp-piecewise*)
next
case *False*
have (*g o* (λ*x*. ((*v*−*u*) ∗ *x* + *u*))) *piecewise-C1-differentiable-on* {*0..1*}
apply (*rule piecewise-C1-differentiable-compose*)
apply (*simp add*: *C1-differentiable-imp-piecewise*)
apply (*simp add*: *image-affinity-atLeastAtMost*)
using *assms False*
apply (*auto simp*: *algebra-simps valid-path-def piecewise-C1-differentiable-on-subset*)
apply (*subst Int-commute*)
apply (*auto simp*: *inj-on-def algebra-simps crossproduct-eq finite-vimage-IntI*)
done
then show *?thesis*
by (*auto simp*: *o-def valid-path-def subpath-def*)

qed

lemma *has-contour-integral-subpath-refl* [*iff*]: (*f has-contour-integral 0*) (*subpath u u g*)
by (*simp add*: *has-contour-integral subpath-def*)

lemma *contour-integrable-subpath-refl* [*iff*]: *f contour-integrable-on* (*subpath u u g*)
using *has-contour-integral-subpath-refl contour-integrable-on-def* **by** *blast*

lemma *contour-integral-subpath-refl* [*simp*]: *contour-integral* (*subpath u u g*) *f* = *0*
by (*simp add*: *has-contour-integral-subpath-refl contour-integral-unique*)

lemma *has-contour-integral-subpath*:
assumes *f*: *f contour-integrable-on g* **and** *g*: *valid-path g*
and *uv*: *u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows (*f has-contour-integral integral* {*u..v*} (λ*x*. *f*(*g x*) ∗ *vector-derivative g* (*at x*)))
(*subpath u v g*)
proof (*cases v*=*u*)
case *True*
then show *?thesis*
using *f* **by** (*simp add*: *contour-integrable-on-def subpath-def has-contour-integral*)
next
case *False*
obtain *s* **where** *s*: ⋀*x*. *x* ∈ {*0..1*} − *s* ⟹ *g differentiable at x* **and** *fs*: *finite s*
using *g* **unfolding** *piecewise-C1-differentiable-on-def C1-differentiable-on-eq valid-path-def* **by** *blast*
have ∗: ((λ*x*. *f* (*g* ((*v* − *u*) ∗ *x* + *u*)) ∗ *vector-derivative g* (*at* ((*v* − *u*) ∗ *x* + *u*)))
has-integral (*1* / (*v* − *u*)) ∗ *integral* {*u..v*} (λ*t*. *f* (*g t*) ∗ *vector-derivative g* (*at t*)))
{*0..1*}
using *f uv*
apply (*simp add*: *contour-integrable-on subpath-def has-contour-integral*)
apply (*drule integrable-on-subcbox* [**where** *a*=*u* **and** *b*=*v*, *simplified*])
apply (*simp-all add*: *has-integral-integral*)
apply (*drule has-integral-affinity* [**where** *m*=*v*−*u* **and** *c*=*u*, *simplified*])
apply (*simp-all add*: *False image-affinity-atLeastAtMost-div-diff scaleR-conv-of-real*)
apply (*simp add*: *divide-simps False*)
done
{ fix *x*
have *x* ∈ {*0..1*} ⟹
x ∉ (λ*t*. (*v*−*u*) ∗$_R$ *t* + *u*) −‘ *s* ⟹
vector-derivative (λ*x*. *g* ((*v*−*u*) ∗ *x* + *u*)) (*at x*) = (*v*−*u*) ∗$_R$ *vector-derivative g* (*at* ((*v*−*u*) ∗ *x* + *u*))
apply (*rule vector-derivative-at* [*OF vector-diff-chain-at* [*simplified o-def*]])
apply (*intro derivative-eq-intros* | *simp*)+

```
    apply (cut-tac s [of (v − u) ∗ x + u])
    using uv mult-left-le [of x v−u]
    apply (auto simp: vector-derivative-works)
    done
  } note vd = this
  show ?thesis
    apply (cut-tac has-integral-cmul [OF ∗, where c = v−u])
    using fs assms
    apply (simp add: False subpath-def has-contour-integral)
    apply (rule-tac s = (λt. ((v−u) ∗R t + u)) −` s in has-integral-spike-finite)
    apply (auto simp: inj-on-def False finite-vimageI vd scaleR-conv-of-real)
    done
qed

lemma contour-integrable-subpath:
  assumes f contour-integrable-on g valid-path g u ∈ {0..1} v ∈ {0..1}
    shows f contour-integrable-on (subpath u v g)
  apply (cases u v rule: linorder-class.le-cases)
   apply (metis contour-integrable-on-def has-contour-integral-subpath [OF assms])
  apply (subst reversepath-subpath [symmetric])
  apply (rule contour-integrable-reversepath)
   using assms apply (blast intro: valid-path-subpath)
  apply (simp add: contour-integrable-on-def)
  using assms apply (blast intro: has-contour-integral-subpath)
  done

lemma has-integral-integrable-integral: (f has-integral i) s ⟷ f integrable-on s ∧
integral s f = i
  by blast

lemma has-integral-contour-integral-subpath:
  assumes f contour-integrable-on g valid-path g u ∈ {0..1} v ∈ {0..1} u ≤ v
    shows (((λx. f(g x) ∗ vector-derivative g (at x)))
           has-integral  contour-integral (subpath u v g) f) {u..v}
  using assms
  apply (auto simp: has-integral-integrable-integral)
  apply (rule integrable-on-subcbox [where a=u and b=v and s = {0..1}, sim-
plified])
 apply (auto simp: contour-integral-unique [OF has-contour-integral-subpath] contour-integrable-on)
  done

lemma contour-integral-subcontour-integral:
  assumes f contour-integrable-on g valid-path g u ∈ {0..1} v ∈ {0..1} u ≤ v
    shows contour-integral (subpath u v g) f =
          integral {u..v} (λx. f(g x) ∗ vector-derivative g (at x))
  using assms has-contour-integral-subpath contour-integral-unique by blast

lemma contour-integral-subpath-combine-less:
  assumes f contour-integrable-on g valid-path g u ∈ {0..1} v ∈ {0..1} w ∈ {0..1}
```

```
        u<v v<w
    shows contour-integral (subpath u v g) f + contour-integral (subpath v w g) f
=
          contour-integral (subpath u w g) f
  using assms apply (auto simp: contour-integral-subcontour-integral)
  apply (rule integral-combine, auto)
  apply (rule integrable-on-subcbox [where a=u and b=w and s = {0..1}, simplified])
  apply (auto simp: contour-integrable-on)
  done

lemma contour-integral-subpath-combine:
  assumes f contour-integrable-on g valid-path g u ∈ {0..1} v ∈ {0..1} w ∈ {0..1}
    shows contour-integral (subpath u v g) f + contour-integral (subpath v w g) f
=
          contour-integral (subpath u w g) f
proof (cases u≠v ∧ v≠w ∧ u≠w)
  case True
    have ∗: subpath v u g = reversepath(subpath u v g) ∧
            subpath w u g = reversepath(subpath u w g) ∧
            subpath w v g = reversepath(subpath v w g)
      by (auto simp: reversepath-subpath)
    have u < v ∧ v < w ∨
         u < w ∧ w < v ∨
         v < u ∧ u < w ∨
         v < w ∧ w < u ∨
         w < u ∧ u < v ∨
         w < v ∧ v < u
      using True assms by linarith
    with assms show ?thesis
      using contour-integral-subpath-combine-less [of f g u v w]
            contour-integral-subpath-combine-less [of f g u w v]
            contour-integral-subpath-combine-less [of f g v u w]
            contour-integral-subpath-combine-less [of f g v w u]
            contour-integral-subpath-combine-less [of f g w u v]
            contour-integral-subpath-combine-less [of f g w v u]
      apply simp
      apply (elim disjE)
      apply (auto simp: ∗ contour-integral-reversepath contour-integrable-subpath
                  valid-path-reversepath valid-path-subpath algebra-simps)
      done
next
  case False
  then show ?thesis
    apply (auto simp: contour-integral-subpath-refl)
    using assms
   by (metis eq-neg-iff-add-eq-0 contour-integrable-subpath contour-integral-reversepath
reversepath-subpath valid-path-subpath)
qed
```

lemma *contour-integral-integral*:
shows *contour-integral g f = integral {0..1} (λx. f (g x) * vector-derivative g (at x))*
by (*simp add*: *contour-integral-def integral-def has-contour-integral*)

52.9 Segments via convex hulls

lemma *segments-subset-convex-hull*:
closed-segment a b ⊆ (convex hull {a,b,c})
closed-segment a c ⊆ (convex hull {a,b,c})
closed-segment b c ⊆ (convex hull {a,b,c})
closed-segment b a ⊆ (convex hull {a,b,c})
closed-segment c a ⊆ (convex hull {a,b,c})
closed-segment c b ⊆ (convex hull {a,b,c})
by (*auto simp*: *segment-convex-hull linepath-of-real elim*!: *rev-subsetD* [*OF - hull-mono*])

lemma *midpoints-in-convex-hull*:
assumes *x ∈ convex hull s y ∈ convex hull s*
shows *midpoint x y ∈ convex hull s*
proof −
have *(1 − inverse(2)) *$_R$ x + inverse(2) *$_R$ y ∈ convex hull s*
apply (*rule convexD-alt*)
using *assms*
apply (*auto simp*: *convex-convex-hull*)
done
then show *?thesis*
by (*simp add*: *midpoint-def algebra-simps*)
qed

lemma *convex-hull-subset*:
s ⊆ convex hull t ⟹ convex hull s ⊆ convex hull t
by (*simp add*: *convex-convex-hull subset-hull*)

lemma *not-in-interior-convex-hull-3*:
fixes *a* :: *complex*
shows *a ∉ interior(convex hull {a,b,c})*
b ∉ interior(convex hull {a,b,c})
c ∉ interior(convex hull {a,b,c})
by (*auto simp*: *card-insert-le-m1 not-in-interior-convex-hull*)

Cauchy's theorem where there's a primitive

lemma *contour-integral-primitive-lemma*:
fixes *f* :: *complex ⇒ complex* **and** *g* :: *real ⇒ complex*
assumes *a ≤ b*
and *⋀x. x ∈ s ⟹ (f has-field-derivative f′ x) (at x within s)*
and *g piecewise-differentiable-on {a..b}* *⋀x. x ∈ {a..b} ⟹ g x ∈ s*
shows *((λx. f′(g x) * vector-derivative g (at x within {a..b}))*
has-integral (f(g b) − f(g a))) {a..b}

proof −
obtain k **where** k: *finite* k $\forall x{\in}\{a..b\} - k$. g *differentiable* (*at* x *within* $\{a..b\}$)
and *cg*: *continuous-on* $\{a..b\}$ g
using *assms* **by** (*auto simp*: *piecewise-differentiable-on-def*)
have *cfg*: *continuous-on* $\{a..b\}$ $(\lambda x.\ f\ (g\ x))$
apply (*rule continuous-on-compose* [*OF cg*, *unfolded o-def*])
using *assms*
apply (*metis complex-differentiable-def complex-differentiable-imp-continuous-at continuous-on-eq-continuous-within continuous-on-subset image-subset-iff*)
done
{ fix x::*real*
assume a: $a < x$ **and** b: $x < b$ **and** xk: $x \notin k$
then have g *differentiable at* x *within* $\{a..b\}$
using k **by** (*simp add*: *differentiable-at-withinI*)
then have (*g has-vector-derivative vector-derivative g* (*at x within* $\{a..b\}$)) (*at x within* $\{a..b\}$)
by (*simp add*: *vector-derivative-works has-field-derivative-def scaleR-conv-of-real*)
then have *gdiff*: (*g has-derivative* $(\lambda u.\ u * $ *vector-derivative g* (*at x within* $\{a..b\}$))) (*at x within* $\{a..b\}$)
by (*simp add*: *has-vector-derivative-def scaleR-conv-of-real*)
have (*f has-field-derivative* (*f'* (*g x*))) (*at* (*g x*) *within g* ' $\{a..b\}$)
using *assms* **by** (*metis a atLeastAtMost-iff b DERIV-subset image-subset-iff less-eq-real-def*)
then have *fdiff*: (*f has-derivative op* $*$ (*f'* (*g x*))) (*at* (*g x*) *within g* ' $\{a..b\}$)
by (*simp add*: *has-field-derivative-def*)
have (($\lambda x.$ *f* (*g x*)) *has-vector-derivative f'* (*g x*) $*$ *vector-derivative g* (*at x within* $\{a..b\}$)) (*at x within* $\{a..b\}$)
using *diff-chain-within* [*OF gdiff fdiff*]
by (*simp add*: *has-vector-derivative-def scaleR-conv-of-real o-def mult-ac*)
} note $* = $ *this*
show *?thesis*
apply (*rule fundamental-theorem-of-calculus-interior-strong*)
using k *assms cfg* $*$
apply (*auto simp*: *at-within-closed-interval*)
done
qed

lemma *contour-integral-primitive*:
assumes $\bigwedge x.\ x \in s \Longrightarrow$ (*f has-field-derivative f'* x) (*at* x *within* s)
and *valid-path g path-image* $g \subseteq s$
shows (*f' has-contour-integral* (*f*(*pathfinish g*) − *f*(*pathstart g*))) *g*
using *assms*
apply (*simp add*: *valid-path-def path-image-def pathfinish-def pathstart-def has-contour-integral-def*)
apply (*auto intro*!: *piecewise-C1-imp-differentiable contour-integral-primitive-lemma* [*of 0 1 s*])
done

corollary *Cauchy-theorem-primitive*:
assumes $\bigwedge x.\ x \in s \Longrightarrow$ (*f has-field-derivative f'* x) (*at* x *within* s)

and *valid-path g path-image g* $\subseteq$ *s pathfinish g* = *pathstart g*
shows (*f' has-contour-integral 0*) *g*
using *assms*
by (*metis diff-self contour-integral-primitive*)

Existence of path integral for continuous function

lemma *contour-integrable-continuous-linepath*:
assumes *continuous-on* (*closed-segment a b*) *f*
shows *f contour-integrable-on* (*linepath a b*)
proof −
have *continuous-on* {*0..1*} ((λ*x*. *f x* ∗ (*b* − *a*)) *o linepath a b*)
apply (*rule continuous-on-compose* [*OF continuous-on-linepath*], *simp add*: *linepath-image-01*)
apply (*rule continuous-intros* | *simp add*: *assms*)+
done
then show *?thesis*
apply (*simp add*: *contour-integrable-on-def has-contour-integral-def integrable-on-def* [*symmetric*])
apply (*rule integrable-continuous* [*of 0 1*::*real*, *simplified*])
apply (*rule continuous-on-eq* [**where** *f* = λ*x*. *f*(*linepath a b x*)∗(*b* − *a*)])
apply (*auto simp*: *vector-derivative-linepath-within*)
done
qed

lemma *has-field-der-id*: ((λ*x*. x^2 / *2*) *has-field-derivative x*) (*at x*)
by (*rule has-derivative-imp-has-field-derivative*)
(*rule derivative-intros* | *simp*)+

lemma *contour-integral-id* [*simp*]: *contour-integral* (*linepath a b*) (λ*y*. *y*) = (*b*^*2* − *a*^*2*)/*2*
apply (*rule contour-integral-unique*)
using *contour-integral-primitive* [*of UNIV* λ*x*. *x*^*2*/*2* λ*x*. *x linepath a b*]
apply (*auto simp*: *field-simps has-field-der-id*)
done

lemma *contour-integrable-on-const* [*iff*]: (λ*x*. *c*) *contour-integrable-on* (*linepath a b*)
by (*simp add*: *continuous-on-const contour-integrable-continuous-linepath*)

lemma *contour-integrable-on-id* [*iff*]: (λ*x*. *x*) *contour-integrable-on* (*linepath a b*)
by (*simp add*: *continuous-on-id contour-integrable-continuous-linepath*)

52.10 Arithmetical combining theorems

lemma *has-contour-integral-neg*:
(*f has-contour-integral i*) *g* ⟹ ((λ*x*. −(*f x*)) *has-contour-integral* (−*i*)) *g*
by (*simp add*: *has-integral-neg has-contour-integral-def*)

lemma *has-contour-integral-add*:

⟦(*f1 has-contour-integral i1*) *g*; (*f2 has-contour-integral i2*) *g*⟧
⟹ ((λ*x*. *f1 x* + *f2 x*) *has-contour-integral* (*i1* + *i2*)) *g*
by (*simp add*: *has-integral-add has-contour-integral-def algebra-simps*)

lemma *has-contour-integral-diff*:
⟦(*f1 has-contour-integral i1*) *g*; (*f2 has-contour-integral i2*) *g*⟧
⟹ ((λ*x*. *f1 x* − *f2 x*) *has-contour-integral* (*i1* − *i2*)) *g*
by (*simp add*: *has-integral-sub has-contour-integral-def algebra-simps*)

lemma *has-contour-integral-lmul*:
(*f has-contour-integral i*) *g* ⟹ ((λ*x*. *c* ∗ (*f x*)) *has-contour-integral* (*c*∗*i*)) *g*
apply (*simp add*: *has-contour-integral-def*)
apply (*drule has-integral-mult-right*)
apply (*simp add*: *algebra-simps*)
done

lemma *has-contour-integral-rmul*:
(*f has-contour-integral i*) *g* ⟹ ((λ*x*. (*f x*) ∗ *c*) *has-contour-integral* (*i*∗*c*)) *g*
apply (*drule has-contour-integral-lmul*)
apply (*simp add*: *mult.commute*)
done

lemma *has-contour-integral-div*:
(*f has-contour-integral i*) *g* ⟹ ((λ*x*. *f x*/*c*) *has-contour-integral* (*i*/*c*)) *g*
by (*simp add*: *field-class.field-divide-inverse*) (*metis has-contour-integral-rmul*)

lemma *has-contour-integral-eq*:
⟦(*f has-contour-integral y*) *p*; ⋀*x*. *x* ∈ *path-image p* ⟹ *f x* = *g x*⟧ ⟹ (*g has-contour-integral y*) *p*
apply (*simp add*: *path-image-def has-contour-integral-def*)
by (*metis* (*no-types*, *lifting*) *image-eqI has-integral-eq*)

lemma *has-contour-integral-bound-linepath*:
assumes (*f has-contour-integral i*) (*linepath a b*)
0 ≤ *B* ⋀*x*. *x* ∈ *closed-segment a b* ⟹ *norm*(*f x*) ≤ *B*
shows *norm i* ≤ *B* ∗ *norm*(*b* − *a*)
proof −
{ fix *x*::*real*
assume *x*: *0* ≤ *x x* ≤ *1*
have *norm* (*f* (*linepath a b x*)) ∗
norm (*vector-derivative* (*linepath a b*) (*at x within* {*0..1*})) ≤ *B* ∗ *norm* (*b* − *a*)
by (*auto intro*: *mult-mono simp*: *assms linepath-in-path of-real-linepath vector-derivative-linepath-within x*)
} note ∗ = *this*
have *norm i* ≤ (*B* ∗ *norm* (*b* − *a*)) ∗ *content* (*cbox 0* (*1*::*real*))
apply (*rule has-integral-bound*
[*of* - λ*x*. *f* (*linepath a b x*) ∗ *vector-derivative* (*linepath a b*) (*at x within* {*0..1*})])

using *assms* ∗ **unfolding** *has-contour-integral-def*
apply (*auto simp*: *norm-mult*)
done
then show *?thesis*
by (*auto simp*: *content-real*)
qed

lemma *has-contour-integral-const-linepath*: ((λ*x*. *c*) *has-contour-integral c*∗(*b* − *a*))(*linepath a b*)
unfolding *has-contour-integral-linepath*
by (*metis content-real diff-0-right has-integral-const-real lambda-one of-real-1 scaleR-conv-of-real zero-le-one*)

lemma *has-contour-integral-0*: ((λ*x*. *0*) *has-contour-integral 0*) *g*
by (*simp add*: *has-contour-integral-def*)

lemma *has-contour-integral-is-0*:
(⋀*z*. *z* ∈ *path-image g* ⟹ *f z* = *0*) ⟹ (*f has-contour-integral 0*) *g*
by (*rule has-contour-integral-eq* [*OF has-contour-integral-0*]) *auto*

lemma *has-contour-integral-setsum*:
⟦*finite s*; ⋀*a*. *a* ∈ *s* ⟹ (*f a has-contour-integral i a*) *p*⟧
⟹ ((λ*x*. *setsum* (λ*a*. *f a x*) *s*) *has-contour-integral setsum i s*) *p*
by (*induction s rule*: *finite-induct*) (*auto simp*: *has-contour-integral-0 has-contour-integral-add*)

52.11 Operations on path integrals

lemma *contour-integral-const-linepath* [*simp*]: *contour-integral* (*linepath a b*) (λ*x*. *c*) = *c*∗(*b* − *a*)
by (*rule contour-integral-unique* [*OF has-contour-integral-const-linepath*])

lemma *contour-integral-neg*:
f contour-integrable-on g ⟹ *contour-integral g* (λ*x*. −(*f x*)) = −(*contour-integral g f*)
by (*simp add*: *contour-integral-unique has-contour-integral-integral has-contour-integral-neg*)

lemma *contour-integral-add*:
f1 contour-integrable-on g ⟹ *f2 contour-integrable-on g* ⟹ *contour-integral g* (λ*x*. *f1 x* + *f2 x*) =
contour-integral g f1 + *contour-integral g f2*
by (*simp add*: *contour-integral-unique has-contour-integral-integral has-contour-integral-add*)

lemma *contour-integral-diff*:
f1 contour-integrable-on g ⟹ *f2 contour-integrable-on g* ⟹ *contour-integral g* (λ*x*. *f1 x* − *f2 x*) =
contour-integral g f1 − *contour-integral g f2*
by (*simp add*: *contour-integral-unique has-contour-integral-integral has-contour-integral-diff*)

lemma *contour-integral-lmul*:
shows *f contour-integrable-on g*
$\Longrightarrow$ *contour-integral g* $(\lambda x.\ c * f\ x) = c$*contour-integral g f*
by (*simp add*: *contour-integral-unique has-contour-integral-integral has-contour-integral-lmul*)

lemma *contour-integral-rmul*:
shows *f contour-integrable-on g*
$\Longrightarrow$ *contour-integral g* $(\lambda x.\ f\ x * c) =$ *contour-integral g f* $*\ c$
by (*simp add*: *contour-integral-unique has-contour-integral-integral has-contour-integral-rmul*)

lemma *contour-integral-div*:
shows *f contour-integrable-on g*
$\Longrightarrow$ *contour-integral g* $(\lambda x.\ f\ x\ /\ c) =$ *contour-integral g f* $/\ c$
by (*simp add*: *contour-integral-unique has-contour-integral-integral has-contour-integral-div*)

lemma *contour-integral-eq*:
$(\bigwedge x.\ x \in$ *path-image p* $\Longrightarrow f\ x = g\ x) \Longrightarrow$ *contour-integral p f = contour-integral p g*
by (*simp add*: *contour-integral-def*) (*metis has-contour-integral-eq*)

lemma *contour-integral-eq-0*:
$(\bigwedge z.\ z \in$ *path-image g* $\Longrightarrow f\ z = 0) \Longrightarrow$ *contour-integral g f = 0*
by (*simp add*: *has-contour-integral-is-0 contour-integral-unique*)

lemma *contour-integral-bound-linepath*:
shows
$[\![f$ *contour-integrable-on* (*linepath a b*);
$0 \le B$; $\bigwedge x.\ x \in$ *closed-segment a b* $\Longrightarrow$ *norm*$(f\ x) \le B]\!]$
$\Longrightarrow$ *norm*(*contour-integral* (*linepath a b*) *f*) $\le B*$*norm*$(b - a)$
apply (*rule has-contour-integral-bound-linepath* [*of f*])
apply (*auto simp*: *has-contour-integral-integral*)
done

lemma *contour-integral-0* [*simp*]: *contour-integral g* $(\lambda x.\ 0) = 0$
by (*simp add*: *contour-integral-unique has-contour-integral-0*)

lemma *contour-integral-setsum*:
$[\![$*finite s*; $\bigwedge a.\ a \in s \Longrightarrow$ (*f a*) *contour-integrable-on p*$]\!]$
$\Longrightarrow$ *contour-integral p* $(\lambda x.$ *setsum* $(\lambda a.\ f\ a\ x)\ s) =$ *setsum* $(\lambda a.$ *contour-integral p* (*f a*)) *s*
by (*auto simp*: *contour-integral-unique has-contour-integral-setsum has-contour-integral-integral*)

lemma *contour-integrable-eq*:
$[\![f$ *contour-integrable-on p*; $\bigwedge x.\ x \in$ *path-image p* $\Longrightarrow f\ x = g\ x]\!] \Longrightarrow g$ *contour-integrable-on p*
unfolding *contour-integrable-on-def*
by (*metis has-contour-integral-eq*)

52.12 Arithmetic theorems for path integrability

lemma *contour-integrable-neg*:
f contour-integrable-on g $\Longrightarrow$ $(\lambda x.\ -(f\ x))$ *contour-integrable-on g*
using *has-contour-integral-neg contour-integrable-on-def* **by** *blast*

lemma *contour-integrable-add*:
$[\![$*f1 contour-integrable-on g*; *f2 contour-integrable-on g*$]\!] \Longrightarrow$ $(\lambda x.\ f1\ x + f2\ x)$ *contour-integrable-on g*
using *has-contour-integral-add contour-integrable-on-def*
by *fastforce*

lemma *contour-integrable-diff*:
$[\![$*f1 contour-integrable-on g*; *f2 contour-integrable-on g*$]\!] \Longrightarrow$ $(\lambda x.\ f1\ x - f2\ x)$ *contour-integrable-on g*
using *has-contour-integral-diff contour-integrable-on-def*
by *fastforce*

lemma *contour-integrable-lmul*:
f contour-integrable-on g $\Longrightarrow$ $(\lambda x.\ c * f\ x)$ *contour-integrable-on g*
using *has-contour-integral-lmul contour-integrable-on-def*
by *fastforce*

lemma *contour-integrable-rmul*:
f contour-integrable-on g $\Longrightarrow$ $(\lambda x.\ f\ x * c)$ *contour-integrable-on g*
using *has-contour-integral-rmul contour-integrable-on-def*
by *fastforce*

lemma *contour-integrable-div*:
f contour-integrable-on g $\Longrightarrow$ $(\lambda x.\ f\ x\ /\ c)$ *contour-integrable-on g*
using *has-contour-integral-div contour-integrable-on-def*
by *fastforce*

lemma *contour-integrable-setsum*:
$[\![$*finite s*; $\bigwedge a.\ a \in s \Longrightarrow$ *(f a) contour-integrable-on p*$]\!]$
$\Longrightarrow$ $(\lambda x.\ setsum\ (\lambda a.\ f\ a\ x)\ s)$ *contour-integrable-on p*
unfolding *contour-integrable-on-def*
by (*metis has-contour-integral-setsum*)

52.13 Reversing a path integral

lemma *has-contour-integral-reverse-linepath*:
(*f has-contour-integral i*) (*linepath a b*)
$\Longrightarrow$ (*f has-contour-integral* $(-i)$) (*linepath b a*)
using *has-contour-integral-reversepath valid-path-linepath* **by** *fastforce*

lemma *contour-integral-reverse-linepath*:
continuous-on (*closed-segment a b*) *f*
$\Longrightarrow$ *contour-integral* (*linepath a b*) *f* $= -$ (*contour-integral*(*linepath b a*) *f*)
apply (*rule contour-integral-unique*)

apply (*rule has-contour-integral-reverse-linepath*)
by (*simp add*: *closed-segment-commute contour-integrable-continuous-linepath has-contour-integral-integral*)

lemma *has-contour-integral-split*:
assumes *f*: (*f has-contour-integral i*) (*linepath a c*) (*f has-contour-integral j*) (*linepath c b*)
and *k*: *0* ≤ *k k* ≤ *1*
and *c*: *c* − *a* = *k* ∗*R* (*b* − *a*)
shows (*f has-contour-integral* (*i* + *j*)) (*linepath a b*)
proof (*cases k* = *0* ∨ *k* = *1*)
case *True*
then show *?thesis*
using *assms*
apply *auto*
apply (*metis add.left-neutral has-contour-integral-trivial has-contour-integral-unique*)
apply (*metis add.right-neutral has-contour-integral-trivial has-contour-integral-unique*)
done
next
case *False*
then have *k*: *0* < *k k* < *1 complex-of-real k* ≠ *1*
using *assms* **apply** *auto*
using *of-real-eq-iff* **by** *fastforce*
have *c′*: *c* = *k* ∗*R* (*b* − *a*) + *a*
by (*metis diff-add-cancel c*)
have *bc*: (*b* − *c*) = (*1* − *k*) ∗*R* (*b* − *a*)
by (*simp add*: *algebra-simps c′*)
{ assume ∗: ((λ*x*. *f* ((*1* − *x*) ∗*R* *a* + *x* ∗*R* *c*) ∗ (*c* − *a*)) *has-integral i*) {*0..1*}
have ∗∗: ⋀*x*. ((*k* − *x*) / *k*) ∗*R* *a* + (*x* / *k*) ∗*R* *c* = (*1* − *x*) ∗*R* *a* + *x* ∗*R* *b*
using *False*
apply (*simp add*: *c′ algebra-simps*)
apply (*simp add*: *real-vector.scale-left-distrib* [*symmetric*] *divide-simps*)
done
have ((λ*x*. *f* ((*1* − *x*) ∗*R* *a* + *x* ∗*R* *b*) ∗ (*b* − *a*)) *has-integral i*) {*0..k*}
using ∗ *k*
apply −
apply (*drule has-integral-affinity* [*of* - - *0 1*::*real inverse k 0*, *simplified*])
apply (*simp-all add*: *divide-simps mult.commute* [*of* - *k*] *image-affinity-atLeastAtMost* ∗∗ *c*)
apply (*drule Integration.has-integral-cmul* [**where** *c* = *inverse k*])
apply (*simp add*: *Integration.has-integral-cmul*)
done
} note *fi* = *this*
{ assume ∗: ((λ*x*. *f* ((*1* − *x*) ∗*R* *c* + *x* ∗*R* *b*) ∗ (*b* − *c*)) *has-integral j*) {*0..1*}
have ∗∗: ⋀*x*. (((*1* − *x*) / (*1* − *k*)) ∗*R* *c* + ((*x* − *k*) / (*1* − *k*)) ∗*R* *b*) = ((*1* − *x*) ∗*R* *a* + *x* ∗*R* *b*)
using *k*

```
      apply (simp add: c' field-simps)
      apply (simp add: scaleR-conv-of-real divide-simps)
      apply (simp add: field-simps)
      done
    have ((λx. f ((1 − x) *R a + x *R b) ∗ (b − a)) has-integral j) {k..1}
      using ∗ k
      apply −
      apply (drule has-integral-affinity [of - - 0 1::real inverse(1 − k) −(k/(1 −
k)), simplified])
     apply (simp-all add: divide-simps mult.commute [of - 1−k] image-affinity-atLeastAtMost
** bc)
      apply (drule Integration.has-integral-cmul [where k = (1 − k) *R j and c
= inverse (1 − k)])
      apply (simp add: Integration.has-integral-cmul)
      done
  } note fj = this
  show ?thesis
    using f k
    apply (simp add: has-contour-integral-linepath)
    apply (simp add: linepath-def)
    apply (rule has-integral-combine [OF - - fi fj], simp-all)
    done
qed

lemma continuous-on-closed-segment-transform:
  assumes f: continuous-on (closed-segment a b) f
      and k: 0 ≤ k k ≤ 1
      and c: c − a = k *R (b − a)
    shows continuous-on (closed-segment a c) f
proof −
  have c': c = (1 − k) *R a + k *R b
    using c by (simp add: algebra-simps)
  show continuous-on (closed-segment a c) f
    apply (rule continuous-on-subset [OF f])
    apply (simp add: segment-convex-hull)
    apply (rule convex-hull-subset)
    using assms
    apply (auto simp: hull-inc c' Convex.convexD-alt)
    done
qed

lemma contour-integral-split:
  assumes f: continuous-on (closed-segment a b) f
      and k: 0 ≤ k k ≤ 1
      and c: c − a = k *R (b − a)
     shows contour-integral(linepath a b) f = contour-integral(linepath a c) f +
contour-integral(linepath c b) f
proof −
  have c': c = (1 − k) *R a + k *R b
```

using *c* **by** (*simp add*: *algebra-simps*)
have ∗: *continuous-on* (*closed-segment a c*) *f continuous-on* (*closed-segment c b*) *f*
apply (*rule-tac* [!] *continuous-on-subset* [*OF f*])
apply (*simp-all add*: *segment-convex-hull*)
apply (*rule-tac* [!] *convex-hull-subset*)
using *assms*
apply (*auto simp*: *hull-inc c′ Convex.convexD-alt*)
done
show *?thesis*
apply (*rule contour-integral-unique*)
apply (*rule has-contour-integral-split* [*OF has-contour-integral-integral has-contour-integral-integral k c*])
apply (*rule contour-integrable-continuous-linepath* ∗)+
done
qed

lemma *contour-integral-split-linepath*:
assumes *f*: *continuous-on* (*closed-segment a b*) *f*
and *c*: *c* ∈ *closed-segment a b*
shows *contour-integral*(*linepath a b*) *f* = *contour-integral*(*linepath a c*) *f* + *contour-integral*(*linepath c b*) *f*
using *c*
by (*auto simp*: *closed-segment-def algebra-simps intro*!: *contour-integral-split* [*OF f*])

lemma *has-contour-integral-midpoint*:
assumes (*f has-contour-integral i*) (*linepath a* (*midpoint a b*))
(*f has-contour-integral j*) (*linepath* (*midpoint a b*) *b*)
shows (*f has-contour-integral* (*i* + *j*)) (*linepath a b*)
apply (*rule has-contour-integral-split* [**where** *c* = *midpoint a b* **and** *k* = *1/2*])
using *assms*
apply (*auto simp*: *midpoint-def algebra-simps scaleR-conv-of-real*)
done

lemma *contour-integral-midpoint*:
continuous-on (*closed-segment a b*) *f*
⟹ *contour-integral* (*linepath a b*) *f* =
contour-integral (*linepath a* (*midpoint a b*)) *f* + *contour-integral* (*linepath* (*midpoint a b*) *b*) *f*
apply (*rule contour-integral-split* [**where** *c* = *midpoint a b* **and** *k* = *1/2*])
using *assms*
apply (*auto simp*: *midpoint-def algebra-simps scaleR-conv-of-real*)
done

A couple of special case lemmas that are useful below

lemma *triangle-linear-has-chain-integral*:

((λx. m∗x + d) *has-contour-integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
apply (*rule Cauchy-theorem-primitive* [*of UNIV* λx. *m/2 ∗ x^2 + d∗x*])
apply (*auto intro*!: *derivative-eq-intros*)
done

lemma *has-chain-integral-chain-integral3*:
(*f has-contour-integral i*) (*linepath a b +++ linepath b c +++ linepath c d*)
⟹ *contour-integral* (*linepath a b*) *f* + *contour-integral* (*linepath b c*) *f* + *contour-integral* (*linepath c d*) *f* = *i*
apply (*subst contour-integral-unique* [*symmetric*], *assumption*)
apply (*drule has-contour-integral-integrable*)
apply (*simp add*: *valid-path-join*)
done

52.14 Reversing the order in a double path integral

The condition is stronger than needed but it's often true in typical situations

lemma *fst-im-cbox* [*simp*]: *cbox c d* ≠ {} ⟹ (*fst* ' *cbox* (*a*,*c*) (*b*,*d*)) = *cbox a b*
by (*auto simp*: *cbox-Pair-eq*)

lemma *snd-im-cbox* [*simp*]: *cbox a b* ≠ {} ⟹ (*snd* ' *cbox* (*a*,*c*) (*b*,*d*)) = *cbox c d*
by (*auto simp*: *cbox-Pair-eq*)

lemma *contour-integral-swap*:
assumes *fcon*: *continuous-on* (*path-image g* × *path-image h*) (λ(*y1*,*y2*). *f y1 y2*)
and *vp*: *valid-path g valid-path h*
and *gvcon*: *continuous-on* {*0..1*} (λt. *vector-derivative g* (*at t*))
and *hvcon*: *continuous-on* {*0..1*} (λt. *vector-derivative h* (*at t*))
shows *contour-integral g* (λw. *contour-integral h* (*f w*)) =
contour-integral h (λz. *contour-integral g* (λw. *f w z*))
proof −
have *gcon*: *continuous-on* {*0..1*} *g* **and** *hcon*: *continuous-on* {*0..1*} *h*
using *assms* **by** (*auto simp*: *valid-path-def piecewise-C1-differentiable-on-def*)
have *fgh1*: ⋀x. (λt. *f* (*g x*) (*h t*)) = (λ(*y1*,*y2*). *f y1 y2*) *o* (λt. (*g x*, *h t*))
by (*rule ext*) *simp*
have *fgh2*: ⋀x. (λt. *f* (*g t*) (*h x*)) = (λ(*y1*,*y2*). *f y1 y2*) *o* (λt. (*g t*, *h x*))
by (*rule ext*) *simp*
have *fcon-im1*: ⋀x. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ *continuous-on* ((λt. (*g x*, *h t*)) ' {*0..1*}) (λ(*x*, *y*). *f x y*)
by (*rule continuous-on-subset* [*OF fcon*]) (*auto simp*: *path-image-def*)
have *fcon-im2*: ⋀x. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ *continuous-on* ((λt. (*g t*, *h x*)) ' {*0..1*}) (λ(*x*, *y*). *f x y*)
by (*rule continuous-on-subset* [*OF fcon*]) (*auto simp*: *path-image-def*)
have *vdg*: ⋀y. *y* ∈ {*0..1*} ⟹ (λx. *f* (*g x*) (*h y*) ∗ *vector-derivative g* (*at x*)) *integrable-on* {*0..1*}
apply (*rule integrable-continuous-real*)
apply (*rule continuous-on-mult* [*OF - gvcon*])

apply (*subst fgh2*)
apply (*rule fcon-im2 gcon continuous-intros* | *simp*)+
done
have (λ*z*. *vector-derivative g* (*at* (*fst z*))) = (λ*x*. *vector-derivative g* (*at x*)) *o fst*
by *auto*
then have *gvcon'*: *continuous-on* (*cbox* (*0*, *0*) (*1*, *1*::*real*)) (λ*x*. *vector-derivative g* (*at* (*fst x*)))
apply (*rule ssubst*)
apply (*rule continuous-intros* | *simp add*: *gvcon*)+
done
have (λ*z*. *vector-derivative h* (*at* (*snd z*))) = (λ*x*. *vector-derivative h* (*at x*)) *o snd*
by *auto*
then have *hvcon'*: *continuous-on* (*cbox* (*0*, *0*) (*1*::*real*, *1*)) (λ*x*. *vector-derivative h* (*at* (*snd x*)))
apply (*rule ssubst*)
apply (*rule continuous-intros* | *simp add*: *hvcon*)+
done
have (λ*x*. *f* (*g* (*fst x*)) (*h* (*snd x*))) = (λ(*y1*,*y2*). *f y1 y2*) *o* (λ*w*. ((*g o fst*) *w*, (*h o snd*) *w*))
by *auto*
then have *fgh*: *continuous-on* (*cbox* (*0*, *0*) (*1*, *1*)) (λ*x*. *f* (*g* (*fst x*)) (*h* (*snd x*)))
apply (*rule ssubst*)
apply (*rule gcon hcon continuous-intros* | *simp*)+
apply (*auto simp*: *path-image-def intro*: *continuous-on-subset* [*OF fcon*])
done
have *integral* {*0..1*} (λ*x*. *contour-integral h* (*f* (*g x*)) ∗ *vector-derivative g* (*at x*)) =
integral {*0..1*} (λ*x*. *contour-integral h* (λ*y*. *f* (*g x*) *y* ∗ *vector-derivative g* (*at x*)))
apply (*rule integral-cong* [*OF contour-integral-rmul* [*symmetric*]])
apply (*clarsimp simp*: *contour-integrable-on*)
apply (*rule integrable-continuous-real*)
apply (*rule continuous-on-mult* [*OF* - *hvcon*])
apply (*subst fgh1*)
apply (*rule fcon-im1 hcon continuous-intros* | *simp*)+
done
also have ... = *integral* {*0..1*}
(λ*y*. *contour-integral g* (λ*x*. *f x* (*h y*) ∗ *vector-derivative h* (*at y*)))
apply (*simp add*: *contour-integral-integral*)
apply (*subst integral-swap-continuous* [**where** *'a* = *real* **and** *'b* = *real*, *of 0 0 1 1*, *simplified*])
apply (*rule fgh gvcon' hvcon' continuous-intros* | *simp add*: *split-def*)+
apply (*simp add*: *algebra-simps*)
done
also have ... = *contour-integral h* (λ*z*. *contour-integral g* (λ*w*. *f w z*))
apply (*simp add*: *contour-integral-integral*)
apply (*rule integral-cong*)
apply (*subst integral-mult-left* [*symmetric*])

apply (*blast intro*: *vdg*)
apply (*simp add*: *algebra-simps*)
done
finally show *?thesis*
by (*simp add*: *contour-integral-integral*)
qed

52.15 The key quadrisection step

lemma *norm-sum-half*:
assumes *norm*(*a* + *b*) >= *e*
shows *norm a* >= *e*/2 ∨ *norm b* >= *e*/2
proof −
have *e* ≤ *norm* (− *a* − *b*)
by (*simp add*: *add.commute assms norm-minus-commute*)
thus *?thesis*
using *norm-triangle-ineq4 order-trans* **by** *fastforce*
qed

lemma *norm-sum-lemma*:
assumes *e* ≤ *norm* (*a* + *b* + *c* + *d*)
shows *e* / 4 ≤ *norm a* ∨ *e* / 4 ≤ *norm b* ∨ *e* / 4 ≤ *norm c* ∨ *e* / 4 ≤ *norm d*
proof −
have *e* ≤ *norm* ((*a* + *b*) + (*c* + *d*)) **using** *assms*
by (*simp add*: *algebra-simps*)
then show *?thesis*
by (*auto dest*!: *norm-sum-half*)
qed

lemma *Cauchy-theorem-quadrisection*:
assumes *f*: *continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
and *dist*: *dist a b* ≤ *K dist b c* ≤ *K dist c a* ≤ *K*
and *e*: *e* ∗ *K*^2 ≤
norm (*contour-integral*(*linepath a b*) *f* + *contour-integral*(*linepath b c*) *f* + *contour-integral*(*linepath c a*) *f*)
shows ∃ *a*′ *b*′ *c*′.
a′ ∈ *convex hull* {*a*,*b*,*c*} ∧ *b*′ ∈ *convex hull* {*a*,*b*,*c*} ∧ *c*′ ∈ *convex hull* {*a*,*b*,*c*} ∧
dist a′ *b*′ ≤ *K*/2 ∧ *dist b*′ *c*′ ≤ *K*/2 ∧ *dist c*′ *a*′ ≤ *K*/2 ∧
e ∗ (*K*/2)^2 ≤ *norm*(*contour-integral*(*linepath a*′ *b*′) *f* + *contour-integral*(*linepath b*′ *c*′) *f* + *contour-integral*(*linepath c*′ *a*′) *f*)
proof −
note *divide-le-eq-numeral1* [*simp del*]
def *a*′ ≡ *midpoint b c*
def *b*′ ≡ *midpoint c a*
def *c*′ ≡ *midpoint a b*
have *fabc*: *continuous-on* (*closed-segment a b*) *f continuous-on* (*closed-segment b c*) *f continuous-on* (*closed-segment c a*) *f*

```
    using f continuous-on-subset segments-subset-convex-hull by metis+
  have fcont': continuous-on (closed-segment c' b') f
             continuous-on (closed-segment a' c') f
             continuous-on (closed-segment b' a') f
    unfolding a'-def b'-def c'-def
    apply (rule continuous-on-subset [OF f],
          metis midpoints-in-convex-hull convex-hull-subset hull-subset insert-subset
segment-convex-hull)+
    done
  let ?pathint = λx y. contour-integral(linepath x y) f
  have ∗: ?pathint a b + ?pathint b c + ?pathint c a =
          (?pathint a c' + ?pathint c' b' + ?pathint b' a) +
          (?pathint a' c' + ?pathint c' b + ?pathint b a') +
          (?pathint a' c + ?pathint c b' + ?pathint b' a') +
          (?pathint a' b' + ?pathint b' c' + ?pathint c' a')
    apply (simp add: fcont' contour-integral-reverse-linepath)
    apply (simp add: a'-def b'-def c'-def contour-integral-midpoint fabc)
    done
  have [simp]: ⋀x y. cmod (x ∗ 2 − y ∗ 2) = cmod (x − y) ∗ 2
    by (metis left-diff-distrib mult.commute norm-mult-numeral1)
  have [simp]: ⋀x y. cmod (x − y) = cmod (y − x)
    by (simp add: norm-minus-commute)
  consider e ∗ K^2 / 4 ≤ cmod (?pathint a c' + ?pathint c' b' + ?pathint b' a) |
           e ∗ K^2 / 4 ≤ cmod (?pathint a' c' + ?pathint c' b + ?pathint b a') |
           e ∗ K^2 / 4 ≤ cmod (?pathint a' c + ?pathint c b' + ?pathint b' a') |
           e ∗ K^2 / 4 ≤ cmod (?pathint a' b' + ?pathint b' c' + ?pathint c' a')
    using assms
    apply (simp only: ∗)
    apply (blast intro: that dest!: norm-sum-lemma)
    done
  then show ?thesis
  proof cases
    case 1 then show ?thesis
      apply (rule-tac x=a in exI)
      apply (rule exI [where x=c'])
      apply (rule exI [where x=b'])
      using assms
        apply (auto simp: a'-def c'-def b'-def midpoints-in-convex-hull hull-subset
[THEN subsetD])
      apply (auto simp: midpoint-def dist-norm scaleR-conv-of-real divide-simps)
      done
  next
    case 2 then show ?thesis
      apply (rule-tac x=a' in exI)
      apply (rule exI [where x=c'])
      apply (rule exI [where x=b])
      using assms
        apply (auto simp: a'-def c'-def b'-def midpoints-in-convex-hull hull-subset
[THEN subsetD])
```

apply (*auto simp*: *midpoint-def dist-norm scaleR-conv-of-real divide-simps*)
done
next
case *3* **then show** *?thesis*
apply (*rule-tac x=a'* **in** *exI*)
apply (*rule exI* [**where** *x=c*])
apply (*rule exI* [**where** *x=b'*])
using *assms*
apply (*auto simp*: *a'-def c'-def b'-def midpoints-in-convex-hull hull-subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint-def dist-norm scaleR-conv-of-real divide-simps*)
done
next
case *4* **then show** *?thesis*
apply (*rule-tac x=a'* **in** *exI*)
apply (*rule exI* [**where** *x=b'*])
apply (*rule exI* [**where** *x=c'*])
using *assms*
apply (*auto simp*: *a'-def c'-def b'-def midpoints-in-convex-hull hull-subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint-def dist-norm scaleR-conv-of-real divide-simps*)
done
qed
qed

52.16 Cauchy's theorem for triangles

lemma *triangle-points-closer*:
fixes *a*::*complex*
shows ⟦*x* ∈ *convex hull* {*a*,*b*,*c*}; *y* ∈ *convex hull* {*a*,*b*,*c*}⟧
⟹ *norm*(*x* − *y*) ≤ *norm*(*a* − *b*) ∨
norm(*x* − *y*) ≤ *norm*(*b* − *c*) ∨
norm(*x* − *y*) ≤ *norm*(*c* − *a*)
using *simplex-extremal-le* [*of* {*a*,*b*,*c*}]
by (*auto simp*: *norm-minus-commute*)

lemma *holomorphic-point-small-triangle*:
assumes *x*: *x* ∈ *s*
and *f*: *continuous-on s f*
and *cd*: *f complex-differentiable* (*at x within s*)
and *e*: *0* < *e*
shows ∃*k*>*0*. ∀ *a b c*. *dist a b* ≤ *k* ∧ *dist b c* ≤ *k* ∧ *dist c a* ≤ *k* ∧
x ∈ *convex hull* {*a*,*b*,*c*} ∧ *convex hull* {*a*,*b*,*c*} ⊆ *s*
⟶ *norm*(*contour-integral*(*linepath a b*) *f* + *contour-integral*(*linepath b c*) *f* +
contour-integral(*linepath c a*) *f*)
≤ *e*∗(*dist a b* + *dist b c* + *dist c a*)^*2*
(**is** ∃*k*>*0*. ∀ *a b c*. - ⟶ *?normle a b c*)
proof −

have *le-of-3*: $\bigwedge a\ x\ y\ z.\ [\![0 \le x*y;\ 0 \le x*z;\ 0 \le y*z;\ a \le (e*(x + y + z))*x + (e*(x + y + z))*y + (e*(x + y + z))*z]\!] \Longrightarrow a \le e*(x + y + z)\hat{}2$
by (*simp add*: *algebra-simps power2-eq-square*)
have *disj-le*: $[\![x \le a \vee x \le b \vee x \le c;\ 0 \le a;\ 0 \le b;\ 0 \le c]\!] \Longrightarrow x \le a + b + c$
for *x*::*real* **and** *a b c*
by *linarith*
have *fabc*: *f contour-integrable-on linepath a b f contour-integrable-on linepath b c f contour-integrable-on linepath c a*
if *convex hull* $\{a, b, c\} \subseteq s$ **for** *a b c*
using *segments-subset-convex-hull that*
by (*metis continuous-on-subset f contour-integrable-continuous-linepath*)+
note *path-bound* = *has-contour-integral-bound-linepath* [*simplified norm-minus-commute*, *OF has-contour-integral-integral*]
{ fix *f' a b c d*
assume *d*: $0 < d$
and *f'*: $\bigwedge y.\ [\![cmod\ (y - x) \le d;\ y \in s]\!] \Longrightarrow cmod\ (f\ y - f\ x - f'\ *\ (y - x)) \le e\ *\ cmod\ (y - x)$
and *le*: *cmod* $(a - b) \le d$ *cmod* $(b - c) \le d$ *cmod* $(c - a) \le d$
and *xc*: $x \in$ *convex hull* $\{a, b, c\}$
and *s*: *convex hull* $\{a, b, c\} \subseteq s$
have *pa*: *contour-integral* (*linepath a b*) *f* + *contour-integral* (*linepath b c*) *f* + *contour-integral* (*linepath c a*) *f* =
contour-integral (*linepath a b*) ($\lambda y.\ f\ y - f\ x - f'*(y - x)$) +
contour-integral (*linepath b c*) ($\lambda y.\ f\ y - f\ x - f'*(y - x)$) +
contour-integral (*linepath c a*) ($\lambda y.\ f\ y - f\ x - f'*(y - x)$)
apply (*simp add*: *contour-integral-diff contour-integral-lmul contour-integrable-lmul contour-integrable-diff fabc* [*OF s*])
apply (*simp add*: *field-simps*)
done
{ fix *y*
assume *yc*: $y \in$ *convex hull* $\{a,b,c\}$
have *cmod* $(f\ y - f\ x - f'\ *\ (y - x)) \le e*norm(y - x)$
apply (*rule f'*)
apply (*metis triangle-points-closer* [*OF xc yc*] *le norm-minus-commute order-trans*)
using *s yc* **by** *blast*
also have ... $\le e\ *\ (cmod\ (a - b) + cmod\ (b - c) + cmod\ (c - a))$
by (*simp add*: *yc e xc disj-le* [*OF triangle-points-closer*])
finally have *cmod* $(f\ y - f\ x - f'\ *\ (y - x)) \le e\ *\ (cmod\ (a - b) + cmod\ (b - c) + cmod\ (c - a))$ **.**
} note *cm-le* = *this*
have *?normle a b c*
apply (*simp add*: *dist-norm pa*)
apply (*rule le-of-3*)
using *f' xc s e*
apply *simp-all*
apply (*intro norm-triangle-le add-mono path-bound*)
apply (*simp-all add*: *contour-integral-diff contour-integral-lmul contour-integrable-lmul*

contour-integrable-diff fabc)
apply (*blast intro*: *cm-le elim*: *dest*: *segments-subset-convex-hull* [*THEN subsetD*])+
done
} note $* = this$
show *?thesis*
using *cd e*
apply (*simp add*: *complex-differentiable-def has-field-derivative-def has-derivative-within-alt approachable-lt-le2 Ball-def*)
apply (*clarify dest*!: *spec mp*)
using $*$
apply (*simp add*: *dist-norm*, *blast*)
done
qed

locale *Chain* =
fixes *x0 At Follows*
assumes *At0*: *At x0 0*
and *AtSuc*: $\bigwedge x\ n.\ At\ x\ n \Longrightarrow \exists x'.\ At\ x'\ (Suc\ n) \wedge Follows\ x'\ x$
begin
primrec *f* **where**
f 0 = *x0*
| *f* (*Suc n*) = (*SOME x*. *At x* (*Suc n*) ∧ *Follows x* (*f n*))

lemma *At*: *At* (*f n*) *n*
proof (*induct n*)
case *0* **show** *?case*
by (*simp add*: *At0*)
next
case (*Suc n*) **show** *?case*
by (*metis* (*no-types*, *lifting*) *AtSuc* [*OF Suc*] *f.simps*(*2*) *someI-ex*)
qed

lemma *Follows*: *Follows* (*f*(*Suc n*)) (*f n*)
by (*metis* (*no-types*, *lifting*) *AtSuc* [*OF At* [*of n*]] *f.simps*(*2*) *someI-ex*)

declare *f.simps*(*2*) [*simp del*]
end

lemma *Chain3*:
assumes *At0*: *At x0 y0 z0 0*
and *AtSuc*: $\bigwedge x\ y\ z\ n.\ At\ x\ y\ z\ n \Longrightarrow \exists x'\ y'\ z'.\ At\ x'\ y'\ z'\ (Suc\ n) \wedge Follows\ x'\ y'\ z'\ x\ y\ z$
obtains *f g h* **where**
f 0 = *x0 g 0* = *y0 h 0* = *z0*
$\bigwedge n.\ At\ (f\ n)\ (g\ n)\ (h\ n)\ n$
$\bigwedge n.\ Follows\ (f(Suc\ n))\ (g(Suc\ n))\ (h(Suc\ n))\ (f\ n)\ (g\ n)\ (h\ n)$

proof −
interpret *three*: *Chain* (*x0*,*y0*,*z0*) λ(*x*,*y*,*z*). *At x y z* λ(*x'*,*y'*,*z'*). λ(*x*,*y*,*z*). *Follows x' y' z' x y z*
apply *unfold-locales*
using *At0 AtSuc* **by** *auto*
show *?thesis*
apply (*rule that* [*of* λ*n*. *fst* (*three.f n*) λ*n*. *fst* (*snd* (*three.f n*)) λ*n*. *snd* (*snd* (*three.f n*))])
apply *simp-all*
using *three.At three.Follows*
apply (*simp-all add*: *split-beta'*)
done
qed

lemma *Cauchy-theorem-triangle*:
assumes *f holomorphic-on* (*convex hull* {*a*,*b*,*c*})
shows (*f has-contour-integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
proof −
have *contf*: *continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
by (*metis assms holomorphic-on-imp-continuous-on*)
let *?pathint* = λ*x y*. *contour-integral*(*linepath x y*) *f*
{ fix *y*::*complex*
assume *fy*: (*f has-contour-integral y*) (*linepath a b +++ linepath b c +++ linepath c a*)
and *ynz*: *y* ≠ *0*
def *K* ≡ *1* + *max* (*dist a b*) (*max* (*dist b c*) (*dist c a*))
def *e* ≡ *norm y* / *K^2*
have *K1*: *K* ≥ *1* **by** (*simp add*: *K-def max.coboundedI1*)
then have *K*: *K* > *0* **by** *linarith*
have [*iff*]: *dist a b* ≤ *K dist b c* ≤ *K dist c a* ≤ *K*
by (*simp-all add*: *K-def*)
have *e*: *e* > *0*
unfolding *e-def* **using** *ynz K1* **by** *simp*
def *At* ≡ λ*x y z n*. *convex hull* {*x*,*y*,*z*} ⊆ *convex hull* {*a*,*b*,*c*} ∧
dist x y ≤ *K*/*2^n* ∧ *dist y z* ≤ *K*/*2^n* ∧ *dist z x* ≤ *K*/*2^n* ∧
norm(*?pathint x y* + *?pathint y z* + *?pathint z x*) ≥ *e*∗(*K*/*2^n*) ^*2*
have *At0*: *At a b c 0*
using *fy*
by (*simp add*: *At-def e-def has-chain-integral-chain-integral3*)
{ fix *x y z n*
assume *At*: *At x y z n*
then have *contf'*: *continuous-on* (*convex hull* {*x*,*y*,*z*}) *f*
using *contf At-def continuous-on-subset* **by** *blast*
have ∃*x' y' z'*. *At x' y' z'* (*Suc n*) ∧ *convex hull* {*x'*,*y'*,*z'*} ⊆ *convex hull* {*x*,*y*,*z*}
using *At*
apply (*simp add*: *At-def*)
using *Cauchy-theorem-quadrisection* [*OF contf', of K*/*2^n e*]

apply *clarsimp*
apply (*rule-tac x=a'* **in** *exI*)
apply (*rule-tac x=b'* **in** *exI*)
apply (*rule-tac x=c'* **in** *exI*)
apply (*simp add*: *algebra-simps*)
apply (*meson convex-hull-subset empty-subsetI insert-subset subsetCE*)
done
} note *AtSuc = this*
obtain *fa fb fc*
where *f0* [*simp*]: *fa 0 = a fb 0 = b fc 0 = c*
and *cosb*: ⋀*n*. *convex hull* {*fa n*, *fb n*, *fc n*} ⊆ *convex hull* {*a*,*b*,*c*}
and *dist*: ⋀*n*. *dist* (*fa n*) (*fb n*) ≤ *K*/2^*n*
⋀*n*. *dist* (*fb n*) (*fc n*) ≤ *K*/2^*n*
⋀*n*. *dist* (*fc n*) (*fa n*) ≤ *K*/2^*n*
and *no*: ⋀*n*. *norm*(*?pathint* (*fa n*) (*fb n*) +
?pathint (*fb n*) (*fc n*) +
?pathint (*fc n*) (*fa n*)) ≥ *e* ∗ (*K*/2^*n*)^2
and *conv-le*: ⋀*n*. *convex hull* {*fa*(*Suc n*), *fb*(*Suc n*), *fc*(*Suc n*)} ⊆ *convex hull* {*fa n*, *fb n*, *fc n*}
apply (*rule Chain3* [*of At*, *OF At0 AtSuc*])
apply (*auto simp*: *At-def*)
done
have ∃*x*. ∀*n*. *x* ∈ *convex hull* {*fa n*, *fb n*, *fc n*}
apply (*rule bounded-closed-nest*)
apply (*simp-all add*: *compact-imp-closed finite-imp-compact-convex-hull finite-imp-bounded-convex-hull*)
apply (*rule allI*)
apply (*rule transitive-stepwise-le*)
apply (*auto simp*: *conv-le*)
done
then obtain *x* **where** *x*: ⋀*n*. *x* ∈ *convex hull* {*fa n*, *fb n*, *fc n*} **by** *auto*
then have *xin*: *x* ∈ *convex hull* {*a*,*b*,*c*}
using *assms f0* **by** *blast*
then have *fx*: *f complex-differentiable at x within* (*convex hull* {*a*,*b*,*c*})
using *assms holomorphic-on-def* **by** *blast*
{ fix *k n*
assume *k*: *0 < k*
and *le*:
⋀*x' y' z'*.
⟦*dist x' y'* ≤ *k*; *dist y' z'* ≤ *k*; *dist z' x'* ≤ *k*;
x ∈ *convex hull* {*x'*,*y'*,*z'*};
convex hull {*x'*,*y'*,*z'*} ⊆ *convex hull* {*a*,*b*,*c*}⟧
⟹
cmod (*?pathint x' y'* + *?pathint y' z'* + *?pathint z' x'*) ∗ *10*
≤ *e* ∗ (*dist x' y'* + *dist y' z'* + *dist z' x'*)^2
and *Kk*: *K* / *k* < 2 ^ *n*
have *K* / 2 ^ *n* < *k* **using** *Kk k*
by (*auto simp*: *field-simps*)
then have *DD*: *dist* (*fa n*) (*fb n*) ≤ *k dist* (*fb n*) (*fc n*) ≤ *k dist* (*fc n*) (*fa n*) ≤ *k*

using *dist* [*of n*] *k*
by *linarith*+
have *dle*: (*dist* (*fa n*) (*fb n*) + *dist* (*fb n*) (*fc n*) + *dist* (*fc n*) (*fa n*))^2
$\leq$ (*3* $*$ *K* / *2* ^ *n*)^2
using *dist* [*of n*] *e K*
by (*simp add*: *abs-le-square-iff* [*symmetric*])
have *less10*: $\bigwedge$*x y*::*real*. *0* $<$ *x* $\Longrightarrow$ *y* $\leq$ *9*$*$*x* $\Longrightarrow$ *y* $<$ *x*$*$*10*
by *linarith*
have *e* $*$ (*dist* (*fa n*) (*fb n*) + *dist* (*fb n*) (*fc n*) + *dist* (*fc n*) (*fa n*))^2 $\leq$ *e* $*$ (*3* $*$ *K* / *2* ^ *n*)^2
using *ynz dle e mult-le-cancel-left-pos* **by** *blast*
also have ... $<$
cmod (*?pathint* (*fa n*) (*fb n*) + *?pathint* (*fb n*) (*fc n*) + *?pathint* (*fc n*) (*fa n*)) $*$ *10*
using *no* [*of n*] *e K*
apply (*simp add*: *e-def field-simps*)
apply (*simp only*: *zero-less-norm-iff* [*symmetric*])
done
finally have *False*
using *le* [*OF DD x cosb*] **by** *auto*
} then
have *?thesis*
using *holomorphic-point-small-triangle* [*OF xin contf fx*, *of e*/*10*] *e*
apply *clarsimp*
apply (*rule-tac x1*=*K*/*k* **in** *exE* [*OF real-arch-pow2*], *blast*)
done
}
moreover have *f contour-integrable-on* (*linepath a b* +++ *linepath b c* +++ *linepath c a*)
by *simp* (*meson contf continuous-on-subset contour-integrable-continuous-linepath segments-subset-convex-hull*(*1*) *segments-subset-convex-hull*(*3*) *segments-subset-convex-hull*(*5*))
ultimately show *?thesis*
using *has-contour-integral-integral* **by** *fastforce*
qed

52.17 Version needing function holomorphic in interior only

lemma *Cauchy-theorem-flat-lemma*:
assumes *f*: *continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
and *c*: *c* $-$ *a* = *k* $*_R$ (*b* $-$ *a*)
and *k*: *0* $\leq$ *k*
shows *contour-integral* (*linepath a b*) *f* + *contour-integral* (*linepath b c*) *f* + *contour-integral* (*linepath c a*) *f* = *0*
proof $-$
have *fabc*: *continuous-on* (*closed-segment a b*) *f continuous-on* (*closed-segment b c*) *f continuous-on* (*closed-segment c a*) *f*
using *f continuous-on-subset segments-subset-convex-hull* **by** *metis*+
show *?thesis*

proof (*cases* $k \leq 1$)
case *True* **show** *?thesis*
by (*simp add*: *contour-integral-split* [*OF fabc*(*1*) *k True c*] *contour-integral-reverse-linepath fabc*)
next
case *False* **then show** *?thesis*
using *fabc c*
apply (*subst contour-integral-split* [*of a c f* 1/*k b*, *symmetric*])
apply (*metis closed-segment-commute fabc*(*3*))
apply (*auto simp*: *k contour-integral-reverse-linepath*)
done
qed
qed

lemma *Cauchy-theorem-flat*:
assumes *f*: *continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
and *c*: $c - a = k *_R (b - a)$
shows *contour-integral* (*linepath a b*) *f* +
contour-integral (*linepath b c*) *f* +
contour-integral (*linepath c a*) *f* = *0*
proof (*cases* $0 \leq k$)
case *True* **with** *assms* **show** *?thesis*
by (*blast intro*: *Cauchy-theorem-flat-lemma*)
next
case *False*
have *continuous-on* (*closed-segment a b*) *f continuous-on* (*closed-segment b c*) *f continuous-on* (*closed-segment c a*) *f*
using *f continuous-on-subset segments-subset-convex-hull* **by** *metis+*
moreover have *contour-integral* (*linepath b a*) *f* + *contour-integral* (*linepath a c*) *f* +
contour-integral (*linepath c b*) *f* = *0*
apply (*rule Cauchy-theorem-flat-lemma* [*of b a c f* 1−*k*])
using *False c*
apply (*auto simp*: *f insert-commute scaleR-conv-of-real algebra-simps*)
done
ultimately show *?thesis*
apply (*auto simp*: *contour-integral-reverse-linepath*)
using *add-eq-0-iff* **by** *force*
qed

lemma *Cauchy-theorem-triangle-interior*:
assumes *contf*: *continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
and *holf*: *f holomorphic-on interior* (*convex hull* {*a*,*b*,*c*})
shows (*f has-contour-integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
proof −
have *fabc*: *continuous-on* (*closed-segment a b*) *f continuous-on* (*closed-segment b c*) *f continuous-on* (*closed-segment c a*) *f*

```
    using contf continuous-on-subset segments-subset-convex-hull by metis+
  have bounded (f ` (convex hull {a,b,c}))
   by (simp add: compact-continuous-image compact-convex-hull compact-imp-bounded
contf)
  then obtain B where 0 < B and Bnf: ⋀x. x ∈ convex hull {a,b,c} ⟹ norm
(f x) ≤ B
     by (auto simp: dest!: bounded-pos [THEN iffD1])
  have bounded (convex hull {a,b,c})
    by (simp add: bounded-convex-hull)
  then obtain C where C: 0 < C and Cno: ⋀y. y ∈ convex hull {a,b,c} ⟹
norm y < C
    using bounded-pos-less by blast
  then have diff-2C: norm(x - y) ≤ 2*C
          if x: x ∈ convex hull {a, b, c} and y: y ∈ convex hull {a, b, c} for x y
  proof -
    have cmod x ≤ C
      using x by (meson Cno not-le not-less-iff-gr-or-eq)
    hence cmod (x - y) ≤ C + C
       using y by (meson Cno add-mono-thms-linordered-field(4) less-eq-real-def
norm-triangle-ineq4 order-trans)
    thus cmod (x - y) ≤ 2 * C
      by (metis mult-2)
  qed
  have contf': continuous-on (convex hull {b,a,c}) f
    using contf by (simp add: insert-commute)
  { fix y::complex
     assume fy: (f has-contour-integral y) (linepath a b +++ linepath b c +++
linepath c a)
       and ynz: y ≠ 0
    have pi-eq-y: contour-integral (linepath a b) f + contour-integral (linepath b c)
f + contour-integral (linepath c a) f = y
      by (rule has-chain-integral-chain-integral3 [OF fy])
    have ?thesis
    proof (cases c=a ∨ a=b ∨ b=c)
      case True then show ?thesis
        using Cauchy-theorem-flat [OF contf, of 0]
        using has-chain-integral-chain-integral3 [OF fy] ynz
        by (force simp: fabc contour-integral-reverse-linepath)
    next
      case False
      then have car3: card {a, b, c} = Suc (DIM(complex))
        by auto
      { assume interior(convex hull {a,b,c}) = {}
        then have collinear{a,b,c}
          using interior-convex-hull-eq-empty [OF car3]
          by (simp add: collinear-3-eq-affine-dependent)
        then have False
          using False
          apply (clarsimp simp add: collinear-3 collinear-lemma)
```

apply (*drule Cauchy-theorem-flat* [*OF contf'*])
using *pi-eq-y ynz*
apply (*simp add*: *fabc add-eq-0-iff contour-integral-reverse-linepath*)
done
}
then obtain *d* **where** *d*: *d* ∈ *interior* (*convex hull* {*a*, *b*, *c*})
by *blast*
{ **fix** *d1*
assume *d1-pos*: *0* < *d1*
and *d1*: ⋀*x x'*. ⟦*x*∈*convex hull* {*a*, *b*, *c*}; *x'*∈*convex hull* {*a*, *b*, *c*}; *cmod* (*x'* − *x*) < *d1*⟧
⟹ *cmod* (*f x'* − *f x*) < *cmod y* / (*24* ∗ *C*)
def *e* ≡ *min 1* (*min* (*d1*/(*4*∗*C*)) ((*norm y* / *24* / *C*) / *B*))
def *shrink* ≡ λ*x*. *x* − *e* ∗*R* (*x* − *d*)
let *?pathint* = λ*x y*. *contour-integral*(*linepath x y*) *f*
have *e*: *0* < *e e* ≤ *1 e* ≤ *d1* / (*4* ∗ *C*) *e* ≤ *cmod y* / *24* / *C* / *B*
using *d1-pos* ‹*C*>*0*› ‹*B*>*0*› *ynz* **by** (*simp-all add*: *e-def*)
then have *eCB*: *24* ∗ *e* ∗ *C* ∗ *B* ≤ *cmod y*
using ‹*C*>*0*› ‹*B*>*0*› **by** (*simp add*: *field-simps*)
have *e-le-d1*: *e* ∗ (*4* ∗ *C*) ≤ *d1*
using *e* ‹*C*>*0*› **by** (*simp add*: *field-simps*)
have *shrink a* ∈ *interior*(*convex hull* {*a*,*b*,*c*})
shrink b ∈ *interior*(*convex hull* {*a*,*b*,*c*})
shrink c ∈ *interior*(*convex hull* {*a*,*b*,*c*})
using *d e* **by** (*auto simp*: *hull-inc mem-interior-convex-shrink shrink-def*)
then have *fhp0*: (*f has-contour-integral 0*)
(*linepath* (*shrink a*) (*shrink b*) +++ *linepath* (*shrink b*) (*shrink c*) +++ *linepath* (*shrink c*) (*shrink a*))
by (*simp add*: *Cauchy-theorem-triangle holomorphic-on-subset* [*OF holf*] *hull-minimal convex-interior*)
then have *f-0-shrink*: *?pathint* (*shrink a*) (*shrink b*) + *?pathint* (*shrink b*) (*shrink c*) + *?pathint* (*shrink c*) (*shrink a*) = *0*
by (*simp add*: *has-chain-integral-chain-integral3*)
have *fpi-abc*: *f contour-integrable-on linepath* (*shrink a*) (*shrink b*)
f contour-integrable-on linepath (*shrink b*) (*shrink c*)
f contour-integrable-on linepath (*shrink c*) (*shrink a*)
using *fhp0* **by** (*auto simp*: *valid-path-join dest*: *has-contour-integral-integrable*)
have *cmod-shr*: ⋀*x y*. *cmod* (*shrink y* − *shrink x* − (*y* − *x*)) = *e* ∗ *cmod* (*x* − *y*)
using *e* **by** (*simp add*: *shrink-def real-vector.scale-right-diff-distrib* [*symmetric*])
have *sh-eq*: ⋀*a b d*::*complex*. (*b* − *e* ∗*R* (*b* − *d*)) − (*a* − *e* ∗*R* (*a* − *d*)) − (*b* − *a*) = *e* ∗*R* (*a* − *b*)
by (*simp add*: *algebra-simps*)
have *cmod y* / (*24* ∗ *C*) ≤ *cmod y* / *cmod* (*b* − *a*) / *12*
using *False* ‹*C*>*0*› *diff-2C* [*of b a*] *ynz*
by (*auto simp*: *divide-simps hull-inc*)
have *less-C*: ⟦*u* ∈ *convex hull* {*a*, *b*, *c*}; *0* ≤ *x*; *x* ≤ *1*⟧ ⟹ *x* ∗ *cmod u* < *C* **for** *x u*

apply (*cases x=0*, *simp add*: ‹*0<C*›)
using *Cno* [*of u*] *mult-left-le-one-le* [*of cmod u x*] *le-less-trans norm-ge-zero*
by *blast*
{ fix *u v*
assume *uv*: *u* ∈ *convex hull* {*a*, *b*, *c*} *v* ∈ *convex hull* {*a*, *b*, *c*} *u≠v*
and *fpi-uv*: *f contour-integrable-on linepath* (*shrink u*) (*shrink v*)
have *shr-uv*: *shrink u* ∈ *interior*(*convex hull* {*a*,*b*,*c*})
shrink v ∈ *interior*(*convex hull* {*a*,*b*,*c*})
using *d e uv*
by (*auto simp*: *hull-inc mem-interior-convex-shrink shrink-def*)
have *cmod-fuv*: ⋀*x*. *0≤x* ⟹ *x≤1* ⟹ *cmod* (*f* (*linepath* (*shrink u*) (*shrink v*) *x*)) ≤ *B*
using *shr-uv* **by** (*blast intro*: *Bnf linepath-in-convex-hull interior-subset* [*THEN subsetD*])
have *By-uv*: *B* ∗ (*12* ∗ (*e* ∗ *cmod* (*u* − *v*))) ≤ *cmod y*
apply (*rule order-trans* [*OF - eCB*])
using *e* ‹*B>0*› *diff-2C* [*of u v*] *uv*
by (*auto simp*: *field-simps*)
{ fix *x*::*real* **assume** *x*: *0≤x x≤1*
have *cmod-less-4C*: *cmod* ((*1* − *x*) ∗*R* *u* − (*1* − *x*) ∗*R* *d*) + *cmod* (*x* ∗*R* *v* − *x* ∗*R* *d*) < (*C*+*C*) + (*C*+*C*)
apply (*rule add-strict-mono*; *rule norm-triangle-half-l* [*of - 0*])
using *uv x d interior-subset*
apply (*auto simp*: *hull-inc intro*!: *less-C*)
done
have *ll*: *linepath* (*shrink u*) (*shrink v*) *x* − *linepath u v x* = −*e* ∗ ((*1* − *x*) ∗*R* (*u* − *d*) + *x* ∗*R* (*v* − *d*))
by (*simp add*: *linepath-def shrink-def algebra-simps scaleR-conv-of-real*)
have *cmod-less-dt*: *cmod* (*linepath* (*shrink u*) (*shrink v*) *x* − *linepath u v x*) < *d1*
using ‹*e>0*›
apply (*simp add*: *ll norm-mult scaleR-diff-right*)
apply (*rule less-le-trans* [*OF - e-le-d1*])
using *cmod-less-4C*
apply (*force intro*: *norm-triangle-lt*)
done
have *cmod* (*f* (*linepath* (*shrink u*) (*shrink v*) *x*) − *f* (*linepath u v x*)) < *cmod y* / (*24* ∗ *C*)
using *x uv shr-uv cmod-less-dt*
by (*auto simp*: *hull-inc intro*: *d1 interior-subset* [*THEN subsetD*] *linepath-in-convex-hull*)
also have ... ≤ *cmod y* / *cmod* (*v* − *u*) / *12*
using *False uv* ‹*C>0*› *diff-2C* [*of v u*] *ynz*
by (*auto simp*: *divide-simps hull-inc*)
finally have *cmod* (*f* (*linepath* (*shrink u*) (*shrink v*) *x*) − *f* (*linepath u v x*)) ≤ *cmod y* / *cmod* (*v* − *u*) / *12*
by *simp*
then have *cmod-12-le*: *cmod* (*v* − *u*) ∗ *cmod* (*f* (*linepath* (*shrink u*) (*shrink v*) *x*) − *f* (*linepath u v x*)) ∗ *12* ≤ *cmod y*

```
          using uv False by (auto simp: field-simps)
         have cmod (f (linepath (shrink u) (shrink v) x)) * cmod (shrink v −
shrink u − (v − u)) +
                cmod (v − u) * cmod (f (linepath (shrink u) (shrink v) x) − f
(linepath u v x))
              ≤ cmod y / 6
           apply (rule order-trans [of - B*((norm y / 24 / C / B)*2*C) +
(2*C)*(norm y /24 / C)])
          apply (rule add-mono [OF mult-mono])
          using By-uv e ‹0 < B› ‹0 < C› x ynz
          apply (simp-all add: cmod-fuv cmod-shr cmod-12-le hull-inc)
          apply (simp add: field-simps)
          done
        } note cmod-diff-le = this
        have f-uv: continuous-on (closed-segment u v) f
      by (blast intro: uv continuous-on-subset [OF contf closed-segment-subset-convex-hull])
        have **: ⋀f' x' f x::complex. f'*x' − f*x = f'*(x' − x) + x*(f' − f)
          by (simp add: algebra-simps)
        have norm (?pathint (shrink u) (shrink v) − ?pathint u v) ≤ norm y / 6
          apply (rule order-trans)
          apply (rule has-integral-bound
                [of B*(norm y /24/C/B)*2*C + (2*C)*(norm y/24/C)
                  λx. f(linepath (shrink u) (shrink v) x) * (shrink v − shrink u)
− f(linepath u v x)*(v − u)
                  - 0 1 ])
          using ynz ‹0 < B› ‹0 < C›
          apply (simp-all del: le-divide-eq-numeral1)
       apply (simp add: has-integral-sub has-contour-integral-linepath [symmetric]
has-contour-integral-integral
                    fpi-uv f-uv contour-integrable-continuous-linepath, clarify)
          apply (simp only: **)
       apply (simp add: norm-triangle-le norm-mult cmod-diff-le del: le-divide-eq-numeral1)
          done
        } note * = this
        have norm (?pathint (shrink a) (shrink b) − ?pathint a b) ≤ norm y / 6
          using False fpi-abc by (rule-tac *) (auto simp: hull-inc)
        moreover
        have norm (?pathint (shrink b) (shrink c) − ?pathint b c) ≤ norm y / 6
          using False fpi-abc by (rule-tac *) (auto simp: hull-inc)
        moreover
        have norm (?pathint (shrink c) (shrink a) − ?pathint c a) ≤ norm y / 6
          using False fpi-abc by (rule-tac *) (auto simp: hull-inc)
        ultimately
        have norm((?pathint (shrink a) (shrink b) − ?pathint a b) +
              (?pathint (shrink b) (shrink c) − ?pathint b c) + (?pathint (shrink
c) (shrink a) − ?pathint c a))
            ≤ norm y / 6 + norm y / 6 + norm y / 6
          by (metis norm-triangle-le add-mono)
        also have ... = norm y / 2
```

by *simp*
finally have *norm((?pathint (shrink a) (shrink b) + ?pathint (shrink b) (shrink c) + ?pathint (shrink c) (shrink a)) −*
(?pathint a b + ?pathint b c + ?pathint c a))
≤ norm y / 2
by (*simp add*: *algebra-simps*)
then
have *norm(?pathint a b + ?pathint b c + ?pathint c a) ≤ norm y / 2*
by (*simp add*: *f-0-shrink*) (*metis* (*mono-tags*) *add.commute minus-add-distrib norm-minus-cancel uminus-add-conv-diff*)
then have *False*
using *pi-eq-y ynz* **by** *auto*
}
moreover have *uniformly-continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
by (*simp add*: *contf compact-convex-hull compact-uniformly-continuous*)
ultimately have *False*
unfolding *uniformly-continuous-on-def*
by (*force simp*: *ynz* ‹*0 < C*› *dist-norm*)
then show *?thesis* **..**
qed
}
moreover have *f contour-integrable-on* (*linepath a b +++ linepath b c +++ linepath c a*)
using *fabc contour-integrable-continuous-linepath* **by** *auto*
ultimately show *?thesis*
using *has-contour-integral-integral* **by** *fastforce*
qed

52.18 Version allowing finite number of exceptional points

lemma *Cauchy-theorem-triangle-cofinite*:
assumes *continuous-on* (*convex hull* {*a*,*b*,*c*}) *f*
and *finite s*
and (⋀*x*. *x* ∈ *interior*(*convex hull* {*a*,*b*,*c*}) − *s* ⟹ *f complex-differentiable* (*at x*))
shows (*f has-contour-integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
using *assms*
proof (*induction card s arbitrary*: *a b c s rule*: *less-induct*)
case (*less s a b c*)
show *?case*
proof (*cases s*={})
case *True* **with** *less* **show** *?thesis*
by (*fastforce simp*: *holomorphic-on-def complex-differentiable-at-within Cauchy-theorem-triangle-interior*)
next
case *False*
then obtain *d s′* **where** *d*: *s* = *insert d s′ d* ∉ *s′*
by (*meson Set.set-insert all-not-in-conv*)

then show *?thesis*
proof (*cases d ∈ convex hull {a,b,c}*)
case *False*
show (*f has-contour-integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
apply (*rule less.hyps* [*of s'*])
using *False d ‹finite s› interior-subset*
apply (*auto intro*!: *less.prems*)
done
next
case *True*
have ∗: *convex hull {a, b, d} ⊆ convex hull {a, b, c}*
by (*meson True hull-subset insert-subset convex-hull-subset*)
have *abd*: (*f has-contour-integral 0*) (*linepath a b +++ linepath b d +++ linepath d a*)
apply (*rule less.hyps* [*of s'*])
using *True d ‹finite s› not-in-interior-convex-hull-3*
apply (*auto intro*!: *less.prems continuous-on-subset* [*OF - ∗*])
apply (*metis ∗ insert-absorb insert-subset interior-mono*)
done
have ∗: *convex hull {b, c, d} ⊆ convex hull {a, b, c}*
by (*meson True hull-subset insert-subset convex-hull-subset*)
have *bcd*: (*f has-contour-integral 0*) (*linepath b c +++ linepath c d +++ linepath d b*)
apply (*rule less.hyps* [*of s'*])
using *True d ‹finite s› not-in-interior-convex-hull-3*
apply (*auto intro*!: *less.prems continuous-on-subset* [*OF - ∗*])
apply (*metis ∗ insert-absorb insert-subset interior-mono*)
done
have ∗: *convex hull {c, a, d} ⊆ convex hull {a, b, c}*
by (*meson True hull-subset insert-subset convex-hull-subset*)
have *cad*: (*f has-contour-integral 0*) (*linepath c a +++ linepath a d +++ linepath d c*)
apply (*rule less.hyps* [*of s'*])
using *True d ‹finite s› not-in-interior-convex-hull-3*
apply (*auto intro*!: *less.prems continuous-on-subset* [*OF - ∗*])
apply (*metis ∗ insert-absorb insert-subset interior-mono*)
done
have *f contour-integrable-on linepath a b*
using *less.prems*
by (*metis continuous-on-subset insert-commute contour-integrable-continuous-linepath segments-subset-convex-hull*(*3*))
moreover have *f contour-integrable-on linepath b c*
using *less.prems*
by (*metis continuous-on-subset contour-integrable-continuous-linepath segments-subset-convex-hull*(*3*))
moreover have *f contour-integrable-on linepath c a*
using *less.prems*
by (*metis continuous-on-subset insert-commute contour-integrable-continuous-linepath segments-subset-convex-hull*(*3*))

ultimately have *fpi*: *f contour-integrable-on* (*linepath a b +++ linepath b c +++ linepath c a*)
by *auto*
{ fix *y*::*complex*
assume *fy*: (*f has-contour-integral y*) (*linepath a b +++ linepath b c +++ linepath c a*)
and *ynz*: *y* ≠ *0*
have *cont-ad*: *continuous-on* (*closed-segment a d*) *f*
by (*meson* ∗ *continuous-on-subset less.prems*(*1*) *segments-subset-convex-hull*(*3*))
have *cont-bd*: *continuous-on* (*closed-segment b d*) *f*
by (*meson True closed-segment-subset-convex-hull continuous-on-subset hull-subset insert-subset less.prems*(*1*))
have *cont-cd*: *continuous-on* (*closed-segment c d*) *f*
by (*meson* ∗ *continuous-on-subset less.prems*(*1*) *segments-subset-convex-hull*(*2*))
have *contour-integral* (*linepath a b*) *f* = − (*contour-integral* (*linepath b d*) *f* + (*contour-integral* (*linepath d a*) *f*))
contour-integral (*linepath b c*) *f* = − (*contour-integral* (*linepath c d*) *f* + (*contour-integral* (*linepath d b*) *f*))
contour-integral (*linepath c a*) *f* = − (*contour-integral* (*linepath a d*) *f* + *contour-integral* (*linepath d c*) *f*)
using *has-chain-integral-chain-integral3* [*OF abd*]
has-chain-integral-chain-integral3 [*OF bcd*]
has-chain-integral-chain-integral3 [*OF cad*]
by (*simp-all add*: *algebra-simps add-eq-0-iff*)
then have *?thesis*
using *cont-ad cont-bd cont-cd fy has-chain-integral-chain-integral3 contour-integral-reverse-linepath*
by *fastforce*
}
then show *?thesis*
using *fpi contour-integrable-on-def* **by** *blast*
qed
qed
qed

52.19 Cauchy's theorem for an open starlike set

lemma *starlike-convex-subset*:
assumes *s*: *a* ∈ *s closed-segment b c* ⊆ *s* **and** *subs*: ⋀*x*. *x* ∈ *s* ⟹ *closed-segment a x* ⊆ *s*
shows *convex hull* {*a*,*b*,*c*} ⊆ *s*
using *s*
apply (*clarsimp simp add*: *convex-hull-insert* [*of* {*b*,*c*} *a*] *segment-convex-hull*)
apply (*meson subs convexD convex-closed-segment ends-in-segment*(*1*) *ends-in-segment*(*2*) *subsetCE*)
done

lemma *triangle-contour-integrals-starlike-primitive*:
assumes *contf*: *continuous-on s f*
and *s*: *a* ∈ *s open s*

and x: $x \in s$
and *subs*: $\bigwedge y.\ y \in s \Longrightarrow$ *closed-segment a y* $\subseteq s$
and *zer*: $\bigwedge b\ c.$ *closed-segment b c* $\subseteq s$
$\Longrightarrow$ *contour-integral* (*linepath a b*) f + *contour-integral* (*linepath b c*) f +
contour-integral (*linepath c a*) $f = 0$
shows ((*λx. contour-integral*(*linepath a x*) f) *has-field-derivative f x*) (*at x*)
proof −
let *?pathint* = *λx y. contour-integral*(*linepath x y*) f
{ fix e y
assume e: $0 < e$ **and** *bxe*: *ball x e* $\subseteq s$ **and** *close*: *cmod* $(y - x) < e$
have y: $y \in s$
using *bxe close* **by** (*force simp*: *dist-norm norm-minus-commute*)
have *cont-ayf*: *continuous-on* (*closed-segment a y*) f
using *contf continuous-on-subset subs y* **by** *blast*
have *xys*: *closed-segment x y* $\subseteq s$
apply (*rule order-trans* [*OF* - *bxe*])
using *close*
by (*auto simp*: *dist-norm ball-def norm-minus-commute dest*: *segment-bound*)
have *?pathint a y − ?pathint a x = ?pathint x y*
using *zer* [*OF xys*] *contour-integral-reverse-linepath* [*OF cont-ayf*] *add-eq-0-iff* **by** *force*
} note [*simp*] = *this*
{ fix *e*::*real*
assume e: $0 < e$
have *cont-atx*: *continuous* (*at x*) f
using *x s contf continuous-on-eq-continuous-at* **by** *blast*
then obtain *d1* **where** *d1*: *d1*>*0* **and** *d1-less*: $\bigwedge y.$ *cmod* $(y - x) <$ *d1* $\Longrightarrow$ *cmod* $(f\ y - f\ x) < e/2$
unfolding *continuous-at Lim-at dist-norm* **using** e
by (*drule-tac x*=*e*/2 **in** *spec*) *force*
obtain *d2* **where** *d2*: *d2*>*0 ball x d2* $\subseteq s$ **using** ‹*open s*› *x*
by (*auto simp*: *open-contains-ball*)
have *dpos*: *min d1 d2* > *0* **using** *d1 d2* **by** *simp*
{ fix y
assume *yx*: $y \neq x$ **and** *close*: *cmod* $(y - x) <$ *min d1 d2*
have y: $y \in s$
using *d2 close* **by** (*force simp*: *dist-norm norm-minus-commute*)
have *fxy*: f *contour-integrable-on linepath x y*
apply (*rule contour-integrable-continuous-linepath*)
apply (*rule continuous-on-subset* [*OF contf*])
using *close d2*
apply (*auto simp*: *dist-norm norm-minus-commute dest*!: *segment-bound*(*1*))
done
then obtain i **where** i: (f *has-contour-integral i*) (*linepath x y*)
by (*auto simp*: *contour-integrable-on-def*)
then have ((*λw.* $f\ w - f\ x$) *has-contour-integral* ($i - f\ x * (y - x)$)) (*linepath x y*)
by (*rule has-contour-integral-diff* [*OF* - *has-contour-integral-const-linepath*])

```
    then have cmod (i - f x * (y - x)) ≤ e / 2 * cmod (y - x)
      apply (rule has-contour-integral-bound-linepath [where B = e/2])
      using e apply simp
      apply (rule d1-less [THEN less-imp-le])
      using close segment-bound
      apply force
      done
    also have ... < e * cmod (y - x)
      by (simp add: e yx)
    finally have cmod (?pathint x y - f x * (y-x)) / cmod (y-x) < e
      using i yx  by (simp add: contour-integral-unique divide-less-eq)
   }
   then have ∃ d>0. ∀ y. y ≠ x ∧ cmod (y-x) < d ⟶ cmod (?pathint x y - f
x * (y-x)) / cmod (y-x) < e
    using dpos by blast
  }
  then have *: (λy. (?pathint x y - f x * (y - x)) /R cmod (y - x)) -x→ 0
   by (simp add: Lim-at dist-norm inverse-eq-divide)
  show ?thesis
   apply (simp add: has-field-derivative-def has-derivative-at bounded-linear-mult-right)
   apply (rule Lim-transform [OF * Lim-eventually])
   apply (simp add: inverse-eq-divide [symmetric] eventually-at)
   using ‹open s› x
   apply (force simp: dist-norm open-contains-ball)
   done
qed



lemma holomorphic-starlike-primitive:
  assumes contf: continuous-on s f
      and s: starlike s and os: open s
      and k: finite k
      and fcd: ⋀x. x ∈ s - k ⟹ f complex-differentiable at x
    shows ∃ g. ∀ x ∈ s. (g has-field-derivative f x) (at x)
proof -
  obtain a where a: a∈s and a-cs: ⋀x. x∈s ⟹ closed-segment a x ⊆ s
    using s by (auto simp: starlike-def)
  { fix x b c
    assume x ∈ s closed-segment b c ⊆ s
    then have abcs: convex hull {a, b, c} ⊆ s
      by (simp add: a a-cs starlike-convex-subset)
    then have *: continuous-on (convex hull {a, b, c}) f
      by (simp add: continuous-on-subset [OF contf])
    have (f has-contour-integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
      apply (rule Cauchy-theorem-triangle-cofinite [OF - k])
      using abcs apply (simp add: continuous-on-subset [OF contf])
      using * abcs interior-subset apply (auto intro: fcd)
```

```
    done
  } note 0 = this
  show ?thesis
    apply (intro exI ballI)
     apply (rule triangle-contour-integrals-starlike-primitive [OF contf a os], as-
sumption)
    apply (metis a-cs)
    apply (metis has-chain-integral-chain-integral3 0)
    done
qed

lemma Cauchy-theorem-starlike:
 ⟦open s; starlike s; finite k; continuous-on s f;
   ⋀x. x ∈ s − k ⟹ f complex-differentiable at x;
   valid-path g; path-image g ⊆ s; pathfinish g = pathstart g⟧
   ⟹ (f has-contour-integral 0) g
 by (metis holomorphic-starlike-primitive Cauchy-theorem-primitive at-within-open)

lemma Cauchy-theorem-starlike-simple:
  ⟦open s; starlike s; f holomorphic-on s; valid-path g; path-image g ⊆ s; pathfinish
g = pathstart g⟧
   ⟹ (f has-contour-integral 0) g
apply (rule Cauchy-theorem-starlike [OF - - finite.emptyI])
apply (simp-all add: holomorphic-on-imp-continuous-on)
apply (metis at-within-open holomorphic-on-def)
done
```

52.20 Cauchy's theorem for a convex set

For a convex set we can avoid assuming openness and boundary analyticity

```
lemma triangle-contour-integrals-convex-primitive:
  assumes contf: continuous-on s f
      and s: a ∈ s convex s
      and x: x ∈ s
      and zer: ⋀b c. ⟦b ∈ s; c ∈ s⟧
                ⟹ contour-integral (linepath a b) f + contour-integral (linepath b
c) f +
                     contour-integral (linepath c a) f = 0
    shows ((λx. contour-integral(linepath a x) f) has-field-derivative f x) (at x
within s)
proof −
  let ?pathint = λx y. contour-integral(linepath x y) f
  { fix y
    assume y: y ∈ s
    have cont-ayf: continuous-on (closed-segment a y) f
      using s y  by (meson contf continuous-on-subset convex-contains-segment)
    have xys: closed-segment x y ⊆ s
      using convex-contains-segment s x y by auto
    have ?pathint a y − ?pathint a x = ?pathint x y
```

using *zer* [*OF x y*] *contour-integral-reverse-linepath* [*OF cont-ayf*] *add-eq-0-iff*
by *force*
} note [*simp*] = *this*
{ fix *e*::*real*
assume *e*: *0* < *e*
have *cont-atx*: *continuous* (*at x within s*) *f*
using *x s contf* **by** (*simp add*: *continuous-on-eq-continuous-within*)
then obtain *d1* **where** *d1*: *d1>0* **and** *d1-less*: ⋀*y*. ⟦*y* ∈ *s*; *cmod* (*y* − *x*) < *d1*⟧ ⟹ *cmod* (*f y* − *f x*) < *e*/*2*
unfolding *continuous-within Lim-within dist-norm* **using** *e*
by (*drule-tac x*=*e*/*2* **in** *spec*) *force*
{ fix *y*
assume *yx*: *y* ≠ *x* **and** *close*: *cmod* (*y* − *x*) < *d1* **and** *y*: *y* ∈ *s*
have *fxy*: *f contour-integrable-on linepath x y*
using *convex-contains-segment s x y*
by (*blast intro*!: *contour-integrable-continuous-linepath continuous-on-subset*
[*OF contf*])
then obtain *i* **where** *i*: (*f has-contour-integral i*) (*linepath x y*)
by (*auto simp*: *contour-integrable-on-def*)
then have ((λ*w*. *f w* − *f x*) *has-contour-integral* (*i* − *f x* ∗ (*y* − *x*))) (*linepath*
x y)
by (*rule has-contour-integral-diff* [*OF - has-contour-integral-const-linepath*])
then have *cmod* (*i* − *f x* ∗ (*y* − *x*)) ≤ *e* / *2* ∗ *cmod* (*y* − *x*)
apply (*rule has-contour-integral-bound-linepath* [**where** *B* = *e*/*2*])
using *e* **apply** *simp*
apply (*rule d1-less* [*THEN less-imp-le*])
using *convex-contains-segment s*(*2*) *x y* **apply** *blast*
using *close segment-bound*(*1*) **apply** *fastforce*
done
also have ... < *e* ∗ *cmod* (*y* − *x*)
by (*simp add*: *e yx*)
finally have *cmod* (*?pathint x y* − *f x* ∗ (*y*−*x*)) / *cmod* (*y*−*x*) < *e*
using *i yx* **by** (*simp add*: *contour-integral-unique divide-less-eq*)
}
then have ∃ *d*>*0*. ∀ *y*∈*s*. *y* ≠ *x* ∧ *cmod* (*y*−*x*) < *d* ⟶ *cmod* (*?pathint x y*
− *f x* ∗ (*y*−*x*)) / *cmod* (*y*−*x*) < *e*
using *d1* **by** *blast*
}
then have ∗: ((λ*y*. (*contour-integral* (*linepath x y*) *f* − *f x* ∗ (*y* − *x*)) /$_R$ *cmod*
(*y* − *x*)) ⟶ *0*) (*at x within s*)
by (*simp add*: *Lim-within dist-norm inverse-eq-divide*)
show *?thesis*
apply (*simp add*: *has-field-derivative-def has-derivative-within bounded-linear-mult-right*)
apply (*rule Lim-transform* [*OF* ∗ *Lim-eventually*])
using *linordered-field-no-ub*
apply (*force simp*: *inverse-eq-divide* [*symmetric*] *eventually-at*)
done
qed

lemma *contour-integral-convex-primitive*:
⟦*convex s*; *continuous-on s f*;
⋀*a b c*. ⟦*a* ∈ *s*; *b* ∈ *s*; *c* ∈ *s*⟧ ⟹ (*f has-contour-integral 0*) (*linepath a b* +++ *linepath b c* +++ *linepath c a*)⟧
⟹ ∃ *g*. ∀ *x* ∈ *s*. (*g has-field-derivative f x*) (*at x within s*)
apply (*cases s*={})
apply (*simp-all add*: *ex-in-conv* [*symmetric*])
apply (*blast intro*: *triangle-contour-integrals-convex-primitive has-chain-integral-chain-integral3*)
done

lemma *holomorphic-convex-primitive*:
⟦*convex s*; *finite k*; *continuous-on s f*;
⋀*x*. *x* ∈ *interior s* − *k* ⟹ *f complex-differentiable at x*⟧
⟹ ∃ *g*. ∀ *x* ∈ *s*. (*g has-field-derivative f x*) (*at x within s*)
apply (*rule contour-integral-convex-primitive* [*OF* - - *Cauchy-theorem-triangle-cofinite*])
prefer *3*
apply (*erule continuous-on-subset*)
apply (*simp add*: *subset-hull continuous-on-subset*, *assumption*+)
by (*metis Diff-iff convex-contains-segment insert-absorb insert-subset interior-mono segment-convex-hull subset-hull*)

lemma *Cauchy-theorem-convex*:
⟦*continuous-on s f*; *convex s*; *finite k*;
⋀*x*. *x* ∈ *interior s* − *k* ⟹ *f complex-differentiable at x*;
valid-path g; *path-image g* ⊆ *s*;
pathfinish g = *pathstart g*⟧ ⟹ (*f has-contour-integral 0*) *g*
by (*metis holomorphic-convex-primitive Cauchy-theorem-primitive*)

lemma *Cauchy-theorem-convex-simple*:
⟦*f holomorphic-on s*; *convex s*;
valid-path g; *path-image g* ⊆ *s*;
pathfinish g = *pathstart g*⟧ ⟹ (*f has-contour-integral 0*) *g*
apply (*rule Cauchy-theorem-convex*)
apply (*simp-all add*: *holomorphic-on-imp-continuous-on*)
apply (*rule finite.emptyI*)
using *at-within-interior holomorphic-on-def interior-subset* **by** *fastforce*

In particular for a disc

lemma *Cauchy-theorem-disc*:
⟦*finite k*; *continuous-on* (*cball a e*) *f*;
⋀*x*. *x* ∈ *ball a e* − *k* ⟹ *f complex-differentiable at x*;
valid-path g; *path-image g* ⊆ *cball a e*;
pathfinish g = *pathstart g*⟧ ⟹ (*f has-contour-integral 0*) *g*
apply (*rule Cauchy-theorem-convex*)
apply (*auto simp*: *convex-cball interior-cball*)
done

lemma *Cauchy-theorem-disc-simple*:
⟦*f holomorphic-on* (*ball a e*); *valid-path g*; *path-image g* ⊆ *ball a e*;

pathfinish g = pathstart g⟧ ⟹ (*f has-contour-integral 0*) *g*
by (*simp add*: *Cauchy-theorem-convex-simple*)

52.21 Generalize integrability to local primitives

lemma *contour-integral-local-primitive-lemma*:
fixes *f* :: *complex*⇒*complex*
shows
⟦*g piecewise-differentiable-on* {*a*..*b*};
⋀*x*. *x* ∈ *s* ⟹ (*f has-field-derivative f* ′ *x*) (*at x within s*);
⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g x* ∈ *s*⟧
⟹ (λ*x*. *f* ′ (*g x*) ∗ *vector-derivative g* (*at x within* {*a*..*b*}))
integrable-on {*a*..*b*}
apply (*cases cbox a b* = {}, *force*)
apply (*simp add*: *integrable-on-def*)
apply (*rule exI*)
apply (*rule contour-integral-primitive-lemma*, *assumption*+)
using *atLeastAtMost-iff* **by** *blast*

lemma *contour-integral-local-primitive-any*:
fixes *f* :: *complex* ⇒ *complex*
assumes *gpd*: *g piecewise-differentiable-on* {*a*..*b*}
and *dh*: ⋀*x*. *x* ∈ *s*
⟹ ∃ *d h*. 0 < *d* ∧
(∀ *y*. *norm*(*y* − *x*) < *d* ⟶ (*h has-field-derivative f y*) (*at y within s*))
and *gs*: ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g x* ∈ *s*
shows (λ*x*. *f*(*g x*) ∗ *vector-derivative g* (*at x*)) *integrable-on* {*a*..*b*}
proof −
{ fix *x*
assume *x*: *a* ≤ *x x* ≤ *b*
obtain *d h* **where** *d*: 0 < *d*
and *h*: (⋀*y*. *norm*(*y* − *g x*) < *d* ⟹ (*h has-field-derivative f y*) (*at y within s*))
using *x gs dh* **by** (*metis atLeastAtMost-iff*)
have *continuous-on* {*a*..*b*} *g* **using** *gpd piecewise-differentiable-on-def* **by** *blast*
then obtain *e* **where** *e*: *e*>0 **and** *lessd*: ⋀*x*′. *x* ′ ∈ {*a*..*b*} ⟹ |*x* ′ − *x*| < *e* ⟹ *cmod* (*g x* ′ − *g x*) < *d*
using *x d*
apply (*auto simp*: *dist-norm continuous-on-iff*)
apply (*drule-tac x*=*x* **in** *bspec*)
using *x* **apply** *simp*
apply (*drule-tac x*=*d* **in** *spec*, *auto*)
done
have ∃ *d*>0. ∀ *u v*. *u* ≤ *x* ∧ *x* ≤ *v* ∧ {*u*..*v*} ⊆ *ball x d* ∧ (*u* ≤ *v* ⟶ *a* ≤ *u* ∧ *v* ≤ *b*) ⟶
(λ*x*. *f* (*g x*) ∗ *vector-derivative g* (*at x*)) *integrable-on* {*u*..*v*}
apply (*rule-tac x*=*e* **in** *exI*)
using *e*

apply (*simp add*: *integrable-on-localized-vector-derivative* [*symmetric*], *clarify*)
apply (*rule-tac f* = *h* **and** *s* = *g* ' {*u*..*v*} **in** *contour-integral-local-primitive-lemma*)
apply (*meson atLeastatMost-subset-iff gpd piecewise-differentiable-on-subset*)
apply (*force simp*: *ball-def dist-norm intro*: *lessd gs DERIV-subset* [*OF h*], *force*)
done
} then
show *?thesis*
by (*force simp*: *intro*!: *integrable-on-little-subintervals* [*of a b*, *simplified*])
qed

lemma *contour-integral-local-primitive*:
fixes *f* :: *complex* ⇒ *complex*
assumes *g*: *valid-path g path-image g* ⊆ *s*
and *dh*: ⋀*x*. *x* ∈ *s*
⟹ ∃ *d h*. 0 < *d* ∧
(∀ *y*. *norm*(*y* − *x*) < *d* ⟶ (*h has-field-derivative f y*) (*at y within s*))
shows *f contour-integrable-on g*
using *g*
apply (*simp add*: *valid-path-def path-image-def contour-integrable-on-def has-contour-integral-def has-integral-localized-vector-derivative integrable-on-def* [*symmetric*])
using *contour-integral-local-primitive-any* [*OF - dh*]
by (*meson image-subset-iff piecewise-C1-imp-differentiable*)

In particular if a function is holomorphic

lemma *contour-integrable-holomorphic*:
assumes *contf*: *continuous-on s f*
and *os*: *open s*
and *k*: *finite k*
and *g*: *valid-path g path-image g* ⊆ *s*
and *fcd*: ⋀*x*. *x* ∈ *s* − *k* ⟹ *f complex-differentiable at x*
shows *f contour-integrable-on g*
proof −
{ fix *z*
assume *z*: *z* ∈ *s*
obtain *d* **where** *d*: *d*>0 *ball z d* ⊆ *s* **using** ‹*open s*› *z*
by (*auto simp*: *open-contains-ball*)
then have *contfb*: *continuous-on* (*ball z d*) *f*
using *contf continuous-on-subset* **by** *blast*
obtain *h* **where** ∀ *y*∈*ball z d*. (*h has-field-derivative f y*) (*at y within ball z d*)
using *holomorphic-convex-primitive* [*OF convex-ball k contfb fcd*] *d interior-subset* **by** *force*
then have ∀ *y*∈*ball z d*. (*h has-field-derivative f y*) (*at y within s*)
by (*metis Topology-Euclidean-Space.open-ball at-within-open d*(2) *os subsetCE*)
then have ∃ *h*. (∀ *y*. *cmod* (*y* − *z*) < *d* ⟶ (*h has-field-derivative f y*) (*at y within s*))
by (*force simp*: *dist-norm norm-minus-commute*)

then have $\exists\, d\ h.\ 0 < d \wedge (\forall\, y.\ cmod\ (y - z) < d \longrightarrow (h\ has\text{-}field\text{-}derivative\ f\ y)\ (at\ y\ within\ s))$
using *d* **by** *blast*
}
then show *?thesis*
by (*rule contour-integral-local-primitive* [*OF g*])
qed

lemma *contour-integrable-holomorphic-simple*:
assumes *fh*: *f holomorphic-on s*
and *os*: *open s*
and *g*: *valid-path g path-image g* $\subseteq$ *s*
shows *f contour-integrable-on g*
apply (*rule contour-integrable-holomorphic* [*OF - os Finite-Set.finite.emptyI g*])
apply (*simp add*: *fh holomorphic-on-imp-continuous-on*)
using *fh* **by** (*simp add*: *complex-differentiable-def holomorphic-on-open os*)

lemma *continuous-on-inversediff*:
fixes *z*:: *'a*::*real-normed-field* **shows** *z* $\notin$ *s* $\Longrightarrow$ *continuous-on s* ($\lambda w.\ 1\ /\ (w - z)$)
by (*rule continuous-intros* | *force*)+

corollary *contour-integrable-inversediff*:
$[\![$*valid-path g*; *z* $\notin$ *path-image g*$]\!] \Longrightarrow$ ($\lambda w.\ 1\ /\ (w-z)$) *contour-integrable-on g*
apply (*rule contour-integrable-holomorphic-simple* [*of - UNIV*$-\{z\}$])
apply (*auto simp*: *holomorphic-on-open open-delete intro*!: *derivative-eq-intros*)
done

Key fact that path integral is the same for a "nearby" path. This is the main lemma for the homotopy form of Cauchy's theorem and is also useful if we want "without loss of generality" to assume some nice properties of a path (e.g. smoothness). It can also be used to define the integrals of analytic functions over arbitrary continuous paths. This is just done for winding numbers now.

A technical definition to avoid duplication of similar proofs, for paths joined at the ends versus looping paths

definition *linked-paths* :: *bool* $\Rightarrow$ (*real* $\Rightarrow$ *'a*) $\Rightarrow$ (*real* $\Rightarrow$ *'a*::*topological-space*) $\Rightarrow$ *bool*
where *linked-paths atends g h* ==
(*if atends then pathstart h* = *pathstart g* $\wedge$ *pathfinish h* = *pathfinish g*
else pathfinish g = *pathstart g* $\wedge$ *pathfinish h* = *pathstart h*)

This formulation covers two cases: g and h share their start and end points; g and h both loop upon themselves.

lemma *contour-integral-nearby*:
assumes *os*: *open s* **and** *p*: *path p path-image p* $\subseteq$ *s*
shows

$\exists\, d.\ 0 < d \wedge$
$(\forall\, g\ h.\ \mathit{valid\text{-}path}\ g \wedge \mathit{valid\text{-}path}\ h \wedge$
$(\forall\, t \in \{0..1\}.\ \mathit{norm}(g\ t - p\ t) < d \wedge \mathit{norm}(h\ t - p\ t) < d) \wedge$
$\mathit{linked\text{-}paths}\ \mathit{atends}\ g\ h$
$\longrightarrow \mathit{path\text{-}image}\ g \subseteq s \wedge \mathit{path\text{-}image}\ h \subseteq s \wedge$
$(\forall f.\ f\ \mathit{holomorphic\text{-}on}\ s \longrightarrow \mathit{contour\text{-}integral}\ h\ f = \mathit{contour\text{-}integral}\ g\ f))$

proof −
have $\forall z.\ \exists e.\ z \in \mathit{path\text{-}image}\ p \longrightarrow 0 < e \wedge \mathit{ball}\ z\ e \subseteq s$
using *open-contains-ball os p(2)* **by** *blast*
then obtain *ee* **where** *ee*: $\bigwedge z.\ z \in \mathit{path\text{-}image}\ p \Longrightarrow 0 < \mathit{ee}\ z \wedge \mathit{ball}\ z\ (\mathit{ee}\ z) \subseteq s$
by *metis*
def *cover* $\equiv (\lambda z.\ \mathit{ball}\ z\ (\mathit{ee}\ z/3))$ ‘ $(\mathit{path\text{-}image}\ p)$
have *compact* $(\mathit{path\text{-}image}\ p)$
by $(\mathit{metis}\ p(1)\ \mathit{compact\text{-}path\text{-}image})$
moreover have $\mathit{path\text{-}image}\ p \subseteq (\bigcup c{\in}\mathit{path\text{-}image}\ p.\ \mathit{ball}\ c\ (\mathit{ee}\ c\ /\ 3))$
using *ee* **by** *auto*
ultimately have $\exists D \subseteq \mathit{cover}.\ \mathit{finite}\ D \wedge \mathit{path\text{-}image}\ p \subseteq \bigcup D$
by $(\mathit{simp\ add}\colon \mathit{compact\text{-}eq\text{-}heine\text{-}borel\ cover\text{-}def})$
then obtain *D* **where** *D*: $D \subseteq \mathit{cover}\ \mathit{finite}\ D\ \mathit{path\text{-}image}\ p \subseteq \bigcup D$
by *blast*
then obtain *k* **where** *k*: $k \subseteq \{0..1\}\ \mathit{finite}\ k$ **and** *D-eq*: $D = ((\lambda z.\ \mathit{ball}\ z\ (\mathit{ee}\ z\ /\ 3)) \circ p)$ ‘ *k*
apply $(\mathit{simp\ add}\colon \mathit{cover\text{-}def\ path\text{-}image\text{-}def\ image\text{-}comp})$
apply $(\mathit{blast\ dest!}\colon \mathit{finite\text{-}subset\text{-}image}\ [\mathit{OF}\ \langle \mathit{finite}\ D\rangle])$
done
then have *kne*: $k \neq \{\}$
using *D* **by** *auto*
have *pi*: $\bigwedge i.\ i \in k \Longrightarrow p\ i \in \mathit{path\text{-}image}\ p$
using *k* **by** $(\mathit{auto\ simp}\colon \mathit{path\text{-}image\text{-}def})$
then have *eepi*: $\bigwedge i.\ i \in k \Longrightarrow 0 < \mathit{ee}((p\ i))$
by $(\mathit{metis\ ee})$
def $e \equiv \mathit{Min}((\mathit{ee}\ o\ p)$ ‘ $k)$
have *fin-eep*: *finite* $((\mathit{ee}\ o\ p)$ ‘ $k)$
using *k* **by** *blast*
have *enz*: $0 < e$
using *ee k* **by** $(\mathit{simp\ add}\colon \mathit{kne\ e\text{-}def\ Min\text{-}gr\text{-}iff}\ [\mathit{OF\ fin\text{-}eep}]\ \mathit{eepi})$
have *uniformly-continuous-on* $\{0..1\}\ p$
using *p* **by** $(\mathit{simp\ add}\colon \mathit{path\text{-}def\ compact\text{-}uniformly\text{-}continuous})$
then obtain $d{::}\mathit{real}$ **where** *d*: $d{>}0$
and *de*: $\bigwedge x\ x'.\ |x' - x| < d \Longrightarrow x{\in}\{0..1\} \Longrightarrow x'{\in}\{0..1\} \Longrightarrow \mathit{cmod}\ (p\ x' - p\ x) < e/3$
unfolding *uniformly-continuous-on-def dist-norm real-norm-def*
by $(\mathit{metis\ divide\text{-}pos\text{-}pos\ enz\ zero\text{-}less\text{-}numeral})$
then obtain $N{::}\mathit{nat}$ **where** *N*: $N{>}0\ \mathit{inverse}\ N < d$
using *real-arch-inv* $[\mathit{of}\ d]$ **by** *auto*
{ fix *g h*
assume *g*: *valid-path g* **and** *gp*: $\forall t{\in}\{0..1\}.\ \mathit{cmod}\ (g\ t - p\ t) < e\ /\ 3$

and *h*: *valid-path h* **and** *hp*: $\forall\, t{\in}\{0..1\}$. *cmod* (*h t* − *p t*) < *e* / *3*
and *joins*: *linked-paths atends g h*
{ fix *t*::*real*
assume *t*: *0* ≤ *t t* ≤ *1*
then obtain *u* **where** *u*: *u* ∈ *k* **and** *ptu*: *p t* ∈ *ball*(*p u*) (*ee*(*p u*) / *3*)
using ‹*path-image p* ⊆ ⋃ *D*› *D-eq* **by** (*force simp*: *path-image-def*)
then have *ele*: *e* ≤ *ee* (*p u*) **using** *fin-eep*
by (*simp add*: *e-def*)
have *cmod* (*g t* − *p t*) < *e* / *3 cmod* (*h t* − *p t*) < *e* / *3*
using *gp hp t* **by** *auto*
with *ele* **have** *cmod* (*g t* − *p t*) < *ee* (*p u*) / *3*
cmod (*h t* − *p t*) < *ee* (*p u*) / *3*
by *linarith*+
then have *g t* ∈ *ball*(*p u*) (*ee*(*p u*)) *h t* ∈ *ball*(*p u*) (*ee*(*p u*))
using *norm-diff-triangle-ineq* [*of g t p t p t p u*]
norm-diff-triangle-ineq [*of h t p t p t p u*] *ptu eepi u*
by (*force simp*: *dist-norm ball-def norm-minus-commute*)+
then have *g t* ∈ *s h t* ∈ *s* **using** *ee u k*
by (*auto simp*: *path-image-def ball-def*)
}
then have *ghs*: *path-image g* ⊆ *s path-image h* ⊆ *s*
by (*auto simp*: *path-image-def*)
moreover
{ fix *f*
assume *fhols*: *f holomorphic-on s*
then have *fpa*: *f contour-integrable-on g f contour-integrable-on h*
using *g ghs h holomorphic-on-imp-continuous-on os contour-integrable-holomorphic-simple*
by *blast*+
have *contf*: *continuous-on s f*
by (*simp add*: *fhols holomorphic-on-imp-continuous-on*)
{ fix *z*
assume *z*: *z* ∈ *path-image p*
have *f holomorphic-on ball z* (*ee z*)
using *fhols ee z holomorphic-on-subset* **by** *blast*
then have ∃*ff*. (∀ *w* ∈ *ball z* (*ee z*). (*ff has-field-derivative f w*) (*at w*))
using *holomorphic-convex-primitive* [*of ball z* (*ee z*) {} *f*, *simplified*]
by (*metis open-ball at-within-open holomorphic-on-def holomorphic-on-imp-continuous-on mem-ball*)
}
then obtain *ff* **where** *ff*:
⋀*z w*. ⟦*z* ∈ *path-image p*; *w* ∈ *ball z* (*ee z*)⟧ ⟹ (*ff z has-field-derivative f w*) (*at w*)
by *metis*
{ fix *n*
assume *n*: *n* ≤ *N*
then have *contour-integral*(*subpath 0* (*n*/*N*) *h*) *f* − *contour-integral*(*subpath 0* (*n*/*N*) *g*) *f* =
contour-integral(*linepath* (*g*(*n*/*N*)) (*h*(*n*/*N*))) *f* − *contour-integral*(*linepath* (*g 0*) (*h 0*)) *f*

```
      proof (induct n)
        case 0 show ?case by simp
      next
        case (Suc n)
        obtain t where t: t ∈ k and p (n/N) ∈ ball(p t) (ee(p t) / 3)
          using ‹path-image p ⊆ ⋃ D› [THEN subsetD, where c=p (n/N)] D-eq
N Suc.prems
          by (force simp: path-image-def)
        then have ptu: cmod (p t − p (n/N)) < ee (p t) / 3
          by (simp add: dist-norm)
        have e3le: e/3 ≤ ee (p t) / 3  using fin-eep t
          by (simp add: e-def)
        { fix x
          assume x: n/N ≤ x x ≤ (1 + n)/N
          then have nN01: 0 ≤ n/N (1 + n)/N ≤ 1
            using Suc.prems by auto
          then have x01: 0 ≤ x x ≤ 1
            using x by linarith+
          have cmod (p t − p x)  < ee (p t) / 3 + e/3
            apply (rule norm-diff-triangle-less [OF ptu de])
            using x N x01 Suc.prems
            apply (auto simp: field-simps)
            done
          then have ptx: cmod (p t − p x) < 2∗ee (p t)/3
            using e3le eepi [OF t] by simp
          have cmod (p t − g x) < 2∗ee (p t)/3 + e/3
            apply (rule norm-diff-triangle-less [OF ptx])
            using gp x01 by (simp add: norm-minus-commute)
          also have ... ≤ ee (p t)
            using e3le eepi [OF t] by simp
          finally have gg: cmod (p t − g x) < ee (p t) .
          have cmod (p t − h x) < 2∗ee (p t)/3 + e/3
            apply (rule norm-diff-triangle-less [OF ptx])
            using hp x01 by (simp add: norm-minus-commute)
          also have ... ≤ ee (p t)
            using e3le eepi [OF t] by simp
          finally have cmod (p t − g x) < ee (p t)
                       cmod (p t − h x) < ee (p t)
            using gg by auto
        } note ptgh-ee = this
        have pi-hgn: path-image (linepath (h (n/N)) (g (n/N))) ⊆ ball (p t) (ee
(p t))
          using ptgh-ee [of n/N] Suc.prems
          by (auto simp: field-simps dist-norm dest: segment-furthest-le [where
y=p t])
        then have gh-ns: closed-segment (g (n/N)) (h (n/N)) ⊆ s
          using ‹N>0› Suc.prems
          apply (simp add: path-image-join field-simps closed-segment-commute)
          apply (erule order-trans)
```

```
          apply (simp add: ee pi t)
          done
        have pi-ghn': path-image (linepath (g ((1 + n) / N)) (h ((1 + n) / N)))
              ⊆ ball (p t) (ee (p t))
          using ptgh-ee [of (1+n)/N] Suc.prems
           by (auto simp: field-simps dist-norm dest: segment-furthest-le [where
y=p t])
         then have gh-n's: closed-segment (g ((1 + n) / N)) (h ((1 + n) / N))
⊆ s
          using ‹N>0› Suc.prems ee pi t
          by (auto simp: Path-Connected.path-image-join field-simps)
        have pi-subset-ball:
            path-image (subpath (n/N) ((1+n) / N) g +++ linepath (g ((1+n)
/ N)) (h ((1+n) / N)) +++
                        subpath ((1+n) / N) (n/N) h +++ linepath (h (n/N)) (g
(n/N)))
              ⊆ ball (p t) (ee (p t))
          apply (intro subset-path-image-join pi-hgn pi-ghn')
          using ‹N>0› Suc.prems
       apply (auto simp: path-image-subpath dist-norm field-simps closed-segment-eq-real-ivl
ptgh-ee)
          done
        have pi0: (f has-contour-integral 0)
                  (subpath (n/ N) ((Suc n)/N) g +++ linepath(g ((Suc n) / N))
(h((Suc n) / N)) +++
                      subpath ((Suc n) / N) (n/N) h +++ linepath(h (n/N)) (g
(n/N)))
          apply (rule Cauchy-theorem-primitive [of ball(p t) (ee(p t)) ff (p t) f])
          apply (metis ff open-ball at-within-open pi t)
          apply (intro valid-path-join)
           using Suc.prems pi-subset-ball apply (simp-all add: valid-path-subpath
g h)
          done
        have fpa1: f contour-integrable-on subpath (real n / real N) (real (Suc n)
/ real N) g
          using Suc.prems by (simp add: contour-integrable-subpath g fpa)
         have fpa2: f contour-integrable-on linepath (g (real (Suc n) / real N)) (h
(real (Suc n) / real N))
          using gh-n's
       by (auto intro!: contour-integrable-continuous-linepath continuous-on-subset
[OF contf])
        have fpa3: f contour-integrable-on linepath (h (real n / real N)) (g (real n
/ real N))
          using gh-ns
       by (auto simp: closed-segment-commute intro!: contour-integrable-continuous-linepath
continuous-on-subset [OF contf])
        have eq0: contour-integral (subpath (n/N) ((Suc n) / real N) g) f +
                  contour-integral (linepath (g ((Suc n) / N)) (h ((Suc n) / N))) f
+
```

```
                contour-integral (subpath ((Suc n) / N) (n/N) h) f +
                contour-integral (linepath (h (n/N)) (g (n/N))) f = 0
          using contour-integral-unique [OF pi0] Suc.prems
          by (simp add: g h fpa valid-path-subpath contour-integrable-subpath
                fpa1 fpa2 fpa3 algebra-simps del: of-nat-Suc)
        have *: ⋀hn he hn' gn gd gn' hgn ghn gh0 ghn'.
                ⟦hn − gn = ghn − gh0;
                 gd + ghn' + he + hgn = (0::complex);
                 hn − he = hn'; gn + gd = gn'; hgn = −ghn⟧ ⟹ hn' − gn' =
ghn' − gh0
          by (auto simp: algebra-simps)
        have contour-integral (subpath 0 (n/N) h) f − contour-integral (subpath
((Suc n) / N) (n/N) h) f =
              contour-integral (subpath 0 (n/N) h) f + contour-integral (subpath
(n/N) ((Suc n) / N) h) f
          unfolding reversepath-subpath [symmetric, of ((Suc n) / N)]
            using Suc.prems by (simp add: h fpa contour-integral-reversepath
valid-path-subpath contour-integrable-subpath)
        also have ... = contour-integral (subpath 0 ((Suc n) / N) h) f
          using Suc.prems by (simp add: contour-integral-subpath-combine h fpa)
        finally have pi0-eq:
             contour-integral (subpath 0 (n/N) h) f − contour-integral (subpath
((Suc n) / N) (n/N) h) f =
            contour-integral (subpath 0 ((Suc n) / N) h) f .
        show ?case
          apply (rule * [OF Suc.hyps eq0 pi0-eq])
          using Suc.prems
          apply (simp-all add: g h fpa contour-integral-subpath-combine
            contour-integral-reversepath [symmetric] contour-integrable-continuous-linepath
                continuous-on-subset [OF contf gh-ns])
          done
      qed
      } note ind = this
      have contour-integral h f = contour-integral g f
        using ind [OF order-refl] N joins
       by (simp add: linked-paths-def pathstart-def pathfinish-def split: split-if-asm)
    }
    ultimately
     have path-image g ⊆ s ∧ path-image h ⊆ s ∧ (∀f. f holomorphic-on s ⟶
contour-integral h f = contour-integral g f)
      by metis
  } note * = this
  show ?thesis
    apply (rule-tac x=e/3 in exI)
    apply (rule conjI)
    using enz apply simp
    apply (clarsimp simp only: ball-conj-distrib)
    apply (rule *; assumption)
    done
```

qed

lemma
assumes *open s path p path-image p* $\subseteq$ *s*
shows *contour-integral-nearby-ends*:
$\exists\, d.\ 0 < d \wedge$
$(\forall\, g\ h.\ \textit{valid-path}\ g \wedge \textit{valid-path}\ h \wedge$
$(\forall\, t \in \{0..1\}.\ \textit{norm}(g\ t - p\ t) < d \wedge \textit{norm}(h\ t - p\ t) < d) \wedge$
$\textit{pathstart}\ h = \textit{pathstart}\ g \wedge \textit{pathfinish}\ h = \textit{pathfinish}\ g$
$\longrightarrow \textit{path-image}\ g \subseteq s \wedge$
$\textit{path-image}\ h \subseteq s \wedge$
$(\forall f.\ f\ \textit{holomorphic-on}\ s$
$\longrightarrow \textit{contour-integral}\ h\ f = \textit{contour-integral}\ g\ f))$
and *contour-integral-nearby-loops*:
$\exists\, d.\ 0 < d \wedge$
$(\forall\, g\ h.\ \textit{valid-path}\ g \wedge \textit{valid-path}\ h \wedge$
$(\forall\, t \in \{0..1\}.\ \textit{norm}(g\ t - p\ t) < d \wedge \textit{norm}(h\ t - p\ t) < d) \wedge$
$\textit{pathfinish}\ g = \textit{pathstart}\ g \wedge \textit{pathfinish}\ h = \textit{pathstart}\ h$
$\longrightarrow \textit{path-image}\ g \subseteq s \wedge$
$\textit{path-image}\ h \subseteq s \wedge$
$(\forall f.\ f\ \textit{holomorphic-on}\ s$
$\longrightarrow \textit{contour-integral}\ h\ f = \textit{contour-integral}\ g\ f))$
using *contour-integral-nearby* [*OF assms*, **where** *atends=True*]
using *contour-integral-nearby* [*OF assms*, **where** *atends=False*]
unfolding *linked-paths-def* **by** *simp-all*

corollary *differentiable-polynomial-function*:
fixes *p* :: *real* $\Rightarrow$ *'a*::*euclidean-space*
shows *polynomial-function p* $\Longrightarrow$ *p differentiable-on s*
by (*meson has-vector-derivative-polynomial-function differentiable-at-imp-differentiable-on differentiable-def has-vector-derivative-def*)

lemma *C1-differentiable-polynomial-function*:
fixes *p* :: *real* $\Rightarrow$ *'a*::*euclidean-space*
shows *polynomial-function p* $\Longrightarrow$ *p C1-differentiable-on s*
by (*metis continuous-on-polymonial-function C1-differentiable-on-def has-vector-derivative-polynomial-functi*

lemma *valid-path-polynomial-function*:
fixes *p* :: *real* $\Rightarrow$ *'a*::*euclidean-space*
shows *polynomial-function p* $\Longrightarrow$ *valid-path p*
by (*force simp*: *valid-path-def piecewise-C1-differentiable-on-def continuous-on-polymonial-function C1-differentiable-polynomial-function*)

lemma *valid-path-subpath-trivial* [*simp*]:
fixes *g* :: *real* $\Rightarrow$ *'a*::*euclidean-space*
shows *z* $\neq$ *g x* $\Longrightarrow$ *valid-path* (*subpath x x g*)
by (*simp add*: *subpath-def valid-path-polynomial-function*)

lemma *contour-integral-bound-exists*:
assumes *s*: *open s*
  and *g*: *valid-path g*
  and *pag*: *path-image g* $\subseteq$ *s*
 shows $\exists L.\ 0 < L \wedge$
    $(\forall f\ B.\ f$ *holomorphic-on s* $\wedge\ (\forall z \in s.\ norm(f\ z) \leq B)$
      $\longrightarrow$ *norm*(*contour-integral g f*) $\leq L{*}B)$
proof −
have *path g* **using** *g*
 by (*simp add*: *valid-path-imp-path*)
then obtain *d*::*real* **and** *p*
 where *d*: *0* < *d*
  and *p*: *polynomial-function p path-image p* $\subseteq$ *s*
  and *pi*: $\bigwedge f.$ *f holomorphic-on s* $\Longrightarrow$ *contour-integral g f* = *contour-integral p f*
 using *contour-integral-nearby-ends* [*OF s* ‹*path g*› *pag*]
 apply *clarify*
 apply (*drule-tac x*=*g* **in** *spec*)
 apply (*simp only*: *assms*)
apply (*force simp*: *valid-path-polynomial-function dest*: *path-approx-polynomial-function*)
 done
then obtain *p'* **where** *p'*: *polynomial-function p'*
      $\bigwedge x.$ (*p has-vector-derivative* (*p' x*)) (*at x*)
 using *has-vector-derivative-polynomial-function* **by** *force*
then have *bounded*(*p'* ‘ {*0..1*})
 using *continuous-on-polymonial-function*
 by (*force simp*: *intro*!: *compact-imp-bounded compact-continuous-image*)
then obtain *L* **where** *L*: *L>0* **and** *nop'*: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow$ *norm* (*p' x*) $\leq L$
 by (*force simp*: *bounded-pos*)
{ fix *f B*
 assume *f*: *f holomorphic-on s*
   and *B*: $\bigwedge z.\ z{\in}s \Longrightarrow$ *cmod* (*f z*) $\leq B$
 then have *f contour-integrable-on p* $\wedge$ *valid-path p*
   using *p s*
  by (*blast intro*: *valid-path-polynomial-function contour-integrable-holomorphic-simple holomorphic-on-imp-continuous-on*)
 moreover have $\bigwedge x.\ x \in \{0..1\} \Longrightarrow$ *cmod* (*vector-derivative p* (*at x*)) $*$ *cmod* (*f* (*p x*)) $\leq L * B$
   apply (*rule mult-mono*)
   apply (*subst Derivative.vector-derivative-at*; *force intro*: *p' nop'*)
   using *L B p*
   apply (*auto simp*: *path-image-def image-subset-iff*)
   done
 ultimately have *cmod* (*contour-integral g f*) $\leq L * B$
   apply (*simp add*: *pi* [*OF f*])
   apply (*simp add*: *contour-integral-integral*)
   apply (*rule order-trans* [*OF integral-norm-bound-integral*])
  apply (*auto simp*: *mult.commute integral-norm-bound-integral contour-integrable-on* [*symmetric*] *norm-mult*)
   done

} then
show *?thesis*
by (*force simp*: *L contour-integral-integral*)
qed

52.22 Constancy of a function from a connected set into a finite, disconnected or discrete set

Still missing: versions for a set that is smaller than R, or countable.

lemma *continuous-disconnected-range-constant*:
assumes *s*: *connected s*
and *conf*: *continuous-on s f*
and *fim*: *f ‘ s ⊆ t*
and *cct*: ⋀*y*. *y ∈ t ⟹ connected-component-set t y = {y}*
shows ∃ *a*. ∀ *x ∈ s*. *f x = a*
proof (*cases s = {}*)
case *True* **then show** *?thesis* **by** *force*
next
case *False*
{ fix *x* **assume** *x ∈ s*
then have *f ‘ s ⊆ {f x}*
by (*metis connected-continuous-image conf connected-component-maximal fim image-subset-iff rev-image-eqI s cct*)
}
with *False* **show** *?thesis*
by *blast*
qed

lemma *discrete-subset-disconnected*:
fixes *s* :: *'a*::*topological-space set*
fixes *t* :: *'b*::*real-normed-vector set*
assumes *conf*: *continuous-on s f*
and *no*: ⋀*x*. *x ∈ s ⟹* ∃ *e>0*. ∀ *y*. *y ∈ s ∧ f y ≠ f x ⟶ e ≤ norm (f y − f x)*
shows *f ‘ s ⊆ {y*. *connected-component-set (f ‘ s) y = {y}}*
proof −
{ fix *x* **assume** *x*: *x ∈ s*
then obtain *e* **where** *e>0* **and** *ele*: ⋀*y*. ⟦*y ∈ s*; *f y ≠ f x*⟧ *⟹ e ≤ norm (f y − f x)*
using *conf no* [*OF x*] **by** *auto*
then have *e2*: *0 ≤ e / 2*
by *simp*
have *f y = f x* **if** *y ∈ s* **and** *ccs*: *f y ∈ connected-component-set (f ‘ s) (f x)* **for** *y*
apply (*rule ccontr*)
using *connected-closed* [*of connected-component-set (f ‘ s) (f x)*] ‹*e>0*›
apply (*simp add*: *del*: *ex-simps*)
apply (*drule spec* [**where** *x=cball (f x) (e / 2)*])
apply (*drule spec* [**where** *x=− ball(f x) e*])

apply (*auto simp*: *dist-norm open-closed* [*symmetric*] *simp del*: *le-divide-eq-numeral1 dest*!: *connected-component-in*)
apply (*metis diff-self e2 ele norm-minus-commute norm-zero not-less*)
using *centre-in-cball connected-component-refl-eq e2 x* **apply** *blast*
using *ccs*
apply (*force simp*: *cball-def dist-norm norm-minus-commute dest*: *ele* [*OF* ‹*y* ∈ *s*›])
done
moreover have *connected-component-set* (*f* ‘ *s*) (*f x*) ⊆ *f* ‘ *s*
by (*auto simp*: *connected-component-in*)
ultimately have *connected-component-set* (*f* ‘ *s*) (*f x*) = {*f x*}
by (*auto simp*: *x*)
}
with *assms* **show** *?thesis*
by *blast*
qed

lemma *finite-implies-discrete*:
fixes *s* :: *'a*::*topological-space set*
assumes *finite* (*f* ‘ *s*)
shows (∀ *x* ∈ *s*. ∃ *e*>*0*. ∀ *y*. *y* ∈ *s* ∧ *f y* ≠ *f x* ⟶ *e* ≤ *norm* (*f y* − *f x*))
proof −
have ∃ *e*>*0*. ∀ *y*. *y* ∈ *s* ∧ *f y* ≠ *f x* ⟶ *e* ≤ *norm* (*f y* − *f x*) **if** *x* ∈ *s* **for** *x*
proof (*cases f* ‘ *s* − {*f x*} = {})
case *True*
with *zero-less-numeral* **show** *?thesis*
by (*fastforce simp add*: *Set.image-subset-iff cong*: *conj-cong*)
next
case *False*
then obtain *z* **where** *z*: *z* ∈ *s f z* ≠ *f x*
by *blast*
have *finn*: *finite* {*norm* (*z* − *f x*) |*z*. *z* ∈ *f* ‘ *s* − {*f x*}}
using *assms* **by** *simp*
then have ∗: *0* < *Inf*{*norm*(*z* − *f x*) | *z*. *z* ∈ *f* ‘ *s* − {*f x*}}
apply (*rule finite-imp-less-Inf*)
using *z* **apply** *force*+
done
show *?thesis*
by (*force intro*!: ∗ *cInf-le-finite* [*OF finn*])
qed
with *assms* **show** *?thesis*
by *blast*
qed

This proof requires the existence of two separate values of the range type.

lemma *finite-range-constant-imp-connected*:
assumes ⋀*f*::*'a*::*topological-space* ⇒ *'b*::*real-normed-algebra-1*.
⟦*continuous-on s f*; *finite*(*f* ‘ *s*)⟧ ⟹ ∃ *a*. ∀ *x* ∈ *s*. *f x* = *a*
shows *connected s*

proof −
{ fix *t u*
assume *clt*: *closedin* (*subtopology euclidean s*) *t*
and *clu*: *closedin* (*subtopology euclidean s*) *u*
and *tue*: $t \cap u = \{\}$ **and** *tus*: $t \cup u = s$
have *conif*: *continuous-on s* (λx. *if* $x \in t$ *then 0 else 1*)
apply (*subst tus* [*symmetric*])
apply (*rule continuous-on-cases-local*)
using *clt clu tue*
apply (*auto simp*: *tus continuous-on-const*)
done
have *fi*: *finite* ((λx. *if* $x \in t$ *then 0 else 1*) ‘ *s*)
by (*rule finite-subset* [*of* - {*0*,*1*}]) *auto*
have $t = \{\} \vee u = \{\}$
using *assms* [*OF conif fi*] *tus* [*symmetric*]
by (*auto simp*: *Ball-def*) (*metis IntI empty-iff one-neq-zero tue*)
}
then show *?thesis*
by (*simp add*: *connected-closed-in-eq*)
qed

lemma *continuous-disconnected-range-constant-eq*:
(*connected s* ⟷
(∀ *f*::*'a*::*topological-space* ⇒ *'b*::*real-normed-algebra-1*.
∀ *t*. *continuous-on s f* ∧ *f* ‘ *s* ⊆ *t* ∧ (∀ *y* ∈ *t*. *connected-component-set t y* = {*y*})
⟶ (∃ *a*::*'b*. ∀ *x* ∈ *s*. *f x* = *a*))) (**is** *?thesis1*)
and *continuous-discrete-range-constant-eq*:
(*connected s* ⟷
(∀ *f*::*'a*::*topological-space* ⇒ *'b*::*real-normed-algebra-1*.
continuous-on s f ∧
(∀ *x* ∈ *s*. ∃ *e*. *0* < *e* ∧ (∀ *y*. *y* ∈ *s* ∧ (*f y* ≠ *f x*) ⟶ *e* ≤ *norm*(*f y* − *f x*)))
⟶ (∃ *a*::*'b*. ∀ *x* ∈ *s*. *f x* = *a*))) (**is** *?thesis2*)
and *continuous-finite-range-constant-eq*:
(*connected s* ⟷
(∀ *f*::*'a*::*topological-space* ⇒ *'b*::*real-normed-algebra-1*.
continuous-on s f ∧ *finite* (*f* ‘ *s*)
⟶ (∃ *a*::*'b*. ∀ *x* ∈ *s*. *f x* = *a*))) (**is** *?thesis3*)
proof −
have ∗: ⋀*s t u v*. ⟦*s* ⟹ *t*; *t* ⟹ *u*; *u* ⟹ *v*; *v* ⟹ *s*⟧
⟹ (*s* ⟷ *t*) ∧ (*s* ⟷ *u*) ∧ (*s* ⟷ *v*)
by *blast*
have *?thesis1* ∧ *?thesis2* ∧ *?thesis3*
apply (*rule* ∗)
using *continuous-disconnected-range-constant* **apply** *metis*
apply *clarify*
apply (*frule discrete-subset-disconnected*; *blast*)
apply (*blast dest*: *finite-implies-discrete*)
apply (*blast intro*!: *finite-range-constant-imp-connected*)

done
then show *?thesis1 ?thesis2 ?thesis3*
by *blast+*
qed

lemma *continuous-discrete-range-constant*:
fixes f :: *'a::topological-space* ⇒ *'b::real-normed-algebra-1*
assumes s: *connected s*
and *continuous-on s f*
and $\bigwedge x.\ x \in s \Longrightarrow \exists e{>}0.\ \forall y.\ y \in s \wedge f\ y \neq f\ x \longrightarrow e \leq norm\ (f\ y - f\ x)$
shows $\exists a.\ \forall x \in s.\ f\ x = a$
using *continuous-discrete-range-constant-eq* [*THEN iffD1*, *OF s*] *assms*
by *blast*

lemma *continuous-finite-range-constant*:
fixes f :: *'a::topological-space* ⇒ *'b::real-normed-algebra-1*
assumes *connected s*
and *continuous-on s f*
and *finite* (*f ' s*)
shows $\exists a.\ \forall x \in s.\ f\ x = a$
using *assms continuous-finite-range-constant-eq*
by *blast*

We can treat even non-rectifiable paths as having a "length" for bounds on analytic functions in open sets.

52.23 Winding Numbers

definition *winding-number*:: [*real* ⇒ *complex*, *complex*] ⇒ *complex* **where**
winding-number $\gamma\ z \equiv$
@$n.\ \forall e > 0.\ \exists p.$ *valid-path* $p \wedge z \notin$ *path-image* $p \wedge$
pathstart p = *pathstart* $\gamma \wedge$
pathfinish p = *pathfinish* $\gamma \wedge$
$(\forall t \in \{0..1\}.\ norm(\gamma\ t - p\ t) < e) \wedge$
contour-integral $p\ (\lambda w.\ 1/(w - z)) = 2 * pi * ii * n$

lemma *winding-number*:
assumes *path* γ $z \notin$ *path-image* γ $0 < e$
shows $\exists p.$ *valid-path* $p \wedge z \notin$ *path-image* $p \wedge$
pathstart p = *pathstart* $\gamma \wedge$
pathfinish p = *pathfinish* $\gamma \wedge$
$(\forall t \in \{0..1\}.\ norm\ (\gamma\ t - p\ t) < e) \wedge$
contour-integral $p\ (\lambda w.\ 1/(w - z)) = 2 * pi * ii *$ *winding-number* γ
z
proof −
have *path-image* $\gamma \subseteq UNIV - \{z\}$
using *assms* **by** *blast*
then obtain d
where d: $d{>}0$

```
    and pi-eq: ⋀h1 h2. valid-path h1 ∧ valid-path h2 ∧
               (∀t∈{0..1}. cmod (h1 t − γ t) < d ∧ cmod (h2 t − γ t) < d) ∧
               pathstart h2 = pathstart h1 ∧ pathfinish h2 = pathfinish h1 ⟶
                path-image h1 ⊆ UNIV − {z} ∧ path-image h2 ⊆ UNIV − {z}
∧
                (∀f. f holomorphic-on UNIV − {z} ⟶ contour-integral h2 f =
contour-integral h1 f)
   using contour-integral-nearby-ends [of UNIV − {z} γ] assms by (auto simp:
open-delete)
 then obtain h where h: polynomial-function h ∧ pathstart h = pathstart γ ∧
pathfinish h = pathfinish γ ∧
                     (∀t ∈ {0..1}. norm(h t − γ t) < d/2)
   using path-approx-polynomial-function [OF ‹path γ›, of d/2] d by auto
 def nn ≡ 1/(2* pi*ii) * contour-integral h (λw. 1/(w − z))
 have ∃n. ∀e > 0. ∃p. valid-path p ∧ z ∉ path-image p ∧
                   pathstart p = pathstart γ ∧  pathfinish p = pathfinish γ ∧
                   (∀t ∈ {0..1}. norm(γ t − p t) < e) ∧
                   contour-integral p (λw. 1/(w − z)) = 2 * pi * ii * n
                 (is ∃n. ∀e > 0. ?PP e n)
   proof (rule-tac x=nn in exI, clarify)
     fix e::real
     assume e: e>0
     obtain p where p: polynomial-function p ∧
          pathstart p = pathstart γ ∧ pathfinish p = pathfinish γ ∧ (∀t∈{0..1}.
cmod (p t − γ t) < min e (d / 2))
       using path-approx-polynomial-function [OF ‹path γ›, of min e (d/2)] d
‹0<e› by auto
     have (λw. 1 / (w − z)) holomorphic-on UNIV − {z}
       by (auto simp: intro!: holomorphic-intros)
     then show ?PP e nn
       apply (rule-tac x=p in exI)
       using pi-eq [of h p] h p d
         apply (auto simp: valid-path-polynomial-function norm-minus-commute
nn-def)
       done
   qed
 then show ?thesis
   unfolding winding-number-def
   apply (rule someI2-ex)
   apply (blast intro: ‹0<e›)
   done
qed

lemma winding-number-unique:
 assumes γ: path γ z ∉ path-image γ
     and pi:
       ⋀e. e>0 ⟹ ∃p. valid-path p ∧ z ∉ path-image p ∧
                   pathstart p = pathstart γ ∧ pathfinish p = pathfinish γ ∧
                   (∀t ∈ {0..1}. norm (γ t − p t) < e) ∧
```

```
                    contour-integral p (λw. 1/(w − z)) = 2 ∗ pi ∗ ii ∗ n
  shows winding-number γ z = n
proof −
  have path-image γ ⊆ UNIV − {z}
    using assms by blast
  then obtain e
    where e: e>0
      and pi-eq: ⋀h1 h2 f. ⟦valid-path h1; valid-path h2;
                  (∀ t∈{0..1}. cmod (h1 t − γ t) < e ∧ cmod (h2 t − γ t) < e);
                     pathstart h2 = pathstart h1; pathfinish h2 = pathfinish h1; f
holomorphic-on UNIV − {z}⟧ ⟹
                  contour-integral h2 f = contour-integral h1 f
    using contour-integral-nearby-ends [of UNIV − {z} γ] assms  by (auto simp:
open-delete)
  obtain p where p:
     valid-path p ∧ z ∉ path-image p ∧
      pathstart p = pathstart γ ∧ pathfinish p = pathfinish γ ∧
      (∀ t ∈ {0..1}. norm (γ t − p t) < e) ∧
      contour-integral p (λw. 1/(w − z)) = 2 ∗ pi ∗ ii ∗ n
    using pi [OF e] by blast
  obtain q where q:
     valid-path q ∧ z ∉ path-image q ∧
      pathstart q = pathstart γ ∧ pathfinish q = pathfinish γ ∧
      (∀ t∈{0..1}. cmod (γ t − q t) < e) ∧ contour-integral q (λw. 1 / (w − z))
= complex-of-real (2 ∗ pi) ∗ i ∗ winding-number γ z
    using winding-number [OF γ e] by blast
  have 2 ∗ complex-of-real pi ∗ i ∗ n = contour-integral p (λw. 1 / (w − z))
    using p by auto
  also have ... = contour-integral q (λw. 1 / (w − z))
    apply (rule pi-eq)
    using p q
     by (auto simp: valid-path-polynomial-function norm-minus-commute intro!:
holomorphic-intros)
  also have ... = 2 ∗ complex-of-real pi ∗ i ∗ winding-number γ z
    using q by auto
  finally have 2 ∗ complex-of-real pi ∗ i ∗ n = 2 ∗ complex-of-real pi ∗ i ∗
winding-number γ z .
  then show ?thesis
    by simp
qed

lemma winding-number-unique-loop:
  assumes γ: path γ z ∉ path-image γ
      and loop: pathfinish γ = pathstart γ
      and pi:
        ⋀e. e>0 ⟹ ∃p. valid-path p ∧ z ∉ path-image p ∧
                      pathfinish p = pathstart p ∧
                      (∀ t ∈ {0..1}. norm (γ t − p t) < e) ∧
                      contour-integral p (λw. 1/(w − z)) = 2 ∗ pi ∗ ii ∗ n
```

shows *winding-number* $\gamma\ z = n$
proof −
have *path-image* $\gamma \subseteq UNIV - \{z\}$
using *assms* **by** *blast*
then obtain e
where e: $e>0$
and *pi-eq*: $\bigwedge h1\ h2\ f.\ [\![valid\text{-}path\ h1;\ valid\text{-}path\ h2;$
$(\forall\, t{\in}\{0..1\}.\ cmod\ (h1\ t - \gamma\ t) < e \wedge cmod\ (h2\ t - \gamma\ t) < e);$
$pathfinish\ h1 = pathstart\ h1;\ pathfinish\ h2 = pathstart\ h2;\ f$ *holomorphic-on* $UNIV - \{z\}]\!] \Longrightarrow$
$contour\text{-}integral\ h2\ f = contour\text{-}integral\ h1\ f$
using *contour-integral-nearby-loops* [*of* $UNIV - \{z\}\ \gamma$] *assms* **by** (*auto simp*: *open-delete*)
obtain p **where** p:
$valid\text{-}path\ p \wedge z \notin path\text{-}image\ p \wedge$
$pathfinish\ p = pathstart\ p \wedge$
$(\forall\, t \in \{0..1\}.\ norm\ (\gamma\ t - p\ t) < e) \wedge$
$contour\text{-}integral\ p\ (\lambda w.\ 1/(w - z)) = 2 * pi * ii * n$
using *pi* [*OF e*] **by** *blast*
obtain q **where** q:
$valid\text{-}path\ q \wedge z \notin path\text{-}image\ q \wedge$
$pathstart\ q = pathstart\ \gamma \wedge pathfinish\ q = pathfinish\ \gamma \wedge$
$(\forall\, t{\in}\{0..1\}.\ cmod\ (\gamma\ t - q\ t) < e) \wedge contour\text{-}integral\ q\ (\lambda w.\ 1\ /\ (w - z))$
$= complex\text{-}of\text{-}real\ (2 * pi) * \mathrm{i} * winding\text{-}number\ \gamma\ z$
using *winding-number* [*OF* $\gamma\ e$] **by** *blast*
have $2 * complex\text{-}of\text{-}real\ pi * \mathrm{i} * n = contour\text{-}integral\ p\ (\lambda w.\ 1\ /\ (w - z))$
using p **by** *auto*
also have ... = $contour\text{-}integral\ q\ (\lambda w.\ 1\ /\ (w - z))$
apply (*rule pi-eq*)
using $p\ q$ *loop*
by (*auto simp*: *valid-path-polynomial-function norm-minus-commute intro*!: *holomorphic-intros*)
also have ... = $2 * complex\text{-}of\text{-}real\ pi * \mathrm{i} * winding\text{-}number\ \gamma\ z$
using q **by** *auto*
finally have $2 * complex\text{-}of\text{-}real\ pi * \mathrm{i} * n = 2 * complex\text{-}of\text{-}real\ pi * \mathrm{i} * winding\text{-}number\ \gamma\ z$ **.**
then show *?thesis*
by *simp*
qed

lemma *winding-number-valid-path*:
assumes $valid\text{-}path\ \gamma\ z \notin path\text{-}image\ \gamma$
shows $winding\text{-}number\ \gamma\ z = 1/(2{*}pi{*}ii) * contour\text{-}integral\ \gamma\ (\lambda w.\ 1/(w - z))$
using *assms* **by** (*auto simp*: *valid-path-imp-path intro*!: *winding-number-unique*)

lemma *has-contour-integral-winding-number*:
assumes γ: $valid\text{-}path\ \gamma\ z \notin path\text{-}image\ \gamma$
shows $((\lambda w.\ 1/(w - z))\ has\text{-}contour\text{-}integral\ (2{*}pi{*}ii{*}winding\text{-}number\ \gamma\ z))$

γ
by (*simp add*: *winding-number-valid-path has-contour-integral-integral contour-integrable-inversediff assms*)

lemma *winding-number-trivial* [*simp*]: $z \neq a \Longrightarrow$ *winding-number*(*linepath a a*) *z* = *0*
by (*simp add*: *winding-number-valid-path*)

lemma *winding-number-subpath-trivial* [*simp*]: $z \neq g\ x \Longrightarrow$ *winding-number* (*subpath x x g*) *z* = *0*
by (*simp add*: *path-image-subpath winding-number-valid-path*)

lemma *winding-number-join*:
assumes *g1*: *path g1 z* $\notin$ *path-image g1*
and *g2*: *path g2 z* $\notin$ *path-image g2*
and *pathfinish g1* = *pathstart g2*
shows *winding-number*(*g1 +++ g2*) *z* = *winding-number g1 z* + *winding-number g2 z*
apply (*rule winding-number-unique*)
using *assms* **apply** (*simp-all add*: *not-in-path-image-join*)
apply (*frule winding-number* [*OF g2*])
apply (*frule winding-number* [*OF g1*], *clarify*)
apply (*rename-tac p2 p1*)
apply (*rule-tac x=p1+++p2* **in** *exI*)
apply (*simp add*: *not-in-path-image-join contour-integrable-inversediff algebra-simps*)
apply (*auto simp*: *joinpaths-def*)
done

lemma *winding-number-reversepath*:
assumes *path* γ *z* $\notin$ *path-image* γ
shows *winding-number*(*reversepath* γ) *z* = $-$ (*winding-number* γ *z*)
apply (*rule winding-number-unique*)
using *assms*
apply *simp-all*
apply (*frule winding-number* [*OF assms*], *clarify*)
apply (*rule-tac x=reversepath p* **in** *exI*)
apply (*simp add*: *contour-integral-reversepath contour-integrable-inversediff valid-path-imp-reverse*)
apply (*auto simp*: *reversepath-def*)
done

lemma *winding-number-shiftpath*:
assumes γ: *path* γ *z* $\notin$ *path-image* γ
and *pathfinish* γ = *pathstart* γ *a* $\in \{0..1\}$
shows *winding-number*(*shiftpath a* γ) *z* = *winding-number* γ *z*
apply (*rule winding-number-unique-loop*)
using *assms*
apply (*simp-all add*: *path-shiftpath path-image-shiftpath pathfinish-shiftpath pathstart-shiftpath*)
apply (*frule winding-number* [*OF* γ], *clarify*)
apply (*rule-tac x=shiftpath a p* **in** *exI*)

apply (*simp add*: *contour-integral-shiftpath path-image-shiftpath pathfinish-shiftpath pathstart-shiftpath valid-path-shiftpath*)
apply (*auto simp*: *shiftpath-def*)
done

lemma *winding-number-split-linepath*:
assumes $c \in$ *closed-segment a b* $z \notin$ *closed-segment a b*
shows *winding-number*(*linepath a b*) $z =$ *winding-number*(*linepath a c*) $z +$ *winding-number*(*linepath c b*) z
proof −
have $z \notin$ *closed-segment a c* $z \notin$ *closed-segment c b*
using *assms* **apply** (*meson convex-contains-segment convex-segment ends-in-segment*(*1*) *subsetCE*)
using *assms* **by** (*meson convex-contains-segment convex-segment ends-in-segment*(*2*) *subsetCE*)
then show *?thesis*
using *assms*
by (*simp add*: *winding-number-valid-path contour-integral-split-linepath* [*symmetric*] *continuous-on-inversediff field-simps*)
qed

lemma *winding-number-cong*:
$(\bigwedge t.\ [\![0 \le t;\ t \le 1]\!] \Longrightarrow p\ t = q\ t) \Longrightarrow$ *winding-number p z = winding-number q z*
by (*simp add*: *winding-number-def pathstart-def pathfinish-def*)

lemma *winding-number-offset*: *winding-number p z = winding-number* ($\lambda w.\ p\ w - z$) *0*
apply (*simp add*: *winding-number-def contour-integral-integral path-image-def valid-path-def pathstart-def pathfinish-def*)
apply (*intro ext arg-cong* [**where** *f = Eps*] *arg-cong* [**where** *f = All*] *imp-cong refl*, *safe*)
apply (*rename-tac g*)
apply (*rule-tac x*=$\lambda t.\ g\ t - z$ **in** *exI*)
apply (*force simp*: *vector-derivative-def has-vector-derivative-diff-const piecewise-C1-differentiable-diff C1-differentiable-imp-piecewise*)
apply (*rename-tac g*)
apply (*rule-tac x*=$\lambda t.\ g\ t + z$ **in** *exI*)
apply (*simp add*: *piecewise-C1-differentiable-add vector-derivative-def has-vector-derivative-add-const C1-differentiable-imp-piecewise*)
apply (*force simp*: *algebra-simps*)
done

lemma *winding-number-join-pos-combined*:
$[\![$*valid-path* $\gamma 1$; $z \notin$ *path-image* $\gamma 1$; $0 <$ *Re*(*winding-number* $\gamma 1\ z$);
valid-path $\gamma 2$; $z \notin$ *path-image* $\gamma 2$; $0 <$ *Re*(*winding-number* $\gamma 2\ z$); *pathfinish* $\gamma 1$ = *pathstart* $\gamma 2$$]\!]$
$\Longrightarrow$ *valid-path*($\gamma 1$ +++ $\gamma 2$) $\wedge$ $z \notin$ *path-image*($\gamma 1$ +++ $\gamma 2$) $\wedge$ $0 <$ *Re*(*winding-number*($\gamma 1$

+++ $\gamma 2$) z)
by (*simp add*: *valid-path-join path-image-join winding-number-join valid-path-imp-path*)

lemma *Re-winding-number*:
⟦*valid-path* γ; $z \notin$ *path-image* γ⟧
$\Longrightarrow$ *Re*(*winding-number* γ z) = *Im*(*contour-integral* γ ($\lambda w.$ $1/(w - z)$)) / ($2*pi$)
by (*simp add*: *winding-number-valid-path field-simps Re-divide power2-eq-square*)

lemma *winding-number-pos-le*:
assumes γ: *valid-path* γ $z \notin$ *path-image* γ
and *ge*: $\bigwedge x.$ ⟦$0 < x$; $x < 1$⟧ $\Longrightarrow$ $0 \le$ *Im* (*vector-derivative* γ (*at* x) $*$ *cnj*(γ $x - z$))
shows $0 \le$ *Re*(*winding-number* γ z)
proof −
have $*$: $0 \le$ *Im* (*vector-derivative* γ (*at* x) / (γ $x - z$)) **if** x: $0 < x$ $x < 1$ **for** x
using *ge* **by** (*simp add*: *Complex.Im-divide algebra-simps* x)
show *?thesis*
apply (*simp add*: *Re-winding-number* [*OF* γ] *field-simps*)
apply (*rule has-integral-component-nonneg*
[*of ii* $\lambda x.$ *if* $x \in \{0<..<1\}$
then $1/(\gamma$ $x - z) * vector\text{-}derivative$ γ (*at* x) *else* 0, *simplified*])
prefer 3 **apply** (*force simp*: $*$)
apply (*simp add*: *Basis-complex-def*)
apply (*rule has-integral-spike-interior* [*of* 0 1 - $\lambda x.$ $1/(\gamma$ $x - z) * vector\text{-}derivative$ γ (*at* x)])
apply *simp*
apply (*simp only*: *box-real*)
apply (*subst has-contour-integral* [*symmetric*])
using γ
apply (*simp add*: *contour-integrable-inversediff has-contour-integral-integral*)
done
qed

lemma *winding-number-pos-lt-lemma*:
assumes γ: *valid-path* γ $z \notin$ *path-image* γ
and e: $0 < e$
and *ge*: $\bigwedge x.$ ⟦$0 < x$; $x < 1$⟧ $\Longrightarrow$ $e \le$ *Im* (*vector-derivative* γ (*at* x) / (γ $x - z$))
shows $0 <$ *Re*(*winding-number* γ z)
proof −
have $e \le$ *Im* (*contour-integral* γ ($\lambda w.$ 1 / ($w - z$)))
apply (*rule has-integral-component-le*
[*of ii* $\lambda x.$ $ii*e$ $ii*e$ $\{0..1\}$
$\lambda x.$ *if* $x \in \{0<..<1\}$ *then* $1/(\gamma$ $x - z) * vector\text{-}derivative$ γ (*at* x) *else* $ii*e$

contour-integral γ ($\lambda w.\ 1/(w - z)$), *simplified*])
using *e*
apply (*simp-all add*: *Basis-complex-def*)
using *has-integral-const-real* [*of - 0 1*] **apply** *force*
apply (*rule has-integral-spike-interior* [*of 0 1 -* $\lambda x.\ 1/(\gamma\ x - z) *$ *vector-derivative* γ (*at x*), *simplified box-real*])
apply *simp*
apply (*subst has-contour-integral* [*symmetric*])
using γ
apply (*simp-all add*: *contour-integrable-inversediff has-contour-integral-integral ge*)
done
with *e* **show** *?thesis*
by (*simp add*: *Re-winding-number* [*OF* γ] *field-simps*)
qed

lemma *winding-number-pos-lt*:
assumes γ: *valid-path* γ *z* $\notin$ *path-image* γ
and *e*: *0* < *e*
and *ge*: $\bigwedge x.\ [\![0 < x; x < 1]\!] \Longrightarrow e \leq$ *Im* (*vector-derivative* γ (*at x*) $*$ *cnj*(γ *x* − *z*))
shows *0* < *Re* (*winding-number* γ *z*)
proof −
have *bm*: *bounded* ((λ*w*. *w* − *z*) ‘ (*path-image* γ))
using *bounded-translation* [*of - −z*] γ **by** (*simp add*: *bounded-valid-path-image*)
then obtain *B* **where** *B*: *B* > *0* **and** *Bno*: $\bigwedge x.\ x \in$ (λ*w*. *w* − *z*) ‘ (*path-image* γ) $\Longrightarrow$ *norm x* ≤ *B*
using *bounded-pos* [*THEN iffD1*, *OF bm*] **by** *blast*
{ fix *x*::*real* **assume** *x*: *0* < *x x* < *1*
then have *B2*: *cmod* (γ *x* − *z*) ^ *2* ≤ *B* ^ *2* **using** *Bno* [*of* γ *x* − *z*]
by (*simp add*: *path-image-def power2-eq-square mult-mono'*)
with *x* **have** γ *x* ≠ *z* **using** γ
using *path-image-def* **by** *fastforce*
then have *e* / $B^2 \leq$ *Im* (*vector-derivative* γ (*at x*) $*$ *cnj* (γ *x* − *z*)) / (*cmod* (γ *x* − *z*))^2
using *B ge* [*OF x*] *B2 e*
apply (*rule-tac y*=*e* / (*cmod* (γ *x* − *z*))^2 **in** *order-trans*)
apply (*auto simp*: *divide-left-mono divide-right-mono*)
done
then have *e* / $B^2 \leq$ *Im* (*vector-derivative* γ (*at x*) / (γ *x* − *z*))
by (*simp add*: *Im-divide-Reals complex-div-cnj* [*of - γ x − z* **for** *x*] *del*: *complex-cnj-diff times-complex.sel*)
} note $*$ = *this*
show *?thesis*
using *e B* **by** (*simp add*: $*$ *winding-number-pos-lt-lemma* [*OF* γ, *of e*/*B* ^ *2*])
qed

52.24 The winding number is an integer

Proof from the book Complex Analysis by Lars V. Ahlfors, Chapter 4, section 2.1, Also on page 134 of Serge Lang's book with the name title, etc.

lemma *exp-fg*:
fixes *z*::*complex*
assumes *g*: (*g has-vector-derivative g'*) (*at x within s*)
and *f*: (*f has-vector-derivative* (*g' / (g x − z)*)) (*at x within s*)
and *z*: *g x* ≠ *z*
shows ((λ*x*. *exp*(−*f x*) ∗ (*g x − z*)) *has-vector-derivative 0*) (*at x within s*)
proof −
have ∗: (*exp o* (λ*x*. (− *f x*)) *has-vector-derivative* − (*g' / (g x − z)*) ∗ *exp* (− *f x*)) (*at x within s*)
using *assms* **unfolding** *has-vector-derivative-def scaleR-conv-of-real*
by (*auto intro*!: *derivative-eq-intros*)
show *?thesis*
apply (*rule has-vector-derivative-eq-rhs*)
apply (*rule bounded-bilinear.has-vector-derivative* [*OF bounded-bilinear-mult*])
using *z*
apply (*auto simp*: *intro*!: *derivative-eq-intros* ∗ [*unfolded o-def*] *g*)
done
qed

lemma *winding-number-exp-integral*:
fixes *z*::*complex*
assumes γ: γ *piecewise-C1-differentiable-on* {*a..b*}
and *ab*: *a* ≤ *b*
and *z*: *z* ∉ γ ‘ {*a..b*}
shows (λ*x*. *vector-derivative* γ (*at x*) / (γ *x* − *z*)) *integrable-on* {*a..b*}
(**is** *?thesis1*)
exp (− (*integral* {*a..b*} (λ*x*. *vector-derivative* γ (*at x*) / (γ *x* − *z*)))) ∗ (γ *b* − *z*) = γ *a* − *z*
(**is** *?thesis2*)
proof −
let *?D*γ = λ*x*. *vector-derivative* γ (*at x*)
have [*simp*]: ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ γ *x* ≠ *z*
using *z* **by** *force*
have *cong*: *continuous-on* {*a..b*} γ
using γ **by** (*simp add*: *piecewise-C1-differentiable-on-def*)
obtain *k* **where** *fink*: *finite k* **and** *g-C1-diff*: γ *C1-differentiable-on* ({*a..b*} − *k*)
using γ **by** (*force simp*: *piecewise-C1-differentiable-on-def*)
have *o*: *open* ({*a*<..<*b*} − *k*)
using ‹*finite k*› **by** (*simp add*: *finite-imp-closed open-Diff*)
moreover have {*a*<..<*b*} − *k* ⊆ {*a..b*} − *k*
by *force*
ultimately have *g-diff-at*: ⋀*x*. ⟦*x* ∉ *k*; *x* ∈ {*a*<..<*b*}⟧ ⟹ γ *differentiable at x*
by (*metis Diff-iff differentiable-on-subset C1-diff-imp-diff* [*OF g-C1-diff*] *differentiable-on-def differentiable-within-open*)

{ fix w
assume $w \neq z$
have *continuous-on* (*ball* w (*cmod* $(w - z)$)) $(\lambda w.\ 1\ /\ (w - z))$
by (*auto simp*: *dist-norm intro*!: *continuous-intros*)
moreover have $\bigwedge x.\ cmod\ (w - x) < cmod\ (w - z) \Longrightarrow \exists f'.\ ((\lambda w.\ 1\ /\ (w - z))\ has\text{-}field\text{-}derivative\ f')\ (at\ x)$
by (*auto simp*: *intro*!: *derivative-eq-intros*)
ultimately have $\exists h.\ \forall y.\ norm(y - w) < norm(w - z) \longrightarrow (h\ has\text{-}field\text{-}derivative\ 1/(y - z))\ (at\ y)$
using *holomorphic-convex-primitive* [*of ball* w $(norm(w - z))$ $\{\}$ $\lambda w.\ 1/(w - z)$]
by (*simp add*: *complex-differentiable-def Ball-def dist-norm at-within-open-NO-MATCH norm-minus-commute*)
}
then obtain h **where** h: $\bigwedge w\ y.\ w \neq z \Longrightarrow norm(y - w) < norm(w - z) \Longrightarrow (h\ w\ has\text{-}field\text{-}derivative\ 1/(y - z))\ (at\ y)$
by *meson*
have *exy*: $\exists y.\ ((\lambda x.\ inverse\ (\gamma\ x - z) * ?D\gamma\ x)\ has\text{-}integral\ y)\ \{a..b\}$
unfolding *integrable-on-def* [*symmetric*]
apply (*rule contour-integral-local-primitive-any* [*OF piecewise-C1-imp-differentiable* [*OF* γ], *of* $-\{z\}$])
apply (*rename-tac* w)
apply (*rule-tac* x=*norm*$(w - z)$ **in** *exI*)
apply (*simp-all add*: *inverse-eq-divide*)
apply (*metis has-field-derivative-at-within h*)
done
have *vg-int*: $(\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z))\ integrable\text{-}on\ \{a..b\}$
unfolding *box-real* [*symmetric*] *divide-inverse-commute*
by (*auto intro*!: *exy integrable-subinterval simp add*: *integrable-on-def ab*)
with *ab* **show** *?thesis1*
by (*simp add*: *divide-inverse-commute integral-def integrable-on-def*)
{ fix t
assume t: $t \in \{a..b\}$
have *cball*: *continuous-on* (*ball* $(\gamma\ t)$ (*dist* $(\gamma\ t)$ z)) $(\lambda x.\ inverse\ (x - z))$
using z **by** (*auto intro*!: *continuous-intros simp*: *dist-norm*)
have *icd*: $\bigwedge x.\ cmod\ (\gamma\ t - x) < cmod\ (\gamma\ t - z) \Longrightarrow (\lambda w.\ inverse\ (w - z))\ complex\text{-}differentiable\ at\ x$
unfolding *complex-differentiable-def* **by** (*force simp*: *intro*!: *derivative-eq-intros*)
obtain h **where** h: $\bigwedge x.\ cmod\ (\gamma\ t - x) < cmod\ (\gamma\ t - z) \Longrightarrow$
$(h\ has\text{-}field\text{-}derivative\ inverse\ (x - z))\ (at\ x\ within\ \{y.\ cmod\ (\gamma\ t - y) < cmod\ (\gamma\ t - z)\})$
using *holomorphic-convex-primitive* [**where** $f = \lambda w.\ inverse(w - z)$, *OF convex-ball finite.emptyI cball icd*]
by *simp* (*auto simp*: *ball-def dist-norm that*)
{ fix x D
assume x: $x \notin k$ $a < x$ $x < b$
then have $x \in interior\ (\{a..b\} - k)$
using *open-subset-interior* [*OF o*] **by** *fastforce*
then have *con*: *isCont* $(\lambda x.\ ?D\gamma\ x)\ x$

using *g-C1-diff x* **by** (*auto simp*: *C1-differentiable-on-eq intro*: *continuous-on-interior*)
then have *con-vd*: *continuous* (*at x within* {*a..b*}) (λ*x*. *?Dγ x*)
by (*rule continuous-at-imp-continuous-within*)
have *gdx*: *γ differentiable at x*
using *x* **by** (*simp add*: *g-diff-at*)
have ((λ*c*. *exp* (− *integral* {*a..c*} (λ*x*. *vector-derivative γ* (*at x*) / (*γ x* − *z*))) ∗ (*γ c* − *z*)) *has-derivative* (λ*h*. *0*))
(*at x within* {*a..b*})
using *x gdx t*
apply (*clarsimp simp add*: *differentiable-iff-scaleR*)
apply (*rule exp-fg* [*unfolded has-vector-derivative-def*, *simplified*], *blast intro*: *has-derivative-at-within*)
apply (*simp-all add*: *has-vector-derivative-def* [*symmetric*])
apply (*rule has-vector-derivative-eq-rhs* [*OF integral-has-vector-derivative-continuous-at*])
apply (*rule con-vd continuous-intros cong vg-int* | *simp add*: *continuous-at-imp-continuous-within has-vector-derivative-continuous vector-derivative-at*)+
done
} note ∗ = *this*
have *exp* (− (*integral* {*a..t*} (λ*x*. *?Dγ x* / (*γ x* − *z*)))) ∗ (*γ t* − *z*) =*γ a* − *z*
apply (*rule has-derivative-zero-unique-strong-interval* [*of* {*a*,*b*} ∪ *k a b*])
using *t*
apply (*auto intro*!: ∗ *continuous-intros fink cong indefinite-integral-continuous* [*OF vg-int*] *simp add*: *ab*)+
done
}
with *ab* **show** *?thesis2*
by (*simp add*: *divide-inverse-commute integral-def*)
qed

corollary *winding-number-exp-2pi*:
⟦*path p*; *z* ∉ *path-image p*⟧
⟹ *pathfinish p* − *z* = *exp* (*2* ∗ *pi* ∗ *ii* ∗ *winding-number p z*) ∗ (*pathstart p* − *z*)
using *winding-number* [*of p z 1*] **unfolding** *valid-path-def path-image-def pathstart-def pathfinish-def*
by (*force dest*: *winding-number-exp-integral*(*2*) [*of - 0 1 z*] *simp*: *field-simps contour-integral-integral exp-minus*)

52.25 The version with complex integers and equality

lemma *integer-winding-number-eq*:
assumes *γ*: *path γ* **and** *z*: *z* ∉ *path-image γ*
shows *winding-number γ z* ∈ ℤ ⟷ *pathfinish γ* = *pathstart γ*
proof −
have ∗: ⋀*i*::*complex*. ⋀*g0 g1*. ⟦*i* ≠ *0*; *g0* ≠ *z*; (*g1* − *z*) / *i* = *g0* − *z*⟧ ⟹ (*i* = *1* ⟷ *g1* = *g0*)
by (*simp add*: *field-simps*) *algebra*
obtain *p* **where** *p*: *valid-path p z* ∉ *path-image p*
pathstart p = *pathstart γ pathfinish p* = *pathfinish γ*

contour-integral p (λ*w*. *1* / (*w* − *z*)) = *complex-of-real* (*2* ∗ *pi*) ∗
i ∗ *winding-number* γ *z*
using *winding-number* [*OF assms*, *of 1*] **by** *auto*
have [*simp*]: (*winding-number* γ *z* ∈ ℤ) = (*exp* (*contour-integral p* (λ*w*. *1* / (*w*
− *z*))) = *1*)
using *p* **by** (*simp add*: *exp-eq-1 complex-is-Int-iff*)
have *winding-number p z* ∈ ℤ ⟷ *pathfinish p* = *pathstart p*
using *p z*
apply (*simp add*: *winding-number-valid-path valid-path-def path-image-def pathstart-def*
pathfinish-def)
using *winding-number-exp-integral*(*2*) [*of p 0 1 z*]
apply (*simp add*: *field-simps contour-integral-integral exp-minus*)
apply (*rule* ∗)
apply (*auto simp*: *path-image-def field-simps*)
done
then show *?thesis* **using** *p*
by (*auto simp*: *winding-number-valid-path*)
qed

theorem *integer-winding-number*:
⟦*path* γ; *pathfinish* γ = *pathstart* γ; *z* ∉ *path-image* γ⟧ ⟹ *winding-number* γ *z*
∈ ℤ
by (*metis integer-winding-number-eq*)

If the winding number's magnitude is at least one, then the path must contain points in every direction.*) We can thus bound the winding number of a path that doesn't intersect a given ray.

lemma *winding-number-pos-meets*:
fixes *z*::*complex*
assumes γ: *valid-path* γ **and** *z*: *z* ∉ *path-image* γ **and** *1*: *Re* (*winding-number*
γ *z*) ≥ *1*
and *w*: *w* ≠ *z*
shows ∃ *a*::*real*. *0* < *a* ∧ *z* + *a*∗(*w* − *z*) ∈ *path-image* γ
proof −
have [*simp*]: ⋀*x*. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ γ *x* ≠ *z*
using *z* **by** (*auto simp*: *path-image-def*)
have [*simp*]: *z* ∉ γ ‘ {*0*..*1*}
using *path-image-def z* **by** *auto*
have *gpd*: γ *piecewise-C1-differentiable-on* {*0*..*1*}
using γ *valid-path-def* **by** *blast*
def *r* ≡ (*w* − *z*) / (γ *0* − *z*)
have [*simp*]: *r* ≠ *0*
using *w z* **by** (*auto simp*: *r-def*)
have *Arg r* ≤ *2*∗*pi*
by (*simp add*: *Arg less-eq-real-def*)
also have ... ≤ *Im* (*integral* {*0*..*1*} (λ*x*. *vector-derivative* γ (*at x*) / (γ *x* − *z*)))
using *1*
apply (*simp add*: *winding-number-valid-path* [*OF* γ *z*] *Cauchy-Integral-Thm.contour-integral-integral*)
apply (*simp add*: *Complex.Re-divide field-simps power2-eq-square*)

done
finally have *Arg r* ≤ *Im* (*integral* {*0..1*} (λ*x*. *vector-derivative* γ (*at x*) / (γ *x* − *z*))) **.**
then have ∃ *t*. *t* ∈ {*0..1*} ∧ *Im*(*integral* {*0..t*} (λ*x*. *vector-derivative* γ (*at x*)/(γ *x* − *z*))) = *Arg r*
apply (*simp add*:)
apply (*rule Topological-Spaces.IVT'*)
apply (*simp-all add*: *Complex-Transcendental.Arg-ge-0*)
apply (*intro continuous-intros indefinite-integral-continuous winding-number-exp-integral* [*OF gpd*]; *simp*)
done
then obtain *t* **where** *t*: *t* ∈ {*0..1*}
and *eqArg*: *Im* (*integral* {*0..t*} (λ*x*. *vector-derivative* γ (*at x*)/(γ *x* − *z*))) = *Arg r*
by *blast*
def *i* ≡ *integral* {*0..t*} (λ*x*. *vector-derivative* γ (*at x*) / (γ *x* − *z*))
have *iArg*: *Arg r* = *Im i*
using *eqArg* **by** (*simp add*: *i-def*)
have *gpdt*: γ *piecewise-C1-differentiable-on* {*0..t*}
by (*metis atLeastAtMost-iff atLeastatMost-subset-iff order-refl piecewise-C1-differentiable-on-subset gpd t*)
have *exp* (− *i*) ∗ (γ *t* − *z*) = γ *0* − *z*
unfolding *i-def*
apply (*rule winding-number-exp-integral* [*OF gpdt*])
using *t z* **unfolding** *path-image-def*
apply *force*+
done
then have ∗: γ *t* − *z* = *exp i* ∗ (γ *0* − *z*)
by (*simp add*: *exp-minus field-simps*)
then have (*w* − *z*) = *r* ∗ (γ *0* − *z*)
by (*simp add*: *r-def*)
then have *z* + *complex-of-real* (*exp* (*Re i*)) ∗ (*w* − *z*) / *complex-of-real* (*cmod r*) = γ *t*
apply (*simp add*:)
apply (*subst Complex-Transcendental.Arg-eq* [*of r*])
apply (*simp add*: *iArg*)
using ∗
apply (*simp add*: *exp-eq-polar field-simps*)
done
with *t* **show** *?thesis*
by (*rule-tac x*=*exp*(*Re i*) / *norm r* **in** *exI*) (*auto simp*: *path-image-def*)
qed

lemma *winding-number-big-meets*:
fixes *z*::*complex*
assumes γ: *valid-path* γ **and** *z*: *z* ∉ *path-image* γ **and** |*Re* (*winding-number* γ *z*)| ≥ *1*
and *w*: *w* ≠ *z*
shows ∃ *a*::*real*. *0* < *a* ∧ *z* + *a*∗(*w* − *z*) ∈ *path-image* γ

proof −
{ assume *Re* (*winding-number* γ *z*) ≤ -1
then have *Re* (*winding-number* (*reversepath* γ) *z*) ≥ 1
by (*simp add*: γ *valid-path-imp-path winding-number-reversepath z*)
moreover have *valid-path* (*reversepath* γ)
using γ *valid-path-imp-reverse* **by** *auto*
moreover have *z* $\notin$ *path-image* (*reversepath* γ)
by (*simp add*: *z*)
ultimately have $\exists$ *a*::*real*. $0 < a \wedge z + a*(w - z) \in$ *path-image* (*reversepath* γ)
using *winding-number-pos-meets w* **by** *blast*
then have *?thesis*
by *simp*
}
then show *?thesis*
using *assms*
by (*simp add*: *Groups.abs-if-class.abs-if winding-number-pos-meets split*: *split-if-asm*)
qed

lemma *winding-number-less-1*:
fixes *z*::*complex*
shows
⟦*valid-path* γ; *z* $\notin$ *path-image* γ; *w* $\neq$ *z*;
$\bigwedge$*a*::*real*. $0 < a \Longrightarrow z + a*(w - z) \notin$ *path-image* γ⟧
$\Longrightarrow |Re(\text{winding-number } \gamma\ z)| < 1$
by (*auto simp*: *not-less dest*: *winding-number-big-meets*)

One way of proving that WN=1 for a loop.

lemma *winding-number-eq-1*:
fixes *z*::*complex*
assumes γ: *valid-path* γ **and** *z*: *z* $\notin$ *path-image* γ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *0*: $0 < Re(\text{winding-number } \gamma\ z)$ **and** *2*: $Re(\text{winding-number } \gamma\ z) < 2$
shows *winding-number* γ *z* = *1*
proof −
have *winding-number* γ *z* $\in$ *Ints*
by (*simp add*: γ *integer-winding-number loop valid-path-imp-path z*)
then show *?thesis*
using *0 2* **by** (*auto simp*: *Ints-def*)
qed

52.26 Continuity of winding number and invariance on connected sets.

lemma *continuous-at-winding-number*:
fixes *z*::*complex*
assumes γ: *path* γ **and** *z*: *z* $\notin$ *path-image* γ
shows *continuous* (*at z*) (*winding-number* γ)
proof −

obtain *e* **where** *e>0* **and** *cbg*: *cball z e* $\subseteq -$ *path-image* γ
using *open-contains-cball* [*of* $-$ *path-image* γ] *z*
by (*force simp*: *closed-def* [*symmetric*] *closed-path-image* [*OF* γ])
then have *ppag*: *path-image* $\gamma \subseteq -$ *cball z* (*e*/2)
by (*force simp*: *cball-def dist-norm*)
have *oc*: *open* (− *cball z* (*e* / 2))
by (*simp add*: *closed-def* [*symmetric*])
obtain *d* **where** *d>0* **and** *pi-eq*:
$\bigwedge$*h1 h2*. ⟦*valid-path h1*; *valid-path h2*;
(∀ *t*∈{*0..1*}. *cmod* (*h1 t* − γ *t*) < *d* ∧ *cmod* (*h2 t* − γ *t*) < *d*);
pathstart h2 = *pathstart h1*; *pathfinish h2* = *pathfinish h1*⟧
⟹
path-image h1 ⊆ − *cball z* (*e* / 2) ∧
path-image h2 ⊆ − *cball z* (*e* / 2) ∧
(∀ *f*. *f holomorphic-on* − *cball z* (*e* / 2) ⟶ *contour-integral h2 f* = *contour-integral h1 f*)
using *contour-integral-nearby-ends* [*OF oc* γ *ppag*] **by** *metis*
obtain *p* **where** *p*: *valid-path p z* ∉ *path-image p*
pathstart p = *pathstart* γ ∧ *pathfinish p* = *pathfinish* γ
and *pg*: $\bigwedge$*t*. *t*∈{*0..1*} ⟹ *cmod* (γ *t* − *p t*) < *min d e* / 2
and *pi*: *contour-integral p* (λ*x*. *1* / (*x* − *z*)) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number* γ *z*
using *winding-number* [*OF* γ *z*, *of min d e* / *2*] ‹*d>0*› ‹*e>0*› **by** *auto*
{ fix *w*
assume *d2*: *cmod* (*w* − *z*) < *d*/2 **and** *e2*: *cmod* (*w* − *z*) < *e*/2
then have *wnotp*: *w* ∉ *path-image p*
using *cbg* ‹*d>0*› ‹*e>0*›
apply (*simp add*: *path-image-def cball-def dist-norm*, *clarify*)
apply (*frule pg*)
apply (*drule-tac c*=γ *x* **in** *subsetD*)
apply (*auto simp*: *less-eq-real-def norm-minus-commute norm-triangle-half-l*)
done
have *wnotg*: *w* ∉ *path-image* γ
using *cbg e2* ‹*e>0*› **by** (*force simp*: *dist-norm norm-minus-commute*)
{ fix *k*::*real*
assume *k*: *k>0*
then obtain *q* **where** *q*: *valid-path q w* ∉ *path-image q*
pathstart q = *pathstart* γ ∧ *pathfinish q* = *pathfinish* γ
and *qg*: $\bigwedge$*t*. *t* ∈ {*0..1*} ⟹ *cmod* (γ *t* − *q t*) < *min k* (*min d e*) / *2*
and *qi*: *contour-integral q* (λ*u*. *1* / (*u* − *w*)) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number* γ *w*
using *winding-number* [*OF* γ *wnotg*, *of min k* (*min d e*) / *2*] ‹*d>0*› ‹*e>0*› *k*
by (*force simp*: *min-divide-distrib-right*)
have *contour-integral p* (λ*u*. *1* / (*u* − *w*)) = *contour-integral q* (λ*u*. *1* / (*u* − *w*))
apply (*rule pi-eq* [*OF* ‹*valid-path q*› ‹*valid-path p*›, *THEN conjunct2*, *THEN conjunct2*, *rule-format*])

apply (*frule pg*)
apply (*frule qg*)
using *p q* ‹*d>0*› *e2*
apply (*auto simp*: *dist-norm norm-minus-commute intro*!: *holomorphic-intros*)
done
then have *contour-integral p* (λ*x*. *1* / (*x* − *w*)) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number* γ *w*
by (*simp add*: *pi qi*)
} note *pip* = *this*
have *path p*
using *p* **by** (*simp add*: *valid-path-imp-path*)
then have *winding-number p w* = *winding-number* γ *w*
apply (*rule winding-number-unique* [*OF - wnotp*])
apply (*rule-tac x*=*p* **in** *exI*)
apply (*simp add*: *p wnotp min-divide-distrib-right pip*)
done
} note *wnwn* = *this*
obtain *pe* **where** *pe>0* **and** *cbp*: *cball z* (*3* / *4* ∗ *pe*) ⊆ − *path-image p*
using *p open-contains-cball* [*of* − *path-image p*]
by (*force simp*: *closed-def* [*symmetric*] *closed-path-image* [*OF valid-path-imp-path*])
obtain *L*
where *L>0*
and *L*: ⋀*f B*. ⟦*f holomorphic-on* − *cball z* (*3* / *4* ∗ *pe*);
∀ *z* ∈ − *cball z* (*3* / *4* ∗ *pe*). *cmod* (*f z*) ≤ *B*⟧ ⟹
cmod (*contour-integral p f*) ≤ *L* ∗ *B*
using *contour-integral-bound-exists* [*of* − *cball z* (*3/4*∗*pe*) *p*] *cbp* ‹*valid-path p*› **by** *force*
{ fix *e*::*real* **and** *w*::*complex*
assume *e*: *0* < *e* **and** *w*: *cmod* (*w* − *z*) < *pe/4 cmod* (*w* − *z*) < *e* ∗ *pe*^{2} / (*8* ∗ *L*)
then have [*simp*]: *w* ∉ *path-image p*
using *cbp p*(*2*) ‹*0* < *pe*›
by (*force simp*: *dist-norm norm-minus-commute path-image-def cball-def*)
have [*simp*]: *contour-integral p* (λ*x*. *1*/(*x* − *w*)) − *contour-integral p* (λ*x*. *1*/(*x* − *z*)) =
contour-integral p (λ*x*. *1*/(*x* − *w*) − *1*/(*x* − *z*))
by (*simp add*: *p contour-integrable-inversediff contour-integral-diff*)
{ fix *x*
assume *pe*: *3/4* ∗ *pe* < *cmod* (*z* − *x*)
have *cmod* (*w* − *x*) < *pe/4* + *cmod* (*z* − *x*)
by (*meson add-less-cancel-right norm-diff-triangle-le order-refl order-trans-rules*(*21*) *w*(*1*))
then have *wx*: *cmod* (*w* − *x*) < *4/3* ∗ *cmod* (*z* − *x*) **using** *pe* **by** *simp*
have *cmod* (*z* − *x*) ≤ *cmod* (*z* − *w*) + *cmod* (*w* − *x*)
using *norm-diff-triangle-le* **by** *blast*
also have ... < *pe/4* + *cmod* (*w* − *x*)
using *w* **by** (*simp add*: *norm-minus-commute*)
finally have *pe/2* < *cmod* (*w* − *x*)
using *pe* **by** *auto*

then have $(pe/2)\hat{}2 < cmod\ (w - x)\ \hat{}\ 2$
apply (*rule power-strict-mono*)
using ‹*pe>0*› **by** *auto*
then have *pe2*: $pe\hat{}2 < 4 * cmod\ (w - x)\ \hat{}\ 2$
by (*simp add*: *power-divide*)
have $8 * L * cmod\ (w - z) < e * pe^2$
using *w* ‹*L>0*› **by** (*simp add*: *field-simps*)
also have $... < e * 4 * cmod\ (w - x) * cmod\ (w - x)$
using *pe2* ‹*e>0*› **by** (*simp add*: *power2-eq-square*)
also have $... < e * 4 * cmod\ (w - x) * (4/3 * cmod\ (z - x))$
using *wx*
apply (*rule mult-strict-left-mono*)
using *pe2 e not-less-iff-gr-or-eq* **by** *fastforce*
finally have $L * cmod\ (w - z) < 2/3 * e * cmod\ (w - x) * cmod\ (z - x)$
by *simp*
also have $... \leq e * cmod\ (w - x) * cmod\ (z - x)$
using *e* **by** *simp*
finally have *Lwz*: $L * cmod\ (w - z) < e * cmod\ (w - x) * cmod\ (z - x)$ **.**
have $L * cmod\ (1\ /\ (x - w) - 1\ /\ (x - z)) \leq e$
apply (*cases* $x{=}z \vee x{=}w$)
using *pe* ‹*pe>0*› *w* ‹*L>0*›
apply (*force simp*: *norm-minus-commute*)
using *wx w*(*2*) ‹*L>0*› *pe pe2 Lwz*
apply (*auto simp*: *divide-simps mult-less-0-iff norm-minus-commute norm-divide norm-mult power2-eq-square*)
done
} note *L-cmod-le* = *this*
have ∗: $cmod\ (contour\text{-}integral\ p\ (\lambda x.\ 1\ /\ (x - w) - 1\ /\ (x - z))) \leq L * (e * pe^2\ /\ L\ /\ 4 * (inverse\ (pe\ /\ 2))^2)$
apply (*rule L*)
using ‹*pe>0*› *w*
apply (*force simp*: *dist-norm norm-minus-commute intro*!: *holomorphic-intros*)
using ‹*pe>0*› *w* ‹*L>0*›
apply (*auto simp*: *cball-def dist-norm field-simps L-cmod-le simp del*: *less-divide-eq-numeral1 le-divide-eq-numeral1*)
done
have $cmod\ (contour\text{-}integral\ p\ (\lambda x.\ 1\ /\ (x - w)) - contour\text{-}integral\ p\ (\lambda x.\ 1\ /\ (x - z))) < 2{*}e$
apply (*simp add*:)
apply (*rule le-less-trans* [*OF* ∗])
using ‹*L>0*› *e*
apply (*force simp*: *field-simps*)
done
then have $cmod\ (winding\text{-}number\ p\ w - winding\text{-}number\ p\ z) < e$
using *pi-ge-two e*
by (*force simp*: *winding-number-valid-path p field-simps norm-divide norm-mult intro*: *less-le-trans*)
} note *cmod-wn-diff* = *this*
then have *isCont* (*winding-number p*) *z*

apply (*simp add*: *continuous-at-eps-delta*, *clarify*)
apply (*rule-tac x=min* (*pe/4*) (*e/2*pe^2/L/4*) **in** *exI*)
using ‹*pe>0*› ‹*L>0*›
apply (*simp add*: *dist-norm cmod-wn-diff*)
done
then show *?thesis*
apply (*rule continuous-transform-within* [**where** *d* = *min d e* / *2*])
apply (*auto simp*: ‹*d>0*› ‹*e>0*› *dist-norm wnwn*)
done
qed

corollary *continuous-on-winding-number*:
path $\gamma \Longrightarrow$ *continuous-on* (− *path-image* γ) (λw. *winding-number* γ *w*)
by (*simp add*: *continuous-at-imp-continuous-on continuous-at-winding-number*)

52.27 The winding number is constant on a connected region

lemma *winding-number-constant*:
assumes γ: *path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ **and** *cs*: *connected s* **and** *sg*: $s \cap$ *path-image* $\gamma = \{\}$
shows $\exists k.\ \forall z \in s.$ *winding-number* γ *z* = *k*
proof −
{ fix *y z*
assume *ne*: *winding-number* γ *y* ≠ *winding-number* γ *z*
assume *y* ∈ *s z* ∈ *s*
then have *winding-number* γ *y* ∈ ℤ *winding-number* γ *z* ∈ ℤ
using *integer-winding-number* [*OF* γ *loop*] *sg* ‹*y* ∈ *s*› **by** *auto*
with *ne* **have** *1* ≤ *cmod* (*winding-number* γ *y* − *winding-number* γ *z*)
by (*auto simp*: *Ints-def of-int-diff* [*symmetric*] *simp del*: *of-int-diff*)
} note ∗ = *this*
show *?thesis*
apply (*rule continuous-discrete-range-constant* [*OF cs*])
using *continuous-on-winding-number* [*OF* γ] *sg*
apply (*metis Diff-Compl Diff-eq-empty-iff continuous-on-subset*)
apply (*rule-tac x=1* **in** *exI*)
apply (*auto simp*: ∗)
done
qed

lemma *winding-number-eq*:
⟦*path* γ; *pathfinish* γ = *pathstart* γ; *w* ∈ *s*; *z* ∈ *s*; *connected s*; $s \cap$ *path-image* $\gamma = \{\}$⟧
$\Longrightarrow$ *winding-number* γ *w* = *winding-number* γ *z*
using *winding-number-constant* **by** *fastforce*

lemma *open-winding-number-levelsets*:
assumes γ: *path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ
shows *open* {*z*. *z* ∉ *path-image* γ ∧ *winding-number* γ *z* = *k*}
proof −

have *op*: *open* (− *path-image* γ)
by (*simp add*: *closed-path-image* γ *open-Compl*)
{ fix z **assume** z: $z \notin$ *path-image* γ **and** k: $k =$ *winding-number* γ z
obtain e **where** e: $e>0$ *ball* z $e \subseteq -$ *path-image* γ
using *open-contains-ball* [*of* − *path-image* γ] *op* z
by *blast*
have $\exists\, e>0.\ \forall\, y.$ *dist* y $z < e \longrightarrow y \notin$ *path-image* $\gamma \wedge$ *winding-number* γ $y =$ *winding-number* γ z
apply (*rule-tac* x=e **in** *exI*)
using e **apply** (*simp add*: *dist-norm ball-def norm-minus-commute*)
apply (*auto simp*: *dist-norm norm-minus-commute intro*!: *winding-number-eq* [*OF assms*, **where** $s =$ *ball* z e])
done
} then
show *?thesis*
by (*auto simp*: *open-dist*)
qed

52.28 Winding number is zero "outside" a curve, in various senses

lemma *winding-number-zero-in-outside*:
assumes γ: *path* γ **and** *loop*: *pathfinish* $\gamma =$ *pathstart* γ **and** z: $z \in$ *outside* (*path-image* γ)
shows *winding-number* γ $z = 0$
proof −
obtain B::*real* **where** $0 < B$ **and** B: *path-image* $\gamma \subseteq$ *ball* 0 B
using *bounded-subset-ballD* [*OF bounded-path-image* [*OF* γ]] **by** *auto*
obtain w::*complex* **where** w: $w \notin$ *ball* 0 $(B + 1)$
by (*metis abs-of-nonneg le-less less-irrefl mem-ball-0 norm-of-real*)
have − *ball* 0 $(B + 1) \subseteq$ *outside* (*path-image* γ)
apply (*rule outside-subset-convex*)
using B *subset-ball* **by** *auto*
then have *wout*: $w \in$ *outside* (*path-image* γ)
using w **by** *blast*
moreover obtain k **where** $\bigwedge z.\ z \in$ *outside* (*path-image* γ) $\Longrightarrow$ *winding-number* γ $z = k$
using *winding-number-constant* [*OF* γ *loop*, *of outside*(*path-image* γ)] *connected-outside*
by (*metis DIM-complex bounded-path-image dual-order.refl* γ *outside-no-overlap*)
ultimately have *winding-number* γ $z =$ *winding-number* γ w
using z **by** *blast*
also have ... $= 0$
proof −
have *wnot*: $w \notin$ *path-image* γ **using** *wout* **by** (*simp add*: *outside-def*)
{ fix e::*real* **assume** $0<e$
obtain p **where** p: *polynomial-function* p *pathstart* $p =$ *pathstart* γ *pathfinish* $p =$ *pathfinish* γ
and *pg1*: $(\bigwedge t.\ [\![0 \le t;\ t \le 1]\!] \Longrightarrow$ *cmod* $(p\ t - \gamma\ t) < 1)$
and *pge*: $(\bigwedge t.\ [\![0 \le t;\ t \le 1]\!] \Longrightarrow$ *cmod* $(p\ t - \gamma\ t) < e)$

```
      using path-approx-polynomial-function [OF γ, of min 1 e] ‹e>0› by force
    have pip: path-image p ⊆ ball 0 (B + 1)
      using B
      apply (clarsimp simp add: path-image-def dist-norm ball-def)
      apply (frule (1) pg1)
      apply (fastforce dest: norm-add-less)
      done
    then have w ∉ path-image p  using w by blast
    then have ∃p. valid-path p ∧ w ∉ path-image p ∧
                  pathstart p = pathstart γ ∧ pathfinish p = pathfinish γ ∧
                  (∀t∈{0..1}. cmod (γ t − p t) < e) ∧ contour-integral p (λwa. 1
/ (wa − w)) = 0
      apply (rule-tac x=p in exI)
      apply (simp add: p valid-path-polynomial-function)
      apply (intro conjI)
      using pge apply (simp add: norm-minus-commute)
     apply (rule contour-integral-unique [OF Cauchy-theorem-convex-simple [OF
- convex-ball [of 0 B+1]]])
      apply (rule holomorphic-intros | simp add: dist-norm)+
      using mem-ball-0 w apply blast
      using p apply (simp-all add: valid-path-polynomial-function loop pip)
      done
   }
   then show ?thesis
     by (auto intro: winding-number-unique [OF γ] simp add: wnot)
 qed
 finally show ?thesis .
qed

lemma winding-number-zero-outside:
    ⟦path γ; convex s; pathfinish γ = pathstart γ; z ∉ s; path-image γ ⊆ s⟧ ⟹
winding-number γ z = 0
 by (meson convex-in-outside outside-mono subsetCE winding-number-zero-in-outside)

lemma winding-number-zero-at-infinity:
  assumes γ: path γ and loop: pathfinish γ = pathstart γ
    shows ∃B. ∀z. B ≤ norm z ⟶ winding-number γ z = 0
proof −
  obtain B::real where 0 < B and B: path-image γ ⊆ ball 0 B
    using bounded-subset-ballD [OF bounded-path-image [OF γ]] by auto
  then show ?thesis
    apply (rule-tac x=B+1 in exI, clarify)
    apply (rule winding-number-zero-outside [OF γ convex-cball [of 0 B] loop])
    apply (meson less-add-one mem-cball-0 not-le order-trans)
    using ball-subset-cball by blast
qed

lemma winding-number-zero-point:
    ⟦path γ; convex s; pathfinish γ = pathstart γ; open s; path-image γ ⊆ s⟧
```

$\Longrightarrow \exists z.\ z \in s \wedge$ *winding-number* $\gamma\ z = 0$
using *outside-compact-in-open* [*of path-image* γ *s*] *path-image-nonempty winding-number-zero-in-outside*
by (*fastforce simp add*: *compact-path-image*)

If a path winds round a set, it winds rounds its inside.

lemma *winding-number-around-inside*:
assumes γ: *path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *cls*: *closed s* **and** *cos*: *connected s* **and** *s-disj*: $s \cap$ *path-image* $\gamma = \{\}$
and z: $z \in s$ **and** *wn-nz*: *winding-number* $\gamma\ z \neq 0$ **and** w: $w \in s \cup$ *inside s*
shows *winding-number* $\gamma\ w$ = *winding-number* $\gamma\ z$
proof −
have *ssb*: $s \subseteq$ *inside*(*path-image* γ)
proof
fix x :: *complex*
assume $x \in s$
hence $x \notin$ *path-image* γ
by (*meson disjoint-iff-not-equal s-disj*)
thus $x \in$ *inside* (*path-image* γ)
using ‹$x \in s$› **by** (*metis* (*no-types*) *ComplI UnE cos* γ *loop s-disj union-with-outside winding-number-eq winding-number-zero-in-outside wn-nz z*)
qed
show *?thesis*
apply (*rule winding-number-eq* [*OF* γ *loop* w])
using z **apply** *blast*
apply (*simp add*: *cls connected-with-inside cos*)
apply (*simp add*: *Int-Un-distrib2 s-disj*, *safe*)
by (*meson ssb inside-inside-compact-connected* [*OF cls*, *of path-image* γ] *compact-path-image connected-path-image contra-subsetD disjoint-iff-not-equal* γ *inside-no-overlap*)
qed

Bounding a WN by 1/2 for a path and point in opposite halfspaces.

lemma *winding-number-subpath-continuous*:
assumes γ: *valid-path* γ **and** z: $z \notin$ *path-image* γ
shows *continuous-on* {*0..1*} ($\lambda x.$ *winding-number*(*subpath 0 x* γ) z)
proof −
have ∗: *integral* {*0..x*} ($\lambda t.$ *vector-derivative* γ (*at t*) / ($\gamma\ t - z$)) / (*2* ∗ *of-real pi* ∗ i) =
winding-number (*subpath 0 x* γ) z
if x: $0 \le x\ x \le 1$ **for** x
proof −
have *integral* {*0..x*} ($\lambda t.$ *vector-derivative* γ (*at t*) / ($\gamma\ t - z$)) / (*2* ∗ *of-real pi* ∗ i) =
1 / (*2*∗*pi*∗*ii*) ∗ *contour-integral* (*subpath 0 x* γ) ($\lambda w.$ *1*/($w - z$))
using *assms x*
apply (*simp add*: *contour-integral-subcontour-integral* [*OF contour-integrable-inversediff*])
done
also have ... = *winding-number* (*subpath 0 x* γ) z
apply (*subst winding-number-valid-path*)
using *assms x*

```
    apply (simp-all add: path-image-subpath valid-path-subpath)
    by (force simp: path-image-def)
  finally show ?thesis .
 qed
 show ?thesis
  apply (rule continuous-on-eq
          [where f = λx. 1 / (2*pi*ii) *
                       integral {0..x} (λt. 1/(γ t − z) * vector-derivative γ
(at t))])
  apply (rule continuous-intros)+
  apply (rule indefinite-integral-continuous)
 apply (rule contour-integrable-inversediff [OF assms, unfolded contour-integrable-on])
   using assms
  apply (simp add: *)
  done
qed

lemma winding-number-ivt-pos:
  assumes γ: valid-path γ and z: z ∉ path-image γ and 0 ≤ w w ≤ Re(winding-number
γ z)
    shows ∃ t ∈ {0..1}. Re(winding-number(subpath 0 t γ) z) = w
  apply (rule ivt-increasing-component-on-1 [of 0 1, where ?k = 1::complex,
simplified complex-inner-1-right])
 apply (simp add:)
 apply (rule winding-number-subpath-continuous [OF γ z])
 using assms
 apply (auto simp: path-image-def image-def)
 done

lemma winding-number-ivt-neg:
   assumes γ: valid-path γ and z: z ∉ path-image γ and Re(winding-number γ
z) ≤ w w ≤ 0
    shows ∃ t ∈ {0..1}. Re(winding-number(subpath 0 t γ) z) = w
  apply (rule ivt-decreasing-component-on-1 [of 0 1, where ?k = 1::complex,
simplified complex-inner-1-right])
 apply (simp add:)
 apply (rule winding-number-subpath-continuous [OF γ z])
 using assms
 apply (auto simp: path-image-def image-def)
 done

lemma winding-number-ivt-abs:
  assumes γ: valid-path γ and z: z ∉ path-image γ and 0 ≤ w w ≤ |Re(winding-number
γ z)|
    shows ∃ t ∈ {0..1}. |Re (winding-number (subpath 0 t γ) z)| = w
 using assms winding-number-ivt-pos [of γ z w] winding-number-ivt-neg [of γ z
−w]
 by force
```

lemma *winding-number-lt-half-lemma*:
assumes γ: *valid-path* γ **and** z: $z \notin$ *path-image* γ **and** *az*: $a \cdot z \leq b$ **and** *pag*: *path-image* $\gamma \subseteq \{w.\ a \cdot w > b\}$
shows *Re*(*winding-number* γ z) $< 1/2$
proof −
{ assume *Re*(*winding-number* γ z) $\geq 1/2$
then obtain t::*real* **where** t: $0 \leq t\ t \leq 1$ **and** *sub12*: *Re* (*winding-number* (*subpath 0 t* γ) z) $= 1/2$
using *winding-number-ivt-pos* [*OF* γ z, *of 1/2*] **by** *auto*
have *gt*: $\gamma\ t - z = -$ (*of-real* (*exp* (− (2 ∗ *pi* ∗ *Im* (*winding-number* (*subpath 0 t* γ) z)))) ∗ (γ 0 − z))
using *winding-number-exp-2pi* [*of subpath 0 t* γ z]
apply (*simp add*: *t* γ *valid-path-imp-path*)
using *closed-segment-eq-real-ivl path-image-def t z* **by** (*fastforce simp*: *path-image-subpath Euler sub12*)
have $b < a \cdot \gamma\ 0$
proof −
have $\gamma\ 0 \in \{c.\ b < a \cdot c\}$
by (*metis* (*no-types*) *pag atLeastAtMost-iff image-subset-iff order-refl path-image-def zero-le-one*)
thus *?thesis*
by *blast*
qed
moreover have $b < a \cdot \gamma\ t$
proof −
have $\gamma\ t \in \{c.\ b < a \cdot c\}$
by (*metis* (*no-types*) *pag atLeastAtMost-iff image-subset-iff path-image-def t*)
thus *?thesis*
by *blast*
qed
ultimately have $0 < a \cdot (\gamma\ 0 - z)$ $0 < a \cdot (\gamma\ t - z)$ **using** *az*
by (*simp add*: *inner-diff-right*)+
then have *False*
by (*simp add*: *gt inner-mult-right mult-less-0-iff*)
}
then show *?thesis* **by** *force*
qed

lemma *winding-number-lt-half*:
assumes *valid-path* γ $a \cdot z \leq b$ *path-image* $\gamma \subseteq \{w.\ a \cdot w > b\}$
shows $|$*Re* (*winding-number* γ z)$| < 1/2$
proof −
have $z \notin$ *path-image* γ **using** *assms* **by** *auto*
with *assms* **show** *?thesis*
apply (*simp add*: *winding-number-lt-half-lemma abs-if del*: *less-divide-eq-numeral1*)
apply (*metis complex-inner-1-right winding-number-lt-half-lemma* [*OF valid-path-imp-reverse*, *of* γ z a b]
winding-number-reversepath valid-path-imp-path inner-minus-left

path-image-reversepath)
done
qed

lemma *winding-number-le-half*:
assumes γ: *valid-path* γ **and** z: $z \notin$ *path-image* γ
and *anz*: $a \neq 0$ **and** *azb*: $a \cdot z \leq b$ **and** *pag*: *path-image* $\gamma \subseteq \{w.\ a \cdot w \geq b\}$
shows $|Re\ (winding\text{-}number\ \gamma\ z)| \leq 1/2$
proof −
{ assume *wnz-12*: $|Re\ (winding\text{-}number\ \gamma\ z)| > 1/2$
have *isCont* (*winding-number* γ) z
by (*metis continuous-at-winding-number valid-path-imp-path* γ z)
then obtain d **where** $d{>}0$ **and** d: $\bigwedge x'.\ dist\ x'\ z < d \Longrightarrow dist\ (winding\text{-}number\ \gamma\ x')\ (winding\text{-}number\ \gamma\ z) < |Re(winding\text{-}number\ \gamma\ z)| - 1/2$
using *continuous-at-eps-delta wnz-12 diff-gt-0-iff-gt* **by** *blast*
def $z' \equiv z - (d\ /\ (2 * cmod\ a)) *_R a$
have $*$: $a \cdot z' \leq b - d\ /\ 3 * cmod\ a$
unfolding z'*-def inner-mult-right' divide-inverse*
apply (*simp add*: *divide-simps algebra-simps dot-square-norm power2-eq-square anz*)
apply (*metis* ‹$0 < d$› *add-increasing azb less-eq-real-def mult-nonneg-nonneg mult-right-mono norm-ge-zero norm-numeral*)
done
have *cmod* $(winding\text{-}number\ \gamma\ z' - winding\text{-}number\ \gamma\ z) < |Re\ (winding\text{-}number\ \gamma\ z)| - 1/2$
using d [*of* z'] *anz* ‹$d{>}0$› **by** (*simp add*: *dist-norm* z'*-def*)
then have $1/2 < |Re\ (winding\text{-}number\ \gamma\ z)| - cmod\ (winding\text{-}number\ \gamma\ z' - winding\text{-}number\ \gamma\ z)$
by *simp*
then have $1/2 < |Re\ (winding\text{-}number\ \gamma\ z)| - |Re\ (winding\text{-}number\ \gamma\ z') - Re\ (winding\text{-}number\ \gamma\ z)|$
using *abs-Re-le-cmod* [*of winding-number* γ z' − *winding-number* γ z] **by** *simp*
then have *wnz-12'*: $|Re\ (winding\text{-}number\ \gamma\ z')| > 1/2$
by *linarith*
moreover have $|Re\ (winding\text{-}number\ \gamma\ z')| < 1/2$
apply (*rule winding-number-lt-half* [*OF* γ $*$])
using *azb* ‹$d{>}0$› *pag*
apply (*auto simp*: *add-strict-increasing anz divide-simps algebra-simps dest*!: *subsetD*)
done
ultimately have *False*
by *simp*
}
then show *?thesis* **by** *force*
qed

lemma *winding-number-lt-half-linepath*: $z \notin$ *closed-segment* a $b \Longrightarrow |Re\ (winding\text{-}number\ (linepath\ a\ b)\ z)| < 1/2$

using *separating-hyperplane-closed-point* [*of closed-segment a b z*]
apply *auto*
apply (*simp add*: *closed-segment-def*)
apply (*drule less-imp-le*)
apply (*frule winding-number-lt-half* [*OF valid-path-linepath* [*of a b*]])
apply (*auto simp*: *segment*)
done

Positivity of WN for a linepath.

lemma *winding-number-linepath-pos-lt*:
assumes *0 < Im ((b − a) ∗ cnj (b − z))*
shows *0 < Re(winding-number(linepath a b) z)*
proof −
have *z*: *z ∉ path-image* (*linepath a b*)
using *assms*
by (*simp add*: *closed-segment-def*) (*force simp*: *algebra-simps*)
show *?thesis*
apply (*rule winding-number-pos-lt* [*OF valid-path-linepath z assms*])
apply (*simp add*: *linepath-def algebra-simps*)
done
qed

52.29 Cauchy's integral formula, again for a convex enclosing set.

lemma *Cauchy-integral-formula-weak*:
assumes *s*: *convex s* **and** *finite k* **and** *conf*: *continuous-on s f*
and *fcd*: (⋀*x*. *x ∈ interior s − k ⟹ f complex-differentiable at x*)
and *z*: *z ∈ interior s − k* **and** *vpg*: *valid-path γ*
and *pasz*: *path-image γ ⊆ s − {z}* **and** *loop*: *pathfinish γ = pathstart γ*
shows ((*λw*. *f w / (w − z)*) *has-contour-integral* (*2∗pi ∗ ii ∗ winding-number γ z ∗ f z*)) *γ*
proof −
obtain *f′* **where** *f′*: (*f has-field-derivative f′*) (*at z*)
using *fcd* [*OF z*] **by** (*auto simp*: *complex-differentiable-def*)
have *pas*: *path-image γ ⊆ s* **and** *znotin*: *z ∉ path-image γ* **using** *pasz* **by** *blast+*
have *c*: *continuous* (*at x within s*) (*λw*. *if w = z then f′ else (f w − f z) / (w − z)*) **if** *x ∈ s* **for** *x*
proof (*cases x = z*)
case *True* **then show** *?thesis*
apply (*simp add*: *continuous-within*)
apply (*rule Lim-transform-away-within* [*of - z+1 - λw::complex*. (*f w − f z*)*/*(*w − z*)])
using *has-field-derivative-at-within DERIV-within-iff f′*
apply (*fastforce simp add*:)+
done
next
case *False*
then have *dxz*: *dist x z > 0* **by** *auto*

have *cf*: *continuous* (*at x within s*) *f*
using *conf continuous-on-eq-continuous-within that* **by** *blast*
have *continuous* (*at x within s*) (λ*w*. (*f w* − *f z*) / (*w* − *z*))
by (*rule cf continuous-intros* | *simp add*: *False*)+
then show *?thesis*
apply (*rule continuous-transform-within* [*OF - dxz that*, *of* λ*w*::*complex*. (*f w* − *f z*)/(*w* − *z*)])
apply (*force simp*: *dist-commute*)
done
qed
have *fink'*: *finite* (*insert z k*) **using** ‹*finite k*› **by** *blast*
have ∗: ((λ*w*. *if w* = *z then f' else* (*f w* − *f z*) / (*w* − *z*)) *has-contour-integral* 0) γ
apply (*rule Cauchy-theorem-convex* [*OF - s fink' - vpg pas loop*])
using *c* **apply** (*force simp*: *continuous-on-eq-continuous-within*)
apply (*rename-tac w*)
apply (*rule-tac d*=*dist w z* **and** *f* = λ*w*. (*f w* − *f z*)/(*w* − *z*) **in** *complex-differentiable-transform-within*)
apply (*simp-all add*: *dist-pos-lt dist-commute*)
apply (*metis less-irrefl*)
apply (*rule derivative-intros fcd* | *simp*)+
done
show *?thesis*
apply (*rule has-contour-integral-eq*)
using *znotin has-contour-integral-add* [*OF has-contour-integral-lmul* [*OF has-contour-integral-winding-numb*
[*OF vpg znotin*], *of f z*] ∗]
apply (*auto simp*: *mult-ac divide-simps*)
done
qed

theorem *Cauchy-integral-formula-convex-simple*:
⟦*convex s*; *f holomorphic-on s*; *z* ∈ *interior s*; *valid-path* γ; *path-image* γ ⊆ *s* − {*z*};
pathfinish γ = *pathstart* γ⟧
⟹ ((λ*w*. *f w* / (*w* − *z*)) *has-contour-integral* (2∗*pi* ∗ *ii* ∗ *winding-number* γ *z* ∗ *f z*)) γ
apply (*rule Cauchy-integral-formula-weak* [**where** *k* = {}])
using *holomorphic-on-imp-continuous-on*
by *auto* (*metis at-within-interior holomorphic-on-def interiorE subsetCE*)

52.30 Homotopy forms of Cauchy's theorem

proposition *Cauchy-theorem-homotopic*:
assumes *hom*: *if atends then homotopic-paths s g h else homotopic-loops s g h*
and *open s* **and** *f*: *f holomorphic-on s*
and *vpg*: *valid-path g* **and** *vph*: *valid-path h*
shows *contour-integral g f* = *contour-integral h f*
proof −
have *pathsf*: *linked-paths atends g h*
using *hom* **by** (*auto simp*: *linked-paths-def homotopic-paths-imp-pathstart*

homotopic-paths-imp-pathfinish homotopic-loops-imp-loop)
obtain *k* :: *real* × *real* ⇒ *complex*
where *contk*: *continuous-on* ({*0..1*} × {*0..1*}) *k*
and *ks*: *k* ' ({*0..1*} × {*0..1*}) ⊆ *s*
and *k* [*simp*]: ∀ *x*. *k* (*0*, *x*) = *g x* ∀ *x*. *k* (*1*, *x*) = *h x*
and *ksf*: ∀ *t*∈{*0..1*}. *linked-paths atends g* (λ*x*. *k* (*t*, *x*))
using *hom pathsf* **by** (*auto simp*: *linked-paths-def homotopic-paths-def homotopic-loops-def homotopic-with-def split*: *split-if-asm*)
have *ucontk*: *uniformly-continuous-on* ({*0..1*} × {*0..1*}) *k*
by (*blast intro*: *compact-Times compact-uniformly-continuous* [*OF contk*])
{ fix *t*::*real* **assume** *t*: *t* ∈ {*0..1*}
have *pak*: *path* (*k o* (λ*u*. (*t*, *u*)))
unfolding *path-def*
apply (*rule continuous-intros continuous-on-subset* [*OF contk*])+
using *t* **by** *force*
have *pik*: *path-image* (*k* ∘ *Pair t*) ⊆ *s*
using *ks t* **by** (*auto simp*: *path-image-def*)
obtain *e* **where** *e>0* **and** *e*:
⋀*g h*. ⟦*valid-path g*; *valid-path h*;
∀ *u*∈{*0..1*}. *cmod* (*g u* − (*k* ∘ *Pair t*) *u*) < *e* ∧ *cmod* (*h u* − (*k* ∘ *Pair t*) *u*) < *e*;
linked-paths atends g h⟧
⟹ *contour-integral h f* = *contour-integral g f*
using *contour-integral-nearby* [*OF* ‹*open s*› *pak pik*, *of atends*] *f* **by** *metis*
obtain *d* **where** *d>0* **and** *d*:
⋀*x x'*. ⟦*x* ∈ {*0..1*} × {*0..1*}; *x'* ∈ {*0..1*} × {*0..1*}; *norm* (*x'*−*x*) < *d*⟧ ⟹ *norm* (*k x'* − *k x*) < *e*/*4*
by (*rule uniformly-continuous-onE* [*OF ucontk*, *of e*/*4*]) (*auto simp*: *dist-norm* ‹*e>0*›)
{ fix *t1 t2*
assume *t1*: *0* ≤ *t1 t1* ≤ *1* **and** *t2*: *0* ≤ *t2 t2* ≤ *1* **and** *ltd*: |*t1* − *t*| < *d* |*t2* − *t*| < *d*
have *no2*: ⋀*g1 k1 kt*. ⟦*norm*(*g1* − *k1*) < *e*/*4*; *norm*(*k1* − *kt*) < *e*/*4*⟧ ⟹ *norm*(*g1* − *kt*) < *e*
using ‹*e* > *0*›
apply (*rule-tac y* = *k1* **in** *norm-triangle-half-l*)
apply (*auto simp*: *norm-minus-commute intro*: *order-less-trans*)
done
have ∃ *d*>*0*. ∀ *g1 g2*. *valid-path g1* ∧ *valid-path g2* ∧
(∀ *u*∈{*0..1*}. *cmod* (*g1 u* − *k* (*t1*, *u*)) < *d* ∧ *cmod* (*g2 u* − *k* (*t2*, *u*)) < *d*) ∧
linked-paths atends g1 g2 ⟶
contour-integral g2 f = *contour-integral g1 f*
apply (*rule-tac x*=*e*/*4* **in** *exI*)
using *t t1 t2 ltd* ‹*e* > *0*›
apply (*auto intro*!: *e simp*: *d no2 simp del*: *less-divide-eq-numeral1*)
done
}
then have ∃ *e*. *0* < *e* ∧

$(\forall\, t1\ t2.\ t1 \in \{0..1\} \wedge t2 \in \{0..1\} \wedge |t1 - t| < e \wedge |t2 - t| < e$
$\longrightarrow (\exists\, d.\ 0 < d \wedge$
$(\forall\, g1\ g2.\ valid\text{-}path\ g1 \wedge valid\text{-}path\ g2 \wedge$
$(\forall\, u \in \{0..1\}.$
$norm(g1\ u - k((t1,u))) < d \wedge norm(g2\ u - k((t2,u))) < d)$
$\wedge$
$linked\text{-}paths\ atends\ g1\ g2$
$\longrightarrow contour\text{-}integral\ g2\ f = contour\text{-}integral\ g1\ f)))$
by (*rule-tac* $x{=}d$ **in** *exI*) (*simp add*: ‹$d > 0$›)
}
then obtain *ee* **where** *ee*:
$\bigwedge t.\ t \in \{0..1\} \Longrightarrow ee\ t > 0 \wedge$
$(\forall\, t1\ t2.\ t1 \in \{0..1\} \longrightarrow t2 \in \{0..1\} \longrightarrow |t1 - t| < ee\ t \longrightarrow |t2 - t| <$
$ee\ t$
$\longrightarrow (\exists\, d.\ 0 < d \wedge$
$(\forall\, g1\ g2.\ valid\text{-}path\ g1 \wedge valid\text{-}path\ g2 \wedge$
$(\forall\, u \in \{0..1\}.$
$norm(g1\ u - k((t1,u))) < d \wedge norm(g2\ u - k((t2,u))) < d) \wedge$
$linked\text{-}paths\ atends\ g1\ g2$
$\longrightarrow contour\text{-}integral\ g2\ f = contour\text{-}integral\ g1\ f)))$
by *metis*
note *ee-rule* = *ee* [*THEN conjunct2*, *rule-format*]
def $C \equiv (\lambda t.\ ball\ t\ (ee\ t\ /\ 3))\ `\ \{0..1\}$
have $\forall\, t \in C.\ open\ t$ **by** (*simp add*: *C-def*)
moreover have $\{0..1\} \subseteq \bigcup C$
using *ee* [*THEN conjunct1*] **by** (*auto simp*: *C-def dist-norm*)
ultimately obtain C' **where** C': $C' \subseteq C$ *finite* C' **and** $C'01$: $\{0..1\} \subseteq \bigcup C'$
by (*rule compactE* [*OF compact-interval*])
def $kk \equiv \{t \in \{0..1\}.\ ball\ t\ (ee\ t\ /\ 3) \in C'\}$
have *kk01*: $kk \subseteq \{0..1\}$ **by** (*auto simp*: *kk-def*)
def $e \equiv Min\ (ee\ `\ kk)$
have C'-*eq*: $C' = (\lambda t.\ ball\ t\ (ee\ t\ /\ 3))\ `\ kk$
using C' **by** (*auto simp*: *kk-def C-def*)
have *ee-pos*[*simp*]: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow ee\ t > 0$
by (*simp add*: *kk-def ee*)
moreover have *finite kk*
using ‹*finite* C'› *kk01* **by** (*force simp*: C'-*eq inj-on-def ball-eq-ball-iff dest*: *ee-pos finite-imageD*)
moreover have $kk \neq \{\}$ **using** ‹$\{0..1\} \subseteq \bigcup C'$› C'-*eq* **by** *force*
ultimately have $e > 0$
using *finite-less-Inf-iff* [*of ee ` kk 0*] *kk01* **by** (*force simp*: *e-def*)
then obtain N::*nat* **where** $N > 0$ **and** N: $1/N < e/3$
by (*meson divide-pos-pos nat-approx-posE zero-less-Suc zero-less-numeral*)
have *e-le-ee*: $\bigwedge i.\ i \in kk \Longrightarrow e \leq ee\ i$
using ‹*finite kk*› **by** (*simp add*: *e-def Min-le-iff* [*of ee ` kk*])
have *plus*: $\exists\, t \in kk.\ x \in ball\ t\ (ee\ t\ /\ 3)$ **if** $x \in \{0..1\}$ **for** x
using C' *subsetD* [*OF C'01 that*] **unfolding** C'-*eq* **by** *blast*
have [*OF order-refl*]:
$\exists\, d.\ 0 < d \wedge (\forall\, j.\ valid\text{-}path\ j \wedge (\forall\, u \in \{0..1\}.\ norm(j\ u - k\ (n/N,\ u)) <$

```
d) ∧ linked-paths atends g j
                    ⟶ contour-integral j f = contour-integral g f)
      if n ≤ N for n
  using that
  proof (induct n)
    case 0 show ?case using ee-rule [of 0 0 0]
      apply clarsimp
      apply (rule-tac x=d in exI, safe)
      by (metis diff-self vpg norm-zero)
  next
    case (Suc n)
    then have N01: n/N ∈ {0..1} (Suc n)/N ∈ {0..1}  by auto
    then obtain t where t: t ∈ kk n/N ∈ ball t (ee t / 3)
      using plus [of n/N] by blast
    then have nN-less: |n/N − t| < ee t
      by (simp add: dist-norm del: less-divide-eq-numeral1)
    have n'N-less: |real (Suc n) / real N − t| < ee t
      using t N ‹N > 0› e-le-ee [of t]
    by (simp add: dist-norm add-divide-distrib abs-diff-less-iff del: less-divide-eq-numeral1)
(simp add: field-simps)
    have t01: t ∈ {0..1} using ‹kk ⊆ {0..1}› ‹t ∈ kk› by blast
    obtain d1 where d1 > 0 and d1:
       ⋀g1 g2. ⟦valid-path g1; valid-path g2;
                  ∀u∈{0..1}. cmod (g1 u − k (n/N, u)) < d1 ∧ cmod (g2 u − k
((Suc n) / N, u)) < d1;
                  linked-paths atends g1 g2⟧
                  ⟹ contour-integral g2 f = contour-integral g1 f
      using ee [THEN conjunct2, rule-format, OF t01 N01 nN-less n'N-less] by
fastforce
    have n ≤ N using Suc.prems by auto
    with Suc.hyps
    obtain d2 where d2 > 0
       and d2: ⋀j. ⟦valid-path j; ∀u∈{0..1}. cmod (j u − k (n/N, u)) < d2;
linked-paths atends g j⟧
                  ⟹ contour-integral j f = contour-integral g f
       by auto
    have continuous-on {0..1} (k o (λu. (n/N, u)))
      apply (rule continuous-intros continuous-on-subset [OF contk])+
      using N01 by auto
    then have pkn: path (λu. k (n/N, u))
      by (simp add: path-def)
    have min12: min d1 d2 > 0 by (simp add: ‹0 < d1› ‹0 < d2›)
    obtain p where polynomial-function p
       and psf: pathstart p = pathstart (λu. k (n/N, u))
              pathfinish p = pathfinish (λu. k (n/N, u))
       and pk-le: ⋀t. t∈{0..1} ⟹ cmod (p t − k (n/N, t)) < min d1 d2
      using path-approx-polynomial-function [OF pkn min12] by blast
    then have vpp: valid-path p using valid-path-polynomial-function by blast
    have lpa: linked-paths atends g p
```

by (*metis* (*mono-tags*, *lifting*) *N01*(*1*) *ksf linked-paths-def pathfinish-def pathstart-def psf*)
show *?case*
apply (*rule-tac x*=*min d1 d2* **in** *exI*)
apply (*simp add*: ‹*0* < *d1*› ‹*0* < *d2*›, *clarify*)
apply (*rule-tac s*=*contour-integral p f* **in** *trans*)
using *pk-le N01*(*1*) *ksf pathfinish-def pathstart-def*
apply (*force intro*!: *vpp d1 simp add*: *linked-paths-def psf ksf*)
using *pk-le N01* **apply** (*force intro*!: *vpp d2 lpa simp add*: *linked-paths-def psf ksf*)
done
qed
then obtain *d* **where** *0* < *d*
⋀*j*. *valid-path j* ∧ (∀ *u* ∈ {*0*..*1*}. *norm*(*j u* − *k* (*1*,*u*)) < *d*) ∧
linked-paths atends g j
⟹ *contour-integral j f* = *contour-integral g f*
using ‹*N*>*0*› **by** *auto*
then have *linked-paths atends g h* ⟹ *contour-integral h f* = *contour-integral g f*
using ‹*N*>*0*› *vph* **by** *fastforce*
then show *?thesis*
by (*simp add*: *pathsf*)
qed

proposition *Cauchy-theorem-homotopic-paths*:
assumes *hom*: *homotopic-paths s g h*
and *open s* **and** *f*: *f holomorphic-on s*
and *vpg*: *valid-path g* **and** *vph*: *valid-path h*
shows *contour-integral g f* = *contour-integral h f*
using *Cauchy-theorem-homotopic* [*of True s g h*] *assms* **by** *simp*

proposition *Cauchy-theorem-homotopic-loops*:
assumes *hom*: *homotopic-loops s g h*
and *open s* **and** *f*: *f holomorphic-on s*
and *vpg*: *valid-path g* **and** *vph*: *valid-path h*
shows *contour-integral g f* = *contour-integral h f*
using *Cauchy-theorem-homotopic* [*of False s g h*] *assms* **by** *simp*

lemma *has-contour-integral-newpath*:
⟦(*f has-contour-integral y*) *h*; *f contour-integrable-on g*; *contour-integral g f* = *contour-integral h f*⟧
⟹ (*f has-contour-integral y*) *g*
using *has-contour-integral-integral contour-integral-unique* **by** *auto*

lemma *Cauchy-theorem-null-homotopic*:
⟦*f holomorphic-on s*; *open s*; *valid-path g*; *homotopic-loops s g* (*linepath a a*)⟧
⟹ (*f has-contour-integral 0*) *g*
apply (*rule has-contour-integral-newpath* [**where** *h* = *linepath a a*], *simp*)
using *contour-integrable-holomorphic-simple*

apply (*blast dest*: *holomorphic-on-imp-continuous-on homotopic-loops-imp-subset*)
by (*simp add*: *Cauchy-theorem-homotopic-loops*)

52.31 More winding number properties

including the fact that it's +-1 inside a simple closed curve.

lemma *winding-number-homotopic-paths*:
assumes *homotopic-paths* $(-\{z\})$ *g h*
shows *winding-number g z* = *winding-number h z*
proof −
have *path g path h* **using** *homotopic-paths-imp-path* [*OF assms*] **by** *auto*
moreover have *pag*: $z \notin$ *path-image g* **and** *pah*: $z \notin$ *path-image h*
using *homotopic-paths-imp-subset* [*OF assms*] **by** *auto*
ultimately obtain *d e* **where** $d > 0$ $e > 0$
and *d*: $\bigwedge p$. ⟦*path p*; *pathstart p* = *pathstart g*; *pathfinish p* = *pathfinish g*;
$\forall t \in \{0..1\}$. *norm* (*p t* − *g t*) < *d*⟧
⟹ *homotopic-paths* $(-\{z\})$ *g p*
and *e*: $\bigwedge q$. ⟦*path q*; *pathstart q* = *pathstart h*; *pathfinish q* = *pathfinish h*;
$\forall t \in \{0..1\}$. *norm* (*q t* − *h t*) < *e*⟧
⟹ *homotopic-paths* $(-\{z\})$ *h q*
using *homotopic-nearby-paths* [*of g* $-\{z\}$] *homotopic-nearby-paths* [*of h* $-\{z\}$]
by *force*
obtain *p* **where** *p*:
valid-path p z $\notin$ *path-image p*
pathstart p = *pathstart g pathfinish p* = *pathfinish g*
and *gp-less*:$\forall t \in \{0..1\}$. *cmod* (*g t* − *p t*) < *d*
and *pap*: *contour-integral p* (λ*w*. *1* / (*w* − *z*)) = *complex-of-real* (*2* ∗ *pi*) ∗
i ∗ *winding-number g z*
using *winding-number* [*OF* ‹*path g*› *pag* ‹*0* < *d*›] **by** *blast*
obtain *q* **where** *q*:
valid-path q z $\notin$ *path-image q*
pathstart q = *pathstart h pathfinish q* = *pathfinish h*
and *hq-less*: $\forall t \in \{0..1\}$. *cmod* (*h t* − *q t*) < *e*
and *paq*: *contour-integral q* (λ*w*. *1* / (*w* − *z*)) = *complex-of-real* (*2* ∗ *pi*)
∗ i ∗ *winding-number h z*
using *winding-number* [*OF* ‹*path h*› *pah* ‹*0* < *e*›] **by** *blast*
have *gp*: *homotopic-paths* (− {*z*}) *g p*
by (*simp add*: *d p valid-path-imp-path norm-minus-commute gp-less*)
have *hq*: *homotopic-paths* (− {*z*}) *h q*
by (*simp add*: *e q valid-path-imp-path norm-minus-commute hq-less*)
have *contour-integral p* (λ*w*. *1*/(*w* − *z*)) = *contour-integral q* (λ*w*. *1*/(*w* − *z*))
apply (*rule Cauchy-theorem-homotopic-paths* [*of* $-\{z\}$])
apply (*blast intro*: *homotopic-paths-trans homotopic-paths-sym gp hq assms*)
apply (*auto intro*!: *holomorphic-intros simp*: *p q*)
done
then show *?thesis*
by (*simp add*: *pap paq*)
qed

lemma *winding-number-homotopic-loops*:
  assumes *homotopic-loops* $(-\{z\})$ *g h*
  shows *winding-number g z* = *winding-number h z*
proof −
 have *path g path h* **using** *homotopic-loops-imp-path* [*OF assms*] **by** *auto*
 moreover have *pag*: $z \notin$ *path-image g* **and** *pah*: $z \notin$ *path-image h*
  using *homotopic-loops-imp-subset* [*OF assms*] **by** *auto*
 moreover have *gloop*: *pathfinish g* = *pathstart g* **and** *hloop*: *pathfinish h* = *pathstart h*
  using *homotopic-loops-imp-loop* [*OF assms*] **by** *auto*
 ultimately obtain *d e* **where** $d > 0$ $e > 0$
   and *d*: $\bigwedge p.$ ⟦*path p*; *pathfinish p* = *pathstart p*; $\forall t \in \{0..1\}$. *norm* (*p t* − *g t*) < *d*⟧
       ⟹ *homotopic-loops* $(-\{z\})$ *g p*
   and *e*: $\bigwedge q.$ ⟦*path q*; *pathfinish q* = *pathstart q*; $\forall t \in \{0..1\}$. *norm* (*q t* − *h t*) < *e*⟧
       ⟹ *homotopic-loops* $(-\{z\})$ *h q*
  using *homotopic-nearby-loops* [*of g* $-\{z\}$] *homotopic-nearby-loops* [*of h* $-\{z\}$] **by** *force*
 obtain *p* **where** *p*:
     valid-path p z $\notin$ *path-image p*
     pathstart p = *pathstart g pathfinish p* = *pathfinish g*
     and *gp-less*:$\forall t \in \{0..1\}$. *cmod* (*g t* − *p t*) < *d*
     and *pap*: *contour-integral p* (λ*w*. *1* / (*w* − *z*)) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number g z*
  using *winding-number* [*OF* ‹*path g*› *pag* ‹*0* < *d*›] **by** *blast*
 obtain *q* **where** *q*:
     valid-path q z $\notin$ *path-image q*
     pathstart q = *pathstart h pathfinish q* = *pathfinish h*
     and *hq-less*: $\forall t \in \{0..1\}$. *cmod* (*h t* − *q t*) < *e*
     and *paq*: *contour-integral q* (λ*w*. *1* / (*w* − *z*)) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number h z*
  using *winding-number* [*OF* ‹*path h*› *pah* ‹*0* < *e*›] **by** *blast*
 have *gp*: *homotopic-loops* (− {*z*}) *g p*
  by (*simp add*: *gloop d gp-less norm-minus-commute p valid-path-imp-path*)
 have *hq*: *homotopic-loops* (− {*z*}) *h q*
  by (*simp add*: *e hloop hq-less norm-minus-commute q valid-path-imp-path*)
 have *contour-integral p* (λ*w*. *1*/(*w* − *z*)) = *contour-integral q* (λ*w*. *1*/(*w* − *z*))
  apply (*rule Cauchy-theorem-homotopic-loops* [*of* $-\{z\}$])
  apply (*blast intro*: *homotopic-loops-trans homotopic-loops-sym gp hq assms*)
  apply (*auto intro*!: *holomorphic-intros simp*: *p q*)
  done
 then show *?thesis*
  by (*simp add*: *pap paq*)
qed

lemma *winding-number-paths-linear-eq*:
 ⟦*path g*; *path h*; *pathstart h* = *pathstart g*; *pathfinish h* = *pathfinish g*;
  $\bigwedge t.$ $t \in \{0..1\} \Longrightarrow z \notin$ *closed-segment* (*g t*) (*h t*)⟧

$\Longrightarrow$ *winding-number h z = winding-number g z*
by (*blast intro*: *sym homotopic-paths-linear winding-number-homotopic-paths elim*:
)

lemma *winding-number-loops-linear-eq*:
⟦*path g*; *path h*; *pathfinish g = pathstart g*; *pathfinish h = pathstart h*;
⋀*t*. *t* ∈ {*0..1*} $\Longrightarrow$ *z* ∉ *closed-segment* (*g t*) (*h t*)⟧
$\Longrightarrow$ *winding-number h z = winding-number g z*
by (*blast intro*: *sym homotopic-loops-linear winding-number-homotopic-loops elim*:
)

lemma *winding-number-nearby-paths-eq*:
⟦*path g*; *path h*;
pathstart h = pathstart g; *pathfinish h = pathfinish g*;
⋀*t*. *t* ∈ {*0..1*} $\Longrightarrow$ *norm*(*h t* − *g t*) < *norm*(*g t* − *z*)⟧
$\Longrightarrow$ *winding-number h z = winding-number g z*
by (*metis segment-bound*(*2*) *norm-minus-commute not-le winding-number-paths-linear-eq*)

lemma *winding-number-nearby-loops-eq*:
⟦*path g*; *path h*;
pathfinish g = pathstart g;
pathfinish h = pathstart h;
⋀*t*. *t* ∈ {*0..1*} $\Longrightarrow$ *norm*(*h t* − *g t*) < *norm*(*g t* − *z*)⟧
$\Longrightarrow$ *winding-number h z = winding-number g z*
by (*metis segment-bound*(*2*) *norm-minus-commute not-le winding-number-loops-linear-eq*)

proposition *winding-number-subpath-combine*:
⟦*path g*; *z* ∉ *path-image g*;
u ∈ {*0..1*}; *v* ∈ {*0..1*}; *w* ∈ {*0..1*}⟧
$\Longrightarrow$ *winding-number* (*subpath u v g*) *z* + *winding-number* (*subpath v w g*) *z* =
winding-number (*subpath u w g*) *z*
apply (*rule trans* [*OF winding-number-join* [*THEN sym*]
winding-number-homotopic-paths [*OF homotopic-join-subpaths*]])
apply (*auto dest*: *path-image-subpath-subset*)
done

52.32 Partial circle path

definition *part-circlepath* :: [*complex*, *real*, *real*, *real*, *real*] ⇒ *complex*
where *part-circlepath z r s t* ≡ λ*x*. *z* + *of-real r* ∗ *exp* (*ii* ∗ *of-real* (*linepath s t x*))

lemma *pathstart-part-circlepath* [*simp*]:
pathstart(*part-circlepath z r s t*) = *z* + *r*∗*exp*(*ii* ∗ *s*)
by (*metis part-circlepath-def pathstart-def pathstart-linepath*)

lemma *pathfinish-part-circlepath* [*simp*]:
pathfinish(*part-circlepath z r s t*) = *z* + *r*∗*exp*(*ii*∗*t*)

by (*metis part-circlepath-def pathfinish-def pathfinish-linepath*)

proposition *has-vector-derivative-part-circlepath* [*derivative-intros*]:
((*part-circlepath z r s t*) *has-vector-derivative*
(*ii* ∗ *r* ∗ (*of-real t* − *of-real s*) ∗ *exp*(*ii* ∗ *linepath s t x*)))
(*at x within X*)
apply (*simp add*: *part-circlepath-def linepath-def scaleR-conv-of-real*)
apply (*rule has-vector-derivative-real-complex*)
apply (*rule derivative-eq-intros* | *simp*)+
done

corollary *vector-derivative-part-circlepath*:
vector-derivative (*part-circlepath z r s t*) (*at x*) =
ii ∗ *r* ∗ (*of-real t* − *of-real s*) ∗ *exp*(*ii* ∗ *linepath s t x*)
using *has-vector-derivative-part-circlepath vector-derivative-at* **by** *blast*

corollary *vector-derivative-part-circlepath01*:
⟦*0* ≤ *x*; *x* ≤ *1*⟧
⟹ *vector-derivative* (*part-circlepath z r s t*) (*at x within* {*0..1*}) =
ii ∗ *r* ∗ (*of-real t* − *of-real s*) ∗ *exp*(*ii* ∗ *linepath s t x*)
using *has-vector-derivative-part-circlepath*
by (*auto simp*: *vector-derivative-at-within-ivl*)

lemma *valid-path-part-circlepath* [*simp*]: *valid-path* (*part-circlepath z r s t*)
apply (*simp add*: *valid-path-def*)
apply (*rule C1-differentiable-imp-piecewise*)
apply (*auto simp*: *C1-differentiable-on-eq vector-derivative-works vector-derivative-part-circlepath has-vector-derivative-part-circlepath*
intro!: *continuous-intros*)
done

lemma *path-part-circlepath* [*simp*]: *path* (*part-circlepath z r s t*)
by (*simp add*: *valid-path-imp-path*)

proposition *path-image-part-circlepath*:
assumes *s* ≤ *t*
shows *path-image* (*part-circlepath z r s t*) = {*z* + *r* ∗ *exp*(*ii* ∗ *of-real x*) | *x*. *s* ≤ *x* ∧ *x* ≤ *t*}
proof −
{ fix *z*::*real*
assume *0* ≤ *z z* ≤ *1*
with ‹*s* ≤ *t*› **have** ∃ *x*. (*exp* (i ∗ *linepath s t z*) = *exp* (i ∗ *of-real x*)) ∧ *s* ≤ *x* ∧ *x* ≤ *t*
apply (*rule-tac x*=(*1* − *z*) ∗ *s* + *z* ∗ *t* **in** *exI*)
apply (*simp add*: *linepath-def scaleR-conv-of-real algebra-simps*)
apply (*rule conjI*)
using *mult-right-mono* **apply** *blast*
using *affine-ineq* **by** (*metis mult.commute*)
}

```
  moreover
  { fix z
    assume s ≤ z z ≤ t
     then have z + of-real r * exp (i * of-real z) ∈ (λx. z + of-real r * exp (i *
linepath s t x)) ‘ {0..1}
      apply (rule-tac x=(z − s)/(t − s) in image-eqI)
      apply (simp add: linepath-def scaleR-conv-of-real divide-simps exp-eq)
      apply (auto simp: algebra-simps divide-simps)
      done
  }
  ultimately show ?thesis
    by (fastforce simp add: path-image-def part-circlepath-def)
qed

corollary path-image-part-circlepath-subset:
    ⟦s ≤ t; 0 ≤ r⟧ ⟹ path-image(part-circlepath z r s t) ⊆ sphere z r
by (auto simp: path-image-part-circlepath sphere-def dist-norm algebra-simps norm-mult)

proposition in-path-image-part-circlepath:
  assumes w ∈ path-image(part-circlepath z r s t) s ≤ t 0 ≤ r
    shows norm(w − z) = r
proof −
  have w ∈ {c. dist z c = r}
   by (metis (no-types) path-image-part-circlepath-subset sphere-def subset-eq assms)
  thus ?thesis
    by (simp add: dist-norm norm-minus-commute)
qed

proposition finite-bounded-log: finite {z::complex. norm z ≤ b ∧ exp z = w}
proof (cases w = 0)
  case True then show ?thesis by auto
next
  case False
  have ∗: finite {x. cmod (complex-of-real (2 * real-of-int x * pi) * i) ≤ b + cmod
(Ln w)}
    apply (simp add: norm-mult finite-int-iff-bounded-le)
    apply (rule-tac x=⌊(b + cmod (Ln w)) / (2*pi)⌋ in exI)
    apply (auto simp: divide-simps le-floor-iff)
    done
  have [simp]: ⋀P f. {z. P z ∧ (∃ n. z = f n)} = f ‘ {n. P (f n)}
    by blast
  show ?thesis
    apply (subst exp-Ln [OF False, symmetric])
    apply (simp add: exp-eq)
    using norm-add-leD apply (fastforce intro: finite-subset [OF - ∗])
    done
qed

lemma finite-bounded-log2:
```

```
  fixes a::complex
    assumes a ≠ 0
    shows finite {z. norm z ≤ b ∧ exp(a*z) = w}
proof −
  have *: finite ((λz. z / a) ` {z. cmod z ≤ b * cmod a ∧ exp z = w})
    by (rule finite-imageI [OF finite-bounded-log])
  show ?thesis
    by (rule finite-subset [OF - *]) (force simp: assms norm-mult)
qed

proposition has-contour-integral-bound-part-circlepath-strong:
  assumes fi: (f has-contour-integral i) (part-circlepath z r s t)
      and finite k and le: 0 ≤ B 0 < r s ≤ t
      and B: ⋀x. x ∈ path-image(part-circlepath z r s t) − k ⟹ norm(f x) ≤ B
    shows cmod i ≤ B * r * (t − s)
proof −
  consider s = t | s < t using ‹s ≤ t› by linarith
  then show ?thesis
  proof cases
    case 1 with fi [unfolded has-contour-integral]
    have i = 0  by (simp add: vector-derivative-part-circlepath)
    with assms show ?thesis by simp
  next
    case 2
    have [simp]: |r| = r using ‹r > 0› by linarith
    have [simp]: cmod (complex-of-real t − complex-of-real s) = t−s
      by (metis 2 abs-of-pos diff-gt-0-iff-gt norm-of-real of-real-diff)
    have finite (part-circlepath z r s t −` {y} ∩ {0..1}) if y ∈ k for y
    proof −
      def w ≡ (y − z)/of-real r / exp(ii * of-real s)
      have fin: finite (of-real −` {z. cmod z ≤ 1 ∧ exp (i * complex-of-real (t −
s) * z) = w})
        apply (rule finite-vimageI [OF finite-bounded-log2])
        using ‹s < t› apply (auto simp: inj-of-real)
        done
      show ?thesis
        apply (simp add: part-circlepath-def linepath-def vimage-def)
        apply (rule finite-subset [OF - fin])
        using le
        apply (auto simp: w-def algebra-simps scaleR-conv-of-real exp-add exp-diff)
        done
    qed
    then have fin01: finite ((part-circlepath z r s t) −` k ∩ {0..1})
      by (rule finite-finite-vimage-IntI [OF ‹finite k›])
    have **: ((λx. if (part-circlepath z r s t x) ∈ k then 0
                   else f(part-circlepath z r s t x) *
                      vector-derivative (part-circlepath z r s t) (at x)) has-integral i)
{0..1}
      apply (rule has-integral-spike
```

[**where** $f = \lambda x.\ f(\textit{part-circlepath}\ z\ r\ s\ t\ x) * \textit{vector-derivative}\ (\textit{part-circlepath}\ z\ r\ s\ t)\ (\textit{at}\ x)$])
apply (*rule negligible-finite* [*OF fin01*])
using *fi has-contour-integral*
apply *auto*
done
have *∗*: $\bigwedge x.\ [\![0 \le x;\ x \le 1;\ \textit{part-circlepath}\ z\ r\ s\ t\ x \notin k]\!] \Longrightarrow \textit{cmod}\ (f\ (\textit{part-circlepath}\ z\ r\ s\ t\ x)) \le B$
by (*auto intro*!: *B* [*unfolded path-image-def image-def*, *simplified*])
show *?thesis*
apply (*rule has-integral-bound* [**where** *'a=real*, *simplified*, *OF - ∗∗*, *simplified*])
using *assms* **apply** *force*
apply (*simp add*: *norm-mult vector-derivative-part-circlepath*)
using *le ∗ 2* ‹$r > 0$› **by** *auto*
qed
qed

corollary *has-contour-integral-bound-part-circlepath*:
$[\![(f\ \textit{has-contour-integral}\ i)\ (\textit{part-circlepath}\ z\ r\ s\ t);$
$0 \le B;\ 0 < r;\ s \le t;$
$\bigwedge x.\ x \in \textit{path-image}(\textit{part-circlepath}\ z\ r\ s\ t) \Longrightarrow \textit{norm}(f\ x) \le B]\!]$
$\Longrightarrow \textit{norm}\ i \le B{*}r{*}(t - s)$
by (*auto intro*: *has-contour-integral-bound-part-circlepath-strong*)

proposition *contour-integrable-continuous-part-circlepath*:
continuous-on (*path-image* (*part-circlepath z r s t*)) *f*
$\Longrightarrow$ *f contour-integrable-on* (*part-circlepath z r s t*)
apply (*simp add*: *contour-integrable-on has-contour-integral-def vector-derivative-part-circlepath path-image-def*)
apply (*rule integrable-continuous-real*)
apply (*fast intro*: *path-part-circlepath* [*unfolded path-def*] *continuous-intros continuous-on-compose2* [**where** *g=f*, *OF - - order-refl*])
done

proposition *winding-number-part-circlepath-pos-less*:
assumes $s < t$ **and** *no*: $\textit{norm}(w - z) < r$
shows $0 < \textit{Re}\ (\textit{winding-number}(\textit{part-circlepath}\ z\ r\ s\ t)\ w)$
proof −
have $0 < r$ **by** (*meson no norm-not-less-zero not-le order.strict-trans2*)
note *valid-path-part-circlepath*
moreover have $w \notin \textit{path-image}\ (\textit{part-circlepath}\ z\ r\ s\ t)$
using *assms* **by** (*auto simp*: *path-image-def image-def part-circlepath-def norm-mult linepath-def*)
moreover have $0 < r * (t - s) * (r - \textit{cmod}\ (w - z))$
using *assms* **by** (*metis* ‹$0 < r$› *diff-gt-0-iff-gt mult-pos-pos*)
ultimately show *?thesis*
apply (*rule winding-number-pos-lt* [**where** $e = r{*}(t - s){*}(r - \textit{norm}(w - z))$])
apply (*simp add*: *vector-derivative-part-circlepath right-diff-distrib* [*symmetric*] *mult-ac*)

apply (*rule mult-left-mono*)+
using *Re-Im-le-cmod* [*of w−z linepath s t x* **for** *x*]
apply (*simp add*: *exp-Euler cos-of-real sin-of-real part-circlepath-def algebra-simps cos-squared-eq* [*unfolded power2-eq-square*])
using *assms* ‹*0 < r*› **by** *auto*
qed

proposition *simple-path-part-circlepath*:
simple-path(*part-circlepath z r s t*) ⟷ (*r* ≠ *0* ∧ *s* ≠ *t* ∧ |*s* − *t*| ≤ *2*∗*pi*)
proof (*cases r* = *0* ∨ *s* = *t*)
case *True*
then show *?thesis*
apply (*rule disjE*)
apply (*force simp*: *part-circlepath-def simple-path-def intro*: *bexI* [**where** *x* = *1/4*] *bexI* [**where** *x* = *1/3*])+
done
next
case *False* **then have** *r* ≠ *0 s* ≠ *t* **by** *auto*
have ∗: ⋀*x y z s t*. *ii*∗((*1* − *x*) ∗ *s* + *x* ∗ *t*) = *ii*∗(((*1* − *y*) ∗ *s* + *y* ∗ *t*)) + *z* ⟷ *ii*∗(*x* − *y*) ∗ (*t* − *s*) = *z*
by (*simp add*: *algebra-simps*)
have *abs01*: ⋀*x y*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1*
⟹ (*x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0* ⟷ |*x* − *y*| ∈ {*0*,*1*})
by *auto*
have *abs-away*: ⋀*P*. (∀ *x*∈{*0*..*1*}. ∀ *y*∈{*0*..*1*}. *P* |*x* − *y*|) ⟷ (∀ *x*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ⟶ *P x*)
by *force*
have ∗∗: ⋀*x y*. (∃ *n*. (*complex-of-real x* − *of-real y*) ∗ (*of-real t* − *of-real s*) = *2* ∗ (*of-int n* ∗ *of-real pi*)) ⟷
(∃ *n*. |*x* − *y*| ∗ (*t* − *s*) = *2* ∗ (*of-int n* ∗ *pi*))
by (*force simp*: *algebra-simps abs-if dest*: *arg-cong* [**where** *f*=*Re*] *arg-cong* [**where** *f*=*complex-of-real*]
intro: *exI* [**where** *x* = −*n* **for** *n*])
have *1*: ∀ *x*. *0* ≤ *x* ∧ *x* ≤ *1* ⟶ (∃ *n*. *x* ∗ (*t* − *s*) = *2* ∗ (*real-of-int n* ∗ *pi*)) ⟶ *x* = *0* ∨ *x* = *1* ⟹ |*s* − *t*| ≤ *2* ∗ *pi*
apply (*rule ccontr*)
apply (*drule-tac x*=*2*∗*pi* / |*t* − *s*| **in** *spec*)
using *False*
apply (*simp add*: *abs-minus-commute divide-simps*)
apply (*frule-tac x*=*1* **in** *spec*)
apply (*drule-tac x*=−*1* **in** *spec*)
apply (*simp add*:)
done
have *2*: |*s* − *t*| = |*2* ∗ (*real-of-int n* ∗ *pi*) / *x*| **if** *x* ≠ *0 x* ∗ (*t* − *s*) = *2* ∗ (*real-of-int n* ∗ *pi*) **for** *x n*
proof −
have *t*−*s* = *2* ∗ (*real-of-int n* ∗ *pi*)/*x*
using *that* **by** (*simp add*: *field-simps*)

```
    then show ?thesis by (metis abs-minus-commute)
  qed
  show ?thesis using False
    apply (simp add: simple-path-def path-part-circlepath)
     apply (simp add: part-circlepath-def linepath-def exp-eq  * ** abs01  del:
Set.insert-iff)
    apply (subst abs-away)
    apply (auto simp: 1)
    apply (rule ccontr)
    apply (auto simp: 2 divide-simps abs-mult dest: of-int-leD)
    done
qed

proposition arc-part-circlepath:
  assumes r ≠ 0 s ≠ t |s − t| < 2*pi
    shows arc (part-circlepath z r s t)
proof −
  have *: x = y if eq: i * (linepath s t x) = i * (linepath s t y) + 2 * of-int n *
complex-of-real pi * i
                and x: x ∈ {0..1} and y: y ∈ {0..1} for x y n
    proof −
      have (linepath s t x) = (linepath s t y) + 2 * of-int n * complex-of-real pi
      by (metis add-divide-eq-iff complex-i-not-zero mult.commute nonzero-mult-divide-cancel-left
eq)
      then have s*y + t*x = s*x + (t*y + of-int n * (pi * 2))
        by (force simp: algebra-simps linepath-def dest: arg-cong [where f=Re])
      then have st: x ≠ y ⟹ (s−t) = (of-int n * (pi * 2) / (y−x))
        by (force simp: field-simps)
      show ?thesis
        apply (rule ccontr)
        using assms x y
        apply (simp add: st abs-mult field-simps)
        using st
        apply (auto simp: dest: of-int-lessD)
        done
    qed
  show ?thesis
    using assms
    apply (simp add: arc-def)
    apply (simp add: part-circlepath-def inj-on-def exp-eq)
    apply (blast intro: *)
    done
qed
```

52.33 Special case of one complete circle

```
definition circlepath :: [complex, real, real] ⇒ complex
  where circlepath z r ≡ part-circlepath z r 0 (2*pi)
```

lemma *circlepath*: *circlepath z r* = (λ*x*. *z* + *r* ∗ *exp*(*2* ∗ *of-real pi* ∗ *ii* ∗ *of-real x*))
by (*simp add*: *circlepath-def part-circlepath-def linepath-def algebra-simps*)

lemma *pathstart-circlepath* [*simp*]: *pathstart* (*circlepath z r*) = *z* + *r*
by (*simp add*: *circlepath-def*)

lemma *pathfinish-circlepath* [*simp*]: *pathfinish* (*circlepath z r*) = *z* + *r*
by (*simp add*: *circlepath-def*) (*metis exp-two-pi-i mult.commute*)

lemma *circlepath-minus*: *circlepath z* (−*r*) *x* = *circlepath z r* (*x* + *1/2*)
proof −
have *z* + *of-real r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ (*x* + *1* / *2*)) =
z + *of-real r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ *x* + *pi* ∗ i)
by (*simp add*: *divide-simps*) (*simp add*: *algebra-simps*)
also have ... = *z* − *r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ *x*)
by (*simp add*: *exp-add*)
finally show *?thesis*
by (*simp add*: *circlepath path-image-def sphere-def dist-norm*)
qed

lemma *circlepath-add1*: *circlepath z r* (*x*+*1*) = *circlepath z r x*
using *circlepath-minus* [*of z r x*+*1/2*] *circlepath-minus* [*of z* −*r x*]
by (*simp add*: *add.commute*)

lemma *circlepath-add-half*: *circlepath z r* (*x* + *1/2*) = *circlepath z r* (*x* − *1/2*)
using *circlepath-add1* [*of z r x*−*1/2*]
by (*simp add*: *add.commute*)

lemma *path-image-circlepath-minus-subset*:
path-image (*circlepath z* (−*r*)) ⊆ *path-image* (*circlepath z r*)
apply (*simp add*: *path-image-def image-def circlepath-minus*, *clarify*)
apply (*case-tac xa* ≤ *1/2*, *force*)
apply (*force simp add*: *circlepath-add-half*)+
done

lemma *path-image-circlepath-minus*: *path-image* (*circlepath z* (−*r*)) = *path-image* (*circlepath z r*)
using *path-image-circlepath-minus-subset* **by** *fastforce*

proposition *has-vector-derivative-circlepath* [*derivative-intros*]:
((*circlepath z r*) *has-vector-derivative* (*2* ∗ *pi* ∗ *ii* ∗ *r* ∗ *exp* (*2* ∗ *of-real pi* ∗ *ii* ∗ *of-real x*)))
(*at x within X*)
apply (*simp add*: *circlepath-def scaleR-conv-of-real*)
apply (*rule derivative-eq-intros*)
apply (*simp add*: *algebra-simps*)
done

corollary *vector-derivative-circlepath*:
vector-derivative (*circlepath z r*) (*at x*) =
*2 * pi * ii * r * exp*(*2 * of-real pi * ii * x*)
using *has-vector-derivative-circlepath vector-derivative-at* **by** *blast*

corollary *vector-derivative-circlepath01*:
⟦*0 ≤ x*; *x ≤ 1*⟧
⟹ *vector-derivative* (*circlepath z r*) (*at x within* {*0..1*}) =
*2 * pi * ii * r * exp*(*2 * of-real pi * ii * x*)
using *has-vector-derivative-circlepath*
by (*auto simp*: *vector-derivative-at-within-ivl*)

lemma *valid-path-circlepath* [*simp*]: *valid-path* (*circlepath z r*)
by (*simp add*: *circlepath-def*)

lemma *path-circlepath* [*simp*]: *path* (*circlepath z r*)
by (*simp add*: *valid-path-imp-path*)

lemma *path-image-circlepath-nonneg*:
assumes *0 ≤ r* **shows** *path-image* (*circlepath z r*) = *sphere z r*
proof −
have ∗: *x* ∈ (λ*u*. *z* + (*cmod* (*x* − *z*)) ∗ *exp* (i ∗ (*of-real u* ∗ (*of-real pi* ∗ *2*)))) ‘ {*0..1*} **for** *x*
proof (*cases x* = *z*)
case *True* **then show** *?thesis* **by** *force*
next
case *False*
def *w* ≡ *x* − *z*
then have *w* ≠ *0* **by** (*simp add*: *False*)
have ∗∗: ⋀*t*. ⟦*Re w* = *cos t* ∗ *cmod w*; *Im w* = *sin t* ∗ *cmod w*⟧ ⟹ *w* = *of-real* (*cmod w*) ∗ *exp* (i ∗ *t*)
using *cis-conv-exp complex-eq-iff* **by** *auto*
show *?thesis*
apply (*rule sincos-total-2pi* [*of Re*(*w*/*of-real*(*norm w*)) *Im*(*w*/*of-real*(*norm w*))])
apply (*simp add*: *divide-simps* ‹*w* ≠ *0*› *cmod-power2* [*symmetric*])
apply (*rule-tac x*=*t* / (*2*∗*pi*) **in** *image-eqI*)
apply (*simp add*: *divide-simps* ‹*w* ≠ *0*›)
using *False* ∗∗
apply (*auto simp*: *w-def*)
done
qed
show *?thesis*
unfolding *circlepath path-image-def sphere-def dist-norm*
by (*force simp*: *assms algebra-simps norm-mult norm-minus-commute intro*: ∗)
qed

proposition *path-image-circlepath* [*simp*]:
path-image (*circlepath z r*) = *sphere z* |*r*|

using *path-image-circlepath-minus*
by (*force simp add*: *path-image-circlepath-nonneg abs-if*)

lemma *has-contour-integral-bound-circlepath-strong*:
⟦(*f has-contour-integral i*) (*circlepath z r*);
finite k; $0 \le B$; $0 < r$;
$\bigwedge x.$ ⟦$norm(x - z) = r$; $x \notin k$⟧ $\Longrightarrow$ $norm(f\ x) \le B$⟧
$\Longrightarrow$ $norm\ i \le B*(2*pi*r)$
unfolding *circlepath-def*
by (*auto simp*: *algebra-simps in-path-image-part-circlepath dest*!: *has-contour-integral-bound-part-circlepath-st*

corollary *has-contour-integral-bound-circlepath*:
⟦(*f has-contour-integral i*) (*circlepath z r*);
$0 \le B$; $0 < r$; $\bigwedge x.$ $norm(x - z) = r \Longrightarrow norm(f\ x) \le B$⟧
$\Longrightarrow$ $norm\ i \le B*(2*pi*r)$
by (*auto intro*: *has-contour-integral-bound-circlepath-strong*)

proposition *contour-integrable-continuous-circlepath*:
continuous-on (*path-image* (*circlepath z r*)) *f*
$\Longrightarrow$ *f contour-integrable-on* (*circlepath z r*)
by (*simp add*: *circlepath-def contour-integrable-continuous-part-circlepath*)

lemma *simple-path-circlepath*: *simple-path*(*circlepath z r*) $\longleftrightarrow$ ($r \neq 0$)
by (*simp add*: *circlepath-def simple-path-part-circlepath*)

lemma *notin-path-image-circlepath* [*simp*]: *cmod* ($w - z$) $< r \Longrightarrow w \notin$ *path-image* (*circlepath z r*)
by (*simp add*: *sphere-def dist-norm norm-minus-commute*)

proposition *contour-integral-circlepath*:
$0 < r \Longrightarrow$ *contour-integral* (*circlepath z r*) ($\lambda w.$ *1 / (w - z)*) = *2 * complex-of-real pi ** i
apply (*rule contour-integral-unique*)
apply (*simp add*: *has-contour-integral-def*)
apply (*subst has-integral-cong*)
apply (*simp add*: *vector-derivative-circlepath01*)
using *has-integral-const-real* [*of - 0 1*]
apply (*force simp*: *circlepath*)
done

lemma *winding-number-circlepath-centre*: $0 < r \Longrightarrow$ *winding-number* (*circlepath z r*) *z = 1*
apply (*rule winding-number-unique-loop*)
apply (*simp-all add*: *sphere-def valid-path-imp-path*)
apply (*rule-tac x=circlepath z r* **in** *exI*)
apply (*simp add*: *sphere-def contour-integral-circlepath*)
done

proposition *winding-number-circlepath*:

assumes *norm*$(w - z) < r$ **shows** *winding-number*(*circlepath z r*) $w = 1$
proof (*cases* $w = z$)
case *True* **then show** *?thesis*
using *assms winding-number-circlepath-centre* **by** *auto*
next
case *False*
have [*simp*]: $r > 0$
using *assms le-less-trans norm-ge-zero* **by** *blast*
def $r' \equiv$ *norm*$(w - z)$
have $r' < r$
by (*simp add*: *assms r'-def*)
have *disjo*: *cball z r'* $\cap$ *sphere z r* $= \{\}$
using ‹$r' < r$› **by** (*force simp*: *cball-def sphere-def*)
have *winding-number*(*circlepath z r*) $w =$ *winding-number*(*circlepath z r*) z
apply (*rule winding-number-around-inside* [**where** $s =$ *cball z r'*])
apply (*simp-all add*: *disjo order.strict-implies-order winding-number-circlepath-centre*)
apply (*simp-all add*: *False r'-def dist-norm norm-minus-commute*)
done
also have ... $= 1$
by (*simp add*: *winding-number-circlepath-centre*)
finally show *?thesis* **.**
qed

Hence the Cauchy formula for points inside a circle.

theorem *Cauchy-integral-circlepath*:
assumes *continuous-on* (*cball z r*) *f f holomorphic-on* (*ball z r*) *norm*$(w - z) <$
r
shows ((λu. *f u*/(*u* − *w*)) *has-contour-integral* (2 ∗ *of-real pi* ∗ *ii* ∗ *f w*))
(*circlepath z r*)
proof −
have $r > 0$
using *assms le-less-trans norm-ge-zero* **by** *blast*
have ((λu. *f u* / (*u* − *w*)) *has-contour-integral* (2 ∗ *pi*) ∗ i ∗ *winding-number*
(*circlepath z r*) *w* ∗ *f w*)
(*circlepath z r*)
apply (*rule Cauchy-integral-formula-weak* [**where** $s =$ *cball z r* **and** $k = \{\}$])
using *assms* ‹$r > 0$›
apply (*simp-all add*: *dist-norm norm-minus-commute*)
apply (*metis at-within-interior dist-norm holomorphic-on-def interior-ball mem-ball*
norm-minus-commute)
apply (*simp add*: *cball-def sphere-def dist-norm*, *clarify*)
apply (*simp add*:)
by (*metis dist-commute dist-norm less-irrefl*)
then show *?thesis*
by (*simp add*: *winding-number-circlepath assms*)
qed

corollary *Cauchy-integral-circlepath-simple*:
assumes *f holomorphic-on cball z r norm*$(w - z) < r$

shows $((\lambda u.\ f\ u/(u - w))$ *has-contour-integral* $(2 * \textit{of-real}\ \textit{pi} * \textit{ii} * f\ w))$
$(\textit{circlepath}\ z\ r)$
using *assms* **by** (*force simp*: *holomorphic-on-imp-continuous-on holomorphic-on-subset Cauchy-integral-circlepath*)

lemma *no-bounded-connected-component-imp-winding-number-zero*:
assumes *g*: *path g path-image g* $\subseteq$ *s pathfinish g* = *pathstart g z* $\notin$ *s*
and *nb*: $\bigwedge z.$ *bounded* (*connected-component-set* $(-\ s)\ z) \longrightarrow z \in s$
shows *winding-number g z* = *0*
apply (*rule winding-number-zero-in-outside*)
apply (*simp-all add*: *assms*)
by (*metis nb* [*of z*] ‹*path-image g* $\subseteq$ *s*› ‹*z* $\notin$ *s*› *contra-subsetD mem-Collect-eq outside outside-mono*)

lemma *no-bounded-path-component-imp-winding-number-zero*:
assumes *g*: *path g path-image g* $\subseteq$ *s pathfinish g* = *pathstart g z* $\notin$ *s*
and *nb*: $\bigwedge z.$ *bounded* (*path-component-set* $(-\ s)\ z) \longrightarrow z \in s$
shows *winding-number g z* = *0*
apply (*rule no-bounded-connected-component-imp-winding-number-zero* [*OF g*])
by (*simp add*: *bounded-subset nb path-component-subset-connected-component*)

52.34 Uniform convergence of path integral

Uniform convergence when the derivative of the path is bounded, and in particular for the special case of a circle.

proposition *contour-integral-uniform-limit*:
assumes *ev-fint*: *eventually* $(\lambda n::'a.\ (f\ n)$ *contour-integrable-on* $\gamma)\ F$
and *ev-no*: $\bigwedge e.\ 0 < e \Longrightarrow$ *eventually* $(\lambda n.\ \forall x \in$ *path-image* $\gamma.\ norm(f\ n\ x - l\ x) < e)\ F$
and *noleB*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow norm\ (\textit{vector-derivative}\ \gamma\ (\textit{at}\ t)) \leq B$
and γ: *valid-path* γ
and [*simp*]: $\sim$ (*trivial-limit F*)
shows *l contour-integrable-on* γ $((\lambda n.\ \textit{contour-integral}\ \gamma\ (f\ n)) \longrightarrow \textit{contour-integral}\ \gamma\ l)\ F$
proof −
have *0* $\leq$ *B* **by** (*meson noleB* [*of 0*] *atLeastAtMost-iff norm-ge-zero order-refl order-trans zero-le-one*)
{ fix *e*::*real*
assume *0* < *e*
then have *eB*: *0* < *e* / (|*B*| + *1*) **by** *simp*
obtain *a* **where** *fga*: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow cmod\ (f\ a\ (\gamma\ x) - l\ (\gamma\ x)) < e\ /\ (|B| + 1)$
and *inta*: $(\lambda t.\ f\ a\ (\gamma\ t) * \textit{vector-derivative}\ \gamma\ (\textit{at}\ t))$ *integrable-on* $\{0..1\}$
using *eventually-happens* [*OF eventually-conj* [*OF ev-no* [*OF eB*] *ev-fint*]]
by (*fastforce simp*: *contour-integrable-on path-image-def*)
have *Ble*: *B* * *e* / (|*B*| + *1*) $\leq$ *e*
using ‹*0* $\leq$ *B*› ‹*0* < *e*› **by** (*simp add*: *divide-simps*)

have $\exists h.\ (\forall x{\in}\{0..1\}.\ cmod\ (l\ (\gamma\ x) * vector\text{-}derivative\ \gamma\ (at\ x) - h\ x) \leq e) \wedge h\ integrable\text{-}on\ \{0..1\}$
apply (*rule-tac* $x{=}\lambda x.\ f\ (a{::}'a)\ (\gamma\ x) * vector\text{-}derivative\ \gamma\ (at\ x)$ **in** *exI*)
apply (*intro inta conjI ballI*)
apply (*rule order-trans* [*OF - Ble*])
apply (*frule noleB*)
apply (*frule fga*)
using ‹$0 \leq B$› ‹$0 < e$›
apply (*simp add*: *norm-mult left-diff-distrib* [*symmetric*] *norm-minus-commute divide-simps*)
apply (*drule* (*1*) *mult-mono* [*OF less-imp-le*])
apply (*simp-all add*: *mult-ac*)
done
}
then show *lintg*: *l contour-integrable-on* γ
apply (*simp add*: *contour-integrable-on*)
apply (*blast intro*: *integrable-uniform-limit-real*)
done
{ fix *e*::*real*
def $B' \equiv B{+}1$
have *B'*: $B' > 0\ B' > B$ **using** ‹$0 \leq B$› **by** (*auto simp*: *B'-def*)
assume $0 < e$
then have *ev-no'*: $\forall_F\ n\ in\ F.\ \forall x{\in}path\text{-}image\ \gamma.\ 2 * cmod\ (f\ n\ x - l\ x) < e\ /\ B'$
using *ev-no* [*of e / B' / 2*] *B'* **by** (*simp add*: *field-simps*)
have *ie*: *integral* $\{0..1{::}real\}$ $(\lambda x.\ e\ /\ 2) < e$ **using** ‹$0 < e$› **by** *simp*
have $*$: *cmod* $(f\ x\ (\gamma\ t) * vector\text{-}derivative\ \gamma\ (at\ t) - l\ (\gamma\ t) * vector\text{-}derivative\ \gamma\ (at\ t)) \leq e\ /\ 2$
if *t*: $t{\in}\{0..1\}$ **and** *leB'*: $2 * cmod\ (f\ x\ (\gamma\ t) - l\ (\gamma\ t)) < e\ /\ B'$ **for** *x t*
proof −
have $2 * cmod\ (f\ x\ (\gamma\ t) - l\ (\gamma\ t)) * cmod\ (vector\text{-}derivative\ \gamma\ (at\ t)) \leq e * (B/\ B')$
using *mult-mono* [*OF less-imp-le* [*OF leB'*] *noleB*] *B'* ‹$0 < e$› *t* **by** *auto*
also have ... $< e$
by (*simp add*: *B'* ‹$0 < e$› *mult-imp-div-pos-less*)
finally have $2 * cmod\ (f\ x\ (\gamma\ t) - l\ (\gamma\ t)) * cmod\ (vector\text{-}derivative\ \gamma\ (at\ t)) < e$ **.**
then show *?thesis*
by (*simp add*: *left-diff-distrib* [*symmetric*] *norm-mult*)
qed
have $\forall_F\ x\ in\ F.\ dist\ (contour\text{-}integral\ \gamma\ (f\ x))\ (contour\text{-}integral\ \gamma\ l) < e$
apply (*rule eventually-mono* [*OF eventually-conj* [*OF ev-no' ev-fint*]])
apply (*simp add*: *dist-norm contour-integrable-on path-image-def contour-integral-integral*)
apply (*simp add*: *lintg integral-diff* [*symmetric*] *contour-integrable-on* [*symmetric*], *clarify*)
apply (*rule le-less-trans* [*OF integral-norm-bound-integral ie*])
apply (*simp add*: *lintg integrable-diff contour-integrable-on* [*symmetric*])
apply (*blast intro*: $*$)+
done

}
then show $((\lambda n.\ \textit{contour-integral}\ \gamma\ (f\ n)) \longrightarrow \textit{contour-integral}\ \gamma\ l)\ F$
by (*rule tendstoI*)
qed

proposition *contour-integral-uniform-limit-circlepath*:
assumes *ev-fint*: *eventually* $(\lambda n{::}'a.\ (f\ n)\ \textit{contour-integrable-on}\ (\textit{circlepath}\ z\ r))$ *F*
and *ev-no*: $\bigwedge e.\ 0 < e \Longrightarrow$ *eventually* $(\lambda n.\ \forall x \in \textit{path-image}\ (\textit{circlepath}\ z\ r).\ \textit{norm}(f\ n\ x - l\ x) < e)\ F$
and [*simp*]: $\sim$ $(\textit{trivial-limit}\ F)\ 0 < r$
shows $l\ \textit{contour-integrable-on}\ (\textit{circlepath}\ z\ r)$ $((\lambda n.\ \textit{contour-integral}\ (\textit{circlepath}\ z\ r)\ (f\ n)) \longrightarrow \textit{contour-integral}\ (\textit{circlepath}\ z\ r)\ l)\ F$
by (*auto simp*: *vector-derivative-circlepath norm-mult intro*: *contour-integral-uniform-limit assms*)

52.35 General stepping result for derivative formulas.

lemma *sum-sqs-eq*:
fixes $x{::}'a{::}\textit{idom}$ **shows** $x * x + y * y = x * (y * 2) \Longrightarrow y = x$
by *algebra*

proposition *Cauchy-next-derivative*:
assumes *continuous-on* $(\textit{path-image}\ \gamma)\ f'$
and *leB*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow \textit{norm}\ (\textit{vector-derivative}\ \gamma\ (\textit{at}\ t)) \le B$
and *int*: $\bigwedge w.\ w \in s - \textit{path-image}\ \gamma \Longrightarrow ((\lambda u.\ f'\ u\ /\ (u - w)\ \hat{}\ k)\ \textit{has-contour-integral}\ f\ w)\ \gamma$
and *k*: $k \neq 0$
and *open s*
and γ: *valid-path* γ
and *w*: $w \in s - \textit{path-image}\ \gamma$
shows $(\lambda u.\ f'\ u\ /\ (u - w)\ \hat{}\ (\textit{Suc}\ k))\ \textit{contour-integrable-on}\ \gamma$
and $(f\ \textit{has-field-derivative}\ (k * \textit{contour-integral}\ \gamma\ (\lambda u.\ f'\ u/(u - w)\ \hat{}\ (\textit{Suc}\ k))))$
$(\textit{at}\ w)$ (**is** *?thes2*)
proof −
have *open* $(s - \textit{path-image}\ \gamma)$ **using** ‹*open s*› *closed-valid-path-image* γ **by** *blast*
then obtain *d* **where** *d*>*0* **and** *d*: *ball w d* $\subseteq s - \textit{path-image}\ \gamma$ **using** *w*
using *open-contains-ball* **by** *blast*
have [*simp*]: $\bigwedge n.\ \textit{cmod}\ (1 + \textit{of-nat}\ n) = 1 + \textit{of-nat}\ n$
by (*metis norm-of-nat of-nat-Suc*)
have *1*: $\forall_F\ n\ \textit{in at}\ w.\ (\lambda x.\ f'\ x * (\textit{inverse}\ (x - n)\ \hat{}\ k - \textit{inverse}\ (x - w)\ \hat{}\ k) / (n - w) / \textit{of-nat}\ k)$
$\textit{contour-integrable-on}\ \gamma$
apply (*simp add*: *eventually-at*)
apply (*rule-tac x*=*d* **in** *exI*)
apply (*simp add*: ‹*d* > *0*› *dist-norm field-simps*, *clarify*)
apply (*rule contour-integrable-div* [*OF contour-integrable-diff*])
using *int w d*

apply (*force simp*: *dist-norm norm-minus-commute intro*!: *has-contour-integral-integrable*)+
done
have *bim-g*: *bounded* (*image f'* (*path-image* γ))
by (*simp add*: *compact-imp-bounded compact-continuous-image compact-valid-path-image assms*)
then obtain *C* **where** $C > 0$ **and** *C*: $\bigwedge x.\ [\![0 \le x; x \le 1]\!] \Longrightarrow$ *cmod* (*f'* (γ *x*)) $\le C$
by (*force simp*: *bounded-pos path-image-def*)
have *twom*: $\forall_F$ *n in at w*.
$\forall x \in$*path-image* γ.
cmod ((*inverse* (*x* − *n*) ^ *k* − *inverse* (*x* − *w*) ^ *k*) / (*n* − *w*) / *k* − *inverse* (*x* − *w*) ^ *Suc k*) < *e*
if 0 < *e* **for** *e*
proof −
have ∗: *cmod* ((*inverse* (*x* − *u*) ^ *k* − *inverse* (*x* − *w*) ^ *k*) / ((*u* − *w*) ∗ *k*) − *inverse* (*x* − *w*) ^ *Suc k*) < *e*
if *x*: *x* ∈ *path-image* γ **and** *u* ≠ *w* **and** *uwd*: *cmod* (*u* − *w*) < *d*/2
and *uw-less*: *cmod* (*u* − *w*) < *e* ∗ (*d* / 2) ^ (*k*+2) / (1 + *real k*)
for *u x*
proof −
def *ff* ≡ λ*n*::*nat*. λ*w*. *if n* = 0 *then inverse*(*x* − *w*)^*k*
else if n = 1 *then k* / (*x* − *w*)^(*Suc k*)
else (*k* ∗ *of-real*(*Suc k*)) / (*x* − *w*)^(*k* + 2)
have *km1*: $\bigwedge z$::*complex*. *z* ≠ 0 ⟹ *z* ^ (*k* − *Suc* 0) = *z* ^ *k* / *z*
by (*simp add*: *field-simps*) (*metis Suc-pred* ‹*k* ≠ 0› *neq0-conv power-Suc*)
have *ff1*: (*ff i has-field-derivative ff* (*Suc i*) *z*) (*at z within ball w* (*d* / 2))
if *z* ∈ *ball w* (*d* / 2) *i* ≤ 1 **for** *i z*
proof −
have *z* ∉ *path-image* γ
using ‹*x* ∈ *path-image* γ› *d that ball-divide-subset-numeral* **by** *blast*
then have *xz*[*simp*]: *x* ≠ *z* **using** ‹*x* ∈ *path-image* γ› **by** *blast*
then have *neq*: *x* ∗ *x* + *z* ∗ *z* ≠ *x* ∗ (*z* ∗ 2)
by (*blast intro*: *dest*!: *sum-sqs-eq*)
with *xz* **have** $\bigwedge v.\ v \ne 0 \Longrightarrow (x * x + z * z) * v \ne (x * (z * 2) * v)$ **by** *auto*
then have *neqq*: $\bigwedge v.\ v \ne 0 \Longrightarrow x * (x * v) + z * (z * v) \ne x * (z * (2 * v))$
by (*simp add*: *algebra-simps*)
show *?thesis* **using** ‹*i* ≤ 1›
apply (*simp add*: *ff-def dist-norm Nat.le-Suc-eq km1*, *safe*)
apply (*rule derivative-eq-intros* | *simp add*: *km1* | *simp add*: *field-simps neq neqq*)+
done
qed
{ fix *a*::*real* **and** *b*::*real* **assume** *ab*: *a* > 0 *b* > 0
then have *k* ∗ (1 + *real k*) ∗ (1 / *a*) ≤ *k* ∗ (1 + *real k*) ∗ (4 / *b*) ⟷ *b* ≤ 4 ∗ *a*
apply (*subst mult-le-cancel-left-pos*)
using ‹*k* ≠ 0›

```
        apply (auto simp: divide-simps)
        done
      with ab have real k * (1 + real k) / a ≤ (real k * 4 + real k * real k * 4)
/ b ⟷ b ≤ 4 * a
        by (simp add: field-simps)
    } note canc = this
    have ff2: cmod (ff (Suc 1) v) ≤ real (k * (k + 1)) / (d / 2) ^ (k + 2)
          if v ∈ ball w (d / 2) for v
    proof −
      have d/2 ≤ cmod (x − v) using d x that
       apply (simp add: dist-norm path-image-def ball-def not-less [symmetric]
del: divide-const-simps, clarify)
        apply (drule subsetD)
         prefer 2 apply blast
        apply (metis norm-minus-commute norm-triangle-half-r CollectI)
        done
      then have d ≤ cmod (x − v) * 2
        by (simp add: divide-simps)
      then have dpow-le: d ^ (k+2) ≤ (cmod (x − v) * 2) ^ (k+2)
        using ‹0 < d› order-less-imp-le power-mono by blast
      have x ≠ v using that
        using ‹x ∈ path-image γ› ball-divide-subset-numeral d by fastforce
      then show ?thesis
      using ‹d > 0›
     apply (simp add: ff-def norm-mult norm-divide norm-power dist-norm canc)
      using dpow-le
      apply (simp add: algebra-simps divide-simps mult-less-0-iff)
      done
    qed
    have ub: u ∈ ball w (d / 2)
      using uwd by (simp add: dist-commute dist-norm)
    have cmod (inverse (x − u) ^ k − (inverse (x − w) ^ k + of-nat k * (u −
w) / ((x − w) * (x − w) ^ k)))
          ≤ (real k * 4 + real k * real k * 4) * (cmod (u − w) * cmod (u −
w)) / (d * (d * (d / 2) ^ k))
      using complex-taylor [OF - ff1 ff2 - ub, of w, simplified]
      by (simp add: ff-def ‹0 < d›)
    then have cmod (inverse (x − u) ^ k − (inverse (x − w) ^ k + of-nat k *
(u − w) / ((x − w) * (x − w) ^ k)))
          ≤ (cmod (u − w) * real k) * (1 + real k) * cmod (u − w) / (d /
2) ^ (k+2)
      by (simp add: field-simps)
    then have cmod (inverse (x − u) ^ k − (inverse (x − w) ^ k + of-nat k *
(u − w) / ((x − w) * (x − w) ^ k)))
          / (cmod (u − w) * real k)
          ≤ (1 + real k) * cmod (u − w) / (d / 2) ^ (k+2)
    using ‹k ≠ 0› ‹u ≠ w› by (simp add: mult-ac zero-less-mult-iff pos-divide-le-eq)
    also have ... < e
      using uw-less ‹0 < d› by (simp add: mult-ac divide-simps)
```

```
      finally have e: cmod (inverse (x−u) ^k − (inverse (x−w) ^k + of-nat k *
(u−w) / ((x−w) * (x−w) ^k)))
                    / cmod ((u − w) * real k)  <  e
        by (simp add: norm-mult)
      have x ≠ u
      using uwd ‹0 < d› x d by (force simp: dist-norm ball-def norm-minus-commute)
      show ?thesis
        apply (rule le-less-trans [OF - e])
        using ‹k ≠ 0› ‹x ≠ u› ‹u ≠ w›
        apply (simp add: field-simps norm-divide [symmetric])
        done
    qed
    show ?thesis
      unfolding eventually-at
      apply (rule-tac x = min (d/2) ((e*(d/2) ^(k + 2))/(Suc k)) in exI)
      apply (force simp: ‹d > 0› dist-norm that simp del: power-Suc intro: *)
      done
  qed
  have 2: ∀F n in at w.
            ∀x∈path-image γ.
             cmod (f' x * (inverse (x − n) ^ k − inverse (x − w) ^ k) / (n − w)
/ of-nat k − f' x / (x − w) ^ Suc k) < e
          if 0 < e for e
  proof −
    have *: cmod (f' (γ x) * (inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) /
((u − w) * k) −
                    f' (γ x) / ((γ x − w) * (γ x − w) ^ k)) < e
              if ec: cmod ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u −
w) * k) −
                      inverse (γ x − w) * inverse (γ x − w) ^ k) < e / C
                and x: 0 ≤ x x ≤ 1
              for u x
    proof (cases (f' (γ x)) = 0)
      case True then show ?thesis by (simp add: ‹0 < e›)
    next
      case False
      have cmod (f' (γ x) * (inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) /
((u − w) * k) −
                    f' (γ x) / ((γ x − w) * (γ x − w) ^ k)) =
           cmod (f' (γ x) * ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u
− w) * k) −
                             inverse (γ x − w) * inverse (γ x − w) ^ k))
        by (simp add: field-simps)
      also have ... = cmod (f' (γ x)) *
                      cmod ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u
− w) * k) −
                             inverse (γ x − w) * inverse (γ x − w) ^ k)
        by (simp add: norm-mult)
      also have ... < cmod (f' (γ x)) * (e/C)
```

```
      apply (rule mult-strict-left-mono [OF ec])
      using False by simp
    also have ... ≤ e using C
     by (metis False ‹0 < e› frac-le less-eq-real-def mult.commute pos-le-divide-eq
x zero-less-norm-iff)
    finally show ?thesis .
  qed
  show ?thesis
   using twom [OF divide-pos-pos [OF that ‹C > 0›]]  unfolding path-image-def
    by (force intro: ∗ elim: eventually-mono)
 qed
 show (λu. f' u / (u − w) ^ (Suc k)) contour-integrable-on γ
  by (rule contour-integral-uniform-limit [OF 1 2 leB γ]) auto
 have ∗: (λn. contour-integral γ (λx. f' x ∗ (inverse (x − n) ^ k − inverse (x −
w) ^ k) / (n − w) / k))
        −w→ contour-integral γ (λu. f' u / (u − w) ^ (Suc k))
  by (rule contour-integral-uniform-limit [OF 1 2 leB γ]) auto
 have ∗∗: contour-integral γ (λx. f' x ∗ (inverse (x − u) ^ k − inverse (x − w)
^ k) / ((u − w) ∗ k)) =
           (f u − f w) / (u − w) / k
         if dist u w < d for u
  apply (rule contour-integral-unique)
  apply (simp add: diff-divide-distrib algebra-simps)
   apply (rule has-contour-integral-diff; rule has-contour-integral-div; simp add:
field-simps; rule int)
  apply (metis contra-subsetD d dist-commute mem-ball that)
  apply (rule w)
  done
 show ?thes2
  apply (simp add: DERIV-within-iff del: power-Suc)
  apply (rule Lim-transform-within [OF tendsto-mult-left [OF ∗] ‹0 < d› ])
  apply (simp add: ‹k ≠ 0› ∗∗)
  done
qed

corollary Cauchy-next-derivative-circlepath:
 assumes contf: continuous-on (path-image (circlepath z r)) f
    and int: ⋀w. w ∈ ball z r ⟹ ((λu. f u / (u − w) ^k) has-contour-integral g
w) (circlepath z r)
    and k: k ≠ 0
    and w: w ∈ ball z r
  shows (λu. f u / (u − w) ^(Suc k)) contour-integrable-on (circlepath z r)
        (is ?thes1)
    and (g has-field-derivative (k ∗ contour-integral (circlepath z r) (λu. f u/(u
− w) ^(Suc k)))) (at w)
        (is ?thes2)
proof −
 have r > 0 using w
  using ball-eq-empty by fastforce
```

have *wim*: $w \in$ *ball z r* − *path-image* (*circlepath z r*)
using *w* **by** (*auto simp*: *dist-norm*)
show *?thes1 ?thes2*
by (*rule Cauchy-next-derivative* [*OF contf - int k open-ball valid-path-circlepath wim*, **where** $B = 2 * pi * |r|$];
auto simp: *vector-derivative-circlepath norm-mult*)+
qed

In particular, the first derivative formula.

proposition *Cauchy-derivative-integral-circlepath*:
assumes *contf*: *continuous-on* (*cball z r*) *f*
and *holf*: *f holomorphic-on ball z r*
and *w*: $w \in$ *ball z r*
shows ($\lambda u.$ *f u*/(*u* − *w*) ^ *2*) *contour-integrable-on* (*circlepath z r*)
(**is** *?thes1*)
and (*f has-field-derivative* (*1* / (*2* ∗ *of-real pi* ∗ *ii*) ∗ *contour-integral*(*circlepath z r*) ($\lambda u.$ *f u* / (*u* − *w*) ^ *2*))) (*at w*)
(**is** *?thes2*)
proof −
have [*simp*]: $r \geq 0$ **using** *w*
using *ball-eq-empty* **by** *fastforce*
have *f*: *continuous-on* (*path-image* (*circlepath z r*)) *f*
by (*rule continuous-on-subset* [*OF contf*]) (*force simp add*: *cball-def sphere-def*)
have *int*: $\bigwedge w.$ *dist z w* < *r* ⟹
(($\lambda u.$ *f u* / (*u* − *w*)) *has-contour-integral* ($\lambda x.$ *2* ∗ *of-real pi* ∗ *ii* ∗ *f x*) *w*) (*circlepath z r*)
by (*rule Cauchy-integral-circlepath* [*OF contf holf*]) (*simp add*: *dist-norm norm-minus-commute*)
show *?thes1*
apply (*simp add*: *power2-eq-square*)
apply (*rule Cauchy-next-derivative-circlepath* [*OF f - - w*, **where** *k=1*, *simplified*])
apply (*blast intro*: *int*)
done
have (($\lambda x.$ *2* ∗ *of-real pi* ∗ i ∗ *f x*) *has-field-derivative contour-integral* (*circlepath z r*) ($\lambda u.$ *f u* / (*u* − *w*) ^ *2*)) (*at w*)
apply (*simp add*: *power2-eq-square*)
apply (*rule Cauchy-next-derivative-circlepath* [*OF f - - w*, **where** *k=1* **and** *g* = $\lambda x.$ *2* ∗ *of-real pi* ∗ *ii* ∗ *f x*, *simplified*])
apply (*blast intro*: *int*)
done
then have *fder*: (*f has-field-derivative contour-integral* (*circlepath z r*) ($\lambda u.$ *f u* / (*u* − *w*) ^ *2*) / (*2* ∗ *of-real pi* ∗ i)) (*at w*)
by (*rule DERIV-cdivide* [**where** *f* = $\lambda x.$ *2* ∗ *of-real pi* ∗ i ∗ *f x* **and** *c* = *2* ∗ *of-real pi* ∗ i, *simplified*])
show *?thes2*
by *simp* (*rule fder*)
qed

52.36 Existence of all higher derivatives.

proposition *derivative-is-holomorphic*:
assumes *open s*
and *fder*: $\bigwedge z.\ z \in s \Longrightarrow (f \text{ has-field-derivative } f'\ z)\ (\text{at } z)$
shows *f' holomorphic-on s*
proof −
have ∗: $\exists h.\ (f' \text{ has-field-derivative } h)\ (\text{at } z)$ **if** $z \in s$ **for** *z*
proof −
obtain *r* **where** $r > 0$ **and** *r*: *cball z r* ⊆ *s*
using *open-contains-cball* ‹*z* ∈ *s*› ‹*open s*› **by** *blast*
then have *holf-cball*: *f holomorphic-on cball z r*
apply (*simp add*: *holomorphic-on-def*)
using *complex-differentiable-at-within complex-differentiable-def fder* **by** *blast*
then have *continuous-on* (*path-image* (*circlepath z r*)) *f*
using ‹*r* > *0*› **by** (*force elim*: *holomorphic-on-subset* [*THEN holomorphic-on-imp-continuous-on*])
then have *contfpi*: *continuous-on* (*path-image* (*circlepath z r*)) (λ*x*. *1*/(*2* ∗ *of-real pi*∗*ii*) ∗ *f x*)
by (*auto intro*: *continuous-intros*)+
have *contf-cball*: *continuous-on* (*cball z r*) *f* **using** *holf-cball*
by (*simp add*: *holomorphic-on-imp-continuous-on holomorphic-on-subset*)
have *holf-ball*: *f holomorphic-on ball z r* **using** *holf-cball*
using *ball-subset-cball holomorphic-on-subset* **by** *blast*
{ fix *w* **assume** *w*: *w* ∈ *ball z r*
have *intf*: (λ*u*. *f u* / (*u* − *w*)²) *contour-integrable-on circlepath z r*
by (*blast intro*: *w Cauchy-derivative-integral-circlepath* [*OF contf-cball holf-ball*])
have *fder'*: (*f has-field-derivative 1* / (*2* ∗ *of-real pi* ∗ i) ∗ *contour-integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)²))
(*at w*)
by (*blast intro*: *w Cauchy-derivative-integral-circlepath* [*OF contf-cball holf-ball*])
have *f'-eq*: *f' w* = *contour-integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)²) / (*2* ∗ *of-real pi* ∗ i)
using *fder' ball-subset-cball r w* **by** (*force intro*: *DERIV-unique* [*OF fder*])
have ((λ*u*. *f u* / (*u* − *w*)² / (*2* ∗ *of-real pi* ∗ i)) *has-contour-integral*
contour-integral (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)²) / (*2* ∗ *of-real pi* ∗ i))
(*circlepath z r*)
by (*rule Cauchy-Integral-Thm.has-contour-integral-div* [*OF has-contour-integral-integral* [*OF intf*]])
then have ((λ*u*. *f u* / (*2* ∗ *of-real pi* ∗ i ∗ (*u* − *w*)²)) *has-contour-integral*
contour-integral (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)²) / (*2* ∗ *of-real pi* ∗ i))
(*circlepath z r*)
by (*simp add*: *algebra-simps*)
then have ((λ*u*. *f u* / (*2* ∗ *of-real pi* ∗ i ∗ (*u* − *w*)²)) *has-contour-integral f' w*) (*circlepath z r*)
by (*simp add*: *f'-eq*)
} note ∗ = *this*

show *?thesis*
apply (*rule exI*)
apply (*rule Cauchy-next-derivative-circlepath* [*OF contfpi*, *of 2 f'*, *simplified*])
apply (*simp-all add*: ‹*0 < r*› ∗ *dist-norm*)
done
qed
show *?thesis*
by (*simp add*: *holomorphic-on-open* [*OF* ‹*open s*›] ∗)
qed

lemma *holomorphic-deriv* [*holomorphic-intros*]:
⟦*f holomorphic-on s*; *open s*⟧ ⟹ (*deriv f*) *holomorphic-on s*
by (*metis DERIV-deriv-iff-complex-differentiable at-within-open derivative-is-holomorphic holomorphic-on-def*)

lemma *analytic-deriv*: *f analytic-on s* ⟹ (*deriv f*) *analytic-on s*
using *analytic-on-holomorphic holomorphic-deriv* **by** *auto*

lemma *holomorphic-higher-deriv* [*holomorphic-intros*]: ⟦*f holomorphic-on s*; *open s*⟧ ⟹ (*deriv* ^^ *n*) *f holomorphic-on s*
by (*induction n*) (*auto simp*: *holomorphic-deriv*)

lemma *analytic-higher-deriv*: *f analytic-on s* ⟹ (*deriv* ^^ *n*) *f analytic-on s*
unfolding *analytic-on-def* **using** *holomorphic-higher-deriv* **by** *blast*

lemma *has-field-derivative-higher-deriv*:
⟦*f holomorphic-on s*; *open s*; *x* ∈ *s*⟧
⟹ ((*deriv* ^^ *n*) *f has-field-derivative* (*deriv* ^^ (*Suc n*)) *f x*) (*at x*)
by (*metis* (*no-types*, *hide-lams*) *DERIV-deriv-iff-complex-differentiable at-within-open comp-apply*
funpow.simps(*2*) *holomorphic-higher-deriv holomorphic-on-def*)

52.37 Morera's theorem.

lemma *Morera-local-triangle-ball*:
assumes ⋀*z*. *z* ∈ *s*
⟹ ∃ *e a*. *0 < e* ∧ *z* ∈ *ball a e* ∧ *continuous-on* (*ball a e*) *f* ∧
(∀ *b c*. *closed-segment b c* ⊆ *ball a e*
⟶ *contour-integral* (*linepath a b*) *f* +
contour-integral (*linepath b c*) *f* +
contour-integral (*linepath c a*) *f* = *0*)
shows *f analytic-on s*
proof −
{ fix *z* **assume** *z* ∈ *s*
with *assms* **obtain** *e a* **where**
0 < e **and** *z*: *z* ∈ *ball a e* **and** *contf*: *continuous-on* (*ball a e*) *f*
and *0*: ⋀*b c*. *closed-segment b c* ⊆ *ball a e*
⟹ *contour-integral* (*linepath a b*) *f* +
contour-integral (*linepath b c*) *f* +

```
                    contour-integral (linepath c a) f = 0
    by fastforce
   have az: dist a z < e using mem-ball z by blast
   have sb-ball: ball z (e − dist a z) ⊆ ball a e
    by (simp add: dist-commute ball-subset-ball-iff)
   have ∃ e>0. f holomorphic-on ball z e
    apply (rule-tac x=e − dist a z in exI)
    apply (simp add: az)
    apply (rule holomorphic-on-subset [OF - sb-ball])
    apply (rule derivative-is-holomorphic[OF open-ball])
   apply (rule triangle-contour-integrals-starlike-primitive [OF contf - open-ball,
of a])
     apply (simp-all add: 0 ‹0 < e›)
    apply (meson ‹0 < e› centre-in-ball convex-ball convex-contains-segment
mem-ball)
    done
 }
 then show ?thesis
   by (simp add: analytic-on-def)
qed

lemma Morera-local-triangle:
 assumes ⋀z. z ∈ s
       ⟹ ∃ t. open t ∧ z ∈ t ∧ continuous-on t f ∧
            (∀ a b c. convex hull {a,b,c} ⊆ t
                  ⟶ contour-integral (linepath a b) f +
                    contour-integral (linepath b c) f +
                    contour-integral (linepath c a) f = 0)
 shows f analytic-on s
proof −
 { fix z assume z ∈ s
   with assms obtain t where
        open t and z: z ∈ t and contf: continuous-on t f
     and 0: ⋀a b c. convex hull {a,b,c} ⊆ t
                ⟹ contour-integral (linepath a b) f +
                   contour-integral (linepath b c) f +
                   contour-integral (linepath c a) f = 0
    by force
   then obtain e where e>0 and e: ball z e ⊆ t
    using open-contains-ball by blast
   have [simp]: continuous-on (ball z e) f using contf
    using continuous-on-subset e by blast
   have ∃ e a. 0 < e ∧
           z ∈ ball a e ∧
           continuous-on (ball a e) f ∧
           (∀ b c. closed-segment b c ⊆ ball a e ⟶
                   contour-integral (linepath a b) f + contour-integral (linepath b
c) f + contour-integral (linepath c a) f = 0)
    apply (rule-tac x=e in exI)
```

apply (*rule-tac x=z* **in** *exI*)
apply (*simp add*: ‹*e* > *0*›, *clarify*)
apply (*rule 0*)
apply (*meson z* ‹*0* < *e*› *centre-in-ball closed-segment-subset convex-ball dual-order.trans e starlike-convex-subset*)
done
}
then show *?thesis*
by (*simp add*: *Morera-local-triangle-ball*)
qed

proposition *Morera-triangle*:
⟦*continuous-on s f*; *open s*;
⋀*a b c*. *convex hull* {*a*,*b*,*c*} ⊆ *s*
⟶ *contour-integral* (*linepath a b*) *f* +
contour-integral (*linepath b c*) *f* +
contour-integral (*linepath c a*) *f* = *0*⟧
⟹ *f analytic-on s*
using *Morera-local-triangle* **by** *blast*

52.38 Combining theorems for higher derivatives including Leibniz rule.

lemma *higher-deriv-linear* [*simp*]:
(*deriv* ^^ *n*) (λ*w*. *c*∗*w*) = (λ*z*. *if n* = *0 then c*∗*z else if n* = *1 then c else 0*)
by (*induction n*) (*auto simp*: *deriv-const deriv-linear*)

lemma *higher-deriv-const* [*simp*]: (*deriv* ^^ *n*) (λ*w*. *c*) = (λ*w*. *if n=0 then c else 0*)
by (*induction n*) (*auto simp*: *deriv-const*)

lemma *higher-deriv-ident* [*simp*]:
(*deriv* ^^ *n*) (λ*w*. *w*) *z* = (*if n* = *0 then z else if n* = *1 then 1 else 0*)
apply (*induction n*)
apply (*simp-all add*: *deriv-ident funpow-Suc-right del*: *funpow.simps*, *simp*)
done

corollary *higher-deriv-id* [*simp*]:
(*deriv* ^^ *n*) *id z* = (*if n* = *0 then z else if n* = *1 then 1 else 0*)
by (*simp add*: *id-def*)

lemma *has-complex-derivative-funpow-1*:
⟦(*f has-field-derivative 1*) (*at z*); *f z* = *z*⟧ ⟹ (*f*^^*n has-field-derivative 1*) (*at z*)
apply (*induction n*)
apply *auto*
apply (*metis DERIV-ident DERIV-transform-at id-apply zero-less-one*)
by (*metis DERIV-chain comp-funpow comp-id funpow-swap1 mult.right-neutral*)

proposition *higher-deriv-uminus*:
assumes *f holomorphic-on s open s* **and** *z*: *z* ∈ *s*
shows (*deriv* ^^ *n*) (λ*w*. −(*f w*)) *z* = − ((*deriv* ^^ *n*) *f z*)
using *z*
proof (*induction n arbitrary*: *z*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc n z*)
have ∗: ((*deriv* ^^ *n*) *f has-field-derivative deriv* ((*deriv* ^^ *n*) *f*) *z*) (*at z*)
using *Suc.prems assms has-field-derivative-higher-deriv* **by** *auto*
show *?case*
apply *simp*
apply (*rule DERIV-imp-deriv*)
apply (*rule DERIV-transform-within-open* [*of* λ*w*. −((*deriv* ^^ *n*) *f w*)])
apply (*rule derivative-eq-intros* | *rule* ∗ *refl assms Suc*)+
apply (*simp add*: *Suc*)
done
qed

proposition *higher-deriv-add*:
fixes *z*::*complex*
assumes *f holomorphic-on s g holomorphic-on s open s* **and** *z*: *z* ∈ *s*
shows (*deriv* ^^ *n*) (λ*w*. *f w* + *g w*) *z* = (*deriv* ^^ *n*) *f z* + (*deriv* ^^ *n*) *g z*
using *z*
proof (*induction n arbitrary*: *z*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc n z*)
have ∗: ((*deriv* ^^ *n*) *f has-field-derivative deriv* ((*deriv* ^^ *n*) *f*) *z*) (*at z*)
((*deriv* ^^ *n*) *g has-field-derivative deriv* ((*deriv* ^^ *n*) *g*) *z*) (*at z*)
using *Suc.prems assms has-field-derivative-higher-deriv* **by** *auto*
show *?case*
apply *simp*
apply (*rule DERIV-imp-deriv*)
apply (*rule DERIV-transform-within-open* [*of* λ*w*. (*deriv* ^^ *n*) *f w* + (*deriv* ^^ *n*) *g w*])
apply (*rule derivative-eq-intros* | *rule* ∗ *refl assms Suc*)+
apply (*simp add*: *Suc*)
done
qed

corollary *higher-deriv-diff*:
fixes *z*::*complex*
assumes *f holomorphic-on s g holomorphic-on s open s* **and** *z*: *z* ∈ *s*
shows (*deriv* ^^ *n*) (λ*w*. *f w* − *g w*) *z* = (*deriv* ^^ *n*) *f z* − (*deriv* ^^ *n*) *g z*
apply (*simp only*: *Groups.group-add-class.diff-conv-add-uminus higher-deriv-add*)
apply (*subst higher-deriv-add*)
using *assms holomorphic-on-minus* **apply** (*auto simp*: *higher-deriv-uminus*)
done

lemma *bb*: *Suc n choose k = (n choose k) + (if k = 0 then 0 else (n choose (k − 1)))*
by (*simp add*: *Binomial.binomial.simps*)

proposition *higher-deriv-mult*:
fixes *z*::*complex*
assumes *f holomorphic-on s g holomorphic-on s open s* **and** *z*: *z ∈ s*
shows (*deriv ^^ n*) (*λw. f w * g w*) *z* =
(∑ *i* = *0..n*. *of-nat* (*n choose i*) * (*deriv ^^ i*) *f z* * (*deriv ^^* (*n − i*)) *g z*)
using *z*
proof (*induction n arbitrary*: *z*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc n z*)
have *: ⋀*n*. ((*deriv ^^ n*) *f has-field-derivative deriv* ((*deriv ^^ n*) *f*) *z*) (*at z*)
⋀*n*. ((*deriv ^^ n*) *g has-field-derivative deriv* ((*deriv ^^ n*) *g*) *z*) (*at z*)
using *Suc.prems assms has-field-derivative-higher-deriv* **by** *auto*
have *sumeq*: (∑ *i* = *0..n*.
of-nat (*n choose i*) * (*deriv* ((*deriv ^^ i*) *f*) *z* * (*deriv ^^* (*n − i*)) *g z* + *deriv* ((*deriv ^^* (*n − i*)) *g*) *z* * (*deriv ^^ i*) *f z*)) =
g z * *deriv* ((*deriv ^^ n*) *f*) *z* + (∑ *i* = *0..n*. (*deriv ^^ i*) *f z* * (*of-nat* (*Suc n choose i*) * (*deriv ^^* (*Suc n − i*)) *g z*))
apply (*simp add*: *bb distrib-right algebra-simps setsum.distrib*)
apply (*subst* (*4*) *setsum-Suc-reindex*)
apply (*auto simp*: *algebra-simps Suc-diff-le intro*: *setsum.cong*)
done
show *?case*
apply (*simp only*: *funpow.simps o-apply*)
apply (*rule DERIV-imp-deriv*)
apply (*rule DERIV-transform-within-open*
[*of λw*. (∑ *i* = *0..n*. *of-nat* (*n choose i*) * (*deriv ^^ i*) *f w* * (*deriv ^^* (*n − i*)) *g w*)])
apply (*simp add*: *algebra-simps*)
apply (*rule DERIV-cong* [*OF DERIV-setsum*])
apply (*rule DERIV-cmult*)
apply (*auto simp*: *intro*: *DERIV-mult * sumeq* ‹*open s*› *Suc.prems Suc.IH* [*symmetric*])
done
qed

proposition *higher-deriv-transform-within-open*:
fixes *z*::*complex*
assumes *f holomorphic-on s g holomorphic-on s open s* **and** *z*: *z ∈ s*
and *fg*: *⋀w. w ∈ s ⟹ f w = g w*
shows (*deriv ^^ i*) *f z* = (*deriv ^^ i*) *g z*

using *z*
by (*induction i arbitrary*: *z*)
(*auto simp*: *fg intro*: *complex-derivative-transform-within-open holomorphic-higher-deriv assms*)

proposition *higher-deriv-compose-linear*:
fixes *z*::*complex*
assumes *f*: *f holomorphic-on t* **and** *s*: *open s* **and** *t*: *open t* **and** *z*: *z* ∈ *s*
and *fg*: ⋀*w*. *w* ∈ *s* ⟹ *u* ∗ *w* ∈ *t*
shows (*deriv* ^^ *n*) (λ*w*. *f* (*u* ∗ *w*)) *z* = *u*^*n* ∗ (*deriv* ^^ *n*) *f* (*u* ∗ *z*)
using *z*
proof (*induction n arbitrary*: *z*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc n z*)
have *holo0*: *f holomorphic-on op* ∗ *u* ' *s*
by (*meson fg f holomorphic-on-subset image-subset-iff*)
have *holo1*: (λ*w*. *f* (*u* ∗ *w*)) *holomorphic-on s*
apply (*rule holomorphic-on-compose* [**where** *g*=*f*, *unfolded o-def*])
apply (*rule holo0 holomorphic-intros*)+
done
have *holo2*: (λ*z*. *u* ^ *n* ∗ (*deriv* ^^ *n*) *f* (*u* ∗ *z*)) *holomorphic-on s*
apply (*rule holomorphic-intros*)+
apply (*rule holomorphic-on-compose* [**where** *g*=(*deriv* ^^ *n*) *f*, *unfolded o-def*])
apply (*rule holomorphic-intros*)
apply (*rule holomorphic-on-subset* [**where** *s*=*t*])
apply (*rule holomorphic-intros assms*)+
apply (*blast intro*: *fg*)
done
have *deriv* ((*deriv* ^^ *n*) (λ*w*. *f* (*u* ∗ *w*))) *z* = *deriv* (λ*z*. *u*^*n* ∗ (*deriv* ^^ *n*) *f* (*u*∗*z*)) *z*
apply (*rule complex-derivative-transform-within-open* [*OF - holo2 s Suc.prems*])
apply (*rule holomorphic-higher-deriv* [*OF holo1 s*])
apply (*simp add*: *Suc.IH*)
done
also have ... = *u*^*n* ∗ *deriv* (λ*z*. (*deriv* ^^ *n*) *f* (*u* ∗ *z*)) *z*
apply (*rule complex-derivative-cmult*)
apply (*rule analytic-on-imp-differentiable-at* [*OF - Suc.prems*])
apply (*rule analytic-on-compose-gen* [**where** *g*=(*deriv* ^^ *n*) *f* **and** *t*=*t*, *unfolded o-def*])
apply (*simp add*: *analytic-on-linear*)
apply (*simp add*: *analytic-on-open f holomorphic-higher-deriv t*)
apply (*blast intro*: *fg*)
done
also have ... = *u* ∗ *u* ^ *n* ∗ *deriv* ((*deriv* ^^ *n*) *f*) (*u* ∗ *z*)
apply (*subst complex-derivative-chain* [**where** *g* = (*deriv* ^^ *n*) *f* **and** *f* = *op*∗*u*, *unfolded o-def*])
apply (*rule derivative-intros*)
using *Suc.prems complex-differentiable-def f fg has-field-derivative-higher-deriv*

t **apply** *blast*
apply (*simp add*: *deriv-linear*)
done
finally show *?case*
by *simp*
qed

lemma *higher-deriv-add-at*:
assumes *f analytic-on* {*z*} *g analytic-on* {*z*}
shows (*deriv* ^^ *n*) (λ*w*. *f w* + *g w*) *z* = (*deriv* ^^ *n*) *f z* + (*deriv* ^^ *n*) *g z*
proof −
have *f analytic-on* {*z*} ∧ *g analytic-on* {*z*}
using *assms* **by** *blast*
with *higher-deriv-add* **show** *?thesis*
by (*auto simp*: *analytic-at-two*)
qed

lemma *higher-deriv-diff-at*:
assumes *f analytic-on* {*z*} *g analytic-on* {*z*}
shows (*deriv* ^^ *n*) (λ*w*. *f w* − *g w*) *z* = (*deriv* ^^ *n*) *f z* − (*deriv* ^^ *n*) *g z*
proof −
have *f analytic-on* {*z*} ∧ *g analytic-on* {*z*}
using *assms* **by** *blast*
with *higher-deriv-diff* **show** *?thesis*
by (*auto simp*: *analytic-at-two*)
qed

lemma *higher-deriv-uminus-at*:
f analytic-on {*z*} ⟹ (*deriv* ^^ *n*) (λ*w*. −(*f w*)) *z* = − ((*deriv* ^^ *n*) *f z*)
using *higher-deriv-uminus*
by (*auto simp*: *analytic-at*)

lemma *higher-deriv-mult-at*:
assumes *f analytic-on* {*z*} *g analytic-on* {*z*}
shows (*deriv* ^^ *n*) (λ*w*. *f w* ∗ *g w*) *z* =
(∑ *i* = *0*..*n*. *of-nat* (*n choose i*) ∗ (*deriv* ^^ *i*) *f z* ∗ (*deriv* ^^ (*n* − *i*)) *g z*)
proof −
have *f analytic-on* {*z*} ∧ *g analytic-on* {*z*}
using *assms* **by** *blast*
with *higher-deriv-mult* **show** *?thesis*
by (*auto simp*: *analytic-at-two*)
qed

Nonexistence of isolated singularities and a stronger integral formula.

proposition *no-isolated-singularity*:
fixes *z*::*complex*
assumes *f*: *continuous-on s f* **and** *holf*: *f holomorphic-on* (*s* − *k*) **and** *s*: *open s* **and** *k*: *finite k*

shows *f holomorphic-on s*
proof −
{ fix *z*
assume *z* ∈ *s* **and** *cdf*: ⋀*x*. *x*∈*s* − *k* ⟹ *f complex-differentiable at x*
have *f complex-differentiable at z*
proof (*cases z* ∈ *k*)
case *False* **then show** *?thesis* **by** (*blast intro*: *cdf* ‹*z* ∈ *s*›)
next
case *True*
with *finite-set-avoid* [*OF k*, *of z*]
obtain *d* **where** *d*>*0* **and** *d*: ⋀*x*. ⟦*x*∈*k*; *x* ≠ *z*⟧ ⟹ *d* ≤ *dist z x*
by *blast*
obtain *e* **where** *e*>*0* **and** *e*: *ball z e* ⊆ *s*
using *s* ‹*z* ∈ *s*› **by** (*force simp add*: *open-contains-ball*)
have *fde*: *continuous-on* (*ball z* (*min d e*)) *f*
by (*metis Int-iff ball-min-Int continuous-on-subset e f subsetI*)
have ∃ *g*. ∀ *w* ∈ *ball z* (*min d e*). (*g has-field-derivative f w*) (*at w within ball z* (*min d e*))
apply (*rule contour-integral-convex-primitive* [*OF convex-ball fde*])
apply (*rule Cauchy-theorem-triangle-cofinite* [*OF - k*])
apply (*metis continuous-on-subset* [*OF fde*] *closed-segment-subset convex-ball starlike-convex-subset*)
apply (*rule cdf*)
apply (*metis Diff-iff Int-iff ball-min-Int bot-least contra-subsetD convex-ball e insert-subset*
interior-mono interior-subset subset-hull)
done
then have *f holomorphic-on ball z* (*min d e*)
by (*metis open-ball at-within-open derivative-is-holomorphic*)
then show *?thesis*
unfolding *holomorphic-on-def*
by (*metis open-ball* ‹*0* < *d*› ‹*0* < *e*› *at-within-open centre-in-ball min-less-iff-conj*)
qed
}
with *holf s k* **show** *?thesis*
by (*simp add*: *holomorphic-on-open open-Diff finite-imp-closed complex-differentiable-def* [*symmetric*])
qed

proposition *Cauchy-integral-formula-convex*:
assumes *s*: *convex s* **and** *k*: *finite k* **and** *contf*: *continuous-on s f*
and *fcd*: (⋀*x*. *x* ∈ *interior s* − *k* ⟹ *f complex-differentiable at x*)
and *z*: *z* ∈ *interior s* **and** *vpg*: *valid-path* γ
and *pasz*: *path-image* γ ⊆ *s* − {*z*} **and** *loop*: *pathfinish* γ = *pathstart* γ
shows ((λ*w*. *f w* / (*w* − *z*)) *has-contour-integral* (*2*pi* ∗ *ii* ∗ *winding-number* γ *z* ∗ *f z*)) γ
apply (*rule Cauchy-integral-formula-weak* [*OF s finite.emptyI contf*])
apply (*simp add*: *holomorphic-on-open* [*symmetric*] *complex-differentiable-def*)
using *no-isolated-singularity* [**where** *s* = *interior s*]

apply (*metis k contf fcd holomorphic-on-open complex-differentiable-def continuous-on-subset has-field-derivative-at-within holomorphic-on-def interior-subset open-interior*)
using *assms*
apply *auto*
done

Formula for higher derivatives.

proposition *Cauchy-has-contour-integral-higher-derivative-circlepath*:
assumes *contf*: *continuous-on* (*cball z r*) *f*
and *holf*: *f holomorphic-on ball z r*
and *w*: $w \in$ *ball z r*
shows ((λ*u*. *f u* / (*u* − *w*) ^ (*Suc k*)) *has-contour-integral* ((2 ∗ *pi* ∗ *ii*) / (*fact k*) ∗ (*deriv* ^^ *k*) *f w*)) (*circlepath z r*)
using *w*
proof (*induction k arbitrary*: *w*)
case *0* **then show** *?case*
using *assms* **by** (*auto simp*: *Cauchy-integral-circlepath dist-commute dist-norm*)
next
case (*Suc k*)
have [*simp*]: *r* > *0* **using** *w*
using *ball-eq-empty* **by** *fastforce*
have *f*: *continuous-on* (*path-image* (*circlepath z r*)) *f*
by (*rule continuous-on-subset* [*OF contf*]) (*force simp add*: *cball-def sphere-def less-imp-le*)
obtain *X* **where** *X*: ((λ*u*. *f u* / (*u* − *w*) ^ *Suc* (*Suc k*)) *has-contour-integral X*) (*circlepath z r*)
using *Cauchy-next-derivative-circlepath*(*1*) [*OF f Suc.IH - Suc.prems*]
by (*auto simp*: *contour-integrable-on-def*)
then have *con*: *contour-integral* (*circlepath z r*) ((λ*u*. *f u* / (*u* − *w*) ^ *Suc* (*Suc k*))) = *X*
by (*rule contour-integral-unique*)
have ⋀*n*. ((*deriv* ^^ *n*) *f has-field-derivative deriv* ((*deriv* ^^ *n*) *f*) *w*) (*at w*)
using *Suc.prems assms has-field-derivative-higher-deriv* **by** *auto*
then have *dnf-diff*: ⋀*n*. (*deriv* ^^ *n*) *f complex-differentiable* (*at w*)
by (*force simp add*: *complex-differentiable-def*)
have *deriv* (λ*w*. *complex-of-real* (*2* ∗ *pi*) ∗ i / (*fact k*) ∗ (*deriv* ^^ *k*) *f w*) *w* = *of-nat* (*Suc k*) ∗ *contour-integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*) ^ *Suc* (*Suc k*))
by (*force intro*!: *DERIV-imp-deriv Cauchy-next-derivative-circlepath* [*OF f Suc.IH - Suc.prems*])
also have ... = *of-nat* (*Suc k*) ∗ *X*
by (*simp only*: *con*)
finally have *deriv* (λ*w*. ((*2* ∗ *pi*) ∗ i / (*fact k*)) ∗ (*deriv* ^^ *k*) *f w*) *w* = *of-nat* (*Suc k*) ∗ *X* **.**
then have ((*2* ∗ *pi*) ∗ i / (*fact k*)) ∗ *deriv* (λ*w*. (*deriv* ^^ *k*) *f w*) *w* = *of-nat* (*Suc k*) ∗ *X*
by (*metis complex-derivative-cmult dnf-diff*)

then have *deriv* (λ*w*. (*deriv* ^^ *k*) *f w*) *w* = *of-nat* (*Suc k*) ∗ *X* / ((*2* ∗ *pi*) ∗ i / (*fact k*))
by (*simp add*: *field-simps*)
then show *?case*
using *of-nat-eq-0-iff X* **by** *fastforce*
qed

proposition *Cauchy-higher-derivative-integral-circlepath*:
assumes *contf*: *continuous-on* (*cball z r*) *f*
and *holf*: *f holomorphic-on ball z r*
and *w*: *w* ∈ *ball z r*
shows (λ*u*. *f u* / (*u* − *w*) ^ (*Suc k*)) *contour-integrable-on* (*circlepath z r*)
(**is** *?thes1*)
and (*deriv* ^^ *k*) *f w* = (*fact k*) / (*2* ∗ *pi* ∗ *ii*) ∗ *contour-integral*(*circlepath z r*) (λ*u*. *f u*/(*u* − *w*) ^ (*Suc k*))
(**is** *?thes2*)
proof −
have ∗: ((λ*u*. *f u* / (*u* − *w*) ^ *Suc k*) *has-contour-integral* (*2* ∗ *pi*) ∗ i / (*fact k*) ∗ (*deriv* ^^ *k*) *f w*)
(*circlepath z r*)
using *Cauchy-has-contour-integral-higher-derivative-circlepath* [*OF assms*]
by *simp*
show *?thes1* **using** ∗
using *contour-integrable-on-def* **by** *blast*
show *?thes2*
unfolding *contour-integral-unique* [*OF* ∗] **by** (*simp add*: *divide-simps*)
qed

corollary *Cauchy-contour-integral-circlepath*:
assumes *continuous-on* (*cball z r*) *f f holomorphic-on ball z r w* ∈ *ball z r*
shows *contour-integral*(*circlepath z r*) (λ*u*. *f u*/(*u* − *w*) ^ (*Suc k*)) = (*2* ∗ *pi* ∗ *ii*) ∗ (*deriv* ^^ *k*) *f w* / (*fact k*)
by (*simp add*: *Cauchy-higher-derivative-integral-circlepath* [*OF assms*])

corollary *Cauchy-contour-integral-circlepath-2*:
assumes *continuous-on* (*cball z r*) *f f holomorphic-on ball z r w* ∈ *ball z r*
shows *contour-integral*(*circlepath z r*) (λ*u*. *f u*/(*u* − *w*) ^ *2*) = (*2* ∗ *pi* ∗ *ii*) ∗ *deriv f w*
using *Cauchy-contour-integral-circlepath* [*OF assms*, *of 1*]
by (*simp add*: *power2-eq-square*)

52.39 A holomorphic function is analytic, i.e. has local power series.

theorem *holomorphic-power-series*:
assumes *holf*: *f holomorphic-on ball z r*
and *w*: *w* ∈ *ball z r*
shows ((λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*) *sums f w*)
proof −

have *fh'*: *f holomorphic-on cball z ((r + dist w z) / 2)*
apply (*rule holomorphic-on-subset* [*OF holf*])
apply (*clarsimp simp del*: *divide-const-simps*)
apply (*metis add.commute dist-commute le-less-trans mem-ball real-gt-half-sum w*)
done
— Replacing *r* and the original (weak) premises
obtain *r* **where** *0 < r* **and** *holfc*: *f holomorphic-on cball z r* **and** *w*: *w ∈ ball z r*
apply (*rule that* [*of (r + dist w z) / 2*])
apply (*simp-all add*: *fh'*)
apply (*metis add-0-iff ball-eq-empty dist-nz dist-self empty-iff not-less pos-add-strict w*)
apply (*metis add-less-cancel-right dist-commute mem-ball mult-2-right w*)
done
then have *holf*: *f holomorphic-on ball z r* **and** *contf*: *continuous-on (cball z r) f*
using *ball-subset-cball holomorphic-on-subset* **apply** *blast*
by (*simp add*: *holfc holomorphic-on-imp-continuous-on*)
have *cint*: *⋀k. (λu. f u / (u − z) ^ Suc k) contour-integrable-on circlepath z r*
apply (*rule Cauchy-higher-derivative-integral-circlepath* [*OF contf holf*])
apply (*simp add*: ‹*0 < r*›)
done
obtain *B* **where** *0 < B* **and** *B*: *⋀u. u ∈ cball z r ⟹ norm(f u) ≤ B*
by (*metis* (*no-types*) *bounded-pos compact-cball compact-continuous-image compact-imp-bounded contf image-eqI*)
obtain *k* **where** *k*: *0 < k k ≤ r* **and** *wz-eq*: *norm(w − z) = r − k*
and *kle*: *⋀u. norm(u − z) = r ⟹ k ≤ norm(u − w)*
apply (*rule-tac k = r − dist z w* **in** *that*)
using *w*
apply (*auto simp*: *dist-norm norm-minus-commute*)
by (*metis add-diff-eq diff-add-cancel norm-diff-ineq norm-minus-commute*)
have *∗*: *∀ F n in sequentially. ∀ x∈path-image (circlepath z r). norm ((∑ k<n. (w − z) ^ k ∗ (f x / (x − z) ^ Suc k)) − f x / (x − w)) < e*
if *0 < e* **for** *e*
proof −
have *rr*: *0 ≤ (r − k) / r (r − k) / r < 1* **using** *k* **by** *auto*
obtain *n* **where** *n*: *((r − k) / r) ^ n < e / B ∗ k*
using *real-arch-pow-inv* [*of e/B∗k (r − k)/r*] ‹*0 < e*› ‹*0 < B*› *k* **by** *force*
have *norm ((∑ k<N. (w − z) ^ k ∗ f u / (u − z) ^ Suc k) − f u / (u − w)) < e*
if *n ≤ N* **and** *r*: *r = dist z u* **for** *N u*
proof −
have *N*: *((r − k) / r) ^ N < e / B ∗ k*
apply (*rule le-less-trans* [*OF power-decreasing n*])
using ‹*n ≤ N*› *k* **by** *auto*
have *u* [*simp*]: *(u ≠ z) ∧ (u ≠ w)*
using ‹*0 < r*› *r w* **by** *auto*
have *wzu-not1*: *(w − z) / (u − z) ≠ 1*

by (*metis* (*no-types*) *dist-norm divide-eq-1-iff less-irrefl mem-ball norm-minus-commute r w*)
have *norm* ((∑ *k*<*N*. (*w* − *z*) ^ *k* ∗ *f u* / (*u* − *z*) ^ *Suc k*) ∗ (*u* − *w*) − *f u*)
= *norm* ((∑ *k*<*N*. (((*w* − *z*) / (*u* − *z*)) ^ *k*)) ∗ *f u* ∗ (*u* − *w*) / (*u* − *z*) − *f u*)
unfolding *setsum-left-distrib setsum-divide-distrib power-divide* **by** (*simp add*: *algebra-simps*)
also have ... = *norm* ((((*w* − *z*) / (*u* − *z*)) ^ *N* − *1*) ∗ (*u* − *w*) / (((*w* − *z*) / (*u* − *z*) − *1*) ∗ (*u* − *z*)) − *1*) ∗ *norm* (*f u*)
using ‹*0* < *B*›
apply (*auto simp*: *geometric-sum* [*OF wzu-not1*])
apply (*simp add*: *field-simps norm-mult* [*symmetric*])
done
also have ... = *norm* ((*u*−*z*) ^ *N* ∗ (*w* − *u*) − ((*w* − *z*) ^ *N* − (*u*−*z*) ^ *N*) ∗ (*u*−*w*)) / (*r* ^ *N* ∗ *norm* (*u*−*w*)) ∗ *norm* (*f u*)
using ‹*0* < *r*› *r* **by** (*simp add*: *divide-simps norm-mult norm-divide norm-power dist-norm norm-minus-commute*)
also have ... = *norm* ((*w* − *z*) ^ *N* ∗ (*w* − *u*)) / (*r* ^ *N* ∗ *norm* (*u* − *w*)) ∗ *norm* (*f u*)
by (*simp add*: *algebra-simps*)
also have ... = *norm* (*w* − *z*) ^ *N* ∗ *norm* (*f u*) / *r* ^ *N*
by (*simp add*: *norm-mult norm-power norm-minus-commute*)
also have ... ≤ (((*r* − *k*)/*r*) ^*N*) ∗ *B*
using ‹*0* < *r*› *w k*
apply (*simp add*: *divide-simps*)
apply (*rule mult-mono* [*OF power-mono*])
apply (*auto simp*: *norm-divide wz-eq norm-power dist-norm norm-minus-commute B r*)
done
also have ... < *e* ∗ *k*
using ‹*0* < *B*› *N* **by** (*simp add*: *divide-simps*)
also have ... ≤ *e* ∗ *norm* (*u* − *w*)
using *r kle* ‹*0* < *e*› **by** (*simp add*: *dist-commute dist-norm*)
finally show *?thesis*
by (*simp add*: *divide-simps norm-divide del*: *power-Suc*)
qed
with ‹*0* < *r*› **show** *?thesis*
by (*auto simp*: *mult-ac less-imp-le eventually-sequentially Ball-def*)
qed
have *eq*: ∀ *F x in sequentially*.
contour-integral (*circlepath z r*) (λ*u*. ∑ *k*<*x*. (*w* − *z*) ^ *k* ∗ (*f u* / (*u* − *z*) ^ *Suc k*)) =
(∑ *k*<*x*. *contour-integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *z*) ^ *Suc k*) ∗ (*w* − *z*) ^ *k*)
apply (*rule eventuallyI*)
apply (*subst contour-integral-setsum*, *simp*)
using *contour-integrable-lmul* [*OF cint*, *of* (*w* − *z*) ^ *a* **for** *a*] **apply** (*simp add*: *field-simps*)
apply (*simp only*: *contour-integral-lmul cint algebra-simps*)

done
have (λ*k*. *contour-integral* (*circlepath z r*) (λ*u*. *f u*/(*u* − *z*) ^(*Suc k*)) ∗ (*w* − *z*) ^*k*)
sums contour-integral (*circlepath z r*) (λ*u*. *f u*/(*u* − *w*))
unfolding *sums-def*
apply (*rule Lim-transform-eventually* [*OF eq*])
apply (*rule contour-integral-uniform-limit-circlepath* [*OF eventuallyI* ∗])
apply (*rule contour-integrable-setsum*, *simp*)
apply (*rule contour-integrable-lmul*)
apply (*rule Cauchy-higher-derivative-integral-circlepath* [*OF contf holf*])
using ‹*0* < *r*›
apply *auto*
done
then have (λ*k*. *contour-integral* (*circlepath z r*) (λ*u*. *f u*/(*u* − *z*) ^(*Suc k*)) ∗ (*w* − *z*) ^*k*)
sums (*2* ∗ *of-real pi* ∗ *ii* ∗ *f w*)
using *w* **by** (*auto simp*: *dist-commute dist-norm contour-integral-unique* [*OF Cauchy-integral-circlepath-simple* [*OF holfc*]])
then have (λ*k*. *contour-integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *z*) ^ *Suc k*) ∗ (*w* − *z*) ^*k* / (i ∗ (*of-real pi* ∗ *2*)))
sums ((*2* ∗ *of-real pi* ∗ *ii* ∗ *f w*) / (i ∗ (*complex-of-real pi* ∗ *2*)))
by (*rule sums-divide*)
then have (λ*n*. (*w* − *z*) ^ *n* ∗ *contour-integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *z*) ^ *Suc n*) / (i ∗ (*of-real pi* ∗ *2*)))
sums f w
by (*simp add*: *field-simps*)
then show *?thesis*
by (*simp add*: *field-simps* ‹*0* < *r*› *Cauchy-higher-derivative-integral-circlepath* [*OF contf holf*])
qed

52.40 The Liouville theorem and the Fundamental Theorem of Algebra.

These weak Liouville versions don't even need the derivative formula.

lemma *Liouville-weak-0*:
assumes *holf*: *f holomorphic-on UNIV* **and** *inf*: (*f* ⟶ *0*) *at-infinity*
shows *f z* = *0*
proof (*rule ccontr*)
assume *fz*: *f z* ≠ *0*
with *inf* [*unfolded Lim-at-infinity*, *rule-format*, *of norm*(*f z*)/*2*]
obtain *B* **where** *B*: ⋀*x*. *B* ≤ *cmod x* ⟹ *norm* (*f x*) ∗ *2* < *cmod* (*f z*)
by (*auto simp*: *dist-norm*)
def *R* ≡ *1* + |*B*| + *norm z*
have *R* > *0* **unfolding** *R-def* **by** (*meson abs-add-one-gt-zero le-less-trans less-add-same-cancel2 norm-ge-zero*)
have ∗: ((λ*u*. *f u* / (*u* − *z*)) *has-contour-integral* *2* ∗ *complex-of-real pi* ∗ i ∗ *f z*) (*circlepath z R*)
apply (*rule Cauchy-integral-circlepath*)

```
   using ‹R > 0› apply (auto intro: holomorphic-on-subset [OF holf] holomorphic-on-imp-continuous-on)+
    done
  have cmod (x − z) = R ⟹ cmod (f x) * 2 ≤ cmod (f z) for x
    apply (simp add: R-def)
    apply (rule less-imp-le)
    apply (rule B)
    using norm-triangle-ineq4 [of x z]
    apply (auto simp:)
    done
  with ‹R > 0› fz show False
    using has-contour-integral-bound-circlepath [OF *, of norm(f z)/2/R]
    by (auto simp: norm-mult norm-divide divide-simps)
qed

proposition Liouville-weak:
  assumes f holomorphic-on UNIV and (f ⟶ l) at-infinity
    shows f z = l
  using Liouville-weak-0 [of λz. f z − l]
  by (simp add: assms holomorphic-on-const holomorphic-on-diff LIM-zero)


proposition Liouville-weak-inverse:
  assumes f holomorphic-on UNIV and unbounded: ⋀B. eventually (λx. norm (f
x) ≥ B) at-infinity
    obtains z where f z = 0
proof −
  { assume f: ⋀z. f z ≠ 0
    have 1: (λx. 1 / f x) holomorphic-on UNIV
      by (simp add: holomorphic-on-divide holomorphic-on-const assms f)
    have 2: ((λx. 1 / f x) ⟶ 0) at-infinity
      apply (rule tendstoI [OF eventually-mono])
      apply (rule-tac B=2/e in unbounded)
      apply (simp add: dist-norm norm-divide divide-simps mult-ac)
      done
    have False
      using Liouville-weak-0 [OF 1 2] f by simp
  }
  then show ?thesis
    using that by blast
qed
```

In particular we get the Fundamental Theorem of Algebra.

```
theorem fundamental-theorem-of-algebra:
    fixes a :: nat ⇒ complex
  assumes a 0 = 0 ∨ (∃ i ∈ {1..n}. a i ≠ 0)
  obtains z where (∑ i≤n. a i * z^i) = 0
using assms
proof (elim disjE bexE)
  assume a 0 = 0 then show ?thesis
```

by (*auto simp*: *that* [*of 0*])
next
fix *i*
assume *i*: $i \in \{1..n\}$ **and** *nz*: $a\ i \neq 0$
have *1*: $(\lambda z.\ \sum i{\leq}n.\ a\ i * z\hat{}i)$ *holomorphic-on UNIV*
by (*rule holomorphic-intros*)+
show *?thesis*
apply (*rule Liouville-weak-inverse* [*OF 1*])
apply (*rule polyfun-extremal*)
apply (*rule nz*)
using *i that*
apply (*auto simp*:)
done
qed

52.41 Weierstrass convergence theorem.

proposition *holomorphic-uniform-limit*:
assumes *cont*: *eventually* $(\lambda n.$ *continuous-on* $(cball\ z\ r)\ (f\ n) \wedge (f\ n)$ *holomorphic-on ball z r*$)$ *F*
and *lim*: $\bigwedge e.\ 0 < e \Longrightarrow$ *eventually* $(\lambda n.\ \forall x \in cball\ z\ r.\ norm(f\ n\ x - g\ x) < e)$ *F*
and *F*: $\sim$ *trivial-limit F*
obtains *continuous-on* $(cball\ z\ r)\ g$ *g holomorphic-on ball z r*
proof (*cases r 0*::*real rule*: *linorder-cases*)
case *less* **then show** *?thesis* **by** (*force simp add*: *ball-empty less-imp-le continuous-on-def holomorphic-on-def intro*: *that*)
next
case *equal* **then show** *?thesis*
by (*force simp add*: *holomorphic-on-def continuous-on-sing intro*: *that*)
next
case *greater*
have *contg*: *continuous-on* $(cball\ z\ r)\ g$
using *cont*
by (*fastforce simp*: *eventually-conj-iff dist-norm intro*: *eventually-mono* [*OF lim*] *continuous-uniform-limit* [*OF F*])
have *1*: *continuous-on* $(path\text{-}image\ (circlepath\ z\ r))\ (\lambda x.\ 1\ /\ (2 * complex\text{-}of\text{-}real\ pi * \mathrm{i}) * g\ x)$
apply (*rule continuous-intros continuous-on-subset* [*OF contg*])+
using ‹*0 < r*› **by** *auto*
have *2*: $((\lambda u.\ 1\ /\ (2 * of\text{-}real\ pi * \mathrm{i}) * g\ u\ /\ (u - w)\ \hat{}\ 1)$ *has-contour-integral g w*$)$ $(circlepath\ z\ r)$
if *w*: $w \in ball\ z\ r$ **for** *w*
proof −
def $d \equiv (r - norm(w - z))$
have $0 < d\ \ d \leq r$ **using** *w* **by** (*auto simp*: *norm-minus-commute d-def dist-norm*)
have *dle*: $\bigwedge u.\ cmod\ (z - u) = r \Longrightarrow d \leq cmod\ (u - w)$
unfolding *d-def* **by** (*metis add-diff-eq diff-add-cancel norm-diff-ineq norm-minus-commute*)

have *ev-int*: $\forall_F\ n$ *in* F. $(\lambda u.\ f\ n\ u\ /\ (u - w))$ *contour-integrable-on circlepath z r*
apply (*rule eventually-mono* [*OF cont*])
using *w*
apply (*auto intro*: *Cauchy-higher-derivative-integral-circlepath* [**where** *k*=*0*, *simplified*])
done
have *ev-less*: $\forall_F\ n$ *in* F. $\forall x \in$ *path-image* (*circlepath z r*). *cmod* $(f\ n\ x\ /\ (x - w) - g\ x\ /\ (x - w)) < e$
if $e > 0$ **for** e
using *greater* ‹*0 < d*› ‹*0 < e*›
apply (*simp add*: *norm-divide diff-divide-distrib* [*symmetric*] *divide-simps*)
apply (*rule-tac e1*=$e * d$ **in** *eventually-mono* [*OF lim*])
apply (*force simp*: *dist-norm intro*: *dle mult-left-mono less-le-trans*)+
done
have *g-cint*: $(\lambda u.\ g\ u/(u - w))$ *contour-integrable-on circlepath z r*
by (*rule contour-integral-uniform-limit-circlepath* [*OF ev-int ev-less F* ‹*0 < r*›])
have *cif-tends-cig*: $((\lambda n.$ *contour-integral*(*circlepath z r*) $(\lambda u.\ f\ n\ u\ /\ (u - w))) \longrightarrow$ *contour-integral*(*circlepath z r*) $(\lambda u.\ g\ u/(u - w)))\ F$
by (*rule contour-integral-uniform-limit-circlepath* [*OF ev-int ev-less F* ‹*0 < r*›])
have *f-tends-cig*: $((\lambda n.\ 2 * of\text{-}real\ pi * ii * f\ n\ w) \longrightarrow$ *contour-integral* (*circlepath z r*) $(\lambda u.\ g\ u\ /\ (u - w)))\ F$
apply (*rule Lim-transform-eventually* [**where** $f = \lambda n.$ *contour-integral* (*circlepath z r*) $(\lambda u.\ f\ n\ u/(u - w))$])
apply (*rule eventually-mono* [*OF cont*])
apply (*rule contour-integral-unique* [*OF Cauchy-integral-circlepath*])
using *w*
apply (*auto simp*: *norm-minus-commute dist-norm cif-tends-cig*)
done
have $((\lambda n.\ 2 * of\text{-}real\ pi * \mathrm{i} * f\ n\ w) \longrightarrow 2 * of\text{-}real\ pi * \mathrm{i} * g\ w)\ F$
apply (*rule tendsto-mult-left* [*OF tendstoI*])
apply (*rule eventually-mono* [*OF lim*], *assumption*)
using *w*
apply (*force simp add*: *dist-norm*)
done
then have $((\lambda u.\ g\ u\ /\ (u - w))$ *has-contour-integral* $2 * of\text{-}real\ pi * \mathrm{i} * g\ w)$ (*circlepath z r*)
using *has-contour-integral-integral* [*OF g-cint*] *tendsto-unique* [*OF F f-tends-cig*] *w*
by (*force simp add*: *dist-norm*)
then have $((\lambda u.\ g\ u\ /\ (2 * of\text{-}real\ pi * \mathrm{i} * (u - w)))$ *has-contour-integral g w*) (*circlepath z r*)
using *has-contour-integral-div* [**where** $c = 2 * of\text{-}real\ pi * \mathrm{i}$]
by (*force simp add*: *field-simps*)
then show *?thesis*
by (*simp add*: *dist-norm*)
qed

show *?thesis*
using *Cauchy-next-derivative-circlepath*(*2*) [*OF 1 2*, *simplified*]
by (*fastforce simp add*: *holomorphic-on-open contg intro*: *that*)
qed

Version showing that the limit is the limit of the derivatives.

proposition *has-complex-derivative-uniform-limit*:
fixes *z*::*complex*
assumes *cont*: *eventually* (λ*n*. *continuous-on* (*cball z r*) (*f n*) ∧
(∀ *w* ∈ *ball z r*. ((*f n*) *has-field-derivative* (*f′ n w*)) (*at w*))) *F*
and *lim*: ⋀*e*. *0* < *e* ⟹ *eventually* (λ*n*. ∀ *x* ∈ *cball z r*. *norm*(*f n x* − *g x*) < *e*) *F*
and *F*: ~ *trivial-limit F* **and** *0* < *r*
obtains *g′* **where**
continuous-on (*cball z r*) *g*
⋀*w*. *w* ∈ *ball z r* ⟹ (*g has-field-derivative* (*g′ w*)) (*at w*) ∧ ((λ*n*. *f′ n w*) ⟶ *g′ w*) *F*
proof −
let *?conint* = *contour-integral* (*circlepath z r*)
have *g*: *continuous-on* (*cball z r*) *g g holomorphic-on ball z r*
by (*rule holomorphic-uniform-limit* [*OF eventually-mono* [*OF cont*] *lim F*];
auto simp: *holomorphic-on-open complex-differentiable-def*)+
then obtain *g′* **where** *g′*: ⋀*x*. *x* ∈ *ball z r* ⟹ (*g has-field-derivative g′ x*) (*at x*)
using *DERIV-deriv-iff-has-field-derivative*
by (*fastforce simp add*: *holomorphic-on-open*)
then have *derg*: ⋀*x*. *x* ∈ *ball z r* ⟹ *deriv g x* = *g′ x*
by (*simp add*: *DERIV-imp-deriv*)
have *tends-f′n-g′*: ((λ*n*. *f′ n w*) ⟶ *g′ w*) *F* **if** *w*: *w* ∈ *ball z r* **for** *w*
proof −
have *eq-f′*: *?conint* (λ*x*. *f n x* / (*x* − *w*)²) − *?conint* (λ*x*. *g x* / (*x* − *w*)²) = (*f′ n w* − *g′ w*) ∗ (*2* ∗ *of-real pi* ∗ i)
if *cont-fn*: *continuous-on* (*cball z r*) (*f n*)
and *fnd*: ⋀*w*. *w* ∈ *ball z r* ⟹ (*f n has-field-derivative f′ n w*) (*at w*)
for *n*
proof −
have *hol-fn*: *f n holomorphic-on ball z r*
using *fnd* **by** (*force simp add*: *holomorphic-on-open*)
have (*f n has-field-derivative 1* / (*2* ∗ *of-real pi* ∗ i) ∗ *?conint* (λ*u*. *f n u* / (*u* − *w*)²)) (*at w*)
by (*rule Cauchy-derivative-integral-circlepath* [*OF cont-fn hol-fn w*])
then have *f′*: *f′ n w* = *1* / (*2* ∗ *of-real pi* ∗ i) ∗ *?conint* (λ*u*. *f n u* / (*u* − *w*)²)
using *DERIV-unique* [*OF fnd*] *w* **by** *blast*
show *?thesis*
by (*simp add*: *f′ Cauchy-contour-integral-circlepath-2* [*OF g w*] *derg* [*OF w*] *divide-simps*)
qed

def $d \equiv (r - norm(w - z))\ \hat{}\ 2$
have $d > 0$
using w **by** (*simp add*: *dist-commute dist-norm d-def*)
have *dle*: $\bigwedge y.\ r = cmod\ (z - y) \Longrightarrow d \leq cmod\ ((y - w)^2)$
apply (*simp add*: *d-def norm-power*)
apply (*rule power-mono*)
apply (*metis add-diff-eq diff-add-cancel norm-diff-ineq norm-minus-commute*)
apply (*metis diff-ge-0-iff-ge dist-commute dist-norm less-eq-real-def mem-ball w*)
done
have *1*: $\forall_F\ n\ in\ F.\ (\lambda x.\ f\ n\ x\ /\ (x - w)^2)$ *contour-integrable-on circlepath z r*
by (*force simp add*: *holomorphic-on-open intro*: *w Cauchy-derivative-integral-circlepath eventually-mono* [*OF cont*])
have *2*: $0 < e \Longrightarrow \forall_F\ n\ in\ F.\ \forall x \in$ *path-image* (*circlepath z r*). *cmod* (*f n x / (x - w)*^{2} *- g x / (x - w)*^{2}) $< e$ **for** e
using ‹$r > 0$›
apply (*simp add*: *diff-divide-distrib* [*symmetric*] *norm-divide divide-simps sphere-def*)
apply (*rule eventually-mono* [*OF lim*, *of e*∗*d*])
apply (*simp add*: ‹$0 < d$›)
apply (*force simp add*: *dist-norm dle intro*: *less-le-trans*)
done
have $((\lambda n.\ contour\text{-}integral\ (circlepath\ z\ r)\ (\lambda x.\ f\ n\ x\ /\ (x - w)^2))$
$\longrightarrow\ contour\text{-}integral\ (circlepath\ z\ r)\ ((\lambda x.\ g\ x\ /\ (x - w)^2)))\ F$
by (*rule Cauchy-Integral-Thm.contour-integral-uniform-limit-circlepath* [*OF 1 2 F* ‹$0 < r$›])
then have *tendsto-0*: $((\lambda n.\ 1\ /\ (2 * of\text{-}real\ pi * \mathrm{i}) * (?conint\ (\lambda x.\ f\ n\ x\ /\ (x - w)^2) - ?conint\ (\lambda x.\ g\ x\ /\ (x - w)^2))) \longrightarrow 0)\ F$
using *Lim-null* **by** (*force intro*!: *tendsto-mult-right-zero*)
have $((\lambda n.\ f'\ n\ w - g'\ w) \longrightarrow 0)\ F$
apply (*rule Lim-transform-eventually* [*OF - tendsto-0*])
apply (*force simp add*: *divide-simps intro*: *eq-f' eventually-mono* [*OF cont*])
done
then show *?thesis* **using** *Lim-null* **by** *blast*
qed
obtain g' **where** $\bigwedge w.\ w \in ball\ z\ r \Longrightarrow (g\ has\text{-}field\text{-}derivative\ (g'\ w))\ (at\ w) \wedge ((\lambda n.\ f'\ n\ w) \longrightarrow g'\ w)\ F$
by (*blast intro*: *tends-f'n-g' g'*)
then show *?thesis* **using** g
using *that* **by** *blast*
qed

52.42 Some more simple/convenient versions for applications.

lemma *holomorphic-uniform-sequence*:
assumes s: *open s*
and *hol-fn*: $\bigwedge n.\ (f\ n)\ holomorphic\text{-}on\ s$
and *to-g*: $\bigwedge x.\ x \in s \Longrightarrow \exists d.\ 0 < d \wedge cball\ x\ d \subseteq s \wedge$

```
                    (∀ e. 0 < e ⟶ eventually (λn. ∀ y ∈ cball x d. norm(f n y
− g y) < e) sequentially)
  shows g holomorphic-on s
proof −
  have ∃ f'. (g has-field-derivative f') (at z) if z ∈ s for z
  proof −
    obtain r where 0 < r and r: cball z r ⊆ s
            and fg: ∀ e. 0 < e ⟶ eventually (λn. ∀ y ∈ cball z r. norm(f n y −
g y) < e) sequentially
      using to-g [OF ‹z ∈ s›] by blast
   have ∗: ∀ F n in sequentially. continuous-on (cball z r) (f n) ∧ f n holomorphic-on
ball z r
      apply (intro eventuallyI conjI)
      using hol-fn holomorphic-on-imp-continuous-on holomorphic-on-subset r ap-
ply blast
      apply (metis hol-fn holomorphic-on-subset interior-cball interior-subset r)
      done
    show ?thesis
      apply (rule holomorphic-uniform-limit [OF ∗])
      using ‹0 < r› centre-in-ball fg
      apply (auto simp: holomorphic-on-open)
      done
  qed
  with s show ?thesis
    by (simp add: holomorphic-on-open)
qed

lemma has-complex-derivative-uniform-sequence:
  fixes s :: complex set
  assumes s: open s
      and hfd: ⋀n x. x ∈ s ⟹ ((f n) has-field-derivative f' n x) (at x)
      and to-g: ⋀x. x ∈ s
            ⟹ ∃ d. 0 < d ∧ cball x d ⊆ s ∧
                  (∀ e. 0 < e ⟶ eventually (λn. ∀ y ∈ cball x d. norm(f n y − g
y) < e) sequentially)
  shows ∃ g'. ∀ x ∈ s. (g has-field-derivative g' x) (at x) ∧ ((λn. f' n x) ⟶ g'
x) sequentially
proof −
  have y: ∃ y. (g has-field-derivative y) (at z) ∧ (λn. f' n z) ⟶ y if z ∈ s for
z
  proof −
    obtain r where 0 < r and r: cball z r ⊆ s
            and fg: ∀ e. 0 < e ⟶ eventually (λn. ∀ y ∈ cball z r. norm(f n y −
g y) < e) sequentially
      using to-g [OF ‹z ∈ s›] by blast
    have ∗: ∀ F n in sequentially. continuous-on (cball z r) (f n) ∧
                          (∀ w ∈ ball z r. ((f n) has-field-derivative (f' n w)) (at
w))
      apply (intro eventuallyI conjI)
```

apply (*meson hfd holomorphic-on-imp-continuous-on holomorphic-on-open holomorphic-on-subset r s*)
using *ball-subset-cball hfd r* **apply** *blast*
done
show *?thesis*
apply (*rule has-complex-derivative-uniform-limit* [*OF* ∗, *of g*])
using ‹*0 < r*› *centre-in-ball fg*
apply *force+*
done
qed
show *?thesis*
by (*rule bchoice*) (*blast intro*: *y*)
qed

52.43 On analytic functions defined by a series.

lemma *series-and-derivative-comparison*:
fixes *s* :: *complex set*
assumes *s*: *open s*
and *h*: *summable h*
and *hfd*: ⋀*n x*. *x* ∈ *s* ⟹ (*f n has-field-derivative f' n x*) (*at x*)
and *to-g*: ⋀*n x*. ⟦*N* ≤ *n*; *x* ∈ *s*⟧ ⟹ *norm*(*f n x*) ≤ *h n*
obtains *g g'* **where** ∀ *x* ∈ *s*. ((λ*n*. *f n x*) *sums g x*) ∧ ((λ*n*. *f' n x*) *sums g' x*) ∧ (*g has-field-derivative g' x*) (*at x*)
proof −
obtain *g* **where** *g*: ⋀*e*. *e*>*0* ⟹ ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *x* ∈ *s* ⟶ *dist* (∑ *n*<*n*. *f n x*) (*g x*) < *e*
using *series-comparison-uniform* [*OF h to-g*, *of N s*] **by** *force*
have ∗: ∃ *d*>*0*. *cball x d* ⊆ *s* ∧ (∀ *e*>*0*. ∀$_F$ *n in sequentially*. ∀ *y*∈*cball x d*. *cmod* ((∑ *a*<*n*. *f a y*) − *g y*) < *e*)
if *x* ∈ *s* **for** *x*
proof −
obtain *d* **where** *d*>*0* **and** *d*: *cball x d* ⊆ *s*
using *open-contains-cball* [*of s*] ‹*x* ∈ *s*› *s* **by** *blast*
then show *?thesis*
apply (*rule-tac x*=*d* **in** *exI*)
apply (*auto simp*: *dist-norm eventually-sequentially*)
apply (*metis g contra-subsetD dist-norm*)
done
qed
have (∀ *x*∈*s*. (λ*n*. ∑ *i*<*n*. *f i x*) ⟶⟶ *g x*)
using *g* **by** (*force simp add*: *lim-sequentially*)
moreover have ∃ *g'*. ∀ *x*∈*s*. (*g has-field-derivative g' x*) (*at x*) ∧ (λ*n*. ∑ *i*<*n*. *f' i x*) ⟶⟶ *g' x*
by (*rule has-complex-derivative-uniform-sequence* [*OF s*]) (*auto intro*: ∗ *hfd DERIV-setsum*)+
ultimately show *?thesis*
by (*force simp add*: *sums-def conj-commute intro*: *that*)
qed

A version where we only have local uniform/comparative convergence.

lemma *series-and-derivative-comparison-local*:
fixes s :: *complex set*
assumes s: *open s*
and *hfd*: $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n\ \textit{has-field-derivative}\ f'\ n\ x)\ (\textit{at}\ x)$
and *to-g*: $\bigwedge x.\ x \in s \Longrightarrow$
$\exists\, d\ h\ N.\ 0 < d \wedge \textit{summable}\ h \wedge (\forall\, n\ y.\ N \leq n \wedge y \in \textit{ball}\ x\ d \longrightarrow \textit{norm}(f\ n\ y) \leq h\ n)$
shows $\exists\, g\ g'.\ \forall\, x \in s.\ ((\lambda n.\ f\ n\ x)\ \textit{sums}\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ \textit{sums}\ g'\ x) \wedge (g\ \textit{has-field-derivative}\ g'\ x)\ (\textit{at}\ x)$
proof −
have $\exists\, y.\ (\lambda n.\ f\ n\ z)\ \textit{sums}\ (\sum n.\ f\ n\ z) \wedge (\lambda n.\ f'\ n\ z)\ \textit{sums}\ y \wedge ((\lambda x.\ \sum n.\ f\ n\ x)\ \textit{has-field-derivative}\ y)\ (\textit{at}\ z)$
if $z \in s$ **for** z
proof −
obtain $d\ h\ N$ **where** $0 < d$ *summable h* **and** *le-h*: $\bigwedge n\ y.\ [\![N \leq n;\ y \in \textit{ball}\ z\ d]\!] \Longrightarrow \textit{norm}(f\ n\ y) \leq h\ n$
using *to-g* ‹$z \in s$› **by** *blast*
then obtain r **where** $r{>}0$ **and** r: *ball z r* $\subseteq$ *ball z d* $\cap$ s **using** ‹$z \in s$› s
by (*metis Int-iff open-ball centre-in-ball open-Int open-contains-ball-eq*)
have *1*: *open* (*ball z d* $\cap$ s)
by (*simp add*: *open-Int s*)
have *2*: $\bigwedge n\ x.\ x \in \textit{ball}\ z\ d \cap s \Longrightarrow (f\ n\ \textit{has-field-derivative}\ f'\ n\ x)\ (\textit{at}\ x)$
by (*auto simp*: *hfd*)
obtain $g\ g'$ **where** *gg'*: $\forall\, x \in \textit{ball}\ z\ d \cap s.\ ((\lambda n.\ f\ n\ x)\ \textit{sums}\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ \textit{sums}\ g'\ x) \wedge (g\ \textit{has-field-derivative}\ g'\ x)\ (\textit{at}\ x)$
by (*auto intro*: *le-h series-and-derivative-comparison* [*OF 1* ‹*summable h*› *hfd*])
then have $(\lambda n.\ f'\ n\ z)\ \textit{sums}\ g'\ z$
by (*meson* ‹$0 < r$› *centre-in-ball contra-subsetD r*)
moreover have $(\lambda n.\ f\ n\ z)\ \textit{sums}\ (\sum n.\ f\ n\ z)$
by (*metis summable-comparison-test' summable-sums centre-in-ball* ‹$0 < d$› ‹*summable h*› *le-h*)
moreover have $((\lambda x.\ \sum n.\ f\ n\ x)\ \textit{has-field-derivative}\ g'\ z)\ (\textit{at}\ z)$
apply (*rule-tac f=g* **in** *DERIV-transform-at* [*OF* - ‹$0 < r$›])
apply (*simp add*: *gg'* ‹$z \in s$› ‹$0 < d$›)
apply (*metis* (*full-types*) *contra-subsetD dist-commute gg' mem-ball r sums-unique*)
done
ultimately show *?thesis* **by** *auto*
qed
then show *?thesis*
by (*rule-tac x*=$\lambda x.\ \textit{suminf}\ (\lambda n.\ f\ n\ x)$ **in** *exI*) *meson*
qed

Sometimes convenient to compare with a complex series of positive reals. (?)

lemma *series-and-derivative-comparison-complex*:
fixes s :: *complex set*

assumes *s*: *open s*
and *hfd*: $\bigwedge n\ x.\ x \in s \Longrightarrow (f\ n\ \textit{has-field-derivative}\ f'\ n\ x)\ (\textit{at}\ x)$
and *to-g*: $\bigwedge x.\ x \in s \Longrightarrow$
$\exists\, d\ h\ N.\ 0 < d \wedge \textit{summable}\ h \wedge \textit{range}\ h \subseteq \textit{nonneg-Reals} \wedge (\forall\, n\ y.\ N \leq n \wedge y \in \textit{ball}\ x\ d \longrightarrow \textit{cmod}(f\ n\ y) \leq \textit{cmod}\ (h\ n))$
shows $\exists\, g\ g'.\ \forall\, x \in s.\ ((\lambda n.\ f\ n\ x)\ \textit{sums}\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ \textit{sums}\ g'\ x) \wedge (g\ \textit{has-field-derivative}\ g'\ x)\ (\textit{at}\ x)$
apply (*rule series-and-derivative-comparison-local* [*OF s hfd*], *assumption*)
apply (*frule to-g*)
apply (*erule ex-forward*)
apply (*erule exE*)
apply (*rule-tac x=Re o h* **in** *exI*)
apply (*erule ex-forward*)
apply (*simp add*: *summable-Re o-def*)
apply (*elim conjE all-forward*)
apply (*simp add*: *nonneg-Reals-cmod-eq-Re image-subset-iff*)
done

In particular, a power series is analytic inside circle of convergence.

lemma *power-series-and-derivative-0*:
fixes *a* :: *nat* ⇒ *complex* **and** *r*::*real*
assumes *summable* $(\lambda n.\ a\ n * r\hat{}n)$
shows $\exists\, g\ g'.\ \forall\, z.\ \textit{cmod}\ z < r \longrightarrow$
$((\lambda n.\ a\ n * z\hat{}n)\ \textit{sums}\ g\ z) \wedge ((\lambda n.\ \textit{of-nat}\ n * a\ n * z\hat{}(n - 1))\ \textit{sums}\ g'\ z) \wedge (g\ \textit{has-field-derivative}\ g'\ z)\ (\textit{at}\ z)$
proof (*cases* $0 < r$)
case *True*
have *der*: $\bigwedge n\ z.\ ((\lambda x.\ a\ n * x\ \hat{}\ n)\ \textit{has-field-derivative}\ \textit{of-nat}\ n * a\ n * z\ \hat{}\ (n - 1))\ (\textit{at}\ z)$
by (*rule derivative-eq-intros* | *simp*)+
have *y-le*: $[\![\textit{cmod}\ (z - y) * 2 < r - \textit{cmod}\ z]\!] \Longrightarrow \textit{cmod}\ y \leq \textit{cmod}\ (\textit{of-real}\ r + \textit{of-real}\ (\textit{cmod}\ z)) / 2$ **for** *z y*
using ‹$r > 0$›
apply (*auto simp*: *algebra-simps norm-mult norm-divide norm-power of-real-add* [*symmetric*] *simp del*: *of-real-add*)
using *norm-triangle-ineq2* [*of y z*]
apply (*simp only*: *diff-le-eq norm-minus-commute mult-2*)
done
have *summable* $(\lambda n.\ a\ n * \textit{complex-of-real}\ r\ \hat{}\ n)$
using *assms* ‹$r > 0$› **by** *simp*
moreover have $\bigwedge z.\ \textit{cmod}\ z < r \Longrightarrow \textit{cmod}\ ((\textit{of-real}\ r + \textit{of-real}\ (\textit{cmod}\ z)) / 2) < \textit{cmod}\ (\textit{of-real}\ r)$
using ‹$r > 0$›
by (*simp add*: *of-real-add* [*symmetric*] *del*: *of-real-add*)
ultimately have *sum*: $\bigwedge z.\ \textit{cmod}\ z < r \Longrightarrow \textit{summable}\ (\lambda n.\ \textit{of-real}\ (\textit{cmod}\ (a\ n)) * ((\textit{of-real}\ r + \textit{complex-of-real}\ (\textit{cmod}\ z)) / 2)\ \hat{}\ n)$
by (*rule power-series-conv-imp-absconv-weak*)
have $\exists\, g\ g'.\ \forall\, z \in \textit{ball}\ 0\ r.\ (\lambda n.\ (a\ n) * z\ \hat{}\ n)\ \textit{sums}\ g\ z \wedge$
$(\lambda n.\ \textit{of-nat}\ n * (a\ n) * z\ \hat{}\ (n - 1))\ \textit{sums}\ g'\ z \wedge (g\ \textit{has-field-derivative}$

```
g' z) (at z)
      apply (rule series-and-derivative-comparison-complex [OF open-ball der])
      apply (rule-tac x=(r - norm z)/2 in exI)
      apply (simp add: dist-norm)
      apply (rule-tac x=λn. of-real(norm(a n)*((r + norm z)/2) ^n) in exI)
      using ‹r > 0›
      apply (auto simp: sum nonneg-Reals-divide-I)
      apply (rule-tac x=0 in exI)
     apply (force simp: norm-mult norm-divide norm-power intro!: mult-left-mono
power-mono y-le)
      done
  then show ?thesis
    by (simp add: dist-0-norm ball-def)
next
  case False then show ?thesis
    apply (simp add: not-less)
    using less-le-trans norm-not-less-zero by blast
qed

proposition power-series-and-derivative:
  fixes a :: nat ⇒ complex and r::real
  assumes summable (λn. a n * r^n)
    obtains g g' where ∀ z ∈ ball w r.
            ((λn. a n * (z - w) ^ n) sums g z) ∧ ((λn. of-nat n * a n * (z - w) ^
(n - 1)) sums g' z) ∧
             (g has-field-derivative g' z) (at z)
  using power-series-and-derivative-0 [OF assms]
  apply clarify
  apply (rule-tac g=(λz. g(z - w)) in that)
  using DERIV-shift [where z=-w]
  apply (auto simp: norm-minus-commute Ball-def dist-norm)
  done

proposition power-series-holomorphic:
  assumes ⋀w. w ∈ ball z r ⟹ ((λn. a n*(w - z) ^n) sums f w)
    shows f holomorphic-on ball z r
proof -
  have ∃f'. (f has-field-derivative f') (at w) if w: dist z w < r for w
  proof -
    have inb: z + complex-of-real ((dist z w + r) / 2) ∈ ball z r
    proof -
      have wz: cmod (w - z) < r using w
        by (auto simp: divide-simps dist-norm norm-minus-commute)
      then have 0 ≤ r
        by (meson less-eq-real-def norm-ge-zero order-trans)
      show ?thesis
          using w by (simp add: dist-norm ‹0≤r› of-real-add [symmetric] del:
of-real-add)
    qed
```

```
    have sum: summable (λn. a n * of-real (((cmod (z - w) + r) / 2) ^ n))
      using assms [OF inb] by (force simp add: summable-def dist-norm)
    obtain g g' where gg': ⋀u. u ∈ ball z ((cmod (z - w) + r) / 2) ⟹
                           (λn. a n * (u - z) ^ n) sums g u ∧
                           (λn. of-nat n * a n * (u - z) ^ (n - 1)) sums g' u ∧ (g
has-field-derivative g' u) (at u)
      by (rule power-series-and-derivative [OF sum, of z]) fastforce
    have [simp]: g u = f u if cmod (u - w) < (r - cmod (z - w)) / 2 for u
    proof -
      have less: cmod (z - u) * 2 < cmod (z - w) + r
        using that dist-triangle2 [of z u w]
        by (simp add: dist-norm [symmetric] algebra-simps)
      show ?thesis
        apply (rule sums-unique2 [of λn. a n*(u - z) ^n])
        using gg' [of u] less w
        apply (auto simp: assms dist-norm)
        done
    qed
    show ?thesis
      apply (rule-tac x=g' w in exI)
     apply (rule DERIV-transform-at [where f=g and d=(r - norm(z - w))/2])
      using w gg' [of w]
      apply (auto simp: dist-norm)
      done
  qed
  then show ?thesis by (simp add: holomorphic-on-open)
qed

corollary holomorphic-iff-power-series:
    f holomorphic-on ball z r ⟷
     (∀ w ∈ ball z r. (λn. (deriv ^^ n) f z / (fact n) * (w - z) ^n) sums f w)
  apply (intro iffI ballI)
   using holomorphic-power-series  apply force
  apply (rule power-series-holomorphic [where a = λn. (deriv ^^ n) f z / (fact
n)])
  apply force
  done

corollary power-series-analytic:
    (⋀w. w ∈ ball z r ⟹ (λn. a n*(w - z) ^n) sums f w) ⟹ f analytic-on ball
z r
  by (force simp add: analytic-on-open intro!: power-series-holomorphic)

corollary analytic-iff-power-series:
    f analytic-on ball z r ⟷
     (∀ w ∈ ball z r. (λn. (deriv ^^ n) f z / (fact n) * (w - z) ^n) sums f w)
  by (simp add: analytic-on-open holomorphic-iff-power-series)
```

52.44 Equality between holomorphic functions, on open ball then connected set.

lemma *holomorphic-fun-eq-on-ball*:
⟦*f holomorphic-on ball z r*; *g holomorphic-on ball z r*;
w ∈ *ball z r*;
⋀*n*. (*deriv* ^^ *n*) *f z* = (*deriv* ^^ *n*) *g z*⟧
⟹ *f w* = *g w*
apply (*rule sums-unique2* [*of* λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*])
apply (*auto simp*: *holomorphic-iff-power-series*)
done

lemma *holomorphic-fun-eq-0-on-ball*:
⟦*f holomorphic-on ball z r*; *w* ∈ *ball z r*;
⋀*n*. (*deriv* ^^ *n*) *f z* = 0⟧
⟹ *f w* = 0
apply (*rule sums-unique2* [*of* λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*])
apply (*auto simp*: *holomorphic-iff-power-series*)
done

lemma *holomorphic-fun-eq-0-on-connected*:
assumes *holf*: *f holomorphic-on s* **and** *open s*
and *cons*: *connected s*
and *der*: ⋀*n*. (*deriv* ^^ *n*) *f z* = 0
and *z* ∈ *s w* ∈ *s*
shows *f w* = 0
proof −
have ∗: ⋀*x e*. ⟦ ∀ *xa*. (*deriv* ^^ *xa*) *f x* = 0; *ball x e* ⊆ *s*⟧
⟹ *ball x e* ⊆ (⋂ *n*. {*w* ∈ *s*. (*deriv* ^^ *n*) *f w* = 0})
apply *auto*
apply (*rule holomorphic-fun-eq-0-on-ball* [*OF holomorphic-higher-deriv*])
apply (*rule holomorphic-on-subset* [*OF holf*], *simp-all*)
by (*metis funpow-add o-apply*)
have *1*: *openin* (*subtopology euclidean s*) (⋂ *n*. {*w* ∈ *s*. (*deriv* ^^ *n*) *f w* = 0})
apply (*rule open-subset*, *force*)
using ‹*open s*›
apply (*simp add*: *open-contains-ball Ball-def*)
apply (*erule all-forward*)
using ∗ **by** *blast*
have *2*: *closedin* (*subtopology euclidean s*) (⋂ *n*. {*w* ∈ *s*. (*deriv* ^^ *n*) *f w* = 0})
using *assms*
by (*auto intro*: *continuous-closed-in-preimage-constant holomorphic-on-imp-continuous-on holomorphic-higher-deriv*)
obtain *e* **where** *e*>0 **and** *e*: *ball w e* ⊆ *s* **using** *openE* [*OF* ‹*open s*› ‹*w* ∈ *s*›] **.**
then have *holfb*: *f holomorphic-on ball w e*
using *holf holomorphic-on-subset* **by** *blast*
have *3*: (⋂ *n*. {*w* ∈ *s*. (*deriv* ^^ *n*) *f w* = 0}) = *s* ⟹ *f w* = 0
using ‹*e*>0› *e* **by** (*force intro*: *holomorphic-fun-eq-0-on-ball* [*OF holfb*])
show *?thesis*
using *cons der* ‹*z* ∈ *s*›

```
    apply (simp add: connected-clopen)
    apply (drule-tac x=⋂ n. {w ∈ s. (deriv ^^ n) f w = 0} in spec)
    apply (auto simp: 1 2 3)
    done
qed

lemma holomorphic-fun-eq-on-connected:
  assumes f holomorphic-on s g holomorphic-on s and open s  connected s
      and ⋀n. (deriv ^^ n) f z = (deriv ^^ n) g z
      and z ∈ s w ∈ s
    shows f w = g w
  apply (rule holomorphic-fun-eq-0-on-connected [of λx. f x − g x s z, simplified])
  apply (intro assms holomorphic-intros)
  using assms apply simp-all
  apply (subst higher-deriv-diff, auto)
  done

lemma holomorphic-fun-eq-const-on-connected:
  assumes holf: f holomorphic-on s and open s
      and cons: connected s
      and der: ⋀n. 0 < n ⟹ (deriv ^^ n) f z = 0
      and z ∈ s w ∈ s
    shows f w = f z
  apply (rule holomorphic-fun-eq-0-on-connected [of λw. f w − f z s z, simplified])
  apply (intro assms holomorphic-intros)
  using assms apply simp-all
  apply (subst higher-deriv-diff)
  apply (intro holomorphic-intros | simp)+
  done
```

52.45 Some basic lemmas about poles/singularities.

```
lemma pole-lemma:
  assumes holf: f holomorphic-on s and a: a ∈ interior s
    shows (λz. if z = a then deriv f a
               else (f z − f a) / (z − a)) holomorphic-on s (is ?F holomorphic-on
s)
proof −
  have F1: ?F complex-differentiable (at u within s) if u ∈ s u ≠ a for u
  proof −
    have fcd: f complex-differentiable at u within s
      using holf holomorphic-on-def by (simp add: ‹u ∈ s›)
    have cd: (λz. (f z − f a) / (z − a)) complex-differentiable at u within s
      by (rule fcd derivative-intros | simp add: that)+
    have 0 < dist a u using that dist-nz by blast
    then show ?thesis
      by (rule complex-differentiable-transform-within [OF - - - cd]) (auto simp: ‹u
∈ s›)
  qed
```

```
  have F2: ?F complex-differentiable at a if 0 < e ball a e ⊆ s for e
  proof −
    have holfb: f holomorphic-on ball a e
      by (rule holomorphic-on-subset [OF holf ‹ball a e ⊆ s›])
    have 2: ?F holomorphic-on ball a e − {a}
      apply (rule holomorphic-on-subset [where s = s − {a}])
     apply (simp add: holomorphic-on-def complex-differentiable-def [symmetric])
      using mem-ball that
      apply (auto intro: F1 complex-differentiable-within-subset)
      done
    have isCont (λz. if z = a then deriv f a else (f z − f a) / (z − a)) x
          if dist a x < e for x
    proof (cases x=a)
      case True then show ?thesis
      using holfb ‹0 < e›
     apply (simp add: holomorphic-on-open complex-differentiable-def [symmetric])
      apply (drule-tac x=a in bspec)
     apply (auto simp: DERIV-deriv-iff-complex-differentiable [symmetric] continuous-at
DERIV-iff2
                 elim: rev-iffD1 [OF - LIM-equal])
      done
    next
      case False with 2 that show ?thesis
        by (force simp: holomorphic-on-open open-Diff complex-differentiable-def
[symmetric] complex-differentiable-imp-continuous-at)
    qed
    then have 1: continuous-on (ball a e) ?F
      by (clarsimp simp: continuous-on-eq-continuous-at)
    have ?F holomorphic-on ball a e
      by (auto intro: no-isolated-singularity [OF 1 2])
    with that show ?thesis
      by (simp add: holomorphic-on-open complex-differentiable-def [symmetric]
                    complex-differentiable-at-within)
  qed
  show ?thesis
  proof
    fix x assume x ∈ s show ?F complex-differentiable at x within s
    proof (cases x=a)
      case True then show ?thesis
       using a by (auto simp: mem-interior intro: complex-differentiable-at-within
F2)
    next
      case False with F1 ‹x ∈ s›
      show ?thesis by blast
    qed
  qed
qed

proposition pole-theorem:
```

assumes *holg*: *g holomorphic-on s* **and** *a*: *a ∈ interior s*
and *eq*: *⋀z. z ∈ s − {a} ⟹ g z = (z − a) * f z*
shows *(λz. if z = a then deriv g a*
else f z − g a/(z − a)) holomorphic-on s
using *pole-lemma* [*OF holg a*]
by (*rule holomorphic-transform*) (*simp add*: *eq divide-simps*)

lemma *pole-lemma-open*:
assumes *f holomorphic-on s open s*
shows *(λz. if z = a then deriv f a else (f z − f a)/(z − a)) holomorphic-on s*
proof (*cases a ∈ s*)
case *True* **with** *assms interior-eq pole-lemma*
show *?thesis* **by** *fastforce*
next
case *False* **with** *assms* **show** *?thesis*
apply (*simp add*: *holomorphic-on-def complex-differentiable-def* [*symmetric*], *clarify*)
apply (*rule complex-differentiable-transform-within* [**where** *f* = *λz. (f z − f a)/(z − a)* **and** *d* = *1*])
apply (*rule derivative-intros* | *force*)+
done
qed

proposition *pole-theorem-open*:
assumes *holg*: *g holomorphic-on s* **and** *s*: *open s*
and *eq*: *⋀z. z ∈ s − {a} ⟹ g z = (z − a) * f z*
shows *(λz. if z = a then deriv g a*
else f z − g a/(z − a)) holomorphic-on s
using *pole-lemma-open* [*OF holg s*]
by (*rule holomorphic-transform*) (*auto simp*: *eq divide-simps*)

proposition *pole-theorem-0*:
assumes *holg*: *g holomorphic-on s* **and** *a*: *a ∈ interior s*
and *eq*: *⋀z. z ∈ s − {a} ⟹ g z = (z − a) * f z*
and [*simp*]: *f a = deriv g a g a = 0*
shows *f holomorphic-on s*
using *pole-theorem* [*OF holg a eq*]
by (*rule holomorphic-transform*) (*auto simp*: *eq divide-simps*)

proposition *pole-theorem-open-0*:
assumes *holg*: *g holomorphic-on s* **and** *s*: *open s*
and *eq*: *⋀z. z ∈ s − {a} ⟹ g z = (z − a) * f z*
and [*simp*]: *f a = deriv g a g a = 0*
shows *f holomorphic-on s*
using *pole-theorem-open* [*OF holg s eq*]
by (*rule holomorphic-transform*) (*auto simp*: *eq divide-simps*)

lemma *pole-theorem-analytic*:
assumes *g*: *g analytic-on s*

```
    and eq: ⋀z. z ∈ s
          ⟹ ∃ d. 0 < d ∧ (∀ w ∈ ball z d − {a}. g w = (w − a) ∗ f w)
  shows (λz. if z = a then deriv g a
             else f z − g a/(z − a)) analytic-on s
using g
apply (simp add: analytic-on-def Ball-def)
apply (safe elim!: all-forward dest!: eq)
apply (rule-tac x=min d e in exI, simp)
apply (rule pole-theorem-open)
apply (auto simp: holomorphic-on-subset subset-ball)
done

lemma pole-theorem-analytic-0:
  assumes g: g analytic-on s
    and eq: ⋀z. z ∈ s ⟹ ∃ d. 0 < d ∧ (∀ w ∈ ball z d − {a}. g w = (w − a)
∗ f w)
    and [simp]: f a = deriv g a g a = 0
  shows f analytic-on s
proof −
  have [simp]: (λz. if z = a then deriv g a else f z − g a / (z − a)) = f
    by auto
  show ?thesis
    using pole-theorem-analytic [OF g eq] by simp
qed

lemma pole-theorem-analytic-open-superset:
  assumes g: g analytic-on s and s ⊆ t open t
    and eq: ⋀z. z ∈ t − {a} ⟹ g z = (z − a) ∗ f z
  shows (λz. if z = a then deriv g a
             else f z − g a/(z − a)) analytic-on s
  apply (rule pole-theorem-analytic [OF g])
  apply (rule openE [OF ‹open t›])
  using assms eq by auto

lemma pole-theorem-analytic-open-superset-0:
  assumes g: g analytic-on s s ⊆ t open t ⋀z. z ∈ t − {a} ⟹ g z = (z − a) ∗ f
z
    and [simp]: f a = deriv g a g a = 0
  shows f analytic-on s
proof −
  have [simp]: (λz. if z = a then deriv g a else f z − g a / (z − a)) = f
    by auto
  have (λz. if z = a then deriv g a else f z − g a/(z − a)) analytic-on s
    by (rule pole-theorem-analytic-open-superset [OF g])
  then show ?thesis by simp
qed
```

52.46 General, homology form of Cauchy's theorem.

Proof is based on Dixon's, as presented in Lang's "Complex Analysis" book (page 147).

This version has *polynomial-function* γ as an additional assumption.

lemma *Cauchy-integral-formula-global-weak*:
assumes *u*: *open u* **and** *holf*: *f holomorphic-on u*
and *z*: $z \in u$ **and** γ: *polynomial-function* γ
and *pasz*: *path-image* $\gamma \subseteq u - \{z\}$ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *zero*: $\bigwedge w.\ w \notin u \Longrightarrow$ *winding-number* $\gamma\ w = 0$
shows $((\lambda w.\ f\ w\ /\ (w - z))$ *has-contour-integral* $(2{*}pi * ii * winding\text{-}number\ \gamma\ z * f\ z))\ \gamma$
proof −
obtain γ' **where** *pf*γ': *polynomial-function* γ' **and** γ': $\bigwedge x.$ (γ *has-vector-derivative* $(\gamma'\ x))$ $(at\ x)$
using *has-vector-derivative-polynomial-function* [*OF* γ] **by** *blast*
then have *bounded*(*path-image* γ')
by (*simp add*: *path-image-def compact-imp-bounded compact-continuous-image continuous-on-polymonial-function*)
then obtain *B* **where** *B*>*0* **and** *B*: $\bigwedge x.\ x \in$ *path-image* $\gamma' \Longrightarrow$ *norm* $x \leq B$
using *bounded-pos* **by** *force*
def $d \equiv \lambda z\ w.$ *if* $w = z$ *then deriv f z else* $(f\ w - f\ z)/(w - z)$
def $v \equiv \{w.\ w \notin$ *path-image* $\gamma \wedge$ *winding-number* $\gamma\ w = 0\}$
have *path* γ *valid-path* γ **using** γ
by (*auto simp*: *path-polynomial-function valid-path-polynomial-function*)
then have *ov*: *open v*
by (*simp add*: *v-def open-winding-number-levelsets loop*)
have *uv-Un*: $u \cup v = UNIV$
using *pasz zero* **by** (*auto simp*: *v-def*)
have *conf*: *continuous-on u f*
by (*metis holf holomorphic-on-imp-continuous-on*)
have *hol-d*: $(d\ y)$ *holomorphic-on u* **if** $y \in u$ **for** *y*
proof −
have $*$: $(\lambda c.$ *if* $c = y$ *then deriv f y else* $(f\ c - f\ y)\ /\ (c - y))$ *holomorphic-on u*
by (*simp add*: *holf pole-lemma-open u*)
then have *isCont* $(\lambda x.$ *if* $x = y$ *then deriv f y else* $(f\ x - f\ y)\ /\ (x - y))\ y$
using *at-within-open complex-differentiable-imp-continuous-at holomorphic-on-def that u* **by** *fastforce*
then have *continuous-on u* $(d\ y)$
apply (*simp add*: *d-def continuous-on-eq-continuous-at u*, *clarify*)
using $*$ *holomorphic-on-def*
by (*meson complex-differentiable-within-open complex-differentiable-imp-continuous-at u*)
moreover have *d y holomorphic-on* $u - \{y\}$
apply (*simp add*: *d-def holomorphic-on-open u open-delete complex-differentiable-def* [*symmetric*])
apply (*intro ballI*)

apply (*rename-tac w*)
apply (*rule-tac d=dist w y* **and** $f = \lambda w.\ (f\,w - f\,y)/(w - y)$ **in** *complex-differentiable-transform-within*)
apply (*auto simp*: *dist-pos-lt dist-commute intro*!: *derivative-intros*)
using *analytic-on-imp-differentiable-at analytic-on-open holf u* **apply** *blast*
done
ultimately show *?thesis*
by (*rule no-isolated-singularity*) (*auto simp*: *u*)
qed
have *cint-fxy*: $(\lambda x.\ (f\ x\ -\ f\ y)\ /\ (x\ -\ y))$ *contour-integrable-on* γ **if** $y \notin$ *path-image* γ **for** *y*
apply (*rule contour-integrable-holomorphic-simple* [**where** $s = u - \{y\}$])
using ‹*valid-path* γ› *pasz*
apply (*auto simp*: *u open-delete*)
apply (*rule continuous-intros holomorphic-intros continuous-on-subset* [*OF conf*] *holomorphic-on-subset* [*OF holf*] |
force simp add: *that*)+
done
def $h \equiv \lambda z.$ *if* $z \in u$ *then contour-integral* γ (*d z*) *else contour-integral* γ $(\lambda w.\ f\ w/(w - z))$
have *U*: $\bigwedge z.\ z \in u \Longrightarrow$ ((*d z*) *has-contour-integral h z*) γ
apply (*simp add*: *h-def*)
apply (*rule has-contour-integral-integral* [*OF contour-integrable-holomorphic-simple* [**where** *s*=*u*]])
using *u pasz* ‹*valid-path* γ›
apply (*auto intro*: *holomorphic-on-imp-continuous-on hol-d*)
done
have *V*: $((\lambda w.\ f\ w\ /\ (w - z))$ *has-contour-integral h z*) γ **if** *z*: $z \in v$ **for** *z*
proof −
have *0*: $0 = (f\ z) * 2 *$ *of-real* $(2 * pi) *$ i $*$ *winding-number* γ *z*
using *v-def z* **by** *auto*
then have $((\lambda x.\ 1\ /\ (x - z))$ *has-contour-integral 0*) γ
using *z v-def has-contour-integral-winding-number* [*OF* ‹*valid-path* γ›] **by** *fastforce*
then have $((\lambda x.\ f\ z * (1\ /\ (x - z)))$ *has-contour-integral 0*) γ
using *has-contour-integral-lmul* **by** *fastforce*
then have $((\lambda x.\ f\ z\ /\ (x - z))$ *has-contour-integral 0*) γ
by (*simp add*: *divide-simps*)
moreover have $((\lambda x.\ (f\,x - f\,z)\ /\ (x - z))$ *has-contour-integral contour-integral* γ (*d z*)) γ
using *z*
apply (*auto simp*: *v-def*)
apply (*metis* (*no-types*, *lifting*) *contour-integrable-eq d-def has-contour-integral-eq has-contour-integral-integral cint-fxy*)
done
ultimately have $*$: $((\lambda x.\ f\,z\ /\ (x - z) + (f\,x - f\,z)\ /\ (x - z))$ *has-contour-integral* $(0 +$ *contour-integral* γ (*d z*))) γ
by (*rule has-contour-integral-add*)
have $((\lambda w.\ f\ w\ /\ (w - z))$ *has-contour-integral contour-integral* γ (*d z*)) γ
if $z \in u$

using $*$ **by** (*auto simp*: *divide-simps has-contour-integral-eq*)
moreover have ((λ*w*. *f w* / (*w* − *z*)) *has-contour-integral contour-integral* γ (λ*w*. *f w* / (*w* − *z*))) γ
if *z* ∉ *u*
apply (*rule has-contour-integral-integral* [*OF contour-integrable-holomorphic-simple* [**where** *s*=*u*]])
using *u pasz* ‹*valid-path* γ› *that*
apply (*auto intro*: *holomorphic-on-imp-continuous-on hol-d*)
apply (*rule continuous-intros conf holomorphic-intros holf* | *force*)+
done
ultimately show *?thesis*
using *z* **by** (*simp add*: *h-def*)
qed
have *znot*: *z* ∉ *path-image* γ
using *pasz* **by** *blast*
obtain *d0* **where** *d0*>*0* **and** *d0*: ⋀*x y*. *x* ∈ *path-image* γ ⟹ *y* ∈ − *u* ⟹ *d0* ≤ *dist x y*
using *separate-compact-closed* [*of path-image* γ −*u*] *pasz u*
by (*fastforce simp add*: ‹*path* γ› *compact-path-image*)
obtain *dd* **where** *0* < *dd* **and** *dd*: {*y* + *k* | *y k*. *y* ∈ *path-image* γ ∧ *k* ∈ *ball 0 dd*} ⊆ *u*
apply (*rule that* [*of d0*/*2*])
using ‹*0* < *d0*›
apply (*auto simp*: *dist-norm dest*: *d0*)
done
have ⋀*x x'*. ⟦*x* ∈ *path-image* γ; *dist x x'* ∗ *2* < *dd*⟧ ⟹ ∃ *y k*. *x'* = *y* + *k* ∧ *y* ∈ *path-image* γ ∧ *dist 0 k* ∗ *2* ≤ *dd*
apply (*rule-tac x*=*x* **in** *exI*)
apply (*rule-tac x*=*x'*−*x* **in** *exI*)
apply (*force simp add*: *dist-norm*)
done
then have *1*: *path-image* γ ⊆ *interior* {*y* + *k* |*y k*. *y* ∈ *path-image* γ ∧ *k* ∈ *cball 0* (*dd* / *2*)}
apply (*clarsimp simp add*: *mem-interior*)
using ‹*0* < *dd*›
apply (*rule-tac x*=*dd*/*2* **in** *exI*, *auto*)
done
obtain *t* **where** *compact t* **and** *subt*: *path-image* γ ⊆ *interior t* **and** *t*: *t* ⊆ *u*
apply (*rule that* [*OF - 1*])
apply (*fastforce simp add*: ‹*valid-path* γ› *compact-valid-path-image intro*!: *compact-sums*)
apply (*rule order-trans* [*OF - dd*])
using ‹*0* < *dd*› **by** *fastforce*
obtain *L* **where** *L*>*0*
and *L*: ⋀*f B*. ⟦*f holomorphic-on interior t*; ⋀*z*. *z*∈*interior t* ⟹ *cmod* (*f z*) ≤ *B*⟧ ⟹
cmod (*contour-integral* γ *f*) ≤ *L* ∗ *B*
using *contour-integral-bound-exists* [*OF open-interior* ‹*valid-path* γ› *subt*]
by *blast*
have *bounded*(*f* ‘ *t*)

by (*meson* ‹*compact t*› *compact-continuous-image compact-imp-bounded conf continuous-on-subset t*)
then obtain *D* **where** *D>0* **and** *D*: $\bigwedge x.\ x \in t \Longrightarrow$ *norm* (*f x*) $\leq D$
by (*auto simp*: *bounded-pos*)
obtain *C* **where** *C>0* **and** *C*: $\bigwedge x.\ x \in t \Longrightarrow$ *norm x* $\leq C$
using ‹*compact t*› *bounded-pos compact-imp-bounded* **by** *force*
have *dist* (*h y*) *0* $\leq e$ **if** *0* $< e$ **and** *le*: *D* $*$ *L* / *e* $+$ *C* $\leq$ *cmod y* **for** *e y*
proof −
have *D* $*$ *L* / *e* > 0 **using** ‹*D>0*› ‹*L>0*› ‹*e>0*› **by** *simp*
with *le* **have** *ybig*: *norm y* $>$ *C* **by** *force*
with *C* **have** *y* $\notin$ *t* **by** *force*
then have *ynot*: *y* $\notin$ *path-image* γ
using *subt interior-subset* **by** *blast*
have [*simp*]: *winding-number* γ *y* $= 0$
apply (*rule winding-number-zero-outside* [*of - cball 0 C*])
using *ybig interior-subset subt*
apply (*force simp add*: *loop* ‹*path* γ› *dist-norm intro*!: *C*)+
done
have [*simp*]: *h y* $=$ *contour-integral* γ ($\lambda w.$ *f w*/(*w* $-$ *y*))
by (*rule contour-integral-unique* [*symmetric*]) (*simp add*: *v-def ynot V*)
have *holint*: ($\lambda w.$ *f w* / (*w* $-$ *y*)) *holomorphic-on interior t*
apply (*rule holomorphic-on-divide*)
using *holf holomorphic-on-subset interior-subset t* **apply** *blast*
apply (*rule holomorphic-intros*)+
using ‹*y* $\notin$ *t*› *interior-subset* **by** *auto*
have *leD*: *cmod* (*f z* / (*z* $-$ *y*)) $\leq$ *D* $*$ (*e* / *L* / *D*) **if** *z*: *z* $\in$ *interior t* **for** *z*
proof −
have *D* $*$ *L* / *e* $+$ *cmod z* $\leq$ *cmod y*
using *le C* [*of z*] *z* **using** *interior-subset* **by** *force*
then have *DL2*: *D* $*$ *L* / *e* $\leq$ *cmod* (*z* $-$ *y*)
using *norm-triangle-ineq2* [*of y z*] **by** (*simp add*: *norm-minus-commute*)
have *cmod* (*f z* / (*z* $-$ *y*)) $=$ *cmod* (*f z*) $*$ *inverse* (*cmod* (*z* $-$ *y*))
by (*simp add*: *norm-mult norm-inverse Fields.field-class.field-divide-inverse*)
also have ... $\leq$ *D* $*$ (*e* / *L* / *D*)
apply (*rule mult-mono*)
using *that D interior-subset* **apply** *blast*
using ‹*L>0*› ‹*e>0*› ‹*D>0*› *DL2*
apply (*auto simp*: *norm-divide divide-simps algebra-simps*)
done
finally show *?thesis* **.**
qed
have *dist* (*h y*) *0* $=$ *cmod* (*contour-integral* γ ($\lambda w.$ *f w* / (*w* $-$ *y*)))
by (*simp add*: *dist-norm*)
also have ... $\leq$ *L* $*$ (*D* $*$ (*e* / *L* / *D*))
by (*rule L* [*OF holint leD*])
also have ... $=$ *e*
using ‹*L>0*› ‹*0* $<$ *D*› **by** *auto*
finally show *?thesis* **.**
qed

```
 then have (h ⟶ 0) at-infinity
  by (meson Lim-at-infinityI)
 moreover have h holomorphic-on UNIV
 proof −
  have con-ff: continuous (at (x,z)) (λy. (f(snd y) − f(fst y)) / (snd y − fst y))
            if x ∈ u z ∈ u x ≠ z for x z
    using that conf
    apply (simp add: continuous-on-eq-continuous-at u)
     apply (simp | rule continuous-intros continuous-within-compose2 [where
g=f])+
    done
   have con-fstsnd: continuous-on UNIV (λx. (fst x − snd x) ::complex)
    by (rule continuous-intros)+
   have open-uu-Id: open (u × u − Id)
    apply (rule open-Diff)
    apply (simp add: open-Times u)
     using continuous-closed-preimage-constant [OF con-fstsnd closed-UNIV, of
0]
    apply (auto simp: Id-fstsnd-eq algebra-simps)
    done
   have con-derf: continuous (at z) (deriv f) if z ∈ u for z
    apply (rule continuous-on-interior [of u])
     apply (simp add: holf holomorphic-deriv holomorphic-on-imp-continuous-on
u)
    by (simp add: interior-open that u)
   have tendsto-f': ((λx. if snd x = fst x then deriv f (fst x)
                          else (f (snd x) − f (fst x)) / (snd x − fst x)) ⟶
deriv f x)
                    (at (x, x) within u × u) if x ∈ u for x
   proof (rule Lim-withinI)
    fix e::real assume 0 < e
    obtain k1 where k1>0 and k1: ⋀x'. norm (x' − x) ≤ k1 ⟹ norm (deriv
f x' − deriv f x) < e
      using ‹0 < e› continuous-within-E [OF con-derf [OF ‹x ∈ u›]]
      by (metis UNIV-I dist-norm)
     obtain k2 where k2>0 and k2: ball x k2 ⊆ u by (blast intro: openE [OF
u] ‹x ∈ u›)
    have neq: norm ((f z' − f x') / (z' − x') − deriv f x) ≤ e
                 if z' ≠ x' and less-k1: norm (x'−x, z'−x) < k1 and less-k2:
norm (x'−x, z'−x) < k2
              for x' z'
    proof −
     have cs-less: w ∈ closed-segment x' z' ⟹ cmod (w − x) ≤ norm (x'−x,
z'−x) for w
      apply (drule segment-furthest-le [where y=x])
       by (metis (no-types) dist-commute dist-norm norm-fst-le norm-snd-le
order-trans)
     have derf-le: w ∈ closed-segment x' z' ⟹ z' ≠ x' ⟹ cmod (deriv f w −
deriv f x) ≤ e for w
```

by (*blast intro*: *cs-less less-k1 k1* [*unfolded divide-const-simps dist-norm*] *less-imp-le le-less-trans*)
have *f-has-der*: $\bigwedge x.\ x \in u \Longrightarrow (f \text{ has-field-derivative } deriv\ f\ x)\ (at\ x\ within\ u)$
by (*metis DERIV-deriv-iff-complex-differentiable at-within-open holf holomorphic-on-def u*)
have *closed-segment* $x'\ z' \subseteq u$
by (*rule order-trans* [*OF - k2*]) (*simp add*: *cs-less le-less-trans* [*OF - less-k2*] *dist-complex-def norm-minus-commute subset-iff*)
then have *cint-derf*: (*deriv f has-contour-integral f* $z' - f\ x'$) (*linepath* $x'\ z'$)
using *contour-integral-primitive* [*OF f-has-der valid-path-linepath*] *pasz* **by** *simp*
then have $*$: $((\lambda x.\ deriv\ f\ x\ /\ (z' - x'))$ *has-contour-integral* $(f\ z' - f\ x')\ /\ (z' - x'))$ (*linepath* $x'\ z'$)
by (*rule has-contour-integral-div*)
have $norm\ ((f\ z' - f\ x')\ /\ (z' - x') - deriv\ f\ x) \le e/norm(z' - x') * norm(z' - x')$
apply (*rule has-contour-integral-bound-linepath* [*OF has-contour-integral-diff* [*OF* $*$]])
using *has-contour-integral-div* [**where** $c = z' - x'$, *OF has-contour-integral-const-linepath* [*of deriv f x z' x'*]]
‹$e > 0$› ‹$z' \neq x'$›
apply (*auto simp*: *norm-divide divide-simps derf-le*)
done
also have ... $\le e$ **using** ‹$0 < e$› **by** *simp*
finally show *?thesis* **.**
qed
show $\exists\, d{>}0.\ \forall\, xa{\in}u \times u.$
$0 < dist\ xa\ (x, x) \wedge dist\ xa\ (x, x) < d \longrightarrow$
$dist\ (if\ snd\ xa = fst\ xa\ then\ deriv\ f\ (fst\ xa)\ else\ (f\ (snd\ xa) - f\ (fst\ xa))\ /\ (snd\ xa - fst\ xa))$
$(deriv\ f\ x) \le e$
apply (*rule-tac x=min k1 k2* **in** *exI*)
using ‹$k1{>}0$› ‹$k2{>}0$› ‹$e{>}0$›
apply (*force simp*: *dist-norm neq intro*: *dual-order.strict-trans2 k1 less-imp-le norm-fst-le*)
done
qed
have *con-pa-f*: *continuous-on* (*path-image* γ) *f*
by (*meson holf holomorphic-on-imp-continuous-on holomorphic-on-subset interior-subset subt t*)
have *le-B*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow cmod\ (vector\text{-}derivative\ \gamma\ (at\ t)) \le B$
apply (*rule B*)
using γ' **using** *path-image-def vector-derivative-at* **by** *fastforce*
have *f-has-cint*: $\bigwedge w.\ w \in v - path\text{-}image\ \gamma \Longrightarrow ((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ 1)$ *has-contour-integral h w*) γ
by (*simp add*: *V*)
have *cond-uu*: *continuous-on* $(u \times u)\ (\lambda y.\ d\ (fst\ y)\ (snd\ y))$

apply (*simp add*: *continuous-on-eq-continuous-within d-def continuous-within tendsto-f'*)
apply (*simp add*: *Lim-within-open-NO-MATCH open-Times u*, *clarify*)
apply (*rule Lim-transform-within-open* [*OF - open-uu-Id*, **where** $f = (\lambda x.\ (f\ (snd\ x) - f\ (fst\ x))\ /\ (snd\ x - fst\ x))$])
using *con-ff*
apply (*auto simp*: *continuous-within*)
done
have *hol-dw*: $(\lambda z.\ d\ z\ w)$ *holomorphic-on u* **if** $w \in u$ **for** *w*
proof −
have *continuous-on u* $((\lambda y.\ d\ (fst\ y)\ (snd\ y))\ o\ (\lambda z.\ (w,z)))$
by (*rule continuous-on-compose continuous-intros continuous-on-subset* [*OF cond-uu*] | *force intro*: *that*)+
then have ∗: *continuous-on u* $(\lambda z.\ if\ w = z\ then\ deriv\ f\ z\ else\ (f\ w - f\ z)\ /\ (w - z))$
by (*rule rev-iffD1* [*OF - continuous-on-cong* [*OF refl*]]) (*simp add*: *d-def field-simps*)
have ∗∗: $\bigwedge x.\ [\![x \in u;\ x \neq w]\!] \Longrightarrow (\lambda z.\ if\ w = z\ then\ deriv\ f\ z\ else\ (f\ w - f\ z)\ /\ (w - z))$ *complex-differentiable at x*
apply (*rule-tac* $f = \lambda x.\ (f\ w - f\ x)/(w - x)$ **and** $d = dist\ x\ w$ **in** *complex-differentiable-transform-within*)
apply (*rule u derivative-intros holomorphic-on-imp-differentiable-at* [*OF holf*] | *force simp add*: *dist-commute*)+
done
show *?thesis*
unfolding *d-def*
apply (*rule no-isolated-singularity* [*OF* ∗ - *u*, **where** $k = \{w\}$])
apply (*auto simp*: *complex-differentiable-def* [*symmetric*] *holomorphic-on-open open-Diff u* ∗∗)
done
qed
{ fix *a b*
assume *abu*: *closed-segment a b* ⊆ *u*
then have *cont-dw*: $\bigwedge w.\ w \in u \Longrightarrow (\lambda z.\ d\ z\ w)$ *contour-integrable-on* (*linepath a b*)
by (*metis hol-dw continuous-on-subset contour-integrable-continuous-linepath holomorphic-on-imp-continuous-on*)
have ∗: $\exists da{>}0.\ \forall x'{\in}u.\ dist\ x'\ w < da \longrightarrow$
$dist\ (contour\text{-}integral\ (linepath\ a\ b)\ (\lambda z.\ d\ z\ x'))$
$(contour\text{-}integral\ (linepath\ a\ b)\ (\lambda z.\ d\ z\ w)) \leq ee$
if $w \in u\ 0 < ee\ a \neq b$ **for** *w ee*
proof −
obtain *dd* **where** *dd*>*0* **and** *dd*: *cball w dd* ⊆ *u* **using** *open-contains-cball u* ‹$w \in u$› **by** *force*
let *?abdd* = $\{(z,t)\ |z\ t.\ z \in closed\text{-}segment\ a\ b \wedge t \in cball\ w\ dd\}$
have *uniformly-continuous-on ?abdd* $(\lambda y.\ d\ (fst\ y)\ (snd\ y))$
apply (*rule compact-uniformly-continuous*)
apply (*rule continuous-on-subset*[*OF cond-uu*])
using *abu dd*

apply (*auto simp*: *compact-Times simp del*: *mem-cball*)
done
then obtain *kk* **where** *kk*>*0*
and *kk*: $\bigwedge x\ x'.\ [\![x \in ?abdd;\ x' \in ?abdd;\ dist\ x'\ x < kk]\!] \Longrightarrow$ *dist* ((λ*y*. *d* (*fst y*) (*snd y*)) *x'*) ((λ*y*. *d* (*fst y*) (*snd y*)) *x*) < *ee*/*norm*(*b* − *a*)
apply (*rule uniformly-continuous-onE* [**where** *e* = *ee*/*norm*(*b* − *a*)])
using ‹*0* < *ee*› ‹*a* ≠ *b*› **by** *auto*
have *kk*: $[\![x1 \in closed\text{-}segment\ a\ b;\ norm\ (w - x2) \le dd;\ x1' \in closed\text{-}segment\ a\ b;\ norm\ (w - x2') \le dd;\ norm\ ((x1', x2') - (x1, x2)) < kk]\!]$ $\Longrightarrow$ *norm* (*d x1' x2'* − *d x1 x2*) ≤ *ee* / *cmod* (*b* − *a*)
for *x1 x2 x1' x2'*
using *kk* [*of* (*x1*,*x2*) (*x1'*,*x2'*)] **by** (*force simp add*: *dist-norm*)
have *le-ee*: *cmod* (*contour-integral* (*linepath a b*) (λ*x*. *d x x'* − *d x w*)) ≤ *ee*
if *x'* ∈ *u cmod* (*x'* − *w*) < *dd cmod* (*x'* − *w*) < *kk* **for** *x'*
proof −
have *cmod* (*contour-integral* (*linepath a b*) (λ*x*. *d x x'* − *d x w*)) ≤ *ee*/*norm*(*b* − *a*) ∗ *norm*(*b* − *a*)
apply (*rule has-contour-integral-bound-linepath* [*OF has-contour-integral-integral* - *kk*])
apply (*rule contour-integrable-diff* [*OF cont-dw cont-dw*])
using ‹*0* < *ee*› ‹*a* ≠ *b*› ‹*0* < *dd*› ‹*w* ∈ *u*› *that*
apply (*auto simp*: *norm-minus-commute*)
done
also have ... = *ee* **using** ‹*a* ≠ *b*› **by** *simp*
finally show *?thesis* **.**
qed
show *?thesis*
apply (*rule-tac x*=*min dd kk* **in** *exI*)
using ‹*0* < *dd*› ‹*0* < *kk*›
apply (*auto simp*: *dist-norm contour-integral-diff* [*OF cont-dw cont-dw*, *symmetric*] ‹*w* ∈ *u*› *intro*: *le-ee*)
done
qed
have *cont-cint-d*: *continuous-on u* (λ*w*. *contour-integral* (*linepath a b*) (λ*z*. *d z w*))
apply (*rule continuous-onI*)
apply (*cases a*=*b*)
apply (*auto intro*: ∗)
done
have *cont-cint-dγ*: *continuous-on* {*0..1*} ((λ*w*. *contour-integral* (*linepath a b*) (λ*z*. *d z w*)) *o* γ)
apply (*rule continuous-on-compose*)
using ‹*path* γ› *path-def pasz*
apply (*auto intro*!: *continuous-on-subset* [*OF cont-cint-d*])
apply (*force simp add*: *path-image-def*)
done
have *cint-cint*: (λ*w*. *contour-integral* (*linepath a b*) (λ*z*. *d z w*)) *contour-integrable-on*

γ
apply (*simp add*: *contour-integrable-on*)
apply (*rule integrable-continuous-real*)
apply (*rule continuous-on-mult* [*OF cont-cint-d*γ [*unfolded o-def*]])
using *pf*γ'
by (*simp add*: *continuous-on-polymonial-function vector-derivative-at* [*OF* γ'])
have *contour-integral* (*linepath a b*) *h* = *contour-integral* (*linepath a b*) (λz. *contour-integral* γ (*d z*))
using *abu* **by** (*force simp add*: *h-def intro*: *contour-integral-eq*)
also have ... = *contour-integral* γ (λw. *contour-integral* (*linepath a b*) (λz. *d z w*))
apply (*rule contour-integral-swap*)
apply (*simp add*: *split-def*)
apply (*rule continuous-on-subset* [*OF cond-uu*])
using *abu pasz* ‹*valid-path* γ›
apply (*auto intro*!: *continuous-intros*)
by (*metis* γ' *continuous-on-eq path-def path-polynomial-function pf*γ' *vector-derivative-at*)
finally have *cint-h-eq*:
contour-integral (*linepath a b*) *h* =
contour-integral γ (λw. *contour-integral* (*linepath a b*) (λz. *d z w*))
.
note *cint-cint cint-h-eq*
} **note** *cint-h* = *this*
have *conthu*: *continuous-on u h*
proof (*simp add*: *continuous-on-sequentially*, *clarify*)
fix *a x*
assume *x*: $x \in u$ **and** *au*: $\forall n.\ a\ n \in u$ **and** *ax*: $a \longrightarrow x$
then have *A1*: $\forall_F\ n$ *in sequentially*. *d* (*a n*) *contour-integrable-on* γ
by (*meson U contour-integrable-on-def eventuallyI*)
obtain *dd* **where** *dd*>*0* **and** *dd*: *cball x dd* $\subseteq$ *u* **using** *open-contains-cball u x* **by** *force*
have *A2*: $\forall_F\ n$ *in sequentially*. $\forall$ *xa*$\in$*path-image* γ. *cmod* (*d* (*a n*) *xa* − *d x xa*) < *ee* **if** *0* < *ee* **for** *ee*
proof −
let *?ddpa* = {(*w*,*z*) |*w z*. *w* $\in$ *cball x dd* $\wedge$ *z* $\in$ *path-image* γ}
have *uniformly-continuous-on ?ddpa* (λy. *d* (*fst y*) (*snd y*))
apply (*rule compact-uniformly-continuous* [*OF continuous-on-subset*[*OF cond-uu*]])
using *dd pasz* ‹*valid-path* γ›
apply (*auto simp*: *compact-Times compact-valid-path-image simp del*: *mem-cball*)
done
then obtain *kk* **where** *kk*>*0*
and *kk*: $\bigwedge x\ x'.\ [\![x \in ?ddpa;\ x' \in ?ddpa;\ dist\ x'\ x < kk]\!] \Longrightarrow$
dist ((λy. *d* (*fst y*) (*snd y*)) *x'*) ((λy. *d* (*fst y*) (*snd y*)) *x*) < *ee*
apply (*rule uniformly-continuous-onE* [**where** *e* = *ee*])

using ‹*0 < ee*› **by** *auto*

have *kk*: ⟦*norm* (*w − x*) ≤ *dd*; *z* ∈ *path-image* *γ*; *norm* ((*w*, *z*) − (*x*, *z*)) < *kk*⟧ ⟹ *norm* (*d w z − d x z*) < *ee*
for *w z*
using ‹*dd>0*› *kk* [*of* (*x*,*z*) (*w*,*z*)] **by** (*force simp add*: *norm-minus-commute dist-norm*)
show *?thesis*
using *ax* **unfolding** *lim-sequentially eventually-sequentially*
apply (*drule-tac x=min dd kk* **in** *spec*)
using ‹*dd > 0*› ‹*kk > 0*›
apply (*fastforce simp*: *kk dist-norm*)
done
qed
have *tendsto-hx*: (λ*n*. *contour-integral* *γ* (*d* (*a n*))) ⟶ *h x*
apply (*simp add*: *contour-integral-unique* [*OF U*, *symmetric*] *x*)
apply (*rule contour-integral-uniform-limit* [*OF A1 A2 le-B*])
apply (*auto simp*: ‹*valid-path* *γ*›)
done
then show (*h* ∘ *a*) ⟶ *h x*
by (*simp add*: *h-def x au o-def*)
qed
show *?thesis*
proof (*simp add*: *holomorphic-on-open complex-differentiable-def* [*symmetric*], *clarify*)
fix *z0*
consider *z0* ∈ *v* | *z0* ∈ *u* **using** *uv-Un* **by** *blast*
then show *h complex-differentiable at z0*
proof *cases*
assume *z0* ∈ *v* **then show** *?thesis*
using *Cauchy-next-derivative* [*OF con-pa-f le-B f-has-cint - ov*] *V f-has-cint* ‹*valid-path* *γ*›
by (*auto simp*: *complex-differentiable-def v-def*)
next
assume *z0* ∈ *u* **then**
obtain *e* **where** *e>0* **and** *e*: *ball z0 e* ⊆ *u* **by** (*blast intro*: *openE* [*OF u*])
have ∗: *contour-integral* (*linepath a b*) *h* + *contour-integral* (*linepath b c*) *h* + *contour-integral* (*linepath c a*) *h* = *0*
if *abc-subset*: *convex hull* {*a*, *b*, *c*} ⊆ *ball z0 e* **for** *a b c*
proof −
have ∗: ⋀*x1 x2 z*. *z* ∈ *u* ⟹ *closed-segment x1 x2* ⊆ *u* ⟹ (λ*w*. *d w z*) *contour-integrable-on linepath x1 x2*
using *hol-dw holomorphic-on-imp-continuous-on u*
by (*auto intro*!: *contour-integrable-holomorphic-simple*)
have *abc*: *closed-segment a b* ⊆ *u* *closed-segment b c* ⊆ *u* *closed-segment c a* ⊆ *u*
using *that e segments-subset-convex-hull* **by** *fastforce+*
have *eq0*: ⋀*w*. *w* ∈ *u* ⟹ *contour-integral* (*linepath a b +++ linepath b c +++ linepath c a*) (λ*z*. *d z w*) = *0*

apply (*rule contour-integral-unique* [*OF Cauchy-theorem-triangle*])
apply (*rule holomorphic-on-subset* [*OF hol-dw*])
using *e abc-subset* **by** *auto*
have *contour-integral* γ
(λx. *contour-integral* (*linepath a b*) (λz. *d z x*) +
(*contour-integral* (*linepath b c*) (λz. *d z x*) +
contour-integral (*linepath c a*) (λz. *d z x*))) = *0*
apply (*rule contour-integral-eq-0*)
using *abc pasz u*
apply (*subst contour-integral-join* [*symmetric*], *auto intro*: *eq0* ∗)+
done
then show *?thesis*
by (*simp add*: *cint-h abc contour-integrable-add contour-integral-add* [*symmetric*] *add-ac*)
qed
show *?thesis*
using *e* ‹*e* > *0*›
by (*auto intro*!: *holomorphic-on-imp-differentiable-at* [*OF* - *open-ball*] *analytic-imp-holomorphic*
Morera-triangle continuous-on-subset [*OF conthu*] ∗)
qed
qed
qed
ultimately have [*simp*]: *h z* = *0* **for** *z*
by (*meson Liouville-weak*)
have ((λw. *1* / (*w* − *z*)) *has-contour-integral complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number* γ *z*) γ
by (*rule has-contour-integral-winding-number* [*OF* ‹*valid-path* γ› *znot*])
then have ((λw. *f z* ∗ (*1* / (*w* − *z*))) *has-contour-integral complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number* γ *z* ∗ *f z*) γ
by (*metis mult.commute has-contour-integral-lmul*)
then have *1*: ((λw. *f z* / (*w* − *z*)) *has-contour-integral complex-of-real* (*2* ∗ *pi*) ∗ i ∗ *winding-number* γ *z* ∗ *f z*) γ
by (*simp add*: *divide-simps*)
moreover have *2*: ((λw. (*f w* − *f z*) / (*w* − *z*)) *has-contour-integral 0*) γ
using *U* [*OF z*] *pasz d-def* **by** (*force elim*: *has-contour-integral-eq* [**where** *g* = λw. (*f w* − *f z*)/(*w* − *z*)])
show *?thesis*
using *has-contour-integral-add* [*OF 1 2*] **by** (*simp add*: *diff-divide-distrib*)
qed

theorem *Cauchy-integral-formula-global*:
assumes *s*: *open s* **and** *holf*: *f holomorphic-on s*
and *z*: *z* ∈ *s* **and** *vpg*: *valid-path* γ
and *pasz*: *path-image* γ ⊆ *s* − {*z*} **and** *loop*: *pathfinish* γ = *pathstart* γ
and *zero*: $\bigwedge w$. *w* ∉ *s* ⟹ *winding-number* γ *w* = *0*
shows ((λw. *f w* / (*w* − *z*)) *has-contour-integral* (*2*∗*pi* ∗ *ii* ∗ *winding-number* γ *z* ∗ *f z*)) γ

proof −
have *path* γ **using** *vpg* **by** (*blast intro*: *valid-path-imp-path*)
have *hols*: $(\lambda w.\ f\ w\ /\ (w - z))$ *holomorphic-on* $s - \{z\}$ $(\lambda w.\ 1\ /\ (w - z))$ *holomorphic-on* $s - \{z\}$
by (*rule holomorphic-intros holomorphic-on-subset* [*OF holf*] | *force*)+
then have *cint-fw*: $(\lambda w.\ f\ w\ /\ (w - z))$ *contour-integrable-on* γ
by (*meson contour-integrable-holomorphic-simple holomorphic-on-imp-continuous-on open-delete s vpg pasz*)
obtain d **where** $d{>}0$
and d: $\bigwedge g\ h.\ [\![$*valid-path* g; *valid-path* h; $\forall t{\in}\{0..1\}.$ *cmod* $(g\ t - \gamma\ t) < d \wedge$ *cmod* $(h\ t - \gamma\ t) < d$;
pathstart h = *pathstart* $g \wedge$ *pathfinish* h = *pathfinish* $g$$]\!]$
$\Longrightarrow$ *path-image* $h \subseteq s - \{z\} \wedge (\forall f.\ f$ *holomorphic-on* $s - \{z\}$ $\longrightarrow$ *contour-integral* $h\ f$ = *contour-integral* $g\ f)$
using *contour-integral-nearby-ends* [*OF* - ‹*path* γ› *pasz*] s **by** (*simp add*: *open-Diff*) *metis*
obtain p **where** *polyp*: *polynomial-function* p
and *ps*: *pathstart* p = *pathstart* γ **and** *pf*: *pathfinish* p = *pathfinish* γ **and** *led*: $\forall t{\in}\{0..1\}.$ *cmod* $(p\ t - \gamma\ t) < d$
using *path-approx-polynomial-function* [*OF* ‹*path* γ› ‹$d > 0$›] **by** *blast*
then have *ploop*: *pathfinish* p = *pathstart* p **using** *loop* **by** *auto*
have *vpp*: *valid-path* p **using** *polyp valid-path-polynomial-function* **by** *blast*
have [*simp*]: $z \notin$ *path-image* γ **using** *pasz* **by** *blast*
have *paps*: *path-image* $p \subseteq s - \{z\}$ **and** *cint-eq*: $(\bigwedge f.\ f$ *holomorphic-on* $s - \{z\}$ $\Longrightarrow$ *contour-integral* $p\ f$ = *contour-integral* $\gamma\ f)$
using *pf ps led d* [*OF vpg vpp*] ‹$d > 0$› **by** *auto*
have *wn-eq*: *winding-number* $p\ z$ = *winding-number* $\gamma\ z$
using *vpp paps*
by (*simp add*: *subset-Diff-insert vpg valid-path-polynomial-function winding-number-valid-path cint-eq hols*)
have *winding-number* $p\ w$ = *winding-number* $\gamma\ w$ **if** $w \notin s$ **for** w
proof −
have *hol*: $(\lambda v.\ 1\ /\ (v - w))$ *holomorphic-on* $s - \{z\}$
using *that* **by** (*force intro*: *holomorphic-intros holomorphic-on-subset* [*OF holf*])
have $w \notin$ *path-image* p $w \notin$ *path-image* γ **using** *paps pasz that* **by** *auto*
then show *?thesis*
using *vpp vpg* **by** (*simp add*: *subset-Diff-insert valid-path-polynomial-function winding-number-valid-path cint-eq* [*OF hol*])
qed
then have *wn0*: $\bigwedge w.\ w \notin s \Longrightarrow$ *winding-number* $p\ w = 0$
by (*simp add*: *zero*)
show *?thesis*
using *Cauchy-integral-formula-global-weak* [*OF s holf z polyp paps ploop wn0*] *hols*
by (*metis wn-eq cint-eq has-contour-integral-eqpath cint-fw cint-eq*)
qed

theorem *Cauchy-theorem-global*:

assumes *s*: *open s* **and** *holf*: *f holomorphic-on s*
and *vpg*: *valid-path γ* **and** *loop*: *pathfinish γ = pathstart γ*
and *pas*: *path-image γ ⊆ s*
and *zero*: $\bigwedge w.\ w \notin s \Longrightarrow$ *winding-number γ w = 0*
shows (*f has-contour-integral 0*) *γ*
proof −
obtain *z* **where** $z \in s$ **and** *znot*: $z \notin$ *path-image γ*
proof −
have *compact* (*path-image γ*)
using *compact-valid-path-image vpg* **by** *blast*
then have *path-image γ* $\neq s$
by (*metis* (*no-types*) *compact-open path-image-nonempty s*)
with *pas* **show** *?thesis* **by** (*blast intro*: *that*)
qed
then have *pasz*: *path-image γ* $\subseteq s - \{z\}$ **using** *pas* **by** *blast*
have *hol*: $(\lambda w.\ (w - z) * f\ w)$ *holomorphic-on s*
by (*rule holomorphic-intros holf*)+
show *?thesis*
using *Cauchy-integral-formula-global* [*OF s hol* ‹$z \in s$› *vpg pasz loop zero*]
by (*auto simp*: *znot elim*!: *has-contour-integral-eq*)
qed

corollary *Cauchy-theorem-global-outside*:
assumes *open s f holomorphic-on s valid-path γ pathfinish γ = pathstart γ*
path-image γ $\subseteq s$
$\bigwedge w.\ w \notin s \Longrightarrow w \in$ *outside*(*path-image γ*)
shows (*f has-contour-integral 0*) *γ*
by (*metis Cauchy-theorem-global assms winding-number-zero-in-outside valid-path-imp-path*)

end

53 Generalised Binomial Theorem

The proof of the Generalised Binomial Theorem and related results. We prove the generalised binomial theorem for complex numbers, following the proof at: https://proofwiki.org/wiki/Binomial_Theorem/General_Binomial_Theorem

theory *Generalised-Binomial-Theorem*
imports
Complex-Main
Complex-Transcendental
Summation
begin

lemma *gbinomial-ratio-limit*:
fixes *a* :: *'a* :: *real-normed-field*
assumes $a \notin \mathbb{N}$

```
  shows (λn. (a gchoose n) / (a gchoose Suc n)) ⟶ −1
proof (rule Lim-transform-eventually)
  let ?f = λn. inverse (a / of-nat (Suc n) − of-nat n / of-nat (Suc n))
  from eventually-gt-at-top[of 0::nat]
    show eventually (λn. ?f n = (a gchoose n) /(a gchoose Suc n)) sequentially
  proof eventually-elim
    fix n :: nat assume n: n > 0
    let ?P = ∏ i = 0..n − 1. a − of-nat i
    from n have (a gchoose n) / (a gchoose Suc n) = (of-nat (Suc n) :: 'a) *
                (?P / (∏ i = 0..n. a − of-nat i)) by (simp add: gbinomial-def)
    also from n have (∏ i = 0..n. a − of-nat i) = ?P * (a − of-nat n)
      by (cases n) (simp-all add: setprod-nat-ivl-Suc)
   also have ?P / ... = (?P / ?P) / (a − of-nat n) by (rule divide-divide-eq-left[symmetric])
    also from assms have ?P / ?P = 1 by auto
    also have of-nat (Suc n) * (1 / (a − of-nat n)) =
                inverse (inverse (of-nat (Suc n)) * (a − of-nat n)) by (simp add:
field-simps)
    also have inverse (of-nat (Suc n)) * (a − of-nat n) = a / of-nat (Suc n) −
of-nat n / of-nat (Suc n)
      by (simp add: field-simps del: of-nat-Suc)
    finally show ?f n = (a gchoose n) / (a gchoose Suc n) by simp
  qed

  have (λn. norm a / (of-nat (Suc n))) ⟶ 0
    unfolding divide-inverse
    by (intro tendsto-mult-right-zero LIMSEQ-inverse-real-of-nat)
  hence (λn. a / of-nat (Suc n)) ⟶ 0
   by (subst tendsto-norm-zero-iff[symmetric]) (simp add: norm-divide del: of-nat-Suc)
  hence ?f ⟶ inverse (0 − 1)
    by (intro tendsto-inverse tendsto-diff LIMSEQ-n-over-Suc-n) simp-all
  thus ?f ⟶ −1 by simp
qed

lemma conv-radius-gchoose:
  fixes a :: 'a :: {real-normed-field,banach}
  shows conv-radius (λn. a gchoose n) = (if a ∈ ℕ then ∞ else 1)
proof (cases a ∈ ℕ)
  assume a: a ∈ ℕ
  have eventually (λn. (a gchoose n) = 0) sequentially
    using eventually-gt-at-top[of nat ⌊norm a⌋]
   by eventually-elim (insert a, auto elim!: Nats-cases simp: binomial-gbinomial[symmetric])
  from conv-radius-cong[OF this] a show ?thesis by simp
next
  assume a: a ∉ ℕ
  from tendsto-norm[OF gbinomial-ratio-limit[OF this]]
    have conv-radius (λn. a gchoose n) = 1
    by (intro conv-radius-ratio-limit-nonzero[of - 1]) (simp-all add: norm-divide)
  with a show ?thesis by simp
qed
```

lemma *gen-binomial-complex*:
fixes *z* :: *complex*
assumes *norm z < 1*
shows (λ*n*. (*a gchoose n*) * *z*^*n*) *sums* (*1* + *z*) *powr a*
proof −
def *K* ≡ *1* − (*1* − *norm z*) / *2*
from *assms* **have** *K*: *K* > *0* *K* < *1* *norm z* < *K*
unfolding *K-def* **by** (*auto simp*: *field-simps intro*!: *add-pos-nonneg*)
let *?f* = λ*n*. *a gchoose n* **and** *?f'* = *diffs* (λ*n*. *a gchoose n*)
have *summable-strong*: *summable* (λ*n*. *?f n* * *z* ^ *n*) **if** *norm z* < *1* **for** *z* **using** *that*
by (*intro summable-in-conv-radius*) (*simp-all add*: *conv-radius-gchoose*)
with *K* **have** *summable*: *summable* (λ*n*. *?f n* * *z* ^ *n*) **if** *norm z* < *K* **for** *z* **using** *that* **by** *auto*
hence *summable'*: *summable* (λ*n*. *?f' n* * *z* ^ *n*) **if** *norm z* < *K* **for** *z* **using** *that*
by (*intro termdiff-converges*[*of* - *K*]) *simp-all*

def *f* ≡ λ*z*. ∑*n*. *?f n* * *z* ^ *n* **and** *f'* ≡ λ*z*. ∑*n*. *?f' n* * *z* ^ *n*
{
fix *z* :: *complex* **assume** *z*: *norm z* < *K*
from *summable-mult2*[*OF summable'*[*OF z*], *of z*]
have *summable1*: *summable* (λ*n*. *?f' n* * *z* ^ *Suc n*) **by** (*simp add*: *mult-ac*)
hence *summable2*: *summable* (λ*n*. *of-nat n* * *?f n* * *z*^*n*)
unfolding *diffs-def* **by** (*subst* (*asm*) *summable-Suc-iff*)

have (*1* + *z*) * *f' z* = (∑*n*. *?f' n* * *z*^*n*) + (∑*n*. *?f' n* * *z*^*Suc n*)
unfolding *f'-def* **using** *summable' z* **by** (*simp add*: *algebra-simps suminf-mult*)
also have (∑*n*. *?f' n* * *z*^*n*) = (∑*n*. *of-nat* (*Suc n*) * *?f* (*Suc n*) * *z*^*n*)
by (*intro suminf-cong*) (*simp add*: *diffs-def*)
also have (∑*n*. *?f' n* * *z*^*Suc n*) = (∑*n*. *of-nat n* * *?f n* * *z* ^ *n*)
using *summable1 suminf-split-initial-segment*[*OF summable1*] **unfolding** *diffs-def*
by (*subst suminf-split-head*, *subst* (*asm*) *summable-Suc-iff*) *simp-all*
also have (∑*n*. *of-nat* (*Suc n*) * *?f* (*Suc n*) * *z*^*n*) + (∑*n*. *of-nat n* * *?f n* * *z*^*n*) =
(∑*n*. *a* * *?f n* * *z*^*n*)
by (*subst gbinomial-mult-1*, *subst suminf-add*)
(*insert summable'*[*OF z*] *summable2*,
simp-all add: *summable-powser-split-head algebra-simps diffs-def*)
also have ... = *a* * *f z* **unfolding** *f-def*
by (*subst suminf-mult*[*symmetric*]) (*simp-all add*: *summable*[*OF z*] *mult-ac*)
finally have *a* * *f z* = (*1* + *z*) * *f' z* **by** *simp*
} **note** *deriv* = *this*

have [*derivative-intros*]: (*f has-field-derivative f' z*) (*at z*) **if** *norm z* < *of-real K* **for** *z*
unfolding *f-def f'-def* **using** *K that*

by (*intro termdiffs-strong*[*of ?f K z*] *summable-strong*) *simp-all*
have *f 0* = (∑ *n*. *if n = 0 then 1 else 0*) **unfolding** *f-def* **by** (*intro suminf-cong*) *simp*
also have ... = *1* **using** *sums-single*[*of 0* λ-. *1*::*complex*] **unfolding** *sums-iff* **by** *simp*
finally have [*simp*]: *f 0 = 1* **.**

have ∃ *c*. ∀ *z*∈*ball 0 K*. *f z* ∗ (*1* + *z*) *powr* (−*a*) = *c*
proof (*rule has-field-derivative-zero-constant*)
fix *z* :: *complex* **assume** *z'*: *z* ∈ *ball 0 K*
hence *z*: *norm z* < *K* **by** (*simp add*: *dist-0-norm*)
with *K* **have** *nz*: *1* + *z* ≠ *0* **by** (*auto dest*!: *minus-unique*)
from *z K* **have** *norm z* < *1* **by** *simp*
hence (*1* + *z*) ∉ ℝ≤0 **by** (*cases z*) (*auto simp*: *complex-nonpos-Reals-iff*)
hence ((λ*z*. *f z* ∗ (*1* + *z*) *powr* (−*a*)) *has-field-derivative*
f' z ∗ (*1* + *z*) *powr* (−*a*) − *a* ∗ *f z* ∗ (*1* + *z*) *powr* (−*a*−*1*)) (*at z*)
using *z*
by (*auto intro*!: *derivative-eq-intros*)
also from *z* **have** *a* ∗ *f z* = (*1* + *z*) ∗ *f' z* **by** (*rule deriv*)
finally show ((λ*z*. *f z* ∗ (*1* + *z*) *powr* (−*a*)) *has-field-derivative 0*) (*at z within ball 0 K*)
using *nz* **by** (*simp add*: *field-simps powr-diff-complex at-within-open*[*OF z'*])
qed *simp-all*
then obtain *c* **where** *c*: ⋀*z*. *z* ∈ *ball 0 K* ⟹ *f z* ∗ (*1* + *z*) *powr* (−*a*) = *c* **by** *blast*
from *c*[*of 0*] **and** *K* **have** *c* = *1* **by** *simp*
with *c*[*of z*] **have** *f z* = (*1* + *z*) *powr a* **using** *K*
by (*simp add*: *powr-minus-complex field-simps dist-complex-def*)
with *summable K* **show** *?thesis* **unfolding** *f-def* **by** (*simp add*: *sums-iff*)
qed

lemma *gen-binomial-complex'*:
fixes *x y* :: *real* **and** *a* :: *complex*
assumes |*x*| < |*y*|
shows (λ*n*. (*a gchoose n*) ∗ *of-real x*^*n* ∗ *of-real y powr* (*a* − *of-nat n*)) *sums*
of-real (*x* + *y*) *powr a* (**is** *?P x y*)
proof −
{
fix *x y* :: *real* **assume** *xy*: |*x*| < |*y*| *y* ≥ *0*
hence *y* > *0* **by** *simp*
note *xy* = *xy this*
from *xy* **have** (λ*n*. (*a gchoose n*) ∗ *of-real* (*x* / *y*) ^ *n*) *sums* (*1* + *of-real* (*x* / *y*)) *powr a*
by (*intro gen-binomial-complex*) (*simp add*: *norm-divide*)
hence (λ*n*. (*a gchoose n*) ∗ *of-real* (*x* / *y*) ^ *n* ∗ *y powr a*) *sums*
((*1* + *of-real* (*x* / *y*)) *powr a* ∗ *y powr a*)
by (*rule sums-mult2*)
also have (*1* + *complex-of-real* (*x* / *y*)) = *complex-of-real* (*1* + *x*/*y*) **by** *simp*
also from *xy* **have** ... *powr a* ∗ *of-real y powr a* = (... ∗ *y*) *powr a*

```
    by (subst powr-times-real[symmetric]) (simp-all add: field-simps)
   also from xy have complex-of-real (1 + x / y) * complex-of-real y = of-real
(x + y)
    by (simp add: field-simps)
   finally have ?P x y using xy by (simp add: field-simps powr-diff-complex
powr-nat)
 } note A = this

 show ?thesis
 proof (cases y < 0)
  assume y: y < 0
  with assms have xy: x + y < 0 by simp
  with assms have |-x| < |-y| -y ≥ 0 by simp-all
  note A[OF this]
  also have complex-of-real (-x + -y) = - complex-of-real (x + y) by simp
  also from xy assms have ... powr a = (-1) powr -a * of-real (x + y) powr a
   by (subst powr-neg-real-complex) (simp add: abs-real-def split: split-if-asm)
  also {
   fix n :: nat
   from y have (a gchoose n) * of-real (-x) ^ n * of-real (-y) powr (a - of-nat
n) =
                    (a gchoose n) * (-of-real x / -of-real y) ^ n * (- of-real y)
powr a
     by (subst power-divide) (simp add: powr-diff-complex powr-nat)
    also from y have (- of-real y) powr a = (-1) powr -a * of-real y powr a
     by (subst powr-neg-real-complex) simp
     also have -complex-of-real x / -complex-of-real y = complex-of-real x /
complex-of-real y
     by simp
    also have ... ^ n = of-real x ^ n / of-real y ^ n by (simp add: power-divide)
    also have (a gchoose n) * ... * ((-1) powr -a * of-real y powr a) =
              (-1) powr -a * ((a gchoose n) * of-real x ^ n * of-real y powr (a
- n))
     by (simp add: algebra-simps powr-diff-complex powr-nat)
    finally have (a gchoose n) * of-real (- x) ^ n * of-real (- y) powr (a -
of-nat n) =
                  (-1) powr -a * ((a gchoose n) * of-real x ^ n * of-real y powr
(a - of-nat n)) .
  }
  note sums-cong[OF this]
  finally show ?thesis by (simp add: sums-mult-iff)
 qed (insert A[of x y] assms, simp-all add: not-less)
qed

lemma gen-binomial-complex'':
 fixes x y :: real and a :: complex
 assumes |y| < |x|
 shows   (λn. (a gchoose n) * of-real x powr (a - of-nat n) * of-real y ^ n) sums
```

```
            of-real (x + y) powr a
  using gen-binomial-complex'[OF assms] by (simp add: mult-ac add.commute)

lemma gen-binomial-real:
  fixes z :: real
  assumes |z| < 1
  shows   (λn. (a gchoose n) * z^n) sums (1 + z) powr a
proof −
  from assms have norm (of-real z :: complex) < 1 by simp
  from gen-binomial-complex[OF this]
    have (λn. (of-real a gchoose n :: complex) * of-real z ^ n) sums
           (of-real (1 + z)) powr (of-real a) by simp
  also have (of-real (1 + z) :: complex) powr (of-real a) = of-real ((1 + z) powr
a)
    using assms by (subst powr-of-real) simp-all
  also have (of-real a gchoose n :: complex) = of-real (a gchoose n) for n
    by (simp add: gbinomial-def)
  hence (λn. (of-real a gchoose n :: complex) * of-real z ^ n) =
        (λn. of-real ((a gchoose n) * z ^ n)) by (intro ext) simp
  finally show ?thesis by (simp only: sums-of-real-iff)
qed

lemma gen-binomial-real':
  fixes x y a :: real
  assumes |x| < y
  shows   (λn. (a gchoose n) * x^n * y powr (a − of-nat n)) sums (x + y) powr a
proof −
  from assms have y > 0 by simp
  note xy = this assms
  from assms have |x / y| < 1 by simp
  hence (λn. (a gchoose n) * (x / y) ^ n) sums (1 + x / y) powr a
    by (rule gen-binomial-real)
  hence (λn. (a gchoose n) * (x / y) ^ n * y powr a) sums ((1 + x / y) powr a
* y powr a)
    by (rule sums-mult2)
  with xy show ?thesis
    by (simp add: field-simps powr-divide powr-divide2[symmetric] powr-realpow)
qed

lemma one-plus-neg-powr-powser:
  fixes z s :: complex
  assumes norm (z :: complex) < 1
  shows (λn. (−1)^n * ((s + n − 1) gchoose n) * z^n) sums (1 + z) powr (−s)
    using gen-binomial-complex[OF assms, of −s] by (simp add: gbinomial-minus)

lemma gen-binomial-real'':
  fixes x y a :: real
  assumes |y| < x
  shows   (λn. (a gchoose n) * x powr (a − of-nat n) * y^n) sums (x + y) powr a
```

using *gen-binomial-real'*[*OF assms*] **by** (*simp add*: *mult-ac add.commute*)

lemma *sqrt-series'*:
$|z| < a \Longrightarrow (\lambda n.\ ((1/2)\ gchoose\ n) * a\ powr\ (1/2 - real\text{-}of\text{-}nat\ n) * z\ \hat{}\ n)\ sums$

$sqrt\ (a + z :: real)$
using *gen-binomial-real''*[*of z a 1/2*] **by** (*simp add*: *powr-half-sqrt*)

lemma *sqrt-series*:
$|z| < 1 \Longrightarrow (\lambda n.\ ((1/2)\ gchoose\ n) * z\ \hat{}\ n)\ sums\ sqrt\ (1 + z)$
using *gen-binomial-real*[*of z 1/2*] **by** (*simp add*: *powr-half-sqrt*)

end

54 Integral Test for Summability

theory *Integral-Test*
imports *Integration*
begin

The integral test for summability. We show here that for a decreasing non-negative function, the infinite sum over that function evaluated at the natural numbers converges iff the corresponding integral converges.

As a useful side result, we also provide some results on the difference between the integral and the partial sum. (This is useful e.g. for the definition of the Euler-Mascheroni constant)

locale *antimono-fun-sum-integral-diff* =
fixes f :: *real* ⇒ *real*
assumes *dec*: $\bigwedge x\ y.\ x \geq 0 \Longrightarrow x \leq y \Longrightarrow f\ x \geq f\ y$
assumes *nonneg*: $\bigwedge x.\ x \geq 0 \Longrightarrow f\ x \geq 0$
assumes *cont*: *continuous-on* {*0*..} f
begin

definition *sum-integral-diff-series* $n = (\sum k{\leq}n.\ f\ (of\text{-}nat\ k)) - (integral\ \{0..of\text{-}nat\ n\}\ f)$

lemma *sum-integral-diff-series-nonneg*:
sum-integral-diff-series $n \geq 0$
proof −
note *int* = *integrable-continuous-real*[*OF continuous-on-subset*[*OF cont*]]
let *?int* = $\lambda a\ b.$ *integral* {*of-nat a*..*of-nat b*} f
have −*sum-integral-diff-series* n = *?int 0* $n - (\sum k{\leq}n.\ f\ (of\text{-}nat\ k))$
by (*simp add*: *sum-integral-diff-series-def*)
also have *?int 0* $n = (\sum k{<}n.$ *?int* k (*Suc* k))
proof (*induction* n)
case (*Suc* n)
have *?int 0* (*Suc* n) = *?int 0* n + *?int* n (*Suc* n)
by (*intro integral-combine*[*symmetric*] *int*) *simp-all*

```
    with Suc show ?case by simp
  qed simp-all
  also have ... ≤ (∑k<n. integral {of-nat k..of-nat (Suc k)} (λ-::real. f (of-nat
k)))
    by (intro setsum-mono integral-le int) (auto intro: dec)
  also have ... = (∑k<n. f (of-nat k)) by simp
  also have ... − (∑k≤n. f (of-nat k)) = −(∑k∈{..n} − {..<n}. f (of-nat k))
    by (subst setsum-diff) auto
  also have ... ≤ 0 by (auto intro!: setsum-nonneg nonneg)
  finally show sum-integral-diff-series n ≥ 0 by simp
qed

lemma sum-integral-diff-series-antimono:
  assumes m ≤ n
  shows   sum-integral-diff-series m ≥ sum-integral-diff-series n
proof −
  let ?int = λa b. integral {of-nat a..of-nat b} f
  note int = integrable-continuous-real[OF continuous-on-subset[OF cont]]
  have d-mono: sum-integral-diff-series (Suc n) ≤ sum-integral-diff-series n for n
  proof −
    fix n :: nat
    have sum-integral-diff-series (Suc n) − sum-integral-diff-series n =
          f (of-nat (Suc n)) + (?int 0 n − ?int 0 (Suc n))
      unfolding sum-integral-diff-series-def by (simp add: algebra-simps)
    also have ?int 0 n − ?int 0 (Suc n) = −?int n (Suc n)
      by (subst integral-combine [symmetric, of of-nat 0 of-nat n of-nat (Suc n)])
        (auto intro!: int simp: algebra-simps)
     also have ?int n (Suc n) ≥ integral {of-nat n..of-nat (Suc n)} (λ-::real. f
(of-nat (Suc n)))
      by (intro integral-le int) (auto intro: dec)
    hence f (of-nat (Suc n)) + −?int n (Suc n) ≤ 0 by (simp add: algebra-simps)
     finally show sum-integral-diff-series (Suc n) ≤ sum-integral-diff-series n by
simp
  qed
  with assms show ?thesis
    by (induction rule: inc-induct) (auto intro: order.trans[OF - d-mono])
qed

lemma sum-integral-diff-series-Bseq: Bseq sum-integral-diff-series
proof −
  from sum-integral-diff-series-nonneg and sum-integral-diff-series-antimono
    have norm (sum-integral-diff-series n) ≤ sum-integral-diff-series 0 for n by
simp
  thus Bseq sum-integral-diff-series by (rule BseqI')
qed

lemma sum-integral-diff-series-monoseq: monoseq sum-integral-diff-series
  using sum-integral-diff-series-antimono unfolding monoseq-def by blast
```

lemma *sum-integral-diff-series-convergent*: *convergent sum-integral-diff-series*
using *sum-integral-diff-series-Bseq sum-integral-diff-series-monoseq*
by (*blast intro*!: *Bseq-monoseq-convergent*)

lemma *integral-test*:
summable (λ*n*. *f* (*of-nat n*)) ⟷ *convergent* (λ*n*. *integral* {*0*..*of-nat n*} *f*)
proof −
have *summable* (λ*n*. *f* (*of-nat n*)) ⟷ *convergent* (λ*n*. ∑ *k*≤*n*. *f* (*of-nat k*))
by (*simp add*: *summable-iff-convergent'*)
also have ... ⟷ *convergent* (λ*n*. *integral* {*0*..*of-nat n*} *f*)
proof
assume *convergent* (λ*n*. ∑ *k*≤*n*. *f* (*of-nat k*))
from *convergent-diff*[*OF this sum-integral-diff-series-convergent*]
show *convergent* (λ*n*. *integral* {*0*..*of-nat n*} *f*)
unfolding *sum-integral-diff-series-def* **by** *simp*
next
assume *convergent* (λ*n*. *integral* {*0*..*of-nat n*} *f*)
from *convergent-add*[*OF this sum-integral-diff-series-convergent*]
show *convergent* (λ*n*. ∑ *k*≤*n*. *f* (*of-nat k*)) **unfolding** *sum-integral-diff-series-def*
by *simp*
qed
finally show *?thesis* **by** *simp*
qed

end

end

55 Harmonic Numbers

theory *Harmonic-Numbers*
imports
Complex-Transcendental
Summation
Integral-Test
begin

The definition of the Harmonic Numbers and the Euler-Mascheroni constant. Also provides a reasonably accurate approximation of *ln 2* and the Euler-Mascheroni constant.

lemma *ln-2-less-1*: *ln 2* < (*1*::*real*)
proof −
have *2* < *5*/(*2*::*real*) **by** *simp*
also have *5*/*2* ≤ *exp* (*1*::*real*) **using** *exp-lower-taylor-quadratic*[*of 1*, *simplified*]
by *simp*
finally have *exp* (*ln 2*) < *exp* (*1*::*real*) **by** *simp*
thus *ln 2* < (*1*::*real*) **by** (*subst* (*asm*) *exp-less-cancel-iff*) *simp*
qed

lemma *setsum-Suc-diff'*:
fixes $f :: nat \Rightarrow 'a::ab\text{-}group\text{-}add$
assumes $m \leq n$
shows $(\sum i = m..<n.\ f\ (Suc\ i) - f\ i) = f\ n - f\ m$
using *assms* **by** (*induct n*) (*auto simp*: *le-Suc-eq*)

55.1 The Harmonic numbers

definition $harm :: nat \Rightarrow 'a :: real\text{-}normed\text{-}field$ **where**
$harm\ n = (\sum k=1..n.\ inverse\ (of\text{-}nat\ k))$

lemma *harm-altdef*: $harm\ n = (\sum k<n.\ inverse\ (of\text{-}nat\ (Suc\ k)))$
unfolding *harm-def* **by** (*induction n*) *simp-all*

lemma *harm-Suc*: $harm\ (Suc\ n) = harm\ n + inverse\ (of\text{-}nat\ (Suc\ n))$
by (*simp add*: *harm-def*)

lemma *harm-nonneg*: $harm\ n \geq (0 :: 'a :: \{real\text{-}normed\text{-}field, linordered\text{-}field\})$
unfolding *harm-def* **by** (*intro setsum-nonneg*) *simp-all*

lemma *harm-pos*: $n > 0 \Longrightarrow harm\ n > (0 :: 'a :: \{real\text{-}normed\text{-}field, linordered\text{-}field\})$
unfolding *harm-def* **by** (*intro setsum-pos*) *simp-all*

lemma *of-real-harm*: $of\text{-}real\ (harm\ n) = harm\ n$
unfolding *harm-def* **by** *simp*

lemma *norm-harm*: $norm\ (harm\ n) = harm\ n$
by (*subst of-real-harm* [*symmetric*]) (*simp add*: *harm-nonneg*)

lemma *harm-expand*:
$harm\ 0 = 0$
$harm\ (Suc\ 0) = 1$
$harm\ (numeral\ n) = harm\ (pred\text{-}numeral\ n) + inverse\ (numeral\ n)$
proof −
have $numeral\ n = Suc\ (pred\text{-}numeral\ n)$ **by** *simp*
also have $harm\ \ldots = harm\ (pred\text{-}numeral\ n) + inverse\ (numeral\ n)$
by (*subst harm-Suc*, *subst numeral-eq-Suc*[*symmetric*]) *simp*
finally show $harm\ (numeral\ n) = harm\ (pred\text{-}numeral\ n) + inverse\ (numeral\ n)$ **.**
qed (*simp-all add*: *harm-def*)

lemma *not-convergent-harm*: $\neg convergent\ (harm :: nat \Rightarrow 'a :: real\text{-}normed\text{-}field)$
proof −
have $convergent\ (\lambda n.\ norm\ (harm\ n :: 'a)) \longleftrightarrow$
$convergent\ (harm :: nat \Rightarrow real)$ **by** (*simp add*: *norm-harm*)
also have $\ldots \longleftrightarrow convergent\ (\lambda n.\ \sum k=Suc\ 0..Suc\ n.\ inverse\ (of\text{-}nat\ k) :: real)$
unfolding *harm-def*[*abs-def*] **by** (*subst convergent-Suc-iff*) *simp-all*
also have $\ldots \longleftrightarrow convergent\ (\lambda n.\ \sum k\leq n.\ inverse\ (of\text{-}nat\ (Suc\ k)) :: real)$

by (*subst setsum-shift-bounds-cl-Suc-ivl*) (*simp add*: *atLeast0AtMost*)
also have ... $\longleftrightarrow$ *summable* (λn. *inverse* (*of-nat n*) :: *real*)
by (*subst summable-Suc-iff* [*symmetric*]) (*simp add*: *summable-iff-convergent'*)
also have ¬... **by** (*rule not-summable-harmonic*)
finally show *?thesis* **by** (*blast dest*: *convergent-norm*)
qed

lemma *harm-pos-iff* [*simp*]: *harm n* > (*0* :: *'a* :: {*real-normed-field*,*linordered-field*})
$\longleftrightarrow$ *n* > *0*
by (*rule iffI*, *cases n*, *simp add*: *harm-expand*, *simp*, *rule harm-pos*)

lemma *ln-diff-le-inverse*:
assumes *x* ≥ (*1*::*real*)
shows *ln* (*x* + *1*) − *ln x* < *1* / *x*
proof −
from *assms* **have** ∃ *z*>*x*. *z* < *x* + *1* ∧ *ln* (*x* + *1*) − *ln x* = (*x* + *1* − *x*) ∗ *inverse z*
by (*intro MVT2*) (*auto intro*!: *derivative-eq-intros simp*: *field-simps*)
then obtain *z* **where** *z*: *z* > *x z* < *x* + *1 ln* (*x* + *1*) − *ln x* = *inverse z* **by** *auto*
have *ln* (*x* + *1*) − *ln x* = *inverse z* **by** *fact*
also from *z*(*1*,*2*) *assms* **have** ... < *1* / *x* **by** (*simp add*: *field-simps*)
finally show *?thesis* **.**
qed

lemma *ln-le-harm*: *ln* (*real n* + *1*) ≤ (*harm n* :: *real*)
proof (*induction n*)
fix *n* **assume** *IH*: *ln* (*real n* + *1*) ≤ *harm n*
have *ln* (*real* (*Suc n*) + *1*) = *ln* (*real n* + *1*) + (*ln* (*real n* + *2*) − *ln* (*real n* + *1*)) **by** *simp*
also have (*ln* (*real n* + *2*) − *ln* (*real n* + *1*)) ≤ *1* / *real* (*Suc n*)
using *ln-diff-le-inverse*[*of real n* + *1*] **by** (*simp add*: *add-ac*)
also note *IH*
also have *harm n* + *1* / *real* (*Suc n*) = *harm* (*Suc n*) **by** (*simp add*: *harm-Suc field-simps*)
finally show *ln* (*real* (*Suc n*) + *1*) ≤ *harm* (*Suc n*) **by** − *simp*
qed (*simp-all add*: *harm-def*)

55.2 The Euler–Mascheroni constant

The limit of the difference between the partial harmonic sum and the natural logarithm (approximately 0.577216). This value occurs e.g. in the definition of the Gamma function.

definition *euler-mascheroni* :: *'a* :: *real-normed-algebra-1* **where**
euler-mascheroni = *of-real* (*lim* (λn. *harm n* − *ln* (*of-nat n*)))

lemma *of-real-euler-mascheroni* [*simp*]: *of-real euler-mascheroni* = *euler-mascheroni*
by (*simp add*: *euler-mascheroni-def*)

interpretation *euler-mascheroni*: *antimono-fun-sum-integral-diff* $\lambda x.$ *inverse* $(x + 1)$
by *unfold-locales* (*auto intro*!: *continuous-intros*)

lemma *euler-mascheroni-sum-integral-diff-series*:
euler-mascheroni.sum-integral-diff-series n = *harm* (*Suc* n) − *ln* (*of-nat* (*Suc* n))
proof −
have *harm* (*Suc* n) = $(\sum k{=}0..n.$ *inverse* (*of-nat* $k + 1$) :: *real*) **unfolding** *harm-def*
unfolding *One-nat-def* **by** (*subst setsum-shift-bounds-cl-Suc-ivl*) (*simp add*: *add-ac*)
moreover have ((λx. *inverse* $(x + 1)$ :: *real*) *has-integral ln* (*of-nat* $n + 1$) − *ln* $(0 + 1)$)
$\{0..$*of-nat* $n\}$
by (*intro fundamental-theorem-of-calculus*)
(*auto intro*!: *derivative-eq-intros simp*: *divide-inverse*
has-field-derivative-iff-has-vector-derivative[*symmetric*])
hence *integral* $\{0..$*of-nat* $n\}$ (λx. *inverse* $(x + 1)$ :: *real*) = *ln* (*of-nat* (*Suc* n))
by (*auto dest*!: *integral-unique*)
ultimately show *?thesis*
by (*simp add*: *euler-mascheroni.sum-integral-diff-series-def atLeast0AtMost*)
qed

lemma *euler-mascheroni-sequence-decreasing*:
$m > 0 \Longrightarrow m \leq n \Longrightarrow$ *harm* n − *ln* (*of-nat* n) ≤ *harm* m − *ln* (*of-nat* m :: *real*)
by (*cases m*, *simp*, *cases n*, *simp*, *hypsubst*,
subst (*1 2*) *euler-mascheroni-sum-integral-diff-series* [*symmetric*],
rule euler-mascheroni.sum-integral-diff-series-antimono, *simp*)

lemma *euler-mascheroni-sequence-nonneg*:
$n > 0 \Longrightarrow$ *harm* n − *ln* (*of-nat* n) ≥ (*0*::*real*)
by (*cases n*, *simp*, *hypsubst*, *subst euler-mascheroni-sum-integral-diff-series* [*symmetric*],
rule euler-mascheroni.sum-integral-diff-series-nonneg)

lemma *euler-mascheroni-convergent*: *convergent* (λn. *harm* n − *ln* (*of-nat* n) :: *real*)
proof −
have *A*: (λn. *harm* (*Suc* n) − *ln* (*of-nat* (*Suc* n))) =
euler-mascheroni.sum-integral-diff-series
by (*subst euler-mascheroni-sum-integral-diff-series* [*symmetric*]) (*rule refl*)
have *convergent* (λn. *harm* (*Suc* n) − *ln* (*of-nat* (*Suc* n) :: *real*))
by (*subst A*) (*fact euler-mascheroni.sum-integral-diff-series-convergent*)
thus *?thesis* **by** (*subst* (*asm*) *convergent-Suc-iff*)
qed

lemma *euler-mascheroni-LIMSEQ*:
(λn. *harm* n − *ln* (*of-nat* n) :: *real*) ⟶ *euler-mascheroni*
unfolding *euler-mascheroni-def*

by (*simp add*: *convergent-LIMSEQ-iff* [*symmetric*] *euler-mascheroni-convergent*)

lemma *euler-mascheroni-LIMSEQ-of-real*:
(λ*n*. *of-real* (*harm n* − *ln* (*of-nat n*))) ⟶
(*euler-mascheroni* :: *'a* :: {*real-normed-algebra-1*, *topological-space*})
proof −
have (λ*n*. *of-real* (*harm n* − *ln* (*of-nat n*))) ⟶ (*of-real* (*euler-mascheroni*) :: *'a*)
by (*intro tendsto-of-real euler-mascheroni-LIMSEQ*)
thus *?thesis* **by** *simp*
qed

lemma *euler-mascheroni-sum*:
(λ*n*. *inverse* (*of-nat* (*n*+1)) + *ln* (*of-nat* (*n*+1)) − *ln* (*of-nat* (*n*+2)) :: *real*)
sums euler-mascheroni
using *sums-add*[*OF telescope-sums*[*OF LIMSEQ-Suc*[*OF euler-mascheroni-LIMSEQ*]]
telescope-sums'[*OF LIMSEQ-inverse-real-of-nat*]]
by (*simp-all add*: *harm-def algebra-simps*)

lemma *alternating-harmonic-series-sums*: (λ*k*. (−1) ^ *k* / *real-of-nat* (*Suc k*)) *sums ln 2*
proof −
let *?f* = λ*n*. *harm n* − *ln* (*real-of-nat n*)
let *?g* = λ*n*. *if even n then 0 else* (2::*real*)
let *?em* = λ*n*. *harm n* − *ln* (*real-of-nat n*)
have *eventually* (λ*n*. *?em* (2∗*n*) − *?em n* + *ln 2* = (∑ *k*<2∗*n*. (−1) ^ *k* / *real-of-nat* (*Suc k*))) *at-top*
using *eventually-gt-at-top*[*of* 0::*nat*]
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
have (∑ *k*<2∗*n*. (−1) ^ *k* / *real-of-nat* (*Suc k*)) =
(∑ *k*<2∗*n*. ((−1) ^ *k* + *?g k*) / *of-nat* (*Suc k*)) − (∑ *k*<2∗*n*. *?g k* / *of-nat* (*Suc k*))
by (*simp add*: *setsum.distrib algebra-simps divide-inverse*)
also have (∑ *k*<2∗*n*. ((−1) ^ *k* + *?g k*) / *real-of-nat* (*Suc k*)) = *harm* (2∗*n*)
unfolding *harm-altdef* **by** (*intro setsum.cong*) (*auto simp*: *field-simps*)
also have (∑ *k*<2∗*n*. *?g k* / *real-of-nat* (*Suc k*)) = (∑ *k*|*k*<2∗*n* ∧ *odd k*. *?g k* / *of-nat* (*Suc k*))
by (*intro setsum.mono-neutral-right*) *auto*
also have ... = (∑ *k*|*k*<2∗*n* ∧ *odd k*. *2* / (*real-of-nat* (*Suc k*)))
by (*intro setsum.cong*) *auto*
also have (∑ *k*|*k*<2∗*n* ∧ *odd k*. *2* / (*real-of-nat* (*Suc k*))) = *harm n*
unfolding *harm-altdef*
by (*intro setsum.reindex-cong*[*of* λ*n*. *2*∗*n*+*1*]) (*auto simp*: *inj-on-def field-simps elim*!: *oddE*)
also have *harm* (2∗*n*) − *harm n* = *?em* (2∗*n*) − *?em n* + *ln 2* **using** *n*
by (*simp-all add*: *algebra-simps ln-mult*)
finally show *?em* (2∗*n*) − *?em n* + *ln 2* = (∑ *k*<2∗*n*. (−1) ^ *k* / *real-of-nat* (*Suc k*)) **..**

qed
moreover have $(\lambda n.\ ?em\ (2*n) - ?em\ n + ln\ (2::real)) \longrightarrow euler\text{-}mascheroni - euler\text{-}mascheroni + ln\ 2$
by (*intro tendsto-intros euler-mascheroni-LIMSEQ filterlim-compose*[*OF euler-mascheroni-LIMSEQ*] *filterlim-subseq*) (*auto simp*: *subseq-def*)
hence $(\lambda n.\ ?em\ (2*n) - ?em\ n + ln\ (2::real)) \longrightarrow ln\ 2$ **by** *simp*
ultimately have $(\lambda n.\ (\sum k{<}2*n.\ (-1)\ \hat{}\ k\ /\ real\text{-}of\text{-}nat\ (Suc\ k))) \longrightarrow ln\ 2$
by (*rule Lim-transform-eventually*)

moreover have *summable* $(\lambda k.\ (-1)\ \hat{}\ k * inverse\ (real\text{-}of\text{-}nat\ (Suc\ k)))$
using *LIMSEQ-inverse-real-of-nat*
by (*intro summable-Leibniz*(1) *decseq-imp-monoseq decseq-SucI*) *simp-all*
hence *A*: $(\lambda n.\ \sum k{<}n.\ (-1)\ \hat{}\ k\ /\ real\text{-}of\text{-}nat\ (Suc\ k)) \longrightarrow (\sum k.\ (-1)\ \hat{}\ k\ /\ real\text{-}of\text{-}nat\ (Suc\ k))$
by (*simp add*: *summable-sums-iff divide-inverse sums-def*)
from *filterlim-compose*[*OF this filterlim-subseq*[*of op* $*$ (2::*nat*)]]
have $(\lambda n.\ \sum k{<}2*n.\ (-1)\ \hat{}\ k\ /\ real\text{-}of\text{-}nat\ (Suc\ k)) \longrightarrow (\sum k.\ (-1)\ \hat{}\ k\ /\ real\text{-}of\text{-}nat\ (Suc\ k))$
by (*simp add*: *subseq-def*)
ultimately have $(\sum k.\ (-1)\ \hat{}\ k\ /\ real\text{-}of\text{-}nat\ (Suc\ k)) = ln\ 2$ **by** (*intro LIMSEQ-unique*)
with *A* **show** *?thesis* **by** (*simp add*: *sums-def*)
qed

lemma *alternating-harmonic-series-sums'*:
$(\lambda k.\ inverse\ (real\text{-}of\text{-}nat\ (2*k+1)) - inverse\ (real\text{-}of\text{-}nat\ (2*k+2)))\ sums\ ln\ 2$
unfolding *sums-def*
proof (*rule Lim-transform-eventually*)
show $(\lambda n.\ \sum k{<}2*n.\ (-1)\ \hat{}\ k\ /\ (real\text{-}of\text{-}nat\ (Suc\ k))) \longrightarrow ln\ 2$
using *alternating-harmonic-series-sums* **unfolding** *sums-def*
by (*rule filterlim-compose*) (*rule mult-nat-left-at-top*, *simp*)
show *eventually* $(\lambda n.\ (\sum k{<}2*n.\ (-1)\ \hat{}\ k\ /\ (real\text{-}of\text{-}nat\ (Suc\ k))) = (\sum k{<}n.\ inverse\ (real\text{-}of\text{-}nat\ (2*k+1)) - inverse\ (real\text{-}of\text{-}nat\ (2*k+2))))$ *sequentially*
proof (*intro always-eventually allI*)
fix *n* :: *nat*
show $(\sum k{<}2*n.\ (-1)\ \hat{}\ k\ /\ (real\text{-}of\text{-}nat\ (Suc\ k))) = (\sum k{<}n.\ inverse\ (real\text{-}of\text{-}nat\ (2*k+1)) - inverse\ (real\text{-}of\text{-}nat\ (2*k+2)))$
by (*induction n*) (*simp-all add*: *inverse-eq-divide*)
qed
qed

55.3 Bounds on the Euler–Mascheroni constant

lemma *ln-inverse-approx-le*:
assumes $(x::real) > 0\ a > 0$
shows $ln\ (x + a) - ln\ x \leq a * (inverse\ x + inverse\ (x + a))/2$ (**is** - $\leq$ *?A*)
proof −
def $f' \equiv (inverse\ (x + a) - inverse\ x)/a$

have *f'-nonpos*: $f' \leq 0$ **using** *assms* **by** (*simp add*: *f'-def divide-simps*)
let *?f* = $\lambda t.\ (t - x) * f' + inverse\ x$
let *?F* = $\lambda t.\ (t - x)\hat{}2 * f' / 2 + t * inverse\ x$
have *diff*: $\forall\, t{\in}\{x..x{+}a\}.$ (*?F has-vector-derivative ?f t*)
(*at t within* $\{x..x{+}a\}$) **using** *assms*
by (*auto intro*!: *derivative-eq-intros*
simp: *has-field-derivative-iff-has-vector-derivative*[*symmetric*])
from *assms* **have** (*?f has-integral* (*?F* $(x{+}a)$ − *?F x*)) $\{x..x{+}a\}$
by (*intro fundamental-theorem-of-calculus*[*OF - diff*])
(*auto simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*] *field-simps*
intro!: *derivative-eq-intros*)
also have *?F* $(x{+}a)$ − *?F x* = $(a{*}2 + f'{*}a^2{*}x) / (2{*}x)$ **using** *assms* **by** (*simp add*: *field-simps*)
also have $f'{*}a\hat{}2 = -(a\hat{}2) / (x{*}(x + a))$ **using** *assms*
by (*simp add*: *divide-simps f'-def power2-eq-square*)
also have $(a{*}2 + -a^2/(x{*}(x{+}a)){*}x) / (2{*}x) =$ *?A* **using** *assms*
by (*simp add*: *divide-simps power2-eq-square*) (*simp add*: *algebra-simps*)
finally have *int1*: $((\lambda t.\ (t - x) * f' + inverse\ x)\ has\text{-}integral\ ?A)\ \{x..x + a\}$ **.**

from *assms* **have** *int2*: (*inverse has-integral* (*ln* $(x + a)$ − *ln x*)) $\{x..x{+}a\}$
by (*intro fundamental-theorem-of-calculus*)
(*auto simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*] *divide-simps*
intro!: *derivative-eq-intros*)
hence *ln* $(x + a) - ln\ x = integral\ \{x..x{+}a\}\ inverse$ **by** (*simp add*: *integral-unique*)
also have *ineq*: $\forall\, xa{\in}\{x..x + a\}.\ inverse\ xa \leq (xa - x) * f' + inverse\ x$
proof
fix *t* **assume** *t'*: $t \in \{x..x{+}a\}$
with *assms* **have** *t*: $0 \leq (t - x) / a\ (t - x) / a \leq 1$ **by** *simp-all*
have *inverse t* = *inverse* $((1 - (t - x) / a) *_R x + ((t - x) / a) *_R (x + a))$ (**is** - = *?A*)
using *assms t'* **by** (*simp add*: *field-simps*)
also from *assms* **have** *convex-on* $\{x..x{+}a\}$ *inverse* **by** (*intro convex-on-inverse*) *auto*
from *convex-onD-Icc*[*OF this - t*] *assms*
have $?A \leq (1 - (t - x) / a) * inverse\ x + (t - x) / a * inverse\ (x + a)$ **by** *simp*
also have $\ldots = (t - x) * f' + inverse\ x$ **using** *assms*
by (*simp add*: *f'-def divide-simps*) (*simp add*: *f'-def field-simps*)
finally show *inverse t* ≤ $(t - x) * f' + inverse\ x$ **.**
qed
hence *integral* $\{x..x{+}a\}$ *inverse* ≤ *integral* $\{x..x{+}a\}$ *?f* **using** *f'-nonpos assms*
by (*intro integral-le has-integral-integrable*[*OF int1*] *has-integral-integrable*[*OF int2*] *ineq*)
also have $\ldots$ = *?A* **using** *int1* **by** (*rule integral-unique*)
finally show *?thesis* **.**
qed

lemma *ln-inverse-approx-ge*:
assumes $(x{::}real) > 0\ x < y$

shows *ln y − ln x ≥ 2 * (y − x) / (x + y)* (**is** - ≥ *?A*)
proof −
def *m* ≡ *(x+y)/2*
def *f′* ≡ *−inverse (m^2)*
from *assms* **have** *m*: *m > 0* **by** (*simp add*: *m-def*)
let *?F* = *λt. (t − m)^2 * f′ / 2 + t / m*
from *assms* **have** *((λt. (t − m) * f′ + inverse m) has-integral (?F y − ?F x)) {x..y}*
by (*intro fundamental-theorem-of-calculus*)
(*auto simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*] *divide-simps intro*!: *derivative-eq-intros*)
also from *m* **have** *?F y − ?F x = ((y − m)^2 − (x − m)^2) * f′ / 2 + (y − x) / m*
by (*simp add*: *field-simps*)
also have *((y − m)^2 − (x − m)^2) = 0* **by** (*simp add*: *m-def power2-eq-square field-simps*)
also have *0 * f′ / 2 + (y − x) / m = ?A* **by** (*simp add*: *m-def*)
finally have *int1*: *((λt. (t − m) * f′ + inverse m) has-integral ?A) {x..y}* **.**

from *assms* **have** *int2*: (*inverse has-integral (ln y − ln x)) {x..y}*
by (*intro fundamental-theorem-of-calculus*)
(*auto simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*] *divide-simps intro*!: *derivative-eq-intros*)
hence *ln y − ln x = integral {x..y} inverse* **by** (*simp add*: *integral-unique*)
also have *ineq*: *∀ xa∈{x..y}. inverse xa ≥ (xa − m) * f′ + inverse m*
proof
fix *t* **assume** *t*: *t ∈ {x..y}*
from *t assms* **have** *inverse t − inverse m ≥ f′ * (t − m)*
by (*intro convex-on-imp-above-tangent*[*of {0<..}*] *convex-on-inverse*)
(*auto simp*: *m-def interior-open f′-def power2-eq-square intro*!: *derivative-eq-intros*)
thus *(t − m) * f′ + inverse m ≤ inverse t* **by** (*simp add*: *algebra-simps*)
qed
hence *integral {x..y} inverse ≥ integral {x..y} (λt. (t − m) * f′ + inverse m)*
using *int1 int2* **by** (*intro integral-le has-integral-integrable*)
also have *integral {x..y} (λt. (t − m) * f′ + inverse m) = ?A*
using *integral-unique*[*OF int1*] **by** *simp*
finally show *?thesis* **.**
qed

lemma *euler-mascheroni-lower*:
*euler-mascheroni ≥ harm (Suc n) − ln (real-of-nat (n + 2)) + 1/real-of-nat (2 * (n + 2))*
and *euler-mascheroni-upper*:
*euler-mascheroni ≤ harm (Suc n) − ln (real-of-nat (n + 2)) + 1/real-of-nat (2 * (n + 1))*
proof −
def *D* ≡ *λn. inverse (of-nat (n+1)) + ln (of-nat (n+1)) − ln (of-nat (n+2))* :: *real*

let *?g* = λ*n*. *ln* (*of-nat* (*n*+2)) − *ln* (*of-nat* (*n*+1)) − *inverse* (*of-nat* (*n*+1)) :: *real*
def *inv* ≡ λ*n*. *inverse* (*real-of-nat n*)
fix *n* :: *nat*
note *summable* = *sums-summable*[*OF euler-mascheroni-sum*, *folded D-def*]
have *sums*: (λ*k*. (*inv* (*Suc* (*k* + (*n*+1))) − *inv* (*Suc* (*Suc k* + (*n*+1))))/2) *sums* ((*inv* (*Suc* (0 + (*n*+1))) − 0)/2)
unfolding *inv-def*
by (*intro sums-divide telescope-sums′ LIMSEQ-ignore-initial-segment LIMSEQ-inverse-real-of-nat*)
have *sums′*: (λ*k*. (*inv* (*Suc* (*k* + *n*)) − *inv* (*Suc* (*Suc k* + *n*)))/2) *sums* ((*inv* (*Suc* (0 + *n*)) − 0)/2)
unfolding *inv-def*
by (*intro sums-divide telescope-sums′ LIMSEQ-ignore-initial-segment LIMSEQ-inverse-real-of-nat*)
from *euler-mascheroni-sum* **have** *euler-mascheroni* = (∑ *k*. *D k*)
by (*simp add*: *sums-iff D-def*)
also have ... = (∑ *k*. *D* (*k* + *Suc n*)) + (∑ *k*≤*n*. *D k*)
by (*subst suminf-split-initial-segment*[*OF summable*, *of Suc n*], *subst lessThan-Suc-atMost*) *simp*
finally have *sum*: (∑ *k*≤*n*. *D k*) − *euler-mascheroni* = −(∑ *k*. *D* (*k* + *Suc n*)) **by** *simp*

note *sum*
also have ... ≤ −(∑ *k*. (*inv* (*k* + *Suc n* + 1) − *inv* (*k* + *Suc n* + 2)) / 2)
proof (*intro le-imp-neg-le suminf-le allI summable-ignore-initial-segment*[*OF summable*])
fix *k′* :: *nat*
def *k* ≡ *k′* + *Suc n*
hence *k*: *k* > 0 **by** (*simp add*: *k-def*)
have *real-of-nat* (*k*+1) > 0 **by** (*simp add*: *k-def*)
with *ln-inverse-approx-le*[*OF this zero-less-one*]
have *ln* (*of-nat k* + 2) − *ln* (*of-nat k* + 1) ≤ (*inv* (*k*+1) + *inv* (*k*+2))/2
by (*simp add*: *inv-def add-ac*)
hence (*inv* (*k*+1) − *inv* (*k*+2))/2 ≤ *inv* (*k*+1) + *ln* (*of-nat* (*k*+1)) − *ln* (*of-nat* (*k*+2))
by (*simp add*: *field-simps*)
also have ... = *D k* **unfolding** *D-def inv-def* **..**
finally show *D* (*k′* + *Suc n*) ≥ (*inv* (*k′* + *Suc n* + 1) − *inv* (*k′* + *Suc n* + 2)) / 2
by (*simp add*: *k-def*)
from *sums-summable*[*OF sums*]
show *summable* (λ*k*. (*inv* (*k* + *Suc n* + 1) − *inv* (*k* + *Suc n* + 2))/2) **by** *simp*
qed
also from *sums* **have** ... = −*inv* (*n*+2) / 2 **by** (*simp add*: *sums-iff*)
finally have *euler-mascheroni* ≥ (∑ *k*≤*n*. *D k*) + 1 / (*of-nat* (2 ∗ (*n*+2)))
by (*simp add*: *inv-def field-simps*)
also have (∑ *k*≤*n*. *D k*) = *harm* (*Suc n*) − (∑ *k*≤*n*. *ln* (*real-of-nat* (*Suc k*+1)) − *ln* (*of-nat* (*k*+1)))
unfolding *harm-altdef D-def* **by** (*subst lessThan-Suc-atMost*) (*simp add*: *setsum.distrib setsum-subtractf*)

also have $(\sum k{\le}n.\ ln\ (real\text{-}of\text{-}nat\ (Suc\ k{+}1)) - ln\ (of\text{-}nat\ (k{+}1))) = ln\ (of\text{-}nat\ (n{+}2))$
by (*subst atLeast0AtMost* [*symmetric*], *subst setsum-Suc-diff*) *simp-all*
finally show *euler-mascheroni* $\geq$ *harm* (*Suc n*) $-$ *ln* (*real-of-nat* (*n* + *2*)) + *1*/*real-of-nat* (*2* $*$ (*n* + *2*))
by *simp*

note *sum*
also have $-(\sum k.\ D\ (k + Suc\ n)) \geq -(\sum k.\ (inv\ (Suc\ (k + n)) - inv\ (Suc\ (Suc\ k + n)))/2)$
proof (*intro le-imp-neg-le suminf-le allI summable-ignore-initial-segment*[*OF summable*])
fix k' :: *nat*
def $k \equiv k' + Suc\ n$
hence *k*: $k > 0$ **by** (*simp add*: *k-def*)
have *real-of-nat* $(k{+}1) > 0$ **by** (*simp add*: *k-def*)
from *ln-inverse-approx-ge*[*of of-nat k + 1 of-nat k + 2*]
have *2* / (*2* $*$ *real-of-nat k* + *3*) $\leq$ *ln* (*of-nat* $(k{+}2)$) $-$ *ln* (*real-of-nat* $(k{+}1)$)
by (*simp add*: *add-ac*)
hence *D k* $\leq$ *1* / *real-of-nat* $(k{+}1)$ $-$ *2* / (*2* $*$ *real-of-nat k* + *3*)
by (*simp add*: *D-def inverse-eq-divide inv-def*)
also have ... $=$ *inv* $((k{+}1){*}(2{*}k{+}3))$ **unfolding** *inv-def* **by** (*simp add*: *field-simps*)
also have ... $\leq$ *inv* $(2{*}k{*}(k{+}1))$ **unfolding** *inv-def* **using** *k*
by (*intro le-imp-inverse-le*)
(*simp add*: *algebra-simps*, *simp del*: *of-nat-add*)
also have ... $=$ $(inv\ k - inv\ (k{+}1))/2$ **unfolding** *inv-def* **using** *k*
by (*simp add*: *divide-simps del*: *of-nat-mult*) (*simp add*: *algebra-simps*)
finally show *D k* $\leq$ $(inv\ (Suc\ (k' + n)) - inv\ (Suc\ (Suc\ k' + n)))/2$ **unfolding** *k-def* **by** *simp*
next
from *sums-summable*[*OF sums'*]
show *summable* $(\lambda k.\ (inv\ (Suc\ (k + n)) - inv\ (Suc\ (Suc\ k + n)))/2)$ **by** *simp*
qed
also from *sums'* **have** $(\sum k.\ (inv\ (Suc\ (k + n)) - inv\ (Suc\ (Suc\ k + n)))/2) = inv\ (n{+}1)/2$
by (*simp add*: *sums-iff*)
finally have *euler-mascheroni* $\leq$ $(\sum k{\le}n.\ D\ k) + 1\ /\ of\text{-}nat\ (2 * (n{+}1))$
by (*simp add*: *inv-def field-simps*)
also have $(\sum k{\le}n.\ D\ k) = harm\ (Suc\ n) - (\sum k{\le}n.\ ln\ (real\text{-}of\text{-}nat\ (Suc\ k{+}1)) - ln\ (of\text{-}nat\ (k{+}1)))$
unfolding *harm-altdef D-def* **by** (*subst lessThan-Suc-atMost*) (*simp add*: *setsum.distrib setsum-subtractf*)
also have $(\sum k{\le}n.\ ln\ (real\text{-}of\text{-}nat\ (Suc\ k{+}1)) - ln\ (of\text{-}nat\ (k{+}1))) = ln\ (of\text{-}nat\ (n{+}2))$
by (*subst atLeast0AtMost* [*symmetric*], *subst setsum-Suc-diff*) *simp-all*
finally show *euler-mascheroni* $\leq$ *harm* (*Suc n*) $-$ *ln* (*real-of-nat* (*n* + *2*)) + *1*/*real-of-nat* (*2* $*$ (*n* + *1*))
by *simp*

qed

lemma *euler-mascheroni-pos*: *euler-mascheroni* > (*0*::*real*)
using *euler-mascheroni-lower*[*of 0*] *ln-2-less-1* **by** (*simp add*: *harm-def*)

context
begin

private lemma *ln-approx-aux*:
fixes *n* :: *nat* **and** *x* :: *real*
defines *y* ≡ (*x*−*1*)/(*x*+*1*)
assumes *x*: *x* > *0* *x* ≠ *1*
shows *inverse* (*2*∗*y*^(*2*∗*n*+*1*)) ∗ (*ln x* − (∑ *k*<*n*. *2*∗*y*^(*2*∗*k*+*1*) / *of-nat* (*2*∗*k*+*1*)))
∈
{*0*..(*1* / (*1* − *y*^*2*) / *of-nat* (*2*∗*n*+*1*))}
proof −
from *x* **have** *norm-y*: *norm y* < *1* **unfolding** *y-def* **by** *simp*
from *power-strict-mono*[*OF this*, *of 2*] **have** *norm-y'*: *norm y*^*2* < *1* **by** *simp*

let *?f* = λ*k*. *2* ∗ *y* ^ (*2*∗*k*+*1*) / *of-nat* (*2*∗*k*+*1*)
note *sums* = *ln-series-quadratic*[*OF x*(*1*)]
def *c* ≡ *inverse* (*2*∗*y*^(*2*∗*n*+*1*))
let *?d* = *c* ∗ (*ln x* − (∑ *k*<*n*. *?f k*))
have ∀ *k*. *y*^*2* ^*k* / *of-nat* (*2*∗(*k*+*n*)+*1*) ≤ *y*^*2* ^ *k* / *of-nat* (*2*∗*n*+*1*)
by (*intro allI divide-left-mono mult-right-mono mult-pos-pos zero-le-power*[*of y*^*2*]) *simp-all*
moreover {
have (λ*k*. *?f* (*k* + *n*)) *sums* (*ln x* − (∑ *k*<*n*. *?f k*))
using *sums-split-initial-segment*[*OF sums*] **by** (*simp add*: *y-def*)
hence (λ*k*. *c* ∗ *?f* (*k* + *n*)) *sums* *?d* **by** (*rule sums-mult*)
also have (λ*k*. *c* ∗ (*2*∗*y*^(*2*∗(*k*+*n*)+*1*) / *of-nat* (*2*∗(*k*+*n*)+*1*))) =
(λ*k*. (*c* ∗ (*2*∗*y*^(*2*∗*n*+*1*))) ∗ ((*y*^*2*) ^*k* / *of-nat* (*2*∗(*k*+*n*)+*1*)))
by (*simp only*: *ring-distribs power-add power-mult*) (*simp add*: *mult-ac*)
also from *x* **have** *c* ∗ (*2*∗*y*^(*2*∗*n*+*1*)) = *1* **by** (*simp add*: *c-def y-def*)
finally have (λ*k*. (*y*^*2*) ^*k* / *of-nat* (*2*∗(*k*+*n*)+*1*)) *sums* *?d* **by** *simp*
} **note** *sums'* = *this*
moreover from *norm-y'* **have** (λ*k*. (*y*^*2*) ^*k* / *of-nat* (*2*∗*n*+*1*)) *sums* (*1* / (*1* − *y*^*2*) / *of-nat* (*2*∗*n*+*1*))
by (*intro sums-divide geometric-sums*) (*simp-all add*: *norm-power*)
ultimately have *?d* ≤ (*1* / (*1* − *y*^*2*) / *of-nat* (*2*∗*n*+*1*)) **by** (*rule sums-le*)
moreover have *c* ∗ (*ln x* − (∑ *k*<*n*. *2* ∗ *y* ^ (*2* ∗ *k* + *1*) / *real-of-nat* (*2* ∗ *k* + *1*))) ≥ *0*
by (*intro sums-le*[*OF - sums-zero sums'*]) *simp-all*
ultimately show *?thesis* **unfolding** *c-def* **by** *simp*
qed

lemma
fixes *n* :: *nat* **and** *x* :: *real*
defines *y* ≡ (*x*−*1*)/(*x*+*1*)

defines *approx* ≡ (∑ *k*<*n*. *2***y*^(*2***k*+*1*) / *of-nat* (*2***k*+*1*))
defines *d* ≡ *y*^(*2***n*+*1*) / (*1* − *y*^*2*) / *of-nat* (*2***n*+*1*)
assumes *x*: *x* > *1*
shows *ln-approx-bounds*: *ln x* ∈ {*approx*..*approx* + *2***d*}
and *ln-approx-abs*: *abs* (*ln x* − (*approx* + *d*)) ≤ *d*
proof −
def *c* ≡ *2***y*^(*2***n*+*1*)
from *x* **have** *c-pos*: *c* > *0* **unfolding** *c-def y-def*
by (*intro mult-pos-pos zero-less-power*) *simp-all*
have *A*: *inverse c* * (*ln x* − (∑ *k*<*n*. *2***y*^(*2***k*+*1*) / *of-nat* (*2***k*+*1*))) ∈
{*0*.. (*1* / (*1* − *y*^*2*) / *of-nat* (*2***n*+*1*))} **using** *assms* **unfolding** *y-def c-def*
by (*intro ln-approx-aux*) *simp-all*
hence *inverse c* * (*ln x* − (∑ *k*<*n*. *2***y*^(*2***k*+*1*)/*of-nat* (*2***k*+*1*))) ≤ (*1* / (*1*−*y*^*2*) / *of-nat* (*2***n*+*1*))
by *simp*
hence (*ln x* − (∑ *k*<*n*. *2***y*^(*2***k*+*1*) / *of-nat* (*2***k*+*1*))) / *c* ≤ (*1* / (*1* − *y*^*2*) / *of-nat* (*2***n*+*1*))
by (*auto simp add*: *divide-simps*)
with *c-pos* **have** *ln x* ≤ *c* / (*1* − *y*^*2*) / *of-nat* (*2***n*+*1*) + *approx*
by (*subst* (*asm*) *pos-divide-le-eq*) (*simp-all add*: *mult-ac approx-def*)
moreover {
from *A c-pos* **have** *0* ≤ *c* * (*inverse c* * (*ln x* − (∑ *k*<*n*. *2***y*^(*2***k*+*1*) / *of-nat* (*2***k*+*1*))))
by (*intro mult-nonneg-nonneg*[*of c*]) *simp-all*
also have ... = (*c* * *inverse c*) * (*ln x* − (∑ *k*<*n*. *2***y*^(*2***k*+*1*) / *of-nat* (*2***k*+*1*)))
by (*simp add*: *mult-ac*)
also from *c-pos* **have** *c* * *inverse c* = *1* **by** *simp*
finally have *ln x* ≥ *approx* **by** (*simp add*: *approx-def*)
}
ultimately show *ln x* ∈ {*approx*..*approx* + *2***d*} **by** (*simp add*: *c-def d-def*)
thus *abs* (*ln x* − (*approx* + *d*)) ≤ *d* **by** *auto*
qed

end

lemma *euler-mascheroni-bounds*:
fixes *n* :: *nat* **assumes** *n* ≥ *1* **defines** *t* ≡ *harm n* − *ln* (*of-nat* (*Suc n*)) :: *real*
shows *euler-mascheroni* ∈ {*t* + *inverse* (*of-nat* (*2**(*n*+*1*)))..*t* + *inverse* (*of-nat* (*2***n*))}
using *assms euler-mascheroni-upper*[*of n*−*1*] *euler-mascheroni-lower*[*of n*−*1*]
unfolding *t-def* **by** (*cases n*) (*simp-all add*: *harm-Suc t-def inverse-eq-divide*)

lemma *euler-mascheroni-bounds'*:
fixes *n* :: *nat* **assumes** *n* ≥ *1 ln* (*real-of-nat* (*Suc n*)) ∈ {*l*<..<*u*}
shows *euler-mascheroni* ∈
{*harm n* − *u* + *inverse* (*of-nat* (*2**(*n*+*1*)))<..<*harm n* − *l* + *inverse* (*of-nat* (*2***n*))}

using *euler-mascheroni-bounds*[*OF assms*(*1*)] *assms*(*2*) **by** *auto*

Approximation of *ln* (*2*::′*a*). The lower bound is accurate to about 0.03; the upper bound is accurate to about 0.0015.

lemma *ln2-ge-two-thirds*: *2*/*3* ≤ *ln* (*2*::*real*)
and *ln2-le-25-over-36*: *ln* (*2*::*real*) ≤ *25*/*36*
using *ln-approx-bounds*[*of 2 1*, *simplified*, *simplified eval-nat-numeral*, *simplified*]
by *simp-all*

Approximation of the Euler–Mascheroni constant. The lower bound is accurate to about 0.0015; the upper bound is accurate to about 0.015.

lemma *euler-mascheroni-gt-19-over-33*: (*euler-mascheroni* :: *real*) > *19*/*33* (**is** *?th1*)
and *euler-mascheroni-less-13-over-22*: (*euler-mascheroni* :: *real*) < *13*/*22* (**is** *?th2*)
proof −
have *ln* (*real* (*Suc 7*)) = *3* ∗ *ln 2* **by** (*simp add*: *ln-powr* [*symmetric*] *powr-numeral*)
also from *ln-approx-bounds*[*of 2 3*] **have** ... ∈ {*3*∗*307*/*443*<..<*3*∗*4615*/*6658*}
by (*simp add*: *eval-nat-numeral*)
finally have *ln* (*real* (*Suc 7*)) ∈ ... .
from *euler-mascheroni-bounds*′[*OF - this*] **have** *?th1* ∧ *?th2* **by** (*simp-all add*: *harm-expand*)
thus *?th1 ?th2* **by** *blast*+
qed

end

56 Periodic Functions

theory *Periodic-Fun*
imports *Complex-Main*
begin

A locale for periodic functions. The idea is that one proves $f(x + p) = f(x)$ for some period p and gets derived results like $f(x - p) = f(x)$ and $f(x + 2p) = f(x)$ for free.

g and *gm* are "plus/minus k periods" functions. *g1* and *gn1* are "plus/minus one period" functions. This is useful e.g. if the period is one; the lemmas one gets are then *f* (*x* + (*1*::′*b*)) = *f x* instead of *f* (*x* + (*1*::′*b*) ∗ (*1*::′*b*)) = *f x* etc.

locale *periodic-fun* =
fixes *f* :: (′*a* :: {*ring-1*}) ⇒ ′*b* **and** *g gm* :: ′*a* ⇒ ′*a* ⇒ ′*a* **and** *g1 gn1* :: ′*a* ⇒ ′*a*
assumes *plus-1*: *f* (*g1 x*) = *f x*
assumes *periodic-arg-plus-0*: *g x 0* = *x*
assumes *periodic-arg-plus-distrib*: *g x* (*of-int* (*m* + *n*)) = *g* (*g x* (*of-int n*)) (*of-int m*)
assumes *plus-1-eq*: *g x 1* = *g1 x* **and** *minus-1-eq*: *g x* (−*1*) = *gn1 x*

and *minus-eq*: $g\ x\ (-y) = gm\ x\ y$
begin

lemma *plus-of-nat*: $f\ (g\ x\ (of\text{-}nat\ n)) = f\ x$
by (*induction n*) (*insert periodic-arg-plus-distrib*[*of - 1 int n* **for** *n*],
simp-all add: *plus-1 periodic-arg-plus-0 plus-1-eq*)

lemma *minus-of-nat*: $f\ (gm\ x\ (of\text{-}nat\ n)) = f\ x$
proof −
have $f\ (g\ x\ (-\ of\text{-}nat\ n)) = f\ (g\ (g\ x\ (-\ of\text{-}nat\ n))\ (of\text{-}nat\ n))$
by (*rule plus-of-nat*[*symmetric*])
also have $\ldots = f\ (g\ (g\ x\ (of\text{-}int\ (-\ of\text{-}nat\ n)))\ (of\text{-}int\ (of\text{-}nat\ n)))$ **by** *simp*
also have $\ldots = f\ x$
by (*subst periodic-arg-plus-distrib* [*symmetric*]) (*simp add*: *periodic-arg-plus-0*)
finally show *?thesis* **by** (*simp add*: *minus-eq*)
qed

lemma *plus-of-int*: $f\ (g\ x\ (of\text{-}int\ n)) = f\ x$
by (*induction n*) (*simp-all add*: *plus-of-nat minus-of-nat minus-eq del*: *of-nat-Suc*)

lemma *minus-of-int*: $f\ (gm\ x\ (of\text{-}int\ n)) = f\ x$
using *plus-of-int*[*of x of-int* $(-n)$] **by** (*simp add*: *minus-eq*)

lemma *plus-numeral*: $f\ (g\ x\ (numeral\ n)) = f\ x$
by (*subst of-nat-numeral*[*symmetric*], *subst plus-of-nat*) (*rule refl*)

lemma *minus-numeral*: $f\ (gm\ x\ (numeral\ n)) = f\ x$
by (*subst of-nat-numeral*[*symmetric*], *subst minus-of-nat*) (*rule refl*)

lemma *minus-1*: $f\ (gn1\ x) = f\ x$
using *minus-of-nat*[*of x 1*] **by** (*simp add*: *minus-1-eq minus-eq*[*symmetric*])

lemmas *periodic-simps* = *plus-of-nat minus-of-nat plus-of-int minus-of-int*
plus-numeral minus-numeral plus-1 minus-1

end

Specialised case of the *periodic-fun* locale for periods that are not 1. Gives lemmas $f\ (x - period) = f\ x$ etc.

locale *periodic-fun-simple* =
fixes $f :: ('a :: \{ring\text{-}1\}) \Rightarrow 'b$ **and** $period :: 'a$
assumes *plus-period*: $f\ (x + period) = f\ x$
begin
sublocale *periodic-fun f* $\lambda z\ x.\ z + x * period$ $\lambda z\ x.\ z - x * period$
$\lambda z.\ z + period$ $\lambda z.\ z - period$
by *standard* (*simp-all add*: *ring-distribs plus-period*)
end

Specialised case of the *periodic-fun* locale for period 1. Gives lemmas $f\ (x$

$-\ (1::'b)) = f\ x$ etc.

locale *periodic-fun-simple'* =
  fixes $f :: ('a :: \{ring\text{-}1\}) \Rightarrow 'b$
  assumes *plus-period*: $f\ (x + 1) = f\ x$
begin
sublocale *periodic-fun* $f\ \lambda z\ x.\ z + x\ \lambda z\ x.\ z - x\ \lambda z.\ z + 1\ \lambda z.\ z - 1$
  by *standard* (*simp-all add*: *ring-distribs plus-period*)

lemma *of-nat*: $f\ (of\text{-}nat\ n) = f\ 0$ **using** *plus-of-nat*[*of 0 n*] **by** *simp*
lemma *uminus-of-nat*: $f\ (-of\text{-}nat\ n) = f\ 0$ **using** *minus-of-nat*[*of 0 n*] **by** *simp*
lemma *of-int*: $f\ (of\text{-}int\ n) = f\ 0$ **using** *plus-of-int*[*of 0 n*] **by** *simp*
lemma *uminus-of-int*: $f\ (-of\text{-}int\ n) = f\ 0$ **using** *minus-of-int*[*of 0 n*] **by** *simp*
lemma *of-numeral*: $f\ (numeral\ n) = f\ 0$ **using** *plus-numeral*[*of 0 n*] **by** *simp*
lemma *of-neg-numeral*: $f\ (-numeral\ n) = f\ 0$ **using** *minus-numeral*[*of 0 n*] **by** *simp*
lemma *of-1*: $f\ 1 = f\ 0$ **using** *plus-of-nat*[*of 0 1*] **by** *simp*
lemma *of-neg-1*: $f\ (-1) = f\ 0$ **using** *minus-of-nat*[*of 0 1*] **by** *simp*

lemmas *periodic-simps'* =
  of-nat uminus-of-nat of-int uminus-of-int of-numeral of-neg-numeral of-1 of-neg-1

end

lemma *sin-plus-pi*: $sin\ ((z :: 'a :: \{real\text{-}normed\text{-}field,banach\}) + of\text{-}real\ pi) = -\ sin\ z$
  by (*simp add*: *sin-add*)

lemma *cos-plus-pi*: $cos\ ((z :: 'a :: \{real\text{-}normed\text{-}field,banach\}) + of\text{-}real\ pi) = -\ cos\ z$
  by (*simp add*: *cos-add*)

interpretation *sin*: *periodic-fun-simple sin* $2 * of\text{-}real\ pi :: 'a :: \{real\text{-}normed\text{-}field,banach\}$
proof
  fix $z :: 'a$
  have $sin\ (z + 2 * of\text{-}real\ pi) = sin\ (z + of\text{-}real\ pi + of\text{-}real\ pi)$ **by** (*simp add*: *ac-simps*)
  also have $\ldots = sin\ z$ **by** (*simp only*: *sin-plus-pi*) *simp*
  finally show $sin\ (z + 2 * of\text{-}real\ pi) = sin\ z$ **.**
qed

interpretation *cos*: *periodic-fun-simple cos* $2 * of\text{-}real\ pi :: 'a :: \{real\text{-}normed\text{-}field,banach\}$
proof
  fix $z :: 'a$
  have $cos\ (z + 2 * of\text{-}real\ pi) = cos\ (z + of\text{-}real\ pi + of\text{-}real\ pi)$ **by** (*simp add*: *ac-simps*)
  also have $\ldots = cos\ z$ **by** (*simp only*: *cos-plus-pi*) *simp*
  finally show $cos\ (z + 2 * of\text{-}real\ pi) = cos\ z$ **.**
qed

interpretation *tan*: *periodic-fun-simple tan 2 * of-real pi :: 'a :: {real-normed-field,banach}*
by *standard* (*simp only*: *tan-def* [*abs-def*] *sin.plus-1 cos.plus-1*)

interpretation *cot*: *periodic-fun-simple cot 2 * of-real pi :: 'a :: {real-normed-field,banach}*
by *standard* (*simp only*: *cot-def* [*abs-def*] *sin.plus-1 cos.plus-1*)

end

57 The Gamma Function

theory *Gamma*
imports
Complex-Transcendental
Summation
Harmonic-Numbers
~~/*src*/*HOL*/*Library*/*Nonpos-Ints*
~~/*src*/*HOL*/*Library*/*Periodic-Fun*
begin

Several equivalent definitions of the Gamma function and its most important properties. Also contains the definition and some properties of the log-Gamma function and the Digamma function and the other Polygamma functions.

Based on the Gamma function, we also prove the Weierstra product form of the sin function and, based on this, the solution of the Basel problem (the sum over all *1 / real* (n^2).

lemma *pochhammer-eq-0-imp-nonpos-Int*:
pochhammer (*x*::'*a*::*field-char-0*) *n* = *0* $\Longrightarrow$ *x* $\in \mathbb{Z}_{\leq 0}$
by (*auto simp*: *pochhammer-eq-0-iff*)

lemma *closed-nonpos-Ints* [*simp*]: *closed* ($\mathbb{Z}_{\leq 0}$ :: *'a* :: *real-normed-algebra-1 set*)
proof −
have $\mathbb{Z}_{\leq 0}$ = (*of-int* ' {*n*. *n* ≤ *0*} :: *'a set*)
by (*auto elim*!: *nonpos-Ints-cases intro*!: *nonpos-Ints-of-int*)
also have *closed* ... **by** (*rule closed-of-int-image*)
finally show *?thesis* **.**
qed

lemma *plus-one-in-nonpos-Ints-imp*: *z* + *1* $\in \mathbb{Z}_{\leq 0} \Longrightarrow$ *z* $\in \mathbb{Z}_{\leq 0}$
using *nonpos-Ints-diff-Nats*[*of z+1 1*] **by** *simp-all*

lemma *fraction-not-in-ints*:
assumes ¬(*n dvd m*) *n* ≠ *0*
shows *of-int m / of-int n* ∉ ($\mathbb{Z}$ :: *'a* :: {*division-ring*,*ring-char-0*} *set*)
proof
assume *of-int m / (of-int n* :: *'a*) ∈ $\mathbb{Z}$
then obtain *k* **where** *of-int m / of-int n* = (*of-int k* :: *'a*) **by** (*elim Ints-cases*)
with *assms* **have** *of-int m* = (*of-int* (*k * n*) :: *'a*) **by** (*auto simp add*: *divide-simps*)

hence $m = k * n$ **by** (*subst* (*asm*) *of-int-eq-iff*)
hence *n dvd m* **by** *simp*
with *assms*(*1*) **show** *False* **by** *contradiction*
qed

lemma *not-in-Ints-imp-not-in-nonpos-Ints*: $z \notin \mathbb{Z} \Longrightarrow z \notin \mathbb{Z}_{\leq 0}$
by (*auto simp*: *Ints-def nonpos-Ints-def*)

lemma *double-in-nonpos-Ints-imp*:
assumes $2 * (z :: 'a :: \textit{field-char-0}) \in \mathbb{Z}_{\leq 0}$
shows $z \in \mathbb{Z}_{\leq 0} \vee z + 1/2 \in \mathbb{Z}_{\leq 0}$
proof−
from *assms* **obtain** *k* **where** *k*: $2 * z = -$ *of-nat k* **by** (*elim nonpos-Ints-cases'*)
thus *?thesis* **by** (*cases even k*) (*auto elim*!: *evenE oddE simp*: *field-simps*)
qed

lemma *sin-series*: $(\lambda n.\ ((-1)\hat{}n\ /\ \textit{fact}\ (2*n+1)) *_R z\hat{}(2*n+1))$ *sums sin z*
proof −
from *sin-converges*[*of z*] **have** $(\lambda n.\ \textit{sin-coeff}\ n *_R z\hat{}n)$ *sums sin z* **.**
also have $(\lambda n.\ \textit{sin-coeff}\ n *_R z\hat{}n)$ *sums sin z* $\longleftrightarrow$
$(\lambda n.\ ((-1)\hat{}n\ /\ \textit{fact}\ (2*n+1)) *_R z\hat{}(2*n+1))$ *sums sin z*
by (*subst sums-mono-reindex*[*of* $\lambda n.\ 2*n+1$, *symmetric*])
(*auto simp*: *sin-coeff-def subseq-def ac-simps elim*!: *oddE*)
finally show *?thesis* **.**
qed

lemma *cos-series*: $(\lambda n.\ ((-1)\hat{}n\ /\ \textit{fact}\ (2*n)) *_R z\hat{}(2*n))$ *sums cos z*
proof −
from *cos-converges*[*of z*] **have** $(\lambda n.\ \textit{cos-coeff}\ n *_R z\hat{}n)$ *sums cos z* **.**
also have $(\lambda n.\ \textit{cos-coeff}\ n *_R z\hat{}n)$ *sums cos z* $\longleftrightarrow$
$(\lambda n.\ ((-1)\hat{}n\ /\ \textit{fact}\ (2*n)) *_R z\hat{}(2*n))$ *sums cos z*
by (*subst sums-mono-reindex*[*of* $\lambda n.\ 2*n$, *symmetric*])
(*auto simp*: *cos-coeff-def subseq-def ac-simps elim*!: *evenE*)
finally show *?thesis* **.**
qed

lemma *sin-z-over-z-series*:
fixes $z :: 'a :: \{\textit{real-normed-field}, \textit{banach}\}$
assumes $z \neq 0$
shows $(\lambda n.\ (-1)\hat{}n\ /\ \textit{fact}\ (2*n+1) * z\hat{}(2*n))$ *sums* $(\textit{sin}\ z\ /\ z)$
proof −
from *sin-series*[*of z*] **have** $(\lambda n.\ z * ((-1)\hat{}n\ /\ \textit{fact}\ (2*n+1)) * z\hat{}(2*n))$ *sums*
sin z
by (*simp add*: *field-simps scaleR-conv-of-real*)
from *sums-mult*[*OF this*, *of inverse z*] **and** *assms* **show** *?thesis*
by (*simp add*: *field-simps*)
qed

lemma *sin-z-over-z-series'*:
fixes $z :: 'a :: \{real\text{-}normed\text{-}field, banach\}$
assumes $z \neq 0$
shows $(\lambda n.\ sin\text{-}coeff\ (n+1) *_R z\hat{}n)\ sums\ (sin\ z\ /\ z)$
proof −
from *sums-split-initial-segment*[*OF sin-converges*[*of z*], *of 1*]
have $(\lambda n.\ z * (sin\text{-}coeff\ (n+1) *_R z\ \hat{}\ n))\ sums\ sin\ z$ **by** *simp*
from *sums-mult*[*OF this*, *of inverse z*] *assms* **show** *?thesis* **by** (*simp add*: *field-simps*)
qed

lemma *has-field-derivative-sin-z-over-z*:
fixes $A :: 'a :: \{real\text{-}normed\text{-}field, banach\}\ set$
shows $((\lambda z.\ if\ z = 0\ then\ 1\ else\ sin\ z\ /\ z)\ has\text{-}field\text{-}derivative\ 0)\ (at\ 0\ within\ A)$
(**is** (*?f has-field-derivative ?f'*) -)
proof (*rule has-field-derivative-at-within*)
have $((\lambda z::'a.\ \sum n.\ of\text{-}real\ (sin\text{-}coeff\ (n+1)) * z\hat{}n)$
has-field-derivative $(\sum n.\ diffs\ (\lambda n.\ of\text{-}real\ (sin\text{-}coeff\ (n+1)))\ n * 0\hat{}n))$ $(at\ 0)$
proof (*rule termdiffs-strong*)
from *summable-ignore-initial-segment*[*OF sums-summable*[*OF sin-converges*[*of* $1::'a$]], *of 1*]
show $summable\ (\lambda n.\ of\text{-}real\ (sin\text{-}coeff\ (n+1)) * (1::'a)\hat{}n)$ **by** (*simp add*: *of-real-def*)
qed *simp*
also have $(\lambda z::'a.\ \sum n.\ of\text{-}real\ (sin\text{-}coeff\ (n+1)) * z\hat{}n) = ?f$
proof
fix *z*
show $(\sum n.\ of\text{-}real\ (sin\text{-}coeff\ (n+1)) * z\hat{}n)\ = ?f\ z$
by (*cases z = 0*) (*insert sin-z-over-z-series'*[*of z*],
simp-all add: *scaleR-conv-of-real sums-iff powser-zero sin-coeff-def*)
qed
also have $(\sum n.\ diffs\ (\lambda n.\ of\text{-}real\ (sin\text{-}coeff\ (n\ +\ 1)))\ n * (0::'a)\ \hat{}\ n) =$
$diffs\ (\lambda n.\ of\text{-}real\ (sin\text{-}coeff\ (Suc\ n)))\ 0$ **by** (*simp add*: *powser-zero*)
also have $\ldots = 0$ **by** (*simp add*: *sin-coeff-def diffs-def*)
finally show $((\lambda z::'a.\ if\ z = 0\ then\ 1\ else\ sin\ z\ /\ z)\ has\text{-}field\text{-}derivative\ 0)\ (at\ 0)$.
qed

lemma *round-Re-minimises-norm*:
$norm\ ((z::complex) - of\text{-}int\ m) \geq norm\ (z - of\text{-}int\ (round\ (Re\ z)))$
proof −
let *?n* = $round\ (Re\ z)$
have $norm\ (z - of\text{-}int\ ?n) = sqrt\ ((Re\ z - of\text{-}int\ ?n)^2 + (Im\ z)^2)$
by (*simp add*: *cmod-def*)
also have $|Re\ z - of\text{-}int\ ?n| \leq |Re\ z - of\text{-}int\ m|$ **by** (*rule round-diff-minimal*)
hence $sqrt\ ((Re\ z - of\text{-}int\ ?n)^2 + (Im\ z)^2) \leq sqrt\ ((Re\ z - of\text{-}int\ m)^2 + (Im\ z)^2)$
by (*intro real-sqrt-le-mono add-mono*) (*simp-all add*: *abs-le-square-iff*)

also have ... = *norm* (*z* − *of-int m*) **by** (*simp add*: *cmod-def*)
finally show *?thesis* **.**
qed

lemma *Re-pos-in-ball*:
assumes *Re z* > *0 t* ∈ *ball z* (*Re z*/*2*)
shows *Re t* > *0*
proof −
have *Re* (*z* − *t*) ≤ *norm* (*z* − *t*) **by** (*rule complex-Re-le-cmod*)
also from *assms* **have** ... < *Re z* / *2* **by** (*simp add*: *dist-complex-def*)
finally show *Re t* > *0* **using** *assms* **by** *simp*
qed

lemma *no-nonpos-Int-in-ball-complex*:
assumes *Re z* > *0 t* ∈ *ball z* (*Re z*/*2*)
shows *t* ∉ $\mathbb{Z}_{\leq 0}$
using *Re-pos-in-ball*[*OF assms*] **by** (*force elim*!: *nonpos-Ints-cases*)

lemma *no-nonpos-Int-in-ball*:
assumes *t* ∈ *ball z* (*dist z* (*round* (*Re z*)))
shows *t* ∉ $\mathbb{Z}_{\leq 0}$
proof
assume *t* ∈ $\mathbb{Z}_{\leq 0}$
then obtain *n* **where** *t* = *of-int n* **by** (*auto elim*!: *nonpos-Ints-cases*)
have *dist z* (*of-int n*) ≤ *dist z t* + *dist t* (*of-int n*) **by** (*rule dist-triangle*)
also from *assms* **have** *dist z t* < *dist z* (*round* (*Re z*)) **by** *simp*
also have ... ≤ *dist z* (*of-int n*)
using *round-Re-minimises-norm*[*of z*] **by** (*simp add*: *dist-complex-def*)
finally have *dist t* (*of-int n*) > *0* **by** *simp*
with ‹*t* = *of-int n*› **show** *False* **by** *simp*
qed

lemma *no-nonpos-Int-in-ball'*:
assumes (*z* :: *'a* :: {*euclidean-space*,*real-normed-algebra-1*}) ∉ $\mathbb{Z}_{\leq 0}$
obtains *d* **where** *d* > *0* ⋀*t*. *t* ∈ *ball z d* ⟹ *t* ∉ $\mathbb{Z}_{\leq 0}$
proof (*rule that*)
from *assms* **show** *setdist* {*z*} $\mathbb{Z}_{\leq 0}$ > *0* **by** (*subst setdist-gt-0-compact-closed*) *auto*
next
fix *t* **assume** *t* ∈ *ball z* (*setdist* {*z*} $\mathbb{Z}_{\leq 0}$)
thus *t* ∉ $\mathbb{Z}_{\leq 0}$ **using** *setdist-le-dist*[*of z* {*z*} *t* $\mathbb{Z}_{\leq 0}$] **by** *force*
qed

lemma *no-nonpos-Real-in-ball*:
assumes *z*: *z* ∉ $\mathbb{R}_{\leq 0}$ **and** *t*: *t* ∈ *ball z* (*if Im z* = *0 then Re z* / *2 else abs* (*Im z*) / *2*)
shows *t* ∉ $\mathbb{R}_{\leq 0}$
using *z*
proof (*cases Im z* = *0*)

assume *A*: *Im z = 0*
with *z* **have** *Re z > 0* **by** (*force simp add*: *complex-nonpos-Reals-iff*)
with *t A Re-pos-in-ball*[*of z t*] **show** *?thesis* **by** (*force simp add*: *complex-nonpos-Reals-iff*)
next
assume *A*: *Im z ≠ 0*
have *abs* (*Im z*) − *abs* (*Im t*) ≤ *abs* (*Im z − Im t*) **by** *linarith*
also have ... = *abs* (*Im* (*z − t*)) **by** *simp*
also have ... ≤ *norm* (*z − t*) **by** (*rule abs-Im-le-cmod*)
also from *A t* **have** ... ≤ *abs* (*Im z*) / *2* **by** (*simp add*: *dist-complex-def*)
finally have *abs* (*Im t*) > *0* **using** *A* **by** *simp*
thus *?thesis* **by** (*force simp add*: *complex-nonpos-Reals-iff*)
qed

57.1 Definitions

We define the Gamma function by first defining its multiplicative inverse *Gamma-inv*. This is more convenient because *Gamma-inv* is entire, which makes proofs of its properties more convenient because one does not have to watch out for discontinuities. (e.g. *Gamma-inv* fulfils *rGamma z = z ∗ rGamma* (*z* + (*1*::*'a*)) everywhere, whereas *Gamma* does not fulfil the analogous equation on the non-positive integers)

We define the Gamma function (resp. its inverse) in the Euler form. This form has the advantage that it is a relatively simple limit that converges everywhere. The limit at the poles is 0 (due to division by 0). The functional equation *Gamma* (*z* + (*1*::*'a*)) = *z ∗ Gamma z* follows immediately from the definition.

definition *Gamma-series* :: (*'a* :: {*banach*,*real-normed-field*}) ⇒ *nat* ⇒ *'a* **where**
Gamma-series z n = fact n ∗ exp (*z ∗ of-real* (*ln* (*of-nat n*))) / *pochhammer z* (*n+1*)

definition *Gamma-series'* :: (*'a* :: {*banach*,*real-normed-field*}) ⇒ *nat* ⇒ *'a* **where**
Gamma-series' z n = fact (*n − 1*) ∗ *exp* (*z ∗ of-real* (*ln* (*of-nat n*))) / *pochhammer z n*

definition *rGamma-series* :: (*'a* :: {*banach*,*real-normed-field*}) ⇒ *nat* ⇒ *'a* **where**
rGamma-series z n = pochhammer z (*n+1*) / (*fact n ∗ exp* (*z ∗ of-real* (*ln* (*of-nat n*))))

lemma *Gamma-series-altdef*: *Gamma-series z n = inverse* (*rGamma-series z n*)
and *rGamma-series-altdef*: *rGamma-series z n = inverse* (*Gamma-series z n*)
unfolding *Gamma-series-def rGamma-series-def* **by** *simp-all*

lemma *rGamma-series-minus-of-nat*:
eventually (λ*n*. *rGamma-series* (− *of-nat k*) *n = 0*) *sequentially*
using *eventually-ge-at-top*[*of k*]
by *eventually-elim* (*auto simp*: *rGamma-series-def pochhammer-of-nat-eq-0-iff*)

lemma *Gamma-series-minus-of-nat*:
eventually (λ*n*. *Gamma-series* (− *of-nat k*) *n* = *0*) *sequentially*
using *eventually-ge-at-top*[*of k*]
by *eventually-elim* (*auto simp*: *Gamma-series-def pochhammer-of-nat-eq-0-iff*)

lemma *Gamma-series'-minus-of-nat*:
eventually (λ*n*. *Gamma-series'* (− *of-nat k*) *n* = *0*) *sequentially*
using *eventually-gt-at-top*[*of k*]
by *eventually-elim* (*auto simp*: *Gamma-series'-def pochhammer-of-nat-eq-0-iff*)

lemma *rGamma-series-nonpos-Ints-LIMSEQ*: $z \in \mathbb{Z}_{\leq 0} \Longrightarrow$ *rGamma-series z* ⟶ *0*
by (*elim nonpos-Ints-cases'*, *hypsubst*, *subst tendsto-cong*, *rule rGamma-series-minus-of-nat*, *simp*)

lemma *Gamma-series-nonpos-Ints-LIMSEQ*: $z \in \mathbb{Z}_{\leq 0} \Longrightarrow$ *Gamma-series z* ⟶ *0*
by (*elim nonpos-Ints-cases'*, *hypsubst*, *subst tendsto-cong*, *rule Gamma-series-minus-of-nat*, *simp*)

lemma *Gamma-series'-nonpos-Ints-LIMSEQ*: $z \in \mathbb{Z}_{\leq 0} \Longrightarrow$ *Gamma-series' z* ⟶ *0*
by (*elim nonpos-Ints-cases'*, *hypsubst*, *subst tendsto-cong*, *rule Gamma-series'-minus-of-nat*, *simp*)

lemma *Gamma-series-Gamma-series'*:
assumes *z*: $z \notin \mathbb{Z}_{\leq 0}$
shows (λ*n*. *Gamma-series' z n* / *Gamma-series z n*) ⟶ *1*
proof (*rule Lim-transform-eventually*)
from *eventually-gt-at-top*[*of 0*::*nat*]
show *eventually* (λ*n*. *z* / *of-nat n* + *1* = *Gamma-series' z n* / *Gamma-series z n*) *sequentially*
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
from *n z* **have** *Gamma-series' z n* / *Gamma-series z n* = (*z* + *of-nat n*) / *of-nat n*
by (*cases n*, *simp*)
(*auto simp add*: *Gamma-series-def Gamma-series'-def pochhammer-rec'*
dest: *pochhammer-eq-0-imp-nonpos-Int plus-of-nat-eq-0-imp*)
also from *n* **have** ... = *z* / *of-nat n* + *1* **by** (*simp add*: *divide-simps*)
finally show *z* / *of-nat n* + *1* = *Gamma-series' z n* / *Gamma-series z n* **..**
qed
have (λ*x*. *z* / *of-nat x*) ⟶ *0*
by (*rule tendsto-norm-zero-cancel*)
(*insert tendsto-mult*[*OF tendsto-const*[*of norm z*] *lim-inverse-n*],
simp add: *norm-divide inverse-eq-divide*)
from *tendsto-add*[*OF this tendsto-const*[*of 1*]] **show** (λ*n*. *z* / *of-nat n* + *1*) ⟶ *1* **by** *simp*
qed

57.2 Convergence of the Euler series form

We now show that the series that defines the Gamma function in the Euler form converges and that the function defined by it is continuous on the complex halfspace with positive real part.

We do this by showing that the logarithm of the Euler series is continuous and converges locally uniformly, which means that the log-Gamma function defined by its limit is also continuous.

This will later allow us to lift holomorphicity and continuity from the log-Gamma function to the inverse of the Gamma function, and from that to the Gamma function itself.

definition *ln-Gamma-series* :: ('*a* :: {*banach*,*real-normed-field*,*ln*}) ⇒ *nat* ⇒ '*a*
where
ln-Gamma-series z n = *z* ∗ *ln* (*of-nat n*) − *ln z* − (∑ *k*=*1*..*n*. *ln* (*z* / *of-nat k* + *1*))

definition *ln-Gamma-series'* :: ('*a* :: {*banach*,*real-normed-field*,*ln*}) ⇒ *nat* ⇒ '*a*
where
ln-Gamma-series' z n =
− *euler-mascheroni*∗*z* − *ln z* + (∑ *k*=*1*..*n*. *z* / *of-nat n* − *ln* (*z* / *of-nat k* + *1*))

definition *ln-Gamma* :: ('*a* :: {*banach*,*real-normed-field*,*ln*}) ⇒ '*a* **where**
ln-Gamma z = *lim* (*ln-Gamma-series z*)

We now show that the log-Gamma series converges locally uniformly for all complex numbers except the non-positive integers. We do this by proving that the series is locally Cauchy, adapting this proof: http://math.stackexchange.com/questions/88715 of-gammaz-lim-n-to-infty-fracnzn-prod-m-0nzm

context
begin

private lemma *ln-Gamma-series-complex-converges-aux*:
fixes *z* :: *complex* **and** *k* :: *nat*
assumes *z*: *z* ≠ *0* **and** *k*: *of-nat k* ≥ *2*∗*norm z k* ≥ *2*
shows *norm* (*z* ∗ *ln* (*1* − *1*/*of-nat k*) + *ln* (*z*/*of-nat k* + *1*)) ≤ *2*∗(*norm z* + *norm z*^*2*) / *of-nat k*^*2*
proof −
let *?k* = *of-nat k* :: *complex* **and** *?z* = *norm z*
have *z* ∗*ln* (*1* − *1*/*?k*) + *ln* (*z*/*?k*+*1*) = *z*∗(*ln* (*1* − *1*/*?k* :: *complex*) + *1*/*?k*) + (*ln* (*1*+*z*/*?k*) − *z*/*?k*)
by (*simp add*: *algebra-simps*)
also have *norm* ... ≤ *?z* ∗ *norm* (*ln* (*1*−*1*/*?k*) + *1*/*?k* :: *complex*) + *norm* (*ln* (*1*+*z*/*?k*) − *z*/*?k*)
by (*subst norm-mult* [*symmetric*], *rule norm-triangle-ineq*)
also have *norm* (*Ln* (*1* + −*1*/*?k*) − (−*1*/*?k*)) ≤ (*norm* (−*1*/*?k*))^2^ / (*1* − *norm*(−*1*/*?k*))

using k **by** (*intro Ln-approx-linear*) (*simp add*: *norm-divide*)
hence *?z* $*$ *norm* (*ln* (*1*−*1* / *?k*) + *1* / *?k*) ≤ *?z* $*$ ((*norm* (*1* / *?k*)) ^ *2* / (*1* − *norm* (*1* / *?k*)))
by (*intro mult-left-mono*) *simp-all*
also have ... ≤ (*?z* $*$ (*of-nat k* / (*of-nat k* − *1*))) / *of-nat k*^*2* **using** k
by (*simp add*: *field-simps power2-eq-square norm-divide*)
also have ... ≤ (*?z* $*$ *2*) / *of-nat k*^*2* **using** k
by (*intro divide-right-mono mult-left-mono*) (*simp-all add*: *field-simps*)
also have *norm* (*ln* (*1*+*z* / *?k*) − *z* / *?k*) ≤ *norm* (*z* / *?k*) ^ *2* / (*1* − *norm* (*z* / *?k*)) **using** k
by (*intro Ln-approx-linear*) (*simp add*: *norm-divide*)
hence *norm* (*ln* (*1*+*z* / *?k*) − *z* / *?k*) ≤ *?z*^*2* / *of-nat k*^*2* / (*1* − *?z* / *of-nat k*)
by (*simp add*: *field-simps norm-divide*)
also have ... ≤ (*?z*^*2* $*$ (*of-nat k* / (*of-nat k* − *?z*))) / *of-nat k*^*2* **using** k
by (*simp add*: *field-simps power2-eq-square*)
also have ... ≤ (*?z*^*2* $*$ *2*) / *of-nat k*^*2* **using** k
by (*intro divide-right-mono mult-left-mono*) (*simp-all add*: *field-simps*)
also note *add-divide-distrib* [*symmetric*]
finally show *?thesis* **by** (*simp only*: *distrib-left mult.commute*)
qed

lemma *ln-Gamma-series-complex-converges*:
assumes z: $z \notin \mathbb{Z}_{\leq 0}$
assumes d: $d > 0$ $\bigwedge n.\ n \in \mathbb{Z}_{\leq 0} \Longrightarrow$ *norm* (z − *of-int n*) > d
shows *uniformly-convergent-on* (*ball z d*) (λ*n z*. *ln-Gamma-series z n* :: *complex*)
proof (*intro Cauchy-uniformly-convergent uniformly-Cauchy-onI'*)
fix e :: *real* **assume** e: $e > 0$
def e'' ≡ *SUP t*:*ball z d*. *norm t* + *norm t*^*2*
def e' ≡ e / (*2*$*$$e''$)
have *bounded* ((λ*t*. *norm t* + *norm t*^*2*) ‘ *cball z d*)
by (*intro compact-imp-bounded compact-continuous-image*) (*auto intro*!: *continuous-intros*)
hence *bounded* ((λ*t*. *norm t* + *norm t*^*2*) ‘ *ball z d*) **by** (*rule bounded-subset*) *auto*
hence *bdd*: *bdd-above* ((λ*t*. *norm t* + *norm t*^*2*) ‘ *ball z d*) **by** (*rule bounded-imp-bdd-above*)

with *z d*(*1*) *d*(*2*)[*of* −*1*] **have** e''*-pos*: $e'' > 0$ **unfolding** e''*-def*
by (*subst less-cSUP-iff*) (*auto intro*!: *add-pos-nonneg bexI*[*of* - *z*])
have e'': *norm t* + *norm t*^*2* ≤ e'' **if** $t \in$ *ball z d* **for** t **unfolding** e''*-def* **using** *that*
by (*rule cSUP-upper*[*OF* - *bdd*])
from *e z* e''*-pos* **have** e': $e' > 0$ **unfolding** e'*-def*
by (*intro divide-pos-pos mult-pos-pos add-pos-pos*) (*simp-all add*: *field-simps*)

have *summable* (λ*k*. *inverse* ((*real-of-nat k*) ^ *2*))
by (*rule inverse-power-summable*) *simp*
from *summable-partial-sum-bound*[*OF this e'*] **guess** M **. note** M = *this*

def N ≡ *max 2* (*max* (*nat* ⌈*2* $*$ (*norm z* + *d*)⌉) M)
{

from *d* **have** $\lceil 2 * (cmod\ z + d) \rceil \geq \lceil 0::real \rceil$
by (*intro ceiling-mono mult-nonneg-nonneg add-nonneg-nonneg*) *simp-all*
hence $2 * (norm\ z + d) \leq of\text{-}nat\ (nat\ \lceil 2 * (norm\ z + d) \rceil)$ **unfolding** *N-def*
by (*simp-all add*: *le-of-int-ceiling*)
also have $... \leq of\text{-}nat\ N$ **unfolding** *N-def*
by (*subst of-nat-le-iff*) (*rule max.coboundedI2*, *rule max.cobounded1*)
finally have $of\text{-}nat\ N \geq 2 * (norm\ z + d)$ **.**
moreover have $N \geq 2\ N \geq M$ **unfolding** *N-def* **by** *simp-all*
moreover have $(\sum k{=}m..n.\ 1/(of\text{-}nat\ k)^2) < e'$ **if** $m \geq N$ **for** *m n*
using *M*[*OF order.trans*[*OF* ‹$N \geq M$› *that*]] **unfolding** *real-norm-def*
by (*subst* (*asm*) *abs-of-nonneg*) (*auto intro*: *setsum-nonneg simp*: *divide-simps*)
moreover note *calculation*
} note $N = this$

show $\exists M.\ \forall t{\in}ball\ z\ d.\ \forall m{\geq}M.\ \forall n{>}m.\ dist\ (ln\text{-}Gamma\text{-}series\ t\ m)\ (ln\text{-}Gamma\text{-}series\ t\ n) < e$
unfolding *dist-complex-def*
proof (*intro exI*[*of - N*] *ballI allI impI*)
fix *t m n* **assume** *t*: $t \in ball\ z\ d$ **and** *mn*: $m \geq N\ n > m$
from *d*(*2*)[*of 0*] *t* **have** $0 < dist\ z\ 0 - dist\ z\ t$ **by** (*simp add*: *field-simps dist-complex-def*)
also have $dist\ z\ 0 - dist\ z\ t \leq dist\ 0\ t$ **using** *dist-triangle*[*of 0 z t*]
by (*simp add*: *dist-commute*)
finally have *t-nz*: $t \neq 0$ **by** *auto*

have $norm\ t \leq norm\ z + norm\ (t - z)$ **by** (*rule norm-triangle-sub*)
also from *t* **have** $norm\ (t - z) < d$ **by** (*simp add*: *dist-complex-def norm-minus-commute*)
also have $2 * (norm\ z + d) \leq of\text{-}nat\ N$ **by** (*rule N*)
also have $N \leq m$ **by** (*rule mn*)
finally have *norm-t*: $2 * norm\ t < of\text{-}nat\ m$ **by** *simp*

have $ln\text{-}Gamma\text{-}series\ t\ m - ln\text{-}Gamma\text{-}series\ t\ n =$
$(-(t * Ln\ (of\text{-}nat\ n)) - (-(t * Ln\ (of\text{-}nat\ m)))) +$
$((\sum k{=}1..n.\ Ln\ (t\ /\ of\text{-}nat\ k + 1)) - (\sum k{=}1..m.\ Ln\ (t\ /\ of\text{-}nat\ k + 1)))$
by (*simp add*: *ln-Gamma-series-def algebra-simps*)
also have $(\sum k{=}1..n.\ Ln\ (t\ /\ of\text{-}nat\ k + 1)) - (\sum k{=}1..m.\ Ln\ (t\ /\ of\text{-}nat\ k + 1)) =$
$(\sum k{\in}\{1..n\}{-}\{1..m\}.\ Ln\ (t\ /\ of\text{-}nat\ k + 1))$ **using** *mn*
by (*simp add*: *setsum-diff*)
also from *mn* **have** $\{1..n\}{-}\{1..m\} = \{Suc\ m..n\}$ **by** *fastforce*
also have $-(t * Ln\ (of\text{-}nat\ n)) - (-(t * Ln\ (of\text{-}nat\ m))) =$
$(\sum k = Suc\ m..n.\ t * Ln\ (of\text{-}nat\ (k - 1)) - t * Ln\ (of\text{-}nat\ k))$ **using** *mn*
by (*subst setsum-telescope''* [*symmetric*]) *simp-all*
also have $... = (\sum k = Suc\ m..n.\ t * Ln\ (of\text{-}nat\ (k - 1)\ /\ of\text{-}nat\ k))$ **using** *mn N*
by (*intro setsum-cong-Suc*)
(*simp-all del*: *of-nat-Suc add*: *field-simps Ln-of-nat Ln-of-nat-over-of-nat*)

```
    also have of-nat (k − 1) / of-nat k = 1 − 1 / (of-nat k :: complex) if k ∈
{Suc m..n} for k
    using that of-nat-eq-0-iff[of Suc i for i] by (cases k) (simp-all add: divide-simps)
    hence (∑ k = Suc m..n. t * Ln (of-nat (k − 1) / of-nat k)) =
          (∑ k = Suc m..n. t * Ln (1 − 1 / of-nat k)) using mn N
      by (intro setsum.cong) simp-all
    also note setsum.distrib [symmetric]
    also have norm (∑ k=Suc m..n. t * Ln (1 − 1/of-nat k) + Ln (t/of-nat k
+ 1)) ≤
       (∑ k=Suc m..n. 2 * (norm t + (norm t)^2) / (real-of-nat k)^2) using t-nz
N(2) mn norm-t
    by (intro order.trans[OF norm-setsum setsum-mono[OF ln-Gamma-series-complex-converges-aux]])
simp-all
    also have ... ≤ 2 * (norm t + norm t^2) * (∑ k=Suc m..n. 1 / (of-nat k)^2)
      by (simp add: setsum-right-distrib)
    also have ... < 2 * (norm t + norm t^2) * e' using mn z t-nz
      by (intro mult-strict-left-mono N mult-pos-pos add-pos-pos) simp-all
    also from e''-pos have ... = e * ((cmod t + (cmod t)^2) / e'')
      by (simp add: e'-def field-simps power2-eq-square)
    also from e''[OF t] e''-pos e
      have ... ≤ e * 1 by (intro mult-left-mono) (simp-all add: field-simps)
    finally show norm (ln-Gamma-series t m − ln-Gamma-series t n) < e by
simp
  qed
qed

end

lemma ln-Gamma-series-complex-converges':
  assumes z: (z :: complex) ∉ ℤ≤0
  shows ∃ d>0. uniformly-convergent-on (ball z d) (λn z. ln-Gamma-series z n)
proof −
  def d' ≡ Re z
  def d ≡ if d' > 0 then d' / 2 else norm (z − of-int (round d')) / 2
  have of-int (round d') ∈ ℤ≤0 if d' ≤ 0 using that
    by (intro nonpos-Ints-of-int) (simp-all add: round-def)
  with assms have d-pos: d > 0 unfolding d-def by (force simp: not-less)

  have d < cmod (z − of-int n) if n ∈ ℤ≤0 for n
  proof (cases Re z > 0)
    case True
    from nonpos-Ints-nonpos[OF that] have n: n ≤ 0 by simp
    from True have d = Re z/2 by (simp add: d-def d'-def)
    also from n True have ... < Re (z − of-int n) by simp
    also have ... ≤ norm (z − of-int n) by (rule complex-Re-le-cmod)
    finally show ?thesis .
  next
    case False
    with assms nonpos-Ints-of-int[of round (Re z)]
```

have $z \neq$ *of-int* (*round* d') **by** (*auto simp*: *not-less*)
with *False* **have** $d <$ *norm* ($z -$ *of-int* (*round* d')) **by** (*simp add*: *d-def d'-def*)
also have $\ldots \leq$ *norm* ($z -$ *of-int* n) **unfolding** *d'-def* **by** (*rule round-Re-minimises-norm*)
finally show *?thesis* **.**
qed
hence *conv*: *uniformly-convergent-on* (*ball z d*) (λn *z*. *ln-Gamma-series z n*)
by (*intro ln-Gamma-series-complex-converges d-pos z*) *simp-all*
from *d-pos conv* **show** *?thesis* **by** *blast*
qed

lemma *ln-Gamma-series-complex-converges''*: (z :: *complex*) $\notin \mathbb{Z}_{\leq 0} \Longrightarrow$ *convergent* (*ln-Gamma-series z*)
by (*drule ln-Gamma-series-complex-converges'*) (*auto intro*: *uniformly-convergent-imp-convergent*)

lemma *ln-Gamma-complex-LIMSEQ*: (z :: *complex*) $\notin \mathbb{Z}_{\leq 0} \Longrightarrow$ *ln-Gamma-series z* $\longrightarrow$ *ln-Gamma z*
using *ln-Gamma-series-complex-converges''* **by** (*simp add*: *convergent-LIMSEQ-iff ln-Gamma-def*)

lemma *exp-ln-Gamma-series-complex*:
assumes $n > 0$ $z \notin \mathbb{Z}_{\leq 0}$
shows *exp* (*ln-Gamma-series z n* :: *complex*) = *Gamma-series z n*
proof −
from *assms* **have** $z \neq 0$ **by** (*intro notI*) *auto*
with *assms* **have** *exp* (*ln-Gamma-series z n*) =
(*of-nat n*) *powr z* / (*z* ∗ (∏ *k*=*1*..*n*. *exp* (*Ln* (*z* / *of-nat k* + *1*))))
unfolding *ln-Gamma-series-def powr-def* **by** (*simp add*: *exp-diff exp-setsum*)
also from *assms* **have** (∏ *k*=*1*..*n*. *exp* (*Ln* (*z* / *of-nat k* + *1*))) = (∏ *k*=*1*..*n*. *z* / *of-nat k* + *1*)
by (*intro setprod.cong*[*OF refl*], *subst exp-Ln*) (*auto simp*: *field-simps plus-of-nat-eq-0-imp*)
also have ... = (∏ *k*=*1*..*n*. *z* + *k*) / *fact n* **unfolding** *fact-altdef*
by (*subst setprod-dividef* [*symmetric*]) (*simp-all add*: *field-simps*)
also from *assms* **have** *z* ∗ ... = (∏ *k*=*0*..*n*. *z* + *k*) / *fact n*
by (*cases n*) (*simp-all add*: *setprod-nat-ivl-1-Suc*)
also have (∏ *k*=*0*..*n*. *z* + *k*) = *pochhammer z* (*Suc n*) **unfolding** *pochhammer-def*
by *simp*
also have *of-nat n powr z* / (*pochhammer z* (*Suc n*) / *fact n*) = *Gamma-series z n*
unfolding *Gamma-series-def* **using** *assms* **by** (*simp add*: *divide-simps powr-def Ln-of-nat*)
finally show *?thesis* **.**
qed

lemma *ln-Gamma-series'-aux*:
assumes (*z*::*complex*) $\notin \mathbb{Z}_{\leq 0}$
shows (λk. *z* / *of-nat* (*Suc k*) − *ln* (*1* + *z* / *of-nat* (*Suc k*))) *sums*
(*ln-Gamma z* + *euler-mascheroni* ∗ *z* + *ln z*) (**is** *?f sums ?s*)
unfolding *sums-def*

proof (*rule Lim-transform*)
show (λ*n*. *ln-Gamma-series z n* + *of-real* (*harm n* − *ln* (*of-nat n*)) ∗ *z* + *ln z*) ⟶ *?s*
(**is** *?g* ⟶ -)
by (*intro tendsto-intros ln-Gamma-complex-LIMSEQ euler-mascheroni-LIMSEQ-of-real assms*)

have *A*: *eventually* (λ*n*. (∑ *k*<*n*. *?f k*) − *?g n* = *0*) *sequentially*
using *eventually-gt-at-top*[*of 0*::*nat*]
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
have (∑ *k*<*n*. *?f k*) = (∑ *k*=*1*..*n*. *z* / *of-nat k* − *ln* (*1* + *z* / *of-nat k*))
by (*subst atLeast0LessThan* [*symmetric*], *subst setsum-shift-bounds-Suc-ivl* [*symmetric*],
subst atLeastLessThanSuc-atLeastAtMost) *simp-all*
also have ... = *z* ∗ *of-real* (*harm n*) − (∑ *k*=*1*..*n*. *ln* (*1* + *z* / *of-nat k*))
by (*simp add*: *harm-def setsum-subtractf setsum-right-distrib divide-inverse*)
also from *n* **have** ... − *?g n* = *0*
by (*simp add*: *ln-Gamma-series-def setsum-subtractf algebra-simps Ln-of-nat*)
finally show (∑ *k*<*n*. *?f k*) − *?g n* = *0* **.**
qed
show (λ*n*. (∑ *k*<*n*. *?f k*) − *?g n*) ⟶ *0* **by** (*subst tendsto-cong*[*OF A*]) *simp-all*
qed

lemma *uniformly-summable-deriv-ln-Gamma*:
assumes *z*: (*z* :: *'a* :: {*real-normed-field*,*banach*}) ≠ *0* **and** *d*: *d* > *0* *d* ≤ *norm z*/*2*
shows *uniformly-convergent-on* (*ball z d*)
(λ*k z*. ∑ *i*<*k*. *inverse* (*of-nat* (*Suc i*)) − *inverse* (*z* + *of-nat* (*Suc i*)))
(**is** *uniformly-convergent-on* - (λ*k z*. ∑ *i*<*k*. *?f i z*))
proof (*rule weierstrass-m-test'-ev*)
{
fix *t* **assume** *t*: *t* ∈ *ball z d*
have *norm z* = *norm* (*t* + (*z* − *t*)) **by** *simp*
have *norm* (*t* + (*z* − *t*)) ≤ *norm t* + *norm* (*z* − *t*) **by** (*rule norm-triangle-ineq*)
also from *t d* **have** *norm* (*z* − *t*) < *norm z* / *2* **by** (*simp add*: *dist-norm*)
finally have *A*: *norm t* > *norm z* / *2* **using** *z* **by** (*simp add*: *field-simps*)

have *norm t* = *norm* (*z* + (*t* − *z*)) **by** *simp*
also have ... ≤ *norm z* + *norm* (*t* − *z*) **by** (*rule norm-triangle-ineq*)
also from *t d* **have** *norm* (*t* − *z*) ≤ *norm z* / *2* **by** (*simp add*: *dist-norm norm-minus-commute*)
also from *z* **have** ... < *norm z* **by** *simp*
finally have *B*: *norm t* < *2* ∗ *norm z* **by** *simp*
note *A B*
} **note** *ball* = *this*

show *eventually* (λ*n*. ∀ *t*∈*ball z d*. *norm* (*?f n t*) ≤ *4* ∗ *norm z* ∗ *inverse* (*of-nat* (*Suc n*) ^ *2*)) *sequentially*
using *eventually-gt-at-top* **apply** *eventually-elim*
proof *safe*
fix *t* :: *'a* **assume** *t*: *t* ∈ *ball z d*
from *z ball*[*OF t*] **have** *t-nz*: *t* ≠ *0* **by** *auto*
fix *n* :: *nat* **assume** *n*: *n* > *nat* ⌈*4* ∗ *norm z*⌉
from *ball*[*OF t*] *t-nz* **have** *4* ∗ *norm z* > *2* ∗ *norm t* **by** *simp*
also from *n* **have** ... < *of-nat n* **by** *linarith*
finally have *n*: *of-nat n* > *2* ∗ *norm t* **.**
hence *of-nat n* > *norm t* **by** *simp*
hence *t'*: *t* ≠ −*of-nat* (*Suc n*) **by** (*intro notI*) (*simp del*: *of-nat-Suc*)

with *t-nz* **have** *?f n t* = *1* / (*of-nat* (*Suc n*) ∗ (*1* + *of-nat* (*Suc n*)/*t*))
by (*simp add*: *divide-simps eq-neg-iff-add-eq-0 del*: *of-nat-Suc*)
also have *norm* ... = *inverse* (*of-nat* (*Suc n*)) ∗ *inverse* (*norm* (*of-nat* (*Suc n*)/*t* + *1*))
by (*simp add*: *norm-divide norm-mult divide-simps add-ac del*: *of-nat-Suc*)
also {
from *z t-nz ball*[*OF t*] **have** *of-nat* (*Suc n*) / (*4* ∗ *norm z*) ≤ *of-nat* (*Suc n*) / (*2* ∗ *norm t*)
by (*intro divide-left-mono mult-pos-pos*) *simp-all*
also have ... < *norm* (*of-nat* (*Suc n*) / *t*) − *norm* (*1* :: *'a*)
using *t-nz n* **by** (*simp add*: *field-simps norm-divide del*: *of-nat-Suc*)
also have ... ≤ *norm* (*of-nat* (*Suc n*)/*t* + *1*) **by** (*rule norm-diff-ineq*)
finally have *inverse* (*norm* (*of-nat* (*Suc n*)/*t* + *1*)) ≤ *4* ∗ *norm z* / *of-nat* (*Suc n*)
using *z* **by** (*simp add*: *divide-simps norm-divide mult-ac del*: *of-nat-Suc*)
}
also have *inverse* (*real-of-nat* (*Suc n*)) ∗ (*4* ∗ *norm z* / *real-of-nat* (*Suc n*)) =
4 ∗ *norm z* ∗ *inverse* (*of-nat* (*Suc n*) ^ *2*)
by (*simp add*: *divide-simps power2-eq-square del*: *of-nat-Suc*)
finally show *norm* (*?f n t*) ≤ *4* ∗ *norm z* ∗ *inverse* (*of-nat* (*Suc n*) ^ *2*)
by (*simp del*: *of-nat-Suc*)
qed
next
show *summable* (λ*n*. *4* ∗ *norm z* ∗ *inverse* ((*of-nat* (*Suc n*)) ^ *2*))
by (*subst summable-Suc-iff*) (*simp add*: *summable-mult inverse-power-summable*)
qed

lemma *summable-deriv-ln-Gamma*:
z ≠ (*0* :: *'a* :: {*real-normed-field*,*banach*}) ⟹
summable (λ*n*. *inverse* (*of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat* (*Suc n*)))
unfolding *summable-iff-convergent*
by (*rule uniformly-convergent-imp-convergent*,
rule uniformly-summable-deriv-ln-Gamma[*of z norm z*/*2*]) *simp-all*

definition *Polygamma* :: *nat* ⇒ (*'a* :: {*real-normed-field*,*banach*}) ⇒ *'a* **where**

Polygamma n z = (if n = 0 then
(∑k. inverse (of-nat (Suc k)) − inverse (z + of-nat k)) − euler-mascheroni
else
*(−1) ^Suc n * fact n * (∑k. inverse ((z + of-nat k) ^Suc n)))*

abbreviation *Digamma* :: ('*a* :: {*real-normed-field*,*banach*}) ⇒ '*a* **where**
Digamma ≡ Polygamma 0

lemma *Digamma-def*:
Digamma z = (∑k. inverse (of-nat (Suc k)) − inverse (z + of-nat k)) − euler-mascheroni
by (*simp add*: *Polygamma-def*)

lemma *summable-Digamma*:
assumes (*z* :: '*a* :: {*real-normed-field*,*banach*}) ≠ *0*
shows *summable* (λ*n*. *inverse* (*of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat n*))
proof −
have *sums*: (λ*n*. *inverse* (*z* + *of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat n*)) *sums*
(*0* − *inverse* (*z* + *of-nat 0*))
by (*intro telescope-sums filterlim-compose*[*OF tendsto-inverse-0*]
tendsto-add-filterlim-at-infinity[*OF tendsto-const*] *tendsto-of-nat*)
from *summable-add*[*OF summable-deriv-ln-Gamma*[*OF assms*] *sums-summable*[*OF sums*]]
show *summable* (λ*n*. *inverse* (*of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat n*)) **by** *simp*
qed

lemma *summable-offset*:
assumes *summable* (λ*n*. *f* (*n* + *k*) :: '*a* :: *real-normed-vector*)
shows *summable f*
proof −
from *assms* **have** *convergent* (λ*m*. ∑ *n*<*m*. *f* (*n* + *k*)) **by** (*simp add*: *summable-iff-convergent*)
hence *convergent* (λ*m*. (∑ *n*<*k*. *f n*) + (∑ *n*<*m*. *f* (*n* + *k*))))
by (*intro convergent-add convergent-const*)
also have (λ*m*. (∑ *n*<*k*. *f n*) + (∑ *n*<*m*. *f* (*n* + *k*))) = (λ*m*. ∑ *n*<*m*+*k*. *f n*)
proof
fix *m* :: *nat*
have {..<*m*+*k*} = {..<*k*} ∪ {*k*..<*m*+*k*} **by** *auto*
also have (∑ *n*∈.... *f n*) = (∑ *n*<*k*. *f n*) + (∑ *n*=*k*..<*m*+*k*. *f n*)
by (*rule setsum.union-disjoint*) *auto*
also have (∑ *n*=*k*..<*m*+*k*. *f n*) = (∑ *n*=*0*..<*m*+*k*−*k*. *f* (*n* + *k*))
by (*intro setsum.reindex-cong*[*of* λ*n*. *n* + *k*])
(*simp*, *subst image-add-atLeastLessThan*, *auto*)
finally show (∑ *n*<*k*. *f n*) + (∑ *n*<*m*. *f* (*n* + *k*)) = (∑ *n*<*m*+*k*. *f n*) **by** (*simp add*: *atLeast0LessThan*)
qed
finally have (λ*a*. *setsum f* {..<*a*}) ⟶ *lim* (λ*m*. *setsum f* {..<*m* + *k*})
by (*auto simp*: *convergent-LIMSEQ-iff dest*: *LIMSEQ-offset*)

thus *?thesis* **by** (*auto simp*: *summable-iff-convergent convergent-def*)
qed

lemma *Polygamma-converges*:
fixes $z :: 'a :: \{real\text{-}normed\text{-}field, banach\}$
assumes z: $z \neq 0$ **and** n: $n \geq 2$
shows *uniformly-convergent-on* (*ball z d*) ($\lambda k\ z.\ \sum i{<}k.\ inverse\ ((z + of\text{-}nat\ i)\ \hat{}\ n)$)
proof (*rule weierstrass-m-test'-ev*)
def $e \equiv (1 + d\ /\ norm\ z)$
def $m \equiv nat\ \lceil norm\ z * e \rceil$
{
fix t **assume** t: $t \in ball\ z\ d$
have $norm\ t = norm\ (z + (t - z))$ **by** *simp*
also have $\ldots \leq norm\ z + norm\ (t - z)$ **by** (*rule norm-triangle-ineq*)
also from t **have** $norm\ (t - z) < d$ **by** (*simp add*: *dist-norm norm-minus-commute*)
finally have $norm\ t < norm\ z * e$ **using** z **by** (*simp add*: *divide-simps e-def*)
} note $ball = this$

show *eventually* ($\lambda k.\ \forall t{\in}ball\ z\ d.\ norm\ (inverse\ ((t + of\text{-}nat\ k)\ \hat{}\ n)) \leq$
$inverse\ (of\text{-}nat\ (k - m)\ \hat{}\ n)$) *sequentially*
using *eventually-gt-at-top*[*of m*] **apply** *eventually-elim*
proof (*intro ballI*)
fix $k :: nat$ **and** $t :: 'a$ **assume** k: $k > m$ **and** t: $t \in ball\ z\ d$
from k **have** $real\text{-}of\text{-}nat\ (k - m) = of\text{-}nat\ k - of\text{-}nat\ m$ **by** (*simp add*: *of-nat-diff*)
also have $\ldots \leq norm\ (of\text{-}nat\ k :: 'a) - norm\ z * e$
unfolding *m-def* **by** (*subst norm-of-nat*) *linarith*
also from *ball*[*OF t*] **have** $\ldots \leq norm\ (of\text{-}nat\ k :: 'a) - norm\ t$ **by** *simp*
also have $\ldots \leq norm\ (of\text{-}nat\ k + t)$ **by** (*rule norm-diff-ineq*)
finally have $inverse\ ((norm\ (t + of\text{-}nat\ k))\ \hat{}\ n) \leq inverse\ (real\text{-}of\text{-}nat\ (k - m)\ \hat{}\ n)$ **using** $k\ n$
by (*intro le-imp-inverse-le power-mono*) (*simp-all add*: *add-ac del*: *of-nat-Suc*)
thus $norm\ (inverse\ ((t + of\text{-}nat\ k)\ \hat{}\ n)) \leq inverse\ (of\text{-}nat\ (k - m)\ \hat{}\ n)$
by (*simp add*: *norm-inverse norm-power power-inverse*)
qed

have *summable* ($\lambda k.\ inverse\ ((real\text{-}of\text{-}nat\ k)\ \hat{}\ n)$)
using *inverse-power-summable*[*of n*] n **by** *simp*
hence *summable* ($\lambda k.\ inverse\ ((real\text{-}of\text{-}nat\ (k + m - m))\ \hat{}\ n)$) **by** *simp*
thus *summable* ($\lambda k.\ inverse\ ((real\text{-}of\text{-}nat\ (k - m))\ \hat{}\ n)$) **by** (*rule summable-offset*)
qed

lemma *Polygamma-converges'*:
fixes $z :: 'a :: \{real\text{-}normed\text{-}field, banach\}$
assumes z: $z \neq 0$ **and** n: $n \geq 2$
shows *summable* ($\lambda k.\ inverse\ ((z + of\text{-}nat\ k)\ \hat{}\ n)$)
using *uniformly-convergent-imp-convergent*[*OF Polygamma-converges*[*OF assms*, *of 1*], *of z*]

by (*simp add*: *summable-iff-convergent*)

lemma *has-field-derivative-ln-Gamma-complex* [*derivative-intros*]:
fixes z :: *complex*
assumes z: $z \notin \mathbb{R}_{\leq 0}$
shows (*ln-Gamma has-field-derivative Digamma z*) (*at z*)
proof −
have *not-nonpos-Int* [*simp*]: $t \notin \mathbb{Z}_{\leq 0}$ **if** *Re t* > *0* **for** t
using *that* **by** (*auto elim*!: *nonpos-Ints-cases'*)
from z **have** z': $z \notin \mathbb{Z}_{\leq 0}$ **and** z'': $z \neq 0$ **using** *nonpos-Ints-subset-nonpos-Reals nonpos-Reals-zero-I*
by *blast*+
let *?f'* = $\lambda z\ k.$ *inverse* (*of-nat* (*Suc k*)) − *inverse* (*z* + *of-nat* (*Suc k*))
let *?f* = $\lambda z\ k.$ *z* / *of-nat* (*Suc k*) − *ln* (*1* + *z* / *of-nat* (*Suc k*)) **and** *?F'* = $\lambda z.$ $\sum n.$ *?f' z n*
def *d* ≡ *min* (*norm z*/*2*) (*if Im z* = *0 then Re z* / *2 else abs* (*Im z*) / *2*)
from z **have** *d*: *d* > *0 norm z*/*2* ≥ *d* **by** (*auto simp add*: *complex-nonpos-Reals-iff d-def*)
have *ball*: *Im t* = *0* ⟶ *Re t* > *0* **if** *dist z t* < *d* **for** t
using *no-nonpos-Real-in-ball*[*OF z*, *of t*] *that* **unfolding** *d-def* **by** (*force simp add*: *complex-nonpos-Reals-iff*)
have *sums*: ($\lambda n.$ *inverse* (*z* + *of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat n*)) *sums* (*0* − *inverse* (*z* + *of-nat 0*))
by (*intro telescope-sums filterlim-compose*[*OF tendsto-inverse-0*] *tendsto-add-filterlim-at-infinity*[*OF tendsto-const*] *tendsto-of-nat*)

have (($\lambda z.$ $\sum n.$ *?f z n*) *has-field-derivative ?F' z*) (*at z*)
using *d z ln-Gamma-series'-aux*[*OF z'*]
apply (*intro has-field-derivative-series'*(*2*)[*of ball z d - - z*] *uniformly-summable-deriv-ln-Gamma*)
apply (*auto intro*!: *derivative-eq-intros add-pos-pos mult-pos-pos dest*!: *ball simp*: *field-simps sums-iff nonpos-Reals-divide-of-nat-iff simp del*: *of-nat-Suc*)
apply (*auto simp add*: *complex-nonpos-Reals-iff*)
done
with *z* **have** (($\lambda z.$ ($\sum k.$ *?f z k*) − *euler-mascheroni* ∗ *z* − *Ln z*) *has-field-derivative ?F' z* − *euler-mascheroni* − *inverse z*) (*at z*)
by (*force intro*!: *derivative-eq-intros simp*: *Digamma-def*)
also have *?F' z* − *euler-mascheroni* − *inverse z* = (*?F' z* + −*inverse z*) − *euler-mascheroni* **by** *simp*
also from *sums* **have** −*inverse z* = ($\sum n.$ *inverse* (*z* + *of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat n*))
by (*simp add*: *sums-iff*)
also from *sums summable-deriv-ln-Gamma*[*OF z''*]
have *?F' z* + ... = ($\sum n.$ *inverse* (*of-nat* (*Suc n*)) − *inverse* (*z* + *of-nat n*))
by (*subst suminf-add*) (*simp-all add*: *add-ac sums-iff*)
also have ... − *euler-mascheroni* = *Digamma z* **by** (*simp add*: *Digamma-def*)
finally have (($\lambda z.$ ($\sum k.$ *?f z k*) − *euler-mascheroni* ∗ *z* − *Ln z*) *has-field-derivative Digamma z*) (*at z*) .
moreover from *eventually-nhds-ball*[*OF d*(*1*), *of z*]

have *eventually* (λz. *ln-Gamma* $z = (\sum k.$ *?f* z $k)$ $-$ *euler-mascheroni* $*$ $z -$ *Ln* z) (*nhds* z)
proof *eventually-elim*
fix t **assume** $t \in$ *ball* z d
hence $t \notin \mathbb{Z}_{\leq 0}$ **by** (*auto dest*!: *ball elim*!: *nonpos-Ints-cases*)
from *ln-Gamma-series'-aux*[*OF this*]
show *ln-Gamma* $t = (\sum k.$ *?f* t $k)$ $-$ *euler-mascheroni* $*$ $t -$ *Ln* t **by** (*simp add*: *sums-iff*)
qed
ultimately show *?thesis* **by** (*subst DERIV-cong-ev*[*OF refl - refl*])
qed

declare *has-field-derivative-ln-Gamma-complex*[*THEN DERIV-chain2*, *derivative-intros*]

lemma *Digamma-1* [*simp*]: *Digamma* (*1* :: *'a* :: {*real-normed-field*,*banach*}) $=$ $-$ *euler-mascheroni*
by (*simp add*: *Digamma-def*)

lemma *Digamma-plus1*:
assumes $z \neq 0$
shows *Digamma* $(z+1)$ $=$ *Digamma* z $+$ $1/z$
proof $-$
have *sums*: $(\lambda k.$ *inverse* $(z +$ *of-nat* $k)$ $-$ *inverse* $(z +$ *of-nat* $(Suc\ k)))$
sums (*inverse* $(z +$ *of-nat* $0)$ $-$ $0)$
by (*intro telescope-sums'*[*OF filterlim-compose*[*OF tendsto-inverse-0*]]
tendsto-add-filterlim-at-infinity[*OF tendsto-const*] *tendsto-of-nat*)
have *Digamma* $(z+1) = (\sum k.$ *inverse* $($*of-nat* $(Suc\ k))$ $-$ *inverse* $(z +$ *of-nat* $(Suc\ k))) -$
euler-mascheroni (**is** - $=$ *suminf ?f* $-$ -) **by** (*simp add*: *Digamma-def add-ac*)
also have *suminf ?f* $= (\sum k.$ *inverse* $($*of-nat* $(Suc\ k))$ $-$ *inverse* $(z +$ *of-nat* $k))$ $+$
$(\sum k.$ *inverse* $(z +$ *of-nat* $k)$ $-$ *inverse* $(z +$ *of-nat* $(Suc\ k)))$
using *summable-Digamma*[*OF assms*] *sums* **by** (*subst suminf-add*) (*simp-all add*: *add-ac sums-iff*)
also have $(\sum k.$ *inverse* $(z +$ *of-nat* $k)$ $-$ *inverse* $(z +$ *of-nat* $(Suc\ k))) = 1/z$
using *sums* **by** (*simp add*: *sums-iff inverse-eq-divide*)
finally show *?thesis* **by** (*simp add*: *Digamma-def*[*of z*])
qed

lemma *Polygamma-plus1*:
assumes $z \neq 0$
shows *Polygamma* n $(z + 1) =$ *Polygamma* n $z + (-1)\hat{}n * fact\ n$ / $(z\ \hat{}\ Suc\ n)$
proof (*cases* $n = 0$)
assume n: $n \neq 0$
let *?f* $= \lambda k.$ *inverse* $((z +$ *of-nat* $k)\ \hat{}\ Suc\ n)$
have *Polygamma* n $(z + 1) = (-1)\ \hat{}\ Suc\ n * fact\ n * (\sum k.$ *?f* $(k+1))$

using *n* **by** (*simp add*: *Polygamma-def add-ac*)
also have $(\sum k.\ ?f\ (k+1)) + (\sum k<1.\ ?f\ k) = (\sum k.\ ?f\ k)$
using *Polygamma-converges'*[*OF assms*, *of Suc n*] *n*
by (*subst suminf-split-initial-segment* [*symmetric*]) *simp-all*
hence $(\sum k.\ ?f\ (k+1)) = (\sum k.\ ?f\ k) - inverse\ (z\ \hat{}\ Suc\ n)$ **by** (*simp add*: *algebra-simps*)
also have $(-1)\ \hat{}\ Suc\ n * fact\ n * ((\sum k.\ ?f\ k) - inverse\ (z\ \hat{}\ Suc\ n)) =$
$Polygamma\ n\ z + (-1)\hat{}n * fact\ n\ /\ (z\ \hat{}\ Suc\ n)$ **using** *n*
by (*simp add*: *inverse-eq-divide algebra-simps Polygamma-def*)
finally show *?thesis* **.**
qed (*insert assms*, *simp add*: *Digamma-plus1 inverse-eq-divide*)

lemma *Digamma-of-nat*:
$Digamma\ (of\text{-}nat\ (Suc\ n) :: 'a :: \{real\text{-}normed\text{-}field, banach\}) = harm\ n - euler\text{-}mascheroni$
proof (*induction n*)
case (*Suc n*)
have $Digamma\ (of\text{-}nat\ (Suc\ (Suc\ n)) :: 'a) = Digamma\ (of\text{-}nat\ (Suc\ n) + 1)$
by *simp*
also have $\ldots = Digamma\ (of\text{-}nat\ (Suc\ n)) + inverse\ (of\text{-}nat\ (Suc\ n))$
by (*subst Digamma-plus1*) (*simp-all add*: *inverse-eq-divide del*: *of-nat-Suc*)
also have $Digamma\ (of\text{-}nat\ (Suc\ n) :: 'a) = harm\ n - euler\text{-}mascheroni$ **by** (*rule Suc*)
also have $\ldots + inverse\ (of\text{-}nat\ (Suc\ n)) = harm\ (Suc\ n) - euler\text{-}mascheroni$
by (*simp add*: *harm-Suc*)
finally show *?case* **.**
qed (*simp add*: *harm-def*)

lemma *Digamma-numeral*: $Digamma\ (numeral\ n) = harm\ (pred\text{-}numeral\ n) - euler\text{-}mascheroni$
by (*subst of-nat-numeral*[*symmetric*], *subst numeral-eq-Suc*, *subst Digamma-of-nat*) (*rule refl*)

lemma *Polygamma-of-real*: $x \neq 0 \Longrightarrow Polygamma\ n\ (of\text{-}real\ x) = of\text{-}real\ (Polygamma\ n\ x)$
unfolding *Polygamma-def* **using** *summable-Digamma*[*of x*] *Polygamma-converges'*[*of x Suc n*]
by (*simp-all add*: *suminf-of-real*)

lemma *Polygamma-Real*: $z \in \mathbb{R} \Longrightarrow z \neq 0 \Longrightarrow Polygamma\ n\ z \in \mathbb{R}$
by (*elim Reals-cases*, *hypsubst*, *subst Polygamma-of-real*) *simp-all*

lemma *Digamma-half-integer*:
$Digamma\ (of\text{-}nat\ n + 1/2 :: 'a :: \{real\text{-}normed\text{-}field, banach\}) =$
$(\sum k<n.\ 2\ /\ (of\text{-}nat\ (2*k+1))) - euler\text{-}mascheroni - of\text{-}real\ (2 * ln\ 2)$
proof (*induction n*)
case *0*
have $Digamma\ (1/2 :: 'a) = of\text{-}real\ (Digamma\ (1/2))$ **by** (*simp add*: *Polygamma-of-real* [*symmetric*])
also have $Digamma\ (1/2::real) =$

$(\sum k.\ inverse\ (of\text{-}nat\ (Suc\ k))\ -\ inverse\ (of\text{-}nat\ k\ +\ 1/2))\ -$
euler-mascheroni
by (*simp add*: *Digamma-def add-ac*)
also have $(\sum k.\ inverse\ (of\text{-}nat\ (Suc\ k)\ ::\ real)\ -\ inverse\ (of\text{-}nat\ k\ +\ 1/2)) =$
$(\sum k.\ inverse\ (1/2)\ *\ (inverse\ (2\ *\ of\text{-}nat\ (Suc\ k))\ -\ inverse\ (2\ *$
of-nat k + 1)))
by (*simp-all add*: *add-ac inverse-mult-distrib*[*symmetric*] *ring-distribs del*: *inverse-divide*)
also have $\ldots = -\ 2\ *\ ln\ 2$ **using** *sums-minus*[*OF alternating-harmonic-series-sums'*]
by (*subst suminf-mult*) (*simp-all add*: *algebra-simps sums-iff*)
finally show *?case* **by** *simp*
next
case (*Suc n*)
have *nz*: $2\ *\ of\text{-}nat\ n\ +\ (1::'a)\ \neq\ 0$
using *of-nat-neq-0*[*of 2*∗*n*] **by** (*simp only*: *of-nat-Suc*) (*simp add*: *add-ac*)
hence *nz'*: $of\text{-}nat\ n\ +\ (1/2::'a)\ \neq\ 0$ **by** (*simp add*: *field-simps*)
have $Digamma\ (of\text{-}nat\ (Suc\ n)\ +\ 1/2\ ::\ 'a) = Digamma\ (of\text{-}nat\ n\ +\ 1/2\ +\ 1)$
by *simp*
also from *nz'* **have** $\ldots = Digamma\ (of\text{-}nat\ n\ +\ 1\ /\ 2)\ +\ 1\ /\ (of\text{-}nat\ n\ +\ 1\ /$
2)
by (*rule Digamma-plus1*)
also from *nz nz'* **have** $1\ /\ (of\text{-}nat\ n\ +\ 1\ /\ 2\ ::\ 'a) = 2\ /\ (2\ *\ of\text{-}nat\ n\ +\ 1)$
by (*subst divide-eq-eq*) *simp-all*
also note *Suc*
finally show *?case* **by** (*simp add*: *add-ac*)
qed

lemma *Digamma-one-half*: $Digamma\ (1/2) = -\ euler\text{-}mascheroni\ -\ of\text{-}real\ (2$
$*\ ln\ 2)$
using *Digamma-half-integer*[*of 0*] **by** *simp*

lemma *Digamma-real-three-halves-pos*: $Digamma\ (3/2\ ::\ real) > 0$
proof −
have $-Digamma\ (3/2\ ::\ real) = -Digamma\ (of\text{-}nat\ 1\ +\ 1/2)$ **by** *simp*
also have $\ldots = 2\ *\ ln\ 2\ +\ euler\text{-}mascheroni\ -\ 2$ **by** (*subst Digamma-half-integer*)
simp
also note *euler-mascheroni-less-13-over-22*
also note *ln2-le-25-over-36*
finally show *?thesis* **by** *simp*
qed

lemma *has-field-derivative-Polygamma* [*derivative-intros*]:
fixes $z\ ::\ 'a\ ::\ \{real\text{-}normed\text{-}field, euclidean\text{-}space\}$
assumes *z*: $z\ \notin\ \mathbb{Z}_{\leq 0}$
shows (*Polygamma n has-field-derivative Polygamma* (*Suc n*) *z*) (*at z within A*)
proof (*rule has-field-derivative-at-within*, *cases n = 0*)
assume *n*: $n = 0$
let *?f* = $\lambda k\ z.\ inverse\ (of\text{-}nat\ (Suc\ k))\ -\ inverse\ (z\ +\ of\text{-}nat\ k)$
let *?F* = $\lambda z.\ \sum k.\ ?f\ k\ z$ **and** *?f'* = $\lambda k\ z.\ inverse\ ((z\ +\ of\text{-}nat\ k)^2)$

from *no-nonpos-Int-in-ball'*[*OF z*] **guess** *d* **. note** *d = this*
from *z* **have** *summable*: *summable* (λ*k*. *inverse* (*of-nat* (*Suc k*)) − *inverse* (*z* + *of-nat k*))
by (*intro summable-Digamma*) *force*
from *z* **have** *conv*: *uniformly-convergent-on* (*ball z d*) (λ*k z*. ∑*i*<*k*. *inverse* ((*z* + *of-nat i*)^2))
by (*intro Polygamma-converges*) *auto*
with *d* **have** *summable* (λ*k*. *inverse* ((*z* + *of-nat k*)^2)) **unfolding** *summable-iff-convergent*
by (*auto dest*!: *uniformly-convergent-imp-convergent simp*: *summable-iff-convergent*
)

have (*?F has-field-derivative* (∑ *k*. *?f' k z*)) (*at z*)
proof (*rule has-field-derivative-series'*[*of ball z d - - z*])
fix *k* :: *nat* **and** *t* :: *'a* **assume** *t*: *t* ∈ *ball z d*
from *t d*(*2*)[*of t*] **show** ((λ*z*. *?f k z*) *has-field-derivative ?f' k t*) (*at t within ball z d*)
by (*auto intro*!: *derivative-eq-intros simp*: *power2-eq-square simp del*: *of-nat-Suc dest*!: *plus-of-nat-eq-0-imp elim*!: *nonpos-Ints-cases*)
qed (*insert d*(*1*) *summable conv*, (*assumption*|*simp*)+)
with *z* **show** (*Polygamma n has-field-derivative Polygamma* (*Suc n*) *z*) (*at z*)
unfolding *Digamma-def* [*abs-def*] *Polygamma-def* [*abs-def*] **using** *n*
by (*force simp*: *power2-eq-square intro*!: *derivative-eq-intros*)
next
assume *n*: *n* ≠ *0*
from *z* **have** *z'*: *z* ≠ *0* **by** *auto*
from *no-nonpos-Int-in-ball'*[*OF z*] **guess** *d* **. note** *d = this*
def *n'* ≡ *Suc n*
from *n* **have** *n'*: *n'* ≥ *2* **by** (*simp add*: *n'-def*)
have ((λ*z*. ∑ *k*. *inverse* ((*z* + *of-nat k*) ^ *n'*)) *has-field-derivative*
(∑ *k*. − *of-nat n'* ∗ *inverse* ((*z* + *of-nat k*) ^ (*n'*+*1*)))) (*at z*)
proof (*rule has-field-derivative-series'*[*of ball z d - - z*])
fix *k* :: *nat* **and** *t* :: *'a* **assume** *t*: *t* ∈ *ball z d*
with *d* **have** *t'*: *t* ∉ $\mathbb{Z}_{\leq 0}$ *t* ≠ *0* **by** *auto*
show ((λ*a*. *inverse* ((*a* + *of-nat k*) ^ *n'*)) *has-field-derivative*
− *of-nat n'* ∗ *inverse* ((*t* + *of-nat k*) ^ (*n'*+*1*))) (*at t within ball z d*)
using *t'*
by (*fastforce intro*!: *derivative-eq-intros simp*: *divide-simps power-diff dest*: *plus-of-nat-eq-0-imp*)
next
have *uniformly-convergent-on* (*ball z d*)
(λ*k z*. (− *of-nat n'* :: *'a*) ∗ (∑ *i*<*k*. *inverse* ((*z* + *of-nat i*) ^ (*n'*+*1*))))
using *z' n* **by** (*intro uniformly-convergent-mult Polygamma-converges*) (*simp-all add*: *n'-def*)
thus *uniformly-convergent-on* (*ball z d*)
(λ*k z*. ∑ *i*<*k*. − *of-nat n'* ∗ *inverse* ((*z* + *of-nat i* :: *'a*) ^ (*n'*+*1*)))
by (*subst* (*asm*) *setsum-right-distrib*) *simp*
qed (*insert Polygamma-converges'*[*OF z' n'*] *d*, *simp-all*)
also have (∑ *k*. − *of-nat n'* ∗ *inverse* ((*z* + *of-nat k*) ^ (*n'* + *1*))) =
(− *of-nat n'*) ∗ (∑ *k*. *inverse* ((*z* + *of-nat k*) ^ (*n'* + *1*)))

using *Polygamma-converges'*[*OF z'*, *of n'+1*] *n'* **by** (*subst suminf-mult*) *simp-all*
finally have ((λ*z*. ∑ *k*. *inverse* ((*z* + *of-nat k*) ^ *n'*)) *has-field-derivative*
− *of-nat n'* ∗ (∑ *k*. *inverse* ((*z* + *of-nat k*) ^ (*n'* + 1)))) (*at z*) .
from *DERIV-cmult*[*OF this*, *of* (−1) ^*Suc n* ∗ *fact n* :: *'a*]
show (*Polygamma n has-field-derivative Polygamma* (*Suc n*) *z*) (*at z*)
unfolding *n'-def Polygamma-def*[*abs-def*] **using** *n* **by** (*simp add*: *algebra-simps*)
qed

declare *has-field-derivative-Polygamma*[*THEN DERIV-chain2*, *derivative-intros*]

lemma *isCont-Polygamma* [*continuous-intros*]:
fixes *f* :: - ⇒ *'a* :: {*real-normed-field*,*euclidean-space*}
shows *isCont f z* ⟹ *f z* ∉ $\mathbb{Z}_{\leq 0}$ ⟹ *isCont* (λ*x*. *Polygamma n* (*f x*)) *z*
by (*rule isCont-o2*[*OF* - *DERIV-isCont*[*OF has-field-derivative-Polygamma*]])

lemma *continuous-on-Polygamma*:
A ∩ $\mathbb{Z}_{\leq 0}$ = {} ⟹ *continuous-on A* (*Polygamma n* :: - ⇒ *'a* :: {*real-normed-field*,*euclidean-space*})
by (*intro continuous-at-imp-continuous-on isCont-Polygamma*[*OF continuous-ident*]
ballI) *blast*

lemma *isCont-ln-Gamma-complex* [*continuous-intros*]:
fixes *f* :: *'a*::*t2-space* ⇒ *complex*
shows *isCont f z* ⟹ *f z* ∉ $\mathbb{R}_{\leq 0}$ ⟹ *isCont* (λ*z*. *ln-Gamma* (*f z*)) *z*
by (*rule isCont-o2*[*OF* - *DERIV-isCont*[*OF has-field-derivative-ln-Gamma-complex*]])

lemma *continuous-on-ln-Gamma-complex* [*continuous-intros*]:
fixes *A* :: *complex set*
shows *A* ∩ $\mathbb{R}_{\leq 0}$ = {} ⟹ *continuous-on A ln-Gamma*
by (*intro continuous-at-imp-continuous-on ballI isCont-ln-Gamma-complex*[*OF*
continuous-ident])
fastforce

We define a type class that captures all the fundamental properties of the inverse of the Gamma function and defines the Gamma function upon that. This allows us to instantiate the type class both for the reals and for the complex numbers with a minimal amount of proof duplication.

class *Gamma* = *real-normed-field* + *complete-space* +
fixes *rGamma* :: *'a* ⇒ *'a*
assumes *rGamma-eq-zero-iff-aux*: *rGamma z* = *0* ⟷ (∃ *n*. *z* = − *of-nat n*)
assumes *differentiable-rGamma-aux1*:
(⋀*n*. *z* ≠ − *of-nat n*) ⟹
let d = (*THE d*. (λ*n*. ∑ *k*<*n*. *inverse* (*of-nat* (*Suc k*)) − *inverse* (*z* + *of-nat*
k))
⟶ *d*) − *scaleR euler-mascheroni 1*
in filterlim (λ*y*. (*rGamma y* − *rGamma z* + *rGamma z* ∗ *d* ∗ (*y* − *z*)) /$_R$
norm (*y* − *z*)) (*nhds 0*) (*at z*)
assumes *differentiable-rGamma-aux2*:
let z = − *of-nat n*
in filterlim (λ*y*. (*rGamma y* − *rGamma z* − (−1) ^*n* ∗ (*setprod of-nat* {*1*..*n*})

$* (y - z)) /_R$
$norm\ (y - z))\ (nhds\ 0)\ (at\ z)$
assumes *rGamma-series-aux*: $(\bigwedge n.\ z \neq - of\text{-}nat\ n) \Longrightarrow$
$let\ fact' = (\lambda n.\ setprod\ of\text{-}nat\ \{1..n\});$
$exp = (\lambda x.\ THE\ e.\ (\lambda n.\ \sum k{<}n.\ x\hat{}k\ /_R\ fact\ k) \longrightarrow e);$
$pochhammer' = (\lambda a\ n.\ (\prod n = 0..n.\ a + of\text{-}nat\ n))$
$in\ \ filterlim\ (\lambda n.\ pochhammer'\ z\ n\ /\ (fact'\ n * exp\ (z * (ln\ (of\text{-}nat\ n)$
$*_R\ 1))))$
$(nhds\ (rGamma\ z))\ sequentially$
begin
subclass *banach* **..**
end

definition $Gamma\ z = inverse\ (rGamma\ z)$

57.3 Basic properties

lemma *Gamma-nonpos-Int*: $z \in \mathbb{Z}_{\leq 0} \Longrightarrow Gamma\ z = 0$
and *rGamma-nonpos-Int*: $z \in \mathbb{Z}_{\leq 0} \Longrightarrow rGamma\ z = 0$
using *rGamma-eq-zero-iff-aux*[*of z*] **unfolding** *Gamma-def* **by** (*auto elim*!: *nonpos-Ints-cases'*)

lemma *Gamma-nonzero*: $z \notin \mathbb{Z}_{\leq 0} \Longrightarrow Gamma\ z \neq 0$
and *rGamma-nonzero*: $z \notin \mathbb{Z}_{\leq 0} \Longrightarrow rGamma\ z \neq 0$
using *rGamma-eq-zero-iff-aux*[*of z*] **unfolding** *Gamma-def* **by** (*auto elim*!: *nonpos-Ints-cases'*)

lemma *Gamma-eq-zero-iff*: $Gamma\ z = 0 \longleftrightarrow z \in \mathbb{Z}_{\leq 0}$
and *rGamma-eq-zero-iff*: $rGamma\ z = 0 \longleftrightarrow z \in \mathbb{Z}_{\leq 0}$
using *rGamma-eq-zero-iff-aux*[*of z*] **unfolding** *Gamma-def* **by** (*auto elim*!: *nonpos-Ints-cases'*)

lemma *rGamma-inverse-Gamma*: $rGamma\ z = inverse\ (Gamma\ z)$
unfolding *Gamma-def* **by** *simp*

lemma *rGamma-series-LIMSEQ* [*tendsto-intros*]:
$rGamma\text{-}series\ z \longrightarrow rGamma\ z$
proof (*cases* $z \in \mathbb{Z}_{\leq 0}$)
case *False*
hence $z \neq - of\text{-}nat\ n$ **for** *n* **by** *auto*
from *rGamma-series-aux*[*OF this*] **show** *?thesis*
by (*simp add*: *rGamma-series-def*[*abs-def*] *fact-altdef pochhammer-Suc-setprod*
exp-def of-real-def[*symmetric*] *suminf-def sums-def*[*abs-def*])
qed (*insert rGamma-eq-zero-iff*[*of z*], *simp-all add*: *rGamma-series-nonpos-Ints-LIMSEQ*)

lemma *Gamma-series-LIMSEQ* [*tendsto-intros*]:
$Gamma\text{-}series\ z \longrightarrow Gamma\ z$
proof (*cases* $z \in \mathbb{Z}_{\leq 0}$)
case *False*
hence $(\lambda n.\ inverse\ (rGamma\text{-}series\ z\ n)) \longrightarrow inverse\ (rGamma\ z)$
by (*intro tendsto-intros*) (*simp-all add*: *rGamma-eq-zero-iff*)
also have $(\lambda n.\ inverse\ (rGamma\text{-}series\ z\ n)) = Gamma\text{-}series\ z$

by (*simp add*: *rGamma-series-def Gamma-series-def*[*abs-def*])
finally show *?thesis* **by** (*simp add*: *Gamma-def*)
qed (*insert Gamma-eq-zero-iff*[*of z*], *simp-all add*: *Gamma-series-nonpos-Ints-LIMSEQ*)

lemma *Gamma-altdef*: *Gamma z* = *lim* (*Gamma-series z*)
using *Gamma-series-LIMSEQ*[*of z*] **by** (*simp add*: *limI*)

lemma *rGamma-1* [*simp*]: *rGamma 1* = *1*
proof −
have *A*: *eventually* (λ*n*. *rGamma-series 1 n* = *of-nat* (*Suc n*) / *of-nat n*) *sequentially*
using *eventually-gt-at-top*[*of 0*::*nat*]
by (*force elim*!: *eventually-mono simp*: *rGamma-series-def exp-of-real pochhammer-fact divide-simps pochhammer-rec*′ *dest*!: *pochhammer-eq-0-imp-nonpos-Int*)
have *rGamma-series 1* ⟶ *1* **by** (*subst tendsto-cong*[*OF A*]) (*rule LIMSEQ-Suc-n-over-n*)
moreover have *rGamma-series 1* ⟶ *rGamma 1* **by** (*rule tendsto-intros*)
ultimately show *?thesis* **by** (*intro LIMSEQ-unique*)
qed

lemma *rGamma-plus1*: *z* ∗ *rGamma* (*z* + *1*) = *rGamma z*
proof −
let *?f* = λ*n*. (*z* + *1*) ∗ *inverse* (*of-nat n*) + *1*
have *eventually* (λ*n*. *?f n* ∗ *rGamma-series z n* = *z* ∗ *rGamma-series* (*z* + *1*) *n*) *sequentially*
using *eventually-gt-at-top*[*of 0*::*nat*]
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
hence *z* ∗ *rGamma-series* (*z* + *1*) *n* = *inverse* (*of-nat n*) ∗ *pochhammer z* (*Suc* (*Suc n*)) / (*fact n* ∗ *exp* (*z* ∗ *of-real* (*ln* (*of-nat n*))))
by (*subst pochhammer-rec*) (*simp add*: *rGamma-series-def field-simps exp-add exp-of-real*)
also from *n* **have** ... = *?f n* ∗ *rGamma-series z n*
by (*subst pochhammer-rec*′) (*simp-all add*: *divide-simps rGamma-series-def add-ac*)
finally show *?f n* ∗ *rGamma-series z n* = *z* ∗ *rGamma-series* (*z* + *1*) *n* **..**
qed
moreover have (λ*n*. *?f n* ∗ *rGamma-series z n*) ⟶ ((*z*+*1*) ∗ *0* + *1*) ∗ *rGamma z*
by (*intro tendsto-intros lim-inverse-n*)
hence (λ*n*. *?f n* ∗ *rGamma-series z n*) ⟶ *rGamma z* **by** *simp*
ultimately have (λ*n*. *z* ∗ *rGamma-series* (*z* + *1*) *n*) ⟶ *rGamma z*
by (*rule Lim-transform-eventually*)
moreover have (λ*n*. *z* ∗ *rGamma-series* (*z* + *1*) *n*) ⟶ *z* ∗ *rGamma* (*z* + *1*)
by (*intro tendsto-intros*)
ultimately show *z* ∗ *rGamma* (*z* + *1*) = *rGamma z* **using** *LIMSEQ-unique* **by** *blast*
qed

lemma *pochhammer-rGamma*: *rGamma z = pochhammer z n * rGamma (z + of-nat n)*
proof (*induction n arbitrary*: *z*)
case (*Suc n z*)
have *rGamma z = pochhammer z n * rGamma (z + of-nat n)* **by** (*rule Suc.IH*)
also note *rGamma-plus1* [*symmetric*]
finally show *?case* **by** (*simp add*: *add-ac pochhammer-rec'*)
qed *simp-all*

lemma *Gamma-plus1*: *z ∉ ℤ≤0 ⟹ Gamma (z + 1) = z * Gamma z*
using *rGamma-plus1*[*of z*] **by** (*simp add*: *rGamma-inverse-Gamma field-simps Gamma-eq-zero-iff*)

lemma *pochhammer-Gamma*: *z ∉ ℤ≤0 ⟹ pochhammer z n = Gamma (z + of-nat n) / Gamma z*
using *pochhammer-rGamma*[*of z*]
by (*simp add*: *rGamma-inverse-Gamma Gamma-eq-zero-iff field-simps*)

lemma *Gamma-0* [*simp*]: *Gamma 0 = 0*
and *rGamma-0* [*simp*]: *rGamma 0 = 0*
and *Gamma-neg-1* [*simp*]: *Gamma (− 1) = 0*
and *rGamma-neg-1* [*simp*]: *rGamma (− 1) = 0*
and *Gamma-neg-numeral* [*simp*]: *Gamma (− numeral n) = 0*
and *rGamma-neg-numeral* [*simp*]: *rGamma (− numeral n) = 0*
and *Gamma-neg-of-nat* [*simp*]: *Gamma (− of-nat m) = 0*
and *rGamma-neg-of-nat* [*simp*]: *rGamma (− of-nat m) = 0*
by (*simp-all add*: *rGamma-eq-zero-iff Gamma-eq-zero-iff*)

lemma *Gamma-1* [*simp*]: *Gamma 1 = 1* **unfolding** *Gamma-def* **by** *simp*

lemma *Gamma-fact*: *Gamma (of-nat (Suc n)) = fact n*
by (*simp add*: *pochhammer-fact pochhammer-Gamma of-nat-in-nonpos-Ints-iff*)

lemma *Gamma-numeral*: *Gamma (numeral n) = fact (pred-numeral n)*
by (*subst of-nat-numeral*[*symmetric*], *subst numeral-eq-Suc*, *subst Gamma-fact*) (*rule refl*)

lemma *Gamma-of-int*: *Gamma (of-int n) = (if n > 0 then fact (nat (n − 1)) else 0)*
proof (*cases n > 0*)
case *True*
hence *Gamma (of-int n) = Gamma (of-nat (Suc (nat (n − 1))))* **by** (*subst of-nat-Suc*) *simp-all*
with *True* **show** *?thesis* **by** (*subst* (*asm*) *Gamma-fact*) *simp*
qed (*simp-all add*: *Gamma-eq-zero-iff nonpos-Ints-of-int*)

lemma *rGamma-of-int*: *rGamma (of-int n) = (if n > 0 then inverse (fact (nat (n − 1))) else 0)*

by (*simp add*: *Gamma-of-int rGamma-inverse-Gamma*)

lemma *Gamma-seriesI*:
assumes (λ*n*. *g n* / *Gamma-series z n*) ⟶ *1*
shows *g* ⟶ *Gamma z*
proof (*rule Lim-transform-eventually*)
have *1/2* > (*0*::*real*) **by** *simp*
from *tendstoD*[*OF assms*, *OF this*]
show *eventually* (λ*n*. *g n* / *Gamma-series z n* ∗ *Gamma-series z n* = *g n*) *sequentially*
by (*force elim*!: *eventually-mono simp*: *dist-real-def dist-0-norm*)
from *assms* **have** (λ*n*. *g n* / *Gamma-series z n* ∗ *Gamma-series z n*) ⟶ *1* ∗ *Gamma z*
by (*intro tendsto-intros*)
thus (λ*n*. *g n* / *Gamma-series z n* ∗ *Gamma-series z n*) ⟶ *Gamma z* **by** *simp*
qed

lemma *Gamma-seriesI'*:
assumes *f* ⟶ *rGamma z*
assumes (λ*n*. *g n* ∗ *f n*) ⟶ *1*
assumes *z* ∉ $\mathbb{Z}_{\leq 0}$
shows *g* ⟶ *Gamma z*
proof (*rule Lim-transform-eventually*)
have *1/2* > (*0*::*real*) **by** *simp*
from *tendstoD*[*OF assms*(*2*), *OF this*] **show** *eventually* (λ*n*. *g n* ∗ *f n* / *f n* = *g n*) *sequentially*
by (*force elim*!: *eventually-mono simp*: *dist-real-def dist-0-norm*)
from *assms* **have** (λ*n*. *g n* ∗ *f n* / *f n*) ⟶ *1* / *rGamma z*
by (*intro tendsto-divide assms*) (*simp-all add*: *rGamma-eq-zero-iff*)
thus (λ*n*. *g n* ∗ *f n* / *f n*) ⟶ *Gamma z* **by** (*simp add*: *Gamma-def divide-inverse*)
qed

lemma *Gamma-series'-LIMSEQ*: *Gamma-series'* *z* ⟶ *Gamma z*
by (*cases z* ∈ $\mathbb{Z}_{\leq 0}$) (*simp-all add*: *Gamma-nonpos-Int Gamma-seriesI*[*OF Gamma-series-Gamma-series'*] *Gamma-series'-nonpos-Ints-LIMSEQ*[*of z*])

57.4 Differentiability

lemma *has-field-derivative-rGamma-no-nonpos-int*:
assumes *z* ∉ $\mathbb{Z}_{\leq 0}$
shows (*rGamma has-field-derivative* −*rGamma z* ∗ *Digamma z*) (*at z within A*)
proof (*rule has-field-derivative-at-within*)
from *assms* **have** *z* ≠ − *of-nat n* **for** *n* **by** *auto*
from *differentiable-rGamma-aux1*[*OF this*]
show (*rGamma has-field-derivative* −*rGamma z* ∗ *Digamma z*) (*at z*)
unfolding *Digamma-def suminf-def sums-def*[*abs-def*] *has-field-derivative-def has-derivative-def netlimit-at*

by (*simp add*: *Let-def bounded-linear-mult-right mult-ac of-real-def* [*symmetric*])
qed

lemma *has-field-derivative-rGamma-nonpos-int*:
(*rGamma has-field-derivative* (−*1*) ^*n* ∗ *fact n*) (*at* (− *of-nat n*) *within A*)
apply (*rule has-field-derivative-at-within*)
using *differentiable-rGamma-aux2*[*of n*]
unfolding *Let-def has-field-derivative-def has-derivative-def netlimit-at*
by (*simp only*: *bounded-linear-mult-right mult-ac of-real-def* [*symmetric*] *fact-altdef*)

lemma *has-field-derivative-rGamma* [*derivative-intros*]:
(*rGamma has-field-derivative* (*if z* ∈ $\mathbb{Z}_{\leq 0}$ *then* (−*1*) ^(*nat* ⌊*norm z*⌋) ∗ *fact* (*nat* ⌊*norm z*⌋)
else −*rGamma z* ∗ *Digamma z*)) (*at z within A*)
using *has-field-derivative-rGamma-no-nonpos-int*[*of z A*]
has-field-derivative-rGamma-nonpos-int[*of nat* ⌊*norm z*⌋ *A*]
by (*auto elim*!: *nonpos-Ints-cases'*)

declare *has-field-derivative-rGamma-no-nonpos-int* [*THEN DERIV-chain2*, *derivative-intros*]
declare *has-field-derivative-rGamma* [*THEN DERIV-chain2*, *derivative-intros*]
declare *has-field-derivative-rGamma-nonpos-int* [*derivative-intros*]
declare *has-field-derivative-rGamma-no-nonpos-int* [*derivative-intros*]
declare *has-field-derivative-rGamma* [*derivative-intros*]

lemma *has-field-derivative-Gamma* [*derivative-intros*]:
z ∉ $\mathbb{Z}_{\leq 0}$ ⟹ (*Gamma has-field-derivative Gamma z* ∗ *Digamma z*) (*at z within A*)
unfolding *Gamma-def* [*abs-def*]
by (*fastforce intro*!: *derivative-eq-intros simp*: *rGamma-eq-zero-iff*)

declare *has-field-derivative-Gamma*[*THEN DERIV-chain2*, *derivative-intros*]

hide-fact *rGamma-eq-zero-iff-aux differentiable-rGamma-aux1 differentiable-rGamma-aux2 differentiable-rGamma-aux2 rGamma-series-aux Gamma-class.rGamma-eq-zero-iff-aux*

57.5 Continuity

lemma *continuous-on-rGamma* [*continuous-intros*]: *continuous-on A rGamma*
by (*rule DERIV-continuous-on has-field-derivative-rGamma*)+

lemma *continuous-on-Gamma* [*continuous-intros*]: *A* ∩ $\mathbb{Z}_{\leq 0}$ = {} ⟹ *continuous-on A Gamma*
by (*rule DERIV-continuous-on has-field-derivative-Gamma*)+ *blast*

lemma *isCont-rGamma* [*continuous-intros*]:
isCont f z ⟹ *isCont* (λ*x*. *rGamma* (*f x*)) *z*
by (*rule isCont-o2*[*OF - DERIV-isCont*[*OF has-field-derivative-rGamma*]])

lemma *isCont-Gamma* [*continuous-intros*]:
isCont f z $\Longrightarrow$ *f z* $\notin \mathbb{Z}_{\leq 0}$ $\Longrightarrow$ *isCont* (λx. *Gamma* (*f x*)) *z*
by (*rule isCont-o2*[*OF - DERIV-isCont*[*OF has-field-derivative-Gamma*]])

The complex Gamma function

instantiation *complex* :: *Gamma*
begin

definition *rGamma-complex* :: *complex* $\Rightarrow$ *complex* **where**
rGamma-complex z = *lim* (*rGamma-series z*)

lemma *rGamma-series-complex-converges*:
convergent (*rGamma-series* (*z* :: *complex*)) (**is** *?thesis1*)
and *rGamma-complex-altdef*:
rGamma z = (*if z* $\in \mathbb{Z}_{\leq 0}$ *then 0 else exp* (−*ln-Gamma z*)) (**is** *?thesis2*)
proof −
have *?thesis1* ∧ *?thesis2*
proof (*cases z* $\in \mathbb{Z}_{\leq 0}$)
case *False*
have *rGamma-series z* $\longrightarrow$ *exp* (− *ln-Gamma z*)
proof (*rule Lim-transform-eventually*)
from *ln-Gamma-series-complex-converges'*[*OF False*] **guess** *d* **by** (*elim exE conjE*)
from *this*(*1*) *uniformly-convergent-imp-convergent*[*OF this*(*2*), *of z*]
have *ln-Gamma-series z* $\longrightarrow$ *lim* (*ln-Gamma-series z*) **by** (*simp add*: *convergent-LIMSEQ-iff*)
thus (λn. *exp* (−*ln-Gamma-series z n*)) $\longrightarrow$ *exp* (− *ln-Gamma z*)
unfolding *convergent-def ln-Gamma-def* **by** (*intro tendsto-exp tendsto-minus*)
from *eventually-gt-at-top*[*of 0*::*nat*] *exp-ln-Gamma-series-complex False*
show *eventually* (λn. *exp* (−*ln-Gamma-series z n*) = *rGamma-series z n*) *sequentially*
by (*force elim*!: *eventually-mono simp*: *exp-minus Gamma-series-def rGamma-series-def*)
qed
with *False* **show** *?thesis*
by (*auto simp*: *convergent-def rGamma-complex-def intro*!: *limI*)
next
case *True*
then obtain *k* **where** *z* = − *of-nat k* **by** (*erule nonpos-Ints-cases'*)
also have *rGamma-series* ... $\longrightarrow$ *0*
by (*subst tendsto-cong*[*OF rGamma-series-minus-of-nat*]) (*simp-all add*: *convergent-const*)
finally show *?thesis* **using** *True*
by (*auto simp*: *rGamma-complex-def convergent-def intro*!: *limI*)
qed
thus *?thesis1 ?thesis2* **by** *blast*+
qed

context

begin

private lemma *rGamma-complex-plus1*: $z * rGamma\ (z + 1) = rGamma\ (z :: complex)$
proof −
let *?f* = $\lambda n.\ (z + 1) * inverse\ (of\text{-}nat\ n) + 1$
have *eventually* $(\lambda n.\ ?f\ n * rGamma\text{-}series\ z\ n = z * rGamma\text{-}series\ (z + 1)\ n)$ *sequentially*
using *eventually-gt-at-top*[*of* 0::*nat*]
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: $n > 0$
hence $z * rGamma\text{-}series\ (z + 1)\ n = inverse\ (of\text{-}nat\ n) *$
$pochhammer\ z\ (Suc\ (Suc\ n)) / (fact\ n * exp\ (z * of\text{-}real\ (ln\ (of\text{-}nat\ n))))$
by (*subst pochhammer-rec*) (*simp add*: *rGamma-series-def field-simps exp-add exp-of-real*)
also from *n* **have** $\ldots = ?f\ n * rGamma\text{-}series\ z\ n$
by (*subst pochhammer-rec'*) (*simp-all add*: *divide-simps rGamma-series-def add-ac*)
finally show $?f\ n * rGamma\text{-}series\ z\ n = z * rGamma\text{-}series\ (z + 1)\ n$ **..**
qed
moreover have $(\lambda n.\ ?f\ n * rGamma\text{-}series\ z\ n) \longrightarrow ((z+1) * 0 + 1) * rGamma\ z$
using *rGamma-series-complex-converges*
by (*intro tendsto-intros lim-inverse-n*)
(*simp-all add*: *convergent-LIMSEQ-iff rGamma-complex-def*)
hence $(\lambda n.\ ?f\ n * rGamma\text{-}series\ z\ n) \longrightarrow rGamma\ z$ **by** *simp*
ultimately have $(\lambda n.\ z * rGamma\text{-}series\ (z + 1)\ n) \longrightarrow rGamma\ z$
by (*rule Lim-transform-eventually*)
moreover have $(\lambda n.\ z * rGamma\text{-}series\ (z + 1)\ n) \longrightarrow z * rGamma\ (z + 1)$
using *rGamma-series-complex-converges*
by (*auto intro*!: *tendsto-mult simp*: *rGamma-complex-def convergent-LIMSEQ-iff*)
ultimately show $z * rGamma\ (z + 1) = rGamma\ z$ **using** *LIMSEQ-unique* **by** *blast*
qed

private lemma *has-field-derivative-rGamma-complex-no-nonpos-Int*:
assumes $(z :: complex) \notin \mathbb{Z}_{\leq 0}$
shows $(rGamma\ has\text{-}field\text{-}derivative - rGamma\ z * Digamma\ z)\ (at\ z)$
proof −
have *diff*: $(rGamma\ has\text{-}field\text{-}derivative - rGamma\ z * Digamma\ z)\ (at\ z)$ **if** $Re\ z > 0$ **for** *z*
proof (*subst DERIV-cong-ev*[*OF refl - refl*])
from *that* **have** *eventually* $(\lambda t.\ t \in ball\ z\ (Re\ z/2))\ (nhds\ z)$
by (*intro eventually-nhds-in-nhd*) *simp-all*
thus *eventually* $(\lambda t.\ rGamma\ t = exp\ (-\ ln\text{-}Gamma\ t))\ (nhds\ z)$
using *no-nonpos-Int-in-ball-complex*[*OF that*]
by (*auto elim*!: *eventually-mono simp*: *rGamma-complex-altdef*)

```
  next
    have z ∉ ℝ_{≤0} using that by (simp add: complex-nonpos-Reals-iff)
    with that show ((λt. exp (− ln-Gamma t)) has-field-derivative (−rGamma z
* Digamma z)) (at z)
    by (force elim!: nonpos-Ints-cases intro!: derivative-eq-intros simp: rGamma-complex-altdef)
  qed

  from assms show (rGamma has-field-derivative − rGamma z * Digamma z) (at
z)
  proof (induction nat ⌊1 − Re z⌋ arbitrary: z)
    case (Suc n z)
    from Suc.prems have z: z ≠ 0 by auto
    from Suc.hyps have n = nat ⌊− Re z⌋ by linarith
    hence A: n = nat ⌊1 − Re (z + 1)⌋ by simp
   from Suc.prems have B: z + 1 ∉ ℤ_{≤0} by (force dest: plus-one-in-nonpos-Ints-imp)

    have ((λz. z * (rGamma ∘ (λz. z + 1)) z) has-field-derivative
      −rGamma (z + 1) * (Digamma (z + 1) * z − 1)) (at z)
    by (rule derivative-eq-intros DERIV-chain Suc refl A B)+ (simp add: algebra-simps)
    also have (λz. z * (rGamma ∘ (λz. z + 1 :: complex)) z) = rGamma
      by (simp add: rGamma-complex-plus1)
    also from z have Digamma (z + 1) * z − 1 = z * Digamma z
      by (subst Digamma-plus1) (simp-all add: field-simps)
    also have −rGamma (z + 1) * (z * Digamma z) = −rGamma z * Digamma
z
      by (simp add: rGamma-complex-plus1[of z, symmetric])
    finally show ?case .
  qed (intro diff, simp)
qed

private lemma rGamma-complex-1: rGamma (1 :: complex) = 1
proof −
   have A: eventually (λn. rGamma-series 1 n = of-nat (Suc n) / of-nat n)
sequentially
    using eventually-gt-at-top[of 0::nat]
   by (force elim!: eventually-mono simp: rGamma-series-def exp-of-real pochhammer-fact
            divide-simps pochhammer-rec' dest!: pochhammer-eq-0-imp-nonpos-Int)
 have rGamma-series 1 ⟶ 1 by (subst tendsto-cong[OF A]) (rule LIMSEQ-Suc-n-over-n)
  thus rGamma 1 = (1 :: complex) unfolding rGamma-complex-def by (rule
limI)
qed

private lemma has-field-derivative-rGamma-complex-nonpos-Int:
  (rGamma has-field-derivative (−1) ^n * fact n) (at (− of-nat n :: complex))
proof (induction n)
  case 0
  have A: (0::complex) + 1 ∉ ℤ_{≤0} by simp
  have ((λz. z * (rGamma ∘ (λz. z + 1 :: complex)) z) has-field-derivative 1) (at
0)
```

by (*rule derivative-eq-intros DERIV-chain refl*
has-field-derivative-rGamma-complex-no-nonpos-Int A)+ (*simp add*: *rGamma-complex-1*)
thus *?case* **by** (*simp add*: *rGamma-complex-plus1*)
next
case (*Suc n*)
hence *A*: (*rGamma has-field-derivative* (−*1*) ^ *n* ∗ *fact n*)
(*at* (− *of-nat* (*Suc n*) + *1* :: *complex*)) **by** *simp*
have ((λ*z*. *z* ∗ (*rGamma* ∘ (λ*z*. *z* + *1* :: *complex*)) *z*) *has-field-derivative*
(− *1*) ^ *Suc n* ∗ *fact* (*Suc n*)) (*at* (− *of-nat* (*Suc n*)))
by (*rule derivative-eq-intros refl A DERIV-chain*)+
(*simp add*: *algebra-simps rGamma-complex-altdef*)
thus *?case* **by** (*simp add*: *rGamma-complex-plus1*)
qed

instance proof
fix *z* :: *complex* **show** (*rGamma z* = *0*) ⟷ (∃ *n*. *z* = − *of-nat n*)
by (*auto simp*: *rGamma-complex-altdef elim*!: *nonpos-Ints-cases*′)
next
fix *z* :: *complex* **assume** ⋀*n*. *z* ≠ − *of-nat n*
hence *z* ∉ ℤ≤0 **by** (*auto elim*!: *nonpos-Ints-cases*′)
from *has-field-derivative-rGamma-complex-no-nonpos-Int*[*OF this*]
show *let d* = (*THE d*. (λ*n*. ∑ *k*<*n*. *inverse* (*of-nat* (*Suc k*)) − *inverse* (*z* + *of-nat k*))
⟶ *d*) − *euler-mascheroni* ∗*R* *1* *in* (λ*y*. (*rGamma y* − *rGamma z* +
rGamma z ∗ *d* ∗ (*y* − *z*)) /*R* *cmod* (*y* − *z*)) −*z*→ *0*
by (*simp add*: *has-field-derivative-def has-derivative-def Digamma-def sums-def* [*abs-def*]
netlimit-at of-real-def[*symmetric*] *suminf-def*)
next
fix *n* :: *nat*
from *has-field-derivative-rGamma-complex-nonpos-Int*[*of n*]
show *let z* = − *of-nat n in* (λ*y*. (*rGamma y* − *rGamma z* − (− *1*) ^ *n* ∗ *setprod of-nat* {*1*..*n*} ∗
(*y* − *z*)) /*R* *cmod* (*y* − *z*)) −*z*→ *0*
by (*simp add*: *has-field-derivative-def has-derivative-def fact-altdef netlimit-at Let-def*)
next
fix *z* :: *complex*
from *rGamma-series-complex-converges*[*of z*] **have** *rGamma-series z* ⟶ *rGamma z*
by (*simp add*: *convergent-LIMSEQ-iff rGamma-complex-def*)
thus *let fact*′ = λ*n*. *setprod of-nat* {*1*..*n*};
exp = λ*x*. *THE e*. (λ*n*. ∑ *k*<*n*. *x* ^ *k* /*R* *fact k*) ⟶ *e*;
pochhammer′ = λ*a n*. ∏ *n* = *0*..*n*. *a* + *of-nat n*
in (λ*n*. *pochhammer*′ *z n* / (*fact*′ *n* ∗ *exp* (*z* ∗ *ln* (*real-of-nat n*) ∗*R* *1*)))
⟶ *rGamma z*
by (*simp add*: *fact-altdef pochhammer-Suc-setprod rGamma-series-def* [*abs-def*]

exp-def
of-real-def [*symmetric*] *suminf-def sums-def* [*abs-def*])
qed

end
end

lemma *Gamma-complex-altdef*:
Gamma z = (*if z* ∈ $\mathbb{Z}_{\leq 0}$ *then 0 else exp* (*ln-Gamma* (*z* :: *complex*)))
unfolding *Gamma-def rGamma-complex-altdef* **by** (*simp add*: *exp-minus*)

lemma *cnj-rGamma*: *cnj* (*rGamma z*) = *rGamma* (*cnj z*)
proof −
have *rGamma-series* (*cnj z*) = (λ*n*. *cnj* (*rGamma-series z n*))
by (*intro ext*) (*simp-all add*: *rGamma-series-def exp-cnj*)
also have ... ⟶ *cnj* (*rGamma z*) **by** (*intro tendsto-cnj tendsto-intros*)
finally show *?thesis* **unfolding** *rGamma-complex-def* **by** (*intro sym*[*OF limI*])
qed

lemma *cnj-Gamma*: *cnj* (*Gamma z*) = *Gamma* (*cnj z*)
unfolding *Gamma-def* **by** (*simp add*: *cnj-rGamma*)

lemma *Gamma-complex-real*:
z ∈ ℝ ⟹ *Gamma z* ∈ (ℝ :: *complex set*) **and** *rGamma-complex-real*: *z* ∈ ℝ
⟹ *rGamma z* ∈ ℝ
by (*simp-all add*: *Reals-cnj-iff cnj-Gamma cnj-rGamma*)

lemma *complex-differentiable-rGamma*: *rGamma complex-differentiable* (*at z within*
A)
using *has-field-derivative-rGamma*[*of z*] **unfolding** *complex-differentiable-def* **by**
blast

lemma *holomorphic-on-rGamma*: *rGamma holomorphic-on A*
unfolding *holomorphic-on-def* **by** (*auto intro*!: *complex-differentiable-rGamma*)

lemma *analytic-on-rGamma*: *rGamma analytic-on A*
unfolding *analytic-on-def* **by** (*auto intro*!: *exI*[*of - 1*] *holomorphic-on-rGamma*)

lemma *complex-differentiable-Gamma*: *z* ∉ $\mathbb{Z}_{\leq 0}$ ⟹ *Gamma complex-differentiable*
(*at z within A*)
using *has-field-derivative-Gamma*[*of z*] **unfolding** *complex-differentiable-def* **by**
auto

lemma *holomorphic-on-Gamma*: *A* ∩ $\mathbb{Z}_{\leq 0}$ = {} ⟹ *Gamma holomorphic-on A*
unfolding *holomorphic-on-def* **by** (*auto intro*!: *complex-differentiable-Gamma*)

lemma *analytic-on-Gamma*: *A* ∩ $\mathbb{Z}_{\leq 0}$ = {} ⟹ *Gamma analytic-on A*

by (*rule analytic-on-subset*[*of - UNIV − $\mathbb{Z}_{\leq 0}$*], *subst analytic-on-open*)
(*auto intro*!: *holomorphic-on-Gamma*)

lemma *has-field-derivative-rGamma-complex'* [*derivative-intros*]:
(*rGamma has-field-derivative* (*if z ∈ $\mathbb{Z}_{\leq 0}$ then* (*−1*) ^ (*nat* ⌊*−Re z*⌋) * *fact* (*nat* ⌊*−Re z*⌋) *else*
−rGamma z * *Digamma z*)) (*at z within A*)
using *has-field-derivative-rGamma*[*of z*] **by** (*auto elim*!: *nonpos-Ints-cases'*)

declare *has-field-derivative-rGamma-complex'*[*THEN DERIV-chain2*, *derivative-intros*]

lemma *complex-differentiable-Polygamma*:
z ∉ $\mathbb{Z}_{\leq 0}$ ⟹ Polygamma n complex-differentiable (*at z within A*)
using *has-field-derivative-Polygamma*[*of z n*] **unfolding** *complex-differentiable-def*
by *auto*

lemma *holomorphic-on-Polygamma*: *A ∩ $\mathbb{Z}_{\leq 0}$ = {} ⟹ Polygamma n holomorphic-on A*
unfolding *holomorphic-on-def* **by** (*auto intro*!: *complex-differentiable-Polygamma*)

lemma *analytic-on-Polygamma*: *A ∩ $\mathbb{Z}_{\leq 0}$ = {} ⟹ Polygamma n analytic-on A*
by (*rule analytic-on-subset*[*of - UNIV − $\mathbb{Z}_{\leq 0}$*], *subst analytic-on-open*)
(*auto intro*!: *holomorphic-on-Polygamma*)

The real Gamma function

lemma *rGamma-series-real*:
eventually (*λn. rGamma-series x n = Re* (*rGamma-series* (*of-real x*) *n*)) *sequentially*
using *eventually-gt-at-top*[*of 0 :: nat*]
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: *n > 0*
have *Re* (*rGamma-series* (*of-real x*) *n*) =
Re (*of-real* (*pochhammer x* (*Suc n*)) / (*fact n * exp* (*of-real* (*x * ln* (*real-of-nat n*)))))
using *n* **by** (*simp add*: *rGamma-series-def powr-def Ln-of-nat pochhammer-of-real*)
also from *n* **have** *... = Re* (*of-real* ((*pochhammer x* (*Suc n*)) /
(*fact n* * (*exp* (*x * ln* (*real-of-nat n*)))))))
by (*subst exp-of-real*) *simp*
also from *n* **have** *... = rGamma-series x n*
by (*subst Re-complex-of-real*) (*simp add*: *rGamma-series-def powr-def*)
finally show *rGamma-series x n = Re* (*rGamma-series* (*of-real x*) *n*) **..**
qed

instantiation *real* :: *Gamma*
begin

definition *rGamma-real x = Re* (*rGamma* (*of-real x* :: *complex*))

instance proof

fix x :: *real*
have *rGamma x = Re (rGamma (of-real x))* **by** (*simp add*: *rGamma-real-def*)
also have *of-real ... = rGamma (of-real x :: complex)*
by (*intro of-real-Re rGamma-complex-real*) *simp-all*
also have $\ldots = 0 \longleftrightarrow x \in \mathbb{Z}_{\leq 0}$ **by** (*simp add*: *rGamma-eq-zero-iff of-real-in-nonpos-Ints-iff*)
also have $\ldots \longleftrightarrow (\exists n.\ x = -\ of\text{-}nat\ n)$ **by** (*auto elim*!: *nonpos-Ints-cases'*)
finally show *(rGamma x) = 0* $\longleftrightarrow$ $(\exists n.\ x = -\ real\text{-}of\text{-}nat\ n)$ **by** *simp*
next
fix x :: *real* **assume** $\bigwedge n.\ x \neq -\ of\text{-}nat\ n$
hence *complex-of-real x* $\notin \mathbb{Z}_{\leq 0}$
by (*subst of-real-in-nonpos-Ints-iff*) (*auto elim*!: *nonpos-Ints-cases'*)
moreover from *this* **have** *x ≠ 0* **by** *auto*
ultimately have (*rGamma has-field-derivative − rGamma x ∗ Digamma x*) (*at x*)
by (*fastforce intro*!: *derivative-eq-intros has-vector-derivative-real-complex*
simp: *Polygamma-of-real rGamma-real-def* [*abs-def*])
thus *let d = (THE d. (λn. ∑k<n. inverse (of-nat (Suc k)) − inverse (x + of-nat k)) ⟶ d) − euler-mascheroni ∗R 1 in (λy. (rGamma y − rGamma x + rGamma x ∗ d ∗ (y − x)) /R norm (y − x)) −x→ 0*
by (*simp add*: *has-field-derivative-def has-derivative-def Digamma-def sums-def* [*abs-def*]
netlimit-at of-real-def [*symmetric*] *suminf-def*)
next
fix n :: *nat*
have (*rGamma has-field-derivative (−1) ^n ∗ fact n*) (*at (− of-nat n :: real)*)
by (*fastforce intro*!: *derivative-eq-intros has-vector-derivative-real-complex*
simp: *Polygamma-of-real rGamma-real-def* [*abs-def*])
thus *let x = − of-nat n in (λy. (rGamma y − rGamma x − (− 1) ^ n ∗ setprod of-nat {1..n} ∗ (y − x)) /R norm (y − x)) −x::real→ 0*
by (*simp add*: *has-field-derivative-def has-derivative-def fact-altdef netlimit-at Let-def*)
next
fix x :: *real*
have *rGamma-series x ⟶ rGamma x*
proof (*rule Lim-transform-eventually*)
show *(λn. Re (rGamma-series (of-real x) n)) ⟶ rGamma x* **unfolding** *rGamma-real-def*
by (*intro tendsto-intros*)
qed (*insert rGamma-series-real*, *simp add*: *eq-commute*)
thus *let fact' = λn. setprod of-nat {1..n}; exp = λx. THE e. (λn. ∑k<n. x ^ k /R fact k) ⟶ e; pochhammer' = λa n. ∏n = 0..n. a + of-nat n in (λn. pochhammer' x n / (fact' n ∗ exp (x ∗ ln (real-of-nat n)) ∗R 1))) ⟶ rGamma x*
by (*simp add*: *fact-altdef pochhammer-Suc-setprod rGamma-series-def* [*abs-def*] *exp-def*

of-real-def [*symmetric*] *suminf-def sums-def* [*abs-def*])
qed

end

lemma *rGamma-complex-of-real*: *rGamma* (*complex-of-real x*) = *complex-of-real* (*rGamma x*)
unfolding *rGamma-real-def* **using** *rGamma-complex-real* **by** *simp*

lemma *Gamma-complex-of-real*: *Gamma* (*complex-of-real x*) = *complex-of-real* (*Gamma x*)
unfolding *Gamma-def* **by** (*simp add*: *rGamma-complex-of-real*)

lemma *rGamma-real-altdef*: *rGamma x* = *lim* (*rGamma-series* (*x* :: *real*))
by (*rule sym*, *rule limI*, *rule tendsto-intros*)

lemma *Gamma-real-altdef1*: *Gamma x* = *lim* (*Gamma-series* (*x* :: *real*))
by (*rule sym*, *rule limI*, *rule tendsto-intros*)

lemma *Gamma-real-altdef2*: *Gamma x* = *Re* (*Gamma* (*of-real x*))
using *rGamma-complex-real*[*OF Reals-of-real*[*of x*]]
by (*elim Reals-cases*)
(*simp only*: *Gamma-def rGamma-real-def of-real-inverse*[*symmetric*] *Re-complex-of-real*)

lemma *ln-Gamma-series-complex-of-real*:
x > *0* ⟹ *n* > *0* ⟹ *ln-Gamma-series* (*complex-of-real x*) *n* = *of-real* (*ln-Gamma-series x n*)
proof −
assume *xn*: *x* > *0 n* > *0*
have *Ln* (*complex-of-real x* / *of-nat k* + *1*) = *of-real* (*ln* (*x* / *of-nat k* + *1*)) **if** *k* ≥ *1* **for** *k*
using *that xn* **by** (*subst Ln-of-real* [*symmetric*]) (*auto intro*!: *add-nonneg-pos simp*: *field-simps*)
with *xn* **show** *?thesis* **by** (*simp add*: *ln-Gamma-series-def Ln-of-nat Ln-of-real*)
qed

lemma *ln-Gamma-real-converges*:
assumes (*x*::*real*) > *0*
shows *convergent* (*ln-Gamma-series x*)
proof −
have (λ*n*. *ln-Gamma-series* (*complex-of-real x*) *n*) ⟶ *ln-Gamma* (*of-real x*)
using *assms*
by (*intro ln-Gamma-complex-LIMSEQ*) (*auto simp*: *of-real-in-nonpos-Ints-iff*)
moreover from *eventually-gt-at-top*[*of 0*::*nat*]
have *eventually* (λ*n*. *complex-of-real* (*ln-Gamma-series x n*) =
ln-Gamma-series (*complex-of-real x*) *n*) *sequentially*
by *eventually-elim* (*simp add*: *ln-Gamma-series-complex-of-real assms*)
ultimately have (λ*n*. *complex-of-real* (*ln-Gamma-series x n*)) ⟶ *ln-Gamma*

(*of-real x*)
by (*subst tendsto-cong*) *assumption+*
from *tendsto-Re*[*OF this*] **show** *?thesis* **by** (*auto simp*: *convergent-def*)
qed

lemma *ln-Gamma-real-LIMSEQ*: (*x*::*real*) > *0* ⟹ *ln-Gamma-series x* ⟶ *ln-Gamma x*
using *ln-Gamma-real-converges*[*of x*] **unfolding** *ln-Gamma-def* **by** (*simp add*: *convergent-LIMSEQ-iff*)

lemma *ln-Gamma-complex-of-real*: *x* > *0* ⟹ *ln-Gamma* (*complex-of-real x*) = *of-real* (*ln-Gamma x*)
proof (*unfold ln-Gamma-def*, *rule limI*, *rule Lim-transform-eventually*)
assume *x*: *x* > *0*
show *eventually* (λ*n*. *of-real* (*ln-Gamma-series x n*) =
ln-Gamma-series (*complex-of-real x*) *n*) *sequentially*
using *eventually-gt-at-top*[*of 0*::*nat*]
by *eventually-elim* (*simp add*: *ln-Gamma-series-complex-of-real x*)
qed (*intro tendsto-of-real*, *insert ln-Gamma-real-LIMSEQ*[*of x*], *simp add*: *ln-Gamma-def*)

lemma *Gamma-real-pos-exp*: *x* > (*0* :: *real*) ⟹ *Gamma x* = *exp* (*ln-Gamma x*)
by (*auto simp*: *Gamma-real-altdef2 Gamma-complex-altdef of-real-in-nonpos-Ints-iff*
ln-Gamma-complex-of-real exp-of-real)

lemma *ln-Gamma-real-pos*: *x* > *0* ⟹ *ln-Gamma x* = *ln* (*Gamma x* :: *real*)
unfolding *Gamma-real-pos-exp* **by** *simp*

lemma *Gamma-real-pos*: *x* > (*0*::*real*) ⟹ *Gamma x* > *0*
by (*simp add*: *Gamma-real-pos-exp*)

lemma *has-field-derivative-ln-Gamma-real* [*derivative-intros*]:
assumes *x*: *x* > (*0*::*real*)
shows (*ln-Gamma has-field-derivative Digamma x*) (*at x*)
proof (*subst DERIV-cong-ev*[*OF refl - refl*])
from *assms* **show** ((*Re* ∘ *ln-Gamma* ∘ *complex-of-real*) *has-field-derivative Digamma x*) (*at x*)
by (*auto intro*!: *derivative-eq-intros has-vector-derivative-real-complex*
simp: *Polygamma-of-real o-def*)
from *eventually-nhds-in-nhd*[*of x* {*0*<..}] *assms*
show *eventually* (λ*y*. *ln-Gamma y* = (*Re* ∘ *ln-Gamma* ∘ *of-real*) *y*) (*nhds x*)
by (*auto elim*!: *eventually-mono simp*: *ln-Gamma-complex-of-real interior-open*)
qed

declare *has-field-derivative-ln-Gamma-real*[*THEN DERIV-chain2*, *derivative-intros*]

lemma *has-field-derivative-rGamma-real'* [*derivative-intros*]:
(*rGamma has-field-derivative* (*if x* ∈ $\mathbb{Z}_{\leq 0}$ *then* (−*1*) ^ (*nat* ⌊−*x*⌋) ∗ *fact* (*nat* ⌊−*x*⌋) *else*

```
      −rGamma x ∗ Digamma x)) (at x within A)
  using has-field-derivative-rGamma[of x] by (force elim!: nonpos-Ints-cases')

declare has-field-derivative-rGamma-real'[THEN DERIV-chain2, derivative-intros]

lemma Polygamma-real-odd-pos:
  assumes (x::real) ∉ ℤ≤0 odd n
  shows   Polygamma n x > 0
proof −
  from assms have x ≠ 0 by auto
  with assms show ?thesis
    unfolding Polygamma-def using Polygamma-converges'[of x Suc n]
    by (auto simp: zero-less-power-eq simp del: power-Suc
          dest: plus-of-nat-eq-0-imp intro!: mult-pos-pos suminf-pos)
qed

lemma Polygamma-real-even-neg:
  assumes (x::real) > 0 n > 0 even n
  shows   Polygamma n x < 0
  using assms unfolding Polygamma-def using Polygamma-converges'[of x Suc
n]
  by (auto intro!: mult-pos-pos suminf-pos)

lemma Polygamma-real-strict-mono:
  assumes x > 0 x < (y::real) even n
  shows   Polygamma n x < Polygamma n y
proof −
  have ∃ξ. x < ξ ∧ ξ < y ∧ Polygamma n y − Polygamma n x = (y − x) ∗
Polygamma (Suc n) ξ
    using assms by (intro MVT2 derivative-intros impI allI) (auto elim!: nonpos-Ints-cases)
  then guess ξ by (elim exE conjE) note ξ = this
  note ξ(3)
  also from ξ(1,2) assms have (y − x) ∗ Polygamma (Suc n) ξ > 0
    by (intro mult-pos-pos Polygamma-real-odd-pos) (auto elim!: nonpos-Ints-cases)
  finally show ?thesis by simp
qed

lemma Polygamma-real-strict-antimono:
  assumes x > 0 x < (y::real) odd n
  shows   Polygamma n x > Polygamma n y
proof −
  have ∃ξ. x < ξ ∧ ξ < y ∧ Polygamma n y − Polygamma n x = (y − x) ∗
Polygamma (Suc n) ξ
    using assms by (intro MVT2 derivative-intros impI allI) (auto elim!: nonpos-Ints-cases)
  then guess ξ by (elim exE conjE) note ξ = this
  note ξ(3)
  also from ξ(1,2) assms have (y − x) ∗ Polygamma (Suc n) ξ < 0
    by (intro mult-pos-neg Polygamma-real-even-neg) simp-all
  finally show ?thesis by simp
```

qed

lemma *Polygamma-real-mono*:
assumes $x > 0$ $x \leq (y::real)$ *even n*
shows *Polygamma n x* $\leq$ *Polygamma n y*
using *Polygamma-real-strict-mono*[*OF assms*(*1*) - *assms*(*3*), *of y*] *assms*(*2*)
by (*cases x* = *y*) *simp-all*

lemma *Digamma-real-ge-three-halves-pos*:
assumes $x \geq 3/2$
shows *Digamma* (*x* :: *real*) > *0*
proof −
have *0* < *Digamma* (*3*/*2* :: *real*) **by** (*fact Digamma-real-three-halves-pos*)
also from *assms* **have** ... $\leq$ *Digamma x* **by** (*intro Polygamma-real-mono*) *simp-all*
finally show *?thesis* **.**
qed

lemma *ln-Gamma-real-strict-mono*:
assumes $x \geq 3/2$ $x < y$
shows *ln-Gamma* (*x* :: *real*) < *ln-Gamma y*
proof −
have $\exists \xi.\ x < \xi \wedge \xi < y \wedge$ *ln-Gamma y* − *ln-Gamma x* = (*y* − *x*) ∗ *Digamma* ξ
using *assms* **by** (*intro MVT2 derivative-intros impI allI*) (*auto elim*!: *nonpos-Ints-cases*)
then guess ξ **by** (*elim exE conjE*) **note** ξ = *this*
note ξ(*3*)
also from ξ(*1*,*2*) *assms* **have** (*y* − *x*) ∗ *Digamma* ξ > *0*
by (*intro mult-pos-pos Digamma-real-ge-three-halves-pos*) *simp-all*
finally show *?thesis* **by** *simp*
qed

lemma *Gamma-real-strict-mono*:
assumes $x \geq 3/2$ $x < y$
shows *Gamma* (*x* :: *real*) < *Gamma y*
proof −
from *Gamma-real-pos-exp*[*of x*] *assms* **have** *Gamma x* = *exp* (*ln-Gamma x*) **by** *simp*
also have ... < *exp* (*ln-Gamma y*) **by** (*intro exp-less-mono ln-Gamma-real-strict-mono assms*)
also from *Gamma-real-pos-exp*[*of y*] *assms* **have** ... = *Gamma y* **by** *simp*
finally show *?thesis* **.**
qed

lemma *log-convex-Gamma-real*: *convex-on* {*0*<..} (*ln* ∘ *Gamma* :: *real* ⇒ *real*)
by (*rule convex-on-realI*[*of* - - *Digamma*])
(*auto intro*!: *derivative-eq-intros Polygamma-real-mono Gamma-real-pos*
simp: *o-def Gamma-eq-zero-iff elim*!: *nonpos-Ints-cases'*)

57.6 Beta function

definition *Beta* **where** *Beta a b = Gamma a * Gamma b / Gamma (a + b)*

lemma *Beta-altdef*: *Beta a b = Gamma a * Gamma b * rGamma (a + b)*
by (*simp add*: *inverse-eq-divide Beta-def Gamma-def*)

lemma *Beta-commute*: *Beta a b = Beta b a*
unfolding *Beta-def* **by** (*simp add*: *ac-simps*)

lemma *has-field-derivative-Beta1* [*derivative-intros*]:
assumes $x \notin \mathbb{Z}_{\leq 0}$ $x + y \notin \mathbb{Z}_{\leq 0}$
shows ((λ*x*. *Beta x y*) *has-field-derivative* (*Beta x y* * (*Digamma x* − *Digamma* (*x* + *y*))))
(*at x within A*) **unfolding** *Beta-altdef*
by (*rule DERIV-cong*, (*rule derivative-intros assms*)+) (*simp add*: *algebra-simps*)

lemma *has-field-derivative-Beta2* [*derivative-intros*]:
assumes $y \notin \mathbb{Z}_{\leq 0}$ $x + y \notin \mathbb{Z}_{\leq 0}$
shows ((λ*y*. *Beta x y*) *has-field-derivative* (*Beta x y* * (*Digamma y* − *Digamma* (*x* + *y*))))
(*at y within A*)
using *has-field-derivative-Beta1*[*of y x A*] *assms* **by** (*simp add*: *Beta-commute add-ac*)

lemma *Beta-plus1-plus1*:
assumes $x \notin \mathbb{Z}_{\leq 0}$ $y \notin \mathbb{Z}_{\leq 0}$
shows *Beta* (*x* + *1*) *y* + *Beta x* (*y* + *1*) = *Beta x y*
proof −
have *Beta* (*x* + *1*) *y* + *Beta x* (*y* + *1*) =
(*Gamma* (*x* + *1*) * *Gamma y* + *Gamma x* * *Gamma* (*y* + *1*)) * *rGamma* ((*x* + *y*) + *1*)
by (*simp add*: *Beta-altdef add-divide-distrib algebra-simps*)
also have ... = (*Gamma x* * *Gamma y*) * ((*x* + *y*) * *rGamma* ((*x* + *y*) + *1*))
by (*subst assms*[*THEN Gamma-plus1*])+ (*simp add*: *algebra-simps*)
also from *assms* **have** ... = *Beta x y* **unfolding** *Beta-altdef* **by** (*subst rGamma-plus1*) *simp*
finally show *?thesis* **.**
qed

lemma *Beta-plus1-left*:
assumes $x \notin \mathbb{Z}_{\leq 0}$ $y \notin \mathbb{Z}_{\leq 0}$
shows (*x* + *y*) * *Beta* (*x* + *1*) *y* = *x* * *Beta x y*
proof −
have (*x* + *y*) * *Beta* (*x* + *1*) *y* = *Gamma* (*x* + *1*) * *Gamma y* * ((*x* + *y*) * *rGamma* ((*x* + *y*) + *1*))
unfolding *Beta-altdef* **by** (*simp only*: *ac-simps*)
also have ... = *x* * *Beta x y* **unfolding** *Beta-altdef*
by (*subst assms*[*THEN Gamma-plus1*] *rGamma-plus1*)+ (*simp only*: *ac-simps*)
finally show *?thesis* **.**

qed

lemma *Beta-plus1-right*:
assumes $x \notin \mathbb{Z}_{\leq 0}$ $y \notin \mathbb{Z}_{\leq 0}$
shows $(x + y) * Beta\ x\ (y + 1) = y * Beta\ x\ y$
using *Beta-plus1-left*[*of y x*] *assms* **by** (*simp add*: *Beta-commute add.commute*)

lemma *Gamma-Gamma-Beta*:
assumes $x \notin \mathbb{Z}_{\leq 0}$ $y \notin \mathbb{Z}_{\leq 0}$ $x + y \notin \mathbb{Z}_{\leq 0}$
shows $Gamma\ x * Gamma\ y = Beta\ x\ y * Gamma\ (x + y)$
unfolding *Beta-altdef* **using** *assms Gamma-eq-zero-iff*[*of x+y*]
by (*simp add*: *rGamma-inverse-Gamma*)

57.7 Legendre duplication theorem

context
begin

private lemma *Gamma-legendre-duplication-aux*:
fixes $z :: 'a :: Gamma$
assumes $z \notin \mathbb{Z}_{\leq 0}$ $z + 1/2 \notin \mathbb{Z}_{\leq 0}$
shows $Gamma\ z * Gamma\ (z + 1/2) = exp\ ((1 - 2*z) * of\text{-}real\ (ln\ 2)) * Gamma\ (1/2) * Gamma\ (2*z)$
proof −
let *?powr* = $\lambda b\ a.\ exp\ (a * of\text{-}real\ (ln\ (of\text{-}nat\ b)))$
let *?h* = $\lambda n.\ (fact\ (n-1))^2\ /\ fact\ (2*n-1) * of\text{-}nat\ (2\hat{}(2*n)) * exp\ (1/2 * of\text{-}real\ (ln\ (real\text{-}of\text{-}nat\ n)))$
{
fix $z :: 'a$ **assume** *z*: $z \notin \mathbb{Z}_{\leq 0}$ $z + 1/2 \notin \mathbb{Z}_{\leq 0}$
let *?g* = $\lambda n.$ *?powr 2* $(2*z) * Gamma\text{-}series'\ z\ n * Gamma\text{-}series'\ (z + 1/2)\ n\ /\ Gamma\text{-}series'\ (2*z)\ (2*n)$
have *eventually* $(\lambda n.\ ?g\ n = ?h\ n)$ *sequentially* **using** *eventually-gt-at-top*
proof *eventually-elim*
fix $n :: nat$ **assume** *n*: $n > 0$
let *?f* = $fact\ (n - 1) :: 'a$ **and** *?f'* = $fact\ (2*n - 1) :: 'a$
have *A*: $exp\ t * exp\ t = exp\ (2*t :: 'a)$ **for** *t* **by** (*subst exp-add* [*symmetric*]) *simp*
have *A*: $Gamma\text{-}series'\ z\ n * Gamma\text{-}series'\ (z + 1/2)\ n = ?f\hat{}2 * ?powr\ n\ (2*z + 1/2)\ /\ (pochhammer\ z\ n * pochhammer\ (z + 1/2)\ n)$
by (*simp add*: *Gamma-series'-def exp-add ring-distribs power2-eq-square A mult-ac*)
have *B*: $Gamma\text{-}series'\ (2*z)\ (2*n) = ?f' * ?powr\ 2\ (2*z) * ?powr\ n\ (2*z)\ /\ (of\text{-}nat\ (2\hat{}(2*n)) * pochhammer\ z\ n * pochhammer\ (z+1/2)\ n)$ **using** *n*
by (*simp add*: *Gamma-series'-def ln-mult exp-add ring-distribs pochhammer-double*)
from *z* **have** $pochhammer\ z\ n \neq 0$ **by** (*auto dest*: *pochhammer-eq-0-imp-nonpos-Int*)

moreover from *z* **have** *pochhammer* (*z* + *1/2*) *n* ≠ *0* **by** (*auto dest*: *pochhammer-eq-0-imp-nonpos-Int*)
ultimately have *?powr 2* (*2∗z*) ∗ (*Gamma-series' z n* ∗ *Gamma-series'* (*z* + *1/2*) *n*) / *Gamma-series'* (*2∗z*) (*2∗n*) =
?f^2 / *?f'* ∗ *of-nat* (*2^*(*2∗n*)) ∗ (*?powr n* ((*4∗z* + *1*)*/2*) ∗ *?powr n* (−*2∗z*))
using *n* **unfolding** *A B* **by** (*simp add*: *divide-simps exp-minus*)
also have *?powr n* ((*4∗z* + *1*)*/2*) ∗ *?powr n* (−*2∗z*) = *?powr n* (*1/2*)
by (*simp add*: *algebra-simps exp-add*[*symmetric*] *add-divide-distrib*)
finally show *?g n* = *?h n* **by** (*simp only*: *mult-ac*)
qed

moreover from *z double-in-nonpos-Ints-imp*[*of z*] **have** *2* ∗ *z* ∉ $\mathbb{Z}_{\leq 0}$ **by** *auto*
hence *?g* ⟶ *?powr 2* (*2∗z*) ∗ *Gamma z* ∗ *Gamma* (*z*+*1/2*) / *Gamma* (*2∗z*)
using *lim-subseq*[*of op* ∗ *2*, *OF - Gamma-series'-LIMSEQ*, *of 2∗z*]
by (*intro tendsto-intros Gamma-series'-LIMSEQ*)
(*simp-all add*: *o-def subseq-def Gamma-eq-zero-iff*)
ultimately have *?h* ⟶ *?powr 2* (*2∗z*) ∗ *Gamma z* ∗ *Gamma* (*z*+*1/2*) / *Gamma* (*2∗z*)
by (*rule Lim-transform-eventually*)
} note *lim* = *this*

from *assms double-in-nonpos-Ints-imp*[*of z*] **have** *z'*: *2* ∗ *z* ∉ $\mathbb{Z}_{\leq 0}$ **by** *auto*
from *fraction-not-in-ints*[*of 2 1*] **have** (*1/2* :: *'a*) ∉ $\mathbb{Z}_{\leq 0}$
by (*intro not-in-Ints-imp-not-in-nonpos-Ints*) *simp-all*
with *lim*[*of 1/2* :: *'a*] **have** *?h* ⟶ *2* ∗ *Gamma* (*1* / *2* :: *'a*) **by** (*simp add*: *exp-of-real*)
from *LIMSEQ-unique*[*OF this lim*[*OF assms*]] *z'* **show** *?thesis*
by (*simp add*: *divide-simps Gamma-eq-zero-iff ring-distribs exp-diff exp-of-real ac-simps*)
qed

private lemma *Gamma-reflection-aux*:
defines *h* ≡ λ*z*::*complex*. *if z* ∈ $\mathbb{Z}$ *then 0 else*
(*of-real pi* ∗ *cot* (*of-real pi∗z*) + *Digamma z* − *Digamma* (*1* − *z*))
defines *a* ≡ *complex-of-real pi*
obtains *h'* **where** *continuous-on UNIV h'* ⋀*z*. (*h has-field-derivative* (*h' z*)) (*at z*)
proof −
def *f* ≡ λ*n*. *a* ∗ *of-real* (*cos-coeff* (*n*+*1*) − *sin-coeff* (*n*+*2*))
def *F* ≡ λ*z*. *if z* = *0 then 0 else* (*cos* (*a∗z*) − *sin* (*a∗z*)/(*a∗z*)) / *z*
def *g* ≡ λ*n*. *complex-of-real* (*sin-coeff* (*n*+*1*))
def *G* ≡ λ*z*. *if z* = *0 then 1 else sin* (*a∗z*)/(*a∗z*)
have *a-nz*: *a* ≠ *0* **unfolding** *a-def* **by** *simp*

have (λ*n*. *f n* ∗ (*a∗z*) ^*n*) *sums* (*F z*) ∧ (λ*n*. *g n* ∗ (*a∗z*) ^*n*) *sums* (*G z*)
if *abs* (*Re z*) < *1* **for** *z*
proof (*cases z* = *0*; *rule conjI*)

assume $z \neq 0$
note $z = this\ that$

from z **have** *sin-nz*: $sin\ (a*z) \neq 0$ **unfolding** *a-def* **by** (*auto simp*: *sin-eq-0*)
have $(\lambda n.\ of\text{-}real\ (sin\text{-}coeff\ n) * (a*z)\ \hat{}\ n)\ sums\ (sin\ (a*z))$ **using** *sin-converges*[*of* $a*z$]
by (*simp add*: *scaleR-conv-of-real*)
from *sums-split-initial-segment*[*OF this*, *of 1*]
have $(\lambda n.\ (a*z) * of\text{-}real\ (sin\text{-}coeff\ (n+1)) * (a*z)\ \hat{}\ n)\ sums\ (sin\ (a*z))$ **by** (*simp add*: *mult-ac*)
from *sums-mult*[*OF this*, *of inverse* $(a*z)$] *z a-nz*
have *A*: $(\lambda n.\ g\ n * (a*z)\ \hat{}\ n)\ sums\ (sin\ (a*z)/(a*z))$
by (*simp add*: *field-simps g-def*)
with *z* **show** $(\lambda n.\ g\ n * (a*z)\ \hat{}\ n)\ sums\ (G\ z)$ **by** (*simp add*: *G-def*)
from *A z a-nz sin-nz* **have** *g-nz*: $(\sum n.\ g\ n * (a*z)\ \hat{}\ n) \neq 0$ **by** (*simp add*: *sums-iff g-def*)

have [*simp*]: *sin-coeff* $(Suc\ 0) = 1$ **by** (*simp add*: *sin-coeff-def*)
from *sums-split-initial-segment*[*OF sums-diff*[*OF cos-converges*[*of* $a*z$] *A*], *of 1*]
have $(\lambda n.\ z * f\ n * (a*z)\ \hat{}\ n)\ sums\ (cos\ (a*z) - sin\ (a*z)\ /\ (a*z))$
by (*simp add*: *mult-ac scaleR-conv-of-real ring-distribs f-def g-def*)
from *sums-mult*[*OF this*, *of inverse z*] *z assms*
show $(\lambda n.\ f\ n * (a*z)\ \hat{}\ n)\ sums\ (F\ z)$ **by** (*simp add*: *divide-simps mult-ac f-def F-def*)
next
assume *z*: $z = 0$
have $(\lambda n.\ f\ n * (a * z)\ \hat{}\ n)\ sums\ f\ 0$ **using** *powser-sums-zero*[*of f*] *z* **by** *simp*
with *z* **show** $(\lambda n.\ f\ n * (a * z)\ \hat{}\ n)\ sums\ (F\ z)$
by (*simp add*: *f-def F-def sin-coeff-def cos-coeff-def*)
have $(\lambda n.\ g\ n * (a * z)\ \hat{}\ n)\ sums\ g\ 0$ **using** *powser-sums-zero*[*of g*] *z* **by** *simp*
with *z* **show** $(\lambda n.\ g\ n * (a * z)\ \hat{}\ n)\ sums\ (G\ z)$
by (*simp add*: *g-def G-def sin-coeff-def cos-coeff-def*)
qed
note *sums* = *conjunct1*[*OF this*] *conjunct2*[*OF this*]

def *h2* $\equiv \lambda z.\ (\sum n.\ f\ n * (a*z)\ \hat{}\ n)\ /\ (\sum n.\ g\ n * (a*z)\ \hat{}\ n) +$
$Digamma\ (1 + z) - Digamma\ (1 - z)$
def *POWSER* $\equiv \lambda f\ z.\ (\sum n.\ f\ n * (z\ \hat{}\ n :: complex))$
def *POWSER'* $\equiv \lambda f\ z.\ (\sum n.\ diffs\ f\ n * (z\ \hat{}\ n :: complex))$

def *h2'* $\equiv \lambda z.\ a * (POWSER\ g\ (a*z) * POWSER'\ f\ (a*z) - POWSER\ f\ (a*z) * POWSER'\ g\ (a*z))\ /$
$(POWSER\ g\ (a*z))\ \hat{}\ 2 + Polygamma\ 1\ (1 + z) + Polygamma\ 1\ (1 - z)$

have *h-eq*: $h\ t = h2\ t$ **if** $abs\ (Re\ t) < 1$ **for** t
proof −
from *that* **have** *t*: $t \in \mathbb{Z} \longleftrightarrow t = 0$ **by** (*auto elim*!: *Ints-cases simp*: *dist-0-norm*)

hence $h\ t = a{*}cot\ (a{*}t) - 1/t + Digamma\ (1 + t) - Digamma\ (1 - t)$
unfolding *h-def* **using** *Digamma-plus1*[*of t*] **by** (*force simp*: *field-simps a-def*)
also have $a{*}cot\ (a{*}t) - 1/t = (F\ t)\ /\ (G\ t)$
using *t* **by** (*auto simp add*: *divide-simps sin-eq-0 cot-def a-def F-def G-def*)
also have $\ldots = (\sum n.\ f\ n * (a{*}t)\ \hat{}\ n)\ /\ (\sum n.\ g\ n * (a{*}t)\ \hat{}\ n)$
using *sums*[*of t*] *that* **by** (*simp add*: *sums-iff dist-0-norm*)
finally show $h\ t = h2\ t$ **by** (*simp only*: *h2-def*)
qed

let *?A* = $\{z.\ abs\ (Re\ z) < 1\}$
have *open* $(\{z.\ Re\ z < 1\} \cap \{z.\ Re\ z > -1\})$
using *open-halfspace-Re-gt open-halfspace-Re-lt* **by** *auto*
also have $(\{z.\ Re\ z < 1\} \cap \{z.\ Re\ z > -1\}) = \{z.\ abs\ (Re\ z) < 1\}$ **by** *auto*
finally have *open-A*: *open ?A* **.**
hence [*simp*]: *interior ?A* = *?A* **by** (*simp add*: *interior-open*)

have *summable-f*: *summable* $(\lambda n.\ f\ n * z\ \hat{}\ n)$ **for** *z*
by (*rule powser-inside*, *rule sums-summable*, *rule sums*[*of* i * *of-real* (*norm z* + *1*) / *a*])
(*simp-all add*: *norm-mult a-def del*: *of-real-add*)
have *summable-g*: *summable* $(\lambda n.\ g\ n * z\ \hat{}\ n)$ **for** *z*
by (*rule powser-inside*, *rule sums-summable*, *rule sums*[*of* i * *of-real* (*norm z* + *1*) / *a*])
(*simp-all add*: *norm-mult a-def del*: *of-real-add*)
have *summable-fg'*: *summable* $(\lambda n.\ diffs\ f\ n * z\ \hat{}\ n)$ *summable* $(\lambda n.\ diffs\ g\ n * z\ \hat{}\ n)$ **for** *z*
by (*intro termdiff-converges-all summable-f summable-g*)+
have (*POWSER f has-field-derivative* (*POWSER' f z*)) (*at z*)
(*POWSER g has-field-derivative* (*POWSER' g z*)) (*at z*) **for** *z*
unfolding *POWSER-def POWSER'-def*
by (*intro termdiffs-strong-converges-everywhere summable-f summable-g*)+
note *derivs* = *this*[*THEN DERIV-chain2*[*OF - DERIV-cmult*[*OF DERIV-ident*]], *unfolded POWSER-def*]
have *isCont* (*POWSER f*) *z isCont* (*POWSER g*) *z isCont* (*POWSER' f*) *z isCont* (*POWSER' g*) *z*
for *z* **unfolding** *POWSER-def POWSER'-def*
by (*intro isCont-powser-converges-everywhere summable-f summable-g summable-fg'*)+
note *cont* = *this*[*THEN isCont-o2*[*rotated*], *unfolded POWSER-def POWSER'-def*]

{
fix *z* :: *complex* **assume** *z*: *abs* (*Re z*) < *1*
def *d* ≡ i * *of-real* (*norm z* + *1*)
have *d*: *abs* (*Re d*) < *1 norm z* < *norm d* **by** (*simp-all add*: *d-def norm-mult del*: *of-real-add*)
have *eventually* $(\lambda z.\ h\ z = h2\ z)$ (*nhds z*)
using *eventually-nhds-in-nhd*[*of z ?A*] **using** *h-eq z*
by (*auto elim*!: *eventually-mono simp*: *dist-0-norm*)

moreover from *sums*(*2*)[*OF z*] *z* **have** *nz*: $(\sum n.\ g\ n * (a * z)\ \hat{}\ n) \neq 0$

```
    unfolding G-def by (auto simp: sums-iff sin-eq-0 a-def)
  have A: z ∈ ℤ ⟷ z = 0 using z by (auto elim!: Ints-cases)
  have no-int: 1 + z ∈ ℤ ⟷ z = 0 using z Ints-diff[of 1+z 1] A
    by (auto elim!: nonpos-Ints-cases)
  have no-int': 1 − z ∈ ℤ ⟷ z = 0 using z Ints-diff[of 1 1−z] A
    by (auto elim!: nonpos-Ints-cases)
  from no-int no-int' have no-int: 1 − z ∉ ℤ≤0 1 + z ∉ ℤ≤0 by auto
  have (h2 has-field-derivative h2' z) (at z) unfolding h2-def
  by (rule DERIV-cong, (rule derivative-intros refl derivs[unfolded POWSER-def]
nz no-int)+)
      (auto simp: h2'-def POWSER-def field-simps power2-eq-square)
  ultimately have deriv: (h has-field-derivative h2' z) (at z)
    by (subst DERIV-cong-ev[OF refl - refl])

  from sums(2)[OF z] z have (∑ n. g n * (a * z) ^ n) ≠ 0
    unfolding G-def by (auto simp: sums-iff a-def sin-eq-0)
   hence isCont h2' z using no-int unfolding h2'-def[abs-def] POWSER-def
POWSER'-def
    by (intro continuous-intros cont
        continuous-on-compose2[OF - continuous-on-Polygamma[of {z. Re z >
0}]]) auto
  note deriv and this
 } note A = this

 interpret h: periodic-fun-simple' h
 proof
  fix z :: complex
  show h (z + 1) = h z
  proof (cases z ∈ ℤ)
    assume z: z ∉ ℤ
    hence A: z + 1 ∉ ℤ z ≠ 0 using Ints-diff[of z+1 1] by auto
    hence Digamma (z + 1) − Digamma (−z) = Digamma z − Digamma (−z
+ 1)
      by (subst (1 2) Digamma-plus1) simp-all
    with A z show h (z + 1) = h z
      by (simp add: h-def sin-plus-pi cos-plus-pi ring-distribs cot-def)
  qed (simp add: h-def)
 qed

 have h2'-eq: h2' (z − 1) = h2' z if z: Re z > 0 Re z < 1 for z
 proof −
  have ((λz. h (z − 1)) has-field-derivative h2' (z − 1)) (at z)
    by (rule DERIV-cong, rule DERIV-chain'[OF - A(1)])
      (insert z, auto intro!: derivative-eq-intros)
  hence (h has-field-derivative h2' (z − 1)) (at z) by (subst (asm) h.minus-1)
   moreover from z have (h has-field-derivative h2' z) (at z) by (intro A)
simp-all
  ultimately show h2' (z − 1) = h2' z by (rule DERIV-unique)
 qed
```

```
  def h2'' ≡ λz. h2' (z − of-int ⌊Re z⌋)
  have deriv: (h has-field-derivative h2'' z) (at z) for z
  proof −
    fix z :: complex
    have B: |Re z − real-of-int ⌊Re z⌋| < 1 by linarith
    have ((λt. h (t − of-int ⌊Re z⌋)) has-field-derivative h2'' z) (at z)
      unfolding h2''-def by (rule DERIV-cong, rule DERIV-chain'[OF - A(1)])
                            (insert B, auto intro!: derivative-intros)
    thus (h has-field-derivative h2'' z) (at z) by (simp add: h.minus-of-int)
  qed

  have cont: continuous-on UNIV h2''
  proof (intro continuous-at-imp-continuous-on ballI)
    fix z :: complex
    def r ≡ ⌊Re z⌋
    def A ≡ {t. of-int r − 1 < Re t ∧ Re t < of-int r + 1}
    have continuous-on A (λt. h2' (t − of-int r)) unfolding A-def
    by (intro continuous-at-imp-continuous-on isCont-o2[OF - A(2)] ballI continuous-intros)
        (simp-all add: abs-real-def)
    moreover have h2'' t = h2' (t − of-int r) if t: t ∈ A for t
    proof (cases Re t ≥ of-int r)
      case True
       from t have of-int r − 1 < Re t Re t < of-int r + 1 by (simp-all add:
A-def)
      with True have ⌊Re t⌋ = ⌊Re z⌋ unfolding r-def by linarith
      thus ?thesis by (auto simp: r-def h2''-def)
    next
      case False
       from t have t: of-int r − 1 < Re t Re t < of-int r + 1 by (simp-all add:
A-def)
      with False have t': ⌊Re t⌋ = ⌊Re z⌋ − 1 unfolding r-def by linarith
      moreover from t False have h2' (t − of-int r + 1 − 1) = h2' (t − of-int
r + 1)
        by (intro h2'-eq) simp-all
      ultimately show ?thesis by (auto simp: r-def h2''-def algebra-simps t')
    qed
    ultimately have continuous-on A h2'' by (subst continuous-on-cong[OF refl])
    moreover {
      have open ({t. of-int r − 1 < Re t} ∩ {t. of-int r + 1 > Re t})
        by (intro open-Int open-halfspace-Re-gt open-halfspace-Re-lt)
      also have {t. of-int r − 1 < Re t} ∩ {t. of-int r + 1 > Re t} = A
        unfolding A-def by blast
      finally have open A .
    }
    ultimately have C: isCont h2'' t if t ∈ A for t using that
      by (subst (asm) continuous-on-eq-continuous-at) auto
    have of-int r − 1 < Re z Re z < of-int r + 1 unfolding r-def by linarith+
    thus isCont h2'' z by (intro C) (simp-all add: A-def)
```

qed

from *that*[*OF cont deriv*] **show** *?thesis* **.**
qed

lemma *Gamma-reflection-complex*:
fixes *z* :: *complex*
shows *Gamma z* ∗ *Gamma* (*1* − *z*) = *of-real pi* / *sin* (*of-real pi* ∗ *z*)
proof −
let *?g* = λ*z*::*complex*. *Gamma z* ∗ *Gamma* (*1* − *z*) ∗ *sin* (*of-real pi* ∗ *z*)
def *g* ≡ λ*z*::*complex*. *if z* ∈ ℤ *then of-real pi else ?g z*
let *?h* = λ*z*::*complex*. (*of-real pi* ∗ *cot* (*of-real pi*∗*z*) + *Digamma z* − *Digamma* (*1* − *z*))
def *h* ≡ λ*z*::*complex*. *if z* ∈ ℤ *then 0 else ?h z*

— *g* is periodic with period 1.
interpret *g*: *periodic-fun-simple′ g*
proof
fix *z* :: *complex*
show *g* (*z* + *1*) = *g z*
proof (*cases z* ∈ ℤ)
case *False*
hence *z* ∗ *g z* = *z* ∗ *Beta z* (− *z* + *1*) ∗ *sin* (*of-real pi* ∗ *z*) **by** (*simp add*: *g-def Beta-def*)
also have *z* ∗ *Beta z* (− *z* + *1*) = (*z* + *1* + −*z*) ∗ *Beta* (*z* + *1*) (− *z* + *1*)
using *False Ints-diff*[*of 1 1* − *z*] *nonpos-Ints-subset-Ints*
by (*subst Beta-plus1-left* [*symmetric*]) *auto*
also have ... ∗ *sin* (*of-real pi* ∗ *z*) = *z* ∗ (*Beta* (*z* + *1*) (−*z*) ∗ *sin* (*of-real pi* ∗ (*z* + *1*)))
using *False Ints-diff*[*of z+1 1*] *Ints-minus*[*of* −*z*] *nonpos-Ints-subset-Ints*
by (*subst Beta-plus1-right*) (*auto simp*: *ring-distribs sin-plus-pi*)
also from *False* **have** *Beta* (*z* + *1*) (−*z*) ∗ *sin* (*of-real pi* ∗ (*z* + *1*)) = *g* (*z* + *1*)
using *Ints-diff*[*of z+1 1*] **by** (*auto simp*: *g-def Beta-def*)
finally show *g* (*z* + *1*) = *g z* **using** *False* **by** (*subst* (*asm*) *mult-left-cancel*) *auto*
qed (*simp add*: *g-def*)
qed

— *g* is entire.
have *g-g′*: (*g has-field-derivative* (*h z* ∗ *g z*)) (*at z*) **for** *z* :: *complex*
proof (*cases z* ∈ ℤ)
let *?h′* = λ*z*. *Beta z* (*1* − *z*) ∗ ((*Digamma z* − *Digamma* (*1* − *z*)) ∗ *sin* (*z* ∗ *of-real pi*) +
of-real pi ∗ *cos* (*z* ∗ *of-real pi*))
case *False*
from *False* **have** *eventually* (λ*t*. *t* ∈ *UNIV* − ℤ) (*nhds z*)
by (*intro eventually-nhds-in-open*) (*auto simp*: *open-Diff*)
hence *eventually* (λ*t*. *g t* = *?g t*) (*nhds z*) **by** *eventually-elim* (*simp add*: *g-def*)

```
    moreover {
      from False Ints-diff[of 1 1−z] have 1 − z ∉ ℤ by auto
      hence (?g has-field-derivative ?h' z) (at z) using nonpos-Ints-subset-Ints
        by (auto intro!: derivative-eq-intros simp: algebra-simps Beta-def)
      also from False have sin (of-real pi * z) ≠ 0 by (subst sin-eq-0) auto
      hence ?h' z = h z * g z
      using False unfolding g-def h-def cot-def by (simp add: field-simps Beta-def)
      finally have (?g has-field-derivative (h z * g z)) (at z) .
    }
    ultimately show ?thesis by (subst DERIV-cong-ev[OF refl - refl])
  next
    case True
    then obtain n where z: z = of-int n by (auto elim!: Ints-cases)
    let ?t = (λz::complex. if z = 0 then 1 else sin z / z) ∘ (λz. of-real pi * z)
    have deriv-0: (g has-field-derivative 0) (at 0)
    proof (subst DERIV-cong-ev[OF refl - refl])
      show eventually (λz. g z = of-real pi * Gamma (1 + z) * Gamma (1 − z)
* ?t z) (nhds 0)
        using eventually-nhds-ball[OF zero-less-one, of 0::complex]
      proof eventually-elim
        fix z :: complex assume z: z ∈ ball 0 1
        show g z = of-real pi * Gamma (1 + z) * Gamma (1 − z) * ?t z
        proof (cases z = 0)
          assume z': z ≠ 0
            with z have z'': z ∉ ℤ≤0 z ∉ ℤ by (auto elim!: Ints-cases simp:
dist-0-norm)
          from Gamma-plus1[OF this(1)] have Gamma z = Gamma (z + 1) / z
by simp
          with z'' z' show ?thesis by (simp add: g-def ac-simps)
        qed (simp add: g-def)
      qed
      have (?t has-field-derivative (0 * of-real pi)) (at 0)
        using has-field-derivative-sin-z-over-z[of UNIV :: complex set]
        by (intro DERIV-chain) simp-all
    thus ((λz. of-real pi * Gamma (1 + z) * Gamma (1 − z) * ?t z) has-field-derivative
0) (at 0)
        by (auto intro!: derivative-eq-intros simp: o-def)
    qed

    have ((g ∘ (λx. x − of-int n)) has-field-derivative 0 * 1) (at (of-int n))
      using deriv-0 by (intro DERIV-chain) (auto intro!: derivative-eq-intros)
    also have g ∘ (λx. x − of-int n) = g by (intro ext) (simp add: g.minus-of-int)
    finally show (g has-field-derivative (h z * g z)) (at z) by (simp add: z h-def)
  qed

  have g-eq: g (z/2) * g ((z+1)/2) = Gamma (1/2) ^2 * g z if Re z > −1 Re z
< 2 for z
  proof (cases z ∈ ℤ)
    case True
```

with *that* **have** $z = 0 \vee z = 1$ **by** (*force elim*!: *Ints-cases*)
moreover have $g\ 0 * g\ (1/2) = Gamma\ (1/2)\hat{}2 * g\ 0$
using *fraction-not-in-ints*[**where** $'a = complex$, *of 2 1*] **by** (*simp add*: *g-def power2-eq-square*)
moreover have $g\ (1/2) * g\ 1 = Gamma\ (1/2)\hat{}2 * g\ 1$
using *fraction-not-in-ints*[**where** $'a = complex$, *of 2 1*]
by (*simp add*: *g-def power2-eq-square Beta-def algebra-simps*)
ultimately show *?thesis* **by** *force*
next
case *False*
hence *z*: $z/2 \notin \mathbb{Z}$ $(z+1)/2 \notin \mathbb{Z}$ **using** *Ints-diff*[*of z+1 1*] **by** (*auto elim*!: *Ints-cases*)
hence *z'*: $z/2 \notin \mathbb{Z}_{\leq 0}$ $(z+1)/2 \notin \mathbb{Z}_{\leq 0}$ **by** (*auto elim*!: *nonpos-Ints-cases*)
from *z* **have** $1-z/2 \notin \mathbb{Z}$ $1-((z+1)/2) \notin \mathbb{Z}$
using *Ints-diff*[*of 1 1−z/2*] *Ints-diff*[*of 1 1−((z+1)/2)*] **by** *auto*
hence *z''*: $1-z/2 \notin \mathbb{Z}_{\leq 0}$ $1-((z+1)/2) \notin \mathbb{Z}_{\leq 0}$ **by** (*auto elim*!: *nonpos-Ints-cases*)
from *z* **have** $g\ (z/2) * g\ ((z+1)/2) =$
$(Gamma\ (z/2) * Gamma\ ((z+1)/2)) * (Gamma\ (1-z/2) * Gamma\ (1-((z+1)/2))) *$
$(sin\ (of\text{-}real\ pi * z/2) * sin\ (of\text{-}real\ pi * (z+1)/2))$
by (*simp add*: *g-def*)
also from *z' Gamma-legendre-duplication-aux*[*of z/2*]
have $Gamma\ (z/2) * Gamma\ ((z+1)/2) = exp\ ((1-z) * of\text{-}real\ (ln\ 2)) * Gamma\ (1/2) * Gamma\ z$
by (*simp add*: *add-divide-distrib*)
also from *z''* *Gamma-legendre-duplication-aux*[*of 1−(z+1)/2*]
have $Gamma\ (1-z/2) * Gamma\ (1-(z+1)/2) =$
$Gamma\ (1-z) * Gamma\ (1/2) * exp\ (z * of\text{-}real\ (ln\ 2))$
by (*simp add*: *add-divide-distrib ac-simps*)
finally have $g\ (z/2) * g\ ((z+1)/2) = Gamma\ (1/2)\hat{}2 * (Gamma\ z * Gamma\ (1-z) *$
$(2 * (sin\ (of\text{-}real\ pi*z/2) * sin\ (of\text{-}real\ pi*(z+1)/2))))$
by (*simp add*: *add-ac power2-eq-square exp-add ring-distribs exp-diff exp-of-real*)
also have $sin\ (of\text{-}real\ pi*(z+1)/2) = cos\ (of\text{-}real\ pi*z/2)$
using *cos-sin-eq*[*of* $-$ *of-real pi * z/2*, *symmetric*]
by (*simp add*: *ring-distribs add-divide-distrib ac-simps*)
also have $2 * (sin\ (of\text{-}real\ pi*z/2) * cos\ (of\text{-}real\ pi*z/2)) = sin\ (of\text{-}real\ pi * z)$
by (*subst sin-times-cos*) (*simp add*: *field-simps*)
also have $Gamma\ z * Gamma\ (1 - z) * sin\ (complex\text{-}of\text{-}real\ pi * z) = g\ z$
using ‹$z \notin \mathbb{Z}$› **by** (*simp add*: *g-def*)
finally show *?thesis* **.**
qed
have *g-eq*: $g\ (z/2) * g\ ((z+1)/2) = Gamma\ (1/2)\hat{}2 * g\ z$ **for** *z*
proof −
def $r \equiv \lfloor Re\ z\ /\ 2 \rfloor$
have $Gamma\ (1/2)\hat{}2 * g\ z = Gamma\ (1/2)\hat{}2 * g\ (z - of\text{-}int\ (2*r))$ **by** (*simp only*: *g.minus-of-int*)
also have $of\text{-}int\ (2*r) = 2 * of\text{-}int\ r$ **by** *simp*

```
    also have Re z − 2 ∗ of-int r > −1 Re z − 2 ∗ of-int r < 2 unfolding r-def
by linarith+
    hence Gamma (1/2) ^2 ∗ g (z − 2 ∗ of-int r) =
                g ((z − 2 ∗ of-int r)/2) ∗ g ((z − 2 ∗ of-int r + 1)/2)
      unfolding r-def by (intro g-eq[symmetric]) simp-all
    also have (z − 2 ∗ of-int r) / 2 = z/2 − of-int r by simp
    also have g ... = g (z/2) by (rule g.minus-of-int)
    also have (z − 2 ∗ of-int r + 1) / 2 = (z + 1)/2 − of-int r by simp
    also have g ... = g ((z+1)/2) by (rule g.minus-of-int)
    finally show ?thesis ..
  qed

  have g-nz [simp]: g z ≠ 0 for z :: complex
  unfolding g-def using Ints-diff[of 1 1 − z]
    by (auto simp: Gamma-eq-zero-iff sin-eq-0 dest!: nonpos-Ints-Int)

  have h-eq: h z = (h (z/2) + h ((z+1)/2)) / 2 for z
  proof −
    have ((λt. g (t/2) ∗ g ((t+1)/2)) has-field-derivative
                  (g (z/2) ∗ g ((z+1)/2)) ∗ ((h (z/2) + h ((z+1)/2)) / 2)) (at
z)
      by (auto intro!: derivative-eq-intros g-g'[THEN DERIV-chain2] simp: field-simps)
    hence ((λt. Gamma (1/2) ^2 ∗ g t) has-field-derivative
            Gamma (1/2) ^2 ∗ g z ∗ ((h (z/2) + h ((z+1)/2)) / 2)) (at z)
      by (subst (1 2) g-eq[symmetric]) simp
    from DERIV-cmult[OF this, of inverse ((Gamma (1/2)) ^2)]
      have (g has-field-derivative (g z ∗ ((h (z/2) + h ((z+1)/2))/2))) (at z)
      using fraction-not-in-ints[where 'a = complex, of 2 1]
    by (simp add: divide-simps Gamma-eq-zero-iff not-in-Ints-imp-not-in-nonpos-Ints)
    moreover have (g has-field-derivative (g z ∗ h z)) (at z)
      using g-g'[of z] by (simp add: ac-simps)
    ultimately have g z ∗ h z = g z ∗ ((h (z/2) + h ((z+1)/2))/2)
      by (intro DERIV-unique)
    thus h z = (h (z/2) + h ((z+1)/2)) / 2 by simp
  qed

  obtain h' where h'-cont: continuous-on UNIV h' and
                h-h': ⋀z. (h has-field-derivative h' z) (at z)
    unfolding h-def by (erule Gamma-reflection-aux)

  have h'-eq: h' z = (h' (z/2) + h' ((z+1)/2)) / 4 for z
  proof −
    have ((λt. (h (t/2) + h ((t+1)/2)) / 2) has-field-derivative
                  ((h' (z/2) + h' ((z+1)/2)) / 4)) (at z)
      by (fastforce intro!: derivative-eq-intros h-h'[THEN DERIV-chain2])
    hence (h has-field-derivative ((h' (z/2) + h' ((z+1)/2))/4)) (at z)
      by (subst (asm) h-eq[symmetric])
    from h-h' and this show h' z = (h' (z/2) + h' ((z+1)/2)) / 4 by (rule
DERIV-unique)
```

qed

have *h'-zero*: $h'\ z = 0$ **for** z
proof −
def $m \equiv max\ 1\ |Re\ z|$
def $B \equiv \{t.\ abs\ (Re\ t) \leq m \wedge abs\ (Im\ t) \leq abs\ (Im\ z)\}$
have *closed* $(\{t.\ Re\ t \geq -m\} \cap \{t.\ Re\ t \leq m\} \cap$
$\{t.\ Im\ t \geq -|Im\ z|\} \cap \{t.\ Im\ t \leq |Im\ z|\})$
(**is** *closed ?B*) **by** (*intro closed-Int closed-halfspace-Re-ge closed-halfspace-Re-le closed-halfspace-Im-ge closed-halfspace-Im-le*)
also have $?B = B$ **unfolding** *B-def* **by** *fastforce*
finally have *closed B* **.**
moreover have *bounded B* **unfolding** *bounded-iff*
proof (*intro ballI exI*)
fix t **assume** t: $t \in B$
have $norm\ t \leq |Re\ t| + |Im\ t|$ **by** (*rule cmod-le*)
also from t **have** $|Re\ t| \leq m$ **unfolding** *B-def* **by** *blast*
also from t **have** $|Im\ t| \leq |Im\ z|$ **unfolding** *B-def* **by** *blast*
finally show $norm\ t \leq m + |Im\ z|$ **by** − *simp*
qed
ultimately have *compact*: *compact B* **by** (*subst compact-eq-bounded-closed*) *blast*

def $M \equiv SUP\ z{:}B.\ norm\ (h'\ z)$
have *compact* $(h'\ `\ B)$
by (*intro compact-continuous-image continuous-on-subset*[*OF h'-cont*] *compact*) *blast*+
hence *bdd*: *bdd-above* $((\lambda z.\ norm\ (h'\ z))\ `\ B)$
using *bdd-above-norm*[*of h' ` B*] **by** (*simp add*: *image-comp o-def compact-imp-bounded*)
have $norm\ (h'\ z) \leq M$ **unfolding** *M-def* **by** (*intro cSUP-upper bdd*) (*simp-all add*: *B-def m-def*)
also have $M \leq M/2$
proof (*subst M-def*, *subst cSUP-le-iff*)
have $z \in B$ **unfolding** *B-def m-def* **by** *simp*
thus $B \neq \{\}$ **by** *auto*
next
show $\forall\, z{\in}B.\ norm\ (h'\ z) \leq M/2$
proof
fix t :: *complex* **assume** t: $t \in B$
from *h'-eq*[*of t*] t **have** $h'\ t = (h'\ (t/2) + h'\ ((t+1)/2))\ /\ 4$ **by** (*simp add*: *dist-0-norm*)
also have $norm\ \ldots = norm\ (h'\ (t/2) + h'\ ((t+1)/2))\ /\ 4$ **by** *simp*
also have $norm\ (h'\ (t/2) + h'\ ((t+1)/2)) \leq norm\ (h'\ (t/2)) + norm\ (h'\ ((t+1)/2))$
by (*rule norm-triangle-ineq*)
also from t **have** $abs\ (Re\ ((t\ +\ 1)/2)) \leq m$ **unfolding** *m-def B-def* **by** *auto*
with t **have** $t/2 \in B$ $(t+1)/2 \in B$ **unfolding** *B-def* **by** *auto*
hence $norm\ (h'\ (t/2)) + norm\ (h'\ ((t+1)/2)) \leq M + M$ **unfolding** *M-def*

```
      by (intro add-mono cSUP-upper bdd) (auto simp: B-def)
     also have (M + M) / 4 = M / 2 by simp
     finally show norm (h' t) ≤ M/2 by − simp-all
    qed
   qed (insert bdd, auto simp: cball-eq-empty)
   hence M ≤ 0 by simp
   finally show h' z = 0 by simp
  qed
  have h-h'-2: (h has-field-derivative 0) (at z) for z
   using h-h'[of z] h'-zero[of z] by simp

  have g-real: g z ∈ ℝ if z ∈ ℝ for z
   unfolding g-def using that by (auto intro!: Reals-mult Gamma-complex-real)
  have h-real: h z ∈ ℝ if z ∈ ℝ for z
    unfolding h-def using that by (auto intro!: Reals-mult Reals-add Reals-diff
Polygamma-Real)
  have g-nz: g z ≠ 0 for z unfolding g-def using Ints-diff[of 1 1−z]
   by (auto simp: Gamma-eq-zero-iff sin-eq-0)

  from h'-zero h-h'-2 have ∃ c. ∀ z∈UNIV. h z = c
   by (intro has-field-derivative-zero-constant) (simp-all add: dist-0-norm)
  then obtain c where c: ⋀z. h z = c by auto
  have ∃ u. u ∈ closed-segment 0 1 ∧ Re (g 1) − Re (g 0) = Re (h u ∗ g u ∗ (1
− 0))
   by (intro complex-mvt-line g-g')
   find-theorems name:deriv Reals
  then guess u by (elim exE conjE) note u = this
  from u(1) have u': u ∈ ℝ unfolding closed-segment-def
   by (auto simp: scaleR-conv-of-real)
  from u' g-real[of u] g-nz[of u] have Re (g u) ≠ 0 by (auto elim!: Reals-cases)
  with u(2) c[of u] g-real[of u] g-nz[of u] u'
   have Re c = 0 by (simp add: complex-is-Real-iff g.of-1)
  with h-real[of 0] c[of 0] have c = 0 by (auto elim!: Reals-cases)
  with c have A: h z ∗ g z = 0 for z by simp
  hence (g has-field-derivative 0) (at z) for z using g-g'[of z] by simp
   hence ∃ c'. ∀ z∈UNIV. g z = c' by (intro has-field-derivative-zero-constant)
simp-all
  then obtain c' where c: ⋀z. g z = c' by (force simp: dist-0-norm)
  moreover from this[of 0] have c' = pi unfolding g-def by simp
  ultimately have g z = pi by simp

  show ?thesis
  proof (cases z ∈ ℤ)
   case False
   with ‹g z = pi› show ?thesis by (auto simp: g-def divide-simps)
  next
   case True
   then obtain n where n: z = of-int n by (elim Ints-cases)
   with sin-eq-0[of of-real pi ∗ z] have sin (of-real pi ∗ z) = 0 by force
```

moreover have *of-int* $(1 - n) \in \mathbb{Z}_{\leq 0}$ **if** $n > 0$ **using** *that* **by** (*intro nonpos-Ints-of-int*) *simp*
ultimately show *?thesis* **using** *n*
by (*cases* $n \leq 0$) (*auto simp*: *Gamma-eq-zero-iff nonpos-Ints-of-int*)
qed
qed

lemma *rGamma-reflection-complex*:
rGamma z $*$ *rGamma* $(1 - z ::$ *complex*$)$ $=$ *sin* $($*of-real pi* $*$ *z*$)$ / *of-real pi*
using *Gamma-reflection-complex*[*of z*]
by (*simp add*: *Gamma-def divide-simps split*: *split-if-asm*)

lemma *rGamma-reflection-complex'*:
rGamma z $*$ *rGamma* $(-$ *z* $::$ *complex*$)$ $= -z *$ *sin* $($*of-real pi* $*$ *z*$)$ / *of-real pi*
proof −
have *rGamma z* $*$ *rGamma* $(-z) = -z *$ $($*rGamma z* $*$ *rGamma* $(1 - z))$
using *rGamma-plus1*[*of* $-z$, *symmetric*] **by** *simp*
also have *rGamma z* $*$ *rGamma* $(1 - z)$ $=$ *sin* $($*of-real pi* $*$ *z*$)$ / *of-real pi*
by (*rule rGamma-reflection-complex*)
finally show *?thesis* **by** *simp*
qed

lemma *Gamma-reflection-complex'*:
Gamma z $*$ *Gamma* $(-$ *z* $::$ *complex*$)$ $= -$ *of-real pi* / $($*z* $*$ *sin* $($*of-real pi* $*$ *z*$))$
using *rGamma-reflection-complex'*[*of z*] **by** (*force simp add*: *Gamma-def divide-simps mult-ac*)

lemma *Gamma-one-half-real*: *Gamma* $(1/2 ::$ *real*$)$ $=$ *sqrt pi*
proof −
from *Gamma-reflection-complex*[*of* $1/2$] *fraction-not-in-ints*[**where** *'a* $=$ *complex*, *of* 2 1]
have *Gamma* $(1/2 ::$ *complex*$)\hat{}2 =$ *of-real pi* **by** (*simp add*: *power2-eq-square*)
hence *of-real pi* $=$ *Gamma* $($*complex-of-real* $(1/2))\hat{}2$ **by** *simp*
also have $\ldots =$ *of-real* $((Gamma\ (1/2))\hat{}2)$ **by** (*subst Gamma-complex-of-real*) *simp-all*
finally have *Gamma* $(1/2)\hat{}2 = pi$ **by** (*subst* (*asm*) *of-real-eq-iff*) *simp-all*
moreover have *Gamma* $(1/2 ::$ *real*$) \geq 0$ **using** *Gamma-real-pos*[*of* $1/2$] **by** *simp*
ultimately show *?thesis* **by** (*rule real-sqrt-unique* [*symmetric*])
qed

lemma *Gamma-one-half-complex*: *Gamma* $(1/2 ::$ *complex*$)$ $=$ *of-real* $($*sqrt pi*$)$
proof −
have *Gamma* $(1/2 ::$ *complex*$)$ $=$ *Gamma* $($*of-real* $(1/2))$ **by** *simp*
also have $\ldots =$ *of-real* $($*sqrt pi*$)$ **by** (*simp only*: *Gamma-complex-of-real Gamma-one-half-real*)
finally show *?thesis* **.**
qed

lemma *Gamma-legendre-duplication*:
fixes z :: *complex*
assumes $z \notin \mathbb{Z}_{\leq 0}$ $z + 1/2 \notin \mathbb{Z}_{\leq 0}$
shows *Gamma z * Gamma (z + 1/2) =*
*exp ((1 − 2*z) * of-real (ln 2)) * of-real (sqrt pi) * Gamma (2*z)*
using *Gamma-legendre-duplication-aux*[*OF assms*] **by** (*simp add*: *Gamma-one-half-complex*)

end

57.8 Limits and residues

The inverse of the Gamma function has simple zeros:

lemma *rGamma-zeros*:
(λ*z*. *rGamma z / (z + of-nat n)) − (− of-nat n) → ((−1) ^ n * fact n* :: *'a* :: *Gamma*)
proof (*subst tendsto-cong*)
let *?f* = λ*z*. *pochhammer z n * rGamma (z + of-nat (Suc n))* :: *'a*
from *eventually-at-ball'*[*OF zero-less-one*, *of − of-nat n* :: *'a UNIV*]
show *eventually* (λ*z*. *rGamma z / (z + of-nat n) = ?f z*) (*at* (*− of-nat n*))
by (*subst pochhammer-rGamma*[*of - Suc n*])
(*auto elim*!: *eventually-mono simp*: *divide-simps pochhammer-rec' eq-neg-iff-add-eq-0*)
have *isCont ?f* (*− of-nat n*) **by** (*intro continuous-intros*)
thus *?f − (− of-nat n) → (− 1) ^ n * fact n* **unfolding** *isCont-def*
by (*simp add*: *pochhammer-same*)
qed

The simple zeros of the inverse of the Gamma function correspond to simple poles of the Gamma function, and their residues can easily be computed from the limit we have just proven:

lemma *Gamma-poles*: *filterlim Gamma at-infinity* (*at* (*− of-nat n* :: *'a* :: *Gamma*))
proof −
from *eventually-at-ball'*[*OF zero-less-one*, *of − of-nat n* :: *'a UNIV*]
have *eventually* (λ*z*. *rGamma z ≠* (*0* :: *'a*)) (*at* (*− of-nat n*))
by (*auto elim*!: *eventually-mono nonpos-Ints-cases'*
simp: *rGamma-eq-zero-iff dist-of-nat dist-minus*)
with *isCont-rGamma*[*of − of-nat n* :: *'a*, *OF continuous-ident*]
have *filterlim* (λ*z*. *inverse* (*rGamma z*) :: *'a*) *at-infinity* (*at* (*− of-nat n*))
unfolding *isCont-def* **by** (*intro filterlim-compose*[*OF filterlim-inverse-at-infinity*])
(*simp-all add*: *filterlim-at*)
moreover have (λ*z*. *inverse* (*rGamma z*) :: *'a*) = *Gamma*
by (*intro ext*) (*simp add*: *rGamma-inverse-Gamma*)
ultimately show *?thesis* **by** (*simp only*:)
qed

lemma *Gamma-residues*:
(λ*z*. *Gamma z * (z + of-nat n)) − (− of-nat n) → ((−1) ^ n / fact n* :: *'a* :: *Gamma*)

proof (*subst tendsto-cong*)
let *?c* = (− *1*) ^ *n* / *fact n* :: *'a*
from *eventually-at-ball'*[*OF zero-less-one*, *of* − *of-nat n* :: *'a UNIV*]
show *eventually* (λ*z*. *Gamma z* ∗ (*z* + *of-nat n*) = *inverse* (*rGamma z* / (*z* + *of-nat n*)))
(*at* (− *of-nat n*))
by (*auto elim*!: *eventually-mono simp*: *divide-simps rGamma-inverse-Gamma*)
have (λ*z*. *inverse* (*rGamma z* / (*z* + *of-nat n*))) − (− *of-nat n*) →
inverse ((− *1*) ^ *n* ∗ *fact n* :: *'a*)
by (*intro tendsto-intros rGamma-zeros*) *simp-all*
also have *inverse* ((− *1*) ^ *n* ∗ *fact n*) = *?c*
by (*simp-all add*: *field-simps power-mult-distrib* [*symmetric*] *del*: *power-mult-distrib*)
finally show (λ*z*. *inverse* (*rGamma z* / (*z* + *of-nat n*))) − (− *of-nat n*) → *?c* **.**
qed

57.9 Alternative definitions

57.9.1 Variant of the Euler form

definition *Gamma-series-euler'* **where**
Gamma-series-euler' z n =
inverse z ∗ (∏ *k*=*1*..*n*. *exp* (*z* ∗ *of-real* (*ln* (*1* + *inverse* (*of-nat k*)))) / (*1* + *z* / *of-nat k*))

context
begin
private lemma *Gamma-euler'-aux1*:
fixes *z* :: *'a* :: {*real-normed-field*,*banach*}
assumes *n*: *n* > *0*
shows *exp* (*z* ∗ *of-real* (*ln* (*of-nat n* + *1*))) = (∏ *k*=*1*..*n*. *exp* (*z* ∗ *of-real* (*ln* (*1* + *1* / *of-nat k*))))
proof −
have (∏ *k*=*1*..*n*. *exp* (*z* ∗ *of-real* (*ln* (*1* + *1* / *of-nat k*)))) =
exp (*z* ∗ *of-real* (∑ *k* = *1*..*n*. *ln* (*1* + *1* / *real-of-nat k*)))
by (*subst exp-setsum* [*symmetric*]) (*simp-all add*: *setsum-right-distrib*)
also have (∑ *k*=*1*..*n*. *ln* (*1* + *1* / *of-nat k*) :: *real*) = *ln* (∏ *k*=*1*..*n*. *1* + *1* / *real-of-nat k*)
by (*subst ln-setprod* [*symmetric*]) (*auto intro*!: *add-pos-nonneg*)
also have (∏ *k*=*1*..*n*. *1* + *1* / *of-nat k* :: *real*) = (∏ *k*=*1*..*n*. (*of-nat k* + *1*) / *of-nat k*)
by (*intro setprod.cong*) (*simp-all add*: *divide-simps*)
also have (∏ *k*=*1*..*n*. (*of-nat k* + *1*) / *of-nat k* :: *real*) = *of-nat n* + *1*
by (*induction n*) (*simp-all add*: *setprod-nat-ivl-Suc' divide-simps*)
finally show *?thesis* **..**
qed

lemma *Gamma-series-euler'*:
assumes *z*: (*z* :: *'a* :: *Gamma*) ∉ $\mathbb{Z}_{\leq 0}$
shows (λ*n*. *Gamma-series-euler' z n*) ⟶ *Gamma z*
proof (*rule Gamma-seriesI*, *rule Lim-transform-eventually*)

let *?f* = λ*n*. *fact n* ∗ *exp* (*z* ∗ *of-real* (*ln* (*of-nat n* + *1*))) / *pochhammer z* (*n* + *1*)
let *?r* = λ*n*. *?f n* / *Gamma-series z n*
let *?r′* = λ*n*. *exp* (*z* ∗ *of-real* (*ln* (*of-nat* (*Suc n*) / *of-nat n*)))
from *z* **have** *z′*: *z* ≠ *0* **by** *auto*

have *eventually* (λ*n*. *?r′ n* = *?r n*) *sequentially* **using** *eventually-gt-at-top*[*of* *0*::*nat*]
using *z* **by** (*auto simp*: *divide-simps Gamma-series-def ring-distribs exp-diff ln-div add-ac*
elim!: *eventually-mono dest*: *pochhammer-eq-0-imp-nonpos-Int*)
moreover have *?r′* ⟶ *exp* (*z* ∗ *of-real* (*ln 1*))
by (*intro tendsto-intros LIMSEQ-Suc-n-over-n*) *simp-all*
ultimately show *?r* ⟶ *1* **by** (*force dest*!: *Lim-transform-eventually*)

from *eventually-gt-at-top*[*of* *0*::*nat*]
show *eventually* (λ*n*. *?r n* = *Gamma-series-euler′ z n* / *Gamma-series z n*) *sequentially*
proof *eventually-elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
from *n z′* **have** *Gamma-series-euler′ z n* =
exp (*z* ∗ *of-real* (*ln* (*of-nat n* + *1*))) / (*z* ∗ (∏*k*=*1*..*n*. (*1* + *z* / *of-nat k*)))
by (*subst Gamma-euler′-aux1*)
(*simp-all add*: *Gamma-series-euler′-def setprod.distrib*
setprod-inversef[*symmetric*] *divide-inverse*)
also have (∏*k*=*1*..*n*. (*1* + *z* / *of-nat k*)) = *pochhammer* (*z* + *1*) *n* / *fact n*
by (*cases n*) (*simp-all add*: *pochhammer-def fact-altdef setprod-shift-bounds-cl-Suc-ivl*
setprod-dividef[*symmetric*] *divide-simps add-ac*)
also have *z* ∗ ... = *pochhammer z* (*Suc n*) / *fact n* **by** (*simp add*: *pochhammer-rec*)
finally show *?r n* = *Gamma-series-euler′ z n* / *Gamma-series z n* **by** *simp*
qed
qed

end

57.9.2 Weierstrass form

definition *Gamma-series-weierstrass* :: *′a* :: {*banach*,*real-normed-field*} ⇒ *nat* ⇒ *′a* **where**
Gamma-series-weierstrass z n =
exp (−*euler-mascheroni* ∗ *z*) / *z* ∗ (∏*k*=*1*..*n*. *exp* (*z* / *of-nat k*) / (*1* + *z* / *of-nat k*))

definition *rGamma-series-weierstrass* :: *′a* :: {*banach*,*real-normed-field*} ⇒ *nat* ⇒ *′a* **where**
rGamma-series-weierstrass z n =
exp (*euler-mascheroni* ∗ *z*) ∗ *z* ∗ (∏*k*=*1*..*n*. (*1* + *z* / *of-nat k*) ∗ *exp* (−*z* / *of-nat k*))

lemma *Gamma-series-weierstrass-nonpos-Ints*:
eventually (λ*k*. *Gamma-series-weierstrass* (− *of-nat n*) *k* = *0*) *sequentially*
using *eventually-ge-at-top*[*of n*] **by** *eventually-elim* (*auto simp*: *Gamma-series-weierstrass-def*)

lemma *rGamma-series-weierstrass-nonpos-Ints*:
eventually (λ*k*. *rGamma-series-weierstrass* (− *of-nat n*) *k* = *0*) *sequentially*
using *eventually-ge-at-top*[*of n*] **by** *eventually-elim* (*auto simp*: *rGamma-series-weierstrass-def*)

lemma *Gamma-weierstrass-complex*: *Gamma-series-weierstrass z* ⟶ *Gamma*
(*z* :: *complex*)
proof (*cases z* ∈ ℤ≤0)
case *True*
then obtain *n* **where** *z* = − *of-nat n* **by** (*elim nonpos-Ints-cases'*)
also from *True* **have** *Gamma-series-weierstrass* ... ⟶ *Gamma z*
by (*simp add*: *tendsto-cong*[*OF Gamma-series-weierstrass-nonpos-Ints*] *Gamma-nonpos-Int*)
finally show *?thesis* **.**
next
case *False*
hence *z*: *z* ≠ *0* **by** *auto*
let *?f* = (λ*x*. ∏ *x* = *Suc 0..x*. *exp* (*z* / *of-nat x*) / (*1* + *z* / *of-nat x*))
have *A*: *exp* (*ln* (*1* + *z* / *of-nat n*)) = (*1* + *z* / *of-nat n*) **if** *n* ≥ *1* **for** *n* :: *nat*
using *False that* **by** (*subst exp-Ln*) (*auto simp*: *field-simps dest*!: *plus-of-nat-eq-0-imp*)
have (λ*n*. ∑ *k*=*1..n*. *z* / *of-nat k* − *ln* (*1* + *z* / *of-nat k*)) ⟶ *ln-Gamma z*
+ *euler-mascheroni* ∗ *z* + *ln z*
using *ln-Gamma-series'-aux*[*OF False*]
by (*simp only*: *atLeastLessThanSuc-atLeastAtMost* [*symmetric*] *One-nat-def*
setsum-shift-bounds-Suc-ivl sums-def atLeast0LessThan)
from *tendsto-exp*[*OF this*] *False z* **have** *?f* ⟶ *z* ∗ *exp* (*euler-mascheroni* ∗
z) ∗ *Gamma z*
by (*simp add*: *exp-add exp-setsum exp-diff mult-ac Gamma-complex-altdef A*)
from *tendsto-mult*[*OF tendsto-const*[*of exp* (−*euler-mascheroni* ∗ *z*) / *z*] *this*] *z*
show *Gamma-series-weierstrass z* ⟶ *Gamma z*
by (*simp add*: *exp-minus divide-simps Gamma-series-weierstrass-def* [*abs-def*])
qed

lemma *tendsto-complex-of-real-iff*: ((λ*x*. *complex-of-real* (*f x*)) ⟶ *of-real c*) *F*
= (*f* ⟶ *c*) *F*
by (*rule tendsto-of-real-iff*)

lemma *Gamma-weierstrass-real*: *Gamma-series-weierstrass x* ⟶ *Gamma* (*x*
:: *real*)
using *Gamma-weierstrass-complex*[*of of-real x*] **unfolding** *Gamma-series-weierstrass-def*[*abs-def*]
by (*subst tendsto-complex-of-real-iff* [*symmetric*])
(*simp-all add*: *exp-of-real*[*symmetric*] *Gamma-complex-of-real*)

lemma *rGamma-weierstrass-complex*: *rGamma-series-weierstrass z* ⟶ *rGamma*
(*z* :: *complex*)
proof (*cases z* ∈ ℤ≤0)
case *True*

then obtain *n* **where** $z = -$ *of-nat n* **by** (*elim nonpos-Ints-cases'*)
also from *True* **have** *rGamma-series-weierstrass* ... $\longrightarrow$ *rGamma z*
by (*simp add*: *tendsto-cong*[*OF rGamma-series-weierstrass-nonpos-Ints*] *rGamma-nonpos-Int*)
finally show *?thesis* **.**
next
case *False*
have *rGamma-series-weierstrass z* = (λ*n*. *inverse* (*Gamma-series-weierstrass z n*))
by (*simp add*: *rGamma-series-weierstrass-def*[*abs-def*] *Gamma-series-weierstrass-def exp-minus divide-inverse setprod-inversef*[*symmetric*] *mult-ac*)
also from *False* **have** ... $\longrightarrow$ *inverse* (*Gamma z*)
by (*intro tendsto-intros Gamma-weierstrass-complex*) (*simp add*: *Gamma-eq-zero-iff*)
finally show *?thesis* **by** (*simp add*: *Gamma-def*)
qed

57.9.3 Binomial coefficient form

lemma *Gamma-binomial*:
(λ*n*. ((*z* + *of-nat n*) *gchoose n*) ∗ *exp* (−*z* ∗ *of-real* (*ln* (*of-nat n*)))) $\longrightarrow$ *rGamma* (*z*+*1*)
proof (*cases z* = *0*)
case *False*
show *?thesis*
proof (*rule Lim-transform-eventually*)
let *?powr* = λ*a b*. *exp* (*b* ∗ *of-real* (*ln* (*of-nat a*)))
show *eventually* (λ*n*. *rGamma-series z n* / *z* =
((*z* + *of-nat n*) *gchoose n*) ∗ *?powr n* (−*z*)) *sequentially*
proof (*intro always-eventually allI*)
fix *n* :: *nat*
from *False* **have** ((*z* + *of-nat n*) *gchoose n*) = *pochhammer z* (*Suc n*) / *z* / *fact n*
by (*simp add*: *gbinomial-pochhammer' pochhammer-rec*)
also have *pochhammer z* (*Suc n*) / *z* / *fact n* ∗ *?powr n* (−*z*) = *rGamma-series z n* / *z*
by (*simp add*: *rGamma-series-def divide-simps exp-minus*)
finally show *rGamma-series z n* / *z* = ((*z* + *of-nat n*) *gchoose n*) ∗ *?powr n* (−*z*) **..**
qed

from *False* **have** (λ*n*. *rGamma-series z n* / *z*) $\longrightarrow$ *rGamma z* / *z* **by** (*intro tendsto-intros*)
also from *False* **have** *rGamma z* / *z* = *rGamma* (*z* + *1*) **using** *rGamma-plus1*[*of z*]
by (*simp add*: *field-simps*)
finally show (λ*n*. *rGamma-series z n* / *z*) $\longrightarrow$ *rGamma* (*z*+*1*) **.**
qed
qed (*simp-all add*: *binomial-gbinomial* [*symmetric*])

lemma *fact-binomial-limit*:

(λn. *of-nat* ((k + n) *choose* n) / *of-nat* (n ^ k) :: 'a :: *Gamma*) ⟶ *1* / *fact* k
proof (*rule Lim-transform-eventually*)
have (λn. *of-nat* ((k + n) *choose* n) / *of-real* (*exp* (*of-nat* k * *ln* (*real-of-nat* n))))
⟶ *1* / *Gamma* (*of-nat* (*Suc* k) :: 'a) (**is** *?f* ⟶ -)
using *Gamma-binomial*[*of of-nat* k :: 'a]
by (*simp add*: *binomial-gbinomial add-ac Gamma-def divide-simps exp-of-real* [*symmetric*] *exp-minus*)
also have *Gamma* (*of-nat* (*Suc* k)) = *fact* k **by** (*rule Gamma-fact*)
finally show *?f* ⟶ *1* / *fact* k **.**

show *eventually* (λn. *?f* n = *of-nat* ((k + n) *choose* n) / *of-nat* (n ^ k)) *sequentially*
using *eventually-gt-at-top*[*of* 0::*nat*]
proof *eventually-elim*
fix n :: *nat* **assume** n: n > 0
from n **have** *exp* (*real-of-nat* k * *ln* (*real-of-nat* n)) = *real-of-nat* (n^k)
by (*simp add*: *exp-of-nat-mult*)
thus *?f* n = *of-nat* ((k + n) *choose* n) / *of-nat* (n ^ k) **by** *simp*
qed
qed

lemma *binomial-asymptotic*:
(λn. *of-nat* ((k + n) *choose* n) / (*of-nat* (n ^ k) / *fact* k) :: 'a :: *Gamma*) ⟶ *1*
using *tendsto-mult*[*OF fact-binomial-limit*[*of* k] *tendsto-const*[*of fact* k :: 'a]] **by** *simp*

57.10 The Weierstra product formula for the sine

lemma *sin-product-formula-complex*:
fixes z :: *complex*
shows (λn. *of-real pi* * z * (∏ k=1..n. 1 − z^2 / *of-nat* k^2)) ⟶ *sin* (*of-real pi* * z)
proof −
let *?f* = *rGamma-series-weierstrass*
have (λn. (− *of-real pi* * *inverse* z) * (*?f* z n * *?f* (− z) n))
⟶ (− *of-real pi* * *inverse* z) * (*rGamma* z * *rGamma* (− z))
by (*intro tendsto-intros rGamma-weierstrass-complex*)
also have (λn. (− *of-real pi* * *inverse* z) * (*?f* z n * *?f* (−z) n)) =
(λn. *of-real pi* * z * (∏ k=1..n. 1 − z^2 / *of-nat* k ^ 2))
proof
fix n :: *nat*
have (− *of-real pi* * *inverse* z) * (*?f* z n * *?f* (−z) n) =
of-real pi * z * (∏ k=1..n. (*of-nat* k − z) * (*of-nat* k + z) / *of-nat* k ^ 2)
by (*simp add*: *rGamma-series-weierstrass-def mult-ac exp-minus*
divide-simps setprod.distrib[*symmetric*] *power2-eq-square*)

also have $(\prod k{=}1..n.\ (\mathit{of\text{-}nat}\ k - z) * (\mathit{of\text{-}nat}\ k + z) \ /\ \mathit{of\text{-}nat}\ k\ \hat{}\ 2) =$
$(\prod k{=}1..n.\ 1 - z\hat{}2\ /\ \mathit{of\text{-}nat}\ k\ \hat{}\ 2)$
by (*intro setprod.cong*) (*simp-all add*: *power2-eq-square field-simps*)
finally show $(-\ \mathit{of\text{-}real}\ \mathit{pi} * \mathit{inverse}\ z) * (?f\ z\ n * ?f\ (-z)\ n) = \mathit{of\text{-}real}\ \mathit{pi} * z$
$* \ldots$
by (*simp add*: *divide-simps*)
qed
also have $(-\ \mathit{of\text{-}real}\ \mathit{pi} * \mathit{inverse}\ z) * (\mathit{rGamma}\ z * \mathit{rGamma}\ (-\ z)) = \mathit{sin}$
$(\mathit{of\text{-}real}\ \mathit{pi} * z)$
by (*subst rGamma-reflection-complex'*) (*simp add*: *divide-simps*)
finally show *?thesis* **.**
qed

lemma *sin-product-formula-real*:
$(\lambda n.\ \mathit{pi} * (x{::}\mathit{real}) * (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ \mathit{of\text{-}nat}\ k\hat{}2)) \longrightarrow \mathit{sin}\ (\mathit{pi} * x)$
proof −
from *sin-product-formula-complex*[*of of-real x*]
have $(\lambda n.\ \mathit{of\text{-}real}\ \mathit{pi} * \mathit{of\text{-}real}\ x * (\prod k{=}1..n.\ 1 - (\mathit{of\text{-}real}\ x)\hat{}2\ /\ (\mathit{of\text{-}nat}\ k)\hat{}2))$
$\longrightarrow \mathit{sin}\ (\mathit{of\text{-}real}\ \mathit{pi} * \mathit{of\text{-}real}\ x :: \mathit{complex})$ (**is** $?f \longrightarrow ?y$) **.**
also have $?f = (\lambda n.\ \mathit{of\text{-}real}\ (\mathit{pi} * x * (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ (\mathit{of\text{-}nat}\ k\hat{}2))))$ **by** *simp*
also have $?y = \mathit{of\text{-}real}\ (\mathit{sin}\ (\mathit{pi} * x))$ **by** (*simp only*: *sin-of-real* [*symmetric*] *of-real-mult*)
finally show *?thesis* **by** (*subst* (*asm*) *tendsto-of-real-iff*)
qed

lemma *sin-product-formula-real'*:
assumes $x \neq (0{::}\mathit{real})$
shows $(\lambda n.\ (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ \mathit{of\text{-}nat}\ k\hat{}2)) \longrightarrow \mathit{sin}\ (\mathit{pi} * x)\ /\ (\mathit{pi} * x)$
using *tendsto-divide*[*OF sin-product-formula-real*[*of x*] *tendsto-const*[*of pi* * *x*]] *assms*
by *simp*

57.11 The Solution to the Basel problem

theorem *inverse-squares-sums*: $(\lambda n.\ 1\ /\ (n + 1)^2)\ \mathit{sums}\ (\mathit{pi}^2\ /\ 6)$
proof −
def $P \equiv \lambda x\ n.\ (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ \mathit{of\text{-}nat}\ k\hat{}2 :: \mathit{real})$
def $K \equiv \sum n.\ \mathit{inverse}\ (\mathit{real\text{-}of\text{-}nat}\ (\mathit{Suc}\ n))\hat{}2$
def $f \equiv \lambda x.\ \sum n.\ P\ x\ n\ /\ \mathit{of\text{-}nat}\ (\mathit{Suc}\ n)\hat{}2$
def $g \equiv \lambda x.\ (1 - \mathit{sin}\ (\mathit{pi} * x)\ /\ (\mathit{pi} * x))$

have *sums*: $(\lambda n.\ P\ x\ n\ /\ \mathit{of\text{-}nat}\ (\mathit{Suc}\ n)\hat{}2)\ \mathit{sums}\ (\mathit{if}\ x = 0\ \mathit{then}\ K\ \mathit{else}\ g\ x\ /\ x\hat{}2)$ **for** x
proof (*cases* $x = 0$)
assume x: $x = 0$
have *summable* $(\lambda n.\ \mathit{inverse}\ ((\mathit{real\text{-}of\text{-}nat}\ (\mathit{Suc}\ n))^2))$
using *inverse-power-summable*[*of 2*] **by** (*subst summable-Suc-iff*) *simp*
thus *?thesis* **by** (*simp add*: *x g-def P-def K-def inverse-eq-divide power-divide*

summable-sums)
next
assume *x*: $x \neq 0$
have $(\lambda n.\ P\ x\ n - P\ x\ (Suc\ n))$ *sums* $(P\ x\ 0 - sin\ (pi * x) / (pi * x))$
unfolding *P-def* **using** *x* **by** (*intro telescope-sums' sin-product-formula-real'*)
also have $(\lambda n.\ P\ x\ n - P\ x\ (Suc\ n)) = (\lambda n.\ (x\hat{}2 / of\text{-}nat\ (Suc\ n)\hat{}2) * P\ x\ n)$
unfolding *P-def* **by** (*simp add*: *setprod-nat-ivl-Suc' algebra-simps*)
also have $P\ x\ 0 = 1$ **by** (*simp add*: *P-def*)
finally have $(\lambda n.\ x^2 / (of\text{-}nat\ (Suc\ n))^2 * P\ x\ n)$ *sums* $(1 - sin\ (pi * x) / (pi * x))$ **.**
from *sums-divide*[*OF this*, *of x^2*] *x* **show** *?thesis* **unfolding** *g-def* **by** *simp*
qed

have *continuous-on* (*ball 0 1*) *f*
proof (*rule uniform-limit-theorem*; (*intro always-eventually allI*)*?*)
show *uniform-limit* (*ball 0 1*) $(\lambda n\ x.\ \sum k{<}n.\ P\ x\ k / of\text{-}nat\ (Suc\ k)\hat{}2)$ *f sequentially*
proof (*unfold f-def*, *rule weierstrass-m-test*)
fix *n* :: *nat* **and** *x* :: *real* **assume** *x*: $x \in ball\ 0\ 1$
{
fix *k* :: *nat* **assume** *k*: $k \geq 1$
from *x* **have** $x\hat{}2 < 1$ **by** (*auto simp*: *dist-0-norm abs-square-less-1*)
also from *k* **have** $\ldots \leq of\text{-}nat\ k\hat{}2$ **by** *simp*
finally have $(1 - x\hat{}2 / of\text{-}nat\ k\hat{}2) \in \{0..1\}$ **using** *k*
by (*simp-all add*: *field-simps del*: *of-nat-Suc*)
}
hence $(\prod k{=}1..n.\ abs\ (1 - x\hat{}2 / of\text{-}nat\ k\hat{}2)) \leq (\prod k{=}1..n.\ 1)$ **by** (*intro setprod-mono*) *simp*
thus *norm* $(P\ x\ n / (of\text{-}nat\ (Suc\ n)\hat{}2)) \leq 1 / of\text{-}nat\ (Suc\ n)\hat{}2$
unfolding *P-def* **by** (*simp add*: *field-simps abs-setprod del*: *of-nat-Suc*)
qed (*subst summable-Suc-iff*, *insert inverse-power-summable*[*of 2*], *simp add*: *inverse-eq-divide*)
qed (*auto simp*: *P-def intro*!: *continuous-intros*)
hence *isCont f 0* **by** (*subst* (*asm*) *continuous-on-eq-continuous-at*) *simp-all*
hence $(f - 0 \to f\ 0)$ **by** (*simp add*: *isCont-def*)
also have $f\ 0 = K$ **unfolding** *f-def P-def K-def* **by** (*simp add*: *inverse-eq-divide power-divide*)
finally have $f - 0 \to K$ **.**

moreover have $f - 0 \to pi\hat{}2 / 6$
proof (*rule Lim-transform-eventually*)
def $f' \equiv \lambda x.\ \sum n.\ -\ sin\text{-}coeff\ (n{+}3) * pi\ \hat{}\ (n{+}2) * x\hat{}n$
have *eventually* $(\lambda x.\ x \neq (0{::}real))\ (at\ 0)$
by (*auto simp add*: *eventually-at intro*!: *exI*[*of - 1*])
thus *eventually* $(\lambda x.\ f'\ x = f\ x)\ (at\ 0)$
proof *eventually-elim*
fix *x* :: *real* **assume** *x*: $x \neq 0$
have *sin-coeff 1* $= (1 :: real)$ *sin-coeff 2* $= (0{::}real)$ **by** (*simp-all add*:

```
sin-coeff-def)
    with sums-split-initial-segment[OF sums-minus[OF sin-converges], of 3 pi*x]
    have (λn. − (sin-coeff (n+3) * (pi*x) ^ (n+3))) sums (pi * x − sin (pi*x))
      by (simp add: eval-nat-numeral)
    from sums-divide[OF this, of x^3 * pi] x
      have (λn. − (sin-coeff (n+3) * pi^(n+2) * x^n)) sums ((1 − sin (pi*x)
/ (pi*x)) / x^2)
      by (simp add: divide-simps eval-nat-numeral power-mult-distrib mult-ac)
    with x have (λn. − (sin-coeff (n+3) * pi^(n+2) * x^n)) sums (g x / x^2)
      by (simp add: g-def)
    hence f' x = g x / x^2 by (simp add: sums-iff f'-def)
    also have ... = f x using sums[of x] x by (simp add: sums-iff g-def f-def)
    finally show f' x = f x .
  qed

  have isCont f' 0 unfolding f'-def
  proof (intro isCont-powser-converges-everywhere)
    fix x :: real show summable (λn. −sin-coeff (n+3) * pi^(n+2) * x^n)
    proof (cases x = 0)
      assume x: x ≠ 0
    from summable-divide[OF sums-summable[OF sums-split-initial-segment[OF
          sin-converges[of pi*x]], of 3], of −pi*x^3] x
        show ?thesis by (simp add: mult-ac power-mult-distrib divide-simps
eval-nat-numeral)
    qed (simp only: summable-0-powser)
  qed
  hence f' − 0 → f' 0 by (simp add: isCont-def)
  also have f' 0 = pi * pi / fact 3 unfolding f'-def
    by (subst powser-zero) (simp add: sin-coeff-def)
  finally show f' − 0 → pi^2 / 6 by (simp add: eval-nat-numeral)
 qed

 ultimately have K = pi^2 / 6 by (rule LIM-unique)
 moreover from inverse-power-summable[of 2]
   have summable (λn. (inverse (real-of-nat (Suc n)))^2)
   by (subst summable-Suc-iff) (simp add: power-inverse)
 ultimately show ?thesis unfolding K-def
   by (auto simp add: sums-iff power-divide inverse-eq-divide)
qed


end
theory Multivariate-Analysis
imports
  Fashoda
  Extended-Real-Limits
  Determinants
  Ordered-Euclidean-Space
  Complex-Analysis-Basics
```

Bounded-Continuous-Function
Weierstrass
Cauchy-Integral-Thm
Generalised-Binomial-Theorem
Gamma
begin

end

58 polynomial functions: extremal behaviour and root counts

theory *PolyRoots*
imports *Complex-Main*
begin

58.1 Geometric progressions

lemma *setsum-gp-basic*:
fixes x :: $'a$::{*comm-ring*,*monoid-mult*}
shows $(1 - x) * (\sum i{\leq}n.\ x\hat{}i) = 1 - x\hat{}Suc\ n$
by (*simp only*: *one-diff-power-eq* [*of Suc n x*] *lessThan-Suc-atMost*)

lemma *setsum-gp0*:
fixes x :: $'a$::{*comm-ring*,*division-ring*}
shows $(\sum i{\leq}n.\ x\hat{}i) = (if\ x = 1\ then\ of\text{-}nat(n + 1)\ else\ (1 - x\hat{}Suc\ n) / (1 - x))$
using *setsum-gp-basic*[*of x n*]
by (*simp add*: *mult.commute divide-simps*)

lemma *setsum-power-add*:
fixes x :: $'a$::{*comm-ring*,*monoid-mult*}
shows $(\sum i{\in}I.\ x\hat{}(m+i)) = x\hat{}m * (\sum i{\in}I.\ x\hat{}i)$
by (*simp add*: *setsum-right-distrib power-add*)

lemma *setsum-power-shift*:
fixes x :: $'a$::{*comm-ring*,*monoid-mult*}
assumes $m \leq n$
shows $(\sum i{=}m..n.\ x\hat{}i) = x\hat{}m * (\sum i{\leq}n-m.\ x\hat{}i)$
proof −
have $(\sum i{=}m..n.\ x\hat{}i) = x\hat{}m * (\sum i{=}m..n.\ x\hat{}(i-m))$
by (*simp add*: *setsum-right-distrib power-add* [*symmetric*])
also have $(\sum i{=}m..n.\ x\hat{}(i-m)) = (\sum i{\leq}n-m.\ x\hat{}i)$
using ‹$m \leq n$› **by** (*intro setsum.reindex-bij-witness*[**where** $j{=}\lambda i.\ i - m$ **and** $i{=}\lambda i.\ i + m$]) *auto*
finally show *?thesis* **.**
qed

lemma *setsum-gp-multiplied*:
fixes x :: *'a*::{*comm-ring*,*monoid-mult*}
assumes $m \leq n$
shows $(1 - x) * (\sum i{=}m..n.\ x\hat{}i) = x\hat{}m - x\hat{}Suc\ n$
proof −
have $(1 - x) * (\sum i{=}m..n.\ x\hat{}i) = x\hat{}m * (1 - x) * (\sum i{\leq}n{-}m.\ x\hat{}i)$
by (*metis mult.assoc mult.commute assms setsum-power-shift*)
also have ... $= x\hat{}m * (1 - x\hat{}Suc(n{-}m))$
by (*metis mult.assoc setsum-gp-basic*)
also have ... $= x\hat{}m - x\hat{}Suc\ n$
using *assms*
by (*simp add*: *algebra-simps*) (*metis le-add-diff-inverse power-add*)
finally show *?thesis* **.**
qed

lemma *setsum-gp*:
fixes x :: *'a*::{*comm-ring*,*division-ring*}
shows $(\sum i{=}m..n.\ x\hat{}i) =$
$(if\ n < m\ then\ 0$
$\ else\ if\ x = 1\ then\ of\text{-}nat((n + 1) - m)$
$\ else\ (x\hat{}m - x\hat{}Suc\ n) / (1 - x))$
using *setsum-gp-multiplied* [*of m n x*]
apply *auto*
by (*metis eq-iff-diff-eq-0 mult.commute nonzero-divide-eq-eq*)

lemma *setsum-gp-offset*:
fixes x :: *'a*::{*comm-ring*,*division-ring*}
shows $(\sum i{=}m..m{+}n.\ x\hat{}i) =$
$(if\ x = 1\ then\ of\text{-}nat\ n + 1\ else\ x\hat{}m * (1 - x\hat{}Suc\ n) / (1 - x))$
using *setsum-gp* [*of x m m+n*]
by (*auto simp*: *power-add algebra-simps*)

lemma *setsum-gp-strict*:
fixes x :: *'a*::{*comm-ring*,*division-ring*}
shows $(\sum i{<}n.\ x\hat{}i) = (if\ x = 1\ then\ of\text{-}nat\ n\ else\ (1 - x\hat{}n) / (1 - x))$
by (*induct n*) (*auto simp*: *algebra-simps divide-simps*)

58.2 Basics about polynomial functions: extremal behaviour and root counts.

lemma *sub-polyfun*:
fixes x :: *'a*::{*comm-ring*,*monoid-mult*}
shows $(\sum i{\leq}n.\ a\ i * x\hat{}i) - (\sum i{\leq}n.\ a\ i * y\hat{}i) =$
$(x - y) * (\sum j{<}n.\ \sum k{=}\ Suc\ j..n.\ a\ k * y\hat{}(k - Suc\ j) * x\hat{}j)$
proof −
have $(\sum i{\leq}n.\ a\ i * x\hat{}i) - (\sum i{\leq}n.\ a\ i * y\hat{}i) =$
$(\sum i{\leq}n.\ a\ i * (x\hat{}i - y\hat{}i))$
by (*simp add*: *algebra-simps setsum-subtractf* [*symmetric*])
also have ... $= (\sum i{\leq}n.\ a\ i * (x - y) * (\sum j{<}i.\ y\hat{}(i - Suc\ j) * x\hat{}j))$

```
    by (simp add: power-diff-sumr2 ac-simps)
  also have ... = (x - y) * (∑i≤n. (∑j<i. a i * y^(i - Suc j) * x^j))
    by (simp add: setsum-right-distrib ac-simps)
  also have ... = (x - y) * (∑j<n. (∑i=Suc j..n. a i * y^(i - Suc j) * x^j))
    by (simp add: nested-setsum-swap')
  finally show ?thesis .
qed

lemma sub-polyfun-alt:
  fixes x :: 'a::{comm-ring,monoid-mult}
  shows   (∑i≤n. a i * x^i) - (∑i≤n. a i * y^i) =
          (x - y) * (∑j<n. ∑k<n-j. a (j+k+1) * y^k * x^j)
proof -
  { fix j
    have (∑k = Suc j..n. a k * y^(k - Suc j) * x^j) =
         (∑k <n - j. a (Suc (j + k)) * y^k * x^j)
      by (rule setsum.reindex-bij-witness[where i=λi. i + Suc j and j=λi. i -
Suc j]) auto }
  then show ?thesis
    by (simp add: sub-polyfun)
qed

lemma polyfun-linear-factor:
  fixes a :: 'a::{comm-ring,monoid-mult}
  shows  ∃b. ∀z. (∑i≤n. c i * z^i) =
                 (z-a) * (∑i<n. b i * z^i) + (∑i≤n. c i * a^i)
proof -
  { fix z
    have (∑i≤n. c i * z^i) - (∑i≤n. c i * a^i) =
         (z - a) * (∑j<n. (∑k = Suc j..n. c k * a^(k - Suc j)) * z^j)
      by (simp add: sub-polyfun setsum-left-distrib)
    then have (∑i≤n. c i * z^i) =
         (z - a) * (∑j<n. (∑k = Suc j..n. c k * a^(k - Suc j)) * z^j)
         + (∑i≤n. c i * a^i)
      by (simp add: algebra-simps) }
  then show ?thesis
    by (intro exI allI)
qed

lemma polyfun-linear-factor-root:
  fixes a :: 'a::{comm-ring,monoid-mult}
  assumes (∑i≤n. c i * a^i) = 0
  shows  ∃b. ∀z. (∑i≤n. c i * z^i) = (z-a) * (∑i<n. b i * z^i)
  using polyfun-linear-factor [of c n a] assms
  by simp

lemma adhoc-norm-triangle: a + norm(y) ≤ b ==> norm(x) ≤ a ==> norm(x
+ y) ≤ b
  by (metis norm-triangle-mono order.trans order-refl)
```

lemma *polyfun-extremal-lemma*:
fixes $c :: nat \Rightarrow 'a$::*real-normed-div-algebra*
assumes $e > 0$
shows $\exists M.\ \forall z.\ M \leq norm\ z \longrightarrow norm(\sum i{\leq}n.\ c\ i * z\hat{}i) \leq e * norm(z)\ \hat{}$ *Suc n*
proof (*induction n*)
case *0*
show *?case*
by (*rule exI* [**where** *x*=*norm* (*c 0*) / *e*]) (*auto simp*: *mult.commute pos-divide-le-eq assms*)
next
case (*Suc n*)
then obtain *M* **where** *M*: $\forall z.\ M \leq norm\ z \longrightarrow norm\ (\sum i{\leq}n.\ c\ i * z\hat{}i) \leq e * norm\ z\ \hat{}\ Suc\ n$ **..**
show *?case*
proof (*rule exI* [**where** *x*=*max 1* (*max M* ((*e* + *norm*(*c*(*Suc n*))) / *e*))], *clarify*)
fix *z*::'*a*
assume *max 1* (*max M* ((*e* + *norm* (*c* (*Suc n*))) / *e*)) ≤ *norm z*
then have *norm1*: *0* < *norm z M* ≤ *norm z* (*e* + *norm* (*c* (*Suc n*))) / *e* ≤ *norm z*
by *auto*
then have *norm2*: (*e* + *norm* (*c* (*Suc n*))) ≤ *e* * *norm z* (*norm z* * *norm z* ^ *n*) > *0*
apply (*metis assms less-divide-eq mult.commute not-le*)
using *norm1* **apply** (*metis mult-pos-pos zero-less-power*)
done
have *e* * (*norm z* * *norm z* ^ *n*) + *norm* (*c* (*Suc n*) * (*z* * *z* ^ *n*)) =
(*e* + *norm* (*c* (*Suc n*))) * (*norm z* * *norm z* ^ *n*)
by (*simp add*: *norm-mult norm-power algebra-simps*)
also have ... ≤ (*e* * *norm z*) * (*norm z* * *norm z* ^ *n*)
using *norm2* **by** (*metis real-mult-le-cancel-iff1*)
also have ... = *e* * (*norm z* * (*norm z* * *norm z* ^ *n*))
by (*simp add*: *algebra-simps*)
finally have *e* * (*norm z* * *norm z* ^ *n*) + *norm* (*c* (*Suc n*) * (*z* * *z* ^ *n*))
≤ *e* * (*norm z* * (*norm z* * *norm z* ^ *n*)) **.**
then show *norm* ($\sum i{\leq}Suc\ n.\ c\ i * z\hat{}i$) ≤ *e* * *norm z* ^ *Suc* (*Suc n*) **using** *M norm1*
by (*drule-tac x*=*z* **in** *spec*) (*auto simp*: *intro*!: *adhoc-norm-triangle*)
qed
qed

lemma *norm-lemma-xy*: ⟦|*b*| + *1* ≤ *norm*(*y*) − *a*; *norm*(*x*) ≤ *a*⟧ ⟹ *b* ≤ *norm*(*x* + *y*)
by (*metis abs-add-one-not-less-self add.commute diff-le-eq dual-order.trans le-less-linear norm-diff-ineq*)

lemma *polyfun-extremal*:
fixes $c :: nat \Rightarrow 'a$::*real-normed-div-algebra*

```
  assumes ∃k. k ≠ 0 ∧ k ≤ n ∧ c k ≠ 0
    shows eventually (λz. norm(∑ i≤n. c i * z^i) ≥ B) at-infinity
using assms
proof (induction n)
  case 0 then show ?case
    by simp
next
  case (Suc n)
  show ?case
  proof (cases c (Suc n) = 0)
    case True
    with Suc show ?thesis
      by auto (metis diff-is-0-eq diffs0-imp-equal less-Suc-eq-le not-less-eq)
  next
    case False
    with polyfun-extremal-lemma [of norm(c (Suc n)) / 2 c n]
    obtain M where M: ⋀z. M ≤ norm z ⟹
            norm (∑ i≤n. c i * z^i) ≤ norm (c (Suc n)) / 2 * norm z ^ Suc n
      by auto
    show ?thesis
    unfolding eventually-at-infinity
    proof (rule exI [where x=max M (max 1 ((|B| + 1) / (norm (c (Suc n)) /
2)))], clarsimp)
      fix z::'a
      assume les: M ≤ norm z  1 ≤ norm z  (|B| * 2 + 2) / norm (c (Suc n)) ≤
norm z
      then have |B| * 2 + 2 ≤ norm z * norm (c (Suc n))
        by (metis False pos-divide-le-eq zero-less-norm-iff)
      then have |B| * 2 + 2 ≤ norm z ^ (Suc n) * norm (c (Suc n))
      by (metis ‹1 ≤ norm z› order.trans mult-right-mono norm-ge-zero self-le-power
zero-less-Suc)
      then show B ≤ norm ((∑ i≤n. c i * z^i) + c (Suc n) * (z * z ^ n)) using
M les
        apply auto
       apply (rule norm-lemma-xy [where a = norm (c (Suc n)) * norm z ^ (Suc
n) / 2])
        apply (simp-all add: norm-mult norm-power)
        done
    qed
  qed
qed

lemma polyfun-rootbound:
 fixes c :: nat ⇒ 'a::{comm-ring,real-normed-div-algebra}
 assumes ∃k. k ≤ n ∧ c k ≠ 0
   shows finite {z. (∑ i≤n. c i * z^i) = 0} ∧ card {z. (∑ i≤n. c i * z^i) = 0}
≤ n
using assms
proof (induction n arbitrary: c)
```

case (*Suc n*) **show** *?case*
proof (*cases* {*z*. (∑ *i*≤*Suc n*. *c i* ∗ *z*^*i*) = *0*} = {})
case *False*
then obtain *a* **where** *a*: (∑ *i*≤*Suc n*. *c i* ∗ *a*^*i*) = *0*
by *auto*
from *polyfun-linear-factor-root* [*OF this*]
obtain *b* **where** ⋀*z*. (∑ *i*≤*Suc n*. *c i* ∗ *z*^*i*) = (*z* − *a*) ∗ (∑ *i*< *Suc n*. *b i* ∗ *z*^*i*)
by *auto*
then have *b*: ⋀*z*. (∑ *i*≤*Suc n*. *c i* ∗ *z*^*i*) = (*z* − *a*) ∗ (∑ *i*≤*n*. *b i* ∗ *z*^*i*)
by (*metis lessThan-Suc-atMost*)
then have *ins-ab*: {*z*. (∑ *i*≤*Suc n*. *c i* ∗ *z*^*i*) = *0*} = *insert a* {*z*. (∑ *i*≤*n*. *b i* ∗ *z*^*i*) = *0*}
by *auto*
have *c0*: *c 0* = − (*a* ∗ *b 0*) **using** *b* [*of 0*]
by *simp*
then have *extr-prem*: ~ (∃ *k*≤*n*. *b k* ≠ *0*) ⟹ ∃ *k*. *k* ≠ *0* ∧ *k* ≤ *Suc n* ∧ *c k* ≠ *0*
by (*metis Suc.prems le0 minus-zero mult-zero-right*)
have ∃ *k*≤*n*. *b k* ≠ *0*
apply (*rule ccontr*)
using *polyfun-extremal* [*OF extr-prem*, *of 1*]
apply (*auto simp*: *eventually-at-infinity b simp del*: *setsum-atMost-Suc*)
apply (*drule-tac x*=*of-real ba* **in** *spec*, *simp*)
done
then show *?thesis* **using** *Suc.IH* [*of b*] *ins-ab*
by (*auto simp*: *card-insert-if*)
qed *simp*
qed *simp*

corollary
fixes *c* :: *nat* ⇒ *'a*::{*comm-ring*,*real-normed-div-algebra*}
assumes ∃ *k*. *k* ≤ *n* ∧ *c k* ≠ *0*
shows *polyfun-rootbound-finite*: *finite* {*z*. (∑ *i*≤*n*. *c i* ∗ *z*^*i*) = *0*}
and *polyfun-rootbound-card*: *card* {*z*. (∑ *i*≤*n*. *c i* ∗ *z*^*i*) = *0*} ≤ *n*
using *polyfun-rootbound* [*OF assms*] **by** *auto*

lemma *polyfun-finite-roots*:
fixes *c* :: *nat* ⇒ *'a*::{*comm-ring*,*real-normed-div-algebra*}
shows *finite* {*z*. (∑ *i*≤*n*. *c i* ∗ *z*^*i*) = *0*} ⟷ (∃ *k*. *k* ≤ *n* ∧ *c k* ≠ *0*)
proof (*cases* ∃ *k*≤*n*. *c k* ≠ *0*)
case *True* **then show** *?thesis*
by (*blast intro*: *polyfun-rootbound-finite*)
next
case *False* **then show** *?thesis*
by (*auto simp*: *infinite-UNIV-char-0*)
qed

lemma *polyfun-eq-0*:

fixes $c :: nat \Rightarrow 'a::\{comm\text{-}ring,real\text{-}normed\text{-}div\text{-}algebra\}$
shows $(\forall z.\ (\sum i{\le}n.\ c\ i * z\hat{}i) = 0) \longleftrightarrow (\forall k.\ k \le n \longrightarrow c\ k = 0)$
proof (*cases* $(\forall z.\ (\sum i{\le}n.\ c\ i * z\hat{}i) = 0)$)
case *True*
then have $\sim$ *finite* $\{z.\ (\sum i{\le}n.\ c\ i * z\hat{}i) = 0\}$
by (*simp add*: *infinite-UNIV-char-0*)
with *True* **show** *?thesis*
by (*metis* (*poly-guards-query*) *polyfun-rootbound-finite*)
next
case *False*
then show *?thesis*
by *auto*
qed

lemma *polyfun-eq-const*:
fixes $c :: nat \Rightarrow 'a::\{comm\text{-}ring,real\text{-}normed\text{-}div\text{-}algebra\}$
shows $(\forall z.\ (\sum i{\le}n.\ c\ i * z\hat{}i) = k) \longleftrightarrow c\ 0 = k \wedge (\forall k.\ k \neq 0 \wedge k \le n \longrightarrow c\ k = 0)$
proof −
{fix z
have $(\sum i{\le}n.\ c\ i * z\hat{}i) = (\sum i{\le}n.\ (\mathit{if}\ i = 0\ \mathit{then}\ c\ 0 - k\ \mathit{else}\ c\ i) * z\hat{}i) + k$
by (*induct n*) *auto*
} then
have $(\forall z.\ (\sum i{\le}n.\ c\ i * z\hat{}i) = k) \longleftrightarrow (\forall z.\ (\sum i{\le}n.\ (\mathit{if}\ i = 0\ \mathit{then}\ c\ 0 - k\ \mathit{else}\ c\ i) * z\hat{}i) = 0)$
by *auto*
also have ... $\longleftrightarrow\ c\ 0 = k \wedge (\forall k.\ k \neq 0 \wedge k \le n \longrightarrow c\ k = 0)$
by (*auto simp*: *polyfun-eq-0*)
finally show *?thesis* **.**
qed

end