

Isabelle/HOL — Higher-Order Logic

April 19, 2009

Contents

1	HOL: The basis of Higher-Order Logic	21
1.1	Primitive logic	21
1.1.1	Core syntax	21
1.1.2	Additional concrete syntax	22
1.1.3	Axioms and basic definitions	24
1.1.4	Generic classes and algebraic operations	25
1.2	Fundamental rules	26
1.2.1	Equality	26
1.2.2	Congruence rules for application	27
1.2.3	Equality of booleans – iff	27
1.2.4	True	28
1.2.5	Universal quantifier	28
1.2.6	False	28
1.2.7	Negation	29
1.2.8	Implication	29
1.2.9	Existential quantifier	30
1.2.10	Conjunction	30
1.2.11	Disjunction	30
1.2.12	Classical logic	31
1.2.13	Unique existence	31
1.2.14	THE: definite description operator	32
1.2.15	Classical intro rules for disjunction and existential quantifiers	32
1.2.16	Intuitionistic Reasoning	33
1.2.17	Atomizing meta-level connectives	34
1.2.18	Atomizing elimination rules	34
1.3	Package setup	35
1.3.1	Classical Reasoner setup	35
1.3.2	Simplifier	36
1.3.3	Generic cases and induction	43
1.3.4	Coherent logic	44

1.4	Other simple lemmas and lemma duplicates	44
1.5	Basic ML bindings	44
1.6	Code generator basics – see further theory <i>Code-Setup</i>	44
1.7	Nitpick hooks	45
1.8	Legacy tactics and ML bindings	45
2	Code-Setup: Setup of code generators and related tools	45
2.1	Generic code generator foundation	45
2.2	Generic code generator preprocessor	47
2.3	Generic code generator target languages	47
2.4	SML code generator setup	48
2.5	Evaluation and normalization by evaluation	48
2.6	Quickcheck	48
3	Orderings: Abstract orderings	48
3.1	Quasi orders	49
3.2	Partial orders	50
3.3	Linear (total) orders	51
3.4	Reasoning tools setup	53
3.5	Name duplicates	56
3.6	Bounded quantifiers	57
3.7	Transitivity reasoning	58
3.8	Monotonicity, least value operator and min/max	62
3.9	Top and bottom elements	63
3.10	Dense orders	63
3.11	Wellorders	64
3.12	Order on bool	64
3.13	Order on functions	65
4	Lattices: Abstract lattices	67
4.1	Lattices	67
4.1.1	Intro and elim rules	67
4.1.2	Equational laws	69
4.2	Distributive lattices	71
4.3	Uniqueness of inf and sup	71
4.4	<i>min/max</i> on linear orders as special case of <i>op</i> \sqcap / <i>op</i> \sqcup	71
4.5	Bool as lattice	72
4.6	Fun as lattice	72
5	Set: Set theory for higher-order logic	73
5.1	Basic syntax	73
5.2	Additional concrete syntax	76
5.2.1	Bounded quantifiers	77
5.3	Rules and definitions	78

5.4	Lemmas and proof tool setup	80
5.4.1	Relating predicates and sets	80
5.4.2	Bounded quantifiers	80
5.4.3	Congruence rules	81
5.4.4	Subsets	82
5.4.5	Equality	83
5.4.6	The universal set – UNIV	83
5.4.7	The empty set	84
5.4.8	The Powerset operator – Pow	84
5.4.9	Set complement	85
5.4.10	Binary union – Un	85
5.4.11	Binary intersection – Int	85
5.4.12	Set difference	86
5.4.13	Augmenting a set – insert	86
5.4.14	Singletons, using insert	87
5.4.15	Unions of families	88
5.4.16	Intersections of families	88
5.4.17	Union	89
5.4.18	Inter	89
5.4.19	Set reasoning tools	90
5.4.20	The “proper subset” relation	91
5.5	Further set-theory lemmas	92
5.5.1	Derived rules involving subsets.	92
5.5.2	Equalities involving union, intersection, inclusion, etc.	93
5.5.3	Monotonicity of various operations	109
5.6	Inverse image of a function	111
5.6.1	Basic rules	111
5.6.2	Equations	111
5.7	Getting the Contents of a Singleton Set	113
5.8	Transitivity rules for calculational reasoning	113
5.9	Least value operator	113
5.10	Rudimentary code generation	113
5.11	Complete lattices	113
5.12	Bool as complete lattice	116
5.13	Fun as complete lattice	117
5.14	Set as lattice	117
5.15	Misc theorem and ML bindings	118
6	Typedef: HOL type definitions	118
7	Fun: Notions about functions	119
7.1	The Identity Function <i>id</i>	120
7.2	The Composition Operator $f \circ g$	120
7.3	The Forward Composition Operator <i>fcomp</i>	121

7.4	Injectivity and Surjectivity	121
7.5	Function Updating	125
7.6	<i>override-on</i>	126
7.7	<i>swap</i>	126
7.8	Proof tool setup	127
7.9	Code generator setup	127
8	Sum-Type: The Disjoint Sum of Two Types	128
8.1	Freeness Properties for <i>Inl</i> and <i>Inr</i>	129
8.2	The Disjoint Sum of Sets	130
8.3	The <i>Part</i> Primitive	130
9	Inductive: Knaster-Tarski Fixpoint Theorem and inductive definitions	131
9.1	Least and greatest fixed points	132
9.2	Proof of Knaster-Tarski Theorem using <i>lfp</i>	132
9.3	General induction rules for least fixed points	132
9.4	Proof of Knaster-Tarski Theorem using <i>gfp</i>	133
9.5	Coinduction rules for greatest fixed points	134
9.6	Even Stronger Coinduction Rule, by Martin Coen	134
9.7	Inductive predicates and sets	135
9.8	Inductive datatypes and primitive recursion	136
10	OrderedGroup: Ordered Groups	136
10.1	Semigroups and Monoids	137
10.2	Groups	139
10.3	(Partially) Ordered Groups	141
10.4	Support for reasoning about signs	143
10.5	Lattice Ordered (Abelian) Groups	148
10.6	Positive Part, Negative Part, Absolute Value	150
10.7	Tools setup	153
11	Ring-and-Field: (Ordered) Rings and Fields	153
11.1	Calculations with fractions	175
11.1.1	Special Cancellation Simprules for Division	175
11.2	Division and Unary Minus	175
11.3	Ordered Fields	176
11.4	Anti-Monotonicity of <i>inverse</i>	176
11.5	Inverses and the Number One	178
11.6	Simplification of Inequalities Involving Literal Divisors	178
11.7	Field simplification	179
11.8	Division and Signs	180
11.9	Cancellation Laws for Division	181
11.10	Division and the Number One	181

11.11	Ordering Rules for Division	182
11.12	Conditional Simplification Rules: No Case Splits	183
11.13	Reasoning about inequalities with division	184
11.14	Ordered Fields are Dense	185
11.15	Absolute Value	185
11.16	Bounds of products via negative and positive Part	187
12	Nat: Natural numbers	187
12.1	Type <i>ind</i>	187
12.2	Type <i>nat</i>	188
12.3	Arithmetic operators	189
12.3.1	Addition	191
12.3.2	Difference	191
12.3.3	Multiplication	192
12.4	Orders on <i>nat</i>	193
12.4.1	Operation definition	193
12.4.2	Introduction properties	194
12.4.3	Elimination properties	194
12.4.4	Inductive (?) properties	195
12.4.5	<i>min</i> and <i>max</i>	198
12.4.6	Monotonicity of Addition	199
12.4.7	Additional theorems about <i>op</i> \leq	200
12.4.8	More results about difference	203
12.4.9	Monotonicity of Multiplication	204
12.5	Embedding of the Naturals into any <i>semiring-1: of-nat</i>	206
12.6	The Set of Natural Numbers	208
12.7	Further Arithmetic Facts Concerning the Natural Numbers	208
12.8	size of a datatype value	211
13	Product-Type: Cartesian products	211
13.1	<i>bool</i> is a datatype	212
13.2	Unit	212
13.3	Pairs	213
13.3.1	Product type, basic operations and concrete syntax	213
13.3.2	Basic rules and proof tools	215
13.3.3	<i>split</i> and <i>curry</i>	216
13.4	Further cases/induct rules for tuples	220
13.4.1	Derived operations	221
13.4.2	Code generator setup	225
13.5	Legacy bindings	226
13.6	Further inductive packages	226

14 Datatype: Analogues of the Cartesian Product and Disjoint Sum for Datatypes	226
14.1 Freeness: Distinctness of Constructors	229
14.2 Set Constructions	231
15 Datatypes	236
15.1 Representing sums	236
16 Power: Exponentiation	236
16.1 Powers for Arbitrary Monoids	237
16.2 Exponentiation for the Natural Numbers	241
17 Finite-Set: Finite sets	242
17.1 Definition and basic properties	242
17.1.1 Finiteness and set theoretic constructions	244
17.2 Class <i>finite</i>	247
17.3 A fold functional for finite sets	247
17.3.1 From <i>fold-graph</i> to <i>fold</i>	248
17.3.2 The derived combinator <i>fold-image</i>	250
17.4 Generalized summation over a set	252
17.4.1 Properties in more restricted classes of structures	255
17.5 Generalized product over a set	259
17.5.1 Properties in more restricted classes of structures	262
17.6 Finite cardinality	263
17.6.1 Cardinality of unions	266
17.6.2 Cardinality of image	266
17.6.3 Cardinality of products	267
17.6.4 Cardinality of sums	267
17.6.5 Cardinality of the Powerset	267
17.6.6 Relating injectivity and surjectivity	268
17.7 A fold functional for non-empty sets	268
17.7.1 Determinacy for <i>fold1Set</i>	270
17.7.2 Lemmas about <i>fold1</i>	270
17.7.3 Fold1 in lattices with <i>inf</i> and <i>sup</i>	271
17.7.4 Fold1 in linear orders with <i>min</i> and <i>max</i>	273
18 Relation: Relations	277
18.1 Definitions	277
18.2 The identity relation	279
18.3 Diagonal: identity over a set	279
18.4 Composition of two relations	279
18.5 Reflexivity	280
18.6 Antisymmetry	281
18.7 Symmetry	281

18.8	Transitivity	282
18.9	Irreflexivity	282
18.10	Totality	282
18.11	Converse	283
18.12	Domain	284
18.13	Range	285
18.14	Field	286
18.15	Image of a set under a relation	286
18.16	Single valued relations	288
18.17	Graphs given by <i>Collect</i>	288
18.18	Inverse image	288
18.19	Finiteness	288
18.20	Version of <i>lfp-induct</i> for binary relations	289
19	Predicate: Predicates as relations and enumerations	289
19.1	Predicates as (complete) lattices	289
19.1.1	$op \sqcup$ on <i>bool</i>	289
19.1.2	Equality and Subsets	289
19.1.3	Top and bottom elements	290
19.1.4	The empty set	290
19.1.5	Binary union	290
19.1.6	Binary intersection	291
19.1.7	Unions of families	292
19.1.8	Intersections of families	292
19.2	Predicates as relations	293
19.2.1	Composition	293
19.2.2	Converse	293
19.2.3	Domain	294
19.2.4	Range	295
19.2.5	Inverse image	295
19.2.6	Powerset	295
19.2.7	Properties of relations	295
19.3	Predicates as enumerations	296
19.3.1	The type of predicate enumerations (a monad)	296
19.3.2	Derived operations	298
19.3.3	Implementation	298
20	Transitive-Closure: Reflexive and Transitive closure of a relation	301
20.1	Reflexive closure	302
20.2	Reflexive-transitive closure	302
20.3	Transitive closure	305
20.4	Setup of transitivity reasoner	310

21 Wellfounded: Well-founded Recursion	310
21.1 Basic Definitions	310
21.2 Basic Results	312
21.3 Well-Foundedness Results for Unions	313
21.3.1 acyclic	314
21.4 Well-Founded Recursion	315
21.5 Code generator setup	315
21.6 <i>nat</i> is well-founded	315
21.7 Accessible Part	316
21.8 Tools for building wellfounded relations	318
21.9 Weakly decreasing sequences (w.r.t. some well-founded order) stabilize.	320
21.10size of a datatype value	320
22 FunDef: Function Definitions and Termination Proofs	321
22.1 Definitions with default value.	321
22.2 Measure Functions	322
22.3 Congruence Rules	323
22.4 Simp rules for termination proofs	323
22.5 Decomposition	324
22.6 Reduction Pairs	324
22.7 Concrete orders for SCNP termination proofs	324
22.8 Tool setup	325
23 Record: Extensible records with structural subtyping	326
23.1 Concrete record syntax	326
24 Option: Datatype option	327
24.0.1 Operations	327
24.0.2 Code generator setup	329
25 Extraction: Program extraction for HOL	329
25.1 Setup	329
25.2 Type of extracted program	330
25.3 Realizability	331
25.4 Computational content of basic inference rules	332
26 Divides: The division operators <i>div</i> and <i>mod</i>	337
26.1 Syntactic division operations	337
26.2 Abstract division in commutative semirings.	337
26.3 Division on <i>nat</i>	341
26.3.1 Quotient	344
26.3.2 Remainder	344
26.3.3 Quotient and Remainder	345

26.3.4	Cancellation of Common Factors in Division	346
26.3.5	Further Facts about Quotient and Remainder	346
26.3.6	The Divides Relation	347
26.3.7	An “induction” law for modulus arithmetic.	349
27	Plain: Plain HOL	349
28	Relation-Power: Powers of Relations and Functions	349
29	Equiv-Relations: Equivalence Relations in Higher-Order Set Theory	352
29.1	Equivalence relations	352
29.2	Equivalence classes	352
29.3	Quotients	353
29.4	Defining unary operations upon equivalence classes	354
29.5	Defining binary operations upon equivalence classes	355
29.6	Quotients and finiteness	356
30	Int: The Integers as Equivalence Classes over Pairs of Natural Numbers	356
30.1	The equivalence relation underlying the integers	357
30.2	Construction of the Integers	358
30.3	Arithmetic Operations	358
30.4	The \leq Ordering	359
30.5	Embedding of the Integers into any <i>ring-1: of-int</i>	360
30.6	Magnitude of an Integer, as a Natural Number: <i>nat</i>	362
30.7	Lemmas about the Function <i>of-nat</i> and Orderings	364
30.8	Cases and induction	365
30.9	Binary representation	365
30.9.1	The constructors <i>Bit0</i> , <i>Bit1</i> , <i>Pls</i> and <i>Min</i>	365
30.9.2	Successor and predecessor functions	367
30.9.3	Binary arithmetic	368
30.9.4	Binary comparisons	369
30.10	Converting Numerals to Rings: <i>number-of</i>	371
30.10.1	Equality of Binary Numbers	373
30.10.2	Comparisons, for Ordered Rings	374
30.10.3	The Less-Than Relation	374
30.10.4	Simplification of arithmetic operations on integer constants.	375
30.10.5	Simplification of arithmetic when nested to the right.	376
30.11	The Set of Integers	376
30.12	<i>setsum</i> and <i>setprod</i>	378
30.13	Inequality Reasoning for the Arithmetic Simproc	378
30.14	Special Arithmetic Rules for Abstract 0 and 1	378

30.15	Setting up simplification procedures	380
30.16	Lemmas About Small Numerals	380
30.17	More Inequality Reasoning	381
30.18	The functions <i>nat</i> and <i>int</i>	381
30.19	Induction principles for <i>int</i>	382
30.20	Intermediate value theorems	383
30.21	Products and 1, by T. M. Rasmussen	383
30.22	Integer Powers	384
30.23	Further theorems on numerals	385
30.23.1	Special Simplification for Constants	385
30.23.2	Optional Simplification Rules Involving Constants	387
30.24	Configuration of the code generator	388
30.25	Legacy theorems	391
31	IntDiv: The Division Operators <i>div</i> and <i>mod</i>	392
31.1	Uniqueness and Monotonicity of Quotients and Remainders	394
31.2	Correctness of <i>posDivAlg</i> , the Algorithm for Non-Negative Dividends	394
31.3	Correctness of <i>negDivAlg</i> , the Algorithm for Negative Dividends	395
31.4	Existence Shown by Proving the Division Algorithm to be Correct	395
31.5	General Properties of <i>div</i> and <i>mod</i>	396
31.6	Laws for <i>div</i> and <i>mod</i> with Unary Minus	397
31.7	Division of a Number by Itself	398
31.8	Computation of Division and Remainder	398
31.9	Monotonicity in the First Argument (Dividend)	401
31.10	Monotonicity in the Second Argument (Divisor)	401
31.11	More Algebraic Laws for <i>div</i> and <i>mod</i>	401
31.12	Proving $a \text{ div } (b * c) = a \text{ div } b \text{ div } c$	402
31.13	Cancellation of Common Factors in <i>div</i>	403
31.14	Distribution of Factors over <i>mod</i>	403
31.15	Splitting Rules for <i>div</i> and <i>mod</i>	404
31.16	Speeding up the Division Algorithm with Shifting	404
31.17	Computing <i>mod</i> by Shifting (proofs resemble those for <i>div</i>)	405
31.18	Quotients of Signs	405
31.19	The Divides Relation	406
31.20	Simproc setup	408
31.21	Code generation	409
32	NatBin: Binary arithmetic for the natural numbers	409
32.1	Predicate for negative binary numbers	410
32.2	Function <i>nat</i> : Coercion from Type <i>int</i> to <i>nat</i>	411
32.3	Function <i>int</i> : Coercion from Type <i>nat</i> to <i>int</i>	411
32.3.1	Successor	412

32.3.2	Addition	412
32.3.3	Subtraction	412
32.3.4	Multiplication	413
32.3.5	Quotient	413
32.3.6	Remainder	413
32.3.7	Divisibility	413
32.4	Comparisons	414
32.4.1	Equals (=)	414
32.4.2	Less-than (<)	414
32.4.3	Less-than-or-equal	414
32.5	Powers with Numeric Exponents	414
32.5.1	Nat	416
32.5.2	Arith	416
32.6	Comparisons involving (0::nat)	417
32.7	Comparisons involving <i>Suc</i>	417
32.8	Max and Min Combined with <i>Suc</i>	418
32.9	Literal arithmetic involving powers	419
32.10	Literal arithmetic and <i>of-nat</i>	420
32.11	Lemmas for the Combination and Cancellation Simprocs	421
32.11.1	For <i>combine-numerals</i>	421
32.11.2	For <i>cancel-numerals</i>	421
32.11.3	For <i>cancel-numeral-factors</i>	422
32.11.4	For <i>cancel-factor</i>	422
32.12	Simprocs for the Naturals	422
32.12.1	For simplifying <i>Suc m - K</i> and <i>K - Suc m</i>	422
32.12.2	For <i>nat-case</i> and <i>nat-rec</i>	423
32.12.3	Various Other Lemmas	423
33	Groebner-Basis: Semiring normalization and Groebner Bases	425
33.1	Semiring normalization	425
33.1.1	Declaring the abstract theory	426
33.2	Groebner Bases	428
33.3	Groebner Bases for fields	431
34	SetInterval: Set intervals	431
34.1	Various equivalences	433
34.2	Logical Equivalences for Set Inclusion and Equality	433
34.3	Two-sided intervals	434
34.3.1	Emptiness and singletons	435
34.4	Intervals of natural numbers	435
34.4.1	The Constant <i>lessThan</i>	435
34.4.2	The Constant <i>greaterThan</i>	435
34.4.3	The Constant <i>atLeast</i>	436
34.4.4	The Constant <i>atMost</i>	436

34.4.5	The Constant <i>atLeastLessThan</i>	436
34.4.6	Intervals of nats with <i>Suc</i>	437
34.4.7	Image	437
34.4.8	Finiteness	437
34.4.9	Cardinality	438
34.5	Intervals of integers	439
34.5.1	Finiteness	439
34.5.2	Cardinality	440
34.6	Lemmas useful with the summation operator <i>setsum</i>	440
34.6.1	Disjoint Unions	440
34.6.2	Disjoint Intersections	441
34.6.3	Some Differences	442
34.6.4	Some Subset Conditions	442
34.7	Summation indexed over intervals	442
34.8	Shifting bounds	444
34.9	The formula for geometric sums	445
34.10	The formula for arithmetic sums	445
34.11	Products indexed over intervals	446
35	Presburger: Decision Procedure for Presburger Arithmetic	446
35.1	The $-\infty$ and $+\infty$ Properties	447
35.2	The A and B sets	448
35.3	Cooper's Theorem $-\infty$ and $+\infty$ Version	449
35.3.1	First some trivial facts about periodic sets or predicates	449
35.3.2	The $-\infty$ Version	449
35.3.3	The $+\infty$ Version	450
36	Recdef: TFL: recursive function definitions	452
37	Hilbert-Choice: Hilbert's Epsilon-Operator and the Axiom of Choice	454
37.1	Hilbert's epsilon	454
37.2	Hilbert's Epsilon-operator	454
37.3	Axiom of Choice, Proved Using the Description Operator	455
37.4	Function Inverse	455
37.5	Inverse of a PI-function (restricted domain)	457
37.6	Other Consequences of Hilbert's Epsilon	458
37.7	Least value operator	458
37.8	Greatest value operator	459
37.9	The Meson proof procedure	460
37.9.1	Negation Normal Form	460
37.9.2	Pulling out the existential quantifiers	461
37.9.3	Generating clauses for the Meson Proof Procedure	461
37.10	Lemmas for Meson, the Model Elimination Procedure	461

37.10.1 Lemmas for Forward Proof	462
37.11 Meson package	463
37.12 Specification package – Hilbertized version	463
38 ATP-Linkup: The Isabelle-ATP Linkup	463
38.1 Setup of external ATPs	464
38.2 The Metis prover	464
39 List: The datatype of finite lists	465
39.1 Basic list processing functions	465
39.1.1 List comprehension	469
39.1.2 $[]$ and $op \#$	471
39.1.3 <i>length</i>	472
39.1.4 $@ -$ append	473
39.1.5 <i>map</i>	475
39.1.6 <i>rev</i>	476
39.1.7 <i>set</i>	477
39.1.8 <i>filter</i>	479
39.1.9 List partitioning	481
39.1.10 <i>concat</i>	481
39.1.11 <i>nth</i>	482
39.1.12 <i>list-update</i>	483
39.1.13 <i>last</i> and <i>butlast</i>	484
39.1.14 <i>take</i> and <i>drop</i>	486
39.1.15 <i>takeWhile</i> and <i>dropWhile</i>	489
39.1.16 <i>zip</i>	490
39.1.17 <i>list-all2</i>	492
39.1.18 <i>foldl</i> and <i>foldr</i>	495
39.1.19 List summation: <i>listsum</i> and \sum	496
39.1.20 <i>upt</i>	497
39.1.21 <i>distinct</i> and <i>remdups</i>	499
39.1.22 <i>remove1</i>	501
39.1.23 <i>removeAll</i>	501
39.1.24 <i>replicate</i>	502
39.1.25 <i>rotate1</i> and <i>rotate</i>	503
39.1.26 <i>sublist</i> — a generalization of <i>nth</i> to sets	505
39.1.27 <i>splice</i>	506
39.1.28 Infiniteness	506
39.2 Sorting	507
39.2.1 <i>sorted-list-of-set</i>	508
39.2.2 <i>upto</i> : the generic interval-list	509
39.2.3 <i>lists</i> : the list-forming operator over sets	510
39.2.4 Inductive definition for membership	511
39.2.5 Lists as Cartesian products	511

39.3	Relations on Lists	511
39.3.1	Length Lexicographic Ordering	511
39.3.2	Lexicographic Ordering	513
39.4	Lexicographic combination of measure functions	514
39.4.1	Lifting a Relation on List Elements to the Lists	514
39.5	Miscellany	515
39.5.1	Characters and strings	515
39.6	Size function	516
39.7	Code generator	517
39.7.1	Setup	517
39.7.2	Generation of efficient code	518
40	Code-Message: Monolithic strings (message strings) for code generation	522
40.1	Datatype of messages	522
40.2	ML interface	523
40.3	Code serialization	523
41	Typerep: Reflecting Pure types into HOL	523
42	Code-Eval: Term evaluation using the generic code generator	524
42.1	Term representation	524
42.1.1	Terms and class <i>term-of</i>	524
42.1.2	<i>term-of</i> instances	524
42.1.3	Code generator setup	525
42.2	Evaluation setup	525
42.2.1	Syntax	525
43	Map: Maps	525
43.1	<i>empty</i>	527
43.2	<i>map-upd</i>	527
43.3	<i>map-of</i>	528
43.4	<i>Option.map</i> related	529
43.5	<i>map-comp</i> related	529
43.6	<i>++</i>	530
43.7	<i>restrict-map</i>	531
43.8	<i>map-upds</i>	532
43.9	<i>dom</i>	533
43.10	<i>ran</i>	534
43.11	<i>map-le</i>	534
44	Refute: Refute	535

45 SAT: Reconstructing external resolution proofs for propositional logic	537
46 Main: Main HOL	538
47 Lubs: Definitions of Upper Bounds and Least Upper Bounds	538
47.1 Rules for the Relations $* \leq$ and $\leq *$	539
47.2 Rules about the Operators <i>leastP</i> , <i>ub</i> and <i>lub</i>	539
48 GCD: The Greatest Common Divisor	540
48.1 Specification of GCD on nats	540
48.2 GCD on nat by Euclid's algorithm	541
48.3 Derived laws for GCD	541
48.4 LCM defined by GCD	544
48.5 GCD and LCM on integers	545
49 Archimedean-Field: Archimedean Fields, Floor and Ceiling Functions	548
49.1 Class of Archimedean fields	548
49.2 Existence and uniqueness of floor function	549
49.3 Floor function	549
49.4 Ceiling function	552
49.5 Negation	554
50 Rational: Rational numbers	554
50.1 Rational numbers as quotient	554
50.1.1 Construction of the type of rational numbers	554
50.1.2 Representation and basic operations	555
50.1.3 The field of rational numbers	558
50.1.4 Various	558
50.1.5 The ordered field of rational numbers	559
50.1.6 Rationals are an Archimedean field	560
50.2 Arithmetic setup	561
50.3 Embedding from Rationals to other Fields	561
50.4 The Set of Rational Numbers	563
50.5 Implementation of rational numbers as pairs of integers	564
51 PReal: Positive real numbers	567
51.1 <i>preal-of-prat</i> : the Injection from <i>prat</i> to <i>preal</i>	570
51.2 Properties of Ordering	571
51.3 Properties of Addition	571
51.4 Properties of Multiplication	572
51.5 Distribution of Multiplication across Addition	574
51.6 Existence of Inverse, a Positive Real	574
51.7 Gleason's Lemma 9-3.4, page 122	575

51.8	Gleason's Lemma 9-3.6	575
51.9	Existence of Inverse: Part 2	576
51.10	Subtraction for Positive Reals	577
51.11	proving that $S \leq R + D$ — trickier	578
51.12	Completeness of type <i>preal</i>	579
51.13	The Embedding from <i>rat</i> into <i>preal</i>	580
52	RealDef: Defining the Reals from the Positive Reals	582
52.1	Equivalence relation over positive reals	583
52.2	Addition and Subtraction	584
52.3	Multiplication	584
52.4	Inverse and Division	585
52.5	The Real Numbers form a Field	585
52.6	The \leq Ordering	585
52.7	The Reals Form an Ordered Field	587
52.8	Theorems About the Ordering	588
52.9	More Lemmas	588
52.10	Embedding numbers into the Reals	589
52.11	Embedding the Naturals into the Reals	591
52.12	Rationals	593
52.13	Numerals and Arithmetic	594
52.14	Simprules combining $x+y$ and 0: ARE THEY NEEDED?	594
52.14.1	Density of the Reals	595
52.15	Absolute Value Function for the Reals	595
52.16	Implementation of rational real numbers	596
53	RComplete: Completeness of the Reals; Floor and Ceiling Functions	598
53.1	Completeness of Positive Reals	598
53.2	The Archimedean Property of the Reals	599
53.3	Density of the Rational Reals in the Reals	600
53.4	Floor and Ceiling Functions from the Reals to the Integers	600
53.5	Versions for the natural numbers	605
54	RealPow: Natural powers theory	608
54.1	Literal Arithmetic Involving Powers, Type <i>real</i>	609
54.2	Properties of Squares	610
54.3	Squares of Reals	611
54.4	Various Other Theorems	612
54.5	Float syntax	612

55 RealVector: Vector Spaces and Algebras over the Reals	613
55.1 Locale for additive functions	613
55.2 Vector spaces	613
55.3 Real vector spaces	614
55.4 Embedding of the Reals into any <i>real-algebra-1: of-real</i> . . .	616
55.5 The Set of Real Numbers	617
55.6 Real normed vector spaces	619
55.7 Sign function	622
55.8 Bounded Linear and Bilinear Operators	623
56 Fact: Factorial Function	625
57 SEQ: Sequences and Convergence	627
57.1 Bounded Sequences	628
57.2 Sequences That Converge to Zero	628
57.3 Limits of Sequences	630
57.4 Convergence	634
57.5 Bounded Monotonic Sequences	635
57.5.1 Upper Bounds and Lubs of Bounded Sequences . . .	636
57.5.2 A Bounded and Monotonic Sequence Converges . . .	636
57.5.3 Increasing and Decreasing Series	637
57.5.4 A Few More Equivalence Theorems for Boundedness .	637
57.6 Cauchy Sequences	638
57.6.1 Cauchy Sequences are Bounded	638
57.6.2 Cauchy Sequences are Convergent	638
57.7 Power Sequences	640
58 Series: Finite Summation and Infinite Series	641
58.1 Infinite Sums, by the Properties of Limits	642
58.2 The Ratio Test	647
58.3 Cauchy Product Formula	648
59 Lim: Limits and Continuity	648
59.1 Limits of Functions	649
59.1.1 Purely standard proofs	649
59.1.2 Derived theorems about <i>LIM</i>	653
59.2 Continuity	654
59.2.1 Purely standard proofs	654
59.3 Uniform Continuity	656
59.4 Relation of LIM and LIMSEQ	656

60 Deriv: Differentiation	656
60.1 Derivatives	657
60.2 Differentiability predicate	660
60.3 Nested Intervals and Bisection	662
60.4 Intermediate Value Theorem	664
60.5 Boundedness of continuous functions	664
60.6 Local extrema	665
60.7 Rolle's Theorem	666
60.8 Mean Value Theorem	667
60.9 Continuous injective functions	668
60.10 Generalized Mean Value Theorem	669
60.11 Theorems about Limits	669
61 Parity: Even and Odd for int and nat	670
61.1 Even and odd are mutually exclusive	671
61.2 Behavior under integer arithmetic operations	671
61.3 Equivalent definitions	672
61.4 even and odd for nats	672
61.5 Equivalent definitions	673
61.6 Parity and powers	673
61.7 General Lemmas About Division	675
61.8 More Even/Odd Results	676
61.9 An Equivalence for $0 \leq a^n$	677
61.10 Miscellaneous	677
62 NthRoot: Nth Roots of Real Numbers	678
62.1 Existence of Nth Root	678
62.2 Nth Root	678
62.3 Square Root	682
62.4 Square Root of Sum of Squares	684
63 Transcendental: Power Series, Transcendental Functions etc.	686
63.1 Properties of Power Series	686
63.2 Alternating series test / Leibniz formula	687
63.3 Term-by-Term Differentiability of Power Series	688
63.4 Some properties of factorials	690
63.5 Derivability of power series	690
63.6 Exponential Function	691
63.6.1 Properties of the Exponential Function	691
63.6.2 Properties of the Exponential Function on Reals	692
63.7 Natural Logarithm	694
63.8 Sine and Cosine	696
63.9 Properties of Sine and Cosine	698
63.10 The Constant Pi	701

63.11Tangent	706
63.12Inverse Trigonometric Functions	708
63.13More Theorems about Sin and Cos	711
63.14Machins formula	712
63.15Introducing the arcus tangens power series	712
63.16Existence of Polar Coordinates	714
64 Complex: Complex Numbers: Rectangular and Polar Representations	714
64.1 Addition and Subtraction	715
64.2 Multiplication and Division	716
64.3 Exponentiation	717
64.4 Numerals and Arithmetic	717
64.5 Scalar Multiplication	718
64.6 Properties of Embedding from Reals	719
64.7 Vector Norm	719
64.8 Completeness of the Complexes	720
64.9 The Complex Number i	721
64.10Complex Conjugation	721
64.11The Functions sgn and arg	723
64.12Finally! Polar Form for Complex Numbers	724
65 Log: Logarithms: Standard Version	727
66 Ln: Properties of ln	730
67 MacLaurin: MacLaurin Series	732
67.1 Maclaurin's Theorem with Lagrange Form of Remainder . . .	732
67.2 More Convenient "Bidirectional" Version.	734
67.3 Version for Exponential Function	735
67.4 Version for Sine Function	735
67.5 Maclaurin Expansion for Cosine Function	737
68 Taylor: Taylor series	738
69 Integration: Theory of Integration	739
69.1 Lemmas for Additivity Theorem of Gauge Integral	744
70 Complex-Main: Comprehensive Complex Theory	747

1 HOL: The basis of Higher-Order Logic

```

theory HOL
imports Pure
uses
  (Tools/hologic.ML)
  ~~ /src/Tools/IsaPlanner/zipper.ML
  ~~ /src/Tools/IsaPlanner/isand.ML
  ~~ /src/Tools/IsaPlanner/rw-tools.ML
  ~~ /src/Tools/IsaPlanner/rw-inst.ML
  ~~ /src/Tools/intuitionistic.ML
  ~~ /src/Tools/project-rule.ML
  ~~ /src/Provers/hypsubst.ML
  ~~ /src/Provers/splitter.ML
  ~~ /src/Provers/classical.ML
  ~~ /src/Provers/blast.ML
  ~~ /src/Provers/clasimp.ML
  ~~ /src/Tools/coherent.ML
  ~~ /src/Tools/eqsubst.ML
  ~~ /src/Provers/quantifier1.ML
  (Tools/simpdata.ML)
  ~~ /src/Tools/random-word.ML
  ~~ /src/Tools/atomize-elim.ML
  ~~ /src/Tools/induct.ML
  (~~ /src/Tools/induct-tacs.ML)
  ~~ /src/Tools/value.ML
  ~~ /src/Tools/code/code-name.ML
  ~~ /src/Tools/code/code-funcgr.ML
  ~~ /src/Tools/code/code-wellsorted.ML
  ~~ /src/Tools/code/code-thingol.ML
  ~~ /src/Tools/code/code-printer.ML
  ~~ /src/Tools/code/code-target.ML
  ~~ /src/Tools/code/code-ml.ML
  ~~ /src/Tools/code/code-haskell.ML
  ~~ /src/Tools/nbe.ML
  (Tools/recfun-codegen.ML)
begin

  <ML>

```

1.1 Primitive logic

1.1.1 Core syntax

```

classes type
defaultsort type
  <ML>

```

```

arities
  fun :: (type, type) type

```

itself :: (type) type

global

typeddecl bool

judgment

Trueprop :: bool => prop ((-) 5)

consts

Not :: bool => bool (\sim - [40] 40)

True :: bool

False :: bool

The :: ('a => bool) => 'a

All :: ('a => bool) => bool (**binder** ALL 10)

Ex :: ('a => bool) => bool (**binder** EX 10)

Ex1 :: ('a => bool) => bool (**binder** EX! 10)

Let :: ['a, 'a => 'b] => 'b

op = :: ['a, 'a] => bool (**infixl** = 50)

op & :: [bool, bool] => bool (**infixr** & 35)

op | :: [bool, bool] => bool (**infixr** | 30)

op --> :: [bool, bool] => bool (**infixr** --> 25)

local

consts

If :: [bool, 'a, 'a] => 'a ((if (-)/ then (-)/ else (-)) 10)

1.1.2 Additional concrete syntax

notation (output)

op = (**infix** = 50)

abbreviation

not-equal :: ['a, 'a] => bool (**infixl** ~= 50) **where**

$x \text{ ~= } y == \sim (x = y)$

notation (output)

not-equal (**infix** ~= 50)

notation (xsymbols)

Not (\neg - [40] 40) **and**

op & (**infixr** \wedge 35) **and**

op | (**infixr** \vee 30) **and**

op --> (**infixr** \longrightarrow 25) **and**

not-equal (**infix** \neq 50)

notation (*HTML output*)

Not (\neg - [40] 40) and
op & (**infixr** \wedge 35) and
op | (**infixr** \vee 30) and
not-equal (**infix** \neq 50)

abbreviation (*iff*)

iff :: [bool, bool] => bool (**infixr** \longleftrightarrow 25) where
 $A \longleftrightarrow B == A = B$

notation (*xsymbols*)

iff (**infixr** \longleftrightarrow 25)

nonterminals

letbinds *letbind*
case-syn *cases-syn*

syntax

-The :: [pttrn, bool] => 'a ((3THE -./ -) [0, 10] 10)

-bind :: [pttrn, 'a] => letbind ((2- =/ -) 10)
 :: letbind => letbinds (-)

-binds :: [letbind, letbinds] => letbinds (-;/ -)

-Let :: [letbinds, 'a] => 'a ((let (-)/ in (-)) 10)

-case-syntax:: ['a, cases-syn] => 'b ((case - of / -) 10)

-case1 :: ['a, 'b] => case-syn ((2- =>/ -) 10)
 :: case-syn => cases-syn (-)

-case2 :: [case-syn, cases-syn] => cases-syn (-/ | -)

translations

THE $x. P == The (\%x. P)$
-Let (*-binds* b bs) $e == -Let\ b\ (-Let\ bs\ e)$
 $let\ x = a\ in\ e == Let\ a\ (\%x. e)$

$\langle ML \rangle$

syntax (*xsymbols*)

-case1 :: ['a, 'b] => case-syn ((2- =>/ -) 10)

notation (*xsymbols*)

All (**binder** \forall 10) and
Ex (**binder** \exists 10) and
Ex1 (**binder** $\exists!$ 10)

notation (*HTML output*)

All (**binder** \forall 10) and
Ex (**binder** \exists 10) and

Ex1 (**binder** $\exists!$ 10)

notation (*HOL*)

All (**binder** ! 10) and

Ex (**binder** ? 10) and

Ex1 (**binder** ?! 10)

1.1.3 Axioms and basic definitions

axioms

refl: $t = (t::'a)$

subst: $s = t \implies P\ s \implies P\ t$

ext: $(!!x::'a. (f\ x :: 'b) = g\ x) \implies (\%x. f\ x) = (\%x. g\ x)$

— Extensionality is built into the meta-logic, and this rule expresses a related property. It is an eta-expanded version of the traditional rule, and similar to the ABS rule of HOL

the-eq-trivial: $(THE\ x. x = a) = (a::'a)$

impI: $(P \implies Q) \implies P \multimap Q$

mp: $[P \multimap Q; P] \implies Q$

defs

True-def: $True == ((\%x::bool. x) = (\%x. x))$

All-def: $All(P) == (P = (\%x. True))$

Ex-def: $Ex(P) == !Q. (!x. P\ x \multimap Q) \multimap Q$

False-def: $False == (!P. P)$

not-def: $\sim P == P \multimap False$

and-def: $P \ \& \ Q == !R. (P \multimap Q \multimap R) \multimap R$

or-def: $P \mid Q == !R. (P \multimap R) \multimap (Q \multimap R) \multimap R$

Ex1-def: $Ex1(P) == ?\ x. P(x) \ \& \ (!\ y. P(y) \multimap y=x)$

axioms

iff: $(P \multimap Q) \multimap (Q \multimap P) \multimap (P=Q)$

True-or-False: $(P=True) \mid (P=False)$

defs

Let-def: $Let\ s\ f == f(s)$

if-def: $If\ P\ x\ y == THE\ z::'a. (P=True \multimap z=x) \ \& \ (P=False \multimap z=y)$

finalconsts

op =

op \multimap

The

axiomatization

undefined $:: 'a$

abbreviation (*input*)
arbitrary \equiv *undefined*

1.1.4 Generic classes and algebraic operations

```

class default =
  fixes default :: 'a

class zero =
  fixes zero :: 'a (0)

class one =
  fixes one :: 'a (1)

hide (open) const zero one

class plus =
  fixes plus :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl + 65)

class minus =
  fixes minus :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl - 65)

class uminus =
  fixes uminus :: 'a  $\Rightarrow$  'a (- - [81] 80)

class times =
  fixes times :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl * 70)

class inverse =
  fixes inverse :: 'a  $\Rightarrow$  'a
  and divide :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl ' / 70)

class abs =
  fixes abs :: 'a  $\Rightarrow$  'a
begin

notation (xsymbols)
  abs (|·|)

notation (HTML output)
  abs (|·|)

end

class sgn =
  fixes sgn :: 'a  $\Rightarrow$  'a

class ord =

```

```

fixes less-eq :: 'a ⇒ 'a ⇒ bool
and less :: 'a ⇒ 'a ⇒ bool
begin

notation
  less-eq (op <=) and
  less-eq ((-/ <= -) [51, 51] 50) and
  less (op <) and
  less ((-/ < -) [51, 51] 50)

notation (xsymbols)
  less-eq (op ≤) and
  less-eq ((-/ ≤ -) [51, 51] 50)

notation (HTML output)
  less-eq (op ≤) and
  less-eq ((-/ ≤ -) [51, 51] 50)

abbreviation (input)
  greater-eq (infix >= 50) where
    x >= y ≡ y <= x

notation (input)
  greater-eq (infix ≥ 50)

abbreviation (input)
  greater (infix > 50) where
    x > y ≡ y < x

end

syntax
  -index1 :: index (1)
translations
  (index)1 ==> (index)◇

```

⟨ML⟩

1.2 Fundamental rules

1.2.1 Equality

lemma *sym*: $s = t \implies t = s$
 ⟨*proof*⟩

lemma *ssubst*: $t = s \implies P\ s \implies P\ t$
 ⟨*proof*⟩

lemma *trans*: $[| r=s; s=t |] \implies r=t$
 ⟨*proof*⟩

lemma *meta-eq-to-obj-eq*:
assumes *meq*: $A == B$
shows $A = B$
 $\langle proof \rangle$

Useful with *erule* for proving equalities from known equalities.

lemma *box-equals*: $[| a=b; a=c; b=d |] ==> c=d$
 $\langle proof \rangle$

For calculational reasoning:

lemma *forw-subst*: $a = b ==> P\ b ==> P\ a$
 $\langle proof \rangle$

lemma *back-subst*: $P\ a ==> a = b ==> P\ b$
 $\langle proof \rangle$

1.2.2 Congruence rules for application

lemma *fun-cong*: $(f::'a=>'b) = g ==> f(x)=g(x)$
 $\langle proof \rangle$

lemma *arg-cong*: $x=y ==> f(x)=f(y)$
 $\langle proof \rangle$

lemma *arg-cong2*: $[| a = b; c = d |] ==> f\ a\ c = f\ b\ d$
 $\langle proof \rangle$

lemma *cong*: $[| f = g; (x::'a) = y |] ==> f(x) = g(y)$
 $\langle proof \rangle$

1.2.3 Equality of booleans – iff

lemma *iffI*: **assumes** $P ==> Q$ **and** $Q ==> P$ **shows** $P=Q$
 $\langle proof \rangle$

lemma *iffD2*: $[| P=Q; Q |] ==> P$
 $\langle proof \rangle$

lemma *rev-iffD2*: $[| Q; P=Q |] ==> P$
 $\langle proof \rangle$

lemma *iffD1*: $Q = P ==> Q ==> P$
 $\langle proof \rangle$

lemma *rev-iffD1*: $Q ==> Q = P ==> P$
 $\langle proof \rangle$

lemma *iffE*:
 assumes *major*: $P=Q$
 and *minor*: $[| P \dashv\vdash Q; Q \dashv\vdash P |] \implies R$
 shows R
 $\langle proof \rangle$

1.2.4 True

lemma *TrueI*: $True$
 $\langle proof \rangle$

lemma *eqTrueI*: $P \implies P = True$
 $\langle proof \rangle$

lemma *eqTrueE*: $P = True \implies P$
 $\langle proof \rangle$

1.2.5 Universal quantifier

lemma *allI*: assumes $!!x::'a. P(x)$ shows $ALL\ x. P(x)$
 $\langle proof \rangle$

lemma *spec*: $ALL\ x::'a. P(x) \implies P(x)$
 $\langle proof \rangle$

lemma *allE*:
 assumes *major*: $ALL\ x. P(x)$
 and *minor*: $P(x) \implies R$
 shows R
 $\langle proof \rangle$

lemma *all-dupE*:
 assumes *major*: $ALL\ x. P(x)$
 and *minor*: $[| P(x); ALL\ x. P(x) |] \implies R$
 shows R
 $\langle proof \rangle$

1.2.6 False

Depends upon *spec*; it is impossible to do propositional logic before quantifiers!

lemma *FalseE*: $False \implies P$
 $\langle proof \rangle$

lemma *False-neq-True*: $False = True \implies P$
 $\langle proof \rangle$

1.2.7 Negation

lemma *notI*:
 assumes $P \implies \text{False}$
 shows $\sim P$
 $\langle \text{proof} \rangle$

lemma *False-not-True*: $\text{False} \sim = \text{True}$
 $\langle \text{proof} \rangle$

lemma *True-not-False*: $\text{True} \sim = \text{False}$
 $\langle \text{proof} \rangle$

lemma *notE*: $[\sim P; P] \implies R$
 $\langle \text{proof} \rangle$

lemma *notI2*: $(P \implies \neg Pa) \implies (P \implies Pa) \implies \neg P$
 $\langle \text{proof} \rangle$

1.2.8 Implication

lemma *impE*:
 assumes $P \multimap Q$ P $Q \implies R$
 shows R
 $\langle \text{proof} \rangle$

lemma *rev-mp*: $[P; P \multimap Q] \implies Q$
 $\langle \text{proof} \rangle$

lemma *contrapos-nn*:
 assumes *major*: $\sim Q$
 and *minor*: $P \implies Q$
 shows $\sim P$
 $\langle \text{proof} \rangle$

lemma *contrapos-pn*:
 assumes *major*: Q
 and *minor*: $P \implies \sim Q$
 shows $\sim P$
 $\langle \text{proof} \rangle$

lemma *not-sym*: $t \sim = s \implies s \sim = t$
 $\langle \text{proof} \rangle$

lemma *eq-neq-eq-imp-neq*: $[x = a; a \sim = b; b = y] \implies x \sim = y$
 $\langle \text{proof} \rangle$

lemma *rev-contrapos*:
 assumes $pq: P \implies Q$
 and $nq: \sim Q$
 shows $\sim P$
 $\langle proof \rangle$

1.2.9 Existential quantifier

lemma *exI*: $P\ x \implies EX\ x::'a.\ P\ x$
 $\langle proof \rangle$

lemma *exE*:
 assumes *major*: $EX\ x::'a.\ P(x)$
 and *minor*: $!!x.\ P(x) \implies Q$
 shows Q
 $\langle proof \rangle$

1.2.10 Conjunction

lemma *conjI*: $[| P; Q |] \implies P \& Q$
 $\langle proof \rangle$

lemma *conjunct1*: $[| P \& Q |] \implies P$
 $\langle proof \rangle$

lemma *conjunct2*: $[| P \& Q |] \implies Q$
 $\langle proof \rangle$

lemma *conjE*:
 assumes *major*: $P \& Q$
 and *minor*: $[| P; Q |] \implies R$
 shows R
 $\langle proof \rangle$

lemma *context-conjI*:
 assumes $P \implies Q$ shows $P \& Q$
 $\langle proof \rangle$

1.2.11 Disjunction

lemma *disjI1*: $P \implies P | Q$
 $\langle proof \rangle$

lemma *disjI2*: $Q \implies P | Q$
 $\langle proof \rangle$

lemma *disjE*:
 assumes *major*: $P | Q$
 and *minorP*: $P \implies R$
 and *minorQ*: $Q \implies R$

shows R
 $\langle proof \rangle$

1.2.12 Classical logic

lemma *classical*:
 assumes *prem*: $\sim P \implies P$
 shows P
 $\langle proof \rangle$

lemmas *ccontr* = *FalseE* [*THEN classical, standard*]

lemma *rev-notE*:
 assumes *premp*: P
 and *premnt*: $\sim R \implies \sim P$
 shows R
 $\langle proof \rangle$

lemma *notnotD*: $\sim\sim P \implies P$
 $\langle proof \rangle$

lemma *contrapos-pp*:
 assumes *p1*: Q
 and *p2*: $\sim P \implies \sim Q$
 shows P
 $\langle proof \rangle$

1.2.13 Unique existence

lemma *ex1I*:
 assumes $P\ a\ \&\&x. P(x) \implies x=a$
 shows $EX!\ x. P(x)$
 $\langle proof \rangle$

Sometimes easier to use: the premises have no shared variables. Safe!

lemma *ex-ex1I*:
 assumes *ex-prem*: $EX\ x. P(x)$
 and *eq*: $\&\&x\ y. [P(x); P(y)] \implies x=y$
 shows $EX!\ x. P(x)$
 $\langle proof \rangle$

lemma *ex1E*:
 assumes *major*: $EX!\ x. P(x)$
 and *minor*: $\&\&x. [P(x); ALL\ y. P(y) \longrightarrow y=x] \implies R$
 shows R
 $\langle proof \rangle$

lemma *ex1-implies-ex*: $EX!\ x. P\ x \implies EX\ x. P\ x$

$\langle proof \rangle$

1.2.14 THE: definite description operator

lemma *the-equality*:

assumes *prema*: $P\ a$

and *premx*: $!!x. P\ x \implies x=a$

shows $(THE\ x. P\ x) = a$

$\langle proof \rangle$

lemma *theI*:

assumes $P\ a$ and $!!x. P\ x \implies x=a$

shows $P\ (THE\ x. P\ x)$

$\langle proof \rangle$

lemma *theI'*: $EX!\ x. P\ x \implies P\ (THE\ x. P\ x)$

$\langle proof \rangle$

lemma *theI2*:

assumes $P\ a\ !!x. P\ x \implies x=a\ !!x. P\ x \implies Q\ x$

shows $Q\ (THE\ x. P\ x)$

$\langle proof \rangle$

lemma *theI12*: assumes $EX!\ x. P\ x \wedge x. P\ x \implies Q\ x$ shows $Q\ (THE\ x. P\ x)$

$\langle proof \rangle$

lemma *the1-equality* [*elim?*]: $[[EX!\ x. P\ x; P\ a] \implies (THE\ x. P\ x) = a$

$\langle proof \rangle$

lemma *the-sym-eq-trivial*: $(THE\ y. x=y) = x$

$\langle proof \rangle$

1.2.15 Classical intro rules for disjunction and existential quantifiers

lemma *disjCI*:

assumes $\sim Q \implies P$ shows $P \mid Q$

$\langle proof \rangle$

lemma *excluded-middle*: $\sim P \mid P$

$\langle proof \rangle$

case distinction as a natural deduction rule. Note that $\neg P$ is the second case, not the first

lemma *case-split* [*case-names* *True False*]:

assumes *prem1*: $P \implies Q$

and *prem2*: $\sim P \implies Q$

shows Q

$\langle proof \rangle$

lemma *impCE*:
 assumes *major*: $P \dashv\dashv Q$
 and *minor*: $\sim P \implies R \quad Q \implies R$
 shows R
 $\langle proof \rangle$

lemma *impCE'*:
 assumes *major*: $P \dashv\dashv Q$
 and *minor*: $Q \implies R \quad \sim P \implies R$
 shows R
 $\langle proof \rangle$

lemma *iffCE*:
 assumes *major*: $P = Q$
 and *minor*: $[[P; Q]] \implies R \quad [[\sim P; \sim Q]] \implies R$
 shows R
 $\langle proof \rangle$

lemma *exCI*:
 assumes $ALL\ x. \sim P(x) \implies P(a)$
 shows $EX\ x. P(x)$
 $\langle proof \rangle$

1.2.16 Intuitionistic Reasoning

lemma *impE'*:
 assumes 1: $P \dashv\dashv Q$
 and 2: $Q \implies R$
 and 3: $P \dashv\dashv Q \implies P$
 shows R
 $\langle proof \rangle$

lemma *allE'*:
 assumes 1: $ALL\ x. P\ x$
 and 2: $P\ x \implies ALL\ x. P\ x \implies Q$
 shows Q
 $\langle proof \rangle$

lemma *notE'*:
 assumes 1: $\sim P$
 and 2: $\sim P \implies P$
 shows R
 $\langle proof \rangle$

lemma *TrueE*: $True ==> P ==> P$ *<proof>*
lemma *notFalseE*: $\sim False ==> P ==> P$ *<proof>*

lemmas [*Pure.elim!*] = *disjE iffE FalseE conjE exE TrueE notFalseE*
and [*Pure.intro!*] = *iffI conjI impI TrueI notI allI refl*
and [*Pure.elim 2*] = *allE notE' impE'*
and [*Pure.intro*] = *exI disjI2 disjI1*

lemmas [*trans*] = *trans*
and [*sym*] = *sym not-sym*
and [*Pure.elim?*] = *iffD1 iffD2 impE*

<ML>

1.2.17 Atomizing meta-level connectives

axiomatization where

eq-reflection: $x = y \implies x \equiv y$

lemma *atomize-all* [*atomize*]: $(!!x. P x) == Trueprop (ALL x. P x)$
<proof>

lemma *atomize-imp* [*atomize*]: $(A ==> B) == Trueprop (A --> B)$
<proof>

lemma *atomize-not*: $(A ==> False) == Trueprop (\sim A)$
<proof>

lemma *atomize-eq* [*atomize*]: $(x == y) == Trueprop (x = y)$
<proof>

lemma *atomize-conj* [*atomize*]: $(A \&\& B) == Trueprop (A \& B)$
<proof>

lemmas [*symmetric, rulify*] = *atomize-all atomize-imp*
and [*symmetric, defn*] = *atomize-all atomize-imp atomize-eq*

1.2.18 Atomizing elimination rules

<ML>

lemma *atomize-exL*[*atomize-elim*]: $(!!x. P x ==> Q) == ((EX x. P x) ==> Q)$
<proof>

lemma *atomize-conjL*[*atomize-elim*]: $(A ==> B ==> C) == (A \& B ==> C)$
<proof>

lemma *atomize-disjL*[*atomize-elim*]: $((A ==> C) ==> (B ==> C) ==> C)$
 $== ((A \mid B ==> C) ==> C)$
<proof>

lemma *atomize-elimL*[*atomize-elim*]: ($\text{!!}B. (A \Rightarrow B) \Rightarrow B$) == *Trueprop* *A*
 $\langle \text{proof} \rangle$

1.3 Package setup

1.3.1 Classical Reasoner setup

lemma *imp-elim*: $P \rightarrow Q \Rightarrow (\sim R \Rightarrow P) \Rightarrow (Q \Rightarrow R) \Rightarrow R$
 $\langle \text{proof} \rangle$

lemma *swap*: $\sim P \Rightarrow (\sim R \Rightarrow P) \Rightarrow R$
 $\langle \text{proof} \rangle$

lemma *thin-refl*:
 $\bigwedge X. \llbracket x=x; \text{PROP } W \rrbracket \Rightarrow \text{PROP } W \langle \text{proof} \rangle$

$\langle ML \rangle$

ResBlacklist holds theorems blacklisted to *sledgehammer*. These theorems typically produce clauses that are prolific (match too many equality or membership literals) and relate to seldom-used facts. Some duplicate other rules.

$\langle ML \rangle$

declare *iffI* [*intro!*]
and *notI* [*intro!*]
and *impI* [*intro!*]
and *disjCI* [*intro!*]
and *conjI* [*intro!*]
and *TrueI* [*intro!*]
and *refl* [*intro!*]

declare *iffCE* [*elim!*]
and *FalseE* [*elim!*]
and *impCE* [*elim!*]
and *disjE* [*elim!*]
and *conjE* [*elim!*]
and *conjE* [*elim!*]

declare *ex-ex1I* [*intro!*]
and *allI* [*intro!*]
and *the-equality* [*intro*]
and *exI* [*intro*]

declare *exE* [*elim!*]
allE [*elim*]

$\langle ML \rangle$

lemma *contrapos-np*: $\sim Q \implies (\sim P \implies Q) \implies P$
 $\langle \text{proof} \rangle$

declare *ex-ex1I* [*rule del, intro!* 2]
and *ex1I* [*intro*]

lemmas [*intro?*] = *ext*
and [*elim?*] = *ex1-implies-ex*

lemma *alt-ex1E* [*elim!*]:
assumes *major*: $\exists!x. P\ x$
and *prem*: $\bigwedge x. \llbracket P\ x; \forall y\ y'. P\ y \wedge P\ y' \longrightarrow y = y' \rrbracket \implies R$
shows *R*
 $\langle \text{proof} \rangle$
 $\langle ML \rangle$

1.3.2 Simplifier

lemma *eta-contract-eq*: $(\%s. f\ s) = f\ \langle \text{proof} \rangle$

lemma *simp-thms*:
shows *not-not*: $(\sim \sim P) = P$
and *Not-eq-iff*: $((\sim P) = (\sim Q)) = (P = Q)$
and
 $(P \sim = Q) = (P = (\sim Q))$
 $(P \mid \sim P) = \text{True} \quad (\sim P \mid P) = \text{True}$
 $(x = x) = \text{True}$
and *not-True-eq-False*: $(\neg \text{True}) = \text{False}$
and *not-False-eq-True*: $(\neg \text{False}) = \text{True}$
and
 $(\sim P) \sim = P \quad P \sim = (\sim P)$
 $(\text{True} = P) = P$
and *eq-True*: $(P = \text{True}) = P$
and $(\text{False} = P) = (\sim P)$
and *eq-False*: $(P = \text{False}) = (\neg P)$
and
 $(\text{True} \dashrightarrow P) = P \quad (\text{False} \dashrightarrow P) = \text{True}$
 $(P \dashrightarrow \text{True}) = \text{True} \quad (P \dashrightarrow P) = \text{True}$
 $(P \dashrightarrow \text{False}) = (\sim P) \quad (P \dashrightarrow \sim P) = (\sim P)$
 $(P \ \& \ \text{True}) = P \quad (\text{True} \ \& \ P) = P$
 $(P \ \& \ \text{False}) = \text{False} \quad (\text{False} \ \& \ P) = \text{False}$
 $(P \ \& \ P) = P \quad (P \ \& \ (P \ \& \ Q)) = (P \ \& \ Q)$
 $(P \ \& \ \sim P) = \text{False} \quad (\sim P \ \& \ P) = \text{False}$
 $(P \mid \text{True}) = \text{True} \quad (\text{True} \mid P) = \text{True}$
 $(P \mid \text{False}) = P \quad (\text{False} \mid P) = P$
 $(P \mid P) = P \quad (P \mid (P \mid Q)) = (P \mid Q)$ **and**
 $(\text{ALL } x. P) = P \quad (\text{EX } x. P) = P \quad \text{EX } x. x=t \quad \text{EX } x. t=x$

— needed for the one-point-rule quantifier simplification procs
 — essential for termination!! **and**

!! P . $(EX\ x. x=t \ \&\ P(x)) = P(t)$
 !! P . $(EX\ x. t=x \ \&\ P(x)) = P(t)$
 !! P . $(ALL\ x. x=t \ \longrightarrow P(x)) = P(t)$
 !! P . $(ALL\ x. t=x \ \longrightarrow P(x)) = P(t)$
 $\langle proof \rangle$

lemma *disj-absorb*: $(A \mid A) = A$
 $\langle proof \rangle$

lemma *disj-left-absorb*: $(A \mid (A \mid B)) = (A \mid B)$
 $\langle proof \rangle$

lemma *conj-absorb*: $(A \ \&\ A) = A$
 $\langle proof \rangle$

lemma *conj-left-absorb*: $(A \ \&\ (A \ \&\ B)) = (A \ \&\ B)$
 $\langle proof \rangle$

lemma *eq-ac*:
 shows *eq-commute*: $(a=b) = (b=a)$
 and *eq-left-commute*: $(P=(Q=R)) = (Q=(P=R))$
 and *eq-assoc*: $((P=Q)=R) = (P=(Q=R))$ $\langle proof \rangle$
lemma *neq-commute*: $(a \sim b) = (b \sim a)$ $\langle proof \rangle$

lemma *conj-comms*:
 shows *conj-commute*: $(P \ \&\ Q) = (Q \ \&\ P)$
 and *conj-left-commute*: $(P \ \&\ (Q \ \&\ R)) = (Q \ \&\ (P \ \&\ R))$ $\langle proof \rangle$
lemma *conj-assoc*: $((P \ \&\ Q) \ \&\ R) = (P \ \&\ (Q \ \&\ R))$ $\langle proof \rangle$

lemmas *conj-ac* = *conj-commute conj-left-commute conj-assoc*

lemma *disj-comms*:
 shows *disj-commute*: $(P \mid Q) = (Q \mid P)$
 and *disj-left-commute*: $(P \mid (Q \mid R)) = (Q \mid (P \mid R))$ $\langle proof \rangle$
lemma *disj-assoc*: $((P \mid Q) \mid R) = (P \mid (Q \mid R))$ $\langle proof \rangle$

lemmas *disj-ac* = *disj-commute disj-left-commute disj-assoc*

lemma *conj-disj-distribL*: $(P \ \&\ (Q \mid R)) = (P \ \&\ Q \mid P \ \&\ R)$ $\langle proof \rangle$
lemma *conj-disj-distribR*: $((P \mid Q) \ \&\ R) = (P \ \&\ R \mid Q \ \&\ R)$ $\langle proof \rangle$

lemma *disj-conj-distribL*: $(P \mid (Q \ \&\ R)) = ((P \mid Q) \ \&\ (P \mid R))$ $\langle proof \rangle$
lemma *disj-conj-distribR*: $((P \ \&\ Q) \mid R) = ((P \ \&\ R) \ \&\ (Q \ \&\ R))$ $\langle proof \rangle$

lemma *imp-conjR*: $(P \ \longrightarrow (Q \ \&\ R)) = ((P \ \longrightarrow Q) \ \&\ (P \ \longrightarrow R))$ $\langle proof \rangle$
lemma *imp-conjL*: $((P \ \&\ Q) \ \longrightarrow R) = (P \ \longrightarrow (Q \ \longrightarrow R))$ $\langle proof \rangle$
lemma *imp-disjL*: $((P \mid Q) \ \longrightarrow R) = ((P \ \longrightarrow R) \ \&\ (Q \ \longrightarrow R))$ $\langle proof \rangle$

These two are specialized, but *imp-disj-not1* is useful in *Auth/Yahalom*.

lemma *imp-disj-not1*: $(P \multimap Q \mid R) = (\sim Q \multimap P \multimap R) \langle proof \rangle$

lemma *imp-disj-not2*: $(P \multimap Q \mid R) = (\sim R \multimap P \multimap Q) \langle proof \rangle$

lemma *imp-disj1*: $((P \multimap Q) \mid R) = (P \multimap Q \mid R) \langle proof \rangle$

lemma *imp-disj2*: $(Q \mid (P \multimap R)) = (P \multimap Q \mid R) \langle proof \rangle$

lemma *imp-cong*: $(P = P') \implies (P' \implies (Q = Q')) \implies ((P \multimap Q) = (P' \multimap Q')) \langle proof \rangle$

lemma *de-Morgan-disj*: $(\sim(P \mid Q)) = (\sim P \ \& \ \sim Q) \langle proof \rangle$

lemma *de-Morgan-conj*: $(\sim(P \ \& \ Q)) = (\sim P \mid \sim Q) \langle proof \rangle$

lemma *not-imp*: $(\sim(P \multimap Q)) = (P \ \& \ \sim Q) \langle proof \rangle$

lemma *not-iff*: $(P \sim Q) = (P = (\sim Q)) \langle proof \rangle$

lemma *disj-not1*: $(\sim P \mid Q) = (P \multimap Q) \langle proof \rangle$

lemma *disj-not2*: $(P \mid \sim Q) = (Q \multimap P) \text{ — changes orientation } :-(\langle proof \rangle)$

lemma *imp-conv-disj*: $(P \multimap Q) = ((\sim P) \mid Q) \langle proof \rangle$

lemma *iff-conv-conj-imp*: $(P = Q) = ((P \multimap Q) \ \& \ (Q \multimap P)) \langle proof \rangle$

lemma *cases-simp*: $((P \multimap Q) \ \& \ (\sim P \multimap Q)) = Q$

— Avoids duplication of subgoals after *split-if*, when the true and false

— cases boil down to the same thing.

$\langle proof \rangle$

lemma *not-all*: $(\sim (! x. P(x))) = (? x. \sim P(x)) \langle proof \rangle$

lemma *imp-all*: $((! x. P \ x) \multimap Q) = (? x. P \ x \multimap Q) \langle proof \rangle$

lemma *not-ex*: $(\sim (? x. P(x))) = (! x. \sim P(x)) \langle proof \rangle$

lemma *imp-ex*: $((? x. P \ x) \multimap Q) = (! x. P \ x \multimap Q) \langle proof \rangle$

lemma *all-not-ex*: $(ALL \ x. P \ x) = (\sim (EX \ x. \sim P \ x)) \langle proof \rangle$

declare *All-def* [noatp]

lemma *ex-disj-distrib*: $(? x. P(x) \mid Q(x)) = ((? x. P(x)) \mid (? x. Q(x))) \langle proof \rangle$

lemma *all-conj-distrib*: $(!x. P(x) \ \& \ Q(x)) = ((! x. P(x)) \ \& \ (! x. Q(x))) \langle proof \rangle$

The $\&$ congruence rule: not included by default! May slow rewrite proofs down by as much as 50%

lemma *conj-cong*:

$(P = P') \implies (P' \implies (Q = Q')) \implies ((P \ \& \ Q) = (P' \ \& \ Q'))$

$\langle proof \rangle$

lemma *rev-conj-cong*:

$(Q = Q') \implies (Q' \implies (P = P')) \implies ((P \ \& \ Q) = (P' \ \& \ Q'))$

$\langle proof \rangle$

The $|$ congruence rule: not included by default!

lemma *disj-cong*:

$$(P = P') ==> (\sim P' ==> (Q = Q')) ==> ((P | Q) = (P' | Q'))$$

<proof>

if-then-else rules

lemma *if-True*: $(\text{if True then } x \text{ else } y) = x$
<proof>

lemma *if-False*: $(\text{if False then } x \text{ else } y) = y$
<proof>

lemma *if-P*: $P ==> (\text{if } P \text{ then } x \text{ else } y) = x$
<proof>

lemma *if-not-P*: $\sim P ==> (\text{if } P \text{ then } x \text{ else } y) = y$
<proof>

lemma *split-if*: $P (\text{if } Q \text{ then } x \text{ else } y) = ((Q \dashrightarrow P(x)) \ \& \ (\sim Q \dashrightarrow P(y)))$
<proof>

lemma *split-if-asm*: $P (\text{if } Q \text{ then } x \text{ else } y) = (\sim((Q \ \& \ \sim P \ x) | (\sim Q \ \& \ \sim P \ y)))$
<proof>

lemmas *if-splits* [noatp] = *split-if split-if-asm*

lemma *if-cancel*: $(\text{if } c \text{ then } x \text{ else } x) = x$
<proof>

lemma *if-eq-cancel*: $(\text{if } x = y \text{ then } y \text{ else } x) = x$
<proof>

lemma *if-bool-eq-conj*: $(\text{if } P \text{ then } Q \text{ else } R) = ((P \dashrightarrow Q) \ \& \ (\sim P \dashrightarrow R))$
 — This form is useful for expanding *ifs* on the RIGHT of the $==>$ symbol.
<proof>

lemma *if-bool-eq-disj*: $(\text{if } P \text{ then } Q \text{ else } R) = ((P \ \& \ Q) | (\sim P \ \& \ R))$
 — And this form is useful for expanding *ifs* on the LEFT.
<proof>

lemma *Eq-TrueI*: $P ==> P == \text{True}$ *<proof>*

lemma *Eq-FalseI*: $\sim P ==> P == \text{False}$ *<proof>*

let rules for *simproc*

lemma *Let-folded*: $f \ x \equiv g \ x \implies \text{Let } x \ f \equiv \text{Let } x \ g$
<proof>

lemma *Let-unfold*: $f \ x \equiv g \implies \text{Let } x \ f \equiv g$

$\langle proof \rangle$

The following copy of the implication operator is useful for fine-tuning congruence rules. It instructs the simplifier to simplify its premise.

constdefs

simp-implies :: [*prop*, *prop*] => *prop* (**infixr** =*simp*=> 1)
 $[code\ del]:\ simp-implies \equiv op ==>$

lemma *simp-impliesI*:

assumes *PQ*: (*PROP P* \implies *PROP Q*)
shows *PROP P* =*simp*=> *PROP Q*
 $\langle proof \rangle$

lemma *simp-impliesE*:

assumes *PQ*: *PROP P* =*simp*=> *PROP Q*
and *P*: *PROP P*
and *QR*: *PROP Q* \implies *PROP R*
shows *PROP R*
 $\langle proof \rangle$

lemma *simp-implies-cong*:

assumes *PP'*: *PROP P* == *PROP P'*
and *P'QQ'*: *PROP P'* ==> (*PROP Q* == *PROP Q'*)
shows (*PROP P* =*simp*=> *PROP Q*) == (*PROP P'* =*simp*=> *PROP Q'*)
 $\langle proof \rangle$

lemma *uncurry*:

assumes *P* \longrightarrow *Q* \longrightarrow *R*
shows *P* \wedge *Q* \longrightarrow *R*
 $\langle proof \rangle$

lemma *iff-allI*:

assumes $\bigwedge x. P\ x = Q\ x$
shows $(\forall x. P\ x) = (\forall x. Q\ x)$
 $\langle proof \rangle$

lemma *iff-exI*:

assumes $\bigwedge x. P\ x = Q\ x$
shows $(\exists x. P\ x) = (\exists x. Q\ x)$
 $\langle proof \rangle$

lemma *all-comm*:

$(\forall x\ y. P\ x\ y) = (\forall y\ x. P\ x\ y)$
 $\langle proof \rangle$

lemma *ex-comm*:

$(\exists x\ y. P\ x\ y) = (\exists y\ x. P\ x\ y)$
 $\langle proof \rangle$

$\langle ML \rangle$

Simproc for proving $(y = x) == \text{False}$ from premise $\sim(x = y)$:

$\langle ML \rangle$

lemma *True-implies-equals*: $(\text{True} \implies \text{PROP } P) \equiv \text{PROP } P$
 $\langle \text{proof} \rangle$

lemma *ex-simps*:

$!!P \ Q. (EX \ x. P \ x \ \& \ Q) = ((EX \ x. P \ x) \ \& \ Q)$
 $!!P \ Q. (EX \ x. P \ \& \ Q \ x) = (P \ \& \ (EX \ x. Q \ x))$
 $!!P \ Q. (EX \ x. P \ x \ | \ Q) = ((EX \ x. P \ x) \ | \ Q)$
 $!!P \ Q. (EX \ x. P \ | \ Q \ x) = (P \ | \ (EX \ x. Q \ x))$
 $!!P \ Q. (EX \ x. P \ x \ --> \ Q) = ((EX \ x. P \ x) \ --> \ Q)$
 $!!P \ Q. (EX \ x. P \ --> \ Q \ x) = (P \ --> \ (EX \ x. Q \ x))$
 — Miniscoping: pushing in existential quantifiers.
 $\langle \text{proof} \rangle$

lemma *all-simps*:

$!!P \ Q. (ALL \ x. P \ x \ \& \ Q) = ((ALL \ x. P \ x) \ \& \ Q)$
 $!!P \ Q. (ALL \ x. P \ \& \ Q \ x) = (P \ \& \ (ALL \ x. Q \ x))$
 $!!P \ Q. (ALL \ x. P \ x \ | \ Q) = ((ALL \ x. P \ x) \ | \ Q)$
 $!!P \ Q. (ALL \ x. P \ | \ Q \ x) = (P \ | \ (ALL \ x. Q \ x))$
 $!!P \ Q. (ALL \ x. P \ x \ --> \ Q) = ((EX \ x. P \ x) \ --> \ Q)$
 $!!P \ Q. (ALL \ x. P \ --> \ Q \ x) = (P \ --> \ (ALL \ x. Q \ x))$
 — Miniscoping: pushing in universal quantifiers.
 $\langle \text{proof} \rangle$

lemmas $[simp] =$
triv-forall-equality
True-implies-equals
if-True
if-False
if-cancel
if-eq-cancel
imp-disjL

conj-assoc
disj-assoc
de-Morgan-conj
de-Morgan-disj
imp-disj1
imp-disj2
not-imp
disj-not1
not-all
not-ex
cases-simp
the-eq-trivial

the-sym-eq-trivial
ex-simps
all-simps
simp-thms

lemmas $[cong] = imp-cong \text{ } simp-implies-cong$
lemmas $[split] = split-if$

$\langle ML \rangle$

Simplifies x assuming c and y assuming $\neg c$

lemma *if-cong*:
assumes $b = c$
and $c \implies x = u$
and $\neg c \implies y = v$
shows $(if\ b\ then\ x\ else\ y) = (if\ c\ then\ u\ else\ v)$
 $\langle proof \rangle$

Prevents simplification of x and y: faster and allows the execution of functional programs.

lemma *if-weak-cong* $[cong]$:
assumes $b = c$
shows $(if\ b\ then\ x\ else\ y) = (if\ c\ then\ x\ else\ y)$
 $\langle proof \rangle$

Prevents simplification of t: much faster

lemma *let-weak-cong*:
assumes $a = b$
shows $(let\ x = a\ in\ t\ x) = (let\ x = b\ in\ t\ x)$
 $\langle proof \rangle$

To tidy up the result of a simproc. Only the RHS will be simplified.

lemma *eq-cong2*:
assumes $u = u'$
shows $(t \equiv u) \equiv (t \equiv u')$
 $\langle proof \rangle$

lemma *if-distrib*:
 $f\ (if\ c\ then\ x\ else\ y) = (if\ c\ then\ f\ x\ else\ f\ y)$
 $\langle proof \rangle$

This lemma restricts the effect of the rewrite rule $u=v$ to the left-hand side of an equality. Used in $\{Integ, Real\}/simproc.ML$

lemma *restrict-to-left*:
assumes $x = y$
shows $(x = z) = (y = z)$
 $\langle proof \rangle$

1.3.3 Generic cases and induction

Rule projections:

$\langle ML \rangle$

constdefs

induct-forall **where** *induct-forall* $P == \forall x. P\ x$
induct-implies **where** *induct-implies* $A\ B == A \longrightarrow B$
induct-equal **where** *induct-equal* $x\ y == x = y$
induct-conj **where** *induct-conj* $A\ B == A \wedge B$

lemma *induct-forall-eq*: $(!!x. P\ x) == \text{Trueprop } (\text{induct-forall } (\lambda x. P\ x))$
 $\langle \text{proof} \rangle$

lemma *induct-implies-eq*: $(A ==> B) == \text{Trueprop } (\text{induct-implies } A\ B)$
 $\langle \text{proof} \rangle$

lemma *induct-equal-eq*: $(x == y) == \text{Trueprop } (\text{induct-equal } x\ y)$
 $\langle \text{proof} \rangle$

lemma *induct-conj-eq*: $(A \ \&\&\& \ B) == \text{Trueprop } (\text{induct-conj } A\ B)$
 $\langle \text{proof} \rangle$

lemmas *induct-atomize* = *induct-forall-eq induct-implies-eq induct-equal-eq induct-conj-eq*

lemmas *induct-rulify* [symmetric, standard] = *induct-atomize*

lemmas *induct-rulify-fallback* =
induct-forall-def induct-implies-def induct-equal-def induct-conj-def

lemma *induct-forall-conj*: *induct-forall* $(\lambda x. \text{induct-conj } (A\ x) (B\ x)) =$
induct-conj (*induct-forall* A) (*induct-forall* B)
 $\langle \text{proof} \rangle$

lemma *induct-implies-conj*: *induct-implies* C (*induct-conj* $A\ B$) =
induct-conj (*induct-implies* $C\ A$) (*induct-implies* $C\ B$)
 $\langle \text{proof} \rangle$

lemma *induct-conj-curry*: $(\text{induct-conj } A\ B ==> \text{PROP } C) == (A ==> B ==>$
 $\text{PROP } C)$
 $\langle \text{proof} \rangle$

lemmas *induct-conj* = *induct-forall-conj induct-implies-conj induct-conj-curry*

hide *const induct-forall induct-implies induct-equal induct-conj*

Method setup.

$\langle ML \rangle$

1.3.4 Coherent logic

$\langle ML \rangle$

1.4 Other simple lemmas and lemma duplicates

lemma *Let-0* [*simp*]: *Let* 0 $f = f$ 0
 $\langle proof \rangle$

lemma *Let-1* [*simp*]: *Let* 1 $f = f$ 1
 $\langle proof \rangle$

lemma *ex1-eq* [*iff*]: $EX! x. x = t \rightarrow EX! x. t = x$
 $\langle proof \rangle$

lemma *choice-eq*: $(ALL x. EX! y. P x y) = (EX! f. ALL x. P x (f x))$
 $\langle proof \rangle$

lemma *mk-left-commute*:
fixes f (**infix** \otimes 60)
assumes a : $\bigwedge x y z. (x \otimes y) \otimes z = x \otimes (y \otimes z)$ **and**
 c : $\bigwedge x y. x \otimes y = y \otimes x$
shows $x \otimes (y \otimes z) = y \otimes (x \otimes z)$
 $\langle proof \rangle$

lemmas *eq-sym-conv* = *eq-commute*

lemma *nnf-simps*:
 $(\neg(P \wedge Q)) = (\neg P \vee \neg Q) \quad (\neg(P \vee Q)) = (\neg P \wedge \neg Q) \quad (P \longrightarrow Q) = (\neg P \vee Q)$
 $(P = Q) = ((P \wedge Q) \vee (\neg P \wedge \neg Q)) \quad (\neg(P = Q)) = ((P \wedge \neg Q) \vee (\neg P \wedge Q))$
 $(\neg \neg(P)) = P$
 $\langle proof \rangle$

1.5 Basic ML bindings

$\langle ML \rangle$

1.6 Code generator basics – see further theory *Code-Setup*

Equality

class *eq* =
fixes $eq :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *eq-equals*: $eq x y \longleftrightarrow x = y$
begin

lemma *eq*: $eq = (op =)$
 $\langle proof \rangle$

lemma *eq-refl*: $eq\ x\ x \longleftrightarrow True$
 $\langle proof \rangle$

end

Module setup

$\langle ML \rangle$

1.7 Nitpick hooks

This will be relocated once Nitpick is moved to HOL.

$\langle ML \rangle$

1.8 Legacy tactics and ML bindings

$\langle ML \rangle$

end

2 Code-Setup: Setup of code generators and related tools

theory *Code-Setup*
imports *HOL*
begin

2.1 Generic code generator foundation

Datatypes

code-datatype *True False*

code-datatype *TYPE('a::{})*

code-datatype *Trueprop prop*

Code equations

lemma [*code*]:
shows $(True \Longrightarrow PROP\ P) \equiv PROP\ P$
and $(False \Longrightarrow Q) \equiv Trueprop\ True$
and $(PROP\ P \Longrightarrow True) \equiv Trueprop\ True$
and $(Q \Longrightarrow False) \equiv Trueprop\ (\neg Q) \langle proof \rangle$

lemma [*code*]:
shows $False \wedge x \longleftrightarrow False$
and $True \wedge x \longleftrightarrow x$
and $x \wedge False \longleftrightarrow False$

and $x \wedge \text{True} \longleftrightarrow x$ $\langle \text{proof} \rangle$

lemma $[code]$:
shows $\text{False} \vee x \longleftrightarrow x$
and $\text{True} \vee x \longleftrightarrow \text{True}$
and $x \vee \text{False} \longleftrightarrow x$
and $x \vee \text{True} \longleftrightarrow \text{True}$ $\langle \text{proof} \rangle$

lemma $[code]$:
shows $\neg \text{True} \longleftrightarrow \text{False}$
and $\neg \text{False} \longleftrightarrow \text{True}$ $\langle \text{proof} \rangle$

lemmas $[code] = \text{Let-def if-True if-False}$

lemmas $[code, code\ unfold, symmetric, code\ post] = \text{imp-conv-disj}$

Equality

context eq
begin

lemma $\text{equals-eq} [code\ inline, code]$: $op = \equiv eq$
 $\langle \text{proof} \rangle$

declare $eq [code\ unfold, code\ inline\ del]$

declare $\text{equals-eq} [symmetric, code\ post]$

end

declare $\text{simp-thms}(6) [code\ nbe]$

hide (open) const eq
hide const eq

$\langle ML \rangle$

Cases

lemma Let-case-cert :
assumes $\text{CASE} \equiv (\lambda x. \text{Let } x\ f)$
shows $\text{CASE } x \equiv f\ x$
 $\langle \text{proof} \rangle$

lemma If-case-cert :
assumes $\text{CASE} \equiv (\lambda b. \text{If } b\ f\ g)$
shows $(\text{CASE } \text{True} \equiv f) \ \&\&\& (\text{CASE } \text{False} \equiv g)$
 $\langle \text{proof} \rangle$

$\langle ML \rangle$

code-abort *undefined*

2.2 Generic code generator preprocessor

$\langle ML \rangle$

2.3 Generic code generator target languages

type bool

code-type *bool*

(*SML bool*)

(*OCaml bool*)

(*Haskell Bool*)

code-const *True and False and Not and op & and op | and If*

(*SML true and false and not*

and **infixl** 1 *andalso and infixl* 0 *orelse*

and *!(if (-)/ then (-)/ else (-))*)

(*OCaml true and false and not*

and **infixl** 4 *&& and infixl* 2 *||*

and *!(if (-)/ then (-)/ else (-))*)

(*Haskell True and False and not*

and **infixl** 3 *&& and infixl* 2 *||*

and *!(if (-)/ then (-)/ else (-))*)

code-reserved *SML*

bool true false not

code-reserved *OCaml*

bool not

using built-in Haskell equality

code-class *eq*

(*Haskell Eq*)

code-const *eq-class.eq*

(*Haskell infixl* 4 *==*)

code-const *op =*

(*Haskell infixl* 4 *==*)

undefined

code-const *undefined*

(*SML !(raise/ Fail/ undefined)*)

(*OCaml failwith/ undefined*)

(*Haskell error/ undefined*)

2.4 SML code generator setup

types-code

```

  bool (bool)
attach (term-of) ⟨⟨
  fun term-of-bool b = if b then HOLogic.true-const else HOLogic.false-const;
  ⟩⟩
attach (test) ⟨⟨
  fun gen-bool i =
    let val b = one-of [false, true]
    in (b, fn () => term-of-bool b) end;
  ⟩⟩
  prop (bool)
attach (term-of) ⟨⟨
  fun term-of-prop b =
    HOLogic.mk-Trueprop (if b then HOLogic.true-const else HOLogic.false-const);
  ⟩⟩

```

consts-code

```

  Trueprop ((-))
  True (true)
  False (false)
  Not (Bool.not)
  op | ((- orelse/ -))
  op & ((- andalso/ -))
  If ((if -/ then -/ else -))

```

⟨ML⟩

2.5 Evaluation and normalization by evaluation

⟨ML⟩

2.6 Quickcheck

⟨ML⟩

```

quickcheck-params [size = 5, iterations = 50]

```

end

3 Orderings: Abstract orderings

```

theory Orderings
imports Code-Setup
uses ~~/src/Provers/order.ML
begin

```


3.1 Quasi orders

class *preorder* = *ord* +
assumes *less-le-not-le*: $x < y \longleftrightarrow x \leq y \wedge \neg (y \leq x)$
and *order-refl* [*iff*]: $x \leq x$
and *order-trans*: $x \leq y \implies y \leq z \implies x \leq z$
begin

Reflexivity.

lemma *eq-refl*: $x = y \implies x \leq y$
 — This form is useful with the classical reasoner.
<proof>

lemma *less-irrefl* [*iff*]: $\neg x < x$
<proof>

lemma *less-imp-le*: $x < y \implies x \leq y$
<proof>

Asymmetry.

lemma *less-not-sym*: $x < y \implies \neg (y < x)$
<proof>

lemma *less-asym*: $x < y \implies (\neg P \implies y < x) \implies P$
<proof>

Transitivity.

lemma *less-trans*: $x < y \implies y < z \implies x < z$
<proof>

lemma *le-less-trans*: $x \leq y \implies y < z \implies x < z$
<proof>

lemma *less-le-trans*: $x < y \implies y \leq z \implies x < z$
<proof>

Useful for simplification, but too risky to include by default.

lemma *less-imp-not-less*: $x < y \implies (\neg y < x) \longleftrightarrow \text{True}$
<proof>

lemma *less-imp-triv*: $x < y \implies (y < x \longrightarrow P) \longleftrightarrow \text{True}$
<proof>

Transitivity rules for calculational reasoning

lemma *less-asym'*: $a < b \implies b < a \implies P$
<proof>

Dual order

lemma *dual-preorder*:
 $preorder\ (op \geq)\ (op >)$
 $\langle proof \rangle$

end

3.2 Partial orders

class *order* = *preorder* +
assumes *antisym*: $x \leq y \implies y \leq x \implies x = y$
begin

Reflexivity.

lemma *less-le*: $x < y \longleftrightarrow x \leq y \wedge x \neq y$
 $\langle proof \rangle$

lemma *le-less*: $x \leq y \longleftrightarrow x < y \vee x = y$
 — NOT suitable for iff, since it can cause PROOF FAILED.
 $\langle proof \rangle$

lemma *le-imp-less-or-eq*: $x \leq y \implies x < y \vee x = y$
 $\langle proof \rangle$

Useful for simplification, but too risky to include by default.

lemma *less-imp-not-eq*: $x < y \implies (x = y) \longleftrightarrow False$
 $\langle proof \rangle$

lemma *less-imp-not-eq2*: $x < y \implies (y = x) \longleftrightarrow False$
 $\langle proof \rangle$

Transitivity rules for calculational reasoning

lemma *neq-le-trans*: $a \neq b \implies a \leq b \implies a < b$
 $\langle proof \rangle$

lemma *le-neq-trans*: $a \leq b \implies a \neq b \implies a < b$
 $\langle proof \rangle$

Asymmetry.

lemma *eq-iff*: $x = y \longleftrightarrow x \leq y \wedge y \leq x$
 $\langle proof \rangle$

lemma *antisym-conv*: $y \leq x \implies x \leq y \longleftrightarrow x = y$
 $\langle proof \rangle$

lemma *less-imp-neq*: $x < y \implies x \neq y$
 $\langle proof \rangle$

Least value operator

definition (in *ord*)

Least :: ('a \Rightarrow bool) \Rightarrow 'a (**binder** *LEAST* 10) **where**
Least *P* = (*THE* *x*. *P* *x* \wedge (\forall *y*. *P* *y* \longrightarrow *x* \leq *y*))

lemma *Least-equality*:

assumes *P* *x*
and $\bigwedge y. P\ y \Longrightarrow x \leq y$
shows *Least* *P* = *x*
 \langle *proof* \rangle

lemma *LeastI2-order*:

assumes *P* *x*
and $\bigwedge y. P\ y \Longrightarrow x \leq y$
and $\bigwedge x. P\ x \Longrightarrow \forall y. P\ y \longrightarrow x \leq y \Longrightarrow Q\ x$
shows *Q* (*Least* *P*)
 \langle *proof* \rangle

Dual order

lemma *dual-order*:

order (*op* \geq) (*op* $>$)
 \langle *proof* \rangle

end

3.3 Linear (total) orders

class *linorder* = *order* +

assumes *linear*: $x \leq y \vee y \leq x$
begin

lemma *less-linear*: $x < y \vee x = y \vee y < x$
 \langle *proof* \rangle

lemma *le-less-linear*: $x \leq y \vee y < x$
 \langle *proof* \rangle

lemma *le-cases* [*case-names* *le* *ge*]:

$(x \leq y \Longrightarrow P) \Longrightarrow (y \leq x \Longrightarrow P) \Longrightarrow P$
 \langle *proof* \rangle

lemma *linorder-cases* [*case-names* *less* *equal* *greater*]:

$(x < y \Longrightarrow P) \Longrightarrow (x = y \Longrightarrow P) \Longrightarrow (y < x \Longrightarrow P) \Longrightarrow P$
 \langle *proof* \rangle

lemma *not-less*: $\neg x < y \longleftrightarrow y \leq x$
 \langle *proof* \rangle

lemma *not-less-iff-gr-or-eq*:

$\neg(x < y) \longleftrightarrow (x > y \mid x = y)$

$\langle \text{proof} \rangle$

lemma *not-le*: $\neg x \leq y \longleftrightarrow y < x$

$\langle \text{proof} \rangle$

lemma *neq-iff*: $x \neq y \longleftrightarrow x < y \vee y < x$

$\langle \text{proof} \rangle$

lemma *neqE*: $x \neq y \implies (x < y \implies R) \implies (y < x \implies R) \implies R$

$\langle \text{proof} \rangle$

lemma *antisym-conv1*: $\neg x < y \implies x \leq y \longleftrightarrow x = y$

$\langle \text{proof} \rangle$

lemma *antisym-conv2*: $x \leq y \implies \neg x < y \longleftrightarrow x = y$

$\langle \text{proof} \rangle$

lemma *antisym-conv3*: $\neg y < x \implies \neg x < y \longleftrightarrow x = y$

$\langle \text{proof} \rangle$

lemma *leI*: $\neg x < y \implies y \leq x$

$\langle \text{proof} \rangle$

lemma *leD*: $y \leq x \implies \neg x < y$

$\langle \text{proof} \rangle$

lemma *not-leE*: $\neg y \leq x \implies x < y$

$\langle \text{proof} \rangle$

Dual order

lemma *dual-linorder*:

linorder (*op* \geq) (*op* $>$)

$\langle \text{proof} \rangle$

min/max

definition (*in ord*) *min* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**

[*code del*]: *min* *a b* = (*if* $a \leq b$ *then* *a* *else* *b*)

definition (*in ord*) *max* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**

[*code del*]: *max* *a b* = (*if* $a \leq b$ *then* *b* *else* *a*)

lemma *min-le-iff-disj*:

min *x y* $\leq z \longleftrightarrow x \leq z \vee y \leq z$

$\langle \text{proof} \rangle$

lemma *le-max-iff-disj*:

$z \leq \text{max } x y \longleftrightarrow z \leq x \vee z \leq y$

$\langle \text{proof} \rangle$

lemma *min-less-iff-disj*:

$\min x\ y < z \iff x < z \vee y < z$
 $\langle \text{proof} \rangle$

lemma *less-max-iff-disj*:

$z < \max x\ y \iff z < x \vee z < y$
 $\langle \text{proof} \rangle$

lemma *min-less-iff-conj* [*simp*]:

$z < \min x\ y \iff z < x \wedge z < y$
 $\langle \text{proof} \rangle$

lemma *max-less-iff-conj* [*simp*]:

$\max x\ y < z \iff x < z \wedge y < z$
 $\langle \text{proof} \rangle$

lemma *split-min* [*noatp*]:

$P (\min i\ j) \iff (i \leq j \longrightarrow P\ i) \wedge (\neg i \leq j \longrightarrow P\ j)$
 $\langle \text{proof} \rangle$

lemma *split-max* [*noatp*]:

$P (\max i\ j) \iff (i \leq j \longrightarrow P\ j) \wedge (\neg i \leq j \longrightarrow P\ i)$
 $\langle \text{proof} \rangle$

end

Explicit dictionaries for code generation

lemma *min-ord-min* [*code*, *code unfold*, *code inline del*]:

$\min = \text{ord.min } (op \leq)$
 $\langle \text{proof} \rangle$

declare *ord.min-def* [*code*]

lemma *max-ord-max* [*code*, *code unfold*, *code inline del*]:

$\max = \text{ord.max } (op \leq)$
 $\langle \text{proof} \rangle$

declare *ord.max-def* [*code*]

3.4 Reasoning tools setup

$\langle ML \rangle$

Declarations to set up transitivity reasoner of partial and linear orders.

context *order*

begin

```

declare less-irrefl [THEN notE, order add less-reflE: order op = :: 'a ⇒ 'a ⇒
bool op <= op <]

declare order-refl [order add le-refl: order op = :: 'a => 'a => bool op <= op
<]

declare less-imp-le [order add less-imp-le: order op = :: 'a => 'a => bool op <=
op <]

declare antisym [order add eqI: order op = :: 'a => 'a => bool op <= op <]

declare eq-refl [order add eqD1: order op = :: 'a => 'a => bool op <= op <]

declare sym [THEN eq-refl, order add eqD2: order op = :: 'a => 'a => bool op
<= op <]

declare less-trans [order add less-trans: order op = :: 'a => 'a => bool op <=
op <]

declare less-le-trans [order add less-le-trans: order op = :: 'a => 'a => bool op
<= op <]

declare le-less-trans [order add le-less-trans: order op = :: 'a => 'a => bool op
<= op <]

declare order-trans [order add le-trans: order op = :: 'a => 'a => bool op <=
op <]

declare le-neq-trans [order add le-neq-trans: order op = :: 'a => 'a => bool op
<= op <]

declare neq-le-trans [order add neq-le-trans: order op = :: 'a => 'a => bool op
<= op <]

declare less-imp-neq [order add less-imp-neq: order op = :: 'a => 'a => bool op
<= op <]

declare eq-neq-eq-imp-neq [order add eq-neq-eq-imp-neq: order op = :: 'a => 'a
=> bool op <= op <]

declare not-sym [order add not-sym: order op = :: 'a => 'a => bool op <= op
<]

end

context linorder
begin

declare [[order del: order op = :: 'a => 'a => bool op <= op <]]

```

declare *less-irrefl* [*THEN notE*, *order add less-reflE*: *linorder op* = :: '*a* => '*a*
=> *bool op* <= *op* <]

declare *order-refl* [*order add le-refl*: *linorder op* = :: '*a* => '*a* => *bool op* <= *op*
<]

declare *less-imp-le* [*order add less-imp-le*: *linorder op* = :: '*a* => '*a* => *bool op*
<= *op* <]

declare *not-less* [*THEN iffD2*, *order add not-lessI*: *linorder op* = :: '*a* => '*a* =>
bool op <= *op* <]

declare *not-le* [*THEN iffD2*, *order add not-leI*: *linorder op* = :: '*a* => '*a* => *bool*
op <= *op* <]

declare *not-less* [*THEN iffD1*, *order add not-lessD*: *linorder op* = :: '*a* => '*a* =>
bool op <= *op* <]

declare *not-le* [*THEN iffD1*, *order add not-leD*: *linorder op* = :: '*a* => '*a* =>
bool op <= *op* <]

declare *antisym* [*order add eqI*: *linorder op* = :: '*a* => '*a* => *bool op* <= *op* <]

declare *eq-refl* [*order add eqD1*: *linorder op* = :: '*a* => '*a* => *bool op* <= *op* <]

declare *sym* [*THEN eq-refl*, *order add eqD2*: *linorder op* = :: '*a* => '*a* => *bool*
op <= *op* <]

declare *less-trans* [*order add less-trans*: *linorder op* = :: '*a* => '*a* => *bool op* <=
op <]

declare *less-le-trans* [*order add less-le-trans*: *linorder op* = :: '*a* => '*a* => *bool*
op <= *op* <]

declare *le-less-trans* [*order add le-less-trans*: *linorder op* = :: '*a* => '*a* => *bool*
op <= *op* <]

declare *order-trans* [*order add le-trans*: *linorder op* = :: '*a* => '*a* => *bool op* <=
op <]

declare *le-neq-trans* [*order add le-neq-trans*: *linorder op* = :: '*a* => '*a* => *bool op*
<= *op* <]

declare *neq-le-trans* [*order add neq-le-trans*: *linorder op* = :: '*a* => '*a* => *bool op*
<= *op* <]

declare *less-imp-neq* [*order add less-imp-neq*: *linorder op* = :: '*a* => '*a* => *bool*
op <= *op* <]

```
declare eq-neq-eq-imp-neq [order add eq-neq-eq-imp-neq: linorder op = :: 'a => 'a  
=> bool op <= op <]
```

```
declare not-sym [order add not-sym: linorder op = :: 'a => 'a => bool op <=  
op <]
```

```
end
```

⟨ML⟩

3.5 Name duplicates

```
lemmas order-less-le = less-le  
lemmas order-eq-refl = preorder-class.eq-refl  
lemmas order-less-irrefl = preorder-class.less-irrefl  
lemmas order-le-less = order-class.le-less  
lemmas order-le-imp-less-or-eq = order-class.le-imp-less-or-eq  
lemmas order-less-imp-le = preorder-class.less-imp-le  
lemmas order-less-imp-not-eq = order-class.less-imp-not-eq  
lemmas order-less-imp-not-eq2 = order-class.less-imp-not-eq2  
lemmas order-neq-le-trans = order-class.neq-le-trans  
lemmas order-le-neq-trans = order-class.le-neq-trans  
  
lemmas order-antisym = antisym  
lemmas order-less-not-sym = preorder-class.less-not-sym  
lemmas order-less-asym = preorder-class.less-asym  
lemmas order-eq-iff = order-class.eq-iff  
lemmas order-antisym-conv = order-class.antisym-conv  
lemmas order-less-trans = preorder-class.less-trans  
lemmas order-le-less-trans = preorder-class.le-less-trans  
lemmas order-less-le-trans = preorder-class.less-le-trans  
lemmas order-less-imp-not-less = preorder-class.less-imp-not-less  
lemmas order-less-imp-triv = preorder-class.less-imp-triv  
lemmas order-less-asym' = preorder-class.less-asym'  
  
lemmas linorder-linear = linear  
lemmas linorder-less-linear = linorder-class.less-linear  
lemmas linorder-le-less-linear = linorder-class.le-less-linear  
lemmas linorder-le-cases = linorder-class.le-cases  
lemmas linorder-not-less = linorder-class.not-less  
lemmas linorder-not-le = linorder-class.not-le  
lemmas linorder-neq-iff = linorder-class.neq-iff  
lemmas linorder-neqE = linorder-class.neqE  
lemmas linorder-antisym-conv1 = linorder-class.antisym-conv1  
lemmas linorder-antisym-conv2 = linorder-class.antisym-conv2  
lemmas linorder-antisym-conv3 = linorder-class.antisym-conv3
```


3.6 Bounded quantifiers

syntax

$-All-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ALL \text{ -<-./ -}) [0, 0, 10] 10)$
 $-Ex-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists EX \text{ -<-./ -}) [0, 0, 10] 10)$
 $-All-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ALL \text{ -<=.-./ -}) [0, 0, 10] 10)$
 $-Ex-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists EX \text{ -<=.-./ -}) [0, 0, 10] 10)$

 $-All-greater :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ALL \text{ ->.-./ -}) [0, 0, 10] 10)$
 $-Ex-greater :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists EX \text{ ->.-./ -}) [0, 0, 10] 10)$
 $-All-greater-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ALL \text{ ->=.-./ -}) [0, 0, 10] 10)$
 $-Ex-greater-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists EX \text{ ->=.-./ -}) [0, 0, 10] 10)$

syntax (xsymbols)

$-All-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ -<-./ -}) [0, 0, 10] 10)$
 $-Ex-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ -<-./ -}) [0, 0, 10] 10)$
 $-All-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ -<=.-./ -}) [0, 0, 10] 10)$
 $-Ex-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ -<=.-./ -}) [0, 0, 10] 10)$

 $-All-greater :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ ->.-./ -}) [0, 0, 10] 10)$
 $-Ex-greater :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ ->.-./ -}) [0, 0, 10] 10)$
 $-All-greater-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ ->=.-./ -}) [0, 0, 10] 10)$
 $-Ex-greater-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ ->=.-./ -}) [0, 0, 10] 10)$

syntax (HOL)

$-All-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ! \text{ -<-./ -}) [0, 0, 10] 10)$
 $-Ex-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ? \text{ -<-./ -}) [0, 0, 10] 10)$
 $-All-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ! \text{ -<=.-./ -}) [0, 0, 10] 10)$
 $-Ex-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists ? \text{ -<=.-./ -}) [0, 0, 10] 10)$

syntax (HTML output)

$-All-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ -<-./ -}) [0, 0, 10] 10)$
 $-Ex-less :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ -<-./ -}) [0, 0, 10] 10)$
 $-All-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ -<=.-./ -}) [0, 0, 10] 10)$
 $-Ex-less-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ -<=.-./ -}) [0, 0, 10] 10)$

 $-All-greater :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ ->.-./ -}) [0, 0, 10] 10)$
 $-Ex-greater :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ ->.-./ -}) [0, 0, 10] 10)$
 $-All-greater-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \forall \text{ ->=.-./ -}) [0, 0, 10] 10)$
 $-Ex-greater-eq :: [idt, 'a, bool] \Rightarrow bool \quad ((\exists \exists \text{ ->=.-./ -}) [0, 0, 10] 10)$

translations

$ALL\ x < y. P \Rightarrow ALL\ x. x < y \longrightarrow P$
 $EX\ x < y. P \Rightarrow EX\ x. x < y \wedge P$
 $ALL\ x <= y. P \Rightarrow ALL\ x. x <= y \longrightarrow P$
 $EX\ x <= y. P \Rightarrow EX\ x. x <= y \wedge P$
 $ALL\ x > y. P \Rightarrow ALL\ x. x > y \longrightarrow P$
 $EX\ x > y. P \Rightarrow EX\ x. x > y \wedge P$
 $ALL\ x >= y. P \Rightarrow ALL\ x. x >= y \longrightarrow P$
 $EX\ x >= y. P \Rightarrow EX\ x. x >= y \wedge P$

$\langle ML \rangle$

3.7 Transitivity reasoning

context *ord*

begin

lemma *ord-le-eq-trans*: $a \leq b \implies b = c \implies a \leq c$
 $\langle proof \rangle$

lemma *ord-eq-le-trans*: $a = b \implies b \leq c \implies a \leq c$
 $\langle proof \rangle$

lemma *ord-less-eq-trans*: $a < b \implies b = c \implies a < c$
 $\langle proof \rangle$

lemma *ord-eq-less-trans*: $a = b \implies b < c \implies a < c$
 $\langle proof \rangle$

end

lemma *order-less-subst2*: $(a::'a::order) < b \implies f\ b < (c::'c::order) \implies$
 $(!!x\ y. x < y \implies f\ x < f\ y) \implies f\ a < c$
 $\langle proof \rangle$

lemma *order-less-subst1*: $(a::'a::order) < f\ b \implies (b::'b::order) < c \implies$
 $(!!x\ y. x < y \implies f\ x < f\ y) \implies a < f\ c$
 $\langle proof \rangle$

lemma *order-le-less-subst2*: $(a::'a::order) <= b \implies f\ b < (c::'c::order) \implies$
 $(!!x\ y. x <= y \implies f\ x <= f\ y) \implies f\ a < c$
 $\langle proof \rangle$

lemma *order-le-less-subst1*: $(a::'a::order) <= f\ b \implies (b::'b::order) < c \implies$
 $(!!x\ y. x < y \implies f\ x < f\ y) \implies a < f\ c$
 $\langle proof \rangle$

lemma *order-less-le-subst2*: $(a::'a::order) < b \implies f\ b <= (c::'c::order) \implies$
 $(!!x\ y. x < y \implies f\ x < f\ y) \implies f\ a < c$
 $\langle proof \rangle$

lemma *order-less-le-subst1*: $(a::'a::order) < f\ b \implies (b::'b::order) <= c \implies$
 $(!!x\ y. x <= y \implies f\ x <= f\ y) \implies a < f\ c$
 $\langle proof \rangle$

lemma *order-subst1*: $(a::'a::order) <= f\ b \implies (b::'b::order) <= c \implies$
 $(!!x\ y. x <= y \implies f\ x <= f\ y) \implies a <= f\ c$
 $\langle proof \rangle$

lemma *order-subst2*: $(a::'a::order) \leq b \implies f b \leq (c::'c::order) \implies$
 $(!!x y. x \leq y \implies f x \leq f y) \implies f a \leq c$
 $\langle proof \rangle$

lemma *ord-le-eq-subst*: $a \leq b \implies f b = c \implies$
 $(!!x y. x \leq y \implies f x \leq f y) \implies f a \leq c$
 $\langle proof \rangle$

lemma *ord-eq-le-subst*: $a = f b \implies b \leq c \implies$
 $(!!x y. x \leq y \implies f x \leq f y) \implies a \leq f c$
 $\langle proof \rangle$

lemma *ord-less-eq-subst*: $a < b \implies f b = c \implies$
 $(!!x y. x < y \implies f x < f y) \implies f a < c$
 $\langle proof \rangle$

lemma *ord-eq-less-subst*: $a = f b \implies b < c \implies$
 $(!!x y. x < y \implies f x < f y) \implies a < f c$
 $\langle proof \rangle$

Note that this list of rules is in reverse order of priorities.

lemmas [*trans*] =
order-less-subst2
order-less-subst1
order-le-less-subst2
order-le-less-subst1
order-less-le-subst2
order-less-le-subst1
order-subst2
order-subst1
ord-le-eq-subst
ord-eq-le-subst
ord-less-eq-subst
ord-eq-less-subst
forw-subst
back-subst
rev-mp
mp

lemmas (**in** *order*) [*trans*] =
neq-le-trans
le-neq-trans

lemmas (**in** *preorder*) [*trans*] =
less-trans
less-asym'
le-less-trans
less-le-trans

order-trans

lemmas (in *order*) [*trans*] =
antisym

lemmas (in *ord*) [*trans*] =
ord-le-eq-trans
ord-eq-le-trans
ord-less-eq-trans
ord-eq-less-trans

lemmas [*trans*] =
trans

lemmas *order-trans-rules* =
order-less-subst2
order-less-subst1
order-le-less-subst2
order-le-less-subst1
order-less-le-subst2
order-less-le-subst1
order-subst2
order-subst1
ord-le-eq-subst
ord-eq-le-subst
ord-less-eq-subst
ord-eq-less-subst
forw-subst
back-subst
rev-mp
mp
neq-le-trans
le-neq-trans
less-trans
less-asym'
le-less-trans
less-le-trans
order-trans
antisym
ord-le-eq-trans
ord-eq-le-trans
ord-less-eq-trans
ord-eq-less-trans
trans

These support proving chains of decreasing inequalities $a \leq b \leq c \dots$ in Isar proofs.

lemma *xt1*:
 $a = b \implies b > c \implies a > c$

$$\begin{aligned}
a > b &\implies b = c \implies a > c \\
a = b &\implies b \geq c \implies a \geq c \\
a \geq b &\implies b = c \implies a \geq c \\
(x::'a::order) > y &\implies y \geq x \implies x = y \\
(x::'a::order) > y &\implies y \geq z \implies x \geq z \\
(x::'a::order) > y &\implies y \geq z \implies x > z \\
(x::'a::order) > y &\implies y > z \implies x > z \\
(a::'a::order) > b &\implies b > a \implies P \\
(x::'a::order) > y &\implies y > z \implies x > z \\
(a::'a::order) > b &\implies a \sim b \implies a > b \\
(a::'a::order) \sim b &\implies a \geq b \implies a > b \\
a = f b &\implies b > c \implies (!x y. x > y \implies f x > f y) \implies a > f c \\
a > b &\implies f b = c \implies (!x y. x > y \implies f x > f y) \implies f a > c \\
a = f b &\implies b \geq c \implies (!x y. x \geq y \implies f x \geq f y) \implies a \geq f c \\
a \geq b &\implies f b = c \implies (!x y. x \geq y \implies f x \geq f y) \implies f a \geq c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt2*:

$$\begin{aligned}
(a::'a::order) > f b &\implies b \geq c \implies (!x y. x \geq y \implies f x \geq f y) \implies \\
a &\geq f c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt3*: $(a::'a::order) > b \implies (f b::'b::order) \geq c \implies$

$$\begin{aligned}
(!x y. x \geq y \implies f x \geq f y) &\implies f a \geq c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt4*: $(a::'a::order) > f b \implies (b::'b::order) \geq c \implies$

$$\begin{aligned}
(!x y. x \geq y \implies f x \geq f y) &\implies a > f c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt5*: $(a::'a::order) > b \implies (f b::'b::order) \geq c \implies$

$$\begin{aligned}
(!x y. x > y \implies f x > f y) &\implies f a > c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt6*: $(a::'a::order) \geq f b \implies b > c \implies$

$$\begin{aligned}
(!x y. x > y \implies f x > f y) &\implies a > f c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt7*: $(a::'a::order) \geq b \implies (f b::'b::order) > c \implies$

$$\begin{aligned}
(!x y. x \geq y \implies f x \geq f y) &\implies f a > c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt8*: $(a::'a::order) > f b \implies (b::'b::order) > c \implies$

$$\begin{aligned}
(!x y. x > y \implies f x > f y) &\implies a > f c \\
\langle proof \rangle
\end{aligned}$$

lemma *xt9*: $(a::'a::order) > b \implies (f b::'b::order) > c \implies$

$$\begin{aligned}
(!x y. x > y \implies f x > f y) &\implies f a > c \\
\langle proof \rangle
\end{aligned}$$

lemmas *xtrans* = *xt1 xt2 xt3 xt4 xt5 xt6 xt7 xt8 xt9*

3.8 Monotonicity, least value operator and min/max

context *order*
begin

definition *mono* :: (*'a* \Rightarrow *'b::order*) \Rightarrow *bool* **where**
 $\text{mono } f \longleftrightarrow (\forall x y. x \leq y \longrightarrow f x \leq f y)$

lemma *monoI* [*intro?*]:
fixes *f* :: *'a* \Rightarrow *'b::order*
shows $(\bigwedge x y. x \leq y \Longrightarrow f x \leq f y) \Longrightarrow \text{mono } f$
 $\langle \text{proof} \rangle$

lemma *monoD* [*dest?*]:
fixes *f* :: *'a* \Rightarrow *'b::order*
shows $\text{mono } f \Longrightarrow x \leq y \Longrightarrow f x \leq f y$
 $\langle \text{proof} \rangle$

definition *strict-mono* :: (*'a* \Rightarrow *'b::order*) \Rightarrow *bool* **where**
 $\text{strict-mono } f \longleftrightarrow (\forall x y. x < y \longrightarrow f x < f y)$

lemma *strict-monoI* [*intro?*]:
assumes $\bigwedge x y. x < y \Longrightarrow f x < f y$
shows *strict-mono* *f*
 $\langle \text{proof} \rangle$

lemma *strict-monoD* [*dest?*]:
 $\text{strict-mono } f \Longrightarrow x < y \Longrightarrow f x < f y$
 $\langle \text{proof} \rangle$

lemma *strict-mono-mono* [*dest?*]:
assumes *strict-mono* *f*
shows *mono* *f*
 $\langle \text{proof} \rangle$

end

context *linorder*
begin

lemma *strict-mono-eq*:
assumes *strict-mono* *f*
shows $f x = f y \longleftrightarrow x = y$
 $\langle \text{proof} \rangle$

lemma *strict-mono-less-eq*:

```

assumes strict-mono f
shows  $f\ x \leq f\ y \iff x \leq y$ 
 $\langle proof \rangle$ 

```

```

lemma strict-mono-less:
assumes strict-mono f
shows  $f\ x < f\ y \iff x < y$ 
 $\langle proof \rangle$ 

```

```

lemma min-of-mono:
fixes  $f :: 'a \Rightarrow 'b::linorder$ 
shows  $mono\ f \implies min\ (f\ m)\ (f\ n) = f\ (min\ m\ n)$ 
 $\langle proof \rangle$ 

```

```

lemma max-of-mono:
fixes  $f :: 'a \Rightarrow 'b::linorder$ 
shows  $mono\ f \implies max\ (f\ m)\ (f\ n) = f\ (max\ m\ n)$ 
 $\langle proof \rangle$ 

```

end

```

lemma min-leastL:  $(!!x. least \leq x) \implies min\ least\ x = least$ 
 $\langle proof \rangle$ 

```

```

lemma max-leastL:  $(!!x. least \leq x) \implies max\ least\ x = x$ 
 $\langle proof \rangle$ 

```

```

lemma min-leastR:  $(\bigwedge x::'a::order. least \leq x) \implies min\ x\ least = least$ 
 $\langle proof \rangle$ 

```

```

lemma max-leastR:  $(\bigwedge x::'a::order. least \leq x) \implies max\ x\ least = x$ 
 $\langle proof \rangle$ 

```

3.9 Top and bottom elements

```

class top = preorder +
fixes top :: 'a
assumes top-greatest [simp]:  $x \leq top$ 

```

```

class bot = preorder +
fixes bot :: 'a
assumes bot-least [simp]:  $bot \leq x$ 

```

3.10 Dense orders

```

class dense-linear-order = linorder +
assumes gt-ex:  $\exists y. x < y$ 
and lt-ex:  $\exists y. y < x$ 
and dense:  $x < y \implies (\exists z. x < z \wedge z < y)$ 

```

3.11 Wellorders

class *wellorder* = *linorder* +
assumes *less-induct* [*case-names less*]: $(\bigwedge x. (\bigwedge y. y < x \implies P y) \implies P x) \implies P a$
begin

lemma *wellorder-Least-lemma*:
fixes *k* :: 'a
assumes *P k*
shows *P (LEAST x. P x)* **and** $(LEAST x. P x) \leq k$
 $\langle proof \rangle$

lemmas *LeastI* = *wellorder-Least-lemma*(1)
lemmas *Least-le* = *wellorder-Least-lemma*(2)

— The following 3 lemmas are due to Brian Huffman

lemma *LeastI-ex*: $\exists x. P x \implies P (Least P)$
 $\langle proof \rangle$

lemma *LeastI2*:
 $P a \implies (\bigwedge x. P x \implies Q x) \implies Q (Least P)$
 $\langle proof \rangle$

lemma *LeastI2-ex*:
 $\exists a. P a \implies (\bigwedge x. P x \implies Q x) \implies Q (Least P)$
 $\langle proof \rangle$

lemma *not-less-Least*: $k < (LEAST x. P x) \implies \neg P k$
 $\langle proof \rangle$

end

3.12 Order on bool

instantiation *bool* :: {*order*, *top*, *bot*}
begin

definition
le-bool-def [*code del*]: $P \leq Q \longleftrightarrow P \longrightarrow Q$

definition
less-bool-def [*code del*]: $(P::bool) < Q \longleftrightarrow \neg P \wedge Q$

definition
top-bool-eq: *top* = *True*

definition
bot-bool-eq: *bot* = *False*

instance $\langle proof \rangle$

end

lemma *le-boolI*: $(P \implies Q) \implies P \leq Q$
 $\langle proof \rangle$

lemma *le-boolI'*: $P \longrightarrow Q \implies P \leq Q$
 $\langle proof \rangle$

lemma *le-boolE*: $P \leq Q \implies P \implies (Q \implies R) \implies R$
 $\langle proof \rangle$

lemma *le-boolD*: $P \leq Q \implies P \longrightarrow Q$
 $\langle proof \rangle$

lemma [*code*]:
 $False \leq b \longleftrightarrow True$
 $True \leq b \longleftrightarrow b$
 $False < b \longleftrightarrow b$
 $True < b \longleftrightarrow False$
 $\langle proof \rangle$

3.13 Order on functions

instantiation *fun* :: (*type*, *ord*) *ord*
begin

definition
le-fun-def [*code del*]: $f \leq g \longleftrightarrow (\forall x. f\ x \leq g\ x)$

definition
less-fun-def [*code del*]: $(f :: 'a \Rightarrow 'b) < g \longleftrightarrow f \leq g \wedge \neg (g \leq f)$

instance $\langle proof \rangle$

end

instance *fun* :: (*type*, *preorder*) *preorder* $\langle proof \rangle$

instance *fun* :: (*type*, *order*) *order* $\langle proof \rangle$

instantiation *fun* :: (*type*, *top*) *top*
begin

definition
top-fun-eq: $top = (\lambda x. top)$

instance $\langle proof \rangle$

end

instantiation *fun* :: (*type*, *bot*) *bot*
begin

definition

bot-fun-eq: *bot* = ($\lambda x. \text{bot}$)

instance $\langle \text{proof} \rangle$

end

lemma *le-funI*: $(\bigwedge x. f\ x \leq g\ x) \implies f \leq g$
 $\langle \text{proof} \rangle$

lemma *le-funE*: $f \leq g \implies (f\ x \leq g\ x \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *le-funD*: $f \leq g \implies f\ x \leq g\ x$
 $\langle \text{proof} \rangle$

Handy introduction and elimination rules for \leq on unary and binary predicates

lemma *predicate1I*:

assumes *PQ*: $\bigwedge x. P\ x \implies Q\ x$

shows $P \leq Q$

$\langle \text{proof} \rangle$

lemma *predicate1D* [*Pure.dest*, *dest*]: $P \leq Q \implies P\ x \implies Q\ x$
 $\langle \text{proof} \rangle$

lemma *predicate2I* [*Pure.intro!*, *intro!*]:

assumes *PQ*: $\bigwedge x\ y. P\ x\ y \implies Q\ x\ y$

shows $P \leq Q$

$\langle \text{proof} \rangle$

lemma *predicate2D* [*Pure.dest*, *dest*]: $P \leq Q \implies P\ x\ y \implies Q\ x\ y$
 $\langle \text{proof} \rangle$

lemma *rev-predicate1D*: $P\ x \implies P \leq Q \implies Q\ x$
 $\langle \text{proof} \rangle$

lemma *rev-predicate2D*: $P\ x\ y \implies P \leq Q \implies Q\ x\ y$
 $\langle \text{proof} \rangle$

end

4 Lattices: Abstract lattices

```
theory Lattices
imports Orderings
begin
```

4.1 Lattices

notation

less-eq (infix \sqsubseteq 50) and
less (infix \sqsubset 50)

```
class lower-semilattice = order +
  fixes inf :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl  $\sqcap$  70)
  assumes inf-le1 [simp]:  $x \sqcap y \sqsubseteq x$ 
  and inf-le2 [simp]:  $x \sqcap y \sqsubseteq y$ 
  and inf-greatest:  $x \sqsubseteq y \Longrightarrow x \sqsubseteq z \Longrightarrow x \sqsubseteq y \sqcap z$ 
```

```
class upper-semilattice = order +
  fixes sup :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl  $\sqcup$  65)
  assumes sup-ge1 [simp]:  $x \sqsubseteq x \sqcup y$ 
  and sup-ge2 [simp]:  $y \sqsubseteq x \sqcup y$ 
  and sup-least:  $y \sqsubseteq x \Longrightarrow z \sqsubseteq x \Longrightarrow y \sqcup z \sqsubseteq x$ 
begin
```

Dual lattice

```
lemma dual-lattice:
  lower-semilattice (op  $\geq$ ) (op  $>$ ) sup
  <proof>
```

end

```
class lattice = lower-semilattice + upper-semilattice
```

4.1.1 Intro and elim rules

```
context lower-semilattice
begin
```

```
lemma le-infI1 [intro]:
  assumes  $a \sqsubseteq x$ 
  shows  $a \sqcap b \sqsubseteq x$ 
  <proof>
lemmas (in  $-$ ) [rule del] = le-infI1
```

```
lemma le-infI2 [intro]:
  assumes  $b \sqsubseteq x$ 
  shows  $a \sqcap b \sqsubseteq x$ 
  <proof>
lemmas (in  $-$ ) [rule del] = le-infI2
```

lemma *le-infI*[*intro!*]: $x \sqsubseteq a \implies x \sqsubseteq b \implies x \sqsubseteq a \sqcap b$
 $\langle \text{proof} \rangle$
lemmas (**in** $-$) [*rule del*] = *le-infI*

lemma *le-infE* [*elim!*]: $x \sqsubseteq a \sqcap b \implies (x \sqsubseteq a \implies x \sqsubseteq b \implies P) \implies P$
 $\langle \text{proof} \rangle$
lemmas (**in** $-$) [*rule del*] = *le-infE*

lemma *le-inf-iff* [*simp*]:
 $x \sqsubseteq y \sqcap z = (x \sqsubseteq y \wedge x \sqsubseteq z)$
 $\langle \text{proof} \rangle$

lemma *le-iff-inf*: $(x \sqsubseteq y) = (x \sqcap y = x)$
 $\langle \text{proof} \rangle$

lemma *mono-inf*:
fixes $f :: 'a \Rightarrow 'b :: \text{lower-semilattice}$
shows $\text{mono } f \implies f (A \sqcap B) \leq f A \sqcap f B$
 $\langle \text{proof} \rangle$

end

context *upper-semilattice*
begin

lemma *le-supI1*[*intro*]: $x \sqsubseteq a \implies x \sqsubseteq a \sqcup b$
 $\langle \text{proof} \rangle$
lemmas (**in** $-$) [*rule del*] = *le-supI1*

lemma *le-supI2*[*intro*]: $x \sqsubseteq b \implies x \sqsubseteq a \sqcup b$
 $\langle \text{proof} \rangle$
lemmas (**in** $-$) [*rule del*] = *le-supI2*

lemma *le-supI*[*intro!*]: $a \sqsubseteq x \implies b \sqsubseteq x \implies a \sqcup b \sqsubseteq x$
 $\langle \text{proof} \rangle$
lemmas (**in** $-$) [*rule del*] = *le-supI*

lemma *le-supE*[*elim!*]: $a \sqcup b \sqsubseteq x \implies (a \sqsubseteq x \implies b \sqsubseteq x \implies P) \implies P$
 $\langle \text{proof} \rangle$
lemmas (**in** $-$) [*rule del*] = *le-supE*

lemma *ge-sup-conv*[*simp*]:
 $x \sqcup y \sqsubseteq z = (x \sqsubseteq z \wedge y \sqsubseteq z)$
 $\langle \text{proof} \rangle$

lemma *le-iff-sup*: $(x \sqsubseteq y) = (x \sqcup y = y)$
 $\langle \text{proof} \rangle$

lemma *mono-sup*:
 fixes $f :: 'a \Rightarrow 'b::\text{upper-semilattice}$
 shows $\text{mono } f \implies f A \sqcup f B \leq f (A \sqcup B)$
 $\langle \text{proof} \rangle$

end

4.1.2 Equational laws

context *lower-semilattice*
begin

lemma *inf-commute*: $(x \sqcap y) = (y \sqcap x)$
 $\langle \text{proof} \rangle$

lemma *inf-assoc*: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
 $\langle \text{proof} \rangle$

lemma *inf-idem[simp]*: $x \sqcap x = x$
 $\langle \text{proof} \rangle$

lemma *inf-left-idem[simp]*: $x \sqcap (x \sqcap y) = x \sqcap y$
 $\langle \text{proof} \rangle$

lemma *inf-absorb1*: $x \sqsubseteq y \implies x \sqcap y = x$
 $\langle \text{proof} \rangle$

lemma *inf-absorb2*: $y \sqsubseteq x \implies x \sqcap y = y$
 $\langle \text{proof} \rangle$

lemma *inf-left-commute*: $x \sqcap (y \sqcap z) = y \sqcap (x \sqcap z)$
 $\langle \text{proof} \rangle$

lemmas *inf-ACI* = *inf-commute inf-assoc inf-left-commute inf-left-idem*

end

context *upper-semilattice*
begin

lemma *sup-commute*: $(x \sqcup y) = (y \sqcup x)$
 $\langle \text{proof} \rangle$

lemma *sup-assoc*: $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
 $\langle \text{proof} \rangle$

lemma *sup-idem[simp]*: $x \sqcup x = x$
 $\langle \text{proof} \rangle$

lemma *sup-left-idem[simp]*: $x \sqcup (x \sqcup y) = x \sqcup y$
 $\langle proof \rangle$

lemma *sup-absorb1*: $y \sqsubseteq x \implies x \sqcup y = x$
 $\langle proof \rangle$

lemma *sup-absorb2*: $x \sqsubseteq y \implies x \sqcup y = y$
 $\langle proof \rangle$

lemma *sup-left-commute*: $x \sqcup (y \sqcup z) = y \sqcup (x \sqcup z)$
 $\langle proof \rangle$

lemmas *sup-ACI = sup-commute sup-assoc sup-left-commute sup-left-idem*

end

context *lattice*
begin

lemma *inf-sup-absorb*: $x \sqcap (x \sqcup y) = x$
 $\langle proof \rangle$

lemma *sup-inf-absorb*: $x \sqcup (x \sqcap y) = x$
 $\langle proof \rangle$

lemmas *ACI = inf-ACI sup-ACI*

lemmas *inf-sup-ord = inf-le1 inf-le2 sup-ge1 sup-ge2*

Towards distributivity

lemma *distrib-sup-le*: $x \sqcup (y \sqcap z) \sqsubseteq (x \sqcup y) \sqcap (x \sqcup z)$
 $\langle proof \rangle$

lemma *distrib-inf-le*: $(x \sqcap y) \sqcup (x \sqcap z) \sqsubseteq x \sqcap (y \sqcup z)$
 $\langle proof \rangle$

If you have one of them, you have them all.

lemma *distrib-imp1*:

assumes *D*: $\forall x y z. x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$

shows $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

$\langle proof \rangle$

lemma *distrib-imp2*:

assumes *D*: $\forall x y z. x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

shows $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$

$\langle proof \rangle$

lemma *modular-le*: $x \sqsubseteq z \implies x \sqcup (y \sqcap z) \sqsubseteq (x \sqcup y) \sqcap z$
 $\langle \text{proof} \rangle$

end

4.2 Distributive lattices

class *distrib-lattice* = *lattice* +
assumes *sup-inf-distrib1*: $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

context *distrib-lattice*
begin

lemma *sup-inf-distrib2*:
 $(y \sqcap z) \sqcup x = (y \sqcup x) \sqcap (z \sqcup x)$
 $\langle \text{proof} \rangle$

lemma *inf-sup-distrib1*:
 $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$
 $\langle \text{proof} \rangle$

lemma *inf-sup-distrib2*:
 $(y \sqcup z) \sqcap x = (y \sqcap x) \sqcup (z \sqcap x)$
 $\langle \text{proof} \rangle$

lemmas *distrib* =
sup-inf-distrib1 sup-inf-distrib2 inf-sup-distrib1 inf-sup-distrib2

end

4.3 Uniqueness of inf and sup

lemma (**in** *lower-semilattice*) *inf-unique*:
fixes *f* (**infixl** \triangle 70)
assumes *le1*: $\bigwedge x y. x \triangle y \leq x$ **and** *le2*: $\bigwedge x y. x \triangle y \leq y$
and *greatest*: $\bigwedge x y z. x \leq y \implies x \leq z \implies x \leq y \triangle z$
shows $x \sqcap y = x \triangle y$
 $\langle \text{proof} \rangle$

lemma (**in** *upper-semilattice*) *sup-unique*:
fixes *f* (**infixl** ∇ 70)
assumes *ge1* [*simp*]: $\bigwedge x y. x \leq x \nabla y$ **and** *ge2*: $\bigwedge x y. y \leq x \nabla y$
and *least*: $\bigwedge x y z. y \leq x \implies z \leq x \implies y \nabla z \leq x$
shows $x \sqcup y = x \nabla y$
 $\langle \text{proof} \rangle$

4.4 *min/max* on linear orders as special case of *op* \sqcap /*op* \sqcup

lemma (**in** *linorder*) *distrib-lattice-min-max*:
distrib-lattice (*op* \leq) (*op* $<$) *min max*

<proof>

interpretation *min-max*: *distrib-lattice* *op* $\leq :: 'a::linorder \Rightarrow 'a \Rightarrow bool$ *op* $<$
min max
<proof>

lemma *inf-min*: *inf* = (*min* :: '*a*::{*lower-semilattice*, *linorder*} $\Rightarrow 'a \Rightarrow 'a$)
<proof>

lemma *sup-max*: *sup* = (*max* :: '*a*::{*upper-semilattice*, *linorder*} $\Rightarrow 'a \Rightarrow 'a$)
<proof>

lemmas *le-maxI1* = *min-max.sup-ge1*
lemmas *le-maxI2* = *min-max.sup-ge2*

lemmas *max-ac* = *min-max.sup-assoc min-max.sup-commute*
mk-left-commute [of max, OF min-max.sup-assoc min-max.sup-commute]

lemmas *min-ac* = *min-max.inf-assoc min-max.inf-commute*
mk-left-commute [of min, OF min-max.inf-assoc min-max.inf-commute]

Now we have inherited antisymmetry as an intro-rule on all linear orders.
 This is a problem because it applies to *bool*, which is undesirable.

lemmas [*rule del*] = *min-max.le-infI min-max.le-supI*
min-max.le-supE min-max.le-infE min-max.le-supI1 min-max.le-supI2
min-max.le-infI1 min-max.le-infI2

4.5 Bool as lattice

instantiation *bool* :: *distrib-lattice*
begin

definition
inf-bool-eq: $P \sqcap Q \longleftrightarrow P \wedge Q$

definition
sup-bool-eq: $P \sqcup Q \longleftrightarrow P \vee Q$

instance
<proof>

end

4.6 Fun as lattice

instantiation *fun* :: (*type*, *lattice*) *lattice*
begin

definition

inf-fun-eq [code del]: $f \sqcap g = (\lambda x. f\ x \sqcap g\ x)$

definition

sup-fun-eq [code del]: $f \sqcup g = (\lambda x. f\ x \sqcup g\ x)$

instance

$\langle proof \rangle$

end

instance *fun* :: (*type*, *distrib-lattice*) *distrib-lattice*

$\langle proof \rangle$

redundant bindings

lemmas *inf-aci* = *inf-ACI*

lemmas *sup-aci* = *sup-ACI*

no-notation

less-eq (**infix** \sqsubseteq 50) **and**

less (**infix** \sqsubset 50) **and**

inf (**infixl** \sqcap 70) **and**

sup (**infixl** \sqcup 65)

end

5 Set: Set theory for higher-order logic

theory *Set*

imports *Lattices*

begin

A set in HOL is simply a predicate.

5.1 Basic syntax

global

types *'a set* = *'a => bool*

consts

Collect :: (*'a => bool*) => *'a set* — comprehension

op : :: *'a => 'a set => bool* — membership

insert :: *'a => 'a set => 'a set*

Ball :: *'a set => ('a => bool) => bool* — bounded universal quantifiers

Bex :: *'a set => ('a => bool) => bool* — bounded existential quantifiers

Bex1 :: *'a set => ('a => bool) => bool* — bounded unique existential
 quantifiers
Pow :: *'a set => 'a set set* — powerset
image :: *('a => 'b) => 'a set => 'b set* (**infixr** ‘90)

local**notation**

op : (*op* :) **and**
op : ((-/ : -) [50, 51] 50)

abbreviation

not-mem *x A* == ~ (*x* : *A*) — non-membership

notation

not-mem (*op* ~:) **and**
not-mem ((-/ ~: -) [50, 51] 50)

notation (*xsymbols*)

op : (*op* ∈) **and**
op : ((-/ ∈ -) [50, 51] 50) **and**
not-mem (*op* ∉) **and**
not-mem ((-/ ∉ -) [50, 51] 50)

notation (*HTML output*)

op : (*op* ∈) **and**
op : ((-/ ∈ -) [50, 51] 50) **and**
not-mem (*op* ∉) **and**
not-mem ((-/ ∉ -) [50, 51] 50)

syntax

@*Coll* :: *pttrn => bool => 'a set* ((1{-/ -}))

translations

{*x. P*} == *Collect* (%*x. P*)

definition *empty* :: *'a set* ({}) **where**

empty ≡ {*x. False*}

definition *UNIV* :: *'a set* **where**

UNIV ≡ {*x. True*}

syntax

@*Finset* :: *args => 'a set* ({(-)})

translations

{*x, xs*} == *insert* *x* {*xs*}
 {*x*} == *insert* *x* {}

definition $Int :: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$ (**infixl** Int 70) **where**
 $A \text{ Int } B \equiv \{x. x \in A \wedge x \in B\}$

definition $Un :: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$ (**infixl** Un 65) **where**
 $A \text{ Un } B \equiv \{x. x \in A \vee x \in B\}$

notation ($xsymbols$)
 Int (**infixl** \cap 70) **and**
 Un (**infixl** \cup 65)

notation ($HTML$ output)
 Int (**infixl** \cap 70) **and**
 Un (**infixl** \cup 65)

syntax
 $-Ball \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists ALL \text{ :-./ -}) [0, 0, 10] 10)$
 $-Bex \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists EX \text{ :-./ -}) [0, 0, 10] 10)$
 $-Bex1 \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists EX! \text{ :-./ -}) [0, 0, 10] 10)$
 $-Bleast \quad :: id \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow 'a \quad ((\exists LEAST \text{ :-./ -}) [0, 0, 10] 10)$

syntax (HOL)
 $-Ball \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists! \text{ :-./ -}) [0, 0, 10] 10)$
 $-Bex \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists? \text{ :-./ -}) [0, 0, 10] 10)$
 $-Bex1 \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists?! \text{ :-./ -}) [0, 0, 10] 10)$

syntax ($xsymbols$)
 $-Ball \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists \forall \text{ -}\in\text{./ -}) [0, 0, 10] 10)$
 $-Bex \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists \exists \text{ -}\in\text{./ -}) [0, 0, 10] 10)$
 $-Bex1 \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists \exists! \text{ -}\in\text{./ -}) [0, 0, 10] 10)$
 $-Bleast \quad :: id \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow 'a \quad ((\exists LEAST \text{ -}\in\text{./ -}) [0, 0, 10] 10)$

syntax ($HTML$ output)
 $-Ball \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists \forall \text{ -}\in\text{./ -}) [0, 0, 10] 10)$
 $-Bex \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists \exists \text{ -}\in\text{./ -}) [0, 0, 10] 10)$
 $-Bex1 \quad :: pttrn \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow bool \quad ((\exists \exists! \text{ -}\in\text{./ -}) [0, 0, 10] 10)$

translations
 $ALL x:A. P == Ball A (\%x. P)$
 $EX x:A. P == Bex A (\%x. P)$
 $EX! x:A. P == Bex1 A (\%x. P)$
 $LEAST x:A. P \Rightarrow LEAST x. x:A \ \& \ P$

definition $INTER :: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow 'b \text{ set}$ **where**
 $INTER A B \equiv \{y. \forall x \in A. y \in B x\}$

definition $UNION :: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow 'b \text{ set}$ **where**
 $UNION A B \equiv \{y. \exists x \in A. y \in B x\}$

definition $Inter :: 'a \text{ set set} \Rightarrow 'a \text{ set}$ **where**

$Inter\ S \equiv INTER\ S\ (\lambda x. x)$

definition $Union :: 'a \text{ set set} \Rightarrow 'a \text{ set}$ **where**

$Union\ S \equiv UNION\ S\ (\lambda x. x)$

notation ($xsymbols$)

$Inter\ (\bigcap - [90] 90)$ **and**

$Union\ (\bigcup - [90] 90)$

5.2 Additional concrete syntax

syntax

$@SetCompr :: 'a \Rightarrow idts \Rightarrow bool \Rightarrow 'a \text{ set} \quad ((1\{-\ |/-/ -\}))$
 $@Collect :: idt \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow 'a \text{ set} \quad ((1\{-\ :/-/ -\}))$
 $@INTER1 :: pttms \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3INT\ -/-/ -) [0, 10] 10)$
 $@UNION1 :: pttms \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3UN\ -/-/ -) [0, 10] 10)$
 $@INTER :: pttm \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3INT\ -:-/-/ -) [0, 10] 10)$
 $@UNION :: pttm \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3UN\ -:-/-/ -) [0, 10] 10)$

syntax ($xsymbols$)

$@Collect :: idt \Rightarrow 'a \text{ set} \Rightarrow bool \Rightarrow 'a \text{ set} \quad ((1\{-\ \in/-/ -\}))$
 $@INTER1 :: pttms \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcap\ -/-/ -) [0, 10] 10)$
 $@UNION1 :: pttms \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcup\ -/-/ -) [0, 10] 10)$
 $@INTER :: pttm \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcap\ -\in/-/ -) [0, 10] 10)$
 $@UNION :: pttm \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcup\ -\in/-/ -) [0, 10] 10)$

syntax ($latex$ **output**)

$@INTER1 :: pttms \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcap\ (00_-)/ -) [0, 10] 10)$
 $@UNION1 :: pttms \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcup\ (00_-)/ -) [0, 10] 10)$
 $@INTER :: pttm \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcap\ (00_{-\in})/ -) [0, 10] 10)$
 $@UNION :: pttm \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((3\bigcup\ (00_{-\in})/ -) [0, 10] 10)$

translations

$\{x:A. P\} \Rightarrow \{x. x:A \ \& \ P\}$
 $INT\ x\ y. B == INT\ x. INT\ y. B$
 $INT\ x. B == CONST\ INTER\ CONST\ UNIV\ (\%x. B)$
 $INT\ x. B == INT\ x:CONST\ UNIV. B$
 $INT\ x:A. B == CONST\ INTER\ A\ (\%x. B)$
 $UN\ x\ y. B == UN\ x. UN\ y. B$
 $UN\ x. B == CONST\ UNION\ CONST\ UNIV\ (\%x. B)$
 $UN\ x. B == UN\ x:CONST\ UNIV. B$
 $UN\ x:A. B == CONST\ UNION\ A\ (\%x. B)$

Note the difference between ordinary xsymbol syntax of indexed unions and intersections (e.g. $\bigcup_{a_1 \in A_1} B$) and their L^AT_EX rendition: $\bigcup_{a_1 \in A_1} B$. The former does not make the index expression a subscript of the union/intersection

symbol because this leads to problems with nested subscripts in Proof General.

abbreviation

$subset :: 'a\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
 $subset \equiv less$

abbreviation

$subset-eq :: 'a\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
 $subset-eq \equiv less-eq$

notation (output)

$subset\ (op\ <)$ **and**
 $subset\ ((-/ < -)\ [50, 51]\ 50)$ **and**
 $subset-eq\ (op\ <=)$ **and**
 $subset-eq\ ((-/ <= -)\ [50, 51]\ 50)$

notation (xsymbols)

$subset\ (op\ \subset)$ **and**
 $subset\ ((-/ \subset -)\ [50, 51]\ 50)$ **and**
 $subset-eq\ (op\ \subseteq)$ **and**
 $subset-eq\ ((-/ \subseteq -)\ [50, 51]\ 50)$

notation (HTML output)

$subset\ (op\ \subset)$ **and**
 $subset\ ((-/ \subset -)\ [50, 51]\ 50)$ **and**
 $subset-eq\ (op\ \subseteq)$ **and**
 $subset-eq\ ((-/ \subseteq -)\ [50, 51]\ 50)$

abbreviation (input)

$supset :: 'a\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
 $supset \equiv greater$

abbreviation (input)

$supset-eq :: 'a\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
 $supset-eq \equiv greater-eq$

notation (xsymbols)

$supset\ (op\ \supset)$ **and**
 $supset\ ((-/ \supset -)\ [50, 51]\ 50)$ **and**
 $supset-eq\ (op\ \supseteq)$ **and**
 $supset-eq\ ((-/ \supseteq -)\ [50, 51]\ 50)$

abbreviation

$range :: ('a \Rightarrow 'b) \Rightarrow 'b\ set$ **where** — of function
 $range\ f == f\ ` UNIV$

5.2.1 Bounded quantifiers

syntax (output)

```

-setlessAll :: [idt, 'a, bool] => bool ((3ALL -<-./ -) [0, 0, 10] 10)
-setlessEx  :: [idt, 'a, bool] => bool ((3EX -<-./ -) [0, 0, 10] 10)
-setleAll   :: [idt, 'a, bool] => bool ((3ALL -<=.-./ -) [0, 0, 10] 10)
-setleEx    :: [idt, 'a, bool] => bool ((3EX -<=.-./ -) [0, 0, 10] 10)
-setleEx1   :: [idt, 'a, bool] => bool ((3EX! -<=.-./ -) [0, 0, 10] 10)

```

syntax (*xsymbols*)

```

-setlessAll :: [idt, 'a, bool] => bool ((3∀ -⊂.-./ -) [0, 0, 10] 10)
-setlessEx  :: [idt, 'a, bool] => bool ((3∃ -⊂.-./ -) [0, 0, 10] 10)
-setleAll   :: [idt, 'a, bool] => bool ((3∀ -⊆.-./ -) [0, 0, 10] 10)
-setleEx    :: [idt, 'a, bool] => bool ((3∃ -⊆.-./ -) [0, 0, 10] 10)
-setleEx1   :: [idt, 'a, bool] => bool ((3∃! -⊆.-./ -) [0, 0, 10] 10)

```

syntax (*HOL output*)

```

-setlessAll :: [idt, 'a, bool] => bool ((3! -<-./ -) [0, 0, 10] 10)
-setlessEx  :: [idt, 'a, bool] => bool ((3? -<-./ -) [0, 0, 10] 10)
-setleAll   :: [idt, 'a, bool] => bool ((3! -<=.-./ -) [0, 0, 10] 10)
-setleEx    :: [idt, 'a, bool] => bool ((3? -<=.-./ -) [0, 0, 10] 10)
-setleEx1   :: [idt, 'a, bool] => bool ((3?! -<=.-./ -) [0, 0, 10] 10)

```

syntax (*HTML output*)

```

-setlessAll :: [idt, 'a, bool] => bool ((3∀ -⊂.-./ -) [0, 0, 10] 10)
-setlessEx  :: [idt, 'a, bool] => bool ((3∃ -⊂.-./ -) [0, 0, 10] 10)
-setleAll   :: [idt, 'a, bool] => bool ((3∀ -⊆.-./ -) [0, 0, 10] 10)
-setleEx    :: [idt, 'a, bool] => bool ((3∃ -⊆.-./ -) [0, 0, 10] 10)
-setleEx1   :: [idt, 'a, bool] => bool ((3∃! -⊆.-./ -) [0, 0, 10] 10)

```

translations

```

∀ A ⊂ B. P  =>  ALL A. A ⊂ B --> P
∃ A ⊂ B. P  =>  EX A. A ⊂ B & P
∀ A ⊆ B. P  =>  ALL A. A ⊆ B --> P
∃ A ⊆ B. P  =>  EX A. A ⊆ B & P
∃! A ⊆ B. P =>  EX! A. A ⊆ B & P

```

⟨ML⟩

Translate between $\{e \mid x1...xn. P\}$ and $\{u. EX\ x1...xn. u = e \ \& \ P\}$; $\{y. EX\ x1...xn. y = e \ \& \ P\}$ is only translated if $[0..n] \text{ subset } \text{bvs}(e)$.

⟨ML⟩

5.3 Rules and definitions

Isomorphisms between predicates and sets.

defs

```

mem-def [code]: x : S == S x
Collect-def [code]: Collect P == P

```

defs

Ball-def: $Ball\ A\ P \quad ==\ ALL\ x.\ x:A \dashv\vdash P(x)$
Bex-def: $Bex\ A\ P \quad ==\ EX\ x.\ x:A \ \&\ P(x)$
Bex1-def: $Bex1\ A\ P \quad ==\ EX!\ x.\ x:A \ \&\ P(x)$

instantiation *fun* :: (*type*, *minus*) *minus*
begin

definition

fun-diff-def: $A - B = (\%x.\ A\ x - B\ x)$

instance $\langle proof \rangle$

end

instantiation *bool* :: *minus*
begin

definition

bool-diff-def: $A - B = (A \ \&\ \sim B)$

instance $\langle proof \rangle$

end

instantiation *fun* :: (*type*, *uminus*) *uminus*
begin

definition

fun-Compl-def: $-\ A = (\%x.\ -\ A\ x)$

instance $\langle proof \rangle$

end

instantiation *bool* :: *uminus*
begin

definition

bool-Compl-def: $-\ A = (\sim\ A)$

instance $\langle proof \rangle$

end

defs

Pow-def: $Pow\ A \quad ==\ \{B.\ B \leq A\}$
insert-def: $insert\ a\ B \quad ==\ \{x.\ x=a\} \cup B$
image-def: $f^*A \quad ==\ \{y.\ EX\ x:A.\ y = f(x)\}$

5.4 Lemmas and proof tool setup

5.4.1 Relating predicates and sets

lemma *mem-Collect-eq* [*iff*]: $(a : \{x. P(x)\}) = P(a)$
 $\langle proof \rangle$

lemma *Collect-mem-eq* [*simp*]: $\{x. x:A\} = A$
 $\langle proof \rangle$

lemma *CollectI*: $P(a) ==> a : \{x. P(x)\}$
 $\langle proof \rangle$

lemma *CollectD*: $a : \{x. P(x)\} ==> P(a)$
 $\langle proof \rangle$

lemma *Collect-cong*: $(!!x. P x = Q x) ==> \{x. P(x)\} = \{x. Q(x)\}$
 $\langle proof \rangle$

lemmas *CollectE* = *CollectD* [*elim-format*]

5.4.2 Bounded quantifiers

lemma *ballI* [*intro!*]: $(!!x. x:A ==> P x) ==> ALL x:A. P x$
 $\langle proof \rangle$

lemmas *strip* = *impI* *allI* *ballI*

lemma *bspec* [*dest?*]: $ALL x:A. P x ==> x:A ==> P x$
 $\langle proof \rangle$

lemma *ballE* [*elim*]: $ALL x:A. P x ==> (P x ==> Q) ==> (x \sim: A ==> Q) ==> Q$
 $\langle proof \rangle$

$\langle ML \rangle$

This tactic takes assumptions $\forall x \in A. P x$ and $a \in A$; creates assumption $P a$.

$\langle ML \rangle$

Gives better instantiation for bound:

$\langle ML \rangle$

lemma *bexI* [*intro*]: $P x ==> x:A ==> EX x:A. P x$
 — Normally the best argument order: $P x$ constrains the choice of $x \in A$.
 $\langle proof \rangle$

lemma *rev-bexI* [*intro?*]: $x:A ==> P x ==> EX x:A. P x$
 — The best argument order when there is only one $x \in A$.

$\langle proof \rangle$

lemma *bexCI*: $(ALL\ x:A. \sim P\ x ==> P\ a) ==> a:A ==> EX\ x:A. P\ x$
 $\langle proof \rangle$

lemma *bexE* [*elim!*]: $EX\ x:A. P\ x ==> (!x. x:A ==> P\ x ==> Q) ==> Q$
 $\langle proof \rangle$

lemma *ball-triv* [*simp*]: $(ALL\ x:A. P) = ((EX\ x. x:A) --> P)$
 — Trivial rewrite rule.
 $\langle proof \rangle$

lemma *bex-triv* [*simp*]: $(EX\ x:A. P) = ((EX\ x. x:A) \& P)$
 — Dual form for existentials.
 $\langle proof \rangle$

lemma *bex-triv-one-point1* [*simp*]: $(EX\ x:A. x = a) = (a:A)$
 $\langle proof \rangle$

lemma *bex-triv-one-point2* [*simp*]: $(EX\ x:A. a = x) = (a:A)$
 $\langle proof \rangle$

lemma *bex-one-point1* [*simp*]: $(EX\ x:A. x = a \& P\ x) = (a:A \& P\ a)$
 $\langle proof \rangle$

lemma *bex-one-point2* [*simp*]: $(EX\ x:A. a = x \& P\ x) = (a:A \& P\ a)$
 $\langle proof \rangle$

lemma *ball-one-point1* [*simp*]: $(ALL\ x:A. x = a --> P\ x) = (a:A --> P\ a)$
 $\langle proof \rangle$

lemma *ball-one-point2* [*simp*]: $(ALL\ x:A. a = x --> P\ x) = (a:A --> P\ a)$
 $\langle proof \rangle$

$\langle ML \rangle$

5.4.3 Congruence rules

lemma *ball-cong*:
 $A = B ==> (!x. x:B ==> P\ x = Q\ x) ==>$
 $(ALL\ x:A. P\ x) = (ALL\ x:B. Q\ x)$
 $\langle proof \rangle$

lemma *strong-ball-cong* [*cong*]:
 $A = B ==> (!x. x:B ==> P\ x = Q\ x) ==>$
 $(ALL\ x:A. P\ x) = (ALL\ x:B. Q\ x)$
 $\langle proof \rangle$

lemma *bex-cong*:

$A = B ==> (!!x. x:B ==> P\ x = Q\ x) ==>$
 $(EX\ x:A. P\ x) = (EX\ x:B. Q\ x)$
 $\langle proof \rangle$

lemma *strong-bex-cong* [*cong*]:
 $A = B ==> (!!x. x:B ==simp==> P\ x = Q\ x) ==>$
 $(EX\ x:A. P\ x) = (EX\ x:B. Q\ x)$
 $\langle proof \rangle$

5.4.4 Subsets

lemma *subsetI* [*atp,intro!*]: $(!!x. x:A ==> x:B) ==> A \subseteq B$
 $\langle proof \rangle$

Map the type '*a set ==> anything*' to just '*a*'; for overloading constants whose first argument has type '*a set*'.

lemma *subsetD* [*elim, intro?*]: $A \subseteq B ==> c \in A ==> c \in B$
 — Rule in Modus Ponens style.
 $\langle proof \rangle$

lemma *rev-subsetD* [*intro?*]: $c \in A ==> A \subseteq B ==> c \in B$
 — The same, with reversed premises for use with *erule* – cf *rev-mp*.
 $\langle proof \rangle$

Converts $A \subseteq B$ to $x \in A \implies x \in B$.

$\langle ML \rangle$

lemma *subsetCE* [*elim*]: $A \subseteq B ==> (c \notin A ==> P) ==> (c \in B ==> P)$
 $==> P$
 — Classical elimination rule.
 $\langle proof \rangle$

lemma *subset-eq*: $A \subseteq B = (\forall x \in A. x \in B)$ $\langle proof \rangle$

Takes assumptions $A \subseteq B$; $c \in A$ and creates the assumption $c \in B$.

$\langle ML \rangle$

lemma *contra-subsetD*: $A \subseteq B ==> c \notin B ==> c \notin A$
 $\langle proof \rangle$

lemma *subset-refl* [*simp,atp*]: $A \subseteq A$
 $\langle proof \rangle$

lemma *subset-trans*: $A \subseteq B ==> B \subseteq C ==> A \subseteq C$
 $\langle proof \rangle$

5.4.5 Equality

lemma *set-ext*: **assumes** *prem*: $(!!x. (x:A) = (x:B))$ **shows** $A = B$
 $\langle proof \rangle$

lemma *expand-set-eq*: $(A = B) = (ALL\ x. (x:A) = (x:B))$
 $\langle proof \rangle$

lemma *subset-antisym* [*intro!*]: $A \subseteq B ==> B \subseteq A ==> A = B$
 — Anti-symmetry of the subset relation.
 $\langle proof \rangle$

Equality rules from ZF set theory – are they appropriate here?

lemma *equalityD1*: $A = B ==> A \subseteq B$
 $\langle proof \rangle$

lemma *equalityD2*: $A = B ==> B \subseteq A$
 $\langle proof \rangle$

Be careful when adding this to the claset as *subset-empty* is in the simpset:
 $A = \{\}$ goes to $\{\} \subseteq A$ and $A \subseteq \{\}$ and then back to $A = \{\}$!

lemma *equalityE*: $A = B ==> (A \subseteq B ==> B \subseteq A ==> P) ==> P$
 $\langle proof \rangle$

lemma *equalityCE* [*elim*]:
 $A = B ==> (c \in A ==> c \in B ==> P) ==> (c \notin A ==> c \notin B ==> P)$
 $==> P$
 $\langle proof \rangle$

lemma *eqset-imp-iff*: $A = B ==> (x : A) = (x : B)$
 $\langle proof \rangle$

lemma *eqelem-imp-iff*: $x = y ==> (x : A) = (y : A)$
 $\langle proof \rangle$

5.4.6 The universal set – UNIV

lemma *UNIV-I* [*simp*]: $x : UNIV$
 $\langle proof \rangle$

declare *UNIV-I* [*intro*] — unsafe makes it less likely to cause problems

lemma *UNIV-witness* [*intro?*]: $EX\ x. x : UNIV$
 $\langle proof \rangle$

lemma *subset-UNIV* [*simp*]: $A \subseteq UNIV$
 $\langle proof \rangle$

Eta-contracting these two rules (to remove P) causes them to be ignored because of their interaction with congruence rules.

lemma *ball-UNIV* [simp]: $Ball\ UNIV\ P = All\ P$
 $\langle proof \rangle$

lemma *bex-UNIV* [simp]: $Bex\ UNIV\ P = Ex\ P$
 $\langle proof \rangle$

lemma *UNIV-eq-I*: $(\bigwedge x. x \in A) \implies UNIV = A$
 $\langle proof \rangle$

5.4.7 The empty set

lemma *empty-iff* [simp]: $(c : \{\}) = False$
 $\langle proof \rangle$

lemma *emptyE* [elim!]: $a : \{\} \implies P$
 $\langle proof \rangle$

lemma *empty-subsetI* [iff]: $\{\} \subseteq A$
 — One effect is to delete the ASSUMPTION $\{\} \subseteq A$
 $\langle proof \rangle$

lemma *equals0I*: $(!!y. y \in A \implies False) \implies A = \{\}$
 $\langle proof \rangle$

lemma *equals0D*: $A = \{\} \implies a \notin A$
 — Use for reasoning about disjointness: $A \cap B = \{\}$
 $\langle proof \rangle$

lemma *ball-empty* [simp]: $Ball\ \{\} P = True$
 $\langle proof \rangle$

lemma *bex-empty* [simp]: $Bex\ \{\} P = False$
 $\langle proof \rangle$

lemma *UNIV-not-empty* [iff]: $UNIV \sim = \{\}$
 $\langle proof \rangle$

5.4.8 The Powerset operator – Pow

lemma *Pow-iff* [iff]: $(A \in Pow\ B) = (A \subseteq B)$
 $\langle proof \rangle$

lemma *PowI*: $A \subseteq B \implies A \in Pow\ B$
 $\langle proof \rangle$

lemma *PowD*: $A \in Pow\ B \implies A \subseteq B$
 $\langle proof \rangle$

lemma *Pow-bottom*: $\{\} \in \text{Pow } B$
 $\langle \text{proof} \rangle$

lemma *Pow-top*: $A \in \text{Pow } A$
 $\langle \text{proof} \rangle$

5.4.9 Set complement

lemma *Compl-iff* [*simp*]: $(c \in -A) = (c \notin A)$
 $\langle \text{proof} \rangle$

lemma *ComplI* [*intro!*]: $(c \in A ==> \text{False}) ==> c \in -A$
 $\langle \text{proof} \rangle$

This form, with negated conclusion, works well with the Classical prover. Negated assumptions behave like formulae on the right side of the notional turnstile ...

lemma *ComplD* [*dest!*]: $c : -A ==> c \sim A$
 $\langle \text{proof} \rangle$

lemmas *ComplE* = *ComplD* [*elim-format*]

lemma *Compl-eq*: $-A = \{x. \sim x : A\}$ $\langle \text{proof} \rangle$

5.4.10 Binary union – Un

lemma *Un-iff* [*simp*]: $(c : A \text{ Un } B) = (c:A \mid c:B)$
 $\langle \text{proof} \rangle$

lemma *UnI1* [*elim?*]: $c:A ==> c : A \text{ Un } B$
 $\langle \text{proof} \rangle$

lemma *UnI2* [*elim?*]: $c:B ==> c : A \text{ Un } B$
 $\langle \text{proof} \rangle$

Classical introduction rule: no commitment to A vs B .

lemma *UnCI* [*intro!*]: $(c \sim B ==> c:A) ==> c : A \text{ Un } B$
 $\langle \text{proof} \rangle$

lemma *UnE* [*elim!*]: $c : A \text{ Un } B ==> (c:A ==> P) ==> (c:B ==> P) ==> P$
 $\langle \text{proof} \rangle$

5.4.11 Binary intersection – Int

lemma *Int-iff* [*simp*]: $(c : A \text{ Int } B) = (c:A \ \& \ c:B)$
 $\langle \text{proof} \rangle$

lemma *IntI* [*intro!*]: $c:A \implies c:B \implies c : A \text{ Int } B$
 $\langle \text{proof} \rangle$

lemma *IntD1*: $c : A \text{ Int } B \implies c:A$
 $\langle \text{proof} \rangle$

lemma *IntD2*: $c : A \text{ Int } B \implies c:B$
 $\langle \text{proof} \rangle$

lemma *IntE* [*elim!*]: $c : A \text{ Int } B \implies (c:A \implies c:B \implies P) \implies P$
 $\langle \text{proof} \rangle$

5.4.12 Set difference

lemma *Diff-iff* [*simp*]: $(c : A - B) = (c:A \ \& \ c\sim:B)$
 $\langle \text{proof} \rangle$

lemma *DiffI* [*intro!*]: $c : A \implies c \sim : B \implies c : A - B$
 $\langle \text{proof} \rangle$

lemma *DiffD1*: $c : A - B \implies c : A$
 $\langle \text{proof} \rangle$

lemma *DiffD2*: $c : A - B \implies c : B \implies P$
 $\langle \text{proof} \rangle$

lemma *DiffE* [*elim!*]: $c : A - B \implies (c:A \implies c\sim:B \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *set-diff-eq*: $A - B = \{x. x : A \ \& \ \sim x : B\}$ $\langle \text{proof} \rangle$

lemma *Compl-eq-Diff-UNIV*: $-A = (UNIV - A)$
 $\langle \text{proof} \rangle$

5.4.13 Augmenting a set – insert

lemma *insert-iff* [*simp*]: $(a : \text{insert } b \ A) = (a = b \mid a:A)$
 $\langle \text{proof} \rangle$

lemma *insertI1*: $a : \text{insert } a \ B$
 $\langle \text{proof} \rangle$

lemma *insertI2*: $a : B \implies a : \text{insert } b \ B$
 $\langle \text{proof} \rangle$

lemma *insertE* [*elim!*]: $a : \text{insert } b \ A \implies (a = b \implies P) \implies (a:A \implies P)$
 $\implies P$
 $\langle \text{proof} \rangle$

lemma *insertCI* [intro!]: $(a \sim B \implies a = b) \implies a : \text{insert } b \ B$
 — Classical introduction rule.
 $\langle \text{proof} \rangle$

lemma *subset-insert-iff*: $(A \subseteq \text{insert } x \ B) = (\text{if } x:A \text{ then } A - \{x\} \subseteq B \text{ else } A \subseteq B)$
 $\langle \text{proof} \rangle$

lemma *set-insert*:
 assumes $x \in A$
 obtains B where $A = \text{insert } x \ B$ and $x \notin B$
 $\langle \text{proof} \rangle$

lemma *insert-ident*: $x \sim A \implies x \sim B \implies (\text{insert } x \ A = \text{insert } x \ B) = (A = B)$
 $\langle \text{proof} \rangle$

5.4.14 Singletons, using insert

lemma *singletonI* [intro!,noatp]: $a : \{a\}$
 — Redundant? But unlike *insertCI*, it proves the subgoal immediately!
 $\langle \text{proof} \rangle$

lemma *singletonD* [dest!,noatp]: $b : \{a\} \implies b = a$
 $\langle \text{proof} \rangle$

lemmas *singletonE* = *singletonD* [elim-format]

lemma *singleton-iff*: $(b : \{a\}) = (b = a)$
 $\langle \text{proof} \rangle$

lemma *singleton-inject* [dest!]: $\{a\} = \{b\} \implies a = b$
 $\langle \text{proof} \rangle$

lemma *singleton-insert-inj-eq* [iff,noatp]:
 $(\{b\} = \text{insert } a \ A) = (a = b \ \& \ A \subseteq \{b\})$
 $\langle \text{proof} \rangle$

lemma *singleton-insert-inj-eq'* [iff,noatp]:
 $(\text{insert } a \ A = \{b\}) = (a = b \ \& \ A \subseteq \{b\})$
 $\langle \text{proof} \rangle$

lemma *subset-singletonD*: $A \subseteq \{x\} \implies A = \{\} \mid A = \{x\}$
 $\langle \text{proof} \rangle$

lemma *singleton-conv* [simp]: $\{x. x = a\} = \{a\}$
 $\langle \text{proof} \rangle$

lemma *singleton-conv2* [simp]: $\{x. a = x\} = \{a\}$

$\langle proof \rangle$

lemma *diff-single-insert*: $A - \{x\} \subseteq B \implies x \in A \implies A \subseteq insert\ x\ B$
 $\langle proof \rangle$

lemma *doubleton-eq-iff*: $(\{a, b\} = \{c, d\}) = (a=c \ \& \ b=d \mid a=d \ \& \ b=c)$
 $\langle proof \rangle$

5.4.15 Unions of families

$UN\ x:A. B\ x$ is $\bigcup B \text{ ‘ } A$.

declare *UNION-def* [noatp]

lemma *UN-iff* [simp]: $(b: (UN\ x:A. B\ x)) = (EX\ x:A. b: B\ x)$
 $\langle proof \rangle$

lemma *UN-I* [intro]: $a:A \implies b: B\ a \implies b: (UN\ x:A. B\ x)$
 — The order of the premises presupposes that A is rigid; b may be flexible.
 $\langle proof \rangle$

lemma *UN-E* [elim!]: $b: (UN\ x:A. B\ x) \implies (!x. x:A \implies b: B\ x \implies R)$
 $\implies R$
 $\langle proof \rangle$

lemma *UN-cong* [cong]:
 $A = B \implies (!x. x:B \implies C\ x = D\ x) \implies (UN\ x:A. C\ x) = (UN\ x:B. D\ x)$
 $\langle proof \rangle$

lemma *strong-UN-cong*:
 $A = B \implies (!x. x:B \implies C\ x = D\ x) \implies (UN\ x:A. C\ x) = (UN\ x:B. D\ x)$
 $\langle proof \rangle$

5.4.16 Intersections of families

$INT\ x:A. B\ x$ is $\bigcap B \text{ ‘ } A$.

lemma *INT-iff* [simp]: $(b: (INT\ x:A. B\ x)) = (ALL\ x:A. b: B\ x)$
 $\langle proof \rangle$

lemma *INT-I* [intro!]: $(!x. x:A \implies b: B\ x) \implies b: (INT\ x:A. B\ x)$
 $\langle proof \rangle$

lemma *INT-D* [elim]: $b: (INT\ x:A. B\ x) \implies a:A \implies b: B\ a$
 $\langle proof \rangle$

lemma *INT-E* [elim]: $b: (INT\ x:A. B\ x) \implies (b: B\ a \implies R) \implies (a \sim A \implies R) \implies R$

— “Classical” elimination – by the Excluded Middle on $a \in A$.
 $\langle proof \rangle$

lemma *INT-cong* [*cong*]:

$A = B \implies (!x. x:B \implies C\ x = D\ x) \implies (INT\ x:A. C\ x) = (INT\ x:B. D\ x)$
 $\langle proof \rangle$

5.4.17 Union

lemma *Union-iff* [*simp, noatp*]: $(A : Union\ C) = (EX\ X:C. A:X)$
 $\langle proof \rangle$

lemma *UnionI* [*intro*]: $X:C \implies A:X \implies A : Union\ C$

— The order of the premises presupposes that C is rigid; A may be flexible.
 $\langle proof \rangle$

lemma *UnionE* [*elim!*]: $A : Union\ C \implies (!X. A:X \implies X:C \implies R) \implies R$
 $\langle proof \rangle$

5.4.18 Inter

lemma *Inter-iff* [*simp, noatp*]: $(A : Inter\ C) = (ALL\ X:C. A:X)$
 $\langle proof \rangle$

lemma *InterI* [*intro!*]: $(!X. X:C \implies A:X) \implies A : Inter\ C$
 $\langle proof \rangle$

A “destruct” rule – every X in C contains A as an element, but $A \in X$ can hold when $X \in C$ does not! This rule is analogous to *spec*.

lemma *InterD* [*elim*]: $A : Inter\ C \implies X:C \implies A:X$
 $\langle proof \rangle$

lemma *InterE* [*elim*]: $A : Inter\ C \implies (X \sim C \implies R) \implies (A:X \implies R) \implies R$

— “Classical” elimination rule – does not require proving $X \in C$.
 $\langle proof \rangle$

Image of a set under a function. Frequently b does not have the syntactic form of $f\ x$.

declare *image-def* [*noatp*]

lemma *image-eqI* [*simp, intro*]: $b = f\ x \implies x:A \implies b : f'A$
 $\langle proof \rangle$

lemma *imageI*: $x : A \implies f\ x : f\ 'A$
 $\langle proof \rangle$

lemma *rev-image-eqI*: $x:A \implies b = f\ x \implies b : f'A$
 — This version’s more effective when we already have the required x .
 $\langle proof \rangle$

lemma *imageE* [*elim!*]:
 $b : (\%x. f\ x) 'A \implies (!x. b = f\ x \implies x:A \implies P) \implies P$
 — The eta-expansion gives variable-name preservation.
 $\langle proof \rangle$

lemma *image-Un*: $f'(A \text{ Un } B) = f'A \text{ Un } f'B$
 $\langle proof \rangle$

lemma *image-eq-UN*: $f'A = (UN\ x:A. \{f\ x\})$
 $\langle proof \rangle$

lemma *image-iff*: $(z : f'A) = (EX\ x:A. z = f\ x)$
 $\langle proof \rangle$

lemma *image-subset-iff*: $(f'A \subseteq B) = (\forall x \in A. f\ x \in B)$
 — This rewrite rule would confuse users if made default.
 $\langle proof \rangle$

lemma *subset-image-iff*: $(B \subseteq f'A) = (EX\ AA. AA \subseteq A \ \& \ B = f'AA)$
 $\langle proof \rangle$

lemma *image-subsetI*: $(!x. x \in A \implies f\ x \in B) \implies f'A \subseteq B$
 — Replaces the three steps *subsetI*, *imageE*, *hypsubst*, but breaks too many existing proofs.
 $\langle proof \rangle$

Range of a function – just a translation for image!

lemma *range-eqI*: $b = f\ x \implies b \in \text{range } f$
 $\langle proof \rangle$

lemma *rangeI*: $f\ x \in \text{range } f$
 $\langle proof \rangle$

lemma *rangeE* [*elim?*]: $b \in \text{range } (\lambda x. f\ x) \implies (!x. b = f\ x \implies P) \implies P$
 $\langle proof \rangle$

5.4.19 Set reasoning tools

Rewrite rules for boolean case-splitting: faster than *split-if* [*split*].

lemma *split-if-eq1*: $((\text{if } Q \text{ then } x \text{ else } y) = b) = ((Q \dashrightarrow x = b) \ \& \ (\sim Q \dashrightarrow y = b))$
 $\langle proof \rangle$

lemma *split-if-eq2*: $(a = (\text{if } Q \text{ then } x \text{ else } y)) = ((Q \multimap a = x) \ \& \ (\sim Q \multimap a = y))$
 $\langle \text{proof} \rangle$

Split ifs on either side of the membership relation. Not for *[simp]* – can cause goals to blow up!

lemma *split-if-mem1*: $((\text{if } Q \text{ then } x \text{ else } y) : b) = ((Q \multimap x : b) \ \& \ (\sim Q \multimap y : b))$
 $\langle \text{proof} \rangle$

lemma *split-if-mem2*: $(a : (\text{if } Q \text{ then } x \text{ else } y)) = ((Q \multimap a : x) \ \& \ (\sim Q \multimap a : y))$
 $\langle \text{proof} \rangle$

lemmas *split-ifs* = *if-bool-eq-conj split-if-eq1 split-if-eq2 split-if-mem1 split-if-mem2*

$\langle ML \rangle$

5.4.20 The “proper subset” relation

lemma *psubsetI* [*intro!,noatp*]: $A \subseteq B \implies A \neq B \implies A \subset B$
 $\langle \text{proof} \rangle$

lemma *psubsetE* [*elim!,noatp*]:
 $[| A \subset B; [| A \subseteq B; \sim (B \subseteq A) |] \implies R |] \implies R$
 $\langle \text{proof} \rangle$

lemma *psubset-insert-iff*:
 $(A \subset \text{insert } x \ B) = (\text{if } x \in B \text{ then } A \subset B \text{ else if } x \in A \text{ then } A - \{x\} \subset B \text{ else } A \subseteq B)$
 $\langle \text{proof} \rangle$

lemma *psubset-eq*: $(A \subset B) = (A \subseteq B \ \& \ A \neq B)$
 $\langle \text{proof} \rangle$

lemma *psubset-imp-subset*: $A \subset B \implies A \subseteq B$
 $\langle \text{proof} \rangle$

lemma *psubset-trans*: $[| A \subset B; B \subset C |] \implies A \subset C$
 $\langle \text{proof} \rangle$

lemma *psubsetD*: $[| A \subset B; c \in A |] \implies c \in B$
 $\langle \text{proof} \rangle$

lemma *psubset-subset-trans*: $A \subset B \implies B \subseteq C \implies A \subset C$
 $\langle \text{proof} \rangle$

lemma *subset-psubset-trans*: $A \subseteq B \implies B \subseteq C \implies A \subseteq C$
 $\langle \text{proof} \rangle$

lemma *psubset-imp-ex-mem*: $A \subseteq B \implies \exists b. b \in (B - A)$
 $\langle \text{proof} \rangle$

lemma *atomize-ball*:
 $(!!x. x \in A \implies P x) == \text{Trueprop } (\forall x \in A. P x)$
 $\langle \text{proof} \rangle$

lemmas [*symmetric, rulify*] = *atomize-ball*
and [*symmetric, defn*] = *atomize-ball*

5.5 Further set-theory lemmas

5.5.1 Derived rules involving subsets.

insert.

lemma *subset-insertI*: $B \subseteq \text{insert } a \ B$
 $\langle \text{proof} \rangle$

lemma *subset-insertI2*: $A \subseteq B \implies A \subseteq \text{insert } b \ B$
 $\langle \text{proof} \rangle$

lemma *subset-insert*: $x \notin A \implies (A \subseteq \text{insert } x \ B) = (A \subseteq B)$
 $\langle \text{proof} \rangle$

Big Union – least upper bound of a set.

lemma *Union-upper*: $B \in A \implies B \subseteq \text{Union } A$
 $\langle \text{proof} \rangle$

lemma *Union-least*: $(!!X. X \in A \implies X \subseteq C) \implies \text{Union } A \subseteq C$
 $\langle \text{proof} \rangle$

General union.

lemma *UN-upper*: $a \in A \implies B \ a \subseteq (\bigcup_{x \in A. B \ x})$
 $\langle \text{proof} \rangle$

lemma *UN-least*: $(!!x. x \in A \implies B \ x \subseteq C) \implies (\bigcup_{x \in A. B \ x}) \subseteq C$
 $\langle \text{proof} \rangle$

Big Intersection – greatest lower bound of a set.

lemma *Inter-lower*: $B \in A \implies \text{Inter } A \subseteq B$
 $\langle \text{proof} \rangle$

lemma *Inter-subset*:
 $[! !X. X \in A \implies X \subseteq B; A \sim = \{\}] \implies \bigcap A \subseteq B$

$\langle proof \rangle$

lemma *Inter-greatest*: $(!!X. X \in A ==> C \subseteq X) ==> C \subseteq Inter\ A$
 $\langle proof \rangle$

lemma *INT-lower*: $a \in A ==> (\bigcap_{x \in A}. B\ x) \subseteq B\ a$
 $\langle proof \rangle$

lemma *INT-greatest*: $(!!x. x \in A ==> C \subseteq B\ x) ==> C \subseteq (\bigcap_{x \in A}. B\ x)$
 $\langle proof \rangle$

Finite Union – the least upper bound of two sets.

lemma *Un-upper1*: $A \subseteq A \cup B$
 $\langle proof \rangle$

lemma *Un-upper2*: $B \subseteq A \cup B$
 $\langle proof \rangle$

lemma *Un-least*: $A \subseteq C ==> B \subseteq C ==> A \cup B \subseteq C$
 $\langle proof \rangle$

Finite Intersection – the greatest lower bound of two sets.

lemma *Int-lower1*: $A \cap B \subseteq A$
 $\langle proof \rangle$

lemma *Int-lower2*: $A \cap B \subseteq B$
 $\langle proof \rangle$

lemma *Int-greatest*: $C \subseteq A ==> C \subseteq B ==> C \subseteq A \cap B$
 $\langle proof \rangle$

Set difference.

lemma *Diff-subset*: $A - B \subseteq A$
 $\langle proof \rangle$

lemma *Diff-subset-conv*: $(A - B \subseteq C) = (A \subseteq B \cup C)$
 $\langle proof \rangle$

5.5.2 Equalities involving union, intersection, inclusion, etc.

$\{\}$.

lemma *Collect-const* [simp]: $\{s. P\} = (if\ P\ then\ UNIV\ else\ \{\})$
 — supersedes *Collect-False-empty*
 $\langle proof \rangle$

lemma *subset-empty* [simp]: $(A \subseteq \{\}) = (A = \{\})$
 $\langle proof \rangle$

lemma *not-psubset-empty* [iff]: $\neg (A < \{\})$
 ⟨proof⟩

lemma *Collect-empty-eq* [simp]: $(\text{Collect } P = \{\}) = (\forall x. \neg P x)$
 ⟨proof⟩

lemma *empty-Collect-eq* [simp]: $(\{\} = \text{Collect } P) = (\forall x. \neg P x)$
 ⟨proof⟩

lemma *Collect-neg-eq*: $\{x. \neg P x\} = - \{x. P x\}$
 ⟨proof⟩

lemma *Collect-disj-eq*: $\{x. P x \mid Q x\} = \{x. P x\} \cup \{x. Q x\}$
 ⟨proof⟩

lemma *Collect-imp-eq*: $\{x. P x \longrightarrow Q x\} = -\{x. P x\} \cup \{x. Q x\}$
 ⟨proof⟩

lemma *Collect-conj-eq*: $\{x. P x \ \& \ Q x\} = \{x. P x\} \cap \{x. Q x\}$
 ⟨proof⟩

lemma *Collect-all-eq*: $\{x. \forall y. P x y\} = (\bigcap y. \{x. P x y\})$
 ⟨proof⟩

lemma *Collect-ball-eq*: $\{x. \forall y \in A. P x y\} = (\bigcap y \in A. \{x. P x y\})$
 ⟨proof⟩

lemma *Collect-ex-eq* [noatp]: $\{x. \exists y. P x y\} = (\bigcup y. \{x. P x y\})$
 ⟨proof⟩

lemma *Collect-bex-eq* [noatp]: $\{x. \exists y \in A. P x y\} = (\bigcup y \in A. \{x. P x y\})$
 ⟨proof⟩

insert.

lemma *insert-is-Un*: $\text{insert } a \ A = \{a\} \ \text{Un } A$
 — NOT SUITABLE FOR REWRITING since $\{a\} == \text{insert } a \ \{\}$
 ⟨proof⟩

lemma *insert-not-empty* [simp]: $\text{insert } a \ A \neq \{\}$
 ⟨proof⟩

lemmas *empty-not-insert* = *insert-not-empty* [symmetric, standard]
declare *empty-not-insert* [simp]

lemma *insert-absorb*: $a \in A ==> \text{insert } a \ A = A$
 — [simp] causes recursive calls when there are nested inserts
 — with *quadratic* running time
 ⟨proof⟩

lemma *insert-absorb2* [simp]: $\text{insert } x (\text{insert } x A) = \text{insert } x A$
 ⟨proof⟩

lemma *insert-commute*: $\text{insert } x (\text{insert } y A) = \text{insert } y (\text{insert } x A)$
 ⟨proof⟩

lemma *insert-subset* [simp]: $(\text{insert } x A \subseteq B) = (x \in B \ \& \ A \subseteq B)$
 ⟨proof⟩

lemma *mk-disjoint-insert*: $a \in A \implies \exists B. A = \text{insert } a B \ \& \ a \notin B$
 — use new B rather than $A - \{a\}$ to avoid infinite unfolding
 ⟨proof⟩

lemma *insert-Collect*: $\text{insert } a (\text{Collect } P) = \{u. u \neq a \longrightarrow P u\}$
 ⟨proof⟩

lemma *UN-insert-distrib*: $u \in A \implies (\bigcup_{x \in A. \text{insert } a (B x)) = \text{insert } a (\bigcup_{x \in A. B x})$
 ⟨proof⟩

lemma *insert-inter-insert*[simp]: $\text{insert } a A \cap \text{insert } a B = \text{insert } a (A \cap B)$
 ⟨proof⟩

lemma *insert-disjoint* [simp,noatp]:
 $(\text{insert } a A \cap B = \{\}) = (a \notin B \ \& \ A \cap B = \{\})$
 $(\{\} = \text{insert } a A \cap B) = (a \notin B \ \& \ \{\} = A \cap B)$
 ⟨proof⟩

lemma *disjoint-insert* [simp,noatp]:
 $(B \cap \text{insert } a A = \{\}) = (a \notin B \ \& \ B \cap A = \{\})$
 $(\{\} = A \cap \text{insert } b B) = (b \notin A \ \& \ \{\} = A \cap B)$
 ⟨proof⟩

image.

lemma *image-empty* [simp]: $f \cdot \{\} = \{\}$
 ⟨proof⟩

lemma *image-insert* [simp]: $f \cdot \text{insert } a B = \text{insert } (f a) (f \cdot B)$
 ⟨proof⟩

lemma *image-constant*: $x \in A \implies (\lambda x. c) \cdot A = \{c\}$
 ⟨proof⟩

lemma *image-constant-conv*: $(\%x. c) \cdot A = (\text{if } A = \{\} \text{ then } \{\} \text{ else } \{c\})$
 ⟨proof⟩

lemma *image-image*: $f \cdot (g \cdot A) = (\lambda x. f (g x)) \cdot A$
 ⟨proof⟩

lemma *insert-image* [simp]: $x \in A \implies \text{insert } (f\ x) (f^{\circ}A) = f^{\circ}A$
 ⟨proof⟩

lemma *image-is-empty* [iff]: $(f^{\circ}A = \{\}) = (A = \{\})$
 ⟨proof⟩

lemma *image-Collect* [noatp]: $f^{\circ} \{x. P\ x\} = \{f\ x \mid x. P\ x\}$
 — NOT suitable as a default simp rule: the RHS isn’t simpler than the LHS, with its implicit quantifier and conjunction. Also image enjoys better equational properties than does the RHS.
 ⟨proof⟩

lemma *if-image-distrib* [simp]:
 $(\lambda x. \text{if } P\ x \text{ then } f\ x \text{ else } g\ x)^{\circ} S$
 $= (f^{\circ} (S \cap \{x. P\ x\})) \cup (g^{\circ} (S \cap \{x. \neg P\ x\}))$
 ⟨proof⟩

lemma *image-cong*: $M = N \implies (!x. x \in N \implies f\ x = g\ x) \implies f^{\circ}M = g^{\circ}N$
 ⟨proof⟩

range.

lemma *full-SetCompr-eq* [noatp]: $\{u. \exists x. u = f\ x\} = \text{range } f$
 ⟨proof⟩

lemma *range-composition*: $\text{range } (\lambda x. f\ (g\ x)) = f^{\circ} \text{range } g$
 ⟨proof⟩

Int

lemma *Int-absorb* [simp]: $A \cap A = A$
 ⟨proof⟩

lemma *Int-left-absorb*: $A \cap (A \cap B) = A \cap B$
 ⟨proof⟩

lemma *Int-commute*: $A \cap B = B \cap A$
 ⟨proof⟩

lemma *Int-left-commute*: $A \cap (B \cap C) = B \cap (A \cap C)$
 ⟨proof⟩

lemma *Int-assoc*: $(A \cap B) \cap C = A \cap (B \cap C)$
 ⟨proof⟩

lemmas *Int-ac = Int-assoc Int-left-absorb Int-commute Int-left-commute*
 — Intersection is an AC-operator

lemma *Int-absorb1*: $B \subseteq A \implies A \cap B = B$
 $\langle proof \rangle$

lemma *Int-absorb2*: $A \subseteq B \implies A \cap B = A$
 $\langle proof \rangle$

lemma *Int-empty-left [simp]*: $\{\} \cap B = \{\}$
 $\langle proof \rangle$

lemma *Int-empty-right [simp]*: $A \cap \{\} = \{\}$
 $\langle proof \rangle$

lemma *disjoint-eq-subset-Compl*: $(A \cap B = \{\}) = (A \subseteq -B)$
 $\langle proof \rangle$

lemma *disjoint-iff-not-equal*: $(A \cap B = \{\}) = (\forall x \in A. \forall y \in B. x \neq y)$
 $\langle proof \rangle$

lemma *Int-UNIV-left [simp]*: $UNIV \cap B = B$
 $\langle proof \rangle$

lemma *Int-UNIV-right [simp]*: $A \cap UNIV = A$
 $\langle proof \rangle$

lemma *Int-eq-Inter*: $A \cap B = \bigcap \{A, B\}$
 $\langle proof \rangle$

lemma *Int-Un-distrib*: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
 $\langle proof \rangle$

lemma *Int-Un-distrib2*: $(B \cup C) \cap A = (B \cap A) \cup (C \cap A)$
 $\langle proof \rangle$

lemma *Int-UNIV [simp,noatp]*: $(A \cap B = UNIV) = (A = UNIV \ \& \ B = UNIV)$
 $\langle proof \rangle$

lemma *Int-subset-iff [simp]*: $(C \subseteq A \cap B) = (C \subseteq A \ \& \ C \subseteq B)$
 $\langle proof \rangle$

lemma *Int-Collect*: $(x \in A \cap \{x. P \ x\}) = (x \in A \ \& \ P \ x)$
 $\langle proof \rangle$

Un.

lemma *Un-absorb [simp]*: $A \cup A = A$
 $\langle proof \rangle$

lemma *Un-left-absorb*: $A \cup (A \cup B) = A \cup B$
 $\langle proof \rangle$

lemma *Un-commute*: $A \cup B = B \cup A$
 $\langle proof \rangle$

lemma *Un-left-commute*: $A \cup (B \cup C) = B \cup (A \cup C)$
 $\langle proof \rangle$

lemma *Un-assoc*: $(A \cup B) \cup C = A \cup (B \cup C)$
 $\langle proof \rangle$

lemmas *Un-ac = Un-assoc Un-left-absorb Un-commute Un-left-commute*
 — Union is an AC-operator

lemma *Un-absorb1*: $A \subseteq B \implies A \cup B = B$
 $\langle proof \rangle$

lemma *Un-absorb2*: $B \subseteq A \implies A \cup B = A$
 $\langle proof \rangle$

lemma *Un-empty-left [simp]*: $\{\} \cup B = B$
 $\langle proof \rangle$

lemma *Un-empty-right [simp]*: $A \cup \{\} = A$
 $\langle proof \rangle$

lemma *Un-UNIV-left [simp]*: $UNIV \cup B = UNIV$
 $\langle proof \rangle$

lemma *Un-UNIV-right [simp]*: $A \cup UNIV = UNIV$
 $\langle proof \rangle$

lemma *Un-eq-Union*: $A \cup B = \bigcup \{A, B\}$
 $\langle proof \rangle$

lemma *Un-insert-left [simp]*: $(insert\ a\ B) \cup C = insert\ a\ (B \cup C)$
 $\langle proof \rangle$

lemma *Un-insert-right [simp]*: $A \cup (insert\ a\ B) = insert\ a\ (A \cup B)$
 $\langle proof \rangle$

lemma *Int-insert-left*:
 $(insert\ a\ B) \cap C = (if\ a \in C\ then\ insert\ a\ (B \cap C)\ else\ B \cap C)$
 $\langle proof \rangle$

lemma *Int-insert-right*:
 $A \cap (insert\ a\ B) = (if\ a \in A\ then\ insert\ a\ (A \cap B)\ else\ A \cap B)$
 $\langle proof \rangle$

lemma *Un-Int-distrib*: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
 $\langle proof \rangle$

lemma *Un-Int-distrib2*: $(B \cap C) \cup A = (B \cup A) \cap (C \cup A)$
 $\langle \text{proof} \rangle$

lemma *Un-Int-crazy*:
 $(A \cap B) \cup (B \cap C) \cup (C \cap A) = (A \cup B) \cap (B \cup C) \cap (C \cup A)$
 $\langle \text{proof} \rangle$

lemma *subset-Un-eq*: $(A \subseteq B) = (A \cup B = B)$
 $\langle \text{proof} \rangle$

lemma *Un-empty [iff]*: $(A \cup B = \{\}) = (A = \{\} \ \& \ B = \{\})$
 $\langle \text{proof} \rangle$

lemma *Un-subset-iff [simp]*: $(A \cup B \subseteq C) = (A \subseteq C \ \& \ B \subseteq C)$
 $\langle \text{proof} \rangle$

lemma *Un-Diff-Int*: $(A - B) \cup (A \cap B) = A$
 $\langle \text{proof} \rangle$

lemma *Diff-Int2*: $A \cap C - B \cap C = A \cap C - B$
 $\langle \text{proof} \rangle$

Set complement

lemma *Compl-disjoint [simp]*: $A \cap -A = \{\}$
 $\langle \text{proof} \rangle$

lemma *Compl-disjoint2 [simp]*: $-A \cap A = \{\}$
 $\langle \text{proof} \rangle$

lemma *Compl-partition*: $A \cup -A = UNIV$
 $\langle \text{proof} \rangle$

lemma *Compl-partition2*: $-A \cup A = UNIV$
 $\langle \text{proof} \rangle$

lemma *double-complement [simp]*: $-(-A) = (A::'a \text{ set})$
 $\langle \text{proof} \rangle$

lemma *Compl-Un [simp]*: $-(A \cup B) = (-A) \cap (-B)$
 $\langle \text{proof} \rangle$

lemma *Compl-Int [simp]*: $-(A \cap B) = (-A) \cup (-B)$
 $\langle \text{proof} \rangle$

lemma *Compl-UN [simp]*: $-(\bigcup x \in A. B \ x) = (\bigcap x \in A. -B \ x)$
 $\langle \text{proof} \rangle$

lemma *Compl-INT [simp]*: $-(\bigcap x \in A. B \ x) = (\bigcup x \in A. -B \ x)$

$\langle proof \rangle$

lemma *subset-Compl-self-eq*: $(A \subseteq -A) = (A = \{\})$
 $\langle proof \rangle$

lemma *Un-Int-assoc-eq*: $((A \cap B) \cup C = A \cap (B \cup C)) = (C \subseteq A)$
 — Halmos, Naive Set Theory, page 16.
 $\langle proof \rangle$

lemma *Compl-UNIV-eq* [simp]: $-UNIV = \{\}$
 $\langle proof \rangle$

lemma *Compl-empty-eq* [simp]: $-\{\} = UNIV$
 $\langle proof \rangle$

lemma *Compl-subset-Compl-iff* [iff]: $(-A \subseteq -B) = (B \subseteq A)$
 $\langle proof \rangle$

lemma *Compl-eq-Compl-iff* [iff]: $(-A = -B) = (A = (B::'a\ set))$
 $\langle proof \rangle$

Union.

lemma *Union-empty* [simp]: $Union(\{\}) = \{\}$
 $\langle proof \rangle$

lemma *Union-UNIV* [simp]: $Union\ UNIV = UNIV$
 $\langle proof \rangle$

lemma *Union-insert* [simp]: $Union\ (insert\ a\ B) = a \cup \bigcup B$
 $\langle proof \rangle$

lemma *Union-Un-distrib* [simp]: $\bigcup (A\ Un\ B) = \bigcup A \cup \bigcup B$
 $\langle proof \rangle$

lemma *Union-Int-subset*: $\bigcup (A \cap B) \subseteq \bigcup A \cap \bigcup B$
 $\langle proof \rangle$

lemma *Union-empty-conv* [simp,noatp]: $(\bigcup A = \{\}) = (\forall x \in A. x = \{\})$
 $\langle proof \rangle$

lemma *empty-Union-conv* [simp,noatp]: $(\{\} = \bigcup A) = (\forall x \in A. x = \{\})$
 $\langle proof \rangle$

lemma *Union-disjoint*: $(\bigcup C \cap A = \{\}) = (\forall B \in C. B \cap A = \{\})$
 $\langle proof \rangle$

Inter.

lemma *Inter-empty* [simp]: $\bigcap \{\} = UNIV$

$\langle proof \rangle$

lemma *Inter-UNIV* [simp]: $\bigcap UNIV = \{\}$
 $\langle proof \rangle$

lemma *Inter-insert* [simp]: $\bigcap (\text{insert } a \ B) = a \cap \bigcap B$
 $\langle proof \rangle$

lemma *Inter-Un-subset*: $\bigcap A \cup \bigcap B \subseteq \bigcap (A \cap B)$
 $\langle proof \rangle$

lemma *Inter-Un-distrib*: $\bigcap (A \cup B) = \bigcap A \cap \bigcap B$
 $\langle proof \rangle$

lemma *Inter-UNIV-conv* [simp, noatp]:
 $(\bigcap A = UNIV) = (\forall x \in A. x = UNIV)$
 $(UNIV = \bigcap A) = (\forall x \in A. x = UNIV)$
 $\langle proof \rangle$

UN and *INT*.

Basic identities:

lemma *UN-empty* [simp, noatp]: $(\bigcup x \in \{\}. B \ x) = \{\}$
 $\langle proof \rangle$

lemma *UN-empty2* [simp]: $(\bigcup x \in A. \{\}) = \{\}$
 $\langle proof \rangle$

lemma *UN-singleton* [simp]: $(\bigcup x \in A. \{x\}) = A$
 $\langle proof \rangle$

lemma *UN-absorb*: $k \in I \implies A \ k \cup (\bigcup i \in I. A \ i) = (\bigcup i \in I. A \ i)$
 $\langle proof \rangle$

lemma *INT-empty* [simp]: $(\bigcap x \in \{\}. B \ x) = UNIV$
 $\langle proof \rangle$

lemma *INT-absorb*: $k \in I \implies A \ k \cap (\bigcap i \in I. A \ i) = (\bigcap i \in I. A \ i)$
 $\langle proof \rangle$

lemma *UN-insert* [simp]: $(\bigcup x \in \text{insert } a \ A. B \ x) = B \ a \cup \text{UNION } A \ B$
 $\langle proof \rangle$

lemma *UN-Un* [simp]: $(\bigcup i \in A \cup B. M \ i) = (\bigcup i \in A. M \ i) \cup (\bigcup i \in B. M \ i)$
 $\langle proof \rangle$

lemma *UN-UN-flatten*: $(\bigcup x \in (\bigcup y \in A. B \ y). C \ x) = (\bigcup y \in A. \bigcup x \in B \ y. C \ x)$
 $\langle proof \rangle$

lemma *UN-subset-iff*: $((\bigcup i \in I. A \ i) \subseteq B) = (\forall i \in I. A \ i \subseteq B)$

$\langle proof \rangle$

lemma *INT-subset-iff*: $(B \subseteq (\bigcap_{i \in I}. A \ i)) = (\forall i \in I. B \subseteq A \ i)$
 $\langle proof \rangle$

lemma *INT-insert [simp]*: $(\bigcap x \in \text{insert } a \ A. B \ x) = B \ a \cap \text{INTER } A \ B$
 $\langle proof \rangle$

lemma *INT-Un*: $(\bigcap i \in A \cup B. M \ i) = (\bigcap i \in A. M \ i) \cap (\bigcap i \in B. M \ i)$
 $\langle proof \rangle$

lemma *INT-insert-distrib*:
 $u \in A ==> (\bigcap x \in A. \text{insert } a \ (B \ x)) = \text{insert } a \ (\bigcap x \in A. B \ x)$
 $\langle proof \rangle$

lemma *Union-image-eq [simp]*: $\bigcup (B' A) = (\bigcup x \in A. B \ x)$
 $\langle proof \rangle$

lemma *image-Union*: $f \ ' \bigcup S = (\bigcup x \in S. f \ ' x)$
 $\langle proof \rangle$

lemma *Inter-image-eq [simp]*: $\bigcap (B' A) = (\bigcap x \in A. B \ x)$
 $\langle proof \rangle$

lemma *UN-constant [simp]*: $(\bigcup y \in A. c) = (\text{if } A = \{\} \text{ then } \{\} \text{ else } c)$
 $\langle proof \rangle$

lemma *INT-constant [simp]*: $(\bigcap y \in A. c) = (\text{if } A = \{\} \text{ then } \text{UNIV} \text{ else } c)$
 $\langle proof \rangle$

lemma *UN-eq*: $(\bigcup x \in A. B \ x) = \bigcup (\{ Y. \exists x \in A. Y = B \ x \})$
 $\langle proof \rangle$

lemma *INT-eq*: $(\bigcap x \in A. B \ x) = \bigcap (\{ Y. \exists x \in A. Y = B \ x \})$
 — Look: it has an *existential* quantifier
 $\langle proof \rangle$

lemma *UNION-empty-conv[simp]*:
 $(\{\} = (\text{UN } x:A. B \ x)) = (\forall x \in A. B \ x = \{\})$
 $((\text{UN } x:A. B \ x) = \{\}) = (\forall x \in A. B \ x = \{\})$
 $\langle proof \rangle$

lemma *INTER-UNIV-conv[simp]*:
 $(\text{UNIV} = (\text{INT } x:A. B \ x)) = (\forall x \in A. B \ x = \text{UNIV})$
 $((\text{INT } x:A. B \ x) = \text{UNIV}) = (\forall x \in A. B \ x = \text{UNIV})$
 $\langle proof \rangle$

Distributive laws:

lemma *Int-Union*: $A \cap \bigcup B = (\bigcup C \in B. A \cap C)$

$\langle proof \rangle$

lemma *Int-Union2*: $\bigcup B \cap A = (\bigcup C \in B. C \cap A)$
 $\langle proof \rangle$

lemma *Un-Union-image*: $(\bigcup x \in C. A \ x \cup B \ x) = \bigcup (A' C) \cup \bigcup (B' C)$
 — Devlin, Fundamentals of Contemporary Set Theory, page 12, exercise 5:
 — Union of a family of unions
 $\langle proof \rangle$

lemma *UN-Un-distrib*: $(\bigcup i \in I. A \ i \cup B \ i) = (\bigcup i \in I. A \ i) \cup (\bigcup i \in I. B \ i)$
 — Equivalent version
 $\langle proof \rangle$

lemma *Un-Inter*: $A \cup \bigcap B = (\bigcap C \in B. A \cup C)$
 $\langle proof \rangle$

lemma *Int-Inter-image*: $(\bigcap x \in C. A \ x \cap B \ x) = \bigcap (A' C) \cap \bigcap (B' C)$
 $\langle proof \rangle$

lemma *INT-Int-distrib*: $(\bigcap i \in I. A \ i \cap B \ i) = (\bigcap i \in I. A \ i) \cap (\bigcap i \in I. B \ i)$
 — Equivalent version
 $\langle proof \rangle$

lemma *Int-UN-distrib*: $B \cap (\bigcup i \in I. A \ i) = (\bigcup i \in I. B \cap A \ i)$
 — Halmos, Naive Set Theory, page 35.
 $\langle proof \rangle$

lemma *Un-INT-distrib*: $B \cup (\bigcap i \in I. A \ i) = (\bigcap i \in I. B \cup A \ i)$
 $\langle proof \rangle$

lemma *Int-UN-distrib2*: $(\bigcup i \in I. A \ i) \cap (\bigcup j \in J. B \ j) = (\bigcup i \in I. \bigcup j \in J. A \ i \cap B \ j)$
 $\langle proof \rangle$

lemma *Un-INT-distrib2*: $(\bigcap i \in I. A \ i) \cup (\bigcap j \in J. B \ j) = (\bigcap i \in I. \bigcap j \in J. A \ i \cup B \ j)$
 $\langle proof \rangle$

Bounded quantifiers.

The following are not added to the default simpset because (a) they duplicate the body and (b) there are no similar rules for *Int*.

lemma *ball-Un*: $(\forall x \in A \cup B. P \ x) = ((\forall x \in A. P \ x) \ \& \ (\forall x \in B. P \ x))$
 $\langle proof \rangle$

lemma *beX-Un*: $(\exists x \in A \cup B. P \ x) = ((\exists x \in A. P \ x) \mid (\exists x \in B. P \ x))$
 $\langle proof \rangle$

lemma *ball-UN*: $(\forall z \in \text{UNION } A \ B. P \ z) = (\forall x \in A. \forall z \in B \ x. P \ z)$
 $\langle \text{proof} \rangle$

lemma *beX-UN*: $(\exists z \in \text{UNION } A \ B. P \ z) = (\exists x \in A. \exists z \in B \ x. P \ z)$
 $\langle \text{proof} \rangle$

Set difference.

lemma *Diff-eq*: $A - B = A \cap (-B)$
 $\langle \text{proof} \rangle$

lemma *Diff-eq-empty-iff* [simp, noatp]: $(A - B = \{\}) = (A \subseteq B)$
 $\langle \text{proof} \rangle$

lemma *Diff-cancel* [simp]: $A - A = \{\}$
 $\langle \text{proof} \rangle$

lemma *Diff-idemp* [simp]: $(A - B) - B = A - (B::'a \text{ set})$
 $\langle \text{proof} \rangle$

lemma *Diff-triv*: $A \cap B = \{\} \implies A - B = A$
 $\langle \text{proof} \rangle$

lemma *empty-Diff* [simp]: $\{\} - A = \{\}$
 $\langle \text{proof} \rangle$

lemma *Diff-empty* [simp]: $A - \{\} = A$
 $\langle \text{proof} \rangle$

lemma *Diff-UNIV* [simp]: $A - \text{UNIV} = \{\}$
 $\langle \text{proof} \rangle$

lemma *Diff-insert0* [simp, noatp]: $x \notin A \implies A - \text{insert } x \ B = A - B$
 $\langle \text{proof} \rangle$

lemma *Diff-insert*: $A - \text{insert } a \ B = A - B - \{a\}$
 — NOT SUITABLE FOR REWRITING since $\{a\} == \text{insert } a \ 0$
 $\langle \text{proof} \rangle$

lemma *Diff-insert2*: $A - \text{insert } a \ B = A - \{a\} - B$
 — NOT SUITABLE FOR REWRITING since $\{a\} == \text{insert } a \ 0$
 $\langle \text{proof} \rangle$

lemma *insert-Diff-if*: $\text{insert } x \ A - B = (\text{if } x \in B \text{ then } A - B \text{ else } \text{insert } x \ (A - B))$
 $\langle \text{proof} \rangle$

lemma *insert-Diff1* [simp]: $x \in B \implies \text{insert } x \ A - B = A - B$
 $\langle \text{proof} \rangle$

lemma *insert-Diff-single* [simp]: $\text{insert } a (A - \{a\}) = \text{insert } a A$
 $\langle \text{proof} \rangle$

lemma *insert-Diff*: $a \in A \implies \text{insert } a (A - \{a\}) = A$
 $\langle \text{proof} \rangle$

lemma *Diff-insert-absorb*: $x \notin A \implies (\text{insert } x A) - \{x\} = A$
 $\langle \text{proof} \rangle$

lemma *Diff-disjoint* [simp]: $A \cap (B - A) = \{\}$
 $\langle \text{proof} \rangle$

lemma *Diff-partition*: $A \subseteq B \implies A \cup (B - A) = B$
 $\langle \text{proof} \rangle$

lemma *double-diff*: $A \subseteq B \implies B \subseteq C \implies B - (C - A) = A$
 $\langle \text{proof} \rangle$

lemma *Un-Diff-cancel* [simp]: $A \cup (B - A) = A \cup B$
 $\langle \text{proof} \rangle$

lemma *Un-Diff-cancel2* [simp]: $(B - A) \cup A = B \cup A$
 $\langle \text{proof} \rangle$

lemma *Diff-Un*: $A - (B \cup C) = (A - B) \cap (A - C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int*: $A - (B \cap C) = (A - B) \cup (A - C)$
 $\langle \text{proof} \rangle$

lemma *Un-Diff*: $(A \cup B) - C = (A - C) \cup (B - C)$
 $\langle \text{proof} \rangle$

lemma *Int-Diff*: $(A \cap B) - C = A \cap (B - C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int-distrib*: $C \cap (A - B) = (C \cap A) - (C \cap B)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int-distrib2*: $(A - B) \cap C = (A \cap C) - (B \cap C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Compl* [simp]: $A - (- B) = A \cap B$
 $\langle \text{proof} \rangle$

lemma *Compl-Diff-eq* [simp]: $- (A - B) = -A \cup B$
 $\langle \text{proof} \rangle$

Quantification over type *bool*.

lemma *bool-induct*: $P \text{ True} \implies P \text{ False} \implies P x$
 $\langle \text{proof} \rangle$

lemma *all-bool-eq*: $(\forall b. P b) \longleftrightarrow P \text{ True} \wedge P \text{ False}$
 $\langle \text{proof} \rangle$

lemma *bool-contrapos*: $P x \implies \neg P \text{ False} \implies P \text{ True}$
 $\langle \text{proof} \rangle$

lemma *ex-bool-eq*: $(\exists b. P b) \longleftrightarrow P \text{ True} \vee P \text{ False}$
 $\langle \text{proof} \rangle$

lemma *Un-eq-UN*: $A \cup B = (\bigcup b. \text{if } b \text{ then } A \text{ else } B)$
 $\langle \text{proof} \rangle$

lemma *UN-bool-eq*: $(\bigcup b::\text{bool}. A b) = (A \text{ True} \cup A \text{ False})$
 $\langle \text{proof} \rangle$

lemma *INT-bool-eq*: $(\bigcap b::\text{bool}. A b) = (A \text{ True} \cap A \text{ False})$
 $\langle \text{proof} \rangle$

Pow

lemma *Pow-empty [simp]*: $\text{Pow } \{\} = \{\{\}\}$
 $\langle \text{proof} \rangle$

lemma *Pow-insert*: $\text{Pow } (\text{insert } a \ A) = \text{Pow } A \cup (\text{insert } a \ ' \text{Pow } A)$
 $\langle \text{proof} \rangle$

lemma *Pow-Compl*: $\text{Pow } (- \ A) = \{-B \mid B. A \in \text{Pow } B\}$
 $\langle \text{proof} \rangle$

lemma *Pow-UNIV [simp]*: $\text{Pow } \text{UNIV} = \text{UNIV}$
 $\langle \text{proof} \rangle$

lemma *Un-Pow-subset*: $\text{Pow } A \cup \text{Pow } B \subseteq \text{Pow } (A \cup B)$
 $\langle \text{proof} \rangle$

lemma *UN-Pow-subset*: $(\bigcup x \in A. \text{Pow } (B \ x)) \subseteq \text{Pow } (\bigcup x \in A. B \ x)$
 $\langle \text{proof} \rangle$

lemma *subset-Pow-Union*: $A \subseteq \text{Pow } (\bigcup A)$
 $\langle \text{proof} \rangle$

lemma *Union-Pow-eq [simp]*: $\bigcup (\text{Pow } A) = A$
 $\langle \text{proof} \rangle$

lemma *Pow-Int-eq [simp]*: $\text{Pow } (A \cap B) = \text{Pow } A \cap \text{Pow } B$
 $\langle \text{proof} \rangle$

lemma *Pow-INT-eq*: $\text{Pow } (\bigcap_{x \in A} B \ x) = (\bigcap_{x \in A} \text{Pow } (B \ x))$
 $\langle \text{proof} \rangle$

Miscellany.

lemma *set-eq-subset*: $(A = B) = (A \subseteq B \ \& \ B \subseteq A)$
 $\langle \text{proof} \rangle$

lemma *subset-iff*: $(A \subseteq B) = (\forall t. t \in A \longrightarrow t \in B)$
 $\langle \text{proof} \rangle$

lemma *subset-iff-psubset-eq*: $(A \subseteq B) = ((A \subset B) \mid (A = B))$
 $\langle \text{proof} \rangle$

lemma *all-not-in-conv* [*simp*]: $(\forall x. x \notin A) = (A = \{\})$
 $\langle \text{proof} \rangle$

lemma *ex-in-conv*: $(\exists x. x \in A) = (A \neq \{\})$
 $\langle \text{proof} \rangle$

lemma *distinct-lemma*: $f \ x \neq f \ y \implies x \neq y$
 $\langle \text{proof} \rangle$

Miniscoping: pushing in quantifiers and big Unions and Intersections.

lemma *UN-simps* [*simp*]:

$!!a \ B \ C. (\text{UN } x:C. \text{insert } a \ (B \ x)) = (\text{if } C=\{\} \text{ then } \{\} \text{ else insert } a \ (\text{UN } x:C. B \ x))$

$!!A \ B \ C. (\text{UN } x:C. A \ x \ \text{Un } B) = ((\text{if } C=\{\} \text{ then } \{\} \text{ else } (\text{UN } x:C. A \ x) \ \text{Un } B))$

$!!A \ B \ C. (\text{UN } x:C. A \ \text{Un } B \ x) = ((\text{if } C=\{\} \text{ then } \{\} \text{ else } A \ \text{Un } (\text{UN } x:C. B \ x)))$

$!!A \ B \ C. (\text{UN } x:C. A \ x \ \text{Int } B) = ((\text{UN } x:C. A \ x) \ \text{Int } B)$

$!!A \ B \ C. (\text{UN } x:C. A \ \text{Int } B \ x) = (A \ \text{Int } (\text{UN } x:C. B \ x))$

$!!A \ B \ C. (\text{UN } x:C. A \ x - B) = ((\text{UN } x:C. A \ x) - B)$

$!!A \ B \ C. (\text{UN } x:C. A - B \ x) = (A - (\text{INT } x:C. B \ x))$

$!!A \ B. (\text{UN } x: \text{Union } A. B \ x) = (\text{UN } y:A. \text{UN } x:y. B \ x)$

$!!A \ B \ C. (\text{UN } z: \text{UNION } A \ B. C \ z) = (\text{UN } x:A. \text{UN } z: B(x). C \ z)$

$!!A \ B \ f. (\text{UN } x:f'A. B \ x) = (\text{UN } a:A. B \ (f \ a))$

$\langle \text{proof} \rangle$

lemma *INT-simps* [*simp*]:

$!!A \ B \ C. (\text{INT } x:C. A \ x \ \text{Int } B) = (\text{if } C=\{\} \text{ then } \text{UNIV} \text{ else } (\text{INT } x:C. A \ x) \ \text{Int } B)$

$!!A \ B \ C. (\text{INT } x:C. A \ \text{Int } B \ x) = (\text{if } C=\{\} \text{ then } \text{UNIV} \text{ else } A \ \text{Int } (\text{INT } x:C. B \ x))$

$!!A \ B \ C. (\text{INT } x:C. A \ x - B) = (\text{if } C=\{\} \text{ then } \text{UNIV} \text{ else } (\text{INT } x:C. A \ x) - B)$

$!!A \ B \ C. (\text{INT } x:C. A - B \ x) = (\text{if } C=\{\} \text{ then } \text{UNIV} \text{ else } A - (\text{UN } x:C. B \ x))$

$!!a B C. (INT x:C. insert a (B x)) = insert a (INT x:C. B x)$
 $!!A B C. (INT x:C. A x Un B) = ((INT x:C. A x) Un B)$
 $!!A B C. (INT x:C. A Un B x) = (A Un (INT x:C. B x))$
 $!!A B. (INT x: Union A. B x) = (INT y:A. INT x:y. B x)$
 $!!A B C. (INT z: UNION A B. C z) = (INT x:A. INT z: B(x). C z)$
 $!!A B f. (INT x:f'A. B x) = (INT a:A. B (f a))$
 $\langle proof \rangle$

lemma *ball-simps* [*simp, noatp*]:

$!!A P Q. (ALL x:A. P x \mid Q) = ((ALL x:A. P x) \mid Q)$
 $!!A P Q. (ALL x:A. P \mid Q x) = (P \mid (ALL x:A. Q x))$
 $!!A P Q. (ALL x:A. P \dashrightarrow Q x) = (P \dashrightarrow (ALL x:A. Q x))$
 $!!A P Q. (ALL x:A. P x \dashrightarrow Q) = ((EX x:A. P x) \dashrightarrow Q)$
 $!!P. (ALL x:\{\}. P x) = True$
 $!!P. (ALL x:UNIV. P x) = (ALL x. P x)$
 $!!a B P. (ALL x:insert a B. P x) = (P a \& (ALL x:B. P x))$
 $!!A P. (ALL x:Union A. P x) = (ALL y:A. ALL x:y. P x)$
 $!!A B P. (ALL x: UNION A B. P x) = (ALL a:A. ALL x: B a. P x)$
 $!!P Q. (ALL x:Collect Q. P x) = (ALL x. Q x \dashrightarrow P x)$
 $!!A P f. (ALL x:f'A. P x) = (ALL x:A. P (f x))$
 $!!A P. (\sim (ALL x:A. P x)) = (EX x:A. \sim P x)$
 $\langle proof \rangle$

lemma *bex-simps* [*simp, noatp*]:

$!!A P Q. (EX x:A. P x \& Q) = ((EX x:A. P x) \& Q)$
 $!!A P Q. (EX x:A. P \& Q x) = (P \& (EX x:A. Q x))$
 $!!P. (EX x:\{\}. P x) = False$
 $!!P. (EX x:UNIV. P x) = (EX x. P x)$
 $!!a B P. (EX x:insert a B. P x) = (P(a) \mid (EX x:B. P x))$
 $!!A P. (EX x:Union A. P x) = (EX y:A. EX x:y. P x)$
 $!!A B P. (EX x: UNION A B. P x) = (EX a:A. EX x:B a. P x)$
 $!!P Q. (EX x:Collect Q. P x) = (EX x. Q x \& P x)$
 $!!A P f. (EX x:f'A. P x) = (EX x:A. P (f x))$
 $!!A P. (\sim (EX x:A. P x)) = (ALL x:A. \sim P x)$
 $\langle proof \rangle$

lemma *ball-conj-distrib*:

$(ALL x:A. P x \& Q x) = ((ALL x:A. P x) \& (ALL x:A. Q x))$
 $\langle proof \rangle$

lemma *bex-disj-distrib*:

$(EX x:A. P x \mid Q x) = ((EX x:A. P x) \mid (EX x:A. Q x))$
 $\langle proof \rangle$

Maxiscoping: pulling out big Unions and Intersections.

lemma *UN-extend-simps*:

$!!a B C. insert a (UN x:C. B x) = (if C=\{\} then \{a\} else (UN x:C. insert a (B x)))$
 $!!A B C. (UN x:C. A x) Un B = (if C=\{\} then B else (UN x:C. A x Un B))$

$!!A B C. A \text{ Un } (UN\ x:C. B\ x) = (if\ C=\{\} \text{ then } A \text{ else } (UN\ x:C. A \text{ Un } B\ x))$
 $!!A B C. ((UN\ x:C. A\ x) \text{ Int } B) = (UN\ x:C. A\ x \text{ Int } B)$
 $!!A B C. (A \text{ Int } (UN\ x:C. B\ x)) = (UN\ x:C. A \text{ Int } B\ x)$
 $!!A B C. ((UN\ x:C. A\ x) - B) = (UN\ x:C. A\ x - B)$
 $!!A B C. (A - (INT\ x:C. B\ x)) = (UN\ x:C. A - B\ x)$
 $!!A B. (UN\ y:A. UN\ x:y. B\ x) = (UN\ x: Union\ A. B\ x)$
 $!!A B C. (UN\ x:A. UN\ z: B(x). C\ z) = (UN\ z: UNION\ A\ B. C\ z)$
 $!!A B f. (UN\ a:A. B\ (f\ a)) = (UN\ x:f'A. B\ x)$
 $\langle proof \rangle$

lemma *INT-extend-simps*:

$!!A B C. (INT\ x:C. A\ x) \text{ Int } B = (if\ C=\{\} \text{ then } B \text{ else } (INT\ x:C. A\ x \text{ Int } B))$
 $!!A B C. A \text{ Int } (INT\ x:C. B\ x) = (if\ C=\{\} \text{ then } A \text{ else } (INT\ x:C. A \text{ Int } B\ x))$
 $!!A B C. (INT\ x:C. A\ x) - B = (if\ C=\{\} \text{ then } UNIV - B \text{ else } (INT\ x:C. A\ x - B))$
 $!!A B C. A - (UN\ x:C. B\ x) = (if\ C=\{\} \text{ then } A \text{ else } (INT\ x:C. A - B\ x))$
 $!!a B C. insert\ a\ (INT\ x:C. B\ x) = (INT\ x:C. insert\ a\ (B\ x))$
 $!!A B C. ((INT\ x:C. A\ x) \text{ Un } B) = (INT\ x:C. A\ x \text{ Un } B)$
 $!!A B C. A \text{ Un } (INT\ x:C. B\ x) = (INT\ x:C. A \text{ Un } B\ x)$
 $!!A B. (INT\ y:A. INT\ x:y. B\ x) = (INT\ x: Union\ A. B\ x)$
 $!!A B C. (INT\ x:A. INT\ z: B(x). C\ z) = (INT\ z: UNION\ A\ B. C\ z)$
 $!!A B f. (INT\ a:A. B\ (f\ a)) = (INT\ x:f'A. B\ x)$
 $\langle proof \rangle$

5.5.3 Monotonicity of various operations

lemma *image-mono*: $A \subseteq B \implies f'A \subseteq f'B$
 $\langle proof \rangle$

lemma *Pow-mono*: $A \subseteq B \implies Pow\ A \subseteq Pow\ B$
 $\langle proof \rangle$

lemma *Union-mono*: $A \subseteq B \implies \bigcup A \subseteq \bigcup B$
 $\langle proof \rangle$

lemma *Inter-anti-mono*: $B \subseteq A \implies \bigcap A \subseteq \bigcap B$
 $\langle proof \rangle$

lemma *UN-mono*:
 $A \subseteq B \implies (!!x. x \in A \implies f\ x \subseteq g\ x) \implies$
 $(\bigcup_{x \in A}. f\ x) \subseteq (\bigcup_{x \in B}. g\ x)$
 $\langle proof \rangle$

lemma *INT-anti-mono*:
 $B \subseteq A \implies (!!x. x \in A \implies f\ x \subseteq g\ x) \implies$
 $(\bigcap_{x \in A}. f\ x) \subseteq (\bigcap_{x \in A}. g\ x)$
 — The last inclusion is POSITIVE!
 $\langle proof \rangle$

lemma *insert-mono*: $C \subseteq D \implies \text{insert } a \ C \subseteq \text{insert } a \ D$
 $\langle \text{proof} \rangle$

lemma *Un-mono*: $A \subseteq C \implies B \subseteq D \implies A \cup B \subseteq C \cup D$
 $\langle \text{proof} \rangle$

lemma *Int-mono*: $A \subseteq C \implies B \subseteq D \implies A \cap B \subseteq C \cap D$
 $\langle \text{proof} \rangle$

lemma *Diff-mono*: $A \subseteq C \implies D \subseteq B \implies A - B \subseteq C - D$
 $\langle \text{proof} \rangle$

lemma *Compl-anti-mono*: $A \subseteq B \implies -B \subseteq -A$
 $\langle \text{proof} \rangle$

Monotonicity of implications.

lemma *in-mono*: $A \subseteq B \implies x \in A \longrightarrow x \in B$
 $\langle \text{proof} \rangle$

lemma *conj-mono*: $P1 \longrightarrow Q1 \implies P2 \longrightarrow Q2 \implies (P1 \ \& \ P2) \longrightarrow (Q1 \ \& \ Q2)$
 $\langle \text{proof} \rangle$

lemma *disj-mono*: $P1 \longrightarrow Q1 \implies P2 \longrightarrow Q2 \implies (P1 \mid P2) \longrightarrow (Q1 \mid Q2)$
 $\langle \text{proof} \rangle$

lemma *imp-mono*: $Q1 \longrightarrow P1 \implies P2 \longrightarrow Q2 \implies (P1 \longrightarrow P2) \longrightarrow (Q1 \longrightarrow Q2)$
 $\langle \text{proof} \rangle$

lemma *imp-refl*: $P \longrightarrow P$ $\langle \text{proof} \rangle$

lemma *ex-mono*: $(!!x. P \ x \longrightarrow Q \ x) \implies (EX \ x. P \ x) \longrightarrow (EX \ x. Q \ x)$
 $\langle \text{proof} \rangle$

lemma *all-mono*: $(!!x. P \ x \longrightarrow Q \ x) \implies (ALL \ x. P \ x) \longrightarrow (ALL \ x. Q \ x)$
 $\langle \text{proof} \rangle$

lemma *Collect-mono*: $(!!x. P \ x \longrightarrow Q \ x) \implies \text{Collect } P \subseteq \text{Collect } Q$
 $\langle \text{proof} \rangle$

lemma *Int-Collect-mono*:

$A \subseteq B \implies (!!x. x \in A \implies P \ x \longrightarrow Q \ x) \implies A \cap \text{Collect } P \subseteq B \cap \text{Collect } Q$
 $\langle \text{proof} \rangle$

lemmas *basic-monos* =
 $\text{subset-refl } \text{imp-refl } \text{disj-mono } \text{conj-mono}$

ex-mono Collect-mono in-mono

lemma *eq-to-mono*: $a = b \implies c = d \implies b \dashv\vdash d \implies a \dashv\vdash c$
 $\langle \text{proof} \rangle$

lemma *eq-to-mono2*: $a = b \implies c = d \implies \sim b \dashv\vdash \sim d \implies \sim a \dashv\vdash \sim c$
 $\langle \text{proof} \rangle$

5.6 Inverse image of a function

constdefs

vimage :: $('a \Rightarrow 'b) \Rightarrow 'b \text{ set} \Rightarrow 'a \text{ set}$ (**infixr** $-'$ 90)
 $[\text{code del}]: f -' B == \{x. f\ x : B\}$

5.6.1 Basic rules

lemma *vimage-eq* [*simp*]: $(a : f -' B) = (f\ a : B)$
 $\langle \text{proof} \rangle$

lemma *vimage-singleton-eq*: $(a : f -' \{b\}) = (f\ a = b)$
 $\langle \text{proof} \rangle$

lemma *vimageI* [*intro*]: $f\ a = b \implies b : B \implies a : f -' B$
 $\langle \text{proof} \rangle$

lemma *vimageI2*: $f\ a : A \implies a : f -' A$
 $\langle \text{proof} \rangle$

lemma *vimageE* [*elim!*]: $a : f -' B \implies (!x. f\ a = x \implies x : B \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *vimageD*: $a : f -' A \implies f\ a : A$
 $\langle \text{proof} \rangle$

5.6.2 Equations

lemma *vimage-empty* [*simp*]: $f -' \{\} = \{\}$
 $\langle \text{proof} \rangle$

lemma *vimage-Compl*: $f -' (-A) = -(f -' A)$
 $\langle \text{proof} \rangle$

lemma *vimage-Un* [*simp*]: $f -' (A \text{ Un } B) = (f -' A) \text{ Un } (f -' B)$
 $\langle \text{proof} \rangle$

lemma *vimage-Int* [*simp*]: $f -' (A \text{ Int } B) = (f -' A) \text{ Int } (f -' B)$
 $\langle \text{proof} \rangle$

lemma *vimage-Union*: $f -' (\text{Union } A) = (\text{UN } X:A. f -' X)$

$\langle proof \rangle$

lemma *vimage-UN*: $f - ' (UN\ x:A. B\ x) = (UN\ x:A. f - ' B\ x)$
 $\langle proof \rangle$

lemma *vimage-INT*: $f - ' (INT\ x:A. B\ x) = (INT\ x:A. f - ' B\ x)$
 $\langle proof \rangle$

lemma *vimage-Collect-eq* [simp]: $f - ' Collect\ P = \{y. P\ (f\ y)\}$
 $\langle proof \rangle$

lemma *vimage-Collect*: $(!!x. P\ (f\ x) = Q\ x) ==> f - ' (Collect\ P) = Collect\ Q$
 $\langle proof \rangle$

lemma *vimage-insert*: $f - ' (insert\ a\ B) = (f - '\{a\})\ Un\ (f - ' B)$
 — NOT suitable for rewriting because of the recurrence of $\{a\}$.
 $\langle proof \rangle$

lemma *vimage-Diff*: $f - ' (A - B) = (f - ' A) - (f - ' B)$
 $\langle proof \rangle$

lemma *vimage-UNIV* [simp]: $f - ' UNIV = UNIV$
 $\langle proof \rangle$

lemma *vimage-eq-UN*: $f - ' B = (UN\ y: B. f - '\{y\})$
 — NOT suitable for rewriting
 $\langle proof \rangle$

lemma *vimage-mono*: $A \subseteq B ==> f - ' A \subseteq f - ' B$
 — monotonicity
 $\langle proof \rangle$

lemma *vimage-image-eq* [noatp]: $f - ' (f - ' A) = \{y. EX\ x:A. f\ x = f\ y\}$
 $\langle proof \rangle$

lemma *image-vimage-subset*: $f - ' (f - ' A) <= A$
 $\langle proof \rangle$

lemma *image-vimage-eq* [simp]: $f - ' (f - ' A) = A\ Int\ range\ f$
 $\langle proof \rangle$

lemma *image-Int-subset*: $f - ' (A\ Int\ B) <= f - ' A\ Int\ f - ' B$
 $\langle proof \rangle$

lemma *image-diff-subset*: $f - ' A - f - ' B <= f - ' (A - B)$
 $\langle proof \rangle$

lemma *image-UN*: $(f - ' (UNION\ A\ B)) = (UN\ x:A. (f - ' (B\ x)))$
 $\langle proof \rangle$

5.7 Getting the Contents of a Singleton Set

definition *contents* :: 'a set \Rightarrow 'a **where**
 $[code\ del]:\ contents\ X = (THE\ x.\ X = \{x\})$

lemma *contents-eq* $[simp]:\ contents\ \{x\} = x$
 $\langle proof \rangle$

5.8 Transitivity rules for calculational reasoning

lemma *set-rev-mp*: $x:A \Rightarrow A \subseteq B \Rightarrow x:B$
 $\langle proof \rangle$

lemma *set-mp*: $A \subseteq B \Rightarrow x:A \Rightarrow x:B$
 $\langle proof \rangle$

lemmas *basic-trans-rules* $[trans] =$
order-trans-rules set-rev-mp set-mp

5.9 Least value operator

lemma *Least-mono*:
 $mono\ (f::'a::order \Rightarrow 'b::order) \Rightarrow EX\ x:S.\ ALL\ y:S.\ x \leq y$
 $\Rightarrow (LEAST\ y.\ y : f\ 'S) = f\ (LEAST\ x.\ x : S)$
 — Courtesy of Stephan Merz
 $\langle proof \rangle$

5.10 Rudimentary code generation

lemma *empty-code* $[code]:\ \{\} \longleftrightarrow False$
 $\langle proof \rangle$

lemma *UNIV-code* $[code]:\ UNIV \longleftrightarrow True$
 $\langle proof \rangle$

lemma *insert-code* $[code]:\ insert\ y\ A\ x \longleftrightarrow y = x \vee A\ x$
 $\langle proof \rangle$

lemma *inter-code* $[code]:\ (A \cap B)\ x \longleftrightarrow A\ x \wedge B\ x$
 $\langle proof \rangle$

lemma *union-code* $[code]:\ (A \cup B)\ x \longleftrightarrow A\ x \vee B\ x$
 $\langle proof \rangle$

lemma *image-code* $[code]:\ (f\ -'A)\ x = A\ (f\ x)$
 $\langle proof \rangle$

5.11 Complete lattices

notation

```

less-eq (infix  $\sqsubseteq$  50) and
less (infix  $\sqsubset$  50) and
inf (infixl  $\sqcap$  70) and
sup (infixl  $\sqcup$  65)

class complete-lattice = lattice + bot + top +
  fixes Inf :: 'a set  $\Rightarrow$  'a ( $\sqcap$  - [900] 900)
  and Sup :: 'a set  $\Rightarrow$  'a ( $\sqcup$  - [900] 900)
  assumes Inf-lower:  $x \in A \Rightarrow \sqcap A \sqsubseteq x$ 
  and Inf-greatest:  $(\bigwedge x. x \in A \Rightarrow z \sqsubseteq x) \Rightarrow z \sqsubseteq \sqcap A$ 
  assumes Sup-upper:  $x \in A \Rightarrow x \sqsubseteq \sqcup A$ 
  and Sup-least:  $(\bigwedge x. x \in A \Rightarrow x \sqsubseteq z) \Rightarrow \sqcup A \sqsubseteq z$ 
begin

lemma Inf-Sup:  $\sqcap A = \sqcup \{b. \forall a \in A. b \leq a\}$ 
  <proof>

lemma Sup-Inf:  $\sqcup A = \sqcap \{b. \forall a \in A. a \leq b\}$ 
  <proof>

lemma Inf-Univ:  $\sqcap UNIV = \sqcup \{\}$ 
  <proof>

lemma Sup-Univ:  $\sqcup UNIV = \sqcap \{\}$ 
  <proof>

lemma Inf-insert:  $\sqcap \text{insert } a \ A = a \sqcap \sqcap A$ 
  <proof>

lemma Sup-insert:  $\sqcup \text{insert } a \ A = a \sqcup \sqcup A$ 
  <proof>

lemma Inf-singleton [simp]:
   $\sqcap \{a\} = a$ 
  <proof>

lemma Sup-singleton [simp]:
   $\sqcup \{a\} = a$ 
  <proof>

lemma Inf-insert-simp:
   $\sqcap \text{insert } a \ A = (\text{if } A = \{\} \text{ then } a \text{ else } a \sqcap \sqcap A)$ 
  <proof>

lemma Sup-insert-simp:
   $\sqcup \text{insert } a \ A = (\text{if } A = \{\} \text{ then } a \text{ else } a \sqcup \sqcup A)$ 
  <proof>

lemma Inf-binary:

```

$\sqcap \{a, b\} = a \sqcap b$
 $\langle proof \rangle$

lemma *Sup-binary*:
 $\sqcup \{a, b\} = a \sqcup b$
 $\langle proof \rangle$

lemma *bot-def*:
 $bot = \sqcup \{\}$
 $\langle proof \rangle$

lemma *top-def*:
 $top = \sqcap \{\}$
 $\langle proof \rangle$

lemma *sup-bot [simp]*:
 $x \sqcup bot = x$
 $\langle proof \rangle$

lemma *inf-top [simp]*:
 $x \sqcap top = x$
 $\langle proof \rangle$

definition *SUPR* :: $'b \text{ set} \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a$ **where**
 $SUPR\ A\ f == \sqcup (f \text{ ` } A)$

definition *INFI* :: $'b \text{ set} \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a$ **where**
 $INFI\ A\ f == \sqcap (f \text{ ` } A)$

end

syntax

-*SUP1* :: $pttrns \Rightarrow 'b \Rightarrow 'b$ $((\exists SUP \text{ -./ -}) [0, 10] 10)$
-*SUP* :: $pttrn \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b$ $((\exists SUP \text{ :-./ -}) [0, 10] 10)$
-*INF1* :: $pttrns \Rightarrow 'b \Rightarrow 'b$ $((\exists INF \text{ -./ -}) [0, 10] 10)$
-*INF* :: $pttrn \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b$ $((\exists INF \text{ :-./ -}) [0, 10] 10)$

translations

$SUP\ x\ y. B == SUP\ x. SUP\ y. B$
 $SUP\ x. B == CONST\ SUPR\ CONST\ UNIV\ (\%x. B)$
 $SUP\ x. B == SUP\ x:CONST\ UNIV. B$
 $SUP\ x:A. B == CONST\ SUPR\ A\ (\%x. B)$
 $INF\ x\ y. B == INF\ x. INF\ y. B$
 $INF\ x. B == CONST\ INFI\ CONST\ UNIV\ (\%x. B)$
 $INF\ x. B == INF\ x:CONST\ UNIV. B$
 $INF\ x:A. B == CONST\ INFI\ A\ (\%x. B)$

$\langle ML \rangle$

context *complete-lattice*

begin

lemma *le-SUPI*: $i : A \implies M\ i \leq (SUP\ i:A.\ M\ i)$
 $\langle proof \rangle$

lemma *SUP-leI*: $(\bigwedge i.\ i : A \implies M\ i \leq u) \implies (SUP\ i:A.\ M\ i) \leq u$
 $\langle proof \rangle$

lemma *INF-leI*: $i : A \implies (INF\ i:A.\ M\ i) \leq M\ i$
 $\langle proof \rangle$

lemma *le-INF*: $(\bigwedge i.\ i : A \implies u \leq M\ i) \implies u \leq (INF\ i:A.\ M\ i)$
 $\langle proof \rangle$

lemma *SUP-const[simp]*: $A \neq \{\} \implies (SUP\ i:A.\ M) = M$
 $\langle proof \rangle$

lemma *INF-const[simp]*: $A \neq \{\} \implies (INF\ i:A.\ M) = M$
 $\langle proof \rangle$

end

5.12 Bool as complete lattice

instantiation *bool* :: *complete-lattice*

begin

definition

Inf-bool-def: $\bigcap A \longleftrightarrow (\forall x \in A.\ x)$

definition

Sup-bool-def: $\bigcup A \longleftrightarrow (\exists x \in A.\ x)$

instance

$\langle proof \rangle$

end

lemma *Inf-empty-bool* [simp]:

$\bigcap \{\}$
 $\langle proof \rangle$

lemma *not-Sup-empty-bool* [simp]:

$\neg \bigcup \{\}$
 $\langle proof \rangle$

5.13 Fun as complete lattice

instantiation *fun* :: (*type*, *complete-lattice*) *complete-lattice*
begin

definition

Inf-fun-def [*code del*]: $\sqcap A = (\lambda x. \sqcap \{y. \exists f \in A. y = f x\})$

definition

Sup-fun-def [*code del*]: $\sqcup A = (\lambda x. \sqcup \{y. \exists f \in A. y = f x\})$

instance

<proof>

end

lemma *Inf-empty-fun*:

$\sqcap \{\} = (\lambda -. \sqcap \{\})$

<proof>

lemma *Sup-empty-fun*:

$\sqcup \{\} = (\lambda -. \sqcup \{\})$

<proof>

5.14 Set as lattice

lemma *inf-set-eq*: $A \sqcap B = A \cap B$

<proof>

lemma *sup-set-eq*: $A \sqcup B = A \cup B$

<proof>

lemma *mono-Int*: $\text{mono } f \implies f (A \cap B) \subseteq f A \cap f B$

<proof>

lemma *mono-Un*: $\text{mono } f \implies f A \cup f B \subseteq f (A \cup B)$

<proof>

lemma *Inf-set-eq*: $\sqcap S = \bigcap S$

<proof>

lemma *Sup-set-eq*: $\sqcup S = \bigcup S$

<proof>

lemma *top-set-eq*: $\text{top} = \text{UNIV}$

<proof>

lemma *bot-set-eq*: $\text{bot} = \{\}$

<proof>

no-notation

less-eq (**infix** \sqsubseteq 50) **and**
less (**infix** \sqsubset 50) **and**
inf (**infixl** \sqcap 70) **and**
sup (**infixl** \sqcup 65) **and**
Inf (**[** \sqcap - [900] 900) **and**
Sup (**[** \sqcup - [900] 900)

5.15 Misc theorem and ML bindings

lemmas *equalityI* = *subset-antisym*

lemmas *mem-simps* =

insert-iff empty-iff Un-iff Int-iff Compl-iff Diff-iff
mem-Collect-eq UN-iff Union-iff INT-iff Inter-iff
 — Each of these has ALREADY been added [*simp*] above.

$\langle ML \rangle$

end

6 Typedef: HOL type definitions

theory *Typedef*

imports *Set*

uses

(*Tools/type-def-package.ML*)
 (*Tools/typecopy-package.ML*)
 (*Tools/type-def-codegen.ML*)

begin

$\langle ML \rangle$

locale *type-definition* =

fixes *Rep* **and** *Abs* **and** *A*

assumes *Rep*: *Rep* *x* \in *A*

and *Rep-inverse*: *Abs* (*Rep* *x*) = *x*

and *Abs-inverse*: *y* \in *A* \implies *Rep* (*Abs* *y*) = *y*

— This will be axiomatized for each typedef!

begin

lemma *Rep-inject*:

(*Rep* *x* = *Rep* *y*) = (*x* = *y*)

$\langle proof \rangle$

lemma *Abs-inject*:

assumes *x*: *x* \in *A* **and** *y*: *y* \in *A*

shows (*Abs* *x* = *Abs* *y*) = (*x* = *y*)

$\langle proof \rangle$

```

lemma Rep-cases [cases set]:
  assumes y:  $y \in A$ 
    and hyp:  $!!x. y = \text{Rep } x \implies P$ 
  shows  $P$ 
 $\langle \text{proof} \rangle$ 

lemma Abs-cases [cases type]:
  assumes r:  $!!y. x = \text{Abs } y \implies y \in A \implies P$ 
  shows  $P$ 
 $\langle \text{proof} \rangle$ 

lemma Rep-induct [induct set]:
  assumes y:  $y \in A$ 
    and hyp:  $!!x. P (\text{Rep } x)$ 
  shows  $P y$ 
 $\langle \text{proof} \rangle$ 

lemma Abs-induct [induct type]:
  assumes r:  $!!y. y \in A \implies P (\text{Abs } y)$ 
  shows  $P x$ 
 $\langle \text{proof} \rangle$ 

lemma Rep-range:  $\text{range Rep} = A$ 
 $\langle \text{proof} \rangle$ 

lemma Abs-image:  $\text{Abs } ` A = \text{UNIV}$ 
 $\langle \text{proof} \rangle$ 

end

 $\langle \text{ML} \rangle$ 

end

```

7 Fun: Notions about functions

```

theory Fun
imports Set
begin

```

As a simplification rule, it replaces all function equalities by first-order equalities.

```

lemma expand-fun-eq:  $f = g \longleftrightarrow (\forall x. f x = g x)$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma apply-inverse:
   $f x = u \implies (\bigwedge x. P x \implies g (f x) = x) \implies P x \implies x = g u$ 

```

$\langle proof \rangle$

7.1 The Identity Function id

definition

$id :: 'a \Rightarrow 'a$

where

$id = (\lambda x. x)$

lemma *id-apply* [simp]: $id\ x = x$

$\langle proof \rangle$

lemma *image-ident* [simp]: $(\%x. x)\ 'Y = Y$

$\langle proof \rangle$

lemma *image-id* [simp]: $id\ 'Y = Y$

$\langle proof \rangle$

lemma *vimage-ident* [simp]: $(\%x. x)\ -'Y = Y$

$\langle proof \rangle$

lemma *vimage-id* [simp]: $id\ -'A = A$

$\langle proof \rangle$

7.2 The Composition Operator $f \circ g$

definition

$comp :: ('b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'c$ (**infixl** \circ 55)

where

$f \circ g = (\lambda x. f\ (g\ x))$

notation (*xsymbols*)

$comp$ (**infixl** \circ 55)

notation (*HTML output*)

$comp$ (**infixl** \circ 55)

compatibility

lemmas *o-def* = *comp-def*

lemma *o-apply* [simp]: $(f \circ g)\ x = f\ (g\ x)$

$\langle proof \rangle$

lemma *o-assoc*: $f \circ (g \circ h) = f \circ g \circ h$

$\langle proof \rangle$

lemma *id-o* [simp]: $id \circ g = g$

$\langle proof \rangle$

lemma *o-id* [*simp*]: $f \circ id = f$
 $\langle proof \rangle$

lemma *image-compose*: $(f \circ g) \circ r = f \circ (g \circ r)$
 $\langle proof \rangle$

lemma *UN-o*: $UNION A (g \circ f) = UNION (f \circ A) g$
 $\langle proof \rangle$

7.3 The Forward Composition Operator *fcomp*

definition

$fcomp :: ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'a \Rightarrow 'c$ (**infixl** *o>* 60)

where

$f \circ> g = (\lambda x. g (f x))$

lemma *fcomp-apply*: $(f \circ> g) x = g (f x)$
 $\langle proof \rangle$

lemma *fcomp-assoc*: $(f \circ> g) \circ> h = f \circ> (g \circ> h)$
 $\langle proof \rangle$

lemma *id-fcomp* [*simp*]: $id \circ> g = g$
 $\langle proof \rangle$

lemma *fcomp-id* [*simp*]: $f \circ> id = f$
 $\langle proof \rangle$

no-notation *fcomp* (**infixl** *o>* 60)

7.4 Injectivity and Surjectivity

constdefs

$inj-on :: ['a \Rightarrow 'b, 'a \text{ set}] \Rightarrow bool$ — injective
 $inj-on f A == ! x:A. ! y:A. f(x)=f(y) \longrightarrow x=y$

A common special case: functions injective over the entire domain type.

abbreviation

$inj f == inj-on f UNIV$

definition

$bij-betw :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow bool$ **where** — bijective
 $[code del]: bij-betw f A B \longleftrightarrow inj-on f A \ \& \ f \circ A = B$

constdefs

$surj :: ('a \Rightarrow 'b) \Rightarrow bool$
 $surj f == ! y. ? x. y=f(x)$

$bij :: ('a \Rightarrow 'b) \Rightarrow bool$

```

lemma injI:
  assumes  $\bigwedge x y. f\ x = f\ y \implies x = y$ 
  shows  $\text{inj } f$ 
   $\langle \text{proof} \rangle$ 

```

lemma *datatype-injI*:

$$(! x. ALL y. f(x) = f(y) \longrightarrow x=y) \implies inj(f)$$
<proof>

lemma *injD*: $[\text{inj}(f); f(x) = f(y)] \implies x = y$
<proof>

lemma *inj-on-id[simp]: inj-on id A*
⟨proof⟩

lemma *surj-id[simp]: surj id*
<proof>

lemma *inj-onI*:

$$(! x y. [! x:A; y:A; f(x) = f(y)] ==> x=y) ==> \text{inj-on } f \ A$$

<proof>

lemma *inj-onD*: $[\mid \text{inj-on } f \ A; \ f(x)=f(y); \ x:A; \ y:A \mid] \implies x=y$
 $\langle \text{proof} \rangle$

lemma *comp-inj-on*:

$$[[\text{inj-on } f \ A; \ \text{inj-on } g \ (f \cdot A)]] ==> \text{inj-on } (g \circ f) \ A$$

$\langle proof \rangle$

lemma *inj-on-imageI*: $inj\text{-}on\ (g\ o\ f)\ A \implies inj\text{-}on\ g\ (f\ 'A)$
 $\langle proof \rangle$

lemma *inj-on-image-iff*: $\llbracket ALL\ x:A.\ ALL\ y:A.\ (g(f\ x) = g(f\ y)) = (g\ x = g\ y);$
 $inj\text{-}on\ f\ A \rrbracket \implies inj\text{-}on\ g\ (f\ 'A) = inj\text{-}on\ g\ A$
 $\langle proof \rangle$

lemma *inj-on-contrad*: $\llbracket inj\text{-}on\ f\ A; \sim x=y; x:A; y:A \rrbracket \implies \sim f(x)=f(y)$
 $\langle proof \rangle$

lemma *inj-singleton*: $inj\ (\%s.\ \{s\})$
 $\langle proof \rangle$

lemma *inj-on-empty[iff]*: $inj\text{-}on\ f\ \{\}$
 $\langle proof \rangle$

lemma *subset-inj-on*: $\llbracket inj\text{-}on\ f\ B; A \leq B \rrbracket \implies inj\text{-}on\ f\ A$
 $\langle proof \rangle$

lemma *inj-on-Un*:
 $inj\text{-}on\ f\ (A\ Un\ B) =$
 $(inj\text{-}on\ f\ A \ \&\ inj\text{-}on\ f\ B \ \&\ f'(A-B)\ Int\ f'(B-A) = \{\})$
 $\langle proof \rangle$

lemma *inj-on-insert[iff]*:
 $inj\text{-}on\ f\ (insert\ a\ A) = (inj\text{-}on\ f\ A \ \&\ f\ a\ \sim: f'(A-\{a\}))$
 $\langle proof \rangle$

lemma *inj-on-diff*: $inj\text{-}on\ f\ A \implies inj\text{-}on\ f\ (A-B)$
 $\langle proof \rangle$

lemma *surjI*: $(!!\ x.\ g(f\ x) = x) \implies surj\ g$
 $\langle proof \rangle$

lemma *surj-range*: $surj\ f \implies range\ f = UNIV$
 $\langle proof \rangle$

lemma *surjD*: $surj\ f \implies EX\ x.\ y = f\ x$
 $\langle proof \rangle$

lemma *surjE*: $surj\ f \implies (!!x.\ y = f\ x \implies C) \implies C$
 $\langle proof \rangle$

lemma *comp-surj*: $\llbracket surj\ f; surj\ g \rrbracket \implies surj\ (g\ o\ f)$
 $\langle proof \rangle$

lemma *bijI*: $\llbracket inj\ f; surj\ f \rrbracket \implies bij\ f$

$\langle proof \rangle$

lemma *bij-is-inj*: $bij\ f ==> inj\ f$
 $\langle proof \rangle$

lemma *bij-is-surj*: $bij\ f ==> surj\ f$
 $\langle proof \rangle$

lemma *bij-betw-imp-inj-on*: $bij\ betw\ f\ A\ B \implies inj\ on\ f\ A$
 $\langle proof \rangle$

lemma *bij-betw-inv*: **assumes** $bij\ betw\ f\ A\ B$ **shows** $EX\ g. bij\ betw\ g\ B\ A$
 $\langle proof \rangle$

lemma *surj-image-vimage-eq*: $surj\ f ==> f^{-1}\ (f^{-1}\ A) = A$
 $\langle proof \rangle$

lemma *inj-vimage-image-eq*: $inj\ f ==> f^{-1}\ (f^{-1}\ A) = A$
 $\langle proof \rangle$

lemma *vimage-subsetD*: $surj\ f ==> f^{-1}\ B \leq A ==> B \leq f^{-1}\ A$
 $\langle proof \rangle$

lemma *vimage-subsetI*: $inj\ f ==> B \leq f^{-1}\ A ==> f^{-1}\ B \leq A$
 $\langle proof \rangle$

lemma *vimage-subset-eq*: $bij\ f ==> (f^{-1}\ B \leq A) = (B \leq f^{-1}\ A)$
 $\langle proof \rangle$

lemma *inj-on-image-Int*:
 $[| inj\ on\ f\ C; A \leq C; B \leq C |] ==> f^{-1}(A \ Int\ B) = f^{-1}A \ Int\ f^{-1}B$
 $\langle proof \rangle$

lemma *inj-on-image-set-diff*:
 $[| inj\ on\ f\ C; A \leq C; B \leq C |] ==> f^{-1}(A - B) = f^{-1}A - f^{-1}B$
 $\langle proof \rangle$

lemma *image-Int*: $inj\ f ==> f^{-1}(A \ Int\ B) = f^{-1}A \ Int\ f^{-1}B$
 $\langle proof \rangle$

lemma *image-set-diff*: $inj\ f ==> f^{-1}(A - B) = f^{-1}A - f^{-1}B$
 $\langle proof \rangle$

lemma *inj-image-mem-iff*: $inj\ f ==> (f\ a : f^{-1}A) = (a : A)$
 $\langle proof \rangle$

lemma *inj-image-subset-iff*: $inj\ f ==> (f^{-1}A \leq f^{-1}B) = (A \leq B)$
 $\langle proof \rangle$

lemma *inj-image-eq-iff*: $\text{inj } f \implies (f^{\cdot}A = f^{\cdot}B) = (A = B)$
 $\langle \text{proof} \rangle$

lemma *image-INT*:
 $[\text{inj-on } f \ C; \ \text{ALL } x:A. \ B \ x \leq C; \ j:A] \implies f^{\cdot} (\text{INTER } A \ B) = (\text{INT } x:A. \ f^{\cdot} B \ x)$
 $\langle \text{proof} \rangle$

lemma *bij-image-INT*: $\text{bij } f \implies f^{\cdot} (\text{INTER } A \ B) = (\text{INT } x:A. \ f^{\cdot} B \ x)$
 $\langle \text{proof} \rangle$

lemma *surj-Compl-image-subset*: $\text{surj } f \implies \neg(f^{\cdot}A) \leq f^{\cdot}(\neg A)$
 $\langle \text{proof} \rangle$

lemma *inj-image-Compl-subset*: $\text{inj } f \implies f^{\cdot}(\neg A) \leq \neg(f^{\cdot}A)$
 $\langle \text{proof} \rangle$

lemma *bij-image-Compl-eq*: $\text{bij } f \implies f^{\cdot}(\neg A) = \neg(f^{\cdot}A)$
 $\langle \text{proof} \rangle$

7.5 Function Updating

constdefs

fun-upd :: $(\text{'a} \Rightarrow \text{'b}) \Rightarrow \text{'a} \Rightarrow \text{'b} \Rightarrow (\text{'a} \Rightarrow \text{'b})$
fun-upd $f \ a \ b == \% x. \text{ if } x=a \text{ then } b \text{ else } f \ x$

nonterminals

updbinds updbind

syntax

-updbind :: $[\text{'a}, \text{'a}] \Rightarrow \text{updbind} \quad ((2- := / -))$
 $\quad \quad \quad :: \text{updbind} \Rightarrow \text{updbinds} \quad (-)$
-updbinds:: $[\text{updbind}, \text{updbinds}] \Rightarrow \text{updbinds} \ (-, / -)$
-Update :: $[\text{'a}, \text{updbinds}] \Rightarrow \text{'a} \quad (-/'((-)) [1000,0] \ 900)$

translations

-Update $f \ (-\text{updbinds } b \ bs) == -\text{Update } (-\text{Update } f \ b) \ bs$
 $f(x:=y) \quad \quad \quad == \text{fun-upd } f \ x \ y$

lemma *fun-upd-idem-iff*: $(f(x:=y) = f) = (f \ x = y)$
 $\langle \text{proof} \rangle$

lemmas *fun-upd-idem* = *fun-upd-idem-iff* [THEN iffD2, standard]

lemmas *fun-upd-triv* = *refl* [*THEN fun-upd-idem*]
declare *fun-upd-triv* [*iff*]

lemma *fun-upd-apply* [*simp*]: $(f(x:=y))z = (\text{if } z=x \text{ then } y \text{ else } f\ z)$
 $\langle \text{proof} \rangle$

lemma *fun-upd-same*: $(f(x:=y))\ x = y$
 $\langle \text{proof} \rangle$

lemma *fun-upd-other*: $z \sim x \implies (f(x:=y))\ z = f\ z$
 $\langle \text{proof} \rangle$

lemma *fun-upd-upd* [*simp*]: $f(x:=y, x:=z) = f(x:=z)$
 $\langle \text{proof} \rangle$

lemma *fun-upd-twist*: $a \sim c \implies (m(a:=b))(c:=d) = (m(c:=d))(a:=b)$
 $\langle \text{proof} \rangle$

lemma *inj-on-fun-updI*: $\llbracket \text{inj-on } f\ A; y \notin f'A \rrbracket \implies \text{inj-on } (f(x:=y))\ A$
 $\langle \text{proof} \rangle$

lemma *fun-upd-image*:
 $f(x:=y)\ 'A = (\text{if } x \in A \text{ then insert } y\ (f\ ' (A - \{x\})) \text{ else } f\ 'A)$
 $\langle \text{proof} \rangle$

7.6 override-on

definition

override-on :: $('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a\ \text{set} \Rightarrow 'a \Rightarrow 'b$

where

override-on $f\ g\ A = (\lambda a. \text{if } a \in A \text{ then } g\ a \text{ else } f\ a)$

lemma *override-on-emptyset*[*simp*]: *override-on* $f\ g\ \{\} = f$
 $\langle \text{proof} \rangle$

lemma *override-on-apply-notin*[*simp*]: $a \sim: A \implies (\text{override-on } f\ g\ A)\ a = f\ a$
 $\langle \text{proof} \rangle$

lemma *override-on-apply-in*[*simp*]: $a : A \implies (\text{override-on } f\ g\ A)\ a = g\ a$
 $\langle \text{proof} \rangle$

7.7 swap

definition

swap :: $'a \Rightarrow 'a \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)$

where

swap $a\ b\ f = f\ (a := f\ b, b := f\ a)$

lemma *swap-self*: *swap* $a\ a\ f = f$

$\langle \text{proof} \rangle$

lemma *swap-commute*: $\text{swap } a \ b \ f = \text{swap } b \ a \ f$

$\langle \text{proof} \rangle$

lemma *swap-nilpotent* [simp]: $\text{swap } a \ b \ (\text{swap } a \ b \ f) = f$

$\langle \text{proof} \rangle$

lemma *inj-on-imp-inj-on-swap*:

$[[\text{inj-on } f \ A; a \in A; b \in A]] \implies \text{inj-on } (\text{swap } a \ b \ f) \ A$

$\langle \text{proof} \rangle$

lemma *inj-on-swap-iff* [simp]:

assumes $A: a \in A \ b \in A$ **shows** $\text{inj-on } (\text{swap } a \ b \ f) \ A = \text{inj-on } f \ A$

$\langle \text{proof} \rangle$

lemma *surj-imp-surj-swap*: $\text{surj } f \implies \text{surj } (\text{swap } a \ b \ f)$

$\langle \text{proof} \rangle$

lemma *surj-swap-iff* [simp]: $\text{surj } (\text{swap } a \ b \ f) = \text{surj } f$

$\langle \text{proof} \rangle$

lemma *bij-swap-iff*: $\text{bij } (\text{swap } a \ b \ f) = \text{bij } f$

$\langle \text{proof} \rangle$

hide (**open**) *const swap*

7.8 Proof tool setup

simplifies terms of the form $f(\dots, x := y, \dots, x := z, \dots)$ to $f(\dots, x := z, \dots)$

$\langle \text{ML} \rangle$

7.9 Code generator setup

types-code

fun $((- \rightarrow / -))$

attach (*term-of*) $\langle\langle$

fun term-of-fun-type - aT - bT - = *Free* ($\langle \text{function} \rangle$, $aT \rightarrow bT$);

$\rangle\rangle$

attach (*test*) $\langle\langle$

fun gen-fun-type aF aT bG bT i =

let

val $\text{tab} = \text{ref } []$;

fun $\text{mk-upd } (x, (-, y)) \ t = \text{Const } (\text{Fun.fun-upd},$

$(aT \rightarrow bT) \rightarrow aT \rightarrow bT \rightarrow aT \rightarrow bT) \ \$ \ t \ \$ \ aF \ x \ \$ \ y \ ()$

in

$(\text{fn } x \Rightarrow$

$\text{case } AList.lookup \ op = (!\text{tab}) \ x \ \text{of}$

$NONE \Rightarrow$

```

      let val p as (y, -) = bG i
      in (tab := (x, p) :: !tab; y) end
    | SOME (y, -) => y,
    fn () => Basics.fold mk-upd (!tab) (Const (HOL.undefined, aT --> bT)))
  end;
>>

```

```

code-const op ◦
  (SML infixl 5 o)
  (Haskell infixr 9 .)

```

```

code-const id
  (Haskell id)

```

```

end

```

8 Sum-Type: The Disjoint Sum of Two Types

```

theory Sum-Type
imports Typedef Fun
begin

```

The representations of the two injections

```

constdefs
  Inl-Rep :: ['a, 'a, 'b, bool] => bool
  Inl-Rep == (%a. %x y p. x=a & p)

  Inr-Rep :: ['b, 'a, 'b, bool] => bool
  Inr-Rep == (%b. %x y p. y=b & ~p)

```

```

global

```

```

typedef (Sum)
  ('a, 'b) + (infixr + 10)
  = {f. (? a. f = Inl-Rep(a::'a)) | (? b. f = Inr-Rep(b::'b))}
  ⟨proof⟩

```

```

local

```

abstract constants and syntax

```

constdefs
  Inl :: 'a => 'a + 'b
  Inl == (%a. Abs-Sum(Inl-Rep(a)))

  Inr :: 'b => 'a + 'b
  Inr == (%b. Abs-Sum(Inr-Rep(b)))

```


Plus :: [*'a set, 'b set*] ==> (*'a + 'b set*) (**infixr** <+> 65)
A <+> B == (*Inl 'A Un (Inr 'B)*)
 — disjoint sum for sets; the operator + is overloaded with wrong type!

Part :: [*'a set, 'b ==> 'a*] ==> *'a set*
Part A h == *A Int {x. ? z. x = h(z)}*
 — for selecting out the components of a mutually recursive definition

lemma *Inl-RepI*: *Inl-Rep(a) : Sum*
<proof>

lemma *Inr-RepI*: *Inr-Rep(b) : Sum*
<proof>

lemma *inj-on-Abs-Sum*: *inj-on Abs-Sum Sum*
<proof>

8.1 Freeness Properties for *Inl* and *Inr*

Distinctness

lemma *Inl-Rep-not-Inr-Rep*: *Inl-Rep(a) ~ = Inr-Rep(b)*
<proof>

lemma *Inl-not-Inr [iff]*: *Inl(a) ~ = Inr(b)*
<proof>

lemmas *Inr-not-Inl* = *Inl-not-Inr [THEN not-sym, standard]*
declare *Inr-not-Inl [iff]*

lemmas *Inl-neq-Inr* = *Inl-not-Inr [THEN notE, standard]*
lemmas *Inr-neq-Inl* = *sym [THEN Inl-neq-Inr, standard]*

Injectiveness

lemma *Inl-Rep-inject*: *Inl-Rep(a) = Inl-Rep(c) ==> a=c*
<proof>

lemma *Inr-Rep-inject*: *Inr-Rep(b) = Inr-Rep(d) ==> b=d*
<proof>

lemma *inj-Inl [simp]*: *inj-on Inl A*
<proof>

lemmas *Inl-inject* = *inj-Inl [THEN injD, standard]*

lemma *inj-Inr* [*simp*]: *inj-on Inr A*
 $\langle proof \rangle$

lemmas *Inr-inject* = *inj-Inr* [*THEN injD, standard*]

lemma *Inl-eq* [*iff*]: $(Inl(x)=Inl(y)) = (x=y)$
 $\langle proof \rangle$

lemma *Inr-eq* [*iff*]: $(Inr(x)=Inr(y)) = (x=y)$
 $\langle proof \rangle$

8.2 The Disjoint Sum of Sets

lemma *InlI* [*intro!*]: $a : A ==> Inl(a) : A <+> B$
 $\langle proof \rangle$

lemma *InrI* [*intro!*]: $b : B ==> Inr(b) : A <+> B$
 $\langle proof \rangle$

lemma *PlusE* [*elim!*]:

$$\begin{aligned} & [[u : A <+> B; \\ & \quad !!x. [[x:A; u=Inl(x)]] ==> P; \\ & \quad !!y. [[y:B; u=Inr(y)]] ==> P \\ &]] ==> P \end{aligned}$$

 $\langle proof \rangle$

Exhaustion rule for sums, a degenerate form of induction

lemma *sumE*:

$$\begin{aligned} & [[!!x::'a. s = Inl(x) ==> P; !!y::'b. s = Inr(y) ==> P \\ &]] ==> P \end{aligned}$$

 $\langle proof \rangle$

lemma *UNIV-Plus-UNIV* [*simp*]: $UNIV <+> UNIV = UNIV$
 $\langle proof \rangle$

8.3 The Part Primitive

lemma *Part-eqI* [*intro*]: $[[a : A; a=h(b)]] ==> a : Part A h$
 $\langle proof \rangle$

lemmas *PartI* = *Part-eqI* [*OF - refl, standard*]

lemma *PartE* [*elim!*]: $[[a : Part A h; !!z. [[a : A; a=h(z)]] ==> P]] ==> P$
 $\langle proof \rangle$

lemma *Part-subset*: $\text{Part } A \ h \leq A$

<proof>

lemma *Part-mono*: $A \leq B \implies \text{Part } A \ h \leq \text{Part } B \ h$

<proof>

lemmas *basic-monos* = *basic-monos* *Part-mono*

lemma *PartD1*: $a : \text{Part } A \ h \implies a : A$

<proof>

lemma *Part-id*: $\text{Part } A \ (\%x. x) = A$

<proof>

lemma *Part-Int*: $\text{Part } (A \ \text{Int } B) \ h = (\text{Part } A \ h) \ \text{Int } (\text{Part } B \ h)$

<proof>

lemma *Part-Collect*: $\text{Part } (A \ \text{Int } \{x. P \ x\}) \ h = (\text{Part } A \ h) \ \text{Int } \{x. P \ x\}$

<proof>

<ML>

end

9 Inductive: Knaster-Tarski Fixpoint Theorem and inductive definitions

theory *Inductive*

imports *Lattices Sum-Type*

uses

(Tools/inductive-package.ML)

Tools/dseq.ML

(Tools/inductive-codegen.ML)

(Tools/datatype-aux.ML)

(Tools/datatype-prop.ML)

(Tools/datatype-rep-proofs.ML)

(Tools/datatype-abs-proofs.ML)

(Tools/datatype-case.ML)

(Tools/datatype-package.ML)

(Tools/old-primrec-package.ML)

(Tools/primrec-package.ML)

(Tools/datatype-codegen.ML)

begin

9.1 Least and greatest fixed points

context *complete-lattice*

begin

definition

$lfp :: ('a \Rightarrow 'a) \Rightarrow 'a$ **where**
 $lfp\ f = Inf\ \{u. f\ u \leq u\}$ — least fixed point

definition

$gfp :: ('a \Rightarrow 'a) \Rightarrow 'a$ **where**
 $gfp\ f = Sup\ \{u. u \leq f\ u\}$ — greatest fixed point

9.2 Proof of Knaster-Tarski Theorem using *lfp*

lfp *f* is the least upper bound of the set $\{u. f\ u \leq u\}$

lemma *lfp-lowerbound*: $f\ A \leq A \implies lfp\ f \leq A$
 $\langle proof \rangle$

lemma *lfp-greatest*: $(!!u. f\ u \leq u \implies A \leq u) \implies A \leq lfp\ f$
 $\langle proof \rangle$

end

lemma *lfp-lemma2*: $mono\ f \implies f\ (lfp\ f) \leq lfp\ f$
 $\langle proof \rangle$

lemma *lfp-lemma3*: $mono\ f \implies lfp\ f \leq f\ (lfp\ f)$
 $\langle proof \rangle$

lemma *lfp-unfold*: $mono\ f \implies lfp\ f = f\ (lfp\ f)$
 $\langle proof \rangle$

lemma *lfp-const*: $lfp\ (\lambda x. t) = t$
 $\langle proof \rangle$

9.3 General induction rules for least fixed points

theorem *lfp-induct*:

assumes *mono*: $mono\ f$ **and** *ind*: $f\ (inf\ (lfp\ f)\ P) \leq P$

shows $lfp\ f \leq P$

$\langle proof \rangle$

lemma *lfp-induct-set*:

assumes *lfp*: $a: lfp(f)$

and *mono*: $mono(f)$

and *indhyp*: $!!x. [\mid x: f(lfp(f))\ Int\ \{x. P(x)\}] \implies P(x)$

shows $P(a)$

$\langle proof \rangle$

lemma *lfp-ordinal-induct*:
 fixes $f :: 'a::\text{complete-lattice} \Rightarrow 'a$
 assumes *mono*: $\text{mono } f$
 and $P\text{-}f$: $\bigwedge S. P\ S \Longrightarrow P\ (f\ S)$
 and $P\text{-}Union$: $\bigwedge M. \forall S \in M. P\ S \Longrightarrow P\ (\text{Sup } M)$
 shows $P\ (\text{lfp } f)$
 $\langle \text{proof} \rangle$

lemma *lfp-ordinal-induct-set*:
 assumes *mono*: $\text{mono } f$
 and $P\text{-}f$: $\forall S. P\ S \Longrightarrow P(f\ S)$
 and $P\text{-}Union$: $\forall M. \forall S \in M. P\ S \Longrightarrow P(\text{Union } M)$
 shows $P(\text{lfp } f)$
 $\langle \text{proof} \rangle$

Definition forms of *lfp-unfold* and *lfp-induct*, to control unfolding

lemma *def-lfp-unfold*: $\llbracket h == \text{lfp}(f); \text{mono}(f) \rrbracket \Longrightarrow h = f(h)$
 $\langle \text{proof} \rangle$

lemma *def-lfp-induct*:
 $\llbracket A == \text{lfp}(f); \text{mono}(f);$
 $f\ (\inf A\ P) \leq P$
 $\rrbracket \Longrightarrow A \leq P$
 $\langle \text{proof} \rangle$

lemma *def-lfp-induct-set*:
 $\llbracket A == \text{lfp}(f); \text{mono}(f); a:A;$
 $\forall x. \llbracket x: f(A\ \text{Int } \{x. P(x)\}) \rrbracket \rrbracket \Longrightarrow P(x)$
 $\rrbracket \Longrightarrow P(a)$
 $\langle \text{proof} \rangle$

lemma *lfp-mono*: $(\forall Z. f\ Z \leq g\ Z) \Longrightarrow \text{lfp } f \leq \text{lfp } g$
 $\langle \text{proof} \rangle$

9.4 Proof of Knaster-Tarski Theorem using *gfp*

gfp f is the greatest lower bound of the set $\{u. u \leq f\ u\}$

lemma *gfp-upperbound*: $X \leq f\ X \Longrightarrow X \leq \text{gfp } f$
 $\langle \text{proof} \rangle$

lemma *gfp-least*: $(\forall u. u \leq f\ u \Longrightarrow u \leq X) \Longrightarrow \text{gfp } f \leq X$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma2*: $\text{mono } f \Longrightarrow \text{gfp } f \leq f\ (\text{gfp } f)$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma3*: $\text{mono } f \Longrightarrow f\ (\text{gfp } f) \leq \text{gfp } f$

$\langle \text{proof} \rangle$

lemma *gfp-unfold*: $\text{mono } f \implies \text{gfp } f = f (\text{gfp } f)$
 $\langle \text{proof} \rangle$

9.5 Coinduction rules for greatest fixed points

weak version

lemma *weak-coinduct*: $\llbracket a : X; X \subseteq f(X) \rrbracket \implies a : \text{gfp}(f)$
 $\langle \text{proof} \rangle$

lemma *weak-coinduct-image*: $\llbracket X. \llbracket a : X; g'X \subseteq f(g'X) \rrbracket \implies g a : \text{gfp } f$
 $\langle \text{proof} \rangle$

lemma *coinduct-lemma*:
 $\llbracket X \leq f(\text{sup } X (\text{gfp } f)); \text{mono } f \rrbracket \implies \text{sup } X (\text{gfp } f) \leq f(\text{sup } X (\text{gfp } f))$
 $\langle \text{proof} \rangle$

strong version, thanks to Coen and Frost

lemma *coinduct-set*: $\llbracket \text{mono}(f); a : X; X \subseteq f(X \text{ Un } \text{gfp}(f)) \rrbracket \implies a : \text{gfp}(f)$
 $\langle \text{proof} \rangle$

lemma *coinduct*: $\llbracket \text{mono}(f); X \leq f(\text{sup } X (\text{gfp } f)) \rrbracket \implies X \leq \text{gfp}(f)$
 $\langle \text{proof} \rangle$

lemma *gfp-fun-UnI2*: $\llbracket \text{mono}(f); a : \text{gfp}(f) \rrbracket \implies a : f(X \text{ Un } \text{gfp}(f))$
 $\langle \text{proof} \rangle$

9.6 Even Stronger Coinduction Rule, by Martin Coen

Weakens the condition $X \subseteq f X$ to one expressed using both *lfp* and *gfp*

lemma *coinduct3-mono-lemma*: $\text{mono}(f) \implies \text{mono}(\%x. f(x) \text{ Un } X \text{ Un } B)$
 $\langle \text{proof} \rangle$

lemma *coinduct3-lemma*:
 $\llbracket X \subseteq f(\text{lfp}(\%x. f(x) \text{ Un } X \text{ Un } \text{gfp}(f))); \text{mono}(f) \rrbracket$
 $\implies \text{lfp}(\%x. f(x) \text{ Un } X \text{ Un } \text{gfp}(f)) \subseteq f(\text{lfp}(\%x. f(x) \text{ Un } X \text{ Un } \text{gfp}(f)))$
 $\langle \text{proof} \rangle$

lemma *coinduct3*:
 $\llbracket \text{mono}(f); a : X; X \subseteq f(\text{lfp}(\%x. f(x) \text{ Un } X \text{ Un } \text{gfp}(f))) \rrbracket \implies a : \text{gfp}(f)$
 $\langle \text{proof} \rangle$

Definition forms of *gfp-unfold* and *coinduct*, to control unfolding

lemma *def-gfp-unfold*: $\llbracket A == \text{gfp}(f); \text{mono}(f) \rrbracket \implies A = f(A)$
 $\langle \text{proof} \rangle$

lemma *def-coinduct*:

$\llbracket A == \text{gfp}(f); \text{mono}(f); X \leq f(\text{sup } X \ A) \rrbracket ==> X \leq A$
 $\langle \text{proof} \rangle$

lemma *def-coinduct-set*:

$\llbracket A == \text{gfp}(f); \text{mono}(f); a:X; X \subseteq f(X \ \text{Un } A) \rrbracket ==> a:A$
 $\langle \text{proof} \rangle$

lemma *def-Collect-coinduct*:

$\llbracket A == \text{gfp}(\%w. \text{Collect}(P(w))); \text{mono}(\%w. \text{Collect}(P(w)));$
 $a: X; \ \forall z. z: X ==> P(X \ \text{Un } A) \ z \rrbracket ==>$
 $a : A$
 $\langle \text{proof} \rangle$

lemma *def-coinduct3*:

$\llbracket A == \text{gfp}(f); \text{mono}(f); a:X; X \subseteq f(\text{lfp}(\%x. f(x) \ \text{Un } X \ \text{Un } A)) \rrbracket ==> a:A$
 $\langle \text{proof} \rangle$

Monotonicity of *gfp*!

lemma *gfp-mono*: $(\forall Z. f \ Z \leq g \ Z) ==> \text{gfp } f \leq \text{gfp } g$
 $\langle \text{proof} \rangle$

9.7 Inductive predicates and sets

Inversion of injective functions.

constdefs

myinv :: $('a ==> 'b) ==> ('b ==> 'a)$
 $\text{myinv } (f :: 'a ==> 'b) == \lambda y. \text{THE } x. f \ x = y$

lemma *myinv-f-f*: $\text{inj } f ==> \text{myinv } f \ (f \ x) = x$
 $\langle \text{proof} \rangle$

lemma *f-myinv-f*: $\text{inj } f ==> y \in \text{range } f ==> f \ (\text{myinv } f \ y) = y$
 $\langle \text{proof} \rangle$

hide *const myinv*

Package setup.

theorems *basic-monos* =

subset-refl imp-refl disj-mono conj-mono ex-mono all-mono if-bool-eq-conj
Collect-mono in-mono vimage-mono
imp-conv-disj not-not de-Morgan-disj de-Morgan-conj
not-all not-ex
Ball-def Bex-def
induct-rulify-fallback

$\langle \text{ML} \rangle$

```

theorems [mono] =
  imp-refl disj-mono conj-mono ex-mono all-mono if-bool-eq-conj
  imp-conv-disj not-not de-Morgan-disj de-Morgan-conj
  not-all not-ex
  Ball-def Bex-def
  induct-rulify-fallback

```

9.8 Inductive datatypes and primitive recursion

Package setup.

⟨ML⟩

Lambda-abstractions with pattern matching:

```

syntax
  -lam-pats-syntax :: cases-syn => 'a => 'b          ((%-) 10)
syntax (xsymbols)
  -lam-pats-syntax :: cases-syn => 'a => 'b          ((λ-) 10)

```

⟨ML⟩

end

10 OrderedGroup: Ordered Groups

```

theory OrderedGroup
imports Lattices
uses ~~ /src/Provers/Arith/abel-cancel.ML
begin

```

The theory of partially ordered groups is taken from the books:

- *Lattice Theory* by Garret Birkhoff, American Mathematical Society 1979
- *Partially Ordered Algebraic Systems*, Pergamon Press 1963

Most of the used notions can also be looked up in

- <http://www.mathworld.com> by Eric Weisstein et. al.
- *Algebra I* by van der Waerden, Springer.

⟨ML⟩

The rewrites accumulated in *algebra-simps* deal with the classical algebraic structures of groups, rings and family. They simplify terms by multiplying everything out (in case of a ring) and bringing sums and products into a

canonical form (by ordered rewriting). As a result it decides group and ring equalities but also helps with inequalities.

Of course it also works for fields, but it knows nothing about multiplicative inverses or division. This is catered for by *field-simps*.

10.1 Semigroups and Monoids

```
class semigroup-add = plus +
  assumes add-assoc[algebra-simps]:  $(a + b) + c = a + (b + c)$ 
```

```
class ab-semigroup-add = semigroup-add +
  assumes add-commute[algebra-simps]:  $a + b = b + a$ 
begin
```

```
lemma add-left-commute[algebra-simps]:  $a + (b + c) = b + (a + c)$ 
  <proof>
```

```
theorems add-ac = add-assoc add-commute add-left-commute
```

```
end
```

```
theorems add-ac = add-assoc add-commute add-left-commute
```

```
class semigroup-mult = times +
  assumes mult-assoc[algebra-simps]:  $(a * b) * c = a * (b * c)$ 
```

```
class ab-semigroup-mult = semigroup-mult +
  assumes mult-commute[algebra-simps]:  $a * b = b * a$ 
begin
```

```
lemma mult-left-commute[algebra-simps]:  $a * (b * c) = b * (a * c)$ 
  <proof>
```

```
theorems mult-ac = mult-assoc mult-commute mult-left-commute
```

```
end
```

```
theorems mult-ac = mult-assoc mult-commute mult-left-commute
```

```
class ab-semigroup-idem-mult = ab-semigroup-mult +
  assumes mult-idem[simp]:  $x * x = x$ 
begin
```

```
lemma mult-left-idem[simp]:  $x * (x * y) = x * y$ 
  <proof>
```

```
end
```

```
class monoid-add = zero + semigroup-add +
```

```

assumes add-0-left [simp]:  $0 + a = a$ 
and add-0-right [simp]:  $a + 0 = a$ 

lemma zero-reorient:  $0 = x \longleftrightarrow x = 0$ 
  <proof>

class comm-monoid-add = zero + ab-semigroup-add +
  assumes add-0:  $0 + a = a$ 
begin

subclass monoid-add
  <proof>

end

class monoid-mult = one + semigroup-mult +
  assumes mult-1-left [simp]:  $1 * a = a$ 
  assumes mult-1-right [simp]:  $a * 1 = a$ 

lemma one-reorient:  $1 = x \longleftrightarrow x = 1$ 
  <proof>

class comm-monoid-mult = one + ab-semigroup-mult +
  assumes mult-1:  $1 * a = a$ 
begin

subclass monoid-mult
  <proof>

end

class cancel-semigroup-add = semigroup-add +
  assumes add-left-imp-eq:  $a + b = a + c \implies b = c$ 
  assumes add-right-imp-eq:  $b + a = c + a \implies b = c$ 
begin

lemma add-left-cancel [simp]:
   $a + b = a + c \longleftrightarrow b = c$ 
  <proof>

lemma add-right-cancel [simp]:
   $b + a = c + a \longleftrightarrow b = c$ 
  <proof>

end

class cancel-ab-semigroup-add = ab-semigroup-add +
  assumes add-imp-eq:  $a + b = a + c \implies b = c$ 
begin

```

```
subclass cancel-semigroup-add
   $\langle proof \rangle$ 
```

```
end
```

```
class cancel-comm-monoid-add = cancel-ab-semigroup-add + comm-monoid-add
```

10.2 Groups

```
class group-add = minus + uminus + monoid-add +
  assumes left-minus [simp]:  $- a + a = 0$ 
  assumes diff-minus:  $a - b = a + (- b)$ 
begin
```

```
lemma minus-add-cancel:  $- a + (a + b) = b$ 
   $\langle proof \rangle$ 
```

```
lemma minus-zero [simp]:  $- 0 = 0$ 
   $\langle proof \rangle$ 
```

```
lemma minus-minus [simp]:  $- (- a) = a$ 
   $\langle proof \rangle$ 
```

```
lemma right-minus [simp]:  $a + - a = 0$ 
   $\langle proof \rangle$ 
```

```
lemma right-minus-eq:  $a - b = 0 \longleftrightarrow a = b$ 
   $\langle proof \rangle$ 
```

```
lemma equals-zero-I:
  assumes  $a + b = 0$  shows  $- a = b$ 
   $\langle proof \rangle$ 
```

```
lemma diff-self [simp]:  $a - a = 0$ 
   $\langle proof \rangle$ 
```

```
lemma diff-0 [simp]:  $0 - a = - a$ 
   $\langle proof \rangle$ 
```

```
lemma diff-0-right [simp]:  $a - 0 = a$ 
   $\langle proof \rangle$ 
```

```
lemma diff-minus-eq-add [simp]:  $a - - b = a + b$ 
   $\langle proof \rangle$ 
```

```
lemma neg-equal-iff-equal [simp]:
   $- a = - b \longleftrightarrow a = b$ 
   $\langle proof \rangle$ 
```

lemma *neg-equal-0-iff-equal* [*simp*]:

$$- a = 0 \longleftrightarrow a = 0$$

\langle *proof* \rangle

lemma *neg-0-equal-iff-equal* [*simp*]:

$$0 = - a \longleftrightarrow 0 = a$$

\langle *proof* \rangle

The next two equations can make the simplifier loop!

lemma *equation-minus-iff*:

$$a = - b \longleftrightarrow b = - a$$

\langle *proof* \rangle

lemma *minus-equation-iff*:

$$- a = b \longleftrightarrow - b = a$$

\langle *proof* \rangle

lemma *diff-add-cancel*: $a - b + b = a$

\langle *proof* \rangle

lemma *add-diff-cancel*: $a + b - b = a$

\langle *proof* \rangle

declare *diff-minus*[*symmetric*, *algebra-simps*]

lemma *eq-neg-iff-add-eq-0*: $a = - b \longleftrightarrow a + b = 0$

\langle *proof* \rangle

end

class *ab-group-add* = *minus* + *uminus* + *comm-monoid-add* +

assumes *ab-left-minus*: $- a + a = 0$

assumes *ab-diff-minus*: $a - b = a + (- b)$

begin

subclass *group-add*

\langle *proof* \rangle

subclass *cancel-comm-monoid-add*

\langle *proof* \rangle

lemma *uminus-add-conv-diff*[*algebra-simps*]:

$$- a + b = b - a$$

\langle *proof* \rangle

lemma *minus-add-distrib* [*simp*]:

$$- (a + b) = - a + - b$$

\langle *proof* \rangle

lemma *minus-diff-eq [simp]*:

$$-(a - b) = b - a$$

$\langle proof \rangle$

lemma *add-diff-eq[algebra-simps]*: $a + (b - c) = (a + b) - c$

$\langle proof \rangle$

lemma *diff-add-eq[algebra-simps]*: $(a - b) + c = (a + c) - b$

$\langle proof \rangle$

lemma *diff-eq-eq[algebra-simps]*: $a - b = c \longleftrightarrow a = c + b$

$\langle proof \rangle$

lemma *eq-diff-eq[algebra-simps]*: $a = c - b \longleftrightarrow a + b = c$

$\langle proof \rangle$

lemma *diff-diff-eq[algebra-simps]*: $(a - b) - c = a - (b + c)$

$\langle proof \rangle$

lemma *diff-diff-eq2[algebra-simps]*: $a - (b - c) = (a + c) - b$

$\langle proof \rangle$

lemma *eq-iff-diff-eq-0*: $a = b \longleftrightarrow a - b = 0$

$\langle proof \rangle$

lemma *diff-eq-0-iff-eq [simp, noatp]*: $a - b = 0 \longleftrightarrow a = b$

$\langle proof \rangle$

end

10.3 (Partially) Ordered Groups

class *pordered-ab-semigroup-add* = *order* + *ab-semigroup-add* +

assumes *add-left-mono*: $a \leq b \implies c + a \leq c + b$

begin

lemma *add-right-mono*:

$$a \leq b \implies a + c \leq b + c$$

$\langle proof \rangle$

non-strict, in both arguments

lemma *add-mono*:

$$a \leq b \implies c \leq d \implies a + c \leq b + d$$

$\langle proof \rangle$

end

class *pordered-cancel-ab-semigroup-add* =

pordered-ab-semigroup-add + *cancel-ab-semigroup-add*
begin

lemma *add-strict-left-mono*:
 $a < b \implies c + a < c + b$
 ⟨*proof*⟩

lemma *add-strict-right-mono*:
 $a < b \implies a + c < b + c$
 ⟨*proof*⟩

Strict monotonicity in both arguments

lemma *add-strict-mono*:
 $a < b \implies c < d \implies a + c < b + d$
 ⟨*proof*⟩

lemma *add-less-le-mono*:
 $a < b \implies c \leq d \implies a + c < b + d$
 ⟨*proof*⟩

lemma *add-le-less-mono*:
 $a \leq b \implies c < d \implies a + c < b + d$
 ⟨*proof*⟩

end

class *pordered-ab-semigroup-add-imp-le* =
pordered-cancel-ab-semigroup-add +
assumes *add-le-imp-le-left*: $c + a \leq c + b \implies a \leq b$
begin

lemma *add-less-imp-less-left*:
assumes *less*: $c + a < c + b$ **shows** $a < b$
 ⟨*proof*⟩

lemma *add-less-imp-less-right*:
 $a + c < b + c \implies a < b$
 ⟨*proof*⟩

lemma *add-less-cancel-left* [*simp*]:
 $c + a < c + b \iff a < b$
 ⟨*proof*⟩

lemma *add-less-cancel-right* [*simp*]:
 $a + c < b + c \iff a < b$
 ⟨*proof*⟩

lemma *add-le-cancel-left* [*simp*]:
 $c + a \leq c + b \iff a \leq b$

<proof>

lemma *add-le-cancel-right* [*simp*]:

$$a + c \leq b + c \longleftrightarrow a \leq b$$

<proof>

lemma *add-le-imp-le-right*:

$$a + c \leq b + c \implies a \leq b$$

<proof>

lemma *max-add-distrib-left*:

$$\max x \ y + z = \max (x + z) \ (y + z)$$

<proof>

lemma *min-add-distrib-left*:

$$\min x \ y + z = \min (x + z) \ (y + z)$$

<proof>

end

10.4 Support for reasoning about signs

class *pordered-comm-monoid-add* =

pordered-cancel-ab-semigroup-add + *comm-monoid-add*

begin

lemma *add-pos-nonneg*:

assumes $0 < a$ **and** $0 \leq b$ **shows** $0 < a + b$

<proof>

lemma *add-pos-pos*:

assumes $0 < a$ **and** $0 < b$ **shows** $0 < a + b$

<proof>

lemma *add-nonneg-pos*:

assumes $0 \leq a$ **and** $0 < b$ **shows** $0 < a + b$

<proof>

lemma *add-nonneg-nonneg*:

assumes $0 \leq a$ **and** $0 \leq b$ **shows** $0 \leq a + b$

<proof>

lemma *add-neg-nonpos*:

assumes $a < 0$ **and** $b \leq 0$ **shows** $a + b < 0$

<proof>

lemma *add-neg-neg*:

assumes $a < 0$ **and** $b < 0$ **shows** $a + b < 0$

<proof>

lemma *add-nonpos-neg*:

assumes $a \leq 0$ and $b < 0$ shows $a + b < 0$
 $\langle \text{proof} \rangle$

lemma *add-nonpos-nonpos*:

assumes $a \leq 0$ and $b \leq 0$ shows $a + b \leq 0$
 $\langle \text{proof} \rangle$

lemmas *add-sign-intros* =

add-pos-nonneg add-pos-pos add-nonneg-pos add-nonneg-nonneg
add-neg-nonpos add-neg-neg add-nonpos-neg add-nonpos-nonpos

lemma *add-nonneg-eq-0-iff*:

assumes $x: 0 \leq x$ and $y: 0 \leq y$
 shows $x + y = 0 \iff x = 0 \wedge y = 0$
 $\langle \text{proof} \rangle$

end

class *pordered-ab-group-add* =

ab-group-add + *pordered-ab-semigroup-add*
begin

subclass *pordered-cancel-ab-semigroup-add* $\langle \text{proof} \rangle$

subclass *pordered-ab-semigroup-add-imp-le*
 $\langle \text{proof} \rangle$

subclass *pordered-comm-monoid-add* $\langle \text{proof} \rangle$

lemma *max-diff-distrib-left*:

shows $\max x y - z = \max (x - z) (y - z)$
 $\langle \text{proof} \rangle$

lemma *min-diff-distrib-left*:

shows $\min x y - z = \min (x - z) (y - z)$
 $\langle \text{proof} \rangle$

lemma *le-imp-neg-le*:

assumes $a \leq b$ shows $-b \leq -a$
 $\langle \text{proof} \rangle$

lemma *neg-le-iff-le* [simp]: $-b \leq -a \iff a \leq b$
 $\langle \text{proof} \rangle$

lemma *neg-le-0-iff-le* [simp]: $-a \leq 0 \iff 0 \leq a$
 $\langle \text{proof} \rangle$

lemma *neg-0-le-iff-le* [*simp*]: $0 \leq -a \longleftrightarrow a \leq 0$
 ⟨*proof*⟩

lemma *neg-less-iff-less* [*simp*]: $-b < -a \longleftrightarrow a < b$
 ⟨*proof*⟩

lemma *neg-less-0-iff-less* [*simp*]: $-a < 0 \longleftrightarrow 0 < a$
 ⟨*proof*⟩

lemma *neg-0-less-iff-less* [*simp*]: $0 < -a \longleftrightarrow a < 0$
 ⟨*proof*⟩

The next several equations can make the simplifier loop!

lemma *less-minus-iff*: $a < -b \longleftrightarrow b < -a$
 ⟨*proof*⟩

lemma *minus-less-iff*: $-a < b \longleftrightarrow -b < a$
 ⟨*proof*⟩

lemma *le-minus-iff*: $a \leq -b \longleftrightarrow b \leq -a$
 ⟨*proof*⟩

lemma *minus-le-iff*: $-a \leq b \longleftrightarrow -b \leq a$
 ⟨*proof*⟩

lemma *less-iff-diff-less-0*: $a < b \longleftrightarrow a - b < 0$
 ⟨*proof*⟩

lemma *diff-less-eq*[*algebra-simps*]: $a - b < c \longleftrightarrow a < c + b$
 ⟨*proof*⟩

lemma *less-diff-eq*[*algebra-simps*]: $a < c - b \longleftrightarrow a + b < c$
 ⟨*proof*⟩

lemma *diff-le-eq*[*algebra-simps*]: $a - b \leq c \longleftrightarrow a \leq c + b$
 ⟨*proof*⟩

lemma *le-diff-eq*[*algebra-simps*]: $a \leq c - b \longleftrightarrow a + b \leq c$
 ⟨*proof*⟩

lemma *le-iff-diff-le-0*: $a \leq b \longleftrightarrow a - b \leq 0$
 ⟨*proof*⟩

Legacy - use *algebra-simps*

lemmas *group-simps*[*noatp*] = *algebra-simps*

end

Legacy - use *algebra-simps*

```

lemmas group-simps[noatp] = algebra-simps

class ordered-ab-semigroup-add =
  linorder + pordered-ab-semigroup-add

class ordered-cancel-ab-semigroup-add =
  linorder + pordered-cancel-ab-semigroup-add
begin

subclass ordered-ab-semigroup-add ⟨proof⟩

subclass pordered-ab-semigroup-add-imp-le
  ⟨proof⟩

end

class ordered-ab-group-add =
  linorder + pordered-ab-group-add
begin

subclass ordered-cancel-ab-semigroup-add ⟨proof⟩

lemma neg-less-eq-nonneg:
   $- a \leq a \longleftrightarrow 0 \leq a$ 
  ⟨proof⟩

lemma less-eq-neg-nonpos:
   $a \leq - a \longleftrightarrow a \leq 0$ 
  ⟨proof⟩

lemma equal-neg-zero:
   $a = - a \longleftrightarrow a = 0$ 
  ⟨proof⟩

lemma neg-equal-zero:
   $- a = a \longleftrightarrow a = 0$ 
  ⟨proof⟩

end

— FIXME localize the following

lemma add-increasing:
  fixes c :: 'a::{pordered-ab-semigroup-add-imp-le, comm-monoid-add}
  shows  $[|0 \leq a; b \leq c|] ==> b \leq a + c$ 
  ⟨proof⟩

lemma add-increasing2:
  fixes c :: 'a::{pordered-ab-semigroup-add-imp-le, comm-monoid-add}

```

shows $[|0 \leq c; b \leq a|] \implies b \leq a + c$
 $\langle \text{proof} \rangle$

lemma *add-strict-increasing*:

fixes $c :: 'a::\{\text{pordered-ab-semigroup-add-imp-le}, \text{comm-monoid-add}\}$
shows $[|0 < a; b \leq c|] \implies b < a + c$
 $\langle \text{proof} \rangle$

lemma *add-strict-increasing2*:

fixes $c :: 'a::\{\text{pordered-ab-semigroup-add-imp-le}, \text{comm-monoid-add}\}$
shows $[|0 \leq a; b < c|] \implies b < a + c$
 $\langle \text{proof} \rangle$

class *pordered-ab-group-add-abs* = *pordered-ab-group-add* + *abs* +
assumes *abs-ge-zero* [*simp*]: $|a| \geq 0$
and *abs-ge-self*: $a \leq |a|$
and *abs-leI*: $a \leq b \implies -a \leq b \implies |a| \leq b$
and *abs-minus-cancel* [*simp*]: $|-a| = |a|$
and *abs-triangle-ineq*: $|a + b| \leq |a| + |b|$
begin

lemma *abs-minus-le-zero*: $-|a| \leq 0$
 $\langle \text{proof} \rangle$

lemma *abs-of-nonneg* [*simp*]:
assumes *nonneg*: $0 \leq a$ **shows** $|a| = a$
 $\langle \text{proof} \rangle$

lemma *abs-idempotent* [*simp*]: $||a|| = |a|$
 $\langle \text{proof} \rangle$

lemma *abs-eq-0* [*simp*]: $|a| = 0 \longleftrightarrow a = 0$
 $\langle \text{proof} \rangle$

lemma *abs-zero* [*simp*]: $|0| = 0$
 $\langle \text{proof} \rangle$

lemma *abs-0-eq* [*simp*, *noatp*]: $0 = |a| \longleftrightarrow a = 0$
 $\langle \text{proof} \rangle$

lemma *abs-le-zero-iff* [*simp*]: $|a| \leq 0 \longleftrightarrow a = 0$
 $\langle \text{proof} \rangle$

lemma *zero-less-abs-iff* [*simp*]: $0 < |a| \longleftrightarrow a \neq 0$
 $\langle \text{proof} \rangle$

lemma *abs-not-less-zero* [*simp*]: $\neg |a| < 0$
 $\langle \text{proof} \rangle$

lemma *abs-ge-minus-self*: $- a \leq |a|$
 $\langle proof \rangle$

lemma *abs-minus-commute*:
 $|a - b| = |b - a|$
 $\langle proof \rangle$

lemma *abs-of-pos*: $0 < a \implies |a| = a$
 $\langle proof \rangle$

lemma *abs-of-nonpos* [*simp*]:
assumes $a \leq 0$ **shows** $|a| = - a$
 $\langle proof \rangle$

lemma *abs-of-neg*: $a < 0 \implies |a| = - a$
 $\langle proof \rangle$

lemma *abs-le-D1*: $|a| \leq b \implies a \leq b$
 $\langle proof \rangle$

lemma *abs-le-D2*: $|a| \leq b \implies - a \leq b$
 $\langle proof \rangle$

lemma *abs-le-iff*: $|a| \leq b \longleftrightarrow a \leq b \wedge - a \leq b$
 $\langle proof \rangle$

lemma *abs-triangle-ineq2*: $|a| - |b| \leq |a - b|$
 $\langle proof \rangle$

lemma *abs-triangle-ineq3*: $||a| - |b|| \leq |a - b|$
 $\langle proof \rangle$

lemma *abs-triangle-ineq4*: $|a - b| \leq |a| + |b|$
 $\langle proof \rangle$

lemma *abs-diff-triangle-ineq*: $|a + b - (c + d)| \leq |a - c| + |b - d|$
 $\langle proof \rangle$

lemma *abs-add-abs* [*simp*]:
 $||a| + |b|| = |a| + |b|$ (**is** ?L = ?R)
 $\langle proof \rangle$

end

10.5 Lattice Ordered (Abelian) Groups

class *lordered-ab-group-add-meet* = *pordered-ab-group-add* + *lower-semilattice*
begin

lemma *add-inf-distrib-left*:
 $a + \inf b\ c = \inf (a + b)\ (a + c)$
 $\langle \text{proof} \rangle$

lemma *add-inf-distrib-right*:
 $\inf a\ b + c = \inf (a + c)\ (b + c)$
 $\langle \text{proof} \rangle$

end

class *lordered-ab-group-add-join* = *pordered-ab-group-add* + *upper-semilattice*
begin

lemma *add-sup-distrib-left*:
 $a + \sup b\ c = \sup (a + b)\ (a + c)$
 $\langle \text{proof} \rangle$

lemma *add-sup-distrib-right*:
 $\sup a\ b + c = \sup (a + c)\ (b + c)$
 $\langle \text{proof} \rangle$

end

class *lordered-ab-group-add* = *pordered-ab-group-add* + *lattice*
begin

subclass *lordered-ab-group-add-meet* $\langle \text{proof} \rangle$
subclass *lordered-ab-group-add-join* $\langle \text{proof} \rangle$

lemmas *add-sup-inf-distribs* = *add-inf-distrib-right* *add-inf-distrib-left* *add-sup-distrib-right*
add-sup-distrib-left

lemma *inf-eq-neg-sup*: $\inf a\ b = -\ \sup (-a)\ (-b)$
 $\langle \text{proof} \rangle$

lemma *sup-eq-neg-inf*: $\sup a\ b = -\ \inf (-a)\ (-b)$
 $\langle \text{proof} \rangle$

lemma *neg-inf-eq-sup*: $-\ \inf a\ b = \sup (-a)\ (-b)$
 $\langle \text{proof} \rangle$

lemma *neg-sup-eq-inf*: $-\ \sup a\ b = \inf (-a)\ (-b)$
 $\langle \text{proof} \rangle$

lemma *add-eq-inf-sup*: $a + b = \sup a\ b + \inf a\ b$
 $\langle \text{proof} \rangle$

10.6 Positive Part, Negative Part, Absolute Value

definition

$nprt :: 'a \Rightarrow 'a$ **where**
 $nprt\ x = \inf\ x\ 0$

definition

$pprt :: 'a \Rightarrow 'a$ **where**
 $pprt\ x = \sup\ x\ 0$

lemma *pprt-neg*: $pprt\ (-\ x) = -\ nprt\ x$
 $\langle proof \rangle$

lemma *nprt-neg*: $nprt\ (-\ x) = -\ pprt\ x$
 $\langle proof \rangle$

lemma *prts*: $a = pprt\ a + nprt\ a$
 $\langle proof \rangle$

lemma *zero-le-pprt[simp]*: $0 \leq pprt\ a$
 $\langle proof \rangle$

lemma *nprt-le-zero[simp]*: $nprt\ a \leq 0$
 $\langle proof \rangle$

lemma *le-eq-neg*: $a \leq -\ b \iff a + b \leq 0$ (**is** ?l = ?r)
 $\langle proof \rangle$

lemma *pprt-0[simp]*: $pprt\ 0 = 0$ $\langle proof \rangle$

lemma *nprt-0[simp]*: $nprt\ 0 = 0$ $\langle proof \rangle$

lemma *pprt-eq-id [simp, noatp]*: $0 \leq x \implies pprt\ x = x$
 $\langle proof \rangle$

lemma *nprt-eq-id [simp, noatp]*: $x \leq 0 \implies nprt\ x = x$
 $\langle proof \rangle$

lemma *pprt-eq-0 [simp, noatp]*: $x \leq 0 \implies pprt\ x = 0$
 $\langle proof \rangle$

lemma *nprt-eq-0 [simp, noatp]*: $0 \leq x \implies nprt\ x = 0$
 $\langle proof \rangle$

lemma *sup-0-imp-0*: $\sup\ a\ (-\ a) = 0 \implies a = 0$
 $\langle proof \rangle$

lemma *inf-0-imp-0*: $\inf\ a\ (-\ a) = 0 \implies a = 0$
 $\langle proof \rangle$

lemma *inf-0-eq-0 [simp, noatp]*: $\inf\ a\ (-\ a) = 0 \iff a = 0$

$\langle \text{proof} \rangle$

lemma *sup-0-eq-0* [*simp*, *noatp*]: $\text{sup } a \ (-\ a) = 0 \longleftrightarrow a = 0$
 $\langle \text{proof} \rangle$

lemma *zero-le-double-add-iff-zero-le-single-add* [*simp*]:
 $0 \leq a + a \longleftrightarrow 0 \leq a$
 $\langle \text{proof} \rangle$

lemma *double-zero*: $a + a = 0 \longleftrightarrow a = 0$
 $\langle \text{proof} \rangle$

lemma *zero-less-double-add-iff-zero-less-single-add*:
 $0 < a + a \longleftrightarrow 0 < a$
 $\langle \text{proof} \rangle$

lemma *double-add-le-zero-iff-single-add-le-zero* [*simp*]:
 $a + a \leq 0 \longleftrightarrow a \leq 0$
 $\langle \text{proof} \rangle$

lemma *double-add-less-zero-iff-single-less-zero* [*simp*]:
 $a + a < 0 \longleftrightarrow a < 0$
 $\langle \text{proof} \rangle$

declare *neg-inf-eq-sup* [*simp*] *neg-sup-eq-inf* [*simp*]

lemma *le-minus-self-iff*: $a \leq -\ a \longleftrightarrow a \leq 0$
 $\langle \text{proof} \rangle$

lemma *minus-le-self-iff*: $-\ a \leq a \longleftrightarrow 0 \leq a$
 $\langle \text{proof} \rangle$

lemma *zero-le-iff-zero-nprt*: $0 \leq a \longleftrightarrow \text{nprt } a = 0$
 $\langle \text{proof} \rangle$

lemma *le-zero-iff-zero-pprt*: $a \leq 0 \longleftrightarrow \text{pprt } a = 0$
 $\langle \text{proof} \rangle$

lemma *le-zero-iff-pprt-id*: $0 \leq a \longleftrightarrow \text{pprt } a = a$
 $\langle \text{proof} \rangle$

lemma *zero-le-iff-nprt-id*: $a \leq 0 \longleftrightarrow \text{nprt } a = a$
 $\langle \text{proof} \rangle$

lemma *pprt-mono* [*simp*, *noatp*]: $a \leq b \implies \text{pprt } a \leq \text{pprt } b$
 $\langle \text{proof} \rangle$

lemma *nprt-mono* [*simp*, *noatp*]: $a \leq b \implies \text{nprt } a \leq \text{nprt } b$
 $\langle \text{proof} \rangle$

end

lemmas *add-sup-inf-distrib* = *add-inf-distrib-right add-inf-distrib-left add-sup-distrib-right add-sup-distrib-left*

class *lordered-ab-group-add-abs* = *lordered-ab-group-add* + *abs* +
assumes *abs-lattice*: $|a| = \sup a \ (-a)$
begin

lemma *abs-prts*: $|a| = \text{pprt } a - \text{nppt } a$
 $\langle \text{proof} \rangle$

subclass *pordered-ab-group-add-abs*
 $\langle \text{proof} \rangle$

end

lemma *sup-eq-if*:
fixes $a :: 'a :: \{ \text{lordered-ab-group-add}, \text{linorder} \}$
shows $\sup a \ (-a) = (\text{if } a < 0 \text{ then } -a \text{ else } a)$
 $\langle \text{proof} \rangle$

lemma *abs-if-lattice*:
fixes $a :: 'a :: \{ \text{lordered-ab-group-add-abs}, \text{linorder} \}$
shows $|a| = (\text{if } a < 0 \text{ then } -a \text{ else } a)$
 $\langle \text{proof} \rangle$

Needed for abelian cancellation simprocs:

lemma *add-cancel-21*: $((x :: 'a :: \text{ab-group-add}) + (y + z) = y + u) = (x + z = u)$
 $\langle \text{proof} \rangle$

lemma *add-cancel-end*: $(x + (y + z) = y) = (x = - (z :: 'a :: \text{ab-group-add}))$
 $\langle \text{proof} \rangle$

lemma *less-eqI*: $(x :: 'a :: \text{pordered-ab-group-add}) - y = x' - y' \implies (x < y) = (x' < y')$
 $\langle \text{proof} \rangle$

lemma *le-eqI*: $(x :: 'a :: \text{pordered-ab-group-add}) - y = x' - y' \implies (y \leq x) = (y' \leq x')$
 $\langle \text{proof} \rangle$

lemma *eq-eqI*: $(x :: 'a :: \text{ab-group-add}) - y = x' - y' \implies (x = y) = (x' = y')$
 $\langle \text{proof} \rangle$

lemma *diff-def*: $(x :: 'a :: \text{ab-group-add}) - y == x + (-y)$
 $\langle \text{proof} \rangle$

lemma *add-minus-cancel*: $(a::'a::ab\text{-group-add}) + (-a + b) = b$
 $\langle proof \rangle$

lemma *le-add-right-mono*:
assumes
 $a \leq b + (c::'a::pordered\text{-ab-group-add})$
 $c \leq d$
shows $a \leq b + d$
 $\langle proof \rangle$

lemma *estimate-by-abs*:
 $a + b \leq (c::'a::lordered\text{-ab-group-add-abs}) \implies a \leq c + abs\ b$
 $\langle proof \rangle$

10.7 Tools setup

lemma *add-mono-thms-ordered-semiring* [noatp]:
fixes $i\ j\ k :: 'a::pordered\text{-ab-semigroup-add}$
shows $i \leq j \wedge k \leq l \implies i + k \leq j + l$
and $i = j \wedge k \leq l \implies i + k \leq j + l$
and $i \leq j \wedge k = l \implies i + k \leq j + l$
and $i = j \wedge k = l \implies i + k = j + l$
 $\langle proof \rangle$

lemma *add-mono-thms-ordered-field* [noatp]:
fixes $i\ j\ k :: 'a::pordered\text{-cancel-ab-semigroup-add}$
shows $i < j \wedge k = l \implies i + k < j + l$
and $i = j \wedge k < l \implies i + k < j + l$
and $i < j \wedge k \leq l \implies i + k < j + l$
and $i \leq j \wedge k < l \implies i + k < j + l$
and $i < j \wedge k < l \implies i + k < j + l$
 $\langle proof \rangle$

Simplification of $x - y < (0::'a)$, etc.

lemmas *diff-less-0-iff-less* [simp, noatp] = *less-iff-diff-less-0* [symmetric]

lemmas *diff-le-0-iff-le* [simp, noatp] = *le-iff-diff-le-0* [symmetric]

$\langle ML \rangle$

end

11 Ring-and-Field: (Ordered) Rings and Fields

theory *Ring-and-Field*
imports *OrderedGroup*
begin

The theory of partially ordered rings is taken from the books:

- *Lattice Theory* by Garret Birkhoff, American Mathematical Society 1979
- *Partially Ordered Algebraic Systems*, Pergamon Press 1963

Most of the used notions can also be looked up in

- <http://www.mathworld.com> by Eric Weisstein et. al.
- *Algebra I* by van der Waerden, Springer.

```
class semiring = ab-semigroup-add + semigroup-mult +
  assumes left-distrib[algebra-simps]:  $(a + b) * c = a * c + b * c$ 
  assumes right-distrib[algebra-simps]:  $a * (b + c) = a * b + a * c$ 
begin
```

For the *combine-numerals* simproc

```
lemma combine-common-factor:
   $a * e + (b * e + c) = (a + b) * e + c$ 
<proof>
```

end

```
class mult-zero = times + zero +
  assumes mult-zero-left [simp]:  $0 * a = 0$ 
  assumes mult-zero-right [simp]:  $a * 0 = 0$ 
```

```
class semiring-0 = semiring + comm-monoid-add + mult-zero
```

```
class semiring-0-cancel = semiring + cancel-comm-monoid-add
begin
```

```
subclass semiring-0
<proof>
```

end

```
class comm-semiring = ab-semigroup-add + ab-semigroup-mult +
  assumes distrib:  $(a + b) * c = a * c + b * c$ 
begin
```

```
subclass semiring
<proof>
```

end

```
class comm-semiring-0 = comm-semiring + comm-monoid-add + mult-zero
```

```

begin

subclass semiring-0 ⟨proof⟩

end

class comm-semiring-0-cancel = comm-semiring + cancel-comm-monoid-add
begin

subclass semiring-0-cancel ⟨proof⟩

subclass comm-semiring-0 ⟨proof⟩

end

class zero-neg-one = zero + one +
  assumes zero-neg-one [simp]: 0 ≠ 1
begin

lemma one-neg-zero [simp]: 1 ≠ 0
  ⟨proof⟩

end

class semiring-1 = zero-neg-one + semiring-0 + monoid-mult

Abstract divisibility

class dvd = times
begin

definition dvd :: 'a ⇒ 'a ⇒ bool (infixl dvd 50) where
  [code del]: b dvd a ⟷ (∃ k. a = b * k)

lemma dvdI [intro?]: a = b * k ⟹ b dvd a
  ⟨proof⟩

lemma dvdE [elim?]: b dvd a ⟹ (∧ k. a = b * k ⟹ P) ⟹ P
  ⟨proof⟩

end

class comm-semiring-1 = zero-neg-one + comm-semiring-0 + comm-monoid-mult
+ dvd

begin

subclass semiring-1 ⟨proof⟩

lemma dvd-refl[simp]: a dvd a

```

$\langle proof \rangle$

lemma *dvd-trans*:

assumes $a \text{ dvd } b$ and $b \text{ dvd } c$

shows $a \text{ dvd } c$

$\langle proof \rangle$

lemma *dvd-0-left-iff* [*noatp*, *simp*]: $0 \text{ dvd } a \iff a = 0$

$\langle proof \rangle$

lemma *dvd-0-right* [*iff*]: $a \text{ dvd } 0$

$\langle proof \rangle$

lemma *one-dvd* [*simp*]: $1 \text{ dvd } a$

$\langle proof \rangle$

lemma *dvd-mult*[*simp*]: $a \text{ dvd } c \implies a \text{ dvd } (b * c)$

$\langle proof \rangle$

lemma *dvd-mult2*[*simp*]: $a \text{ dvd } b \implies a \text{ dvd } (b * c)$

$\langle proof \rangle$

lemma *dvd-triv-right* [*simp*]: $a \text{ dvd } b * a$

$\langle proof \rangle$

lemma *dvd-triv-left* [*simp*]: $a \text{ dvd } a * b$

$\langle proof \rangle$

lemma *mult-dvd-mono*:

assumes $a \text{ dvd } b$

and $c \text{ dvd } d$

shows $a * c \text{ dvd } b * d$

$\langle proof \rangle$

lemma *dvd-mult-left*: $a * b \text{ dvd } c \implies a \text{ dvd } c$

$\langle proof \rangle$

lemma *dvd-mult-right*: $a * b \text{ dvd } c \implies b \text{ dvd } c$

$\langle proof \rangle$

lemma *dvd-0-left*: $0 \text{ dvd } a \implies a = 0$

$\langle proof \rangle$

lemma *dvd-add*[*simp*]:

assumes $a \text{ dvd } b$ and $a \text{ dvd } c$ shows $a \text{ dvd } (b + c)$

$\langle proof \rangle$

end

```

class no-zero-divisors = zero + times +
  assumes no-zero-divisors:  $a \neq 0 \implies b \neq 0 \implies a * b \neq 0$ 

class semiring-1-cancel = semiring + cancel-comm-monoid-add
  + zero-neq-one + monoid-mult
begin

subclass semiring-0-cancel <proof>

subclass semiring-1 <proof>

end

class comm-semiring-1-cancel = comm-semiring + cancel-comm-monoid-add
  + zero-neq-one + comm-monoid-mult
begin

subclass semiring-1-cancel <proof>
subclass comm-semiring-0-cancel <proof>
subclass comm-semiring-1 <proof>

end

class ring = semiring + ab-group-add
begin

subclass semiring-0-cancel <proof>

Distribution rules

lemma minus-mult-left:  $-(a * b) = -a * b$ 
<proof>

lemma minus-mult-right:  $-(a * b) = a * -b$ 
<proof>

Extract signs from products

lemmas mult-minus-left [simp, noatp] = minus-mult-left [symmetric]
lemmas mult-minus-right [simp, noatp] = minus-mult-right [symmetric]

lemma minus-mult-minus [simp]:  $-a * -b = a * b$ 
<proof>

lemma minus-mult-commute:  $-a * b = a * -b$ 
<proof>

lemma right-diff-distrib[algebra-simps]:  $a * (b - c) = a * b - a * c$ 
<proof>

```

lemma *left-diff-distrib*[*algebra-simps*]: $(a - b) * c = a * c - b * c$
 $\langle proof \rangle$

lemmas *ring-distrib*[*noatp*] =
right-distrib left-distrib left-diff-distrib right-diff-distrib

Legacy - use *algebra-simps*

lemmas *ring-simps*[*noatp*] = *algebra-simps*

lemma *eq-add-iff1*:
 $a * e + c = b * e + d \longleftrightarrow (a - b) * e + c = d$
 $\langle proof \rangle$

lemma *eq-add-iff2*:
 $a * e + c = b * e + d \longleftrightarrow c = (b - a) * e + d$
 $\langle proof \rangle$

end

lemmas *ring-distrib*[*noatp*] =
right-distrib left-distrib left-diff-distrib right-diff-distrib

class *comm-ring* = *comm-semiring* + *ab-group-add*
begin

subclass *ring* $\langle proof \rangle$
subclass *comm-semiring-0-cancel* $\langle proof \rangle$

end

class *ring-1* = *ring* + *zero-neq-one* + *monoid-mult*
begin

subclass *semiring-1-cancel* $\langle proof \rangle$

end

class *comm-ring-1* = *comm-ring* + *zero-neq-one* + *comm-monoid-mult*
begin

subclass *ring-1* $\langle proof \rangle$
subclass *comm-semiring-1-cancel* $\langle proof \rangle$

lemma *dvd-minus-iff* [*simp*]: $x \text{ dvd } - y \longleftrightarrow x \text{ dvd } y$
 $\langle proof \rangle$

lemma *minus-dvd-iff* [*simp*]: $- x \text{ dvd } y \longleftrightarrow x \text{ dvd } y$
 $\langle proof \rangle$

lemma *dvd-diff* [*simp*]: $x \text{ dvd } y \implies x \text{ dvd } z \implies x \text{ dvd } (y - z)$
 $\langle \text{proof} \rangle$

end

class *ring-no-zero-divisors* = *ring* + *no-zero-divisors*
begin

lemma *mult-eq-0-iff* [*simp*]:
shows $a * b = 0 \iff (a = 0 \vee b = 0)$
 $\langle \text{proof} \rangle$

Cancellation of equalities with a common factor

lemma *mult-cancel-right* [*simp*, *noatp*]:
 $a * c = b * c \iff c = 0 \vee a = b$
 $\langle \text{proof} \rangle$

lemma *mult-cancel-left* [*simp*, *noatp*]:
 $c * a = c * b \iff c = 0 \vee a = b$
 $\langle \text{proof} \rangle$

end

class *ring-1-no-zero-divisors* = *ring-1* + *ring-no-zero-divisors*
begin

lemma *mult-cancel-right1* [*simp*]:
 $c = b * c \iff c = 0 \vee b = 1$
 $\langle \text{proof} \rangle$

lemma *mult-cancel-right2* [*simp*]:
 $a * c = c \iff c = 0 \vee a = 1$
 $\langle \text{proof} \rangle$

lemma *mult-cancel-left1* [*simp*]:
 $c = c * b \iff c = 0 \vee b = 1$
 $\langle \text{proof} \rangle$

lemma *mult-cancel-left2* [*simp*]:
 $c * a = c \iff c = 0 \vee a = 1$
 $\langle \text{proof} \rangle$

end

class *idom* = *comm-ring-1* + *no-zero-divisors*
begin

subclass *ring-1-no-zero-divisors* $\langle \text{proof} \rangle$

lemma *square-eq-iff*: $a * a = b * b \longleftrightarrow (a = b \vee a = -b)$
 $\langle \text{proof} \rangle$

lemma *dvd-mult-cancel-right* [simp]:
 $a * c \text{ dvd } b * c \longleftrightarrow c = 0 \vee a \text{ dvd } b$
 $\langle \text{proof} \rangle$

lemma *dvd-mult-cancel-left* [simp]:
 $c * a \text{ dvd } c * b \longleftrightarrow c = 0 \vee a \text{ dvd } b$
 $\langle \text{proof} \rangle$

end

class *division-ring* = *ring-1* + *inverse* +
assumes *left-inverse* [simp]: $a \neq 0 \implies \text{inverse } a * a = 1$
assumes *right-inverse* [simp]: $a \neq 0 \implies a * \text{inverse } a = 1$
begin

subclass *ring-1-no-zero-divisors*
 $\langle \text{proof} \rangle$

lemma *nonzero-imp-inverse-nonzero*:
 $a \neq 0 \implies \text{inverse } a \neq 0$
 $\langle \text{proof} \rangle$

lemma *inverse-zero-imp-zero*:
 $\text{inverse } a = 0 \implies a = 0$
 $\langle \text{proof} \rangle$

lemma *inverse-unique*:
assumes *ab*: $a * b = 1$
shows $\text{inverse } a = b$
 $\langle \text{proof} \rangle$

lemma *nonzero-inverse-minus-eq*:
 $a \neq 0 \implies \text{inverse } (-a) = -\text{inverse } a$
 $\langle \text{proof} \rangle$

lemma *nonzero-inverse-inverse-eq*:
 $a \neq 0 \implies \text{inverse } (\text{inverse } a) = a$
 $\langle \text{proof} \rangle$

lemma *nonzero-inverse-eq-imp-eq*:
assumes $\text{inverse } a = \text{inverse } b$ **and** $a \neq 0$ **and** $b \neq 0$
shows $a = b$
 $\langle \text{proof} \rangle$

lemma *inverse-1* [simp]: $\text{inverse } 1 = 1$

$\langle proof \rangle$

lemma *nonzero-inverse-mult-distrib*:

assumes $a \neq 0$ **and** $b \neq 0$

shows $inverse (a * b) = inverse b * inverse a$

$\langle proof \rangle$

lemma *division-ring-inverse-add*:

$a \neq 0 \implies b \neq 0 \implies inverse a + inverse b = inverse a * (a + b) * inverse b$

$\langle proof \rangle$

lemma *division-ring-inverse-diff*:

$a \neq 0 \implies b \neq 0 \implies inverse a - inverse b = inverse a * (b - a) * inverse b$

$\langle proof \rangle$

end

class *field* = *comm-ring-1* + *inverse* +

assumes *field-inverse*: $a \neq 0 \implies inverse a * a = 1$

assumes *divide-inverse*: $a / b = a * inverse b$

begin

subclass *division-ring*

$\langle proof \rangle$

subclass *idom* $\langle proof \rangle$

lemma *right-inverse-eq*: $b \neq 0 \implies a / b = 1 \longleftrightarrow a = b$

$\langle proof \rangle$

lemma *nonzero-inverse-eq-divide*: $a \neq 0 \implies inverse a = 1 / a$

$\langle proof \rangle$

lemma *divide-self* [*simp*]: $a \neq 0 \implies a / a = 1$

$\langle proof \rangle$

lemma *divide-zero-left* [*simp*]: $0 / a = 0$

$\langle proof \rangle$

lemma *inverse-eq-divide*: $inverse a = 1 / a$

$\langle proof \rangle$

lemma *add-divide-distrib*: $(a+b) / c = a/c + b/c$

$\langle proof \rangle$

There is no slick version using division by zero.

lemma *inverse-add*:

$[[a \neq 0; b \neq 0]]$

$\implies inverse a + inverse b = (a + b) * inverse a * inverse b$

$\langle proof \rangle$

lemma *nonzero-mult-divide-mult-cancel-left* [simp, noatp]:
assumes [simp]: $b \neq 0$ **and** [simp]: $c \neq 0$ **shows** $(c * a) / (c * b) = a / b$
 $\langle proof \rangle$

lemma *nonzero-mult-divide-mult-cancel-right* [simp, noatp]:
 $\llbracket b \neq 0; c \neq 0 \rrbracket \implies (a * c) / (b * c) = a / b$
 $\langle proof \rangle$

lemma *divide-1* [simp]: $a / 1 = a$
 $\langle proof \rangle$

lemma *times-divide-eq-right*: $a * (b / c) = (a * b) / c$
 $\langle proof \rangle$

lemma *times-divide-eq-left*: $(b / c) * a = (b * a) / c$
 $\langle proof \rangle$

These are later declared as simp rules.

lemmas *times-divide-eq* [noatp] = *times-divide-eq-right times-divide-eq-left*

lemma *add-frac-eq*:
assumes $y \neq 0$ **and** $z \neq 0$
shows $x / y + w / z = (x * z + w * y) / (y * z)$
 $\langle proof \rangle$

Special Cancellation Simprules for Division

lemma *nonzero-mult-divide-cancel-right* [simp, noatp]:
 $b \neq 0 \implies a * b / b = a$
 $\langle proof \rangle$

lemma *nonzero-mult-divide-cancel-left* [simp, noatp]:
 $a \neq 0 \implies a * b / a = b$
 $\langle proof \rangle$

lemma *nonzero-divide-mult-cancel-right* [simp, noatp]:
 $\llbracket a \neq 0; b \neq 0 \rrbracket \implies b / (a * b) = 1 / a$
 $\langle proof \rangle$

lemma *nonzero-divide-mult-cancel-left* [simp, noatp]:
 $\llbracket a \neq 0; b \neq 0 \rrbracket \implies a / (a * b) = 1 / b$
 $\langle proof \rangle$

lemma *nonzero-mult-divide-mult-cancel-left2* [simp, noatp]:
 $\llbracket b \neq 0; c \neq 0 \rrbracket \implies (c * a) / (b * c) = a / b$
 $\langle proof \rangle$

lemma *nonzero-mult-divide-mult-cancel-right2* [simp, noatp]:

$\llbracket b \neq 0; c \neq 0 \rrbracket \implies (a * c) / (c * b) = a / b$
 $\langle \text{proof} \rangle$

lemma *minus-divide-left*: $-(a / b) = (-a) / b$
 $\langle \text{proof} \rangle$

lemma *nonzero-minus-divide-right*: $b \neq 0 \implies -(a / b) = a / (-b)$
 $\langle \text{proof} \rangle$

lemma *nonzero-minus-divide-divide*: $b \neq 0 \implies (-a) / (-b) = a / b$
 $\langle \text{proof} \rangle$

lemma *divide-minus-left* [*simp, noatp*]: $(-a) / b = -(a / b)$
 $\langle \text{proof} \rangle$

lemma *diff-divide-distrib*: $(a - b) / c = a / c - b / c$
 $\langle \text{proof} \rangle$

lemma *add-divide-eq-iff*:
 $z \neq 0 \implies x + y / z = (z * x + y) / z$
 $\langle \text{proof} \rangle$

lemma *divide-add-eq-iff*:
 $z \neq 0 \implies x / z + y = (x + z * y) / z$
 $\langle \text{proof} \rangle$

lemma *diff-divide-eq-iff*:
 $z \neq 0 \implies x - y / z = (z * x - y) / z$
 $\langle \text{proof} \rangle$

lemma *divide-diff-eq-iff*:
 $z \neq 0 \implies x / z - y = (x - z * y) / z$
 $\langle \text{proof} \rangle$

lemma *nonzero-eq-divide-eq*: $c \neq 0 \implies a = b / c \longleftrightarrow a * c = b$
 $\langle \text{proof} \rangle$

lemma *nonzero-divide-eq-eq*: $c \neq 0 \implies b / c = a \longleftrightarrow b = a * c$
 $\langle \text{proof} \rangle$

lemma *divide-eq-imp*: $c \neq 0 \implies b = a * c \implies b / c = a$
 $\langle \text{proof} \rangle$

lemma *eq-divide-imp*: $c \neq 0 \implies a * c = b \implies a = b / c$
 $\langle \text{proof} \rangle$

lemmas *field-eq-simps*[*noatp*] = *algebra-simps*

add-divide-eq-iff divide-add-eq-iff

diff-divide-eq-iff divide-diff-eq-iff

nonzero-eq-divide-eq nonzero-divide-eq-eq

An example:

lemma $\llbracket a \neq b; c \neq d; e \neq f \rrbracket \implies ((a-b)*(c-d)*(e-f))/((c-d)*(e-f)*(a-b)) = 1$
 $\langle \text{proof} \rangle$

lemma *diff-frac-eq*:

$y \neq 0 \implies z \neq 0 \implies x / y - w / z = (x * z - w * y) / (y * z)$
 $\langle \text{proof} \rangle$

lemma *frac-eq-eq*:

$y \neq 0 \implies z \neq 0 \implies (x / y = w / z) = (x * z = w * y)$
 $\langle \text{proof} \rangle$

end

class *division-by-zero* = *zero* + *inverse* +
assumes *inverse-zero* [*simp*]: *inverse 0 = 0*

lemma *divide-zero* [*simp*]:

$a / 0 = (0 :: 'a :: \{\text{field}, \text{division-by-zero}\})$
 $\langle \text{proof} \rangle$

lemma *divide-self-if* [*simp*]:

$a / (a :: 'a :: \{\text{field}, \text{division-by-zero}\}) = (\text{if } a=0 \text{ then } 0 \text{ else } 1)$
 $\langle \text{proof} \rangle$

class *mult-mono* = *times* + *zero* + *ord* +

assumes *mult-left-mono*: $a \leq b \implies 0 \leq c \implies c * a \leq c * b$

assumes *mult-right-mono*: $a \leq b \implies 0 \leq c \implies a * c \leq b * c$

class *pordered-semiring* = *mult-mono* + *semiring-0* + *pordered-ab-semigroup-add*

begin

lemma *mult-mono*:

$a \leq b \implies c \leq d \implies 0 \leq b \implies 0 \leq c$
 $\implies a * c \leq b * d$
 $\langle \text{proof} \rangle$

lemma *mult-mono'*:

$a \leq b \implies c \leq d \implies 0 \leq a \implies 0 \leq c$
 $\implies a * c \leq b * d$
 $\langle \text{proof} \rangle$

end

```

class pordered-cancel-semiring = mult-mono + pordered-ab-semigroup-add
  + semiring + cancel-comm-monoid-add
begin

```

```

subclass semiring-0-cancel <proof>
subclass pordered-semiring <proof>

```

```

lemma mult-nonneg-nonneg:  $0 \leq a \implies 0 \leq b \implies 0 \leq a * b$ 
<proof>

```

```

lemma mult-nonneg-nonpos:  $0 \leq a \implies b \leq 0 \implies a * b \leq 0$ 
<proof>

```

```

lemma mult-nonpos-nonneg:  $a \leq 0 \implies 0 \leq b \implies a * b \leq 0$ 
<proof>

```

Legacy - use *mult-nonpos-nonneg*

```

lemma mult-nonneg-nonpos2:  $0 \leq a \implies b \leq 0 \implies b * a \leq 0$ 
<proof>

```

```

lemma split-mult-neg-le:  $(0 \leq a \ \& \ b \leq 0) \mid (a \leq 0 \ \& \ 0 \leq b) \implies a * b \leq 0$ 
<proof>

```

```

end

```

```

class ordered-semiring = semiring + comm-monoid-add + ordered-cancel-ab-semigroup-add
  + mult-mono
begin

```

```

subclass pordered-cancel-semiring <proof>

```

```

subclass pordered-comm-monoid-add <proof>

```

```

lemma mult-left-less-imp-less:
   $c * a < c * b \implies 0 \leq c \implies a < b$ 
<proof>

```

```

lemma mult-right-less-imp-less:
   $a * c < b * c \implies 0 \leq c \implies a < b$ 
<proof>

```

```

end

```

```

class ordered-semiring-strict = semiring + comm-monoid-add + ordered-cancel-ab-semigroup-add
  +
  assumes mult-strict-left-mono:  $a < b \implies 0 < c \implies c * a < c * b$ 
  assumes mult-strict-right-mono:  $a < b \implies 0 < c \implies a * c < b * c$ 
begin

```

subclass *semiring-0-cancel* $\langle \text{proof} \rangle$

subclass *ordered-semiring*
 $\langle \text{proof} \rangle$

lemma *mult-left-le-imp-le*:
 $c * a \leq c * b \implies 0 < c \implies a \leq b$
 $\langle \text{proof} \rangle$

lemma *mult-right-le-imp-le*:
 $a * c \leq b * c \implies 0 < c \implies a \leq b$
 $\langle \text{proof} \rangle$

lemma *mult-pos-pos*: $0 < a \implies 0 < b \implies 0 < a * b$
 $\langle \text{proof} \rangle$

lemma *mult-pos-neg*: $0 < a \implies b < 0 \implies a * b < 0$
 $\langle \text{proof} \rangle$

lemma *mult-neg-pos*: $a < 0 \implies 0 < b \implies a * b < 0$
 $\langle \text{proof} \rangle$

Legacy - use *mult-neg-pos*

lemma *mult-pos-neg2*: $0 < a \implies b < 0 \implies b * a < 0$
 $\langle \text{proof} \rangle$

lemma *zero-less-mult-pos*:
 $0 < a * b \implies 0 < a \implies 0 < b$
 $\langle \text{proof} \rangle$

lemma *zero-less-mult-pos2*:
 $0 < b * a \implies 0 < a \implies 0 < b$
 $\langle \text{proof} \rangle$

Strict monotonicity in both arguments

lemma *mult-strict-mono*:
assumes $a < b$ **and** $c < d$ **and** $0 < b$ **and** $0 \leq c$
shows $a * c < b * d$
 $\langle \text{proof} \rangle$

This weaker variant has more natural premises

lemma *mult-strict-mono'*:
assumes $a < b$ **and** $c < d$ **and** $0 \leq a$ **and** $0 \leq c$
shows $a * c < b * d$
 $\langle \text{proof} \rangle$

lemma *mult-less-le-imp-less*:
assumes $a < b$ **and** $c \leq d$ **and** $0 \leq a$ **and** $0 < c$
shows $a * c < b * d$

$\langle proof \rangle$

lemma *mult-le-less-imp-less*:

assumes $a \leq b$ **and** $c < d$ **and** $0 < a$ **and** $0 \leq c$

shows $a * c < b * d$

$\langle proof \rangle$

lemma *mult-less-imp-less-left*:

assumes *less*: $c * a < c * b$ **and** *nonneg*: $0 \leq c$

shows $a < b$

$\langle proof \rangle$

lemma *mult-less-imp-less-right*:

assumes *less*: $a * c < b * c$ **and** *nonneg*: $0 \leq c$

shows $a < b$

$\langle proof \rangle$

end

class *mult-mono1* = *times* + *zero* + *ord* +

assumes *mult-mono1*: $a \leq b \implies 0 \leq c \implies c * a \leq c * b$

class *pordered-comm-semiring* = *comm-semiring-0*

+ *pordered-ab-semigroup-add* + *mult-mono1*

begin

subclass *pordered-semiring*

$\langle proof \rangle$

end

class *pordered-cancel-comm-semiring* = *comm-semiring-0-cancel*

+ *pordered-ab-semigroup-add* + *mult-mono1*

begin

subclass *pordered-comm-semiring* $\langle proof \rangle$

subclass *pordered-cancel-semiring* $\langle proof \rangle$

end

class *ordered-comm-semiring-strict* = *comm-semiring-0* + *ordered-cancel-ab-semigroup-add*

+

assumes *mult-strict-left-mono-comm*: $a < b \implies 0 < c \implies c * a < c * b$

begin

subclass *ordered-semiring-strict*

$\langle proof \rangle$

subclass *pordered-cancel-comm-semiring*

$\langle proof \rangle$

end

class *pordered-ring* = *ring* + *pordered-cancel-semiring*
begin

subclass *pordered-ab-group-add* $\langle proof \rangle$

Legacy - use *algebra-simps*

lemmas *ring-simps*[*noatp*] = *algebra-simps*

lemma *less-add-iff1*:

$a * e + c < b * e + d \longleftrightarrow (a - b) * e + c < d$
 $\langle proof \rangle$

lemma *less-add-iff2*:

$a * e + c < b * e + d \longleftrightarrow c < (b - a) * e + d$
 $\langle proof \rangle$

lemma *le-add-iff1*:

$a * e + c \leq b * e + d \longleftrightarrow (a - b) * e + c \leq d$
 $\langle proof \rangle$

lemma *le-add-iff2*:

$a * e + c \leq b * e + d \longleftrightarrow c \leq (b - a) * e + d$
 $\langle proof \rangle$

lemma *mult-left-mono-neg*:

$b \leq a \implies c \leq 0 \implies c * a \leq c * b$
 $\langle proof \rangle$

lemma *mult-right-mono-neg*:

$b \leq a \implies c \leq 0 \implies a * c \leq b * c$
 $\langle proof \rangle$

lemma *mult-nonpos-nonpos*: $a \leq 0 \implies b \leq 0 \implies 0 \leq a * b$

$\langle proof \rangle$

lemma *split-mult-pos-le*:

$(0 \leq a \wedge 0 \leq b) \vee (a \leq 0 \wedge b \leq 0) \implies 0 \leq a * b$
 $\langle proof \rangle$

end

class *abs-if* = *minus* + *uminus* + *ord* + *zero* + *abs* +
assumes *abs-if*: $|a| = (\text{if } a < 0 \text{ then } -a \text{ else } a)$

class *sgn-if* = *minus* + *uminus* + *zero* + *one* + *ord* + *sgn* +


```

assumes sgn-if: sgn x = (if x = 0 then 0 else if 0 < x then 1 else - 1)

lemma (in sgn-if) sgn0[simp]: sgn 0 = 0
<proof>

class ordered-ring = ring + ordered-semiring
  + ordered-ab-group-add + abs-if
begin

subclass pordered-ring <proof>

subclass pordered-ab-group-add-abs
<proof>

end

class ordered-ring-strict = ring + ordered-semiring-strict
  + ordered-ab-group-add + abs-if
begin

subclass ordered-ring <proof>

lemma mult-strict-left-mono-neg:  $b < a \implies c < 0 \implies c * a < c * b$ 
<proof>

lemma mult-strict-right-mono-neg:  $b < a \implies c < 0 \implies a * c < b * c$ 
<proof>

lemma mult-neg-neg:  $a < 0 \implies b < 0 \implies 0 < a * b$ 
<proof>

subclass ring-no-zero-divisors
<proof>

lemma zero-less-mult-iff:
 $0 < a * b \iff 0 < a \wedge 0 < b \vee a < 0 \wedge b < 0$ 
<proof>

lemma zero-le-mult-iff:
 $0 \leq a * b \iff 0 \leq a \wedge 0 \leq b \vee a \leq 0 \wedge b \leq 0$ 
<proof>

lemma mult-less-0-iff:
 $a * b < 0 \iff 0 < a \wedge b < 0 \vee a < 0 \wedge 0 < b$ 
<proof>

lemma mult-le-0-iff:
 $a * b \leq 0 \iff 0 \leq a \wedge b \leq 0 \vee a \leq 0 \wedge 0 \leq b$ 

```

$\langle \text{proof} \rangle$

lemma *zero-le-square* [simp]: $0 \leq a * a$
 $\langle \text{proof} \rangle$

lemma *not-square-less-zero* [simp]: $\neg (a * a < 0)$
 $\langle \text{proof} \rangle$

Cancellation laws for $c * a < c * b$ and $a * c < b * c$, also with the relations \leq and equality.

These “disjunction” versions produce two cases when the comparison is an assumption, but effectively four when the comparison is a goal.

lemma *mult-less-cancel-right-disj*:
 $a * c < b * c \longleftrightarrow 0 < c \wedge a < b \vee c < 0 \wedge b < a$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-left-disj*:
 $c * a < c * b \longleftrightarrow 0 < c \wedge a < b \vee c < 0 \wedge b < a$
 $\langle \text{proof} \rangle$

The “conjunction of implication” lemmas produce two cases when the comparison is a goal, but give four when the comparison is an assumption.

lemma *mult-less-cancel-right*:
 $a * c < b * c \longleftrightarrow (0 \leq c \longrightarrow a < b) \wedge (c \leq 0 \longrightarrow b < a)$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-left*:
 $c * a < c * b \longleftrightarrow (0 \leq c \longrightarrow a < b) \wedge (c \leq 0 \longrightarrow b < a)$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-right*:
 $a * c \leq b * c \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-left*:
 $c * a \leq c * b \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-left-pos*:
 $0 < c \implies c * a \leq c * b \longleftrightarrow a \leq b$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-left-neg*:
 $c < 0 \implies c * a \leq c * b \longleftrightarrow b \leq a$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-left-pos*:
 $0 < c \implies c * a < c * b \longleftrightarrow a < b$

⟨proof⟩

lemma *mult-less-cancel-left-neg*:

$c < 0 \implies c * a < c * b \longleftrightarrow b < a$

⟨proof⟩

end

Legacy - use *algebra-simps*

lemmas *ring-simps*[noatp] = *algebra-simps*

lemmas *mult-sign-intros* =

mult-nonneg-nonneg mult-nonneg-nonpos

mult-nonpos-nonneg mult-nonpos-nonpos

mult-pos-pos mult-pos-neg

mult-neg-pos mult-neg-neg

class *pordered-comm-ring* = *comm-ring* + *pordered-comm-semiring*

begin

subclass *pordered-ring* ⟨proof⟩

subclass *pordered-cancel-comm-semiring* ⟨proof⟩

end

class *ordered-semidom* = *comm-semiring-1-cancel* + *ordered-comm-semiring-strict*

+

assumes *zero-less-one* [simp]: $0 < 1$

begin

lemma *pos-add-strict*:

shows $0 < a \implies b < c \implies b < a + c$

⟨proof⟩

lemma *zero-le-one* [simp]: $0 \leq 1$

⟨proof⟩

lemma *not-one-le-zero* [simp]: $\neg 1 \leq 0$

⟨proof⟩

lemma *not-one-less-zero* [simp]: $\neg 1 < 0$

⟨proof⟩

lemma *less-1-mult*:

assumes $1 < m$ **and** $1 < n$

shows $1 < m * n$

⟨proof⟩

end

class *ordered-idom* = *comm-ring-1* +
ordered-comm-semiring-strict + *ordered-ab-group-add* +
abs-if + *sgn-if*

begin

subclass *ordered-ring-strict* $\langle \text{proof} \rangle$
subclass *pordered-comm-ring* $\langle \text{proof} \rangle$
subclass *idom* $\langle \text{proof} \rangle$

subclass *ordered-semidom*
 $\langle \text{proof} \rangle$

lemma *linorder-negE-ordered-idom*:
assumes $x \neq y$ **obtains** $x < y \mid y < x$
 $\langle \text{proof} \rangle$

These cancellation simprules also produce two cases when the comparison is a goal.

lemma *mult-le-cancel-right1*:
 $c \leq b * c \longleftrightarrow (0 < c \longrightarrow 1 \leq b) \wedge (c < 0 \longrightarrow b \leq 1)$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-right2*:
 $a * c \leq c \longleftrightarrow (0 < c \longrightarrow a \leq 1) \wedge (c < 0 \longrightarrow 1 \leq a)$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-left1*:
 $c \leq c * b \longleftrightarrow (0 < c \longrightarrow 1 \leq b) \wedge (c < 0 \longrightarrow b \leq 1)$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-left2*:
 $c * a \leq c \longleftrightarrow (0 < c \longrightarrow a \leq 1) \wedge (c < 0 \longrightarrow 1 \leq a)$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-right1*:
 $c < b * c \longleftrightarrow (0 \leq c \longrightarrow 1 < b) \wedge (c \leq 0 \longrightarrow b < 1)$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-right2*:
 $a * c < c \longleftrightarrow (0 \leq c \longrightarrow a < 1) \wedge (c \leq 0 \longrightarrow 1 < a)$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-left1*:
 $c < c * b \longleftrightarrow (0 \leq c \longrightarrow 1 < b) \wedge (c \leq 0 \longrightarrow b < 1)$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel-left2*:

$$c * a < c \longleftrightarrow (0 \leq c \longrightarrow a < 1) \wedge (c \leq 0 \longrightarrow 1 < a)$$

<proof>

lemma *sgn-sgn [simp]*:

$$\text{sgn } (\text{sgn } a) = \text{sgn } a$$

<proof>

lemma *sgn-0-0*:

$$\text{sgn } a = 0 \longleftrightarrow a = 0$$

<proof>

lemma *sgn-1-pos*:

$$\text{sgn } a = 1 \longleftrightarrow a > 0$$

<proof>

lemma *sgn-1-neg*:

$$\text{sgn } a = -1 \longleftrightarrow a < 0$$

<proof>

lemma *sgn-pos [simp]*:

$$0 < a \implies \text{sgn } a = 1$$

<proof>

lemma *sgn-neg [simp]*:

$$a < 0 \implies \text{sgn } a = -1$$

<proof>

lemma *sgn-times*:

$$\text{sgn } (a * b) = \text{sgn } a * \text{sgn } b$$

<proof>

lemma *abs-sgn*: $\text{abs } k = k * \text{sgn } k$

<proof>

lemma *sgn-greater [simp]*:

$$0 < \text{sgn } a \longleftrightarrow 0 < a$$

<proof>

lemma *sgn-less [simp]*:

$$\text{sgn } a < 0 \longleftrightarrow a < 0$$

<proof>

lemma *abs-dvd-iff [simp]*: $(\text{abs } m) \text{ dvd } k \longleftrightarrow m \text{ dvd } k$

<proof>

lemma *dvd-abs-iff [simp]*: $m \text{ dvd } (\text{abs } k) \longleftrightarrow m \text{ dvd } k$

<proof>

end

class *ordered-field* = *field* + *ordered-idom*

Simprules for comparisons where common factors can be cancelled.

lemmas *mult-compare-simps*[*noatp*] =
 mult-le-cancel-right mult-le-cancel-left
 mult-le-cancel-right1 mult-le-cancel-right2
 mult-le-cancel-left1 mult-le-cancel-left2
 mult-less-cancel-right mult-less-cancel-left
 mult-less-cancel-right1 mult-less-cancel-right2
 mult-less-cancel-left1 mult-less-cancel-left2
 mult-cancel-right mult-cancel-left
 mult-cancel-right1 mult-cancel-right2
 mult-cancel-left1 mult-cancel-left2

— FIXME continue localization here

lemma *inverse-nonzero-iff-nonzero* [*simp*]:
 (*inverse a* = 0) = (*a* = (0::'a::{*division-ring,division-by-zero*}))
 ⟨*proof*⟩

lemma *inverse-minus-eq* [*simp*]:
 inverse(−*a*) = −*inverse*(*a*::'a::{*division-ring,division-by-zero*})
 ⟨*proof*⟩

lemma *inverse-eq-imp-eq*:
 inverse a = *inverse b* ==> *a* = (*b*::'a::{*division-ring,division-by-zero*})
 ⟨*proof*⟩

lemma *inverse-eq-iff-eq* [*simp*]:
 (*inverse a* = *inverse b*) = (*a* = (*b*::'a::{*division-ring,division-by-zero*}))
 ⟨*proof*⟩

lemma *inverse-inverse-eq* [*simp*]:
 inverse(*inverse* (*a*::'a::{*division-ring,division-by-zero*})) = *a*
 ⟨*proof*⟩

This version builds in division by zero while also re-orienting the right-hand side.

lemma *inverse-mult-distrib* [*simp*]:
 inverse(*a***b*) = *inverse*(*a*) * *inverse*(*b*::'a::{*field,division-by-zero*})
 ⟨*proof*⟩

lemma *inverse-divide* [*simp*]:
 inverse (*a*/*b*) = *b* / (*a*::'a::{*field,division-by-zero*})
 ⟨*proof*⟩

11.1 Calculations with fractions

There is a whole bunch of simp-rules just for class *field* but none for class *field* and *nonzero-divides* because the latter are covered by a simproc.

lemma *mult-divide-mult-cancel-left*:

$c \neq 0 \implies (c*a) / (c*b) = a / (b::'a::\{field, division-by-zero\})$
 $\langle proof \rangle$

lemma *mult-divide-mult-cancel-right*:

$c \neq 0 \implies (a*c) / (b*c) = a / (b::'a::\{field, division-by-zero\})$
 $\langle proof \rangle$

lemma *divide-divide-eq-right* [simp, noatp]:

$a / (b/c) = (a*c) / (b::'a::\{field, division-by-zero\})$
 $\langle proof \rangle$

lemma *divide-divide-eq-left* [simp, noatp]:

$(a / b) / (c::'a::\{field, division-by-zero\}) = a / (b*c)$
 $\langle proof \rangle$

11.1.1 Special Cancellation Simprules for Division

lemma *mult-divide-mult-cancel-left-if* [simp, noatp]:

fixes $c :: 'a :: \{field, division-by-zero\}$
shows $(c*a) / (c*b) = (if\ c=0\ then\ 0\ else\ a/b)$
 $\langle proof \rangle$

11.2 Division and Unary Minus

lemma *minus-divide-right*: $-(a/b) = a / -(b::'a::\{field, division-by-zero\})$
 $\langle proof \rangle$

lemma *divide-minus-right* [simp, noatp]:

$a / -(b::'a::\{field, division-by-zero\}) = -(a / b)$
 $\langle proof \rangle$

lemma *minus-divide-divide*:

$(-a)/(-b) = a / (b::'a::\{field, division-by-zero\})$
 $\langle proof \rangle$

lemma *eq-divide-eq*:

$((a::'a::\{field, division-by-zero\}) = b/c) = (if\ c \neq 0\ then\ a*c = b\ else\ a=0)$
 $\langle proof \rangle$

lemma *divide-eq-eq*:

$(b/c = (a::'a::\{field, division-by-zero\})) = (if\ c \neq 0\ then\ b = a*c\ else\ a=0)$
 $\langle proof \rangle$

11.3 Ordered Fields

lemma *positive-imp-inverse-positive*:

assumes *a-gt-0*: $0 < a$ **shows** $0 < \text{inverse } (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *negative-imp-inverse-negative*:

$a < 0 ==> \text{inverse } a < (0::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-le-imp-le*:

assumes *invle*: $\text{inverse } a \leq \text{inverse } b$ **and** *apos*: $0 < a$
shows $b \leq (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-positive-imp-positive*:

assumes *inv-gt-0*: $0 < \text{inverse } a$ **and** *nz*: $a \neq 0$
shows $0 < (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-positive-iff-positive* [simp]:

$(0 < \text{inverse } a) = (0 < (a::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *inverse-negative-imp-negative*:

assumes *inv-less-0*: $\text{inverse } a < 0$ **and** *nz*: $a \neq 0$
shows $a < (0::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-negative-iff-negative* [simp]:

$(\text{inverse } a < 0) = (a < (0::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *inverse-nonnegative-iff-nonnegative* [simp]:

$(0 \leq \text{inverse } a) = (0 \leq (a::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *inverse-nonpositive-iff-nonpositive* [simp]:

$(\text{inverse } a \leq 0) = (a \leq (0::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *ordered-field-no-lb*: $\forall x. \exists y. y < (x::'a::\text{ordered-field})$

$\langle \text{proof} \rangle$

lemma *ordered-field-no-ub*: $\forall x. \exists y. y > (x::'a::\text{ordered-field})$

$\langle \text{proof} \rangle$

11.4 Anti-Monotonicity of *inverse*

lemma *less-imp-inverse-less*:

assumes *less*: $a < b$ **and** *apos*: $0 < a$
shows $\text{inverse } b < \text{inverse } (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-less-imp-less*:
 $[[\text{inverse } a < \text{inverse } b; 0 < a]] \implies b < (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

Both premises are essential. Consider -1 and 1.

lemma *inverse-less-iff-less* [*simp, noatp*]:
 $[[0 < a; 0 < b]] \implies (\text{inverse } a < \text{inverse } b) = (b < (a::'a::\text{ordered-field}))$
 $\langle \text{proof} \rangle$

lemma *le-imp-inverse-le*:
 $[[a \leq b; 0 < a]] \implies \text{inverse } b \leq \text{inverse } (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-le-iff-le* [*simp, noatp*]:
 $[[0 < a; 0 < b]] \implies (\text{inverse } a \leq \text{inverse } b) = (b \leq (a::'a::\text{ordered-field}))$
 $\langle \text{proof} \rangle$

These results refer to both operands being negative. The opposite-sign case is trivial, since inverse preserves signs.

lemma *inverse-le-imp-le-neg*:
 $[[\text{inverse } a \leq \text{inverse } b; b < 0]] \implies b \leq (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *less-imp-inverse-less-neg*:
 $[[a < b; b < 0]] \implies \text{inverse } b < \text{inverse } (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-less-imp-less-neg*:
 $[[\text{inverse } a < \text{inverse } b; b < 0]] \implies b < (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-less-iff-less-neg* [*simp, noatp*]:
 $[[a < 0; b < 0]] \implies (\text{inverse } a < \text{inverse } b) = (b < (a::'a::\text{ordered-field}))$
 $\langle \text{proof} \rangle$

lemma *le-imp-inverse-le-neg*:
 $[[a \leq b; b < 0]] \implies \text{inverse } b \leq \text{inverse } (a::'a::\text{ordered-field})$
 $\langle \text{proof} \rangle$

lemma *inverse-le-iff-le-neg* [*simp, noatp*]:
 $[[a < 0; b < 0]] \implies (\text{inverse } a \leq \text{inverse } b) = (b \leq (a::'a::\text{ordered-field}))$
 $\langle \text{proof} \rangle$

11.5 Inverses and the Number One

lemma *one-less-inverse-iff*:

$(1 < \text{inverse } x) = (0 < x \ \& \ x < (1::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *inverse-eq-1-iff* [simp]:

$(\text{inverse } x = 1) = (x = (1::'a::\{\text{field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *one-le-inverse-iff*:

$(1 \leq \text{inverse } x) = (0 < x \ \& \ x \leq (1::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *inverse-less-1-iff*:

$(\text{inverse } x < 1) = (x \leq 0 \mid 1 < (x::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *inverse-le-1-iff*:

$(\text{inverse } x \leq 1) = (x \leq 0 \mid 1 \leq (x::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

11.6 Simplification of Inequalities Involving Literal Divisors

lemma *pos-le-divide-eq*: $0 < (c::'a::\text{ordered-field}) \implies (a \leq b/c) = (a*c \leq b)$

$\langle \text{proof} \rangle$

lemma *neg-le-divide-eq*: $c < (0::'a::\text{ordered-field}) \implies (a \leq b/c) = (b \leq a*c)$

$\langle \text{proof} \rangle$

lemma *le-divide-eq*:

$(a \leq b/c) =$
 $(\text{if } 0 < c \text{ then } a*c \leq b$
 $\quad \text{else if } c < 0 \text{ then } b \leq a*c$
 $\quad \text{else } a \leq (0::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$

$\langle \text{proof} \rangle$

lemma *pos-divide-le-eq*: $0 < (c::'a::\text{ordered-field}) \implies (b/c \leq a) = (b \leq a*c)$

$\langle \text{proof} \rangle$

lemma *neg-divide-le-eq*: $c < (0::'a::\text{ordered-field}) \implies (b/c \leq a) = (a*c \leq b)$

$\langle \text{proof} \rangle$

lemma *divide-le-eq*:

$(b/c \leq a) =$
 $(\text{if } 0 < c \text{ then } b \leq a*c$
 $\quad \text{else if } c < 0 \text{ then } a*c \leq b$
 $\quad \text{else } 0 \leq (a::'a::\{\text{ordered-field}, \text{division-by-zero}\}))$

$\langle \text{proof} \rangle$

lemma *pos-less-divide-eq*:

$0 < (c :: 'a :: \text{ordered-field}) \implies (a < b/c) = (a*c < b)$
 $\langle \text{proof} \rangle$

lemma *neg-less-divide-eq*:

$c < (0 :: 'a :: \text{ordered-field}) \implies (a < b/c) = (b < a*c)$
 $\langle \text{proof} \rangle$

lemma *less-divide-eq*:

$(a < b/c) =$
 $(\text{if } 0 < c \text{ then } a*c < b$
 $\quad \text{else if } c < 0 \text{ then } b < a*c$
 $\quad \text{else } a < (0 :: 'a :: \{\text{ordered-field, division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *pos-divide-less-eq*:

$0 < (c :: 'a :: \text{ordered-field}) \implies (b/c < a) = (b < a*c)$
 $\langle \text{proof} \rangle$

lemma *neg-divide-less-eq*:

$c < (0 :: 'a :: \text{ordered-field}) \implies (b/c < a) = (a*c < b)$
 $\langle \text{proof} \rangle$

lemma *divide-less-eq*:

$(b/c < a) =$
 $(\text{if } 0 < c \text{ then } b < a*c$
 $\quad \text{else if } c < 0 \text{ then } a*c < b$
 $\quad \text{else } 0 < (a :: 'a :: \{\text{ordered-field, division-by-zero}\}))$
 $\langle \text{proof} \rangle$

11.7 Field simplification

Lemmas *field-simps* multiply with denominators in in(equations) if they can be proved to be non-zero (for equations) or positive/negative (for inequations). Can be too aggressive and is therefore separate from the more benign *algebra-simps*.

lemmas *field-simps*[noatp] = *field-eq-simps*

pos-divide-less-eq neg-divide-less-eq
pos-less-divide-eq neg-less-divide-eq
pos-divide-le-eq neg-divide-le-eq
pos-le-divide-eq neg-le-divide-eq

Lemmas *sign-simps* is a first attempt to automate proofs of positivity/negativity needed for *field-simps*. Have not added *sign-simps* to *field-simps* because the former can lead to case explosions.

lemmas *sign-simps*[noatp] = *group-simps*

zero-less-mult-iff mult-less-0-iff

11.8 Division and Signs

lemma *zero-less-divide-iff*:

$$((0::'a::\{\text{ordered-field}, \text{division-by-zero}\}) < a/b) = (0 < a \ \& \ 0 < b \mid a < 0 \ \& \ b < 0)$$

<proof>

lemma *divide-less-0-iff*:

$$(a/b < (0::'a::\{\text{ordered-field}, \text{division-by-zero}\})) = (0 < a \ \& \ b < 0 \mid a < 0 \ \& \ 0 < b)$$

<proof>

lemma *zero-le-divide-iff*:

$$((0::'a::\{\text{ordered-field}, \text{division-by-zero}\}) \leq a/b) = (0 \leq a \ \& \ 0 \leq b \mid a \leq 0 \ \& \ b \leq 0)$$

<proof>

lemma *divide-le-0-iff*:

$$(a/b \leq (0::'a::\{\text{ordered-field}, \text{division-by-zero}\})) = (0 \leq a \ \& \ b \leq 0 \mid a \leq 0 \ \& \ 0 \leq b)$$

<proof>

lemma *divide-eq-0-iff* [simp, noatp]:

$$(a/b = 0) = (a=0 \mid b=(0::'a::\{\text{field}, \text{division-by-zero}\}))$$

<proof>

lemma *divide-pos-pos*:

$$0 < (x::'a::\text{ordered-field}) \implies 0 < y \implies 0 < x / y$$

<proof>

lemma *divide-nonneg-pos*:

$$0 \leq (x::'a::\text{ordered-field}) \implies 0 < y \implies 0 \leq x / y$$

<proof>

lemma *divide-neg-pos*:

$$(x::'a::\text{ordered-field}) < 0 \implies 0 < y \implies x / y < 0$$

<proof>

lemma *divide-nonpos-pos*:

$$(x::'a::\text{ordered-field}) \leq 0 \implies 0 < y \implies x / y \leq 0$$

<proof>

lemma *divide-pos-neg*:

$$0 < (x::'a::\text{ordered-field}) \implies y < 0 \implies x / y < 0$$

<proof>

lemma *divide-nonneg-neg*:

$$0 \leq (x::'a::\text{ordered-field}) \implies y < 0 \implies x / y \leq 0$$

<proof>

lemma *divide-neg-neg*:

$(x::'a::\text{ordered-field}) < 0 \implies y < 0 \implies 0 < x / y$
 $\langle \text{proof} \rangle$

lemma *divide-nonpos-neg*:

$(x::'a::\text{ordered-field}) \leq 0 \implies y < 0 \implies 0 \leq x / y$
 $\langle \text{proof} \rangle$

11.9 Cancellation Laws for Division

lemma *divide-cancel-right* [*simp, noatp*]:

$(a/c = b/c) = (c = 0 \mid a = (b::'a::\{\text{field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *divide-cancel-left* [*simp, noatp*]:

$(c/a = c/b) = (c = 0 \mid a = (b::'a::\{\text{field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

11.10 Division and the Number One

Simplify expressions equated with 1

lemma *divide-eq-1-iff* [*simp, noatp*]:

$(a/b = 1) = (b \neq 0 \ \& \ a = (b::'a::\{\text{field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *one-eq-divide-iff* [*simp, noatp*]:

$(1 = a/b) = (b \neq 0 \ \& \ a = (b::'a::\{\text{field}, \text{division-by-zero}\}))$
 $\langle \text{proof} \rangle$

lemma *zero-eq-1-divide-iff* [*simp, noatp*]:

$((0::'a::\{\text{ordered-field}, \text{division-by-zero}\}) = 1/a) = (a = 0)$
 $\langle \text{proof} \rangle$

lemma *one-divide-eq-0-iff* [*simp, noatp*]:

$(1/a = (0::'a::\{\text{ordered-field}, \text{division-by-zero}\})) = (a = 0)$
 $\langle \text{proof} \rangle$

Simplify expressions such as $0 < 1/x$ to $0 < x$

lemmas *zero-less-divide-1-iff* = *zero-less-divide-iff* [*of 1, simplified*]

lemmas *divide-less-0-1-iff* = *divide-less-0-iff* [*of 1, simplified*]

lemmas *zero-le-divide-1-iff* = *zero-le-divide-iff* [*of 1, simplified*]

lemmas *divide-le-0-1-iff* = *divide-le-0-iff* [*of 1, simplified*]

declare *zero-less-divide-1-iff* [*simp, noatp*]

declare *divide-less-0-1-iff* [*simp, noatp*]

declare *zero-le-divide-1-iff* [*simp, noatp*]

declare *divide-le-0-1-iff* [*simp, noatp*]

11.11 Ordering Rules for Division

lemma *divide-strict-right-mono*:

$\llbracket a < b; 0 < c \rrbracket \implies a / c < b / c \text{ (} c :: 'a :: \text{ordered-field} \text{)}$
 $\langle \text{proof} \rangle$

lemma *divide-right-mono*:

$\llbracket a \leq b; 0 \leq c \rrbracket \implies a / c \leq b / c \text{ (} c :: 'a :: \{ \text{ordered-field}, \text{division-by-zero} \} \text{)}$
 $\langle \text{proof} \rangle$

lemma *divide-right-mono-neg*: $(a :: 'a :: \{ \text{division-by-zero}, \text{ordered-field} \}) <= b$

$\implies c <= 0 \implies b / c <= a / c$
 $\langle \text{proof} \rangle$

lemma *divide-strict-right-mono-neg*:

$\llbracket b < a; c < 0 \rrbracket \implies a / c < b / c \text{ (} c :: 'a :: \text{ordered-field} \text{)}$
 $\langle \text{proof} \rangle$

The last premise ensures that a and b have the same sign

lemma *divide-strict-left-mono*:

$\llbracket b < a; 0 < c; 0 < a * b \rrbracket \implies c / a < c / b \text{ (} b :: 'a :: \text{ordered-field} \text{)}$
 $\langle \text{proof} \rangle$

lemma *divide-left-mono*:

$\llbracket b \leq a; 0 \leq c; 0 < a * b \rrbracket \implies c / a \leq c / b \text{ (} b :: 'a :: \text{ordered-field} \text{)}$
 $\langle \text{proof} \rangle$

lemma *divide-left-mono-neg*: $(a :: 'a :: \{ \text{division-by-zero}, \text{ordered-field} \}) <= b$

$\implies c <= 0 \implies 0 < a * b \implies c / a <= c / b$
 $\langle \text{proof} \rangle$

lemma *divide-strict-left-mono-neg*:

$\llbracket a < b; c < 0; 0 < a * b \rrbracket \implies c / a < c / b \text{ (} b :: 'a :: \text{ordered-field} \text{)}$
 $\langle \text{proof} \rangle$

Simplify quotients that are compared with the value 1.

lemma *le-divide-eq-1* [noatp]:

fixes $a :: 'a :: \{ \text{ordered-field}, \text{division-by-zero} \}$
shows $(1 \leq b / a) = ((0 < a \ \& \ a \leq b) \mid (a < 0 \ \& \ b \leq a))$
 $\langle \text{proof} \rangle$

lemma *divide-le-eq-1* [noatp]:

fixes $a :: 'a :: \{ \text{ordered-field}, \text{division-by-zero} \}$
shows $(b / a \leq 1) = ((0 < a \ \& \ b \leq a) \mid (a < 0 \ \& \ a \leq b) \mid a = 0)$
 $\langle \text{proof} \rangle$

lemma *less-divide-eq-1* [noatp]:

fixes $a :: 'a :: \{ \text{ordered-field}, \text{division-by-zero} \}$
shows $(1 < b / a) = ((0 < a \ \& \ a < b) \mid (a < 0 \ \& \ b < a))$
 $\langle \text{proof} \rangle$

lemma *divide-less-eq-1* [*noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $(b / a < 1) = ((0 < a \ \& \ b < a) \mid (a < 0 \ \& \ a < b) \mid a=0)$
 $\langle \text{proof} \rangle$

11.12 Conditional Simplification Rules: No Case Splits

lemma *le-divide-eq-1-pos* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $0 < a \implies (1 \leq b/a) = (a \leq b)$
 $\langle \text{proof} \rangle$

lemma *le-divide-eq-1-neg* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $a < 0 \implies (1 \leq b/a) = (b \leq a)$
 $\langle \text{proof} \rangle$

lemma *divide-le-eq-1-pos* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $0 < a \implies (b/a \leq 1) = (b \leq a)$
 $\langle \text{proof} \rangle$

lemma *divide-le-eq-1-neg* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $a < 0 \implies (b/a \leq 1) = (a \leq b)$
 $\langle \text{proof} \rangle$

lemma *less-divide-eq-1-pos* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $0 < a \implies (1 < b/a) = (a < b)$
 $\langle \text{proof} \rangle$

lemma *less-divide-eq-1-neg* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $a < 0 \implies (1 < b/a) = (b < a)$
 $\langle \text{proof} \rangle$

lemma *divide-less-eq-1-pos* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $0 < a \implies (b/a < 1) = (b < a)$
 $\langle \text{proof} \rangle$

lemma *divide-less-eq-1-neg* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $a < 0 \implies b/a < 1 \iff a < b$
 $\langle \text{proof} \rangle$

lemma *eq-divide-eq-1* [*simp, noatp*]:

fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $(1 = b/a) = ((a \neq 0 \ \& \ a = b))$
 $\langle \text{proof} \rangle$

lemma *divide-eq-eq-1* [*simp, noatp*]:
fixes $a :: 'a :: \{\text{ordered-field}, \text{division-by-zero}\}$
shows $(b/a = 1) = ((a \neq 0 \ \& \ a = b))$
 $\langle \text{proof} \rangle$

11.13 Reasoning about inequalities with division

lemma *mult-right-le-one-le*: $0 \leq (x :: 'a :: \text{ordered-idom}) \implies 0 \leq y \implies y \leq 1 \implies x * y \leq x$
 $\langle \text{proof} \rangle$

lemma *mult-left-le-one-le*: $0 \leq (x :: 'a :: \text{ordered-idom}) \implies 0 \leq y \implies y \leq 1 \implies y * x \leq x$
 $\langle \text{proof} \rangle$

lemma *mult-imp-div-pos-le*: $0 < (y :: 'a :: \text{ordered-field}) \implies x \leq z * y \implies x / y \leq z$
 $\langle \text{proof} \rangle$

lemma *mult-imp-le-div-pos*: $0 < (y :: 'a :: \text{ordered-field}) \implies z * y \leq x \implies z \leq x / y$
 $\langle \text{proof} \rangle$

lemma *mult-imp-div-pos-less*: $0 < (y :: 'a :: \text{ordered-field}) \implies x < z * y \implies x / y < z$
 $\langle \text{proof} \rangle$

lemma *mult-imp-less-div-pos*: $0 < (y :: 'a :: \text{ordered-field}) \implies z * y < x \implies z < x / y$
 $\langle \text{proof} \rangle$

lemma *frac-le*: $(0 :: 'a :: \text{ordered-field}) \leq x \implies x \leq y \implies 0 < w \implies w \leq z \implies x / z \leq y / w$
 $\langle \text{proof} \rangle$

lemma *frac-less*: $(0 :: 'a :: \text{ordered-field}) < x \implies x < y \implies 0 < w \implies w \leq z \implies x / z < y / w$
 $\langle \text{proof} \rangle$

lemma *frac-less2*: $(0 :: 'a :: \text{ordered-field}) < x \implies x \leq y \implies 0 < w \implies w < z \implies x / z < y / w$
 $\langle \text{proof} \rangle$

It's not obvious whether these should be *simprules* or not. Their effect is

to gather terms into one big fraction, like $a*b*c / x*y*z$. The rationale for that is unclear, but many proofs seem to need them.

declare *times-divide-eq* [*simp*]

11.14 Ordered Fields are Dense

context *ordered-semidom*
begin

lemma *less-add-one*: $a < a + 1$
<proof>

lemma *zero-less-two*: $0 < 1 + 1$
<proof>

end

lemma *less-half-sum*: $a < b \implies a < (a+b) / (1+1::'a::ordered-field)$
<proof>

lemma *gt-half-sum*: $a < b \implies (a+b)/(1+1::'a::ordered-field) < b$
<proof>

instance *ordered-field < dense-linear-order*
<proof>

11.15 Absolute Value

context *ordered-idom*
begin

lemma *mult-sgn-abs*: $\text{sgn } x * \text{abs } x = x$
<proof>

end

lemma *abs-one* [*simp*]: $\text{abs } 1 = (1::'a::ordered-idom)$
<proof>

class *pordered-ring-abs* = *pordered-ring* + *pordered-ab-group-add-abs* +
assumes *abs-eq-mult*:
 $(0 \leq a \vee a \leq 0) \wedge (0 \leq b \vee b \leq 0) \implies |a * b| = |a| * |b|$

class *lordered-ring* = *pordered-ring* + *lordered-ab-group-add-abs*
begin

subclass *lordered-ab-group-add-meet* *<proof>*
subclass *lordered-ab-group-add-join* *<proof>*

end

lemma *abs-le-mult*: $\text{abs } (a * b) \leq (\text{abs } a) * (\text{abs } (b::'a::\text{lordered-ring}))$
 $\langle \text{proof} \rangle$

instance *lordered-ring* \subseteq *pordered-ring-abs*
 $\langle \text{proof} \rangle$

instance *ordered-idom* \subseteq *pordered-ring-abs*
 $\langle \text{proof} \rangle$

lemma *abs-mult*: $\text{abs } (a * b) = \text{abs } a * \text{abs } (b::'a::\text{ordered-idom})$
 $\langle \text{proof} \rangle$

lemma *abs-mult-self*: $\text{abs } a * \text{abs } a = a * (a::'a::\text{ordered-idom})$
 $\langle \text{proof} \rangle$

lemma *nonzero-abs-inverse*:
 $a \neq 0 \implies \text{abs } (\text{inverse } (a::'a::\text{ordered-field})) = \text{inverse } (\text{abs } a)$
 $\langle \text{proof} \rangle$

lemma *abs-inverse* [simp]:
 $\text{abs } (\text{inverse } (a::'a::\{\text{ordered-field}, \text{division-by-zero}\})) =$
 $\text{inverse } (\text{abs } a)$
 $\langle \text{proof} \rangle$

lemma *nonzero-abs-divide*:
 $b \neq 0 \implies \text{abs } (a / (b::'a::\text{ordered-field})) = \text{abs } a / \text{abs } b$
 $\langle \text{proof} \rangle$

lemma *abs-divide* [simp]:
 $\text{abs } (a / (b::'a::\{\text{ordered-field}, \text{division-by-zero}\})) = \text{abs } a / \text{abs } b$
 $\langle \text{proof} \rangle$

lemma *abs-mult-less*:
 $[\text{abs } a < c; \text{abs } b < d] \implies \text{abs } a * \text{abs } b < c * (d::'a::\text{ordered-idom})$
 $\langle \text{proof} \rangle$

lemmas *eq-minus-self-iff* [noatp] = *equal-neg-zero*

lemma *less-minus-self-iff*: $(a < -a) = (a < (0::'a::\text{ordered-idom}))$
 $\langle \text{proof} \rangle$

lemma *abs-less-iff*: $(\text{abs } a < b) = (a < b \ \& \ -a < (b::'a::\text{ordered-idom}))$
 $\langle \text{proof} \rangle$

lemma *abs-mult-pos*: $(0::'a::\text{ordered-idom}) \leq x \implies$
 $(\text{abs } y) * x = \text{abs } (y * x)$

$\langle \text{proof} \rangle$

lemma *abs-div-pos*: $(0 :: 'a :: \{\text{division-by-zero, ordered-field}\}) < y ==>$
 $\text{abs } x / y = \text{abs } (x / y)$
 $\langle \text{proof} \rangle$

11.16 Bounds of products via negative and positive Part

lemma *mult-le-prts*:

assumes

$a1 \leq (a :: 'a :: \text{ordered-ring})$

$a \leq a2$

$b1 \leq b$

$b \leq b2$

shows

$a * b \leq \text{pprt } a2 * \text{pprt } b2 + \text{pprt } a1 * \text{nprt } b2 + \text{nprt } a2 * \text{pprt } b1 + \text{nprt } a1$
 $* \text{nprt } b1$
 $\langle \text{proof} \rangle$

lemma *mult-ge-prts*:

assumes

$a1 \leq (a :: 'a :: \text{ordered-ring})$

$a \leq a2$

$b1 \leq b$

$b \leq b2$

shows

$a * b \geq \text{nprt } a1 * \text{pprt } b2 + \text{nprt } a2 * \text{nprt } b2 + \text{pprt } a1 * \text{pprt } b1 + \text{pprt } a2$
 $* \text{nprt } b1$
 $\langle \text{proof} \rangle$

end

12 Nat: Natural numbers

theory *Nat*

imports *Inductive Ring-and-Field*

uses

$\sim\sim / \text{src} / \text{Tools} / \text{rat.ML}$

$\sim\sim / \text{src} / \text{Provers} / \text{Arith} / \text{cancel-sums.ML}$

$\text{Tools} / \text{arith-data.ML}$

$(\text{Tools} / \text{nat-arith.ML})$

$\sim\sim / \text{src} / \text{Provers} / \text{Arith} / \text{fast-lin-arith.ML}$

$(\text{Tools} / \text{lin-arith.ML})$

begin

12.1 Type *ind*

typedecl *ind*

axiomatization

Zero-Rep :: *ind* **and**

Suc-Rep :: *ind* => *ind*

where

— the axiom of infinity in 2 parts

inj-Suc-Rep: *inj Suc-Rep* **and**

Suc-Rep-not-Zero-Rep: *Suc-Rep* *x* ≠ *Zero-Rep*

12.2 Type nat

Type definition

inductive *Nat* :: *ind* => *bool*

where

Zero-RepI: *Nat Zero-Rep*

| *Suc-RepI*: *Nat i* => *Nat (Suc-Rep i)*

global

typedef (**open** *Nat*)

nat = *Nat*

⟨*proof*⟩

constdefs

Suc :: *nat* => *nat*

Suc-def: *Suc* == (%*n*. *Abs-Nat (Suc-Rep (Rep-Nat n))*)

local

instantiation *nat* :: *zero*

begin

definition *Zero-nat-def* [*code del*]:

0 = *Abs-Nat Zero-Rep*

instance ⟨*proof*⟩

end

lemma *Suc-not-Zero*: *Suc m* ≠ *0*

⟨*proof*⟩

lemma *Zero-not-Suc*: *0* ≠ *Suc m*

⟨*proof*⟩

rep-datatype *0* :: *nat Suc*

⟨*proof*⟩

lemma *nat-induct* [*case-names 0 Suc, induct type: nat*]:

— for backward compatibility – names of variables differ
fixes n
assumes $P\ 0$
and $\bigwedge n. P\ n \implies P\ (Suc\ n)$
shows $P\ n$
 $\langle proof \rangle$

declare $nat.exhaust$ [$case-names\ 0\ Suc$, $cases\ type:\ nat$]

lemmas $nat-rec-0 = nat.recs(1)$
and $nat-rec-Suc = nat.recs(2)$

lemmas $nat-case-0 = nat.cases(1)$
and $nat-case-Suc = nat.cases(2)$

Injectiveness and distinctness lemmas

lemma $inj-Suc[simp]: inj-on\ Suc\ N$
 $\langle proof \rangle$

lemma $Suc-neq-Zero: Suc\ m = 0 \implies R$
 $\langle proof \rangle$

lemma $Zero-neq-Suc: 0 = Suc\ m \implies R$
 $\langle proof \rangle$

lemma $Suc-inject: Suc\ x = Suc\ y \implies x = y$
 $\langle proof \rangle$

lemma $n-not-Suc-n: n \neq Suc\ n$
 $\langle proof \rangle$

lemma $Suc-n-not-n: Suc\ n \neq n$
 $\langle proof \rangle$

A special form of induction for reasoning about $m < n$ and $m - n$

lemma $diff-induct: (!x. P\ x\ 0) ==> (!y. P\ 0\ (Suc\ y)) ==>$
 $(!x\ y. P\ x\ y ==> P\ (Suc\ x)\ (Suc\ y)) ==> P\ m\ n$
 $\langle proof \rangle$

12.3 Arithmetic operators

instantiation $nat :: \{minus, comm-monoid-add\}$
begin

primrec $plus-nat$

where

$add-0: \quad 0 + n = (n::nat)$
 $| add-Suc: \quad Suc\ m + n = Suc\ (m + n)$

lemma *add-0-right* [*simp*]: $m + 0 = (m::nat)$
 ⟨*proof*⟩

lemma *add-Suc-right* [*simp*]: $m + Suc\ n = Suc\ (m + n)$
 ⟨*proof*⟩

declare *add-0* [*code*]

lemma *add-Suc-shift* [*code*]: $Suc\ m + n = m + Suc\ n$
 ⟨*proof*⟩

primrec *minus-nat*

where

diff-0: $m - 0 = (m::nat)$
 | *diff-Suc*: $m - Suc\ n = (case\ m - n\ of\ 0 ==> 0 \mid Suc\ k ==> k)$

declare *diff-Suc* [*simp del*]

declare *diff-0* [*code*]

lemma *diff-0-eq-0* [*simp, code*]: $0 - n = (0::nat)$
 ⟨*proof*⟩

lemma *diff-Suc-Suc* [*simp, code*]: $Suc\ m - Suc\ n = m - n$
 ⟨*proof*⟩

instance ⟨*proof*⟩

end

instantiation *nat* :: *comm-semiring-1-cancel*

begin

definition

One-nat-def [*simp*]: $1 = Suc\ 0$

primrec *times-nat*

where

mult-0: $0 * n = (0::nat)$
 | *mult-Suc*: $Suc\ m * n = n + (m * n)$

lemma *mult-0-right* [*simp*]: $(m::nat) * 0 = 0$
 ⟨*proof*⟩

lemma *mult-Suc-right* [*simp*]: $m * Suc\ n = m + (m * n)$
 ⟨*proof*⟩

lemma *add-mult-distrib*: $(m + n) * k = (m * k) + ((n * k)::nat)$
 ⟨*proof*⟩

instance $\langle proof \rangle$

end

12.3.1 Addition

lemma *nat-add-assoc*: $(m + n) + k = m + ((n + k)::nat)$
 $\langle proof \rangle$

lemma *nat-add-commute*: $m + n = n + (m::nat)$
 $\langle proof \rangle$

lemma *nat-add-left-commute*: $x + (y + z) = y + ((x + z)::nat)$
 $\langle proof \rangle$

lemma *nat-add-left-cancel* [simp]: $(k + m = k + n) = (m = (n::nat))$
 $\langle proof \rangle$

lemma *nat-add-right-cancel* [simp]: $(m + k = n + k) = (m = (n::nat))$
 $\langle proof \rangle$

Reasoning about $m + 0 = 0$, etc.

lemma *add-is-0* [iff]:
fixes $m\ n :: nat$
shows $(m + n = 0) = (m = 0 \ \& \ n = 0)$
 $\langle proof \rangle$

lemma *add-is-1*:
 $(m + n = Suc\ 0) = (m = Suc\ 0 \ \& \ n = 0 \mid m = 0 \ \& \ n = Suc\ 0)$
 $\langle proof \rangle$

lemma *one-is-add*:
 $(Suc\ 0 = m + n) = (m = Suc\ 0 \ \& \ n = 0 \mid m = 0 \ \& \ n = Suc\ 0)$
 $\langle proof \rangle$

lemma *add-eq-self-zero*:
fixes $m\ n :: nat$
shows $m + n = m \implies n = 0$
 $\langle proof \rangle$

lemma *inj-on-add-nat* [simp]: *inj-on* $(\%n::nat. n+k)\ N$
 $\langle proof \rangle$

12.3.2 Difference

lemma *diff-self-eq-0* [simp]: $(m::nat) - m = 0$
 $\langle proof \rangle$

lemma *diff-diff-left*: $(i::nat) - j - k = i - (j + k)$
 $\langle proof \rangle$

lemma *Suc-diff-diff* [simp]: $(\text{Suc } m - n) - \text{Suc } k = m - n - k$
 $\langle \text{proof} \rangle$

lemma *diff-commute*: $(i::\text{nat}) - j - k = i - k - j$
 $\langle \text{proof} \rangle$

lemma *diff-add-inverse*: $(n + m) - n = (m::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *diff-add-inverse2*: $(m + n) - n = (m::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *diff-cancel*: $(k + m) - (k + n) = m - (n::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *diff-cancel2*: $(m + k) - (n + k) = m - (n::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *diff-add-0*: $n - (n + m) = (0::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *diff-Suc-1* [simp]: $\text{Suc } n - 1 = n$
 $\langle \text{proof} \rangle$

Difference distributes over multiplication

lemma *diff-mult-distrib*: $((m::\text{nat}) - n) * k = (m * k) - (n * k)$
 $\langle \text{proof} \rangle$

lemma *diff-mult-distrib2*: $k * ((m::\text{nat}) - n) = (k * m) - (k * n)$
 $\langle \text{proof} \rangle$

12.3.3 Multiplication

lemma *nat-mult-assoc*: $(m * n) * k = m * ((n * k)::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *nat-mult-commute*: $m * n = n * (m::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *add-mult-distrib2*: $k * (m + n) = (k * m) + ((k * n)::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *mult-is-0* [simp]: $((m::\text{nat}) * n = 0) = (m=0 \mid n=0)$
 $\langle \text{proof} \rangle$

lemmas *nat-distrib* =
add-mult-distrib add-mult-distrib2 diff-mult-distrib diff-mult-distrib2

lemma *mult-eq-1-iff* [simp]: $(m * n = \text{Suc } 0) = (m = \text{Suc } 0 \ \& \ n = \text{Suc } 0)$
 $\langle \text{proof} \rangle$

lemma *one-eq-mult-iff* [simp, noatp]: $(\text{Suc } 0 = m * n) = (m = \text{Suc } 0 \ \& \ n = \text{Suc } 0)$
 $\langle \text{proof} \rangle$

lemma *nat-mult-eq-1-iff* [simp]: $m * n = (1::\text{nat}) \longleftrightarrow m = 1 \ \wedge \ n = 1$
 $\langle \text{proof} \rangle$

lemma *nat-1-eq-mult-iff* [simp]: $(1::\text{nat}) = m * n \longleftrightarrow m = 1 \ \wedge \ n = 1$
 $\langle \text{proof} \rangle$

lemma *mult-cancel1* [simp]: $(k * m = k * n) = (m = n \mid (k = (0::\text{nat})))$
 $\langle \text{proof} \rangle$

lemma *mult-cancel2* [simp]: $(m * k = n * k) = (m = n \mid (k = (0::\text{nat})))$
 $\langle \text{proof} \rangle$

lemma *Suc-mult-cancel1*: $(\text{Suc } k * m = \text{Suc } k * n) = (m = n)$
 $\langle \text{proof} \rangle$

12.4 Orders on *nat*

12.4.1 Operation definition

instantiation *nat* :: *linorder*
begin

primrec *less-eq-nat* **where**
 $(0::\text{nat}) \leq n \longleftrightarrow \text{True}$
 $\mid \text{Suc } m \leq n \longleftrightarrow (\text{case } n \text{ of } 0 \Rightarrow \text{False} \mid \text{Suc } n \Rightarrow m \leq n)$

declare *less-eq-nat.simps* [simp del]
lemma [code]: $(0::\text{nat}) \leq n \longleftrightarrow \text{True}$ $\langle \text{proof} \rangle$
lemma *le0* [iff]: $0 \leq (n::\text{nat})$ $\langle \text{proof} \rangle$

definition *less-nat* **where**
 $\text{less-eq-Suc-le}: n < m \longleftrightarrow \text{Suc } n \leq m$

lemma *Suc-le-mono* [iff]: $\text{Suc } n \leq \text{Suc } m \longleftrightarrow n \leq m$
 $\langle \text{proof} \rangle$

lemma *Suc-le-eq* [code]: $\text{Suc } m \leq n \longleftrightarrow m < n$
 $\langle \text{proof} \rangle$

lemma *le-0-eq* [iff]: $(n::\text{nat}) \leq 0 \longleftrightarrow n = 0$
 $\langle \text{proof} \rangle$

lemma *not-less0* [iff]: $\neg n < (0::\text{nat})$

$\langle proof \rangle$

lemma *less-nat-zero-code* [code]: $n < (0::nat) \longleftrightarrow False$
 $\langle proof \rangle$

lemma *Suc-less-eq* [iff]: $Suc\ m < Suc\ n \longleftrightarrow m < n$
 $\langle proof \rangle$

lemma *less-Suc-eq-le* [code]: $m < Suc\ n \longleftrightarrow m \leq n$
 $\langle proof \rangle$

lemma *le-SucI*: $m \leq n \implies m \leq Suc\ n$
 $\langle proof \rangle$

lemma *Suc-leD*: $Suc\ m \leq n \implies m \leq n$
 $\langle proof \rangle$

lemma *less-SucI*: $m < n \implies m < Suc\ n$
 $\langle proof \rangle$

lemma *Suc-lessD*: $Suc\ m < n \implies m < n$
 $\langle proof \rangle$

instance
 $\langle proof \rangle$

end

instantiation *nat* :: *bot*
begin

definition *bot-nat* :: *nat* **where**
bot-nat = 0

instance $\langle proof \rangle$

end

12.4.2 Introduction properties

lemma *lessI* [iff]: $n < Suc\ n$
 $\langle proof \rangle$

lemma *zero-less-Suc* [iff]: $0 < Suc\ n$
 $\langle proof \rangle$

12.4.3 Elimination properties

lemma *less-not-refl*: $\sim n < (n::nat)$
 $\langle proof \rangle$

lemma *less-not-refl2*: $n < m \implies m \neq (n::nat)$
 $\langle proof \rangle$

lemma *less-not-refl3*: $(s::nat) < t \implies s \neq t$
 $\langle proof \rangle$

lemma *less-irrefl-nat*: $(n::nat) < n \implies R$
 $\langle proof \rangle$

lemma *less-zeroE*: $(n::nat) < 0 \implies R$
 $\langle proof \rangle$

lemma *less-Suc-eq*: $(m < Suc\ n) = (m < n \mid m = n)$
 $\langle proof \rangle$

lemma *less-Suc0* [iff]: $(n < Suc\ 0) = (n = 0)$
 $\langle proof \rangle$

lemma *less-one* [iff, noatp]: $(n < (1::nat)) = (n = 0)$
 $\langle proof \rangle$

lemma *Suc-mono*: $m < n \implies Suc\ m < Suc\ n$
 $\langle proof \rangle$

”Less than” is antisymmetric, sort of

lemma *less-antisym*: $\llbracket \neg n < m; n < Suc\ m \rrbracket \implies m = n$
 $\langle proof \rangle$

lemma *nat-neq-iff*: $((m::nat) \neq n) = (m < n \mid n < m)$
 $\langle proof \rangle$

lemma *nat-less-cases*: **assumes** *major*: $(m::nat) < n \implies P\ n\ m$
and *eqCase*: $m = n \implies P\ n\ m$ **and** *lessCase*: $n < m \implies P\ n\ m$
shows $P\ n\ m$
 $\langle proof \rangle$

12.4.4 Inductive (?) properties

lemma *Suc-lessI*: $m < n \implies Suc\ m \neq n \implies Suc\ m < n$
 $\langle proof \rangle$

lemma *lessE*:
assumes *major*: $i < k$
and *p1*: $k = Suc\ i \implies P$ **and** *p2*: $\forall j. i < j \implies k = Suc\ j \implies P$
shows P
 $\langle proof \rangle$

lemma *less-SucE*: **assumes** *major*: $m < Suc\ n$

and *less*: $m < n \implies P$ **and** *eq*: $m = n \implies P$ **shows** P
 ⟨proof⟩

lemma *Suc-lessE*: **assumes** *major*: $\text{Suc } i < k$
and *minor*: $!!j. i < j \implies k = \text{Suc } j \implies P$ **shows** P
 ⟨proof⟩

lemma *Suc-less-SucD*: $\text{Suc } m < \text{Suc } n \implies m < n$
 ⟨proof⟩

lemma *less-trans-Suc*:
assumes *le*: $i < j$ **shows** $j < k \implies \text{Suc } i < k$
 ⟨proof⟩

Can be used with *less-Suc-eq* to get $n = m \vee n < m$

lemma *not-less-eq*: $\neg m < n \longleftrightarrow n < \text{Suc } m$
 ⟨proof⟩

lemma *not-less-eq-eq*: $\neg m \leq n \longleftrightarrow \text{Suc } n \leq m$
 ⟨proof⟩

Properties of “less than or equal”

lemma *le-imp-less-Suc*: $m \leq n \implies m < \text{Suc } n$
 ⟨proof⟩

lemma *Suc-n-not-le-n*: $\sim \text{Suc } n \leq n$
 ⟨proof⟩

lemma *le-Suc-eq*: $(m \leq \text{Suc } n) = (m \leq n \mid m = \text{Suc } n)$
 ⟨proof⟩

lemma *le-SucE*: $m \leq \text{Suc } n \implies (m \leq n \implies R) \implies (m = \text{Suc } n \implies R) \implies R$
 ⟨proof⟩

lemma *Suc-leI*: $m < n \implies \text{Suc}(m) \leq n$
 ⟨proof⟩

Stronger version of *Suc-leD*

lemma *Suc-le-lessD*: $\text{Suc } m \leq n \implies m < n$
 ⟨proof⟩

lemma *less-imp-le-nat*: $m < n \implies m \leq (n::\text{nat})$
 ⟨proof⟩

For instance, $(\text{Suc } m < \text{Suc } n) = (\text{Suc } m \leq n) = (m < n)$

lemmas *le-simps* = *less-imp-le-nat less-Suc-eq-le Suc-le-eq*

Equivalence of $m \leq n$ and $m < n \vee m = n$

lemma *less-or-eq-imp-le*: $m < n \mid m = n \implies m \leq (n::nat)$
 $\langle proof \rangle$

lemma *le-eq-less-or-eq*: $(m \leq (n::nat)) = (m < n \mid m = n)$
 $\langle proof \rangle$

Useful with *blast*.

lemma *eq-imp-le*: $(m::nat) = n \implies m \leq n$
 $\langle proof \rangle$

lemma *le-refl*: $n \leq (n::nat)$
 $\langle proof \rangle$

lemma *le-trans*: $[\mid i \leq j; j \leq k \mid] \implies i \leq (k::nat)$
 $\langle proof \rangle$

lemma *le-anti-sym*: $[\mid m \leq n; n \leq m \mid] \implies m = (n::nat)$
 $\langle proof \rangle$

lemma *nat-less-le*: $((m::nat) < n) = (m \leq n \ \& \ m \neq n)$
 $\langle proof \rangle$

lemma *le-neq-implies-less*: $(m::nat) \leq n \implies m \neq n \implies m < n$
 $\langle proof \rangle$

lemma *nat-le-linear*: $(m::nat) \leq n \mid n \leq m$
 $\langle proof \rangle$

lemmas *linorder-neqE-nat* = *linorder-neqE* [**where** 'a = nat]

lemma *le-less-Suc-eq*: $m \leq n \implies (n < Suc \ m) = (n = m)$
 $\langle proof \rangle$

lemma *not-less-less-Suc-eq*: $\sim n < m \implies (n < Suc \ m) = (n = m)$
 $\langle proof \rangle$

lemmas *not-less-simps* = *not-less-less-Suc-eq* *le-less-Suc-eq*

These two rules ease the use of primitive recursion. NOTE USE OF ==

lemma *def-nat-rec-0*: $(!!n. f \ n == nat-rec \ c \ h \ n) \implies f \ 0 = c$
 $\langle proof \rangle$

lemma *def-nat-rec-Suc*: $(!!n. f \ n == nat-rec \ c \ h \ n) \implies f \ (Suc \ n) = h \ n \ (f \ n)$
 $\langle proof \rangle$

lemma *not0-implies-Suc*: $n \neq 0 \implies \exists m. n = Suc \ m$
 $\langle proof \rangle$

lemma *gr0-implies-Suc*: $n > 0 \implies \exists m. n = Suc \ m$

$\langle proof \rangle$

lemma *gr-implies-not0*: **fixes** $n :: nat$ **shows** $m < n ==> n \neq 0$
 $\langle proof \rangle$

lemma *neq0-conv*[*iff*]: **fixes** $n :: nat$ **shows** $(n \neq 0) = (0 < n)$
 $\langle proof \rangle$

This theorem is useful with *blast*

lemma *gr0I*: $((n :: nat) = 0 ==> False) ==> 0 < n$
 $\langle proof \rangle$

lemma *gr0-conv-Suc*: $(0 < n) = (\exists m. n = Suc\ m)$
 $\langle proof \rangle$

lemma *not-gr0* [*iff*, *noatp*]: $!!n :: nat. (\sim (0 < n)) = (n = 0)$
 $\langle proof \rangle$

lemma *Suc-le-D*: $(Suc\ n \leq m') ==> (? m. m' = Suc\ m)$
 $\langle proof \rangle$

Useful in certain inductive arguments

lemma *less-Suc-eq-0-disj*: $(m < Suc\ n) = (m = 0 \mid (\exists j. m = Suc\ j \ \& \ j < n))$
 $\langle proof \rangle$

12.4.5 *min* and *max*

lemma *mono-Suc*: *mono* *Suc*
 $\langle proof \rangle$

lemma *min-0L* [*simp*]: $min\ 0\ n = (0 :: nat)$
 $\langle proof \rangle$

lemma *min-0R* [*simp*]: $min\ n\ 0 = (0 :: nat)$
 $\langle proof \rangle$

lemma *min-Suc-Suc* [*simp*]: $min\ (Suc\ m)\ (Suc\ n) = Suc\ (min\ m\ n)$
 $\langle proof \rangle$

lemma *min-Suc1*:
 $min\ (Suc\ n)\ m = (case\ m\ of\ 0 ==> 0 \mid Suc\ m' ==> Suc\ (min\ n\ m'))$
 $\langle proof \rangle$

lemma *min-Suc2*:
 $min\ m\ (Suc\ n) = (case\ m\ of\ 0 ==> 0 \mid Suc\ m' ==> Suc\ (min\ m'\ n))$
 $\langle proof \rangle$

lemma *max-0L* [*simp*]: $max\ 0\ n = (n :: nat)$
 $\langle proof \rangle$

lemma *max-0R* [simp]: $\max n 0 = (n::nat)$
 $\langle proof \rangle$

lemma *max-Suc-Suc* [simp]: $\max (Suc\ m) (Suc\ n) = Suc(\max\ m\ n)$
 $\langle proof \rangle$

lemma *max-Suc1*:
 $\max (Suc\ n)\ m = (case\ m\ of\ 0 ==> Suc\ n \mid Suc\ m' ==> Suc(\max\ n\ m'))$
 $\langle proof \rangle$

lemma *max-Suc2*:
 $\max\ m\ (Suc\ n) = (case\ m\ of\ 0 ==> Suc\ n \mid Suc\ m' ==> Suc(\max\ m'\ n))$
 $\langle proof \rangle$

12.4.6 Monotonicity of Addition

lemma *Suc-pred* [simp]: $n > 0 ==> Suc\ (n - Suc\ 0) = n$
 $\langle proof \rangle$

lemma *Suc-diff-1* [simp]: $0 < n ==> Suc\ (n - 1) = n$
 $\langle proof \rangle$

lemma *nat-add-left-cancel-le* [simp]: $(k + m \leq k + n) = (m \leq (n::nat))$
 $\langle proof \rangle$

lemma *nat-add-left-cancel-less* [simp]: $(k + m < k + n) = (m < (n::nat))$
 $\langle proof \rangle$

lemma *add-gr-0* [iff]: $!!m::nat. (m + n > 0) = (m > 0 \mid n > 0)$
 $\langle proof \rangle$

strict, in 1st argument

lemma *add-less-mono1*: $i < j ==> i + k < j + (k::nat)$
 $\langle proof \rangle$

strict, in both arguments

lemma *add-less-mono*: $[[i < j; k < l]] ==> i + k < j + (l::nat)$
 $\langle proof \rangle$

Deleted *less-natE*; use *less-imp-Suc-add RS exE*

lemma *less-imp-Suc-add*: $m < n ==> (\exists k. n = Suc\ (m + k))$
 $\langle proof \rangle$

strict, in 1st argument; proof is by induction on $k > 0$

lemma *mult-less-mono2*: $(i::nat) < j ==> 0 < k ==> k * i < k * j$
 $\langle proof \rangle$

The naturals form an ordered *comm-semiring-1-cancel*

instance *nat* :: *ordered-semidom*
 ⟨*proof*⟩

instance *nat* :: *no-zero-divisors*
 ⟨*proof*⟩

lemma *nat-mult-1*: $(1::nat) * n = n$
 ⟨*proof*⟩

lemma *nat-mult-1-right*: $n * (1::nat) = n$
 ⟨*proof*⟩

12.4.7 Additional theorems about $op \leq$

Complete induction, aka course-of-values induction

instance *nat* :: *wellorder* ⟨*proof*⟩

lemma *Least-Suc*:
 $[| P\ n; \sim P\ 0\ |] ==> (LEAST\ n.\ P\ n) = Suc\ (LEAST\ m.\ P\ (Suc\ m))$
 ⟨*proof*⟩

lemma *Least-Suc2*:
 $[| P\ n; Q\ m; \sim P\ 0; !k.\ P\ (Suc\ k) = Q\ k\ |] ==> Least\ P = Suc\ (Least\ Q)$
 ⟨*proof*⟩

lemma *ex-least-nat-le*: $\neg P(0) \implies P(n::nat) \implies \exists k \leq n. (\forall i < k. \neg P\ i) \ \&\ P(k)$
 ⟨*proof*⟩

lemma *ex-least-nat-less*: $\neg P(0) \implies P(n::nat) \implies \exists k < n. (\forall i \leq k. \neg P\ i) \ \&\ P(k+1)$
 ⟨*proof*⟩

lemma *nat-less-induct*:
assumes $!!n. \forall m::nat. m < n \dashv\rightarrow P\ m \implies P\ n$ **shows** $P\ n$
 ⟨*proof*⟩

lemma *measure-induct-rule* [*case-names less*]:
fixes $f :: 'a \Rightarrow nat$
assumes *step*: $\bigwedge x. (\bigwedge y. f\ y < f\ x \implies P\ y) \implies P\ x$
shows $P\ a$
 ⟨*proof*⟩

old style induction rules:

lemma *measure-induct*:
fixes $f :: 'a \Rightarrow nat$
shows $(\bigwedge x. \forall y. f\ y < f\ x \longrightarrow P\ y \implies P\ x) \implies P\ a$
 ⟨*proof*⟩

lemma *full-nat-induct*:
assumes *step*: $(!!n. (ALL\ m. Suc\ m \leq n \dashv\rightarrow P\ m) \implies P\ n)$

shows $P\ n$
 $\langle proof \rangle$

An induction rule for establishing binary relations

lemma *less-Suc-induct*:

assumes *less*: $i < j$
and *step*: $!!i. P\ i\ (Suc\ i)$
and *trans*: $!!i\ j\ k. P\ i\ j \implies P\ j\ k \implies P\ i\ k$
shows $P\ i\ j$
 $\langle proof \rangle$

The method of infinite descent, frequently used in number theory. Provided by Roelof Oosterhuis. $P(n)$ is true for all $n \in \mathbb{N}$ if

- case “0”: given $n = 0$ prove $P(n)$,
- case “smaller”: given $n > 0$ and $\neg P(n)$ prove there exists a smaller integer m such that $\neg P(m)$.

A compact version without explicit base case:

lemma *infinite-descent*:

$\llbracket !!n::nat. \neg P\ n \implies \exists m < n. \neg P\ m \rrbracket \implies P\ n$
 $\langle proof \rangle$

lemma *infinite-descent0*[*case-names 0 smaller*]:

$\llbracket P\ 0; !!n. n > 0 \implies \neg P\ n \implies (\exists m::nat. m < n \wedge \neg P\ m) \rrbracket \implies P\ n$
 $\langle proof \rangle$

Infinite descent using a mapping to \mathbb{N} : $P(x)$ is true for all $x \in D$ if there exists a $V : D \rightarrow \mathbb{N}$ and

- case “0”: given $V(x) = 0$ prove $P(x)$,
- case “smaller”: given $V(x) > 0$ and $\neg P(x)$ prove there exists a $y \in D$ such that $V(y) < V(x)$ and $\neg P(y)$.

NB: the proof also shows how to use the previous lemma.

corollary *infinite-descent0-measure* [*case-names 0 smaller*]:

assumes *A0*: $!!x. V\ x = (0::nat) \implies P\ x$
and *A1*: $!!x. V\ x > 0 \implies \neg P\ x \implies (\exists y. V\ y < V\ x \wedge \neg P\ y)$
shows $P\ x$
 $\langle proof \rangle$

Again, without explicit base case:

lemma *infinite-descent-measure*:

assumes $!!x. \neg P\ x \implies \exists y. (V::'a \Rightarrow nat)\ y < V\ x \wedge \neg P\ y$ **shows** $P\ x$
 $\langle proof \rangle$

A [clumsy] way of lifting $<$ monotonicity to \leq monotonicity

lemma *less-mono-imp-le-mono*:

$\llbracket \text{!}i\ j::\text{nat}. i < j \implies f\ i < f\ j; i \leq j \rrbracket \implies f\ i \leq (f\ j)::\text{nat}$
 $\langle\text{proof}\rangle$

non-strict, in 1st argument

lemma *add-le-mono1*: $i \leq j \implies i + k \leq j + (k::\text{nat})$

$\langle\text{proof}\rangle$

non-strict, in both arguments

lemma *add-le-mono*: $[i \leq j; k \leq l] \implies i + k \leq j + (l::\text{nat})$

$\langle\text{proof}\rangle$

lemma *le-add2*: $n \leq ((m + n)::\text{nat})$

$\langle\text{proof}\rangle$

lemma *le-add1*: $n \leq ((n + m)::\text{nat})$

$\langle\text{proof}\rangle$

lemma *less-add-Suc1*: $i < \text{Suc}\ (i + m)$

$\langle\text{proof}\rangle$

lemma *less-add-Suc2*: $i < \text{Suc}\ (m + i)$

$\langle\text{proof}\rangle$

lemma *less-iff-Suc-add*: $(m < n) = (\exists k. n = \text{Suc}\ (m + k))$

$\langle\text{proof}\rangle$

lemma *trans-le-add1*: $(i::\text{nat}) \leq j \implies i \leq j + m$

$\langle\text{proof}\rangle$

lemma *trans-le-add2*: $(i::\text{nat}) \leq j \implies i \leq m + j$

$\langle\text{proof}\rangle$

lemma *trans-less-add1*: $(i::\text{nat}) < j \implies i < j + m$

$\langle\text{proof}\rangle$

lemma *trans-less-add2*: $(i::\text{nat}) < j \implies i < m + j$

$\langle\text{proof}\rangle$

lemma *add-lessD1*: $i + j < (k::\text{nat}) \implies i < k$

$\langle\text{proof}\rangle$

lemma *not-add-less1* [iff]: $\sim (i + j < (i::\text{nat}))$

$\langle\text{proof}\rangle$

lemma *not-add-less2* [iff]: $\sim (j + i < (i::\text{nat}))$

$\langle\text{proof}\rangle$

lemma *add-leD1*: $m + k \leq n \implies m \leq (n::nat)$
 $\langle proof \rangle$

lemma *add-leD2*: $m + k \leq n \implies k \leq (n::nat)$
 $\langle proof \rangle$

lemma *add-leE*: $(m::nat) + k \leq n \implies (m \leq n \implies k \leq n \implies R) \implies R$
 $\langle proof \rangle$

needs !!*k* for *add-ac* to work

lemma *less-add-eq-less*: $!!k::nat. k < l \implies m + l = k + n \implies m < n$
 $\langle proof \rangle$

12.4.8 More results about difference

Addition is the inverse of subtraction: if $n \leq m$ then $n + (m - n) = m$.

lemma *add-diff-inverse*: $\sim m < n \implies n + (m - n) = (m::nat)$
 $\langle proof \rangle$

lemma *le-add-diff-inverse* [*simp*]: $n \leq m \implies n + (m - n) = (m::nat)$
 $\langle proof \rangle$

lemma *le-add-diff-inverse2* [*simp*]: $n \leq m \implies (m - n) + n = (m::nat)$
 $\langle proof \rangle$

lemma *Suc-diff-le*: $n \leq m \implies Suc\ m - n = Suc\ (m - n)$
 $\langle proof \rangle$

lemma *diff-less-Suc*: $m - n < Suc\ m$
 $\langle proof \rangle$

lemma *diff-le-self* [*simp*]: $m - n \leq (m::nat)$
 $\langle proof \rangle$

lemma *le-iff-add*: $(m::nat) \leq n = (\exists k. n = m + k)$
 $\langle proof \rangle$

lemma *less-imp-diff-less*: $(j::nat) < k \implies j - n < k$
 $\langle proof \rangle$

lemma *diff-Suc-less* [*simp*]: $0 < n \implies n - Suc\ i < n$
 $\langle proof \rangle$

lemma *diff-add-assoc*: $k \leq (j::nat) \implies (i + j) - k = i + (j - k)$
 $\langle proof \rangle$

lemma *diff-add-assoc2*: $k \leq (j::nat) \implies (j + i) - k = (j - k) + i$
 $\langle proof \rangle$

lemma *le-imp-diff-is-add*: $i \leq (j::nat) \implies (j - i = k) = (j = k + i)$
 $\langle proof \rangle$

lemma *diff-is-0-eq* [simp]: $((m::nat) - n = 0) = (m \leq n)$
 $\langle proof \rangle$

lemma *diff-is-0-eq'* [simp]: $m \leq n \implies (m::nat) - n = 0$
 $\langle proof \rangle$

lemma *zero-less-diff* [simp]: $(0 < n - (m::nat)) = (m < n)$
 $\langle proof \rangle$

lemma *less-imp-add-positive*:
 assumes $i < j$
 shows $\exists k::nat. 0 < k \ \& \ i + k = j$
 $\langle proof \rangle$

a nice rewrite for bounded subtraction

lemma *nat-minus-add-max*:
 fixes $n \ m :: nat$
 shows $n - m + m = \max \ n \ m$
 $\langle proof \rangle$

lemma *nat-diff-split*:
 $P(a - b::nat) = ((a < b \longrightarrow P \ 0) \ \& \ (ALL \ d. \ a = b + d \longrightarrow P \ d))$
 — elimination of $-$ on *nat*
 $\langle proof \rangle$

lemma *nat-diff-split-asm*:
 $P(a - b::nat) = (\sim (a < b \ \& \ \sim P \ 0 \mid (EX \ d. \ a = b + d \ \& \ \sim P \ d)))$
 — elimination of $-$ on *nat* in assumptions
 $\langle proof \rangle$

12.4.9 Monotonicity of Multiplication

lemma *mult-le-mono1*: $i \leq (j::nat) \implies i * k \leq j * k$
 $\langle proof \rangle$

lemma *mult-le-mono2*: $i \leq (j::nat) \implies k * i \leq k * j$
 $\langle proof \rangle$

\leq monotonicity, BOTH arguments

lemma *mult-le-mono*: $i \leq (j::nat) \implies k \leq l \implies i * k \leq j * l$
 $\langle proof \rangle$

lemma *mult-less-mono1*: $(i::nat) < j \implies 0 < k \implies i * k < j * k$
 $\langle proof \rangle$

Differs from the standard *zero-less-mult-iff* in that there are no negative numbers.

lemma *nat-0-less-mult-iff* [simp]: $(0 < (m::nat) * n) = (0 < m \ \& \ 0 < n)$
 $\langle proof \rangle$

lemma *one-le-mult-iff* [simp]: $(Suc \ 0 \leq m * n) = (Suc \ 0 \leq m \ \& \ Suc \ 0 \leq n)$
 $\langle proof \rangle$

lemma *mult-less-cancel2* [simp]: $((m::nat) * k < n * k) = (0 < k \ \& \ m < n)$
 $\langle proof \rangle$

lemma *mult-less-cancel1* [simp]: $(k * (m::nat) < k * n) = (0 < k \ \& \ m < n)$
 $\langle proof \rangle$

lemma *mult-le-cancel1* [simp]: $(k * (m::nat) \leq k * n) = (0 < k \longrightarrow m \leq n)$
 $\langle proof \rangle$

lemma *mult-le-cancel2* [simp]: $((m::nat) * k \leq n * k) = (0 < k \longrightarrow m \leq n)$
 $\langle proof \rangle$

lemma *Suc-mult-less-cancel1*: $(Suc \ k * m < Suc \ k * n) = (m < n)$
 $\langle proof \rangle$

lemma *Suc-mult-le-cancel1*: $(Suc \ k * m \leq Suc \ k * n) = (m \leq n)$
 $\langle proof \rangle$

lemma *le-square*: $m \leq m * (m::nat)$
 $\langle proof \rangle$

lemma *le-cube*: $(m::nat) \leq m * (m * m)$
 $\langle proof \rangle$

Lemma for *gcd*

lemma *mult-eq-self-implies-10*: $(m::nat) = m * n \implies n = 1 \mid m = 0$
 $\langle proof \rangle$

the lattice order on *nat*

instantiation *nat* :: *distrib-lattice*
begin

definition
 $(inf :: nat \Rightarrow nat \Rightarrow nat) = min$

definition
 $(sup :: nat \Rightarrow nat \Rightarrow nat) = max$

instance $\langle proof \rangle$

end

12.5 Embedding of the Naturals into any *semiring-1*: *of-nat*

context *semiring-1*

begin

primrec

of-nat :: *nat* \Rightarrow *'a*

where

of-nat-0: *of-nat* 0 = 0

| *of-nat-Suc*: *of-nat* (Suc *m*) = 1 + *of-nat m*

lemma *of-nat-1* [*simp*]: *of-nat* 1 = 1

\langle *proof* \rangle

lemma *of-nat-add* [*simp*]: *of-nat* (*m* + *n*) = *of-nat m* + *of-nat n*

\langle *proof* \rangle

lemma *of-nat-mult*: *of-nat* (*m* * *n*) = *of-nat m* * *of-nat n*

\langle *proof* \rangle

primrec *of-nat-aux* :: (*'a* \Rightarrow *'a*) \Rightarrow *nat* \Rightarrow *'a* \Rightarrow *'a* **where**

of-nat-aux inc 0 *i* = *i*

| *of-nat-aux inc* (Suc *n*) *i* = *of-nat-aux inc n* (*inc i*) — tail recursive

lemma *of-nat-code* [*code*, *code unfold*, *code inline del*]:

of-nat n = *of-nat-aux* ($\lambda i. i + 1$) *n* 0

\langle *proof* \rangle

end

Class for unital semirings with characteristic zero. Includes non-ordered rings like the complex numbers.

class *semiring-char-0* = *semiring-1* +

assumes *of-nat-eq-iff* [*simp*]: *of-nat m* = *of-nat n* \longleftrightarrow *m* = *n*

begin

Special cases where either operand is zero

lemma *of-nat-0-eq-iff* [*simp*, *noatp*]: 0 = *of-nat n* \longleftrightarrow 0 = *n*

\langle *proof* \rangle

lemma *of-nat-eq-0-iff* [*simp*, *noatp*]: *of-nat m* = 0 \longleftrightarrow *m* = 0

\langle *proof* \rangle

lemma *inj-of-nat*: *inj of-nat*

\langle *proof* \rangle

end

context *ordered-semidom*

begin

lemma *zero-le-imp-of-nat*: $0 \leq \text{of-nat } m$
 $\langle \text{proof} \rangle$

lemma *less-imp-of-nat-less*: $m < n \implies \text{of-nat } m < \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *of-nat-less-imp-less*: $\text{of-nat } m < \text{of-nat } n \implies m < n$
 $\langle \text{proof} \rangle$

lemma *of-nat-less-iff [simp]*: $\text{of-nat } m < \text{of-nat } n \iff m < n$
 $\langle \text{proof} \rangle$

lemma *of-nat-le-iff [simp]*: $\text{of-nat } m \leq \text{of-nat } n \iff m \leq n$
 $\langle \text{proof} \rangle$

Every *ordered-semidom* has characteristic zero.

subclass *semiring-char-0*
 $\langle \text{proof} \rangle$

Special cases where either operand is zero

lemma *of-nat-0-le-iff [simp]*: $0 \leq \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *of-nat-le-0-iff [simp, noatp]*: $\text{of-nat } m \leq 0 \iff m = 0$
 $\langle \text{proof} \rangle$

lemma *of-nat-0-less-iff [simp]*: $0 < \text{of-nat } n \iff 0 < n$
 $\langle \text{proof} \rangle$

lemma *of-nat-less-0-iff [simp]*: $\neg \text{of-nat } m < 0$
 $\langle \text{proof} \rangle$

end

context *ring-1*

begin

lemma *of-nat-diff*: $n \leq m \implies \text{of-nat } (m - n) = \text{of-nat } m - \text{of-nat } n$
 $\langle \text{proof} \rangle$

end

context *ordered-idom*

begin

lemma *abs-of-nat [simp]*: $|\text{of-nat } n| = \text{of-nat } n$
 $\langle \text{proof} \rangle$

end

lemma *of-nat-id* [*simp*]: *of-nat* $n = n$
 ⟨*proof*⟩

lemma *of-nat-eq-id* [*simp*]: *of-nat* = *id*
 ⟨*proof*⟩

12.6 The Set of Natural Numbers

context *semiring-1*
begin

definition
Nats :: 'a set **where**
 [*code del*]: *Nats* = *range of-nat*

notation (*xsymbols*)
Nats (\mathbb{N})

lemma *of-nat-in-Nats* [*simp*]: *of-nat* $n \in \mathbb{N}$
 ⟨*proof*⟩

lemma *Nats-0* [*simp*]: $0 \in \mathbb{N}$
 ⟨*proof*⟩

lemma *Nats-1* [*simp*]: $1 \in \mathbb{N}$
 ⟨*proof*⟩

lemma *Nats-add* [*simp*]: $a \in \mathbb{N} \implies b \in \mathbb{N} \implies a + b \in \mathbb{N}$
 ⟨*proof*⟩

lemma *Nats-mult* [*simp*]: $a \in \mathbb{N} \implies b \in \mathbb{N} \implies a * b \in \mathbb{N}$
 ⟨*proof*⟩

end

12.7 Further Arithmetic Facts Concerning the Natural Numbers

lemma *subst-equals*:
assumes *1*: $t = s$ **and** *2*: $u = t$
shows $u = s$
 ⟨*proof*⟩

⟨*ML*⟩

lemmas [*arith-split*] = *nat-diff-split split-min split-max*

context *order*
begin

lemma *lift-Suc-mono-le*:
 assumes *mono*: $!!n. f\ n \leq f(\text{Suc } n)$ and $n \leq n'$
 shows $f\ n \leq f\ n'$
 $\langle \text{proof} \rangle$

lemma *lift-Suc-mono-less*:
 assumes *mono*: $!!n. f\ n < f(\text{Suc } n)$ and $n < n'$
 shows $f\ n < f\ n'$
 $\langle \text{proof} \rangle$

lemma *lift-Suc-mono-less-iff*:
 $(!!n. f\ n < f(\text{Suc } n)) \implies f(n) < f(m) \longleftrightarrow n < m$
 $\langle \text{proof} \rangle$

end

lemma *mono-iff-le-Suc*: $\text{mono } f = (\forall n. f\ n \leq f(\text{Suc } n))$
 $\langle \text{proof} \rangle$

lemma *mono-nat-linear-lb*:
 $(!!m\ n::\text{nat}. m < n \implies f\ m < f\ n) \implies f(m) + k \leq f(m + k)$
 $\langle \text{proof} \rangle$

Subtraction laws, mostly by Clemens Ballarin

lemma *diff-less-mono*: $[| a < (b::\text{nat}); c \leq a |] \implies a - c < b - c$
 $\langle \text{proof} \rangle$

lemma *less-diff-conv*: $(i < j - k) = (i + k < (j::\text{nat}))$
 $\langle \text{proof} \rangle$

lemma *le-diff-conv*: $(j - k \leq (i::\text{nat})) = (j \leq i + k)$
 $\langle \text{proof} \rangle$

lemma *le-diff-conv2*: $k \leq j \implies (i \leq j - k) = (i + k \leq (j::\text{nat}))$
 $\langle \text{proof} \rangle$

lemma *diff-diff-cancel* [*simp*]: $i \leq (n::\text{nat}) \implies n - (n - i) = i$
 $\langle \text{proof} \rangle$

lemma *le-add-diff*: $k \leq (n::\text{nat}) \implies m \leq n + m - k$
 $\langle \text{proof} \rangle$

lemma *diff-less* [*simp*]: $!!m::\text{nat}. [| 0 < n; 0 < m |] \implies m - n < m$
 $\langle \text{proof} \rangle$

Simplification of relational expressions involving subtraction

lemma *diff-diff-eq*: $[[k \leq m; k \leq (n::nat)]] \implies ((m-k) - (n-k)) = (m-n)$
 $\langle proof \rangle$

lemma *eq-diff-iff*: $[[k \leq m; k \leq (n::nat)]] \implies (m-k = n-k) = (m=n)$
 $\langle proof \rangle$

lemma *less-diff-iff*: $[[k \leq m; k \leq (n::nat)]] \implies (m-k < n-k) = (m < n)$
 $\langle proof \rangle$

lemma *le-diff-iff*: $[[k \leq m; k \leq (n::nat)]] \implies (m-k \leq n-k) = (m \leq n)$
 $\langle proof \rangle$

(Anti)Monotonicity of subtraction – by Stephan Merz

lemma *diff-le-mono*: $m \leq (n::nat) \implies (m-l) \leq (n-l)$
 $\langle proof \rangle$

lemma *diff-le-mono2*: $m \leq (n::nat) \implies (l-n) \leq (l-m)$
 $\langle proof \rangle$

lemma *diff-less-mono2*: $[[m < (n::nat); m < l]] \implies (l-n) < (l-m)$
 $\langle proof \rangle$

lemma *diffs0-imp-equal*: $!!m::nat. [[m-n = 0; n-m = 0]] \implies m=n$
 $\langle proof \rangle$

lemma *min-diff*: $\min (m - (i::nat)) (n - i) = \min m n - i$
 $\langle proof \rangle$

lemma *inj-on-diff-nat*:
assumes *k-le-n*: $\forall n \in N. k \leq (n::nat)$
shows *inj-on* $(\lambda n. n - k) N$
 $\langle proof \rangle$

Rewriting to pull differences out

lemma *diff-diff-right* [*simp*]: $k \leq j \implies i - (j - k) = i + (k::nat) - j$
 $\langle proof \rangle$

lemma *diff-Suc-diff-eq1* [*simp*]: $k \leq j \implies m - \text{Suc } (j - k) = m + k - \text{Suc } j$
 $\langle proof \rangle$

lemma *diff-Suc-diff-eq2* [*simp*]: $k \leq j \implies \text{Suc } (j - k) - m = \text{Suc } j - (k + m)$
 $\langle proof \rangle$

Lemmas for ex/Factorization

lemma *one-less-mult*: $[[\text{Suc } 0 < n; \text{Suc } 0 < m]] \implies \text{Suc } 0 < m*n$
 $\langle proof \rangle$

lemma *n-less-m-mult-n*: $[| \text{Suc } 0 < n; \text{Suc } 0 < m |] \implies n < m * n$
 $\langle \text{proof} \rangle$

lemma *n-less-n-mult-m*: $[| \text{Suc } 0 < n; \text{Suc } 0 < m |] \implies n < n * m$
 $\langle \text{proof} \rangle$

Specialized induction principles that work ”backwards”:

lemma *inc-induct*[*consumes 1, case-names base step*]:
assumes *less*: $i \leq j$
assumes *base*: $P \ j$
assumes *step*: $!!i. [i < j; P (\text{Suc } i)] \implies P \ i$
shows $P \ i$
 $\langle \text{proof} \rangle$

lemma *strict-inc-induct*[*consumes 1, case-names base step*]:
assumes *less*: $i < j$
assumes *base*: $!!i. j = \text{Suc } i \implies P \ i$
assumes *step*: $!!i. [i < j; P (\text{Suc } i)] \implies P \ i$
shows $P \ i$
 $\langle \text{proof} \rangle$

lemma *zero-induct-lemma*: $P \ k \implies (!!n. P (\text{Suc } n) \implies P \ n) \implies P \ (k - i)$
 $\langle \text{proof} \rangle$

lemma *zero-induct*: $P \ k \implies (!!n. P (\text{Suc } n) \implies P \ n) \implies P \ 0$
 $\langle \text{proof} \rangle$

lemma *nat-not-singleton*: $(\forall x. x = (0::\text{nat})) = \text{False}$
 $\langle \text{proof} \rangle$

lemmas *add-diff-assoc* = *diff-add-assoc* [*symmetric*]
lemmas *add-diff-assoc2* = *diff-add-assoc2* [*symmetric*]
declare *diff-diff-left* [*simp*] *add-diff-assoc* [*simp*] *add-diff-assoc2* [*simp*]

At present we prove no analogue of *not-less-Least* or *Least-Suc*, since there appears to be no need.

12.8 size of a datatype value

class *size* =
fixes *size* :: $'a \Rightarrow \text{nat}$ — see further theory *Wellfounded*
end

13 Product-Type: Cartesian products

theory *Product-Type*

```

imports Inductive
uses
  (Tools/split-rule.ML)
  (Tools/inductive-set-package.ML)
  (Tools/inductive-realizer.ML)
  (Tools/datatype-realizer.ML)
begin

```

13.1 *bool* is a datatype

```

rep-datatype True False  $\langle proof \rangle$ 

```

```

declare case-split [cases type: bool]
  — prefer plain propositional version

```

```

lemma
  shows [code]: eq-class.eq False P  $\longleftrightarrow \neg P$ 
    and [code]: eq-class.eq True P  $\longleftrightarrow P$ 
    and [code]: eq-class.eq P False  $\longleftrightarrow \neg P$ 
    and [code]: eq-class.eq P True  $\longleftrightarrow P$ 
    and [code nbe]: eq-class.eq P P  $\longleftrightarrow True$ 
   $\langle proof \rangle$ 

```

```

code-const eq-class.eq :: bool  $\Rightarrow$  bool  $\Rightarrow$  bool
  (Haskell infixl 4 ==)

```

```

code-instance bool :: eq
  (Haskell -)

```

13.2 Unit

```

typedef unit = { True }
 $\langle proof \rangle$ 

```

```

definition
  Unity :: unit    ('())
where
  () = Abs-unit True

```

```

lemma unit-eq [noatp]: u = ()
   $\langle proof \rangle$ 

```

Simplification procedure for *unit-eq*. Cannot use this rule directly — it loops!

$\langle ML \rangle$

```

rep-datatype ()  $\langle proof \rangle$ 

```

```

lemma unit-all-eq1: (!!x::unit. PROP P x) == PROP P ()

```

<proof>

lemma *unit-all-eq2*: ($!!x::unit. PROP P$) == *PROP P*
<proof>

This rewrite counters the effect of *unit-eq-proc* on $\%u::unit. f u$, replacing it by f rather than by $\%u. f ()$.

lemma *unit-abs-eta-conv* [*simp, noatp*]: ($\%u::unit. f ()$) = f
<proof>

code generator setup

instance *unit* :: *eq* *<proof>*

lemma [*code*]:
eq-class.eq ($u::unit$) $v \longleftrightarrow True$ *<proof>*

code-type *unit*
 (*SML* *unit*)
 (*OCaml* *unit*)
 (*Haskell* ())

code-instance *unit* :: *eq*
 (*Haskell* $-$)

code-const *eq-class.eq* :: *unit* \Rightarrow *unit* \Rightarrow *bool*
 (*Haskell* **infixl** 4 ==)

code-const *Unity*
 (*SML* ())
 (*OCaml* ())
 (*Haskell* ())

code-reserved *SML*
unit

code-reserved *OCaml*
unit

13.3 Pairs

13.3.1 Product type, basic operations and concrete syntax

definition

Pair-Rep :: $'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$

where

Pair-Rep $a b = (\lambda x y. x = a \wedge y = b)$

global

```

typedef (Prod)
  ('a, 'b) * (infixr * 20)
  = {f.  $\exists a\ b. f = \text{Pair-Rep } (a::'a) (b::'b)$ }
  <proof>

```

```

syntax (xsymbols)
  * :: [type, type] => type          ((-  $\times$  / -) [21, 20] 20)
syntax (HTML output)
  * :: [type, type] => type          ((-  $\times$  / -) [21, 20] 20)

```

```

consts
  Pair    :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'a  $\times$  'b
  fst     :: 'a  $\times$  'b  $\Rightarrow$  'a
  snd     :: 'a  $\times$  'b  $\Rightarrow$  'b
  split   :: ('a  $\Rightarrow$  'b  $\Rightarrow$  'c)  $\Rightarrow$  'a  $\times$  'b  $\Rightarrow$  'c
  curry   :: ('a  $\times$  'b  $\Rightarrow$  'c)  $\Rightarrow$  'a  $\Rightarrow$  'b  $\Rightarrow$  'c

```

local

```

defs
  Pair-def:   Pair a b == Abs-Prod (Pair-Rep a b)
  fst-def:    fst p == THE a. EX b. p = Pair a b
  snd-def:    snd p == THE b. EX a. p = Pair a b
  split-def:  split == (%c p. c (fst p) (snd p))
  curry-def:  curry == (%c x y. c (Pair x y))

```

Patterns – extends pre-defined type *pttrn* used in abstractions.

nonterminals

tuple-args patterns

```

syntax
  -tuple      :: 'a => tuple-args => 'a * 'b          ((1'(-, / -)))
  -tuple-arg  :: 'a => tuple-args                      (-)
  -tuple-args :: 'a => tuple-args => tuple-args        (-, / -)
  -pattern    :: [pttrn, patterns] => pttrn           ('(-, / -'))
               :: pttrn => patterns                    (-)
  -patterns   :: [pttrn, patterns] => patterns        (-, / -)

```

translations

```

  (x, y)      == Pair x y
  -tuple x (-tuple-args y z) == -tuple x (-tuple-arg (-tuple y z))
  %(x,y,zs).b == split(%x (y,zs).b)
  %(x,y).b    == split(%x y. b)
  -abs (Pair x y) t == %(x,y).t

```

<ML>

Towards a datatype declaration

lemma *surj-pair* [*simp*]: $EX\ x\ y.\ p = (x, y)$
 $\langle proof \rangle$

lemma *PairE* [*cases type: **]:
obtains $x\ y$ **where** $p = (x, y)$
 $\langle proof \rangle$

lemma *ProdI*: $Pair\text{-}Rep\ a\ b \in Prod$
 $\langle proof \rangle$

lemma *Pair-Rep-inject*: $Pair\text{-}Rep\ a\ b = Pair\text{-}Rep\ a'\ b' \implies a = a' \wedge b = b'$
 $\langle proof \rangle$

lemma *inj-on-Abs-Prod*: $inj\text{-}on\ Abs\text{-}Prod\ Prod$
 $\langle proof \rangle$

lemma *Pair-inject*:
assumes $(a, b) = (a', b')$
and $a = a' \implies b = b' \implies R$
shows R
 $\langle proof \rangle$

rep-datatype (*prod*) *Pair*
 $\langle proof \rangle$

lemmas $Pair\text{-}eq = prod.inject$

lemma *fst-conv* [*simp, code*]: $fst\ (a, b) = a$
 $\langle proof \rangle$

lemma *snd-conv* [*simp, code*]: $snd\ (a, b) = b$
 $\langle proof \rangle$

13.3.2 Basic rules and proof tools

lemma *fst-eqD*: $fst\ (x, y) = a \implies x = a$
 $\langle proof \rangle$

lemma *snd-eqD*: $snd\ (x, y) = a \implies y = a$
 $\langle proof \rangle$

lemma *pair-collapse* [*simp*]: $(fst\ p, snd\ p) = p$
 $\langle proof \rangle$

lemmas $surjective\text{-}pairing = pair\text{-}collapse\ [symmetric]$

lemma *split-paired-all*: $(!!x.\ PROP\ P\ x) == (!!a\ b.\ PROP\ P\ (a, b))$
 $\langle proof \rangle$

The rule *split-paired-all* does not work with the Simplifier because it also affects premises in congruence rules, where this can lead to premises of the form $!!a\ b.\ \dots = ?P(a, b)$ which cannot be solved by reflexivity.

lemmas *split-tupled-all* = *split-paired-all* *unit-all-eq2*

$\langle ML \rangle$

lemma *split-paired-All* [*simp*]: $(ALL\ x.\ P\ x) = (ALL\ a\ b.\ P\ (a, b))$
 — [*iff*] is not a good idea because it makes *blast* loop
 $\langle proof \rangle$

lemma *split-paired-Ex* [*simp*]: $(EX\ x.\ P\ x) = (EX\ a\ b.\ P\ (a, b))$
 $\langle proof \rangle$

lemma *Pair-fst-snd-eq*: $s = t \longleftrightarrow fst\ s = fst\ t \wedge snd\ s = snd\ t$
 $\langle proof \rangle$

lemma *prod-eqI* [*intro?*]: $fst\ p = fst\ q \implies snd\ p = snd\ q \implies p = q$
 $\langle proof \rangle$

13.3.3 *split* and *curry*

lemma *split-conv* [*simp*, *code*]: $split\ f\ (a, b) = f\ a\ b$
 $\langle proof \rangle$

lemma *curry-conv* [*simp*, *code*]: $curry\ f\ a\ b = f\ (a, b)$
 $\langle proof \rangle$

lemmas *split* = *split-conv* — for backwards compatibility

lemma *splitI*: $f\ a\ b \implies split\ f\ (a, b)$
 $\langle proof \rangle$

lemma *splitD*: $split\ f\ (a, b) \implies f\ a\ b$
 $\langle proof \rangle$

lemma *curryI* [*intro!*]: $f\ (a, b) \implies curry\ f\ a\ b$
 $\langle proof \rangle$

lemma *curryD* [*dest!*]: $curry\ f\ a\ b \implies f\ (a, b)$
 $\langle proof \rangle$

lemma *curryE*: $curry\ f\ a\ b \implies (f\ (a, b) \implies Q) \implies Q$
 $\langle proof \rangle$

lemma *curry-split* [*simp*]: $curry\ (split\ f) = f$
 $\langle proof \rangle$

lemma *split-curry* [*simp*]: $split\ (curry\ f) = f$

$\langle proof \rangle$

lemma *split-Pair* [*simp*]: $(\lambda(x, y). (x, y)) = id$
 $\langle proof \rangle$

lemma *split-eta*: $(\lambda(x, y). f (x, y)) = f$
 — Subsumes the old *split-Pair* when f is the identity function.
 $\langle proof \rangle$

lemma *split-comp*: $split (f \circ g) x = f (g (fst x)) (snd x)$
 $\langle proof \rangle$

lemma *split-twice*: $split f (split g p) = split (\lambda x y. split f (g x y)) p$
 $\langle proof \rangle$

lemma *split-paired-The*: $(THE x. P x) = (THE (a, b). P (a, b))$
 — Can’t be added to simpset: loops!
 $\langle proof \rangle$

lemma *The-split*: $The (split P) = (THE xy. P (fst xy) (snd xy))$
 $\langle proof \rangle$

lemma *split-weak-cong*: $p = q \implies split c p = split c q$
 — Prevents simplification of c : much faster
 $\langle proof \rangle$

lemma *cond-split-eta*: $(!!x y. f x y = g (x, y)) \implies (\%(x, y). f x y) = g$
 $\langle proof \rangle$

Simplification procedure for *cond-split-eta*. Using *split-eta* as a rewrite rule is not general enough, and using *cond-split-eta* directly would render some existing proofs very inefficient; similarly for *split-beta*.

$\langle ML \rangle$

lemma *split-beta* [*mono*]: $(\%(x, y). P x y) z = P (fst z) (snd z)$
 $\langle proof \rangle$

lemma *split-split* [*noatp*]: $R(split c p) = (ALL x y. p = (x, y) \longrightarrow R(c x y))$
 — For use with *split* and the Simplifier.
 $\langle proof \rangle$

split-split could be declared as [*split*] done after the Splitter has been speeded up significantly; precompute the constants involved and don’t do anything unless the current goal contains one of those constants.

lemma *split-split-asm* [*noatp*]: $R (split c p) = (\sim (EX x y. p = (x, y) \ \& \ (\sim R (c x y))))$
 $\langle proof \rangle$

split used as a logical connective or set former.

These rules are for use with *blast*; could instead call *simp* using *split* as rewrite.

lemma *splitI2*: $!!p. [\![\![a\ b.\ p = (a, b) ==> c\ a\ b\]\]] ==> \text{split}\ c\ p$
 $\langle \text{proof} \rangle$

lemma *splitI2'*: $!!p. [\![\![a\ b.\ (a, b) = p ==> c\ a\ b\ x\]\]] ==> \text{split}\ c\ p\ x$
 $\langle \text{proof} \rangle$

lemma *splitE*: $\text{split}\ c\ p ==> (!x\ y.\ p = (x, y) ==> c\ x\ y ==> Q) ==> Q$
 $\langle \text{proof} \rangle$

lemma *splitE'*: $\text{split}\ c\ p\ z ==> (!x\ y.\ p = (x, y) ==> c\ x\ y\ z ==> Q) ==> Q$
 $\langle \text{proof} \rangle$

lemma *splitE2*:
 $[\![\ Q\ (\text{split}\ P\ z);\ !x\ y.\ [\![z = (x, y);\ Q\ (P\ x\ y)]\]] ==> R\]\]] ==> R$
 $\langle \text{proof} \rangle$

lemma *splitD'*: $\text{split}\ R\ (a, b)\ c ==> R\ a\ b\ c$
 $\langle \text{proof} \rangle$

lemma *mem-splitI*: $z: c\ a\ b ==> z: \text{split}\ c\ (a, b)$
 $\langle \text{proof} \rangle$

lemma *mem-splitI2*: $!!p. [\![\![a\ b.\ p = (a, b) ==> z: c\ a\ b\]\]] ==> z: \text{split}\ c\ p$
 $\langle \text{proof} \rangle$

lemma *mem-splitE*:
assumes *major*: $z: \text{split}\ c\ p$
and cases: $!!x\ y.\ [\![p = (x, y);\ z: c\ x\ y\]\]] ==> Q$
shows Q
 $\langle \text{proof} \rangle$

declare *mem-splitI2* [*intro!*] *mem-splitI* [*intro!*] *splitI2'* [*intro!*] *splitI2* [*intro!*] *splitI* [*intro!*]

declare *mem-splitE* [*elim!*] *splitE'* [*elim!*] *splitE* [*elim!*]

$\langle \text{ML} \rangle$

lemma *split-eta-SetCompr* [*simp, noatp*]: $(\%u.\ EX\ x\ y.\ u = (x, y) \ \&\ P\ (x, y)) = P$
 $\langle \text{proof} \rangle$

lemma *split-eta-SetCompr2* [*simp, noatp*]: $(\%u.\ EX\ x\ y.\ u = (x, y) \ \&\ P\ x\ y) = \text{split}\ P$
 $\langle \text{proof} \rangle$

lemma *split-part* [*simp*]: $(\%(a, b).\ P \ \&\ Q\ a\ b) = (\%ab.\ P \ \&\ \text{split}\ Q\ ab)$
 — Allows simplifications of nested splits in case of independent predicates.

$\langle proof \rangle$

lemma *split-comp-eq*:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c$ **and** $g :: 'd \Rightarrow 'a$
shows $(\%u. f (g (fst u)) (snd u)) = (split (\%x. f (g x)))$
 $\langle proof \rangle$

lemma *pair-imageI* [intro]: $(a, b) : A \implies f a b : (\%(a, b). f a b) ' A$
 $\langle proof \rangle$

lemma *The-split-eq* [simp]: $(THE (x', y'). x = x' \ \& \ y = y') = (x, y)$
 $\langle proof \rangle$

Setup of internal *split-rule*.

definition

internal-split :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c$

where

internal-split == *split*

lemma *internal-split-conv*: $internal-split \ c \ (a, b) = c \ a \ b$
 $\langle proof \rangle$

hide *const internal-split*

$\langle ML \rangle$

lemmas *prod-caseI* = *prod.cases* [THEN *iffD2*, *standard*]

lemma *prod-caseI2*: $!!p. [] !!a \ b. p = (a, b) \implies c \ a \ b \ [] \implies prod-case \ c \ p$
 $\langle proof \rangle$

lemma *prod-caseI2'*: $!!p. [] !!a \ b. (a, b) = p \implies c \ a \ b \ x \ [] \implies prod-case \ c \ p \ x$
 $\langle proof \rangle$

lemma *prod-caseE*: $prod-case \ c \ p \implies (!x \ y. p = (x, y) \implies c \ x \ y \implies Q) \implies Q$
 $\langle proof \rangle$

lemma *prod-caseE'*: $prod-case \ c \ p \ z \implies (!x \ y. p = (x, y) \implies c \ x \ y \ z \implies Q) \implies Q$
 $\langle proof \rangle$

lemma *prod-case-unfold*: $prod-case = (\%c \ p. c \ (fst \ p) \ (snd \ p))$
 $\langle proof \rangle$

declare *prod-caseI2'* [intro!] *prod-caseI2* [intro!] *prod-caseI* [intro!]

declare *prod-caseE'* [elim!] *prod-caseE* [elim!]

lemma *prod-case-split*:

prod-case = *split*

<proof>

lemma *prod-case-beta*:

prod-case *f p* = *f* (*fst p*) (*snd p*)

<proof>

13.4 Further cases/induct rules for tuples

lemma *prod-cases3* [*cases type*]:

obtains (*fields*) *a b c* **where** *y* = (*a*, *b*, *c*)

<proof>

lemma *prod-induct3* [*case-names fields, induct type*]:

(!!*a b c. P* (*a*, *b*, *c*)) ==> *P x*

<proof>

lemma *prod-cases4* [*cases type*]:

obtains (*fields*) *a b c d* **where** *y* = (*a*, *b*, *c*, *d*)

<proof>

lemma *prod-induct4* [*case-names fields, induct type*]:

(!!*a b c d. P* (*a*, *b*, *c*, *d*)) ==> *P x*

<proof>

lemma *prod-cases5* [*cases type*]:

obtains (*fields*) *a b c d e* **where** *y* = (*a*, *b*, *c*, *d*, *e*)

<proof>

lemma *prod-induct5* [*case-names fields, induct type*]:

(!!*a b c d e. P* (*a*, *b*, *c*, *d*, *e*)) ==> *P x*

<proof>

lemma *prod-cases6* [*cases type*]:

obtains (*fields*) *a b c d e f* **where** *y* = (*a*, *b*, *c*, *d*, *e*, *f*)

<proof>

lemma *prod-induct6* [*case-names fields, induct type*]:

(!!*a b c d e f. P* (*a*, *b*, *c*, *d*, *e*, *f*)) ==> *P x*

<proof>

lemma *prod-cases7* [*cases type*]:

obtains (*fields*) *a b c d e f g* **where** *y* = (*a*, *b*, *c*, *d*, *e*, *f*, *g*)

<proof>

lemma *prod-induct7* [*case-names fields, induct type*]:

(!!*a b c d e f g. P* (*a*, *b*, *c*, *d*, *e*, *f*, *g*)) ==> *P x*

<proof>

13.4.1 Derived operations

The composition-uncurry combinator.

notation *fcomp* (**infixl** *o>* 60)

definition

scomp :: (*'a* \Rightarrow *'b* \times *'c*) \Rightarrow (*'b* \Rightarrow *'c* \Rightarrow *'d*) \Rightarrow *'a* \Rightarrow *'d* (**infixl** *o \rightarrow* 60)

where

f o \rightarrow g = ($\lambda x. \text{split } g \text{ (f } x)$)

lemma *scomp-apply*: (*f o \rightarrow g*) *x* = *split g (f x)*
<proof>

lemma *Pair-scomp*: *Pair x o \rightarrow f* = *f x*
<proof>

lemma *scomp-Pair*: *x o \rightarrow Pair* = *x*
<proof>

lemma *scomp-scomp*: (*f o \rightarrow g*) *o \rightarrow h* = *f o \rightarrow (\lambda x. g x o \rightarrow h)*
<proof>

lemma *scomp-fcomp*: (*f o \rightarrow g*) *o> h* = *f o \rightarrow (\lambda x. g x o> h)*
<proof>

lemma *fcomp-scomp*: (*f o> g*) *o \rightarrow h* = *f o> (g o \rightarrow h)*
<proof>

no-notation *fcomp* (**infixl** *o>* 60)

no-notation *scomp* (**infixl** *o \rightarrow* 60)

prod-fun — action of the product functor upon functions.

definition *prod-fun* :: (*'a* \Rightarrow *'c*) \Rightarrow (*'b* \Rightarrow *'d*) \Rightarrow *'a* \times *'b* \Rightarrow *'c* \times *'d* **where**
[code del]: *prod-fun f g* = ($\lambda (x, y). (f x, g y)$)

lemma *prod-fun [simp, code]*: *prod-fun f g (a, b)* = (*f a, g b*)
<proof>

lemma *prod-fun-compose*: *prod-fun (f1 o f2) (g1 o g2)* = (*prod-fun f1 g1 o prod-fun f2 g2*)
<proof>

lemma *prod-fun-ident [simp]*: *prod-fun (%x. x) (%y. y)* = (%z. z)
<proof>

lemma *prod-fun-imageI [intro]*: (*a, b*) : *r* \Rightarrow (*f a, g b*) : *prod-fun f g ‘ r*
<proof>

lemma *prod-fun-imageE [elim!]*:

assumes *major*: $c: (prod\text{-}fun\ f\ g)\ 'r$
and cases: $!!x\ y. [\mid c=(f(x),g(y));\ (x,y):r\ \mid] ==> P$
shows P
 $\langle proof \rangle$

definition

$apfst :: ('a \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c \times 'b$

where

$[code\ del]:\ apfst\ f = prod\text{-}fun\ f\ id$

definition

$apsnd :: ('b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'a \times 'c$

where

$[code\ del]:\ apsnd\ f = prod\text{-}fun\ id\ f$

lemma *apfst-conv* $[simp, code]:$

$apfst\ f\ (x, y) = (f\ x, y)$

$\langle proof \rangle$

lemma *upd-snd-conv* $[simp, code]:$

$apsnd\ f\ (x, y) = (x, f\ y)$

$\langle proof \rangle$

Disjoint union of a family of sets – Sigma.

definition *Sigma* $:: ['a\ set, 'a \Rightarrow 'b\ set] \Rightarrow ('a \times 'b)\ set$ **where**

Sigma-def: $Sigma\ A\ B == \bigcup x:A. \bigcup y:B\ x. \{Pair\ x\ y\}$

abbreviation

Times $:: ['a\ set, 'b\ set] \Rightarrow ('a * 'b)\ set$

(**infixr** $<*>$ 80) **where**

$A\ <*>\ B == Sigma\ A\ (\%-. B)$

notation (*xsymbols*)

Times (**infixr** \times 80)

notation (*HTML output*)

Times (**infixr** \times 80)

syntax

$@Sigma :: [pttrn, 'a\ set, 'b\ set] \Rightarrow ('a * 'b)\ set\ ((3SIGMA\ \text{:-./}\ -)\ [0, 0, 10]\ 10)$

translations

$SIGMA\ x:A. B == Product\text{-}Type.Sigma\ A\ (\%x. B)$

lemma *SigmaI* $[intro!]: [\mid a:A;\ b:B(a)\ \mid] ==> (a,b) : Sigma\ A\ B$

$\langle proof \rangle$

lemma *SigmaE* $[elim!]:$

$[\mid c: Sigma\ A\ B;$

$$\begin{array}{l} !!x\ y. [\![\ x:A;\ y:B(x);\ c=(x,y)\]\!] ==> P \\ [\!] ==> P \\ \text{— The general elimination rule.} \\ \langle proof \rangle \end{array}$$

Elimination of $(a, b) \in A \times B$ – introduces no eigenvariables.

lemma *SigmaD1*: $(a, b) : \text{Sigma } A\ B ==> a : A$
 $\langle proof \rangle$

lemma *SigmaD2*: $(a, b) : \text{Sigma } A\ B ==> b : B\ a$
 $\langle proof \rangle$

lemma *SigmaE2*:

$$\begin{array}{l} [\!] (a, b) : \text{Sigma } A\ B; \\ [\!] a:A;\ b:B(a)\] ==> P \\ [\!] ==> P \\ \langle proof \rangle \end{array}$$

lemma *Sigma-cong*:

$$\begin{array}{l} \llbracket A = B; !!x. x \in B \implies C\ x = D\ x \rrbracket \\ \implies (\text{SIGMA } x: A. C\ x) = (\text{SIGMA } x: B. D\ x) \\ \langle proof \rangle \end{array}$$

lemma *Sigma-mono*: $[\!] A <= C; !!x. x:A ==> B\ x <= D\ x\] ==> \text{Sigma } A\ B$
 $<= \text{Sigma } C\ D$
 $\langle proof \rangle$

lemma *Sigma-empty1* [*simp*]: $\text{Sigma } \{\} B = \{\}$
 $\langle proof \rangle$

lemma *Sigma-empty2* [*simp*]: $A <*> \{\} = \{\}$
 $\langle proof \rangle$

lemma *UNIV-Times-UNIV* [*simp*]: $\text{UNIV } <*> \text{UNIV} = \text{UNIV}$
 $\langle proof \rangle$

lemma *Compl-Times-UNIV1* [*simp*]: $-(\text{UNIV } <*> A) = \text{UNIV } <*> (-A)$
 $\langle proof \rangle$

lemma *Compl-Times-UNIV2* [*simp*]: $-(A <*> \text{UNIV}) = (-A) <*> \text{UNIV}$
 $\langle proof \rangle$

lemma *mem-Sigma-iff* [*iff*]: $((a,b): \text{Sigma } A\ B) = (a:A \ \& \ b:B(a))$
 $\langle proof \rangle$

lemma *Times-subset-cancel2*: $x:C ==> (A <*> C <= B <*> C) = (A <= B)$
 $\langle proof \rangle$

lemma *Times-eq-cancel2*: $x:C ==> (A <*> C = B <*> C) = (A = B)$

$\langle proof \rangle$

lemma *SetCompr-Sigma-eq*:

$Collect (split (\%x y. P x \& Q x y)) = (SIGMA x:Collect P. Collect (Q x))$
 $\langle proof \rangle$

lemma *Collect-split [simp]*: $\{(a,b). P a \& Q b\} = Collect P <*> Collect Q$

$\langle proof \rangle$

lemma *UN-Times-distrib*:

$(UN (a,b):(A <*> B). E a <*> F b) = (UNION A E) <*> (UNION B F)$
 — Suggested by Pierre Chartier

$\langle proof \rangle$

lemma *split-paired-Ball-Sigma [simp,noatp]*:

$(ALL z: Sigma A B. P z) = (ALL x:A. ALL y: B x. P(x,y))$
 $\langle proof \rangle$

lemma *split-paired-Bex-Sigma [simp,noatp]*:

$(EX z: Sigma A B. P z) = (EX x:A. EX y: B x. P(x,y))$
 $\langle proof \rangle$

lemma *Sigma-Un-distrib1*: $(SIGMA i:I Un J. C(i)) = (SIGMA i:I. C(i)) Un (SIGMA j:J. C(j))$

$\langle proof \rangle$

lemma *Sigma-Un-distrib2*: $(SIGMA i:I. A(i) Un B(i)) = (SIGMA i:I. A(i)) Un (SIGMA i:I. B(i))$

$\langle proof \rangle$

lemma *Sigma-Int-distrib1*: $(SIGMA i:I Int J. C(i)) = (SIGMA i:I. C(i)) Int (SIGMA j:J. C(j))$

$\langle proof \rangle$

lemma *Sigma-Int-distrib2*: $(SIGMA i:I. A(i) Int B(i)) = (SIGMA i:I. A(i)) Int (SIGMA i:I. B(i))$

$\langle proof \rangle$

lemma *Sigma-Diff-distrib1*: $(SIGMA i:I - J. C(i)) = (SIGMA i:I. C(i)) - (SIGMA j:J. C(j))$

$\langle proof \rangle$

lemma *Sigma-Diff-distrib2*: $(SIGMA i:I. A(i) - B(i)) = (SIGMA i:I. A(i)) - (SIGMA i:I. B(i))$

$\langle proof \rangle$

lemma *Sigma-Union*: $Sigma (Union X) B = (UN A:X. Sigma A B)$

$\langle proof \rangle$

Non-dependent versions are needed to avoid the need for higher-order match-

ing, especially when the rules are re-oriented.

lemma *Times-Un-distrib1*: $(A \text{ Un } B) <*> C = (A <*> C) \text{ Un } (B <*> C)$
 $\langle \text{proof} \rangle$

lemma *Times-Int-distrib1*: $(A \text{ Int } B) <*> C = (A <*> C) \text{ Int } (B <*> C)$
 $\langle \text{proof} \rangle$

lemma *Times-Diff-distrib1*: $(A - B) <*> C = (A <*> C) - (B <*> C)$
 $\langle \text{proof} \rangle$

lemma *insert-times-insert[simp]*:
 $\text{insert } a \ A \times \text{insert } b \ B =$
 $\text{insert } (a,b) \ (A \times \text{insert } b \ B \cup \text{insert } a \ A \times B)$
 $\langle \text{proof} \rangle$

13.4.2 Code generator setup

instance $*$:: $(eq, eq) \Rightarrow eq$ $\langle \text{proof} \rangle$

lemma *[code]*:
 $eq\text{-class}.eq \ (x1::'a::eq, y1::'b::eq) \ (x2, y2) \longleftrightarrow x1 = x2 \wedge y1 = y2 \ \langle \text{proof} \rangle$

lemma *split-case-cert*:
assumes $CASE \equiv \text{split } f$
shows $CASE \ (a, b) \equiv f \ a \ b$
 $\langle \text{proof} \rangle$

$\langle ML \rangle$

code-type $*$
 $(SML \ \text{infix } 2 \ *)$
 $(OCaml \ \text{infix } 2 \ *)$
 $(Haskell \ !((-), / \ (-)))$

code-instance $*$:: eq
 $(Haskell \ -)$

code-const $eq\text{-class}.eq$:: $'a::eq \times 'b::eq \Rightarrow 'a \times 'b \Rightarrow bool$
 $(Haskell \ \text{infixl } 4 \ ==)$

code-const *Pair*
 $(SML \ !((-), / \ (-)))$
 $(OCaml \ !((-), / \ (-)))$
 $(Haskell \ !((-), / \ (-)))$

code-const *fst and snd*
 $(Haskell \ \text{fst and snd})$

types-code

```

*      ((- */ -))
attach (term-of) ⟨⟨
  fun term-of-id-42 aF aT bF bT (x, y) = HOLogic.pair-const aT bT $ aF x $ bF
  y;
  ⟩⟩
attach (test) ⟨⟨
  fun gen-id-42 aG aT bG bT i =
    let
      val (x, t) = aG i;
      val (y, u) = bG i
    in ((x, y), fn () => HOLogic.pair-const aT bT $ t () $ u ()) end;
  ⟩⟩

```

consts-code

```
Pair      ((-,/ -))
```

⟨ML⟩

13.5 Legacy bindings

⟨ML⟩

13.6 Further inductive packages

⟨ML⟩

end

14 Datatype: Analogues of the Cartesian Product and Disjoint Sum for Datatypes

theory Datatype

imports Nat Product-Type

begin

typedef (Node)

```

('a, 'b) node = {p. EX f x k. p = (f::nat=>'b+nat, x::'a+nat) & f k = Inr 0}
  — it is a subtype of (nat=>'b+nat) * ('a+nat)
  ⟨proof⟩

```

Datatypes will be represented by sets of type *node*

```

types 'a item      = ('a, unit) node set
      ('a, 'b) dtree = ('a, 'b) node set

```

consts

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

Push-Node :: $[(b + \text{nat}), (a, b) \text{ node}] \Rightarrow (a, b) \text{ node}$
ndepth :: $(a, b) \text{ node} \Rightarrow \text{nat}$

Atom :: $(a + \text{nat}) \Rightarrow (a, b) \text{ dtree}$
Leaf :: $a \Rightarrow (a, b) \text{ dtree}$
Numb :: $\text{nat} \Rightarrow (a, b) \text{ dtree}$
Scons :: $[(a, b) \text{ dtree}, (a, b) \text{ dtree}] \Rightarrow (a, b) \text{ dtree}$
In0 :: $(a, b) \text{ dtree} \Rightarrow (a, b) \text{ dtree}$
In1 :: $(a, b) \text{ dtree} \Rightarrow (a, b) \text{ dtree}$
Lim :: $(b \Rightarrow (a, b) \text{ dtree}) \Rightarrow (a, b) \text{ dtree}$

ntrunc :: $[\text{nat}, (a, b) \text{ dtree}] \Rightarrow (a, b) \text{ dtree}$

uprod :: $[(a, b) \text{ dtree set}, (a, b) \text{ dtree set}] \Rightarrow (a, b) \text{ dtree set}$
usum :: $[(a, b) \text{ dtree set}, (a, b) \text{ dtree set}] \Rightarrow (a, b) \text{ dtree set}$

Split :: $[(a, b) \text{ dtree}, (a, b) \text{ dtree}] \Rightarrow c, (a, b) \text{ dtree} \Rightarrow c$
Case :: $[(a, b) \text{ dtree}] \Rightarrow c, [(a, b) \text{ dtree}] \Rightarrow c, (a, b) \text{ dtree} \Rightarrow c$

dprod :: $[((a, b) \text{ dtree} * (a, b) \text{ dtree}) \text{ set}, ((a, b) \text{ dtree} * (a, b) \text{ dtree}) \text{ set}]$
 $\Rightarrow ((a, b) \text{ dtree} * (a, b) \text{ dtree}) \text{ set}$
dsum :: $[((a, b) \text{ dtree} * (a, b) \text{ dtree}) \text{ set}, ((a, b) \text{ dtree} * (a, b) \text{ dtree}) \text{ set}]$
 $\Rightarrow ((a, b) \text{ dtree} * (a, b) \text{ dtree}) \text{ set}$

defs

Push-Node-def: $\text{Push-Node} == (\%n \ x. \text{Abs-Node} (\text{apfst} (\text{Push } n) (\text{Rep-Node } x)))$

Push-def: $\text{Push} == (\%b \ h. \text{nat-case } b \ h)$

Atom-def: $\text{Atom} == (\%x. \{\text{Abs-Node}((\%k. \text{Inr } 0, x))\})$
Scons-def: $\text{Scons } M \ N == (\text{Push-Node } (\text{Inr } 1) \text{ ' } M) \ \text{Un} \ (\text{Push-Node } (\text{Inr } (\text{Suc } 1)) \text{ ' } N)$

Leaf-def: $\text{Leaf} == \text{Atom } o \ \text{Inl}$
Numb-def: $\text{Numb} == \text{Atom } o \ \text{Inr}$

In0-def: $\text{In0}(M) == \text{Scons } (\text{Numb } 0) \ M$
In1-def: $\text{In1}(M) == \text{Scons } (\text{Numb } 1) \ M$

Lim-def: $\text{Lim } f == \text{Union } \{z. ? x. z = \text{Push-Node } (\text{Inl } x) \text{ ' } (f x)\}$

ndepth-def: $\text{ndepth}(n) == (\% (f, x). \text{LEAST } k. f k = \text{Inr } 0) (\text{Rep-Node } n)$
ntrunc-def: $\text{ntrunc } k N == \{n. n:N \ \& \ \text{ndepth}(n) < k\}$

uprod-def: $\text{uprod } A B == \text{UN } x:A. \text{UN } y:B. \{ \text{Scons } x y \}$
usum-def: $\text{usum } A B == \text{Inl } 0 ' A \text{ Un } \text{Inl } 1 ' B$

Split-def: $\text{Split } c M == \text{THE } u. \text{EX } x y. M = \text{Scons } x y \ \& \ u = c \ x \ y$

Case-def: $\text{Case } c d M == \text{THE } u. (\text{EX } x. M = \text{Inl } 0 (x) \ \& \ u = c(x))$
 $\quad \quad \quad | (\text{EX } y. M = \text{Inl } 1 (y) \ \& \ u = d(y))$

dprod-def: $\text{dprod } r s == \text{UN } (x, x'):r. \text{UN } (y, y'):s. \{(\text{Scons } x y, \text{Scons } x' y')\}$

dsum-def: $\text{dsum } r s == (\text{UN } (x, x'):r. \{(\text{Inl } 0 (x), \text{Inl } 0 (x'))\}) \text{ Un }$
 $\quad \quad \quad (\text{UN } (y, y'):s. \{(\text{Inl } 1 (y), \text{Inl } 1 (y'))\})$

lemma *apfst-convE:*

$\llbracket q = \text{apfst } f p; \ !\!x \ y. \llbracket p = (x, y); \ q = (f(x), y) \rrbracket ==> R$
 $\llbracket \rrbracket ==> R$
 $\langle \text{proof} \rangle$

lemma *Push-inject1:* $\text{Push } i f = \text{Push } j g ==> i=j$
 $\langle \text{proof} \rangle$

lemma *Push-inject2:* $\text{Push } i f = \text{Push } j g ==> f=g$
 $\langle \text{proof} \rangle$

lemma *Push-inject:*

$\llbracket \text{Push } i f = \text{Push } j g; \llbracket i=j; \ f=g \rrbracket ==> P \rrbracket ==> P$
 $\langle \text{proof} \rangle$

lemma *Push-neq-K0:* $\text{Push } (\text{Inr } (\text{Suc } k)) f = (\%z. \text{Inr } 0) ==> P$
 $\langle \text{proof} \rangle$

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

lemma *Node-K0-I*: ($\%k. \text{Inr } 0, a$) : *Node*
 $\langle \text{proof} \rangle$

lemma *Node-Push-I*: $p : \text{Node} \implies \text{apfst } (\text{Push } i) p : \text{Node}$
 $\langle \text{proof} \rangle$

14.1 Freeness: Distinctness of Constructors

lemma *Scons-not-Atom* [*iff*]: $\text{Scons } M N \neq \text{Atom}(a)$
 $\langle \text{proof} \rangle$

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

lemma *inj-Atom*: $\text{inj}(\text{Atom})$
 $\langle \text{proof} \rangle$

lemmas *Atom-inject* = *inj-Atom* [*THEN injD, standard*]

lemma *Atom-Atom-eq* [*iff*]: $(\text{Atom}(a) = \text{Atom}(b)) = (a = b)$
 $\langle \text{proof} \rangle$

lemma *inj-Leaf*: $\text{inj}(\text{Leaf})$
 $\langle \text{proof} \rangle$

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

lemma *inj-Numb*: $\text{inj}(\text{Numb})$
 $\langle \text{proof} \rangle$

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

lemma *Push-Node-inject*:

$$[\text{Push-Node } i m = \text{Push-Node } j n; [\text{ } i=j; m=n \text{ }] \implies P] \implies P$$
 $\langle \text{proof} \rangle$

lemma *Scons-inject-lemma1*: $\text{Scons } M N \leq \text{Scons } M' N' \implies M \leq M'$

$\langle proof \rangle$

lemma *Scons-inject-lemma2*: $Scons\ M\ N\ \leq\ Scons\ M'\ N' \implies N\ \leq\ N'$
 $\langle proof \rangle$

lemma *Scons-inject1*: $Scons\ M\ N = Scons\ M'\ N' \implies M = M'$
 $\langle proof \rangle$

lemma *Scons-inject2*: $Scons\ M\ N = Scons\ M'\ N' \implies N = N'$
 $\langle proof \rangle$

lemma *Scons-inject*:
 $[[Scons\ M\ N = Scons\ M'\ N';\ [[M = M';\ N = N']] \implies P]] \implies P$
 $\langle proof \rangle$

lemma *Scons-Scons-eq* [iff]: $(Scons\ M\ N = Scons\ M'\ N') = (M = M' \ \&\ N = N')$
 $\langle proof \rangle$

lemma *Scons-not-Leaf* [iff]: $Scons\ M\ N \neq Leaf(a)$
 $\langle proof \rangle$

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

lemma *Scons-not-Numb* [iff]: $Scons\ M\ N \neq Numb(k)$
 $\langle proof \rangle$

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

lemma *Leaf-not-Numb* [iff]: $Leaf(a) \neq Numb(k)$
 $\langle proof \rangle$

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

lemma *ndepth-K0*: $ndepth\ (Abs-Node(\%k.\ Inr\ 0,\ x)) = 0$
 $\langle proof \rangle$

lemma *ndepth-Push-Node-aux*:

nat-case (*Inr* (*Suc i*)) *f k* = *Inr 0* $\dashv\dashv$ *Suc*(*LEAST x. f x* = *Inr 0*) \leq *k*
 $\langle proof \rangle$

lemma *ndepth-Push-Node*:
ndepth (*Push-Node* (*Inr* (*Suc i*)) *n*) = *Suc*(*ndepth*(*n*))
 $\langle proof \rangle$

lemma *ntrunc-0* [*simp*]: *ntrunc 0 M* = {}
 $\langle proof \rangle$

lemma *ntrunc-Atom* [*simp*]: *ntrunc* (*Suc k*) (*Atom a*) = *Atom*(*a*)
 $\langle proof \rangle$

lemma *ntrunc-Leaf* [*simp*]: *ntrunc* (*Suc k*) (*Leaf a*) = *Leaf*(*a*)
 $\langle proof \rangle$

lemma *ntrunc-Numb* [*simp*]: *ntrunc* (*Suc k*) (*Numb i*) = *Numb*(*i*)
 $\langle proof \rangle$

lemma *ntrunc-Scons* [*simp*]:
ntrunc (*Suc k*) (*Scons M N*) = *Scons* (*ntrunc k M*) (*ntrunc k N*)
 $\langle proof \rangle$

lemma *ntrunc-one-In0* [*simp*]: *ntrunc* (*Suc 0*) (*In0 M*) = {}
 $\langle proof \rangle$

lemma *ntrunc-In0* [*simp*]: *ntrunc* (*Suc*(*Suc k*)) (*In0 M*) = *In0* (*ntrunc* (*Suc k*)
M)
 $\langle proof \rangle$

lemma *ntrunc-one-In1* [*simp*]: *ntrunc* (*Suc 0*) (*In1 M*) = {}
 $\langle proof \rangle$

lemma *ntrunc-In1* [*simp*]: *ntrunc* (*Suc*(*Suc k*)) (*In1 M*) = *In1* (*ntrunc* (*Suc k*)
M)
 $\langle proof \rangle$

14.2 Set Constructions

lemma *uprodI* [*intro!*]: [*M:A; N:B*] \implies *Scons M N* : *uprod A B*
 $\langle proof \rangle$

lemma *uprodE* [*elim!*]:

$$\begin{aligned} & [| c : \text{uprod } A \ B; \\ & \quad !!x \ y. [| x:A; \ y:B; \ c = \text{Scons } x \ y \ |] ==> P \\ & \quad |] ==> P \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *uprodE2*: $[| \text{Scons } M \ N : \text{uprod } A \ B; \ [| M:A; \ N:B \ |] ==> P \ |] ==> P$
 $\langle \text{proof} \rangle$

lemma *usum-In0I* [*intro*]: $M:A ==> \text{In0}(M) : \text{usum } A \ B$
 $\langle \text{proof} \rangle$

lemma *usum-In1I* [*intro*]: $N:B ==> \text{In1}(N) : \text{usum } A \ B$
 $\langle \text{proof} \rangle$

lemma *usumE* [*elim!*]:

$$\begin{aligned} & [| u : \text{usum } A \ B; \\ & \quad !!x. [| x:A; \ u=\text{In0}(x) \ |] ==> P; \\ & \quad !!y. [| y:B; \ u=\text{In1}(y) \ |] ==> P \\ & \quad |] ==> P \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *In0-not-In1* [*iff*]: $\text{In0}(M) \neq \text{In1}(N)$
 $\langle \text{proof} \rangle$

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

lemma *In0-inject*: $\text{In0}(M) = \text{In0}(N) ==> M=N$
 $\langle \text{proof} \rangle$

lemma *In1-inject*: $\text{In1}(M) = \text{In1}(N) ==> M=N$
 $\langle \text{proof} \rangle$

lemma *In0-eq* [*iff*]: $(\text{In0 } M = \text{In0 } N) = (M=N)$
 $\langle \text{proof} \rangle$

lemma *In1-eq* [*iff*]: $(\text{In1 } M = \text{In1 } N) = (M=N)$
 $\langle \text{proof} \rangle$

lemma *inj-In0*: *inj* *In0*

$\langle proof \rangle$

lemma *inj-In1*: *inj In1*
 $\langle proof \rangle$

lemma *Lim-inject*: *Lim f = Lim g ==> f = g*
 $\langle proof \rangle$

lemma *ntrunc-subsetI*: *ntrunc k M <= M*
 $\langle proof \rangle$

lemma *ntrunc-subsetD*: *(!!k. ntrunc k M <= N) ==> M <= N*
 $\langle proof \rangle$

lemma *ntrunc-equality*: *(!!k. ntrunc k M = ntrunc k N) ==> M = N*
 $\langle proof \rangle$

lemma *ntrunc-o-equality*:
 $\llbracket !!k. (ntrunc(k) \circ h1) = (ntrunc(k) \circ h2) \rrbracket ==> h1 = h2$
 $\langle proof \rangle$

lemma *uprod-mono*: $\llbracket A <= A'; B <= B' \rrbracket ==> uprod A B <= uprod A' B'$
 $\langle proof \rangle$

lemma *usum-mono*: $\llbracket A <= A'; B <= B' \rrbracket ==> usum A B <= usum A' B'$
 $\langle proof \rangle$

lemma *Scons-mono*: $\llbracket M <= M'; N <= N' \rrbracket ==> Scons M N <= Scons M' N'$
 $\langle proof \rangle$

lemma *In0-mono*: *M <= N ==> In0(M) <= In0(N)*
 $\langle proof \rangle$

lemma *In1-mono*: *M <= N ==> In1(M) <= In1(N)*
 $\langle proof \rangle$

lemma *Split* [simp]: $\text{Split } c \ (\text{Scons } M \ N) = c \ M \ N$
 $\langle \text{proof} \rangle$

lemma *Case-In0* [simp]: $\text{Case } c \ d \ (\text{In0 } M) = c(M)$
 $\langle \text{proof} \rangle$

lemma *Case-In1* [simp]: $\text{Case } c \ d \ (\text{In1 } N) = d(N)$
 $\langle \text{proof} \rangle$

lemma *ntrunc-UN1*: $\text{ntrunc } k \ (\text{UN } x. f(x)) = (\text{UN } x. \text{ntrunc } k \ (f \ x))$
 $\langle \text{proof} \rangle$

lemma *Scons-UN1-x*: $\text{Scons } (\text{UN } x. f \ x) \ M = (\text{UN } x. \text{Scons } (f \ x) \ M)$
 $\langle \text{proof} \rangle$

lemma *Scons-UN1-y*: $\text{Scons } M \ (\text{UN } x. f \ x) = (\text{UN } x. \text{Scons } M \ (f \ x))$
 $\langle \text{proof} \rangle$

lemma *In0-UN1*: $\text{In0}(\text{UN } x. f(x)) = (\text{UN } x. \text{In0}(f(x)))$
 $\langle \text{proof} \rangle$

lemma *In1-UN1*: $\text{In1}(\text{UN } x. f(x)) = (\text{UN } x. \text{In1}(f(x)))$
 $\langle \text{proof} \rangle$

lemma *dprodI* [intro!]:
 $\llbracket (M, M'):r; \ (N, N'):s \rrbracket \implies (\text{Scons } M \ N, \text{Scons } M' \ N') : \text{dprod } r \ s$
 $\langle \text{proof} \rangle$

lemma *dprodE* [elim!]:
 $\llbracket c : \text{dprod } r \ s; \quad \begin{array}{l} \text{!!}x \ y \ x' \ y'. \llbracket (x, x') : r; \ (y, y') : s; \\ \quad c = (\text{Scons } x \ y, \text{Scons } x' \ y') \rrbracket \implies P \end{array} \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *dsum-In0I* [intro]: $(M, M'):r \implies (\text{In0}(M), \text{In0}(M')) : \text{dsum } r \ s$
 $\langle \text{proof} \rangle$

lemma *dsum-In1I* [intro]: $(N, N') : s ==> (In1(N), In1(N')) : dsum\ r\ s$
 $\langle proof \rangle$

lemma *dsumE* [elim!]:

$$\begin{aligned} &[[\ w : dsum\ r\ s; \\ &\quad !!x\ x'.\ [[\ (x, x') : r;\ w = (In0(x), In0(x'))\]\] ==> P; \\ &\quad !!y\ y'.\ [[\ (y, y') : s;\ w = (In1(y), In1(y'))\]\] ==> P \\ &\quad]\] ==> P \end{aligned}$$

 $\langle proof \rangle$

lemma *dprod-mono*: $[[\ r <= r';\ s <= s'\]\] ==> dprod\ r\ s <= dprod\ r'\ s'$
 $\langle proof \rangle$

lemma *dsum-mono*: $[[\ r <= r';\ s <= s'\]\] ==> dsum\ r\ s <= dsum\ r'\ s'$
 $\langle proof \rangle$

lemma *dprod-Sigma*: $(dprod\ (A\ <*>\ B)\ (C\ <*>\ D)) <= (uprod\ A\ C)\ <*>\ (uprod\ B\ D)$
 $\langle proof \rangle$

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

lemma *dprod-subset-Sigma2*:

$$(dprod\ (Sigma\ A\ B)\ (Sigma\ C\ D)) <=$$

$$Sigma\ (uprod\ A\ C)\ (Split\ (\%x\ y.\ uprod\ (B\ x)\ (D\ y)))$$

 $\langle proof \rangle$

lemma *dsum-Sigma*: $(dsum\ (A\ <*>\ B)\ (C\ <*>\ D)) <= (usum\ A\ C)\ <*>\ (usum\ B\ D)$
 $\langle proof \rangle$

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

hides popular names

hide (**open**) *type node item*

hide (**open**) *const Push Node Atom Leaf Numb Lim Split Case*

15 Datatypes

15.1 Representing sums

rep-datatype (*sum*) *Inl Inr*
 $\langle proof \rangle$

lemma *sum-case-KK[simp]*: *sum-case* ($\%x. a$) ($\%x. a$) = ($\%x. a$)
 $\langle proof \rangle$

lemma *surjective-sum*: *sum-case* ($\%x::'a. f (Inl x)$) ($\%y::'b. f (Inr y)$) *s* = *f(s)*
 $\langle proof \rangle$

lemma *sum-case-weak-cong*: *s* = *t* \implies *sum-case f g s* = *sum-case f g t*
 — Prevents simplification of *f* and *g*: much faster.
 $\langle proof \rangle$

lemma *sum-case-inject*:
 $sum_case\ f1\ f2 = sum_case\ g1\ g2 \implies (f1 = g1 \implies f2 = g2 \implies P) \implies P$
 $\langle proof \rangle$

constdefs

Suml :: ($'a \Rightarrow 'c$) \Rightarrow $'a + 'b \Rightarrow 'c$
Suml == ($\%f. sum_case\ f\ undefined$)

Sumr :: ($'b \Rightarrow 'c$) \Rightarrow $'a + 'b \Rightarrow 'c$
Sumr == *sum-case undefined*

lemma *Suml-inject*: *Suml f* = *Suml g* $\implies f = g$
 $\langle proof \rangle$

lemma *Sumr-inject*: *Sumr f* = *Sumr g* $\implies f = g$
 $\langle proof \rangle$

primrec *Projl* :: $'a + 'b \Rightarrow 'a$
where *Projl-Inl*: *Projl (Inl x)* = *x*

primrec *Projr* :: $'a + 'b \Rightarrow 'b$
where *Projr-Inr*: *Projr (Inr x)* = *x*

hide (open) *const Suml Sumr Projl Projr*

end

16 Power: Exponentiation

theory *Power*

```

imports Nat
begin

class power =
  fixes power :: 'a  $\Rightarrow$  nat  $\Rightarrow$  'a          (infixr ^ 80)

```

16.1 Powers for Arbitrary Monoids

```

class recpower = monoid-mult + power +
  assumes power-0 [simp]: a ^ 0 = 1
  assumes power-Suc [simp]: a ^ Suc n = a * (a ^ n)

lemma power-0-Suc [simp]: (0::'a::{recpower,semiring-0}) ^ (Suc n) = 0
  <proof>

```

It looks plausible as a simprule, but its effect can be strange.

```

lemma power-0-left: 0 ^ n = (if n=0 then 1 else (0::'a::{recpower,semiring-0}))
  <proof>

```

```

lemma power-one [simp]: 1 ^ n = (1::'a::recpower)
  <proof>

```

```

lemma power-one-right [simp]: (a::'a::recpower) ^ 1 = a
  <proof>

```

```

lemma power-commutes: (a::'a::recpower) ^ n * a = a * a ^ n
  <proof>

```

```

lemma power-Suc2: (a::'a::recpower) ^ Suc n = a ^ n * a
  <proof>

```

```

lemma power-add: (a::'a::recpower) ^ (m+n) = (a ^ m) * (a ^ n)
  <proof>

```

```

lemma power-mult: (a::'a::recpower) ^ (m*n) = (a ^ m) ^ n
  <proof>

```

```

lemma power-mult-distrib: ((a::'a::{recpower,comm-monoid-mult}) * b) ^ n =
  (a ^ n) * (b ^ n)
  <proof>

```

```

lemma zero-less-power[simp]:
  0 < (a::'a::{ordered-semidom,recpower}) ==> 0 < a ^ n
  <proof>

```

```

lemma zero-le-power[simp]:
  0 ≤ (a::'a::{ordered-semidom,recpower}) ==> 0 ≤ a ^ n
  <proof>

```

lemma *one-le-power* [simp]:

$1 \leq (a::'a::\{\text{ordered-semidom}, \text{recpower}\}) \implies 1 \leq a^n$
 <proof>

lemma *gt1-imp-ge0*: $1 < a \implies 0 \leq (a::'a::\text{ordered-semidom})$
 <proof>

lemma *power-gt1-lemma*:

assumes *gt1*: $1 < (a::'a::\{\text{ordered-semidom}, \text{recpower}\})$
shows $1 < a * a^n$
 <proof>

lemma *one-less-power* [simp]:

$\llbracket 1 < (a::'a::\{\text{ordered-semidom}, \text{recpower}\}); 0 < n \rrbracket \implies 1 < a^n$
 <proof>

lemma *power-gt1*:

$1 < (a::'a::\{\text{ordered-semidom}, \text{recpower}\}) \implies 1 < a^{(\text{Suc } n)}$
 <proof>

lemma *power-le-imp-le-exp*:

assumes *gt1*: $(1::'a::\{\text{recpower}, \text{ordered-semidom}\}) < a$
shows $!!n. a^m \leq a^n \implies m \leq n$
 <proof>

Surely we can strengthen this? It holds for $0 < a < 1$ too.

lemma *power-inject-exp* [simp]:

$1 < (a::'a::\{\text{ordered-semidom}, \text{recpower}\}) \implies (a^m = a^n) = (m = n)$
 <proof>

Can relax the first premise to $(0::'a) < a$ in the case of the natural numbers.

lemma *power-less-imp-less-exp*:

$\llbracket (1::'a::\{\text{recpower}, \text{ordered-semidom}\}) < a; a^m < a^n \rrbracket \implies m < n$
 <proof>

lemma *power-mono*:

$\llbracket a \leq b; (0::'a::\{\text{recpower}, \text{ordered-semidom}\}) \leq a \rrbracket \implies a^n \leq b^n$
 <proof>

lemma *power-strict-mono* [rule-format]:

$\llbracket a < b; (0::'a::\{\text{recpower}, \text{ordered-semidom}\}) \leq a \rrbracket$
 $\implies 0 < n \longrightarrow a^n < b^n$
 <proof>

lemma *power-eq-0-iff* [simp]:

$(a^n = 0) \longleftrightarrow$
 $(a = (0::'a::\{\text{mult-zero}, \text{zero-neq-one}, \text{no-zero-divisors}, \text{recpower}\}) \ \& \ n \neq 0)$
 <proof>

lemma *field-power-not-zero*:

$a \neq 0 \implies (a :: 'a :: \{\text{ring-1-no-zero-divisors}, \text{recpower}\}) \implies a^n \neq 0$
 <proof>

lemma *nonzero-power-inverse*:

fixes $a :: 'a :: \{\text{division-ring}, \text{recpower}\}$
shows $a \neq 0 \implies \text{inverse } (a^n) = (\text{inverse } a)^n$
 <proof>

Perhaps these should be simprules.

lemma *power-inverse*:

fixes $a :: 'a :: \{\text{division-ring}, \text{division-by-zero}, \text{recpower}\}$
shows $\text{inverse } (a^n) = (\text{inverse } a)^n$
 <proof>

lemma *power-one-over*: $1 / (a :: 'a :: \{\text{field}, \text{division-by-zero}, \text{recpower}\})^n =$
 $(1 / a)^n$
 <proof>

lemma *nonzero-power-divide*:

$b \neq 0 \implies (a/b)^n = ((a :: 'a :: \{\text{field}, \text{recpower}\})^n) / (b^n)$
 <proof>

lemma *power-divide*:

$(a/b)^n = ((a :: 'a :: \{\text{field}, \text{division-by-zero}, \text{recpower}\})^n) / b^n$
 <proof>

lemma *power-abs*: $\text{abs}(a^n) = \text{abs}(a :: 'a :: \{\text{ordered-idom}, \text{recpower}\})^n$
 <proof>

lemma *abs-power-minus* [simp]:

fixes $a :: 'a :: \{\text{ordered-idom}, \text{recpower}\}$ **shows** $\text{abs}((-a)^n) = \text{abs}(a^n)$
 <proof>

lemma *zero-less-power-abs-iff* [simp, noatp]:

$(0 < (\text{abs } a)^n) = (a \neq 0 \implies (a :: 'a :: \{\text{ordered-idom}, \text{recpower}\}) \mid n=0)$
 <proof>

lemma *zero-le-power-abs* [simp]:

$(0 :: 'a :: \{\text{ordered-idom}, \text{recpower}\}) \leq (\text{abs } a)^n$
 <proof>

lemma *power-minus*: $(-a)^n = (-1)^n * (a :: 'a :: \{\text{ring-1}, \text{recpower}\})^n$
 <proof>

Lemma for *power-strict-decreasing*

lemma *power-Suc-less*:

$$[(0::'a::\{\text{ordered-semidom}, \text{recpower}\}) < a; a < 1] \\ \implies a * a^n < a^n$$
 <proof>

lemma *power-strict-decreasing*:

$$[n < N; 0 < a; a < (1::'a::\{\text{ordered-semidom}, \text{recpower}\})] \\ \implies a^N < a^n$$
 <proof>

Proof resembles that of *power-strict-decreasing*

lemma *power-decreasing*:

$$[n \leq N; 0 \leq a; a \leq (1::'a::\{\text{ordered-semidom}, \text{recpower}\})] \\ \implies a^N \leq a^n$$
 <proof>

lemma *power-Suc-less-one*:

$$[0 < a; a < (1::'a::\{\text{ordered-semidom}, \text{recpower}\})] \implies a^{\text{Suc } n} < 1$$
 <proof>

Proof again resembles that of *power-strict-decreasing*

lemma *power-increasing*:

$$[n \leq N; (1::'a::\{\text{ordered-semidom}, \text{recpower}\}) \leq a] \implies a^n \leq a^N$$
 <proof>

Lemma for *power-strict-increasing*

lemma *power-less-power-Suc*:

$$(1::'a::\{\text{ordered-semidom}, \text{recpower}\}) < a \implies a^n < a * a^n$$
 <proof>

lemma *power-strict-increasing*:

$$[n < N; (1::'a::\{\text{ordered-semidom}, \text{recpower}\}) < a] \implies a^n < a^N$$
 <proof>

lemma *power-increasing-iff [simp]*:

$$1 < (b::'a::\{\text{ordered-semidom}, \text{recpower}\}) \implies (b^x \leq b^y) = (x \leq y)$$
 <proof>

lemma *power-strict-increasing-iff [simp]*:

$$1 < (b::'a::\{\text{ordered-semidom}, \text{recpower}\}) \implies (b^x < b^y) = (x < y)$$
 <proof>

lemma *power-le-imp-le-base*:

assumes *le*: $a^{\text{Suc } n} \leq b^{\text{Suc } n}$

and *ynonneg*: $(0::'a::\{\text{ordered-semidom}, \text{recpower}\}) \leq b$

shows $a \leq b$

<proof>

lemma *power-less-imp-less-base*:

fixes $a \ b :: 'a::\{\text{ordered-semidom}, \text{recpower}\}$

assumes *less*: $a^n < b^n$
assumes *nonneg*: $0 \leq b$
shows $a < b$
 <proof>

lemma *power-inject-base*:
 $\llbracket a^{Suc\ n} = b^{Suc\ n}; 0 \leq a; 0 \leq b \rrbracket$
 $\implies a = (b :: 'a :: \{ordered-semidom, recpower\})$
 <proof>

lemma *power-eq-imp-eq-base*:
fixes $a\ b :: 'a :: \{ordered-semidom, recpower\}$
shows $\llbracket a^n = b^n; 0 \leq a; 0 \leq b; 0 < n \rrbracket \implies a = b$
 <proof>

The divides relation

lemma *le-imp-power-dvd*:
fixes $a :: 'a :: \{comm-semiring-1, recpower\}$
assumes $m \leq n$ **shows** $a^m \text{ dvd } a^n$
 <proof>

lemma *power-le-dvd*:
fixes $a\ b :: 'a :: \{comm-semiring-1, recpower\}$
shows $a^n \text{ dvd } b \implies m \leq n \implies a^m \text{ dvd } b$
 <proof>

lemma *dvd-power-same*:
 $(x :: 'a :: \{comm-semiring-1, recpower\}) \text{ dvd } y \implies x^n \text{ dvd } y^n$
 <proof>

lemma *dvd-power-le*:
 $(x :: 'a :: \{comm-semiring-1, recpower\}) \text{ dvd } y \implies m \geq n \implies x^n \text{ dvd } y^m$
 <proof>

lemma *dvd-power [simp]*:
 $n > 0 \mid (x :: 'a :: \{comm-semiring-1, recpower\}) = 1 \implies x \text{ dvd } x^n$
 <proof>

16.2 Exponentiation for the Natural Numbers

instantiation *nat* :: *recpower*
begin

primrec *power-nat* **where**
 $p^0 = (1 :: nat)$
 $\mid p^{(Suc\ n)} = (p :: nat) * (p^n)$

instance <proof>

declare *power-nat.simps* [*simp del*]

end

lemma *of-nat-power*:

of-nat ($m \wedge n$) = (*of-nat* $m::'a::\{\text{semiring-1}, \text{recpower}\}$) $\wedge n$
 $\langle \text{proof} \rangle$

lemma *nat-one-le-power* [*simp*]: $\text{Suc } 0 \leq i \implies \text{Suc } 0 \leq i \wedge n$
 $\langle \text{proof} \rangle$

lemma *nat-zero-less-power-iff* [*simp*]: $(x \wedge n > 0) = (x > (0::\text{nat}) \mid n=0)$
 $\langle \text{proof} \rangle$

lemma *nat-power-eq-Suc-0-iff* [*simp*]:
 $((x::\text{nat}) \wedge m = \text{Suc } 0) = (m = 0 \mid x = \text{Suc } 0)$
 $\langle \text{proof} \rangle$

lemma *power-Suc-0* [*simp*]: $(\text{Suc } 0) \wedge n = \text{Suc } 0$
 $\langle \text{proof} \rangle$

Valid for the naturals, but what if $0 < i < 1$? Premises cannot be weakened:
 consider the case where $i = (0::'a)$, $m = (1::'a)$ and $n = (0::'a)$.

lemma *nat-power-less-imp-less*:

assumes *nonneg*: $0 < (i::\text{nat})$
assumes *less*: $i \wedge m < i \wedge n$
shows $m < n$
 $\langle \text{proof} \rangle$

lemma *power-diff*:

assumes *nz*: $a \sim 0$
shows $n \leq m \implies (a::'a::\{\text{recpower}, \text{field}\}) \wedge (m-n) = (a \wedge m) / (a \wedge n)$
 $\langle \text{proof} \rangle$

end

17 Finite-Set: Finite sets

theory *Finite-Set*

imports *Nat Product-Type Power*

begin

17.1 Definition and basic properties

inductive *finite* :: $'a \text{ set} \implies \text{bool}$

where

emptyI [*simp, intro!*]: *finite* $\{\}$

| *insertI* [*simp*, *intro!*]: *finite A* ==> *finite (insert a A)*

lemma *ex-new-if-finite*: — does not depend on def of finite at all
assumes $\neg \text{finite } (UNIV :: 'a \text{ set})$ **and** *finite A*
shows $\exists a :: 'a. a \notin A$
 <proof>

lemma *finite-induct* [*case-names empty insert, induct set: finite*]:
finite F ==>
 $P \{ \} ==> (!x F. \text{finite } F ==> x \notin F ==> P F ==> P (\text{insert } x F)) ==>$
 $P F$
 — Discharging $x \notin F$ entails extra work.
 <proof>

lemma *finite-ne-induct*[*case-names singleton insert, consumes 2*]:
assumes *fin: finite F* **shows** $F \neq \{ \} \implies$
 $\llbracket \bigwedge x. P \{x\};$
 $\bigwedge x F. \llbracket \text{finite } F; F \neq \{ \}; x \notin F; P F \rrbracket \implies P (\text{insert } x F) \rrbracket$
 $\implies P F$
 <proof>

lemma *finite-subset-induct* [*consumes 2, case-names empty insert*]:
assumes *finite F* **and** $F \subseteq A$
and empty: $P \{ \}$
and insert: $!!a F. \text{finite } F ==> a \in A ==> a \notin F ==> P F ==> P (\text{insert } a F)$
shows $P F$
 <proof>

A finite choice principle. Does not need the SOME choice operator.

lemma *finite-set-choice*:
 $\text{finite } A \implies \text{ALL } x:A. (\text{EX } y. P x y) \implies \text{EX } f. \text{ALL } x:A. P x (f x)$
 <proof>

Finite sets are the images of initial segments of natural numbers:

lemma *finite-imp-nat-seg-image-inj-on*:
assumes *fin: finite A*
shows $\exists (n :: \text{nat}) f. A = f ` \{i. i < n\} \ \& \ \text{inj-on } f \ \{i. i < n\}$
 <proof>

lemma *nat-seg-image-imp-finite*:
 $!!f A. A = f ` \{i :: \text{nat}. i < n\} \implies \text{finite } A$
 <proof>

lemma *finite-conv-nat-seg-image*:
 $\text{finite } A = (\exists (n :: \text{nat}) f. A = f ` \{i :: \text{nat}. i < n\})$
 <proof>

lemma *finite-Collect-less-nat*[*iff*]: $\text{finite} \{n :: \text{nat}. n < k\}$

lemma *finite-Diff2 [simp]*:
assumes *finite B* **shows** *finite (A - B) = finite A*

$\langle \text{proof} \rangle$

lemma *finite-compl*[simp]:

$\text{finite}(A::'a \text{ set}) \implies \text{finite}(-A) = \text{finite}(\text{UNIV}::'a \text{ set})$
 $\langle \text{proof} \rangle$

lemma *finite-Collect-not*[simp]:

$\text{finite}\{x::'a. P\ x\} \implies \text{finite}\{x. \sim P\ x\} = \text{finite}(\text{UNIV}::'a \text{ set})$
 $\langle \text{proof} \rangle$

lemma *finite-Diff-insert* [iff]: $\text{finite}\ (A - \text{insert}\ a\ B) = \text{finite}\ (A - B)$
 $\langle \text{proof} \rangle$

Image and Inverse Image over Finite Sets

lemma *finite-imageI*[simp]: $\text{finite}\ F \implies \text{finite}\ (h\ ' F)$
 — The image of a finite set is finite.
 $\langle \text{proof} \rangle$

lemma *finite-surj*: $\text{finite}\ A \implies B \leq f\ ' A \implies \text{finite}\ B$
 $\langle \text{proof} \rangle$

lemma *finite-range-imageI*:

$\text{finite}\ (\text{range}\ g) \implies \text{finite}\ (\text{range}\ (\%x. f\ (g\ x)))$
 $\langle \text{proof} \rangle$

lemma *finite-imageD*: $\text{finite}\ (f\ ' A) \implies \text{inj-on}\ f\ A \implies \text{finite}\ A$
 $\langle \text{proof} \rangle$

lemma *inj-vimage-singleton*: $\text{inj}\ f \implies f\ -'\{a\} \subseteq \{\text{THE}\ x. f\ x = a\}$

— The inverse image of a singleton under an injective function is included in a singleton.
 $\langle \text{proof} \rangle$

lemma *finite-vimageI*: $[\text{finite}\ F; \text{inj}\ h] \implies \text{finite}\ (h\ -'\ F)$

— The inverse image of a finite set under an injective function is finite.
 $\langle \text{proof} \rangle$

The finite UNION of finite sets

lemma *finite-UN-I*: $\text{finite}\ A \implies (!a. a:A \implies \text{finite}\ (B\ a)) \implies \text{finite}\ (\text{UNION}\ A\ B)$
 $\langle \text{proof} \rangle$

Strengthen RHS to $(\forall x \in A. \text{finite}\ (B\ x)) \wedge \text{finite}\ \{x \in A. B\ x \neq \{\}\}$?

We'd need to prove $\text{finite}\ C \implies \forall A\ B. \text{UNION}\ A\ B \subseteq C \longrightarrow \text{finite}\ \{x \in A. B\ x \neq \{\}\}$ by induction.

lemma *finite-UN* [simp]:

$\text{finite}\ A \implies \text{finite}\ (\text{UNION}\ A\ B) = (\text{ALL}\ x:A. \text{finite}\ (B\ x))$

$\langle \text{proof} \rangle$

lemma *finite-Collect-bex*[simp]: $\text{finite } A \implies$
 $\text{finite}\{x. \text{EX } y:A. Q \ x \ y\} = (\text{ALL } y:A. \text{finite}\{x. Q \ x \ y\})$
 $\langle \text{proof} \rangle$

lemma *finite-Collect-bounded-ex*[simp]: $\text{finite}\{y. P \ y\} \implies$
 $\text{finite}\{x. \text{EX } y. P \ y \ \& \ Q \ x \ y\} = (\text{ALL } y. P \ y \longrightarrow \text{finite}\{x. Q \ x \ y\})$
 $\langle \text{proof} \rangle$

lemma *finite-Plus*: $[\text{finite } A; \text{finite } B] \implies \text{finite } (A <+> B)$
 $\langle \text{proof} \rangle$

Sigma of finite sets

lemma *finite-SigmaI* [simp]:
 $\text{finite } A \implies (!a. a:A \implies \text{finite } (B \ a)) \implies \text{finite } (\text{SIGMA } a:A. B \ a)$
 $\langle \text{proof} \rangle$

lemma *finite-cartesian-product*: $[\text{finite } A; \text{finite } B] \implies$
 $\text{finite } (A <*> B)$
 $\langle \text{proof} \rangle$

lemma *finite-Prod-UNIV*:
 $\text{finite } (\text{UNIV}::'a \ \text{set}) \implies \text{finite } (\text{UNIV}::'b \ \text{set}) \implies \text{finite } (\text{UNIV}::('a * 'b) \ \text{set})$
 $\langle \text{proof} \rangle$

lemma *finite-cartesian-productD1*:
 $[\text{finite } (A <*> B); B \neq \{\}] \implies \text{finite } A$
 $\langle \text{proof} \rangle$

lemma *finite-cartesian-productD2*:
 $[\text{finite } (A <*> B); A \neq \{\}] \implies \text{finite } B$
 $\langle \text{proof} \rangle$

The powerset of a finite set

lemma *finite-Pow-iff* [iff]: $\text{finite } (\text{Pow } A) = \text{finite } A$
 $\langle \text{proof} \rangle$

lemma *finite-Collect-subsets*[simp,intro]: $\text{finite } A \implies \text{finite}\{B. B \subseteq A\}$
 $\langle \text{proof} \rangle$

lemma *finite-UnionD*: $\text{finite}(\bigcup A) \implies \text{finite } A$
 $\langle \text{proof} \rangle$

17.2 Class *finite*

```

<ML>
class finite =
  assumes finite-UNIV: finite (UNIV :: 'a set)
<ML>
hide const finite

```

```

context finite
begin

```

```

lemma finite [simp]: finite (A :: 'a set)
  <proof>

```

```

end

```

```

lemma UNIV-unit [noatp]:
  UNIV = {()} <proof>

```

```

instance unit :: finite
  <proof>

```

```

lemma UNIV-bool [noatp]:
  UNIV = {False, True} <proof>

```

```

instance bool :: finite
  <proof>

```

```

instance * :: (finite, finite) finite
  <proof>

```

```

lemma inj-graph: inj (%f. {(x, y). y = f x})
  <proof>

```

```

instance fun :: (finite, finite) finite
  <proof>

```

```

instance + :: (finite, finite) finite
  <proof>

```

17.3 A fold functional for finite sets

The intended behaviour is $\text{fold } f \ z \ \{x_1, \dots, x_n\} = f \ x_1 \ (\dots (f \ x_n \ z) \dots)$ if f is “left-commutative”:

```

locale fun-left-comm =
  fixes f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'b
  assumes fun-left-comm:  $f \ x \ (f \ y \ z) = f \ y \ (f \ x \ z)$ 
begin

```

On a functional level it looks much nicer:

lemma *fun-comp-comm*: $f x \circ f y = f y \circ f x$
 $\langle proof \rangle$

end

inductive *fold-graph* :: ($'a \Rightarrow 'b \Rightarrow 'b$) $\Rightarrow 'b \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow \text{bool}$
for $f :: 'a \Rightarrow 'b \Rightarrow 'b$ **and** $z :: 'b$ **where**
 emptyI [intro]: *fold-graph* $f z \{\}$ $z \mid$
 insertI [intro]: $x \notin A \Longrightarrow \text{fold-graph } f z A y$
 $\Longrightarrow \text{fold-graph } f z (\text{insert } x A) (f x y)$

inductive-cases *empty-fold-graphE* [elim!]: *fold-graph* $f z \{\}$ x

definition *fold* :: ($'a \Rightarrow 'b \Rightarrow 'b$) $\Rightarrow 'b \Rightarrow 'a \text{ set} \Rightarrow 'b$ **where**
 $[code\ del]: \text{fold } f z A = (\text{THE } y. \text{fold-graph } f z A y)$

A tempting alternative for the definiens is *if finite A then THE y. fold-graph f z A y else e*. It allows the removal of finiteness assumptions from the theorems *fold-comm*, *fold-reindex* and *fold-distrib*. The proofs become ugly. It is not worth the effort. (???)

lemma *Diff1-fold-graph*:
 $\text{fold-graph } f z (A - \{x\}) y \Longrightarrow x \in A \Longrightarrow \text{fold-graph } f z A (f x y)$
 $\langle proof \rangle$

lemma *fold-graph-imp-finite*: $\text{fold-graph } f z A x \Longrightarrow \text{finite } A$
 $\langle proof \rangle$

lemma *finite-imp-fold-graph*: $\text{finite } A \Longrightarrow \exists x. \text{fold-graph } f z A x$
 $\langle proof \rangle$

17.3.1 From *fold-graph* to *fold*

lemma *image-less-Suc*: $h \text{ ‘ } \{i. i < \text{Suc } m\} = \text{insert } (h m) (h \text{ ‘ } \{i. i < m\})$
 $\langle proof \rangle$

lemma *insert-image-inj-on-eq*:
 $[[\text{insert } (h m) A = h \text{ ‘ } \{i. i < \text{Suc } m\}; h m \notin A;$
 $\text{inj-on } h \{i. i < \text{Suc } m\}]]$
 $\Longrightarrow A = h \text{ ‘ } \{i. i < m\}$
 $\langle proof \rangle$

lemma *insert-inj-onE*:
assumes $aA: \text{insert } a A = h \text{ ‘ } \{i::\text{nat}. i < n\}$ **and** $\text{anot}: a \notin A$
and $\text{inj-on}: \text{inj-on } h \{i::\text{nat}. i < n\}$
shows $\exists hm m. \text{inj-on } hm \{i::\text{nat}. i < m\} \ \& \ A = hm \text{ ‘ } \{i. i < m\} \ \& \ m < n$
 $\langle proof \rangle$

context *fun-left-comm*

begin

lemma *fold-graph-determ-aux*:

$$\begin{aligned} A = h' \{i :: \text{nat}. i < n\} &\implies \text{inj-on } h \{i. i < n\} \\ &\implies \text{fold-graph } f \ z \ A \ x \implies \text{fold-graph } f \ z \ A \ x' \\ &\implies x' = x \end{aligned}$$

<proof>

lemma *fold-graph-determ*:

$$\text{fold-graph } f \ z \ A \ x \implies \text{fold-graph } f \ z \ A \ y \implies y = x$$

<proof>

lemma *fold-equality*:

$$\text{fold-graph } f \ z \ A \ y \implies \text{fold } f \ z \ A = y$$

<proof>

The base case for *fold*:

lemma (**in** $-$) *fold-empty* [*simp*]: $\text{fold } f \ z \ \{\} = z$

<proof>

The various recursion equations for *fold*:

lemma *fold-insert-aux*: $x \notin A$

$$\begin{aligned} &\implies \text{fold-graph } f \ z \ (\text{insert } x \ A) \ v \longleftrightarrow \\ &\quad (\exists y. \text{fold-graph } f \ z \ A \ y \wedge v = f \ x \ y) \end{aligned}$$

<proof>

lemma *fold-insert* [*simp*]:

$$\text{finite } A \implies x \notin A \implies \text{fold } f \ z \ (\text{insert } x \ A) = f \ x \ (\text{fold } f \ z \ A)$$

<proof>

lemma *fold-fun-comm*:

$$\text{finite } A \implies f \ x \ (\text{fold } f \ z \ A) = \text{fold } f \ (f \ x \ z) \ A$$

<proof>

lemma *fold-insert2*:

$$\text{finite } A \implies x \notin A \implies \text{fold } f \ z \ (\text{insert } x \ A) = \text{fold } f \ (f \ x \ z) \ A$$

<proof>

lemma *fold-rec*:

assumes *finite* A **and** $x \in A$

shows $\text{fold } f \ z \ A = f \ x \ (\text{fold } f \ z \ (A - \{x\}))$

<proof>

lemma *fold-insert-remove*:

assumes *finite* A

shows $\text{fold } f \ z \ (\text{insert } x \ A) = f \ x \ (\text{fold } f \ z \ (A - \{x\}))$

<proof>

end

A simplified version for idempotent functions:

```
locale fun-left-comm-idem = fun-left-comm +
  assumes fun-left-idem:  $f\ x\ (f\ x\ z) = f\ x\ z$ 
begin
```

The nice version:

```
lemma fun-comp-idem :  $f\ x\ o\ f\ x = f\ x$ 
 $\langle$ proof $\rangle$ 
```

```
lemma fold-insert-idem:
  assumes fin: finite A
  shows  $fold\ f\ z\ (insert\ x\ A) = f\ x\ (fold\ f\ z\ A)$ 
 $\langle$ proof $\rangle$ 
```

```
declare fold-insert[simp del] fold-insert-idem[simp]
```

```
lemma fold-insert-idem2:
   $finite\ A \implies fold\ f\ z\ (insert\ x\ A) = fold\ f\ (f\ x\ z)\ A$ 
 $\langle$ proof $\rangle$ 
```

```
end
```

17.3.2 The derived combinator *fold-image*

```
definition fold-image ::  $('b \Rightarrow 'b \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ set \Rightarrow 'b$ 
where  $fold-image\ f\ g = fold\ (\%x\ y. f\ (g\ x)\ y)$ 
```

```
lemma fold-image-empty[simp]:  $fold-image\ f\ g\ z\ \{\} = z$ 
 $\langle$ proof $\rangle$ 
```

```
context ab-semigroup-mult
begin
```

```
lemma fold-image-insert[simp]:
assumes finite A and  $a \notin A$ 
shows  $fold-image\ times\ g\ z\ (insert\ a\ A) = g\ a * (fold-image\ times\ g\ z\ A)$ 
 $\langle$ proof $\rangle$ 
```

```
lemma fold-image-reindex:
assumes fin: finite A
shows  $inj-on\ h\ A \implies fold-image\ times\ g\ z\ (h\ 'A) = fold-image\ times\ (g \circ h)\ z\ A$ 
 $\langle$ proof $\rangle$ 
```

```
lemma fold-image-cong:
   $finite\ A \implies$ 
```

($\text{!}x. x:A \implies g\ x = h\ x$) \implies *fold-image times* $g\ z\ A = \text{fold-image times}\ h\ z\ A$
 <proof>

end

context *comm-monoid-mult*
begin

lemma *fold-image-Un-Int*:
 $\text{finite}\ A \implies \text{finite}\ B \implies$
 $\text{fold-image times}\ g\ 1\ A * \text{fold-image times}\ g\ 1\ B =$
 $\text{fold-image times}\ g\ 1\ (A\ \text{Un}\ B) * \text{fold-image times}\ g\ 1\ (A\ \text{Int}\ B)$
 <proof>

corollary *fold-Un-disjoint*:
 $\text{finite}\ A \implies \text{finite}\ B \implies A\ \text{Int}\ B = \{\} \implies$
 $\text{fold-image times}\ g\ 1\ (A\ \text{Un}\ B) =$
 $\text{fold-image times}\ g\ 1\ A * \text{fold-image times}\ g\ 1\ B$
 <proof>

lemma *fold-image-UN-disjoint*:
 $\llbracket \text{finite}\ I; \text{ALL}\ i:I. \text{finite}\ (A\ i);$
 $\text{ALL}\ i:I. \text{ALL}\ j:I. i \neq j \dashrightarrow A\ i\ \text{Int}\ A\ j = \{\} \rrbracket$
 $\implies \text{fold-image times}\ g\ 1\ (\text{UNION}\ I\ A) =$
 $\text{fold-image times}\ (\%i. \text{fold-image times}\ g\ 1\ (A\ i))\ 1\ I$
 <proof>

lemma *fold-image-Sigma*: $\text{finite}\ A \implies \text{ALL}\ x:A. \text{finite}\ (B\ x) \implies$
 $\text{fold-image times}\ (\%x. \text{fold-image times}\ (g\ x)\ 1\ (B\ x))\ 1\ A =$
 $\text{fold-image times}\ (\text{split}\ g)\ 1\ (\text{SIGMA}\ x:A. B\ x)$
 <proof>

lemma *fold-image-distrib*: $\text{finite}\ A \implies$
 $\text{fold-image times}\ (\%x. g\ x * h\ x)\ 1\ A =$
 $\text{fold-image times}\ g\ 1\ A * \text{fold-image times}\ h\ 1\ A$
 <proof>

lemma *fold-image-related*:
assumes $Re: R\ e\ e$
and $Rop: \forall x1\ y1\ x2\ y2. R\ x1\ x2 \wedge R\ y1\ y2 \longrightarrow R\ (x1 * y1)\ (x2 * y2)$
and $fS: \text{finite}\ S$ **and** $Rfg: \forall x \in S. R\ (h\ x)\ (g\ x)$
shows $R\ (\text{fold-image}\ (op\ *)\ h\ e\ S)\ (\text{fold-image}\ (op\ *)\ g\ e\ S)$
 <proof>

lemma *fold-image-eq-general*:
assumes $fS: \text{finite}\ S$
and $h: \forall y \in S'. \exists!x. x \in S \wedge h(x) = y$
and $f12: \forall x \in S. h\ x \in S' \wedge f2(h\ x) = f1\ x$
shows $\text{fold-image}\ (op\ *)\ f1\ e\ S = \text{fold-image}\ (op\ *)\ f2\ e\ S'$

<proof>

lemma *fold-image-eq-general-inverses*:

assumes *fS*: *finite S*

and *kh*: $\bigwedge y. y \in T \implies k\ y \in S \wedge h\ (k\ y) = y$

and *hk*: $\bigwedge x. x \in S \implies h\ x \in T \wedge k\ (h\ x) = x \wedge g\ (h\ x) = f\ x$

shows *fold-image (op *) f e S = fold-image (op *) g e T*

<proof>

end

17.4 Generalized summation over a set

interpretation *comm-monoid-add*: *comm-monoid-mult 0::'a::comm-monoid-add*

op +

<proof>

definition *setsum* :: (*'a* => *'b*) => *'a set* => *'b::comm-monoid-add*

where *setsum f A* == *if finite A then fold-image (op +) f 0 A else 0*

abbreviation

Setsum (\sum - [1000] 999) **where**

$\sum A$ == *setsum* (%*x*. *x*) *A*

Now: lot's of fancy syntax. First, *setsum* ($\lambda x. e$) *A* is written $\sum_{x \in A}. e$.

syntax

-setsum :: *pttrn* => *'a set* => *'b* => *'b::comm-monoid-add* ((*3SUM* -:-. -) [0, 51, 10] 10)

syntax (*xsymbols*)

-setsum :: *pttrn* => *'a set* => *'b* => *'b::comm-monoid-add* ((*3SUM* -∈-. -) [0, 51, 10] 10)

syntax (*HTML output*)

-setsum :: *pttrn* => *'a set* => *'b* => *'b::comm-monoid-add* ((*3SUM* -∈-. -) [0, 51, 10] 10)

translations — Beware of argument permutation!

SUM i:A. b == *CONST setsum* (%*i*. *b*) *A*

$\sum_{i \in A}. b$ == *CONST setsum* (%*i*. *b*) *A*

Instead of $\sum_{x \in \{x. P\}}. e$ we introduce the shorter $\sum x|P. e$.

syntax

-qsetsum :: *pttrn* => *bool* => *'a* => *'a* ((*3SUM* - | / - / -) [0,0,10] 10)

syntax (*xsymbols*)

-qsetsum :: *pttrn* => *bool* => *'a* => *'a* ((*3SUM* - | (-). / -) [0,0,10] 10)

syntax (*HTML output*)

-qsetsum :: *pttrn* => *bool* => *'a* => *'a* ((*3SUM* - | (-). / -) [0,0,10] 10)

translations

$$\begin{aligned} & \text{SUM } x | P. t \Rightarrow \text{CONST setsum } (\%x. t) \{x. P\} \\ & \text{SUM } x | P. t \Rightarrow \text{CONST setsum } (\%x. t) \{x. P\} \end{aligned}$$

$\langle \text{ML} \rangle$

lemma *setsum-empty* [simp]: $\text{setsum } f \ \{\} = 0$
 $\langle \text{proof} \rangle$

lemma *setsum-insert* [simp]:
 $\text{finite } F \Rightarrow a \notin F \Rightarrow \text{setsum } f \ (\text{insert } a \ F) = f \ a + \text{setsum } f \ F$
 $\langle \text{proof} \rangle$

lemma *setsum-infinite* [simp]: $\sim \text{finite } A \Rightarrow \text{setsum } f \ A = 0$
 $\langle \text{proof} \rangle$

lemma *setsum-reindex*:
 $\text{inj-on } f \ B \Rightarrow \text{setsum } h \ (f \ ' \ B) = \text{setsum } (h \circ f) \ B$
 $\langle \text{proof} \rangle$

lemma *setsum-reindex-id*:
 $\text{inj-on } f \ B \Rightarrow \text{setsum } f \ B = \text{setsum } \text{id} \ (f \ ' \ B)$
 $\langle \text{proof} \rangle$

lemma *setsum-reindex-nonzero*:
assumes fS : $\text{finite } S$
and nz : $\bigwedge x \ y. x \in S \Rightarrow y \in S \Rightarrow x \neq y \Rightarrow f \ x = f \ y \Rightarrow h \ (f \ x) = 0$
shows $\text{setsum } h \ (f \ ' \ S) = \text{setsum } (h \circ f) \ S$
 $\langle \text{proof} \rangle$

lemma *setsum-cong*:
 $A = B \Rightarrow (!x. x:B \Rightarrow f \ x = g \ x) \Rightarrow \text{setsum } f \ A = \text{setsum } g \ B$
 $\langle \text{proof} \rangle$

lemma *strong-setsum-cong*[cong]:
 $A = B \Rightarrow (!x. x:B \Rightarrow \text{simp} \Rightarrow f \ x = g \ x)$
 $\Rightarrow \text{setsum } (\%x. f \ x) \ A = \text{setsum } (\%x. g \ x) \ B$
 $\langle \text{proof} \rangle$

lemma *setsum-cong2*: $[\bigwedge x. x \in A \Rightarrow f \ x = g \ x] \Rightarrow \text{setsum } f \ A = \text{setsum } g \ A$
 $\langle \text{proof} \rangle$

lemma *setsum-reindex-cong*:
 $[\text{inj-on } f \ A; B = f \ ' \ A; !a. a:A \Rightarrow g \ a = h \ (f \ a)]$
 $\Rightarrow \text{setsum } h \ B = \text{setsum } g \ A$
 $\langle \text{proof} \rangle$

lemma *setsum-0*[simp]: $\text{setsum } (\%i. 0) \ A = 0$

<proof>

lemma *setsum-0'*: $ALL\ a:A.\ f\ a = 0 \implies setsum\ f\ A = 0$

<proof>

lemma *setsum-Un-Int*: $finite\ A \implies finite\ B \implies$

$setsum\ g\ (A\ Un\ B) + setsum\ g\ (A\ Int\ B) = setsum\ g\ A + setsum\ g\ B$

— The reversed orientation looks more natural, but LOOPS as a simprule!

<proof>

lemma *setsum-Un-disjoint*: $finite\ A \implies finite\ B$

$\implies A\ Int\ B = \{\} \implies setsum\ g\ (A\ Un\ B) = setsum\ g\ A + setsum\ g\ B$

<proof>

lemma *setsum-mono-zero-left*:

assumes fT : $finite\ T$ **and** ST : $S \subseteq T$

and z : $\forall i \in T - S.\ f\ i = 0$

shows $setsum\ f\ S = setsum\ f\ T$

<proof>

lemma *setsum-mono-zero-right*:

$finite\ T \implies S \subseteq T \implies \forall i \in T - S.\ f\ i = 0 \implies setsum\ f\ T = setsum\ f\ S$

<proof>

lemma *setsum-mono-zero-cong-left*:

assumes fT : $finite\ T$ **and** ST : $S \subseteq T$

and z : $\forall i \in T - S.\ g\ i = 0$

and fg : $\bigwedge x.\ x \in S \implies f\ x = g\ x$

shows $setsum\ f\ S = setsum\ g\ T$

<proof>

lemma *setsum-mono-zero-cong-right*:

assumes fT : $finite\ T$ **and** ST : $S \subseteq T$

and z : $\forall i \in T - S.\ f\ i = 0$

and fg : $\bigwedge x.\ x \in S \implies f\ x = g\ x$

shows $setsum\ f\ T = setsum\ g\ S$

<proof>

lemma *setsum-delta*:

assumes fS : $finite\ S$

shows $setsum\ (\lambda k.\ if\ k=a\ then\ b\ k\ else\ 0)\ S = (if\ a \in S\ then\ b\ a\ else\ 0)$

<proof>

lemma *setsum-delta'*:

assumes fS : $finite\ S$ **shows**

$setsum\ (\lambda k.\ if\ a = k\ then\ b\ k\ else\ 0)\ S =$
 $(if\ a \in S\ then\ b\ a\ else\ 0)$

<proof>

lemma *setsum-restrict-set*:

assumes fA : *finite* A
shows $\text{setsum } f \ (A \cap B) = \text{setsum } (\lambda x. \text{if } x \in B \text{ then } f \ x \text{ else } 0) \ A$
 $\langle \text{proof} \rangle$

lemma *setsum-cases*:
assumes fA : *finite* A
shows $\text{setsum } (\lambda x. \text{if } x \in B \text{ then } f \ x \text{ else } g \ x) \ A =$
 $\text{setsum } f \ (A \cap B) + \text{setsum } g \ (A \cap - \ B)$
 $\langle \text{proof} \rangle$

lemma *setsum-UN-disjoint*:
 $\text{finite } I \implies (\text{ALL } i:I. \text{finite } (A \ i)) \implies$
 $(\text{ALL } i:I. \text{ALL } j:I. i \neq j \longrightarrow A \ i \ \text{Int } A \ j = \{\}) \implies$
 $\text{setsum } f \ (\text{UNION } I \ A) = (\sum i \in I. \text{setsum } f \ (A \ i))$
 $\langle \text{proof} \rangle$

No need to assume that C is finite. If infinite, the rhs is directly 0, and $\bigcup C$ is also infinite, hence the lhs is also 0.

lemma *setsum-Union-disjoint*:
 $[\text{ALL } A:C. \text{finite } A];$
 $(\text{ALL } A:C. \text{ALL } B:C. A \neq B \longrightarrow A \ \text{Int } B = \{\}) \ [\]$
 $\implies \text{setsum } f \ (\text{Union } C) = \text{setsum } (\text{setsum } f) \ C$
 $\langle \text{proof} \rangle$

lemma *setsum-Sigma*: $\text{finite } A \implies \text{ALL } x:A. \text{finite } (B \ x) \implies$
 $(\sum x \in A. (\sum y \in B \ x. f \ x \ y)) = (\sum (x,y) \in (\text{SIGMA } x:A. B \ x). f \ x \ y)$
 $\langle \text{proof} \rangle$

Here we can eliminate the finiteness assumptions, by cases.

lemma *setsum-cartesian-product*:
 $(\sum x \in A. (\sum y \in B. f \ x \ y)) = (\sum (x,y) \in A \ <*> B. f \ x \ y)$
 $\langle \text{proof} \rangle$

lemma *setsum-addf*: $\text{setsum } (\%x. f \ x + g \ x) \ A = (\text{setsum } f \ A + \text{setsum } g \ A)$
 $\langle \text{proof} \rangle$

17.4.1 Properties in more restricted classes of structures

lemma *setsum-SucD*: $\text{setsum } f \ A = \text{Suc } n \implies \text{EX } a:A. 0 < f \ a$
 $\langle \text{proof} \rangle$

lemma *setsum-eq-0-iff [simp]*:
 $\text{finite } F \implies (\text{setsum } f \ F = 0) = (\text{ALL } a:F. f \ a = (0::\text{nat}))$
 $\langle \text{proof} \rangle$

lemma *setsum-eq-Suc0-iff*: $\text{finite } A \implies$

$(\text{setsum } f \ A = \text{Suc } 0) = (\text{EX } a:A. f \ a = \text{Suc } 0 \ \& \ (\text{ALL } b:A. a \neq b \longrightarrow f \ b = 0))$
 $\langle \text{proof} \rangle$

lemmas *setsum-eq-1-iff* = *setsum-eq-Suc0-iff*[*simplified One-nat-def*[*symmetric*]]

lemma *setsum-Un-nat*: *finite A ==> finite B ==>*
 $(\text{setsum } f \ (A \ \text{Un } B) :: \text{nat}) = \text{setsum } f \ A + \text{setsum } f \ B - \text{setsum } f \ (A \ \text{Int } B)$
 — For the natural numbers, we have subtraction.
 $\langle \text{proof} \rangle$

lemma *setsum-Un*: *finite A ==> finite B ==>*
 $(\text{setsum } f \ (A \ \text{Un } B) :: 'a :: \text{ab-group-add}) =$
 $\text{setsum } f \ A + \text{setsum } f \ B - \text{setsum } f \ (A \ \text{Int } B)$
 $\langle \text{proof} \rangle$

lemma (*in comm-monoid-mult*) *fold-image-1*: *finite S ==> ($\forall x \in S. f \ x = 1$) ==>*
 $\text{fold-image } \text{op} \ * \ f \ 1 \ S = 1$
 $\langle \text{proof} \rangle$

lemma (*in comm-monoid-mult*) *fold-image-Un-one*:
assumes *fS: finite S and fT: finite T*
and *I0: $\forall x \in S \cap T. f \ x = 1$*
shows $\text{fold-image } (\text{op} \ *) \ f \ 1 \ (S \cup T) = \text{fold-image } (\text{op} \ *) \ f \ 1 \ S * \text{fold-image } (\text{op} \ *) \ f \ 1 \ T$
 $\langle \text{proof} \rangle$

lemma *setsum-eq-general-reverses*:
assumes *fS: finite S and fT: finite T*
and *kh: $\bigwedge y. y \in T \Longrightarrow k \ y \in S \wedge h \ (k \ y) = y$*
and *hk: $\bigwedge x. x \in S \Longrightarrow h \ x \in T \wedge k \ (h \ x) = x \wedge g \ (h \ x) = f \ x$*
shows $\text{setsum } f \ S = \text{setsum } g \ T$
 $\langle \text{proof} \rangle$

lemma *setsum-Un-zero*:
assumes *fS: finite S and fT: finite T*
and *I0: $\forall x \in S \cap T. f \ x = 0$*
shows $\text{setsum } f \ (S \cup T) = \text{setsum } f \ S + \text{setsum } f \ T$
 $\langle \text{proof} \rangle$

lemma *setsum-UNION-zero*:
assumes *fS: finite S and fSS: $\forall T \in S. \text{finite } T$*
and *f0: $\bigwedge T1 \ T2 \ x. T1 \in S \Longrightarrow T2 \in S \Longrightarrow T1 \neq T2 \Longrightarrow x \in T1 \Longrightarrow x \in T2 \Longrightarrow f \ x = 0$*
shows $\text{setsum } f \ (\bigcup S) = \text{setsum } (\lambda T. \text{setsum } f \ T) \ S$
 $\langle \text{proof} \rangle$

lemma *setsum-diff1-nat*: $(\text{setsum } f (A - \{a\}) :: \text{nat}) =$
 $(\text{if } a:A \text{ then } \text{setsum } f A - f a \text{ else } \text{setsum } f A)$
 $\langle \text{proof} \rangle$

lemma *setsum-diff1*: $\text{finite } A \implies$
 $(\text{setsum } f (A - \{a\}) :: ('a::\text{ab-group-add})) =$
 $(\text{if } a:A \text{ then } \text{setsum } f A - f a \text{ else } \text{setsum } f A)$
 $\langle \text{proof} \rangle$

lemma *setsum-diff1 '[rule-format]*:
 $\text{finite } A \implies a \in A \longrightarrow (\sum x \in A. f x) = f a + (\sum x \in (A - \{a\}). f x)$
 $\langle \text{proof} \rangle$

lemma *setsum-diff-nat*:
assumes *finite B and B* $\subseteq A$
shows $(\text{setsum } f (A - B) :: \text{nat}) = (\text{setsum } f A) - (\text{setsum } f B)$
 $\langle \text{proof} \rangle$

lemma *setsum-diff*:
assumes *le: finite A B* $\subseteq A$
shows $\text{setsum } f (A - B) = \text{setsum } f A - ((\text{setsum } f B)::('a::\text{ab-group-add}))$
 $\langle \text{proof} \rangle$

lemma *setsum-mono*:
assumes *le: $\bigwedge i. i \in K \implies f (i::'a) \leq ((g i)::('b::\{\text{comm-monoid-add}, \text{pordered-ab-semigroup-add}\}))$*
shows $(\sum i \in K. f i) \leq (\sum i \in K. g i)$
 $\langle \text{proof} \rangle$

lemma *setsum-strict-mono*:
fixes *f :: 'a \Rightarrow 'b::{\text{pordered-cancel-ab-semigroup-add}, comm-monoid-add}*
assumes *finite A A* $\neq \{\}$
and $\forall x. x:A \implies f x < g x$
shows $\text{setsum } f A < \text{setsum } g A$
 $\langle \text{proof} \rangle$

lemma *setsum-negf*:
 $\text{setsum } (\%x. - (f x)::'a::\text{ab-group-add}) A = - \text{setsum } f A$
 $\langle \text{proof} \rangle$

lemma *setsum-subtractf*:
 $\text{setsum } (\%x. ((f x)::'a::\text{ab-group-add}) - g x) A =$
 $\text{setsum } f A - \text{setsum } g A$
 $\langle \text{proof} \rangle$

lemma *setsum-nonneg*:
assumes *nn: $\forall x \in A. (0::'a::\{\text{pordered-ab-semigroup-add}, \text{comm-monoid-add}\}) \leq$*

$f\ x$
shows $0 \leq \text{setsum } f\ A$
 $\langle \text{proof} \rangle$

lemma *setsum-nonpos*:
assumes $np: \forall x \in A. f\ x \leq (0 :: 'a :: \{\text{pordered-ab-semigroup-add}, \text{comm-monoid-add}\})$
shows $\text{setsum } f\ A \leq 0$
 $\langle \text{proof} \rangle$

lemma *setsum-mono2*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{pordered-ab-semigroup-add-imp-le}, \text{comm-monoid-add}\}$
assumes $\text{fin}: \text{finite } B$ **and** $\text{sub}: A \subseteq B$ **and** $\text{nn}: \bigwedge b. b \in B - A \implies 0 \leq f\ b$
shows $\text{setsum } f\ A \leq \text{setsum } f\ B$
 $\langle \text{proof} \rangle$

lemma *setsum-mono3*: $\text{finite } B \implies A \leq B \implies$
 $\text{ALL } x: B - A.$
 $0 \leq ((f\ x) :: 'a :: \{\text{comm-monoid-add}, \text{pordered-ab-semigroup-add}\}) \implies$
 $\text{setsum } f\ A \leq \text{setsum } f\ B$
 $\langle \text{proof} \rangle$

lemma *setsum-right-distrib*:
fixes $f :: 'a \Rightarrow ('b :: \text{semiring-0})$
shows $r * \text{setsum } f\ A = \text{setsum } (\%n. r * f\ n)\ A$
 $\langle \text{proof} \rangle$

lemma *setsum-left-distrib*:
 $\text{setsum } f\ A * (r :: 'a :: \text{semiring-0}) = (\sum n \in A. f\ n * r)$
 $\langle \text{proof} \rangle$

lemma *setsum-divide-distrib*:
 $\text{setsum } f\ A / (r :: 'a :: \text{field}) = (\sum n \in A. f\ n / r)$
 $\langle \text{proof} \rangle$

lemma *setsum-abs[iff]*:
fixes $f :: 'a \Rightarrow ('b :: \text{pordered-ab-group-add-abs})$
shows $\text{abs } (\text{setsum } f\ A) \leq \text{setsum } (\%i. \text{abs}(f\ i))\ A$
 $\langle \text{proof} \rangle$

lemma *setsum-abs-ge-zero[iff]*:
fixes $f :: 'a \Rightarrow ('b :: \text{pordered-ab-group-add-abs})$
shows $0 \leq \text{setsum } (\%i. \text{abs}(f\ i))\ A$
 $\langle \text{proof} \rangle$

lemma *abs-setsum-abs[simp]*:
fixes $f :: 'a \Rightarrow ('b :: \text{pordered-ab-group-add-abs})$
shows $\text{abs } (\sum a \in A. \text{abs}(f\ a)) = (\sum a \in A. \text{abs}(f\ a))$
 $\langle \text{proof} \rangle$

Commuting outer and inner summation

lemma *swap-inj-on*:

inj-on ($\% (i, j). (j, i)$) ($A \times B$)
 $\langle \text{proof} \rangle$

lemma *swap-product*:

$(\% (i, j). (j, i)) \text{ } ^{\text{‘}} (A \times B) = B \times A$
 $\langle \text{proof} \rangle$

lemma *setsum-commute*:

$(\sum_{i \in A}. \sum_{j \in B}. f \ i \ j) = (\sum_{j \in B}. \sum_{i \in A}. f \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *setsum-product*:

fixes $f :: 'a \Rightarrow ('b::\text{semiring-0})$
shows $\text{setsum } f \ A * \text{setsum } g \ B = (\sum_{i \in A}. \sum_{j \in B}. f \ i * g \ j)$
 $\langle \text{proof} \rangle$

17.5 Generalized product over a set

definition *setprod* :: $('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b::\text{comm-monoid-mult}$

where $\text{setprod } f \ A == \text{if finite } A \text{ then fold-image } (op \ *) \ f \ 1 \ A \text{ else } 1$

abbreviation

$\text{Setprod } (\prod - [1000] \ 999) \text{ where}$
 $\prod A == \text{setprod } (\%x. x) \ A$

syntax

$\text{-setprod} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b::\text{comm-monoid-mult} \ ((\exists \text{PROD} \text{ } \text{--} \text{ } \text{--}) \ [0, 51, 10] \ 10)$

syntax (*xsymbols*)

$\text{-setprod} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b::\text{comm-monoid-mult} \ ((\exists \prod \text{ } \text{--} \text{ } \text{--}) \ [0, 51, 10] \ 10)$

syntax (*HTML output*)

$\text{-setprod} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b::\text{comm-monoid-mult} \ ((\exists \prod \text{ } \text{--} \text{ } \text{--}) \ [0, 51, 10] \ 10)$

translations — Beware of argument permutation!

$\text{PROD } i:A. b == \text{CONST } \text{setprod } (\%i. b) \ A$
 $\prod i \in A. b == \text{CONST } \text{setprod } (\%i. b) \ A$

Instead of $\prod x \in \{x. P\}. e$ we introduce the shorter $\prod x | P. e$.

syntax

$\text{-qsetprod} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((\exists \text{PROD} \text{ } \text{--} \text{ } \text{--}) \ [0, 0, 10] \ 10)$

syntax (*xsymbols*)

$\text{-qsetprod} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((\exists \prod \text{ } \text{--} \text{ } \text{--}) \ [0, 0, 10] \ 10)$

syntax (*HTML output*)

$\text{-qsetprod} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((\exists \prod \text{ } \text{--} \text{ } \text{--}) \ [0, 0, 10] \ 10)$

translations

$$\begin{aligned} & PROD\ x|P. t ==> CONST\ setprod\ (\%x. t)\ \{x. P\} \\ & \prod x|P. t ==> CONST\ setprod\ (\%x. t)\ \{x. P\} \end{aligned}$$

lemma *setprod-empty* [simp]: $setprod\ f\ \{\} = 1$
 <proof>

lemma *setprod-insert* [simp]: $[| finite\ A; a \notin A |] ==>$
 $setprod\ f\ (insert\ a\ A) = f\ a * setprod\ f\ A$
 <proof>

lemma *setprod-infinite* [simp]: $\sim finite\ A ==> setprod\ f\ A = 1$
 <proof>

lemma *setprod-reindex*:
 $inj-on\ f\ B ==> setprod\ h\ (f\ ' B) = setprod\ (h \circ f)\ B$
 <proof>

lemma *setprod-reindex-id*: $inj-on\ f\ B ==> setprod\ f\ B = setprod\ id\ (f\ ' B)$
 <proof>

lemma *setprod-cong*:
 $A = B ==> (!x. x:B ==> f\ x = g\ x) ==> setprod\ f\ A = setprod\ g\ B$
 <proof>

lemma *strong-setprod-cong*[cong]:
 $A = B ==> (!x. x:B ==> f\ x = g\ x) ==> setprod\ f\ A = setprod\ g\ B$
 <proof>

lemma *setprod-reindex-cong*: $inj-on\ f\ A ==>$
 $B = f\ ' A ==> g = h \circ f ==> setprod\ h\ B = setprod\ g\ A$
 <proof>

lemma *strong-setprod-reindex-cong*: **assumes** $i: inj-on\ f\ A$
and $B: B = f\ ' A$ **and** $eq: \bigwedge x. x \in A \implies g\ x = (h \circ f)\ x$
shows $setprod\ h\ B = setprod\ g\ A$
 <proof>

lemma *setprod-Un-one*:
assumes $fS: finite\ S$ **and** $fT: finite\ T$
and $I0: \forall x \in S \cap T. f\ x = 1$
shows $setprod\ f\ (S \cup T) = setprod\ f\ S * setprod\ f\ T$
 <proof>

lemma *setprod-1*: $setprod\ (\%i. 1)\ A = 1$
 <proof>

lemma *setprod-1'*: $ALL\ a:F. f\ a = 1 ==> setprod\ f\ F = 1$

$\langle \text{proof} \rangle$

lemma *setprod-Un-Int*: $\text{finite } A \implies \text{finite } B$
 $\implies \text{setprod } g (A \text{ Un } B) * \text{setprod } g (A \text{ Int } B) = \text{setprod } g A * \text{setprod } g B$
 $\langle \text{proof} \rangle$

lemma *setprod-Un-disjoint*: $\text{finite } A \implies \text{finite } B$
 $\implies A \text{ Int } B = \{\} \implies \text{setprod } g (A \text{ Un } B) = \text{setprod } g A * \text{setprod } g B$
 $\langle \text{proof} \rangle$

lemma *setprod-mono-one-left*:
assumes fT : $\text{finite } T$ **and** ST : $S \subseteq T$
and z : $\forall i \in T - S. f i = 1$
shows $\text{setprod } f S = \text{setprod } f T$
 $\langle \text{proof} \rangle$

lemmas *setprod-mono-one-right* = *setprod-mono-one-left* [THEN sym]

lemma *setprod-delta*:
assumes fS : $\text{finite } S$
shows $\text{setprod } (\lambda k. \text{if } k=a \text{ then } b \text{ } k \text{ else } 1) S = (\text{if } a \in S \text{ then } b \text{ } a \text{ else } 1)$
 $\langle \text{proof} \rangle$

lemma *setprod-delta'*:
assumes fS : $\text{finite } S$ **shows**
 $\text{setprod } (\lambda k. \text{if } a = k \text{ then } b \text{ } k \text{ else } 1) S =$
 $(\text{if } a \in S \text{ then } b \text{ } a \text{ else } 1)$
 $\langle \text{proof} \rangle$

lemma *setprod-UN-disjoint*:
 $\text{finite } I \implies (\text{ALL } i:I. \text{finite } (A \text{ } i)) \implies$
 $(\text{ALL } i:I. \text{ALL } j:I. i \neq j \longrightarrow A \text{ } i \text{ Int } A \text{ } j = \{\}) \implies$
 $\text{setprod } f (\text{UNION } I A) = \text{setprod } (\%i. \text{setprod } f (A \text{ } i)) I$
 $\langle \text{proof} \rangle$

lemma *setprod-Union-disjoint*:
 $\llbracket (\text{ALL } A:C. \text{finite } A);$
 $(\text{ALL } A:C. \text{ALL } B:C. A \neq B \longrightarrow A \text{ Int } B = \{\}) \rrbracket$
 $\implies \text{setprod } f (\text{Union } C) = \text{setprod } (\text{setprod } f) C$
 $\langle \text{proof} \rangle$

lemma *setprod-Sigma*: $\text{finite } A \implies \text{ALL } x:A. \text{finite } (B \text{ } x) \implies$
 $(\prod_{x \in A}. (\prod_{y \in B \text{ } x}. f \text{ } x \text{ } y)) =$
 $(\prod_{(x,y) \in (\text{SIGMA } x:A. B \text{ } x)}. f \text{ } x \text{ } y)$
 $\langle \text{proof} \rangle$

Here we can eliminate the finiteness assumptions, by cases.

lemma *setprod-cartesian-product*:

$$\langle \text{proof} \rangle \quad \left(\prod_{x \in A}. \left(\prod_{y \in B}. f \ x \ y \right) \right) = \left(\prod_{(x,y) \in (A \lt * > B)}. f \ x \ y \right)$$

lemma *setprod-timesf*:

$$\langle \text{proof} \rangle \quad \text{setprod } (\%x. f \ x \ * \ g \ x) \ A = (\text{setprod } f \ A \ * \ \text{setprod } g \ A)$$

17.5.1 Properties in more restricted classes of structures

lemma *setprod-eq-1-iff* [simp]:

$$\langle \text{proof} \rangle \quad \text{finite } F \implies (\text{setprod } f \ F = 1) = (\text{ALL } a:F. f \ a = (1::\text{nat}))$$

lemma *setprod-zero*:

$$\langle \text{proof} \rangle \quad \text{finite } A \implies \text{EX } x: A. f \ x = (0::'a::\text{comm-semiring-1}) \implies \text{setprod } f \ A = 0$$

lemma *setprod-nonneg* [rule-format]:

$$\langle \text{proof} \rangle \quad (\text{ALL } x: A. (0::'a::\text{ordered-semidom}) \leq f \ x) \longrightarrow 0 \leq \text{setprod } f \ A$$

lemma *setprod-pos* [rule-format]: $(\text{ALL } x: A. (0::'a::\text{ordered-semidom}) < f \ x)$

$$\longrightarrow 0 < \text{setprod } f \ A$$

$\langle \text{proof} \rangle$

lemma *setprod-zero-iff* [simp]: $\text{finite } A \implies$

$$\langle \text{proof} \rangle \quad (\text{setprod } f \ A = (0::'a::\{\text{comm-semiring-1}, \text{no-zero-divisors}\})) = (\text{EX } x: A. f \ x = 0)$$

lemma *setprod-pos-nat*:

$$\langle \text{proof} \rangle \quad \text{finite } S \implies (\text{ALL } x: S. f \ x > (0::\text{nat})) \implies \text{setprod } f \ S > 0$$

lemma *setprod-pos-nat-iff* [simp]:

$$\langle \text{proof} \rangle \quad \text{finite } S \implies (\text{setprod } f \ S > 0) = (\text{ALL } x: S. f \ x > (0::\text{nat}))$$

lemma *setprod-Un*: $\text{finite } A \implies \text{finite } B \implies (\text{ALL } x: A \ \text{Int } B. f \ x \neq 0) \implies$

$$\langle \text{proof} \rangle \quad (\text{setprod } f \ (A \ \text{Un } B) :: 'a :: \{\text{field}\}) = \text{setprod } f \ A \ * \ \text{setprod } f \ B \ / \ \text{setprod } f \ (A \ \text{Int } B)$$

lemma *setprod-diff1*: $\text{finite } A \implies f \ a \neq 0 \implies$

$$\langle \text{proof} \rangle \quad (\text{setprod } f \ (A - \{a\}) :: 'a :: \{\text{field}\}) = (\text{if } a:A \text{ then } \text{setprod } f \ A \ / \ f \ a \text{ else } \text{setprod } f \ A)$$

lemma *setprod-inversef*: $\text{finite } A \implies$

$ALL\ x : A. f\ x \neq (0 :: 'a :: \{field, division-by-zero\}) ==>$
 $setprod\ (inverse \circ f)\ A = inverse\ (setprod\ f\ A)$
 $\langle proof \rangle$

lemma *setprod-dividef*:
 $[|finite\ A;$
 $\forall x \in A. g\ x \neq (0 :: 'a :: \{field, division-by-zero\})|]$
 $==> setprod\ (\%x. f\ x / g\ x)\ A = setprod\ f\ A / setprod\ g\ A$
 $\langle proof \rangle$

lemma *setprod-dvd-setprod* [rule-format]:
 $(ALL\ x : A. f\ x\ dvd\ g\ x) \longrightarrow setprod\ f\ A\ dvd\ setprod\ g\ A$
 $\langle proof \rangle$

lemma *setprod-dvd-setprod-subset*:
 $finite\ B \implies A \leq B \implies setprod\ f\ A\ dvd\ setprod\ f\ B$
 $\langle proof \rangle$

lemma *setprod-dvd-setprod-subset2*:
 $finite\ B \implies A \leq B \implies ALL\ x : A. (f\ x :: 'a :: comm-semiring-1)\ dvd\ g\ x \implies$
 $setprod\ f\ A\ dvd\ setprod\ g\ B$
 $\langle proof \rangle$

lemma *dvd-setprod*: $finite\ A \implies i : A \implies$
 $(f\ i :: 'a :: comm-semiring-1)\ dvd\ setprod\ f\ A$
 $\langle proof \rangle$

lemma *dvd-setsum* [rule-format]: $(ALL\ i : A. d\ dvd\ f\ i) \longrightarrow$
 $(d :: 'a :: comm-semiring-1)\ dvd\ (SUM\ x : A. f\ x)$
 $\langle proof \rangle$

17.6 Finite cardinality

This definition, although traditional, is ugly to work with: $card\ A == LEAST\ n. EX\ f. A = \{f\ i \mid i. i < n\}$. But now that we have *setsum* things are easy:

definition *card* :: $'a\ set \Rightarrow nat$
where $card\ A = setsum\ (\lambda x. 1)\ A$

lemma *card-empty* [simp]: $card\ \{\} = 0$
 $\langle proof \rangle$

lemma *card-infinite* [simp]: $\sim finite\ A ==> card\ A = 0$
 $\langle proof \rangle$

lemma *card-eq-setsum*: $card\ A = setsum\ (\%x. 1)\ A$
 $\langle proof \rangle$

lemma *card-insert-disjoint* [simp]:

finite A ==> x ∉ A ==> card (insert x A) = Suc(card A)
 ⟨proof⟩

lemma *card-insert-if*:

finite A ==> card (insert x A) = (if x:A then card A else Suc(card(A)))
 ⟨proof⟩

lemma *card-0-eq* [simp, noatp]: *finite A ==> (card A = 0) = (A = {})*
 ⟨proof⟩

lemma *card-eq-0-iff*: *(card A = 0) = (A = {} | ~ finite A)*
 ⟨proof⟩

lemma *card-Suc-Diff1*: *finite A ==> x: A ==> Suc (card (A - {x})) = card A*
 ⟨proof⟩

lemma *card-Diff-singleton*:

finite A ==> x: A ==> card (A - {x}) = card A - 1
 ⟨proof⟩

lemma *card-Diff-singleton-if*:

finite A ==> card (A - {x}) = (if x : A then card A - 1 else card A)
 ⟨proof⟩

lemma *card-Diff-insert*[simp]:

assumes *finite A and a:A and a ~: B*

shows *card(A - insert a B) = card(A - B) - 1*
 ⟨proof⟩

lemma *card-insert*: *finite A ==> card (insert x A) = Suc (card (A - {x}))*
 ⟨proof⟩

lemma *card-insert-le*: *finite A ==> card A <= card (insert x A)*
 ⟨proof⟩

lemma *card-mono*: $\llbracket \text{finite } B; A \subseteq B \rrbracket \implies \text{card } A \leq \text{card } B$
 ⟨proof⟩

lemma *card-seteq*: *finite B ==> (!A. A <= B ==> card B <= card A ==> A = B)*
 ⟨proof⟩

lemma *psubset-card-mono*: *finite B ==> A < B ==> card A < card B*
 ⟨proof⟩

lemma *card-Un-Int*: *finite A ==> finite B*

==> card A + card B = card (A Un B) + card (A Int B)
 ⟨proof⟩

lemma *card-Un-disjoint*: $\text{finite } A \implies \text{finite } B$
 $\implies A \text{ Int } B = \{\} \implies \text{card } (A \text{ Un } B) = \text{card } A + \text{card } B$
 $\langle \text{proof} \rangle$

lemma *card-Diff-subset*:
 $\text{finite } B \implies B \leq A \implies \text{card } (A - B) = \text{card } A - \text{card } B$
 $\langle \text{proof} \rangle$

lemma *card-Diff1-less*: $\text{finite } A \implies x: A \implies \text{card } (A - \{x\}) < \text{card } A$
 $\langle \text{proof} \rangle$

lemma *card-Diff2-less*:
 $\text{finite } A \implies x: A \implies y: A \implies \text{card } (A - \{x\} - \{y\}) < \text{card } A$
 $\langle \text{proof} \rangle$

lemma *card-Diff1-le*: $\text{finite } A \implies \text{card } (A - \{x\}) \leq \text{card } A$
 $\langle \text{proof} \rangle$

lemma *card-psubset*: $\text{finite } B \implies A \subseteq B \implies \text{card } A < \text{card } B \implies A < B$
 $\langle \text{proof} \rangle$

lemma *insert-partition*:
 $\llbracket x \notin F; \forall c1 \in \text{insert } x F. \forall c2 \in \text{insert } x F. c1 \neq c2 \longrightarrow c1 \cap c2 = \{\} \rrbracket$
 $\implies x \cap \bigcup F = \{\}$
 $\langle \text{proof} \rangle$

main cardinality theorem

lemma *card-partition* [rule-format]:
 $\text{finite } C \implies$
 $\text{finite } (\bigcup C) \dashrightarrow$
 $(\forall c \in C. \text{card } c = k) \dashrightarrow$
 $(\forall c1 \in C. \forall c2 \in C. c1 \neq c2 \dashrightarrow c1 \cap c2 = \{\}) \dashrightarrow$
 $k * \text{card}(C) = \text{card } (\bigcup C)$
 $\langle \text{proof} \rangle$

The form of a finite set of given cardinality

lemma *card-eq-SucD*:
assumes $\text{card } A = \text{Suc } k$
shows $\exists b B. A = \text{insert } b B \ \& \ b \notin B \ \& \ \text{card } B = k \ \& \ (k=0 \longrightarrow B=\{\})$
 $\langle \text{proof} \rangle$

lemma *card-Suc-eq*:
 $(\text{card } A = \text{Suc } k) =$
 $(\exists b B. A = \text{insert } b B \ \& \ b \notin B \ \& \ \text{card } B = k \ \& \ (k=0 \longrightarrow B=\{\}))$
 $\langle \text{proof} \rangle$

lemma *setsum-constant* [simp]: $(\sum x \in A. y) = \text{of-nat}(\text{card } A) * y$
 $\langle \text{proof} \rangle$

lemma *setprod-constant*: $\text{finite } A \implies (\prod x \in A. (y :: 'a :: \{\text{recpower}, \text{comm-monoid-mult}\}))$
 $= y^{\text{card } A}$
 $\langle \text{proof} \rangle$

lemma *setprod-gen-delta*:
assumes fS : $\text{finite } S$
shows $\text{setprod } (\lambda k. \text{if } k=a \text{ then } b \text{ k else } c) S = (\text{if } a \in S \text{ then } (b \ a :: 'a :: \{\text{comm-monoid-mult}, \text{recpower}\}) * c^{\text{card } S - 1} \text{ else } c^{\text{card } S})$
 $\langle \text{proof} \rangle$

lemma *setsum-bounded*:
assumes le : $\bigwedge i. i \in A \implies f\ i \leq (K :: 'a :: \{\text{semiring-1}, \text{pordered-ab-semigroup-add}\})$
shows $\text{setsum } f\ A \leq \text{of-nat}(\text{card } A) * K$
 $\langle \text{proof} \rangle$

17.6.1 Cardinality of unions

lemma *card-UN-disjoint*:
 $\text{finite } I \implies (\text{ALL } i:I. \text{finite } (A\ i)) \implies$
 $(\text{ALL } i:I. \text{ALL } j:I. i \neq j \longrightarrow A\ i \text{ Int } A\ j = \{\})$
 $\implies \text{card } (\text{UNION } I\ A) = (\sum i \in I. \text{card}(A\ i))$
 $\langle \text{proof} \rangle$

lemma *card-Union-disjoint*:
 $\text{finite } C \implies (\text{ALL } A:C. \text{finite } A) \implies$
 $(\text{ALL } A:C. \text{ALL } B:C. A \neq B \longrightarrow A \text{ Int } B = \{\})$
 $\implies \text{card } (\text{Union } C) = \text{setsum } \text{card } C$
 $\langle \text{proof} \rangle$

17.6.2 Cardinality of image

The image of a finite set can be expressed using *fold-image*.

lemma *image-eq-fold-image*:
 $\text{finite } A \implies f\ ' A = \text{fold-image } (op\ Un) (\%x. \{f\ x\}) \{\} A$
 $\langle \text{proof} \rangle$

lemma *card-image-le*: $\text{finite } A \implies \text{card } (f\ ' A) \leq \text{card } A$
 $\langle \text{proof} \rangle$

lemma *card-image*: $\text{inj-on } f\ A \implies \text{card } (f\ ' A) = \text{card } A$
 $\langle \text{proof} \rangle$

lemma *endo-inj-surj*: $\text{finite } A \implies f\ ' A \subseteq A \implies \text{inj-on } f\ A \implies f\ ' A = A$
 $\langle \text{proof} \rangle$

lemma *eq-card-imp-inj-on*:
 $[\text{finite } A; \text{card}(f\ ' A) = \text{card } A] \implies \text{inj-on } f\ A$

$\langle \text{proof} \rangle$

lemma *inj-on-iff-eq-card*:

$\text{finite } A \implies \text{inj-on } f \ A = (\text{card}(f \text{ ` } A) = \text{card } A)$
 $\langle \text{proof} \rangle$

lemma *card-inj-on-le*:

$[\text{inj-on } f \ A; f \text{ ` } A \subseteq B; \text{finite } B] \implies \text{card } A \leq \text{card } B$
 $\langle \text{proof} \rangle$

lemma *card-bij-eq*:

$[\text{inj-on } f \ A; f \text{ ` } A \subseteq B; \text{inj-on } g \ B; g \text{ ` } B \subseteq A; \\ \text{finite } A; \text{finite } B] \implies \text{card } A = \text{card } B$
 $\langle \text{proof} \rangle$

17.6.3 Cardinality of products

lemma *card-SigmaI* [*simp*]:

$[\text{finite } A; \text{ALL } a:A. \text{finite } (B \ a)] \\ \implies \text{card } (\text{SIGMA } x: A. B \ x) = (\sum a \in A. \text{card } (B \ a))$
 $\langle \text{proof} \rangle$

lemma *card-cartesian-product*: $\text{card } (A \lt*> B) = \text{card}(A) * \text{card}(B)$
 $\langle \text{proof} \rangle$

lemma *card-cartesian-product-singleton*: $\text{card}(\{x\} \lt*> A) = \text{card}(A)$
 $\langle \text{proof} \rangle$

17.6.4 Cardinality of sums

lemma *card-Plus*:

assumes *finite A and finite B*
shows $\text{card } (A \lt+> B) = \text{card } A + \text{card } B$
 $\langle \text{proof} \rangle$

17.6.5 Cardinality of the Powerset

lemma *card-Pow*: $\text{finite } A \implies \text{card } (\text{Pow } A) = \text{Suc } (\text{Suc } 0) \wedge \text{card } A$
 $\langle \text{proof} \rangle$

Relates to equivalence classes. Based on a theorem of F. Kammüller.

lemma *dvd-partition*:

$\text{finite } (\text{Union } C) \implies \\ \text{ALL } c : C. k \text{ dvd } \text{card } c \implies \\ (\text{ALL } c1: C. \text{ALL } c2: C. c1 \neq c2 \longrightarrow c1 \text{ Int } c2 = \{\}) \implies \\ k \text{ dvd } \text{card } (\text{Union } C)$
 $\langle \text{proof} \rangle$

17.6.6 Relating injectivity and surjectivity

lemma *finite-surj-inj*: $\text{finite}(A) \implies A \leq f^*A \implies \text{inj-on } f \ A$
 $\langle \text{proof} \rangle$

lemma *finite-UNIV-surj-inj*: **fixes** $f :: 'a \Rightarrow 'a$
shows $\text{finite}(\text{UNIV} :: 'a \text{ set}) \implies \text{surj } f \implies \text{inj } f$
 $\langle \text{proof} \rangle$

lemma *finite-UNIV-inj-surj*: **fixes** $f :: 'a \Rightarrow 'a$
shows $\text{finite}(\text{UNIV} :: 'a \text{ set}) \implies \text{inj } f \implies \text{surj } f$
 $\langle \text{proof} \rangle$

corollary *infinite-UNIV-nat*: $\sim \text{finite}(\text{UNIV} :: \text{nat set})$
 $\langle \text{proof} \rangle$

lemma *infinite-UNIV-char-0*:
 $\neg \text{finite}(\text{UNIV} :: 'a :: \text{semiring-char-0 set})$
 $\langle \text{proof} \rangle$

17.7 A fold functional for non-empty sets

Does not require start value.

inductive

$\text{fold1Set} :: ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \text{ set} \Rightarrow 'a \Rightarrow \text{bool}$
for $f :: 'a \Rightarrow 'a \Rightarrow 'a$

where

fold1Set-insertI [intro]:
 $\llbracket \text{fold-graph } f \ a \ A \ x; a \notin A \rrbracket \implies \text{fold1Set } f \ (\text{insert } a \ A) \ x$

constdefs

$\text{fold1} :: ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \text{ set} \Rightarrow 'a$
 $\text{fold1 } f \ A == \text{THE } x. \text{fold1Set } f \ A \ x$

lemma *fold1Set-nonempty*:

$\text{fold1Set } f \ A \ x \implies A \neq \{\}$
 $\langle \text{proof} \rangle$

inductive-cases *empty-fold1SetE* [elim!]: $\text{fold1Set } f \ \{\} \ x$

inductive-cases *insert-fold1SetE* [elim!]: $\text{fold1Set } f \ (\text{insert } a \ X) \ x$

lemma *fold1Set-sing* [iff]: $(\text{fold1Set } f \ \{a\} \ b) = (a = b)$
 $\langle \text{proof} \rangle$

lemma *fold1-singleton* [simp]: $\text{fold1 } f \ \{a\} = a$
 $\langle \text{proof} \rangle$

lemma *finite-nonempty-imp-fold1Set*:
 $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \exists x. \text{fold1Set } f \ A \ x$
 <proof>

First, some lemmas about *fold-graph*.

context *ab-semigroup-mult*
begin

lemma *fun-left-comm*: *fun-left-comm*(*op* *)
 <proof>

lemma *fold-graph-insert-swap*:
assumes *fold*: *fold-graph times* (*b::'a*) *A* *y* **and** $b \notin A$
shows *fold-graph times* *z* (*insert* *b* *A*) (*z* * *y*)
 <proof>

lemma *fold-graph-permute-diff*:
assumes *fold*: *fold-graph times* *b* *A* *x*
shows $\llbracket a \in A; b \notin A \rrbracket \implies \text{fold-graph times } a \ (\text{insert } b \ (A - \{a\})) \ x$
 <proof>

lemma *fold1-eq-fold*:
assumes *finite* *A* $a \notin A$ **shows** *fold1 times* (*insert* *a* *A*) = *fold times* *a* *A*
 <proof>

lemma *nonempty-iff*: $(A \neq \{\}) = (\exists x \ B. A = \text{insert } x \ B \ \& \ x \notin B)$
 <proof>

lemma *fold1-insert*:
assumes *nonempty*: $A \neq \{\}$ **and** *A*: *finite* *A* $x \notin A$
shows *fold1 times* (*insert* *x* *A*) = *x* * *fold1 times* *A*
 <proof>

end

context *ab-semigroup-idem-mult*
begin

lemma *fun-left-comm-idem*: *fun-left-comm-idem*(*op* *)
 <proof>

lemma *fold1-insert-idem* [*simp*]:
assumes *nonempty*: $A \neq \{\}$ **and** *A*: *finite* *A*
shows *fold1 times* (*insert* *x* *A*) = *x* * *fold1 times* *A*
 <proof>

lemma *hom-fold1-commute*:
assumes *hom*: $\llbracket x \ y. h \ (x * y) = h \ x * h \ y \rrbracket$

and N : *finite* N $N \neq \{\}$ **shows** h (*fold1 times* N) = *fold1 times* ($h \text{ ‘ } N$)
 ⟨*proof*⟩

end

Now the recursion rules for definitions:

lemma *fold1-singleton-def*: $g = \text{fold1 } f \implies g \{a\} = a$
 ⟨*proof*⟩

lemma (**in** *ab-semigroup-mult*) *fold1-insert-def*:
 $\llbracket g = \text{fold1 times}; \text{finite } A; x \notin A; A \neq \{\} \rrbracket \implies g (\text{insert } x \ A) = x * g \ A$
 ⟨*proof*⟩

lemma (**in** *ab-semigroup-idem-mult*) *fold1-insert-idem-def*:
 $\llbracket g = \text{fold1 times}; \text{finite } A; A \neq \{\} \rrbracket \implies g (\text{insert } x \ A) = x * g \ A$
 ⟨*proof*⟩

17.7.1 Determinacy for *fold1Set*

declare

empty-fold-graphE [*rule del*] *fold-graph.intros* [*rule del*]
empty-fold1SetE [*rule del*] *insert-fold1SetE* [*rule del*]
 — No more proofs involve these relations.

17.7.2 Lemmas about *fold1*

context *ab-semigroup-mult*
begin

lemma *fold1-Un*:
assumes A : *finite* A $A \neq \{\}$
shows $\text{finite } B \implies B \neq \{\} \implies A \text{ Int } B = \{\} \implies$
 $\text{fold1 times } (A \text{ Un } B) = \text{fold1 times } A * \text{fold1 times } B$
 ⟨*proof*⟩

lemma *fold1-in*:
assumes A : *finite* (A) $A \neq \{\}$ **and** *elem*: $\bigwedge x \ y. x * y \in \{x, y\}$
shows $\text{fold1 times } A \in A$
 ⟨*proof*⟩

end

lemma (**in** *ab-semigroup-idem-mult*) *fold1-Un2*:
assumes A : *finite* A $A \neq \{\}$
shows $\text{finite } B \implies B \neq \{\} \implies$
 $\text{fold1 times } (A \text{ Un } B) = \text{fold1 times } A * \text{fold1 times } B$
 ⟨*proof*⟩

17.7.3 Fold1 in lattices with \inf and \sup

As an application of *fold1* we define infimum and supremum in (not necessarily complete!) lattices over (non-empty) sets by means of *fold1*.

context *lower-semilattice*
begin

lemma *ab-semigroup-idem-mult-inf*:
ab-semigroup-idem-mult inf
 $\langle \text{proof} \rangle$

lemma *below-fold1-iff*:
assumes *finite A A $\neq \{\}$*
shows $x \leq \text{fold1 } \inf A \iff (\forall a \in A. x \leq a)$
 $\langle \text{proof} \rangle$

lemma *fold1-belowI*:
assumes *finite A*
and $a \in A$
shows $\text{fold1 } \inf A \leq a$
 $\langle \text{proof} \rangle$

end

lemma (**in** *upper-semilattice*) *ab-semigroup-idem-mult-sup*:
ab-semigroup-idem-mult sup
 $\langle \text{proof} \rangle$

context *lattice*
begin

definition
 $\text{Inf-fin} :: 'a \text{ set} \Rightarrow 'a \text{ } (\bigcap_{\text{fin}} [900] 900)$
where
 $\text{Inf-fin} = \text{fold1 } \inf$

definition
 $\text{Sup-fin} :: 'a \text{ set} \Rightarrow 'a \text{ } (\bigcup_{\text{fin}} [900] 900)$
where
 $\text{Sup-fin} = \text{fold1 } \sup$

lemma *Inf-le-Sup [simp]*: $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \bigcap_{\text{fin}} A \leq \bigcup_{\text{fin}} A$
 $\langle \text{proof} \rangle$

lemma *sup-Inf-absorb [simp]*:
 $\text{finite } A \implies a \in A \implies \sup a (\bigcap_{\text{fin}} A) = a$
 $\langle \text{proof} \rangle$

lemma *inf-Sup-absorb [simp]*:

$finite\ A \implies a \in A \implies inf\ a\ (\sqcup_{fin} A) = a$
 $\langle proof \rangle$

end

context *distrib-lattice*
begin

lemma *sup-Inf1-distrib*:
assumes *finite A*
and $A \neq \{\}$
shows $sup\ x\ (\sqcap_{fin} A) = \sqcap_{fin} \{sup\ x\ a \mid a. a \in A\}$
 $\langle proof \rangle$

lemma *sup-Inf2-distrib*:
assumes $A: finite\ A\ A \neq \{\}$ **and** $B: finite\ B\ B \neq \{\}$
shows $sup\ (\sqcap_{fin} A)\ (\sqcap_{fin} B) = \sqcap_{fin} \{sup\ a\ b \mid a\ b. a \in A \wedge b \in B\}$
 $\langle proof \rangle$

lemma *inf-Sup1-distrib*:
assumes *finite A* **and** $A \neq \{\}$
shows $inf\ x\ (\sqcup_{fin} A) = \sqcup_{fin} \{inf\ x\ a \mid a. a \in A\}$
 $\langle proof \rangle$

lemma *inf-Sup2-distrib*:
assumes $A: finite\ A\ A \neq \{\}$ **and** $B: finite\ B\ B \neq \{\}$
shows $inf\ (\sqcup_{fin} A)\ (\sqcup_{fin} B) = \sqcup_{fin} \{inf\ a\ b \mid a\ b. a \in A \wedge b \in B\}$
 $\langle proof \rangle$

end

context *complete-lattice*
begin

Coincidence on finite sets in complete lattices:

lemma *Inf-fin-Inf*:
assumes *finite A* **and** $A \neq \{\}$
shows $\sqcap_{fin} A = Inf\ A$
 $\langle proof \rangle$

lemma *Sup-fin-Sup*:
assumes *finite A* **and** $A \neq \{\}$
shows $\sqcup_{fin} A = Sup\ A$
 $\langle proof \rangle$

end

17.7.4 Fold1 in linear orders with \min and \max

As an application of *fold1* we define minimum and maximum in (not necessarily complete!) linear orders over (non-empty) sets by means of *fold1*.

context *linorder*
begin

lemma *ab-semigroup-idem-mult-min*:
ab-semigroup-idem-mult min
 ⟨proof⟩

lemma *ab-semigroup-idem-mult-max*:
ab-semigroup-idem-mult max
 ⟨proof⟩

lemma *min-lattice*:
lower-semilattice (op ≤) (op <) min
 ⟨proof⟩

lemma *max-lattice*:
lower-semilattice (op ≥) (op >) max
 ⟨proof⟩

lemma *dual-max*:
ord.max (op ≥) = min
 ⟨proof⟩

lemma *dual-min*:
ord.min (op ≥) = max
 ⟨proof⟩

lemma *strict-below-fold1-iff*:
assumes *finite A and $A \neq \{\}$*
shows $x < \text{fold1 min } A \longleftrightarrow (\forall a \in A. x < a)$
 ⟨proof⟩

lemma *fold1-below-iff*:
assumes *finite A and $A \neq \{\}$*
shows $\text{fold1 min } A \leq x \longleftrightarrow (\exists a \in A. a \leq x)$
 ⟨proof⟩

lemma *fold1-strict-below-iff*:
assumes *finite A and $A \neq \{\}$*
shows $\text{fold1 min } A < x \longleftrightarrow (\exists a \in A. a < x)$
 ⟨proof⟩

lemma *fold1-antimono*:
assumes $A \neq \{\}$ **and** $A \subseteq B$ **and** *finite B*
shows $\text{fold1 min } B \leq \text{fold1 min } A$

$\langle proof \rangle$

definition

$Min :: 'a \text{ set} \Rightarrow 'a$

where

$Min = fold1 \ min$

definition

$Max :: 'a \text{ set} \Rightarrow 'a$

where

$Max = fold1 \ max$

lemmas $Min-singleton \ [simp] = fold1-singleton-def \ [OF \ Min-def]$

lemmas $Max-singleton \ [simp] = fold1-singleton-def \ [OF \ Max-def]$

lemma $Min-insert \ [simp]$:

assumes $finite \ A$ **and** $A \neq \{\}$

shows $Min \ (insert \ x \ A) = min \ x \ (Min \ A)$

$\langle proof \rangle$

lemma $Max-insert \ [simp]$:

assumes $finite \ A$ **and** $A \neq \{\}$

shows $Max \ (insert \ x \ A) = max \ x \ (Max \ A)$

$\langle proof \rangle$

lemma $Min-in \ [simp]$:

assumes $finite \ A$ **and** $A \neq \{\}$

shows $Min \ A \in A$

$\langle proof \rangle$

lemma $Max-in \ [simp]$:

assumes $finite \ A$ **and** $A \neq \{\}$

shows $Max \ A \in A$

$\langle proof \rangle$

lemma $Min-Un$:

assumes $finite \ A$ **and** $A \neq \{\}$ **and** $finite \ B$ **and** $B \neq \{\}$

shows $Min \ (A \cup B) = min \ (Min \ A) \ (Min \ B)$

$\langle proof \rangle$

lemma $Max-Un$:

assumes $finite \ A$ **and** $A \neq \{\}$ **and** $finite \ B$ **and** $B \neq \{\}$

shows $Max \ (A \cup B) = max \ (Max \ A) \ (Max \ B)$

$\langle proof \rangle$

lemma $hom-Min-commute$:

assumes $\bigwedge x \ y. \ h \ (min \ x \ y) = min \ (h \ x) \ (h \ y)$

and $finite \ N$ **and** $N \neq \{\}$

shows $h \ (Min \ N) = Min \ (h \ ` \ N)$

$\langle proof \rangle$

lemma *hom-Max-commute*:

assumes $\bigwedge x y. h (max\ x\ y) = max\ (h\ x)\ (h\ y)$
 and *finite* N and $N \neq \{\}$
 shows $h (Max\ N) = Max\ (h\ ` N)$

$\langle proof \rangle$

lemma *Min-le [simp]*:

assumes *finite* A and $x \in A$
 shows $Min\ A \leq x$

$\langle proof \rangle$

lemma *Max-ge [simp]*:

assumes *finite* A and $x \in A$
 shows $x \leq Max\ A$

$\langle proof \rangle$

lemma *Min-ge-iff [simp, noatp]*:

assumes *finite* A and $A \neq \{\}$
 shows $x \leq Min\ A \longleftrightarrow (\forall a \in A. x \leq a)$

$\langle proof \rangle$

lemma *Max-le-iff [simp, noatp]*:

assumes *finite* A and $A \neq \{\}$
 shows $Max\ A \leq x \longleftrightarrow (\forall a \in A. a \leq x)$

$\langle proof \rangle$

lemma *Min-gr-iff [simp, noatp]*:

assumes *finite* A and $A \neq \{\}$
 shows $x < Min\ A \longleftrightarrow (\forall a \in A. x < a)$

$\langle proof \rangle$

lemma *Max-less-iff [simp, noatp]*:

assumes *finite* A and $A \neq \{\}$
 shows $Max\ A < x \longleftrightarrow (\forall a \in A. a < x)$

$\langle proof \rangle$

lemma *Min-le-iff [noatp]*:

assumes *finite* A and $A \neq \{\}$
 shows $Min\ A \leq x \longleftrightarrow (\exists a \in A. a \leq x)$

$\langle proof \rangle$

lemma *Max-ge-iff [noatp]*:

assumes *finite* A and $A \neq \{\}$
 shows $x \leq Max\ A \longleftrightarrow (\exists a \in A. x \leq a)$

$\langle proof \rangle$

lemma *Min-less-iff [noatp]*:

assumes *finite A* and $A \neq \{\}$
 shows $\text{Min } A < x \iff (\exists a \in A. a < x)$
 $\langle \text{proof} \rangle$

lemma *Max-gr-iff* [noatp]:
 assumes *finite A* and $A \neq \{\}$
 shows $x < \text{Max } A \iff (\exists a \in A. x < a)$
 $\langle \text{proof} \rangle$

lemma *Min-eqI*:
 assumes *finite A*
 assumes $\bigwedge y. y \in A \implies y \geq x$
 and $x \in A$
 shows $\text{Min } A = x$
 $\langle \text{proof} \rangle$

lemma *Max-eqI*:
 assumes *finite A*
 assumes $\bigwedge y. y \in A \implies y \leq x$
 and $x \in A$
 shows $\text{Max } A = x$
 $\langle \text{proof} \rangle$

lemma *Min-antimono*:
 assumes $M \subseteq N$ and $M \neq \{\}$ and *finite N*
 shows $\text{Min } N \leq \text{Min } M$
 $\langle \text{proof} \rangle$

lemma *Max-mono*:
 assumes $M \subseteq N$ and $M \neq \{\}$ and *finite N*
 shows $\text{Max } M \leq \text{Max } N$
 $\langle \text{proof} \rangle$

lemma *finite-linorder-induct*[consumes 1, case-names empty insert]:
 $\text{finite } A \implies P \{\} \implies$
 $(\forall b. \text{finite } A \implies \forall a \in A. a < b \implies P A \implies P(\text{insert } b A))$
 $\implies P A$
 $\langle \text{proof} \rangle$

end

context *ordered-ab-semigroup-add*
 begin

lemma *add-Min-commute*:
 fixes k
 assumes *finite N* and $N \neq \{\}$
 shows $k + \text{Min } N = \text{Min } \{k + m \mid m. m \in N\}$
 $\langle \text{proof} \rangle$

```

lemma add-Max-commute:
  fixes k
  assumes finite N and  $N \neq \{\}$ 
  shows  $k + \text{Max } N = \text{Max } \{k + m \mid m. m \in N\}$ 
  <proof>

end

end

```

18 Relation: Relations

```

theory Relation
imports Datatype Finite-Set
begin

```

18.1 Definitions

```

definition
  converse :: ('a * 'b) set => ('b * 'a) set
  ((-^-1) [1000] 999) where
   $r^{\wedge}-1 == \{(y, x). (x, y) : r\}$ 

```

```

notation (xsymbols)
  converse ((--1) [1000] 999)

```

```

definition
  rel-comp :: [('b * 'c) set, ('a * 'b) set] => ('a * 'c) set
  (infixr 0 75) where
   $r \text{ O } s == \{(x, z). \text{EX } y. (x, y) : s \ \& \ (y, z) : r\}$ 

```

```

definition
  Image :: [('a * 'b) set, 'a set] => 'b set
  (infixl “ 90) where
   $r \text{ “ } s == \{y. \text{EX } x:s. (x, y):r\}$ 

```

```

definition
  Id :: ('a * 'a) set where — the identity relation
   $\text{Id} == \{p. \text{EX } x. p = (x, x)\}$ 

```

```

definition
  Id-on :: 'a set => ('a * 'a) set where — diagonal: identity over a set
   $\text{Id-on } A == \bigcup_{x \in A}. \{(x, x)\}$ 

```

```

definition
  Domain :: ('a * 'b) set => 'a set where
   $\text{Domain } r == \{x. \text{EX } y. (x, y):r\}$ 

```

definition

$Range :: ('a * 'b) set \Rightarrow 'b set$ **where**
 $Range\ r == Domain(r^{-1})$

definition

$Field :: ('a * 'a) set \Rightarrow 'a set$ **where**
 $Field\ r == Domain\ r \cup Range\ r$

definition

$refl-on :: ['a set, ('a * 'a) set] \Rightarrow bool$ **where** — reflexivity over a set
 $refl-on\ A\ r == r \subseteq A \times A \ \& \ (ALL\ x: A. (x,x) : r)$

abbreviation

$refl :: ('a * 'a) set \Rightarrow bool$ **where** — reflexivity over a type
 $refl == refl-on\ UNIV$

definition

$sym :: ('a * 'a) set \Rightarrow bool$ **where** — symmetry predicate
 $sym\ r == ALL\ x\ y. (x,y):r \longrightarrow (y,x):r$

definition

$antisym :: ('a * 'a) set \Rightarrow bool$ **where** — antisymmetry predicate
 $antisym\ r == ALL\ x\ y. (x,y):r \longrightarrow (y,x):r \longrightarrow x=y$

definition

$trans :: ('a * 'a) set \Rightarrow bool$ **where** — transitivity predicate
 $trans\ r == (ALL\ x\ y\ z. (x,y):r \longrightarrow (y,z):r \longrightarrow (x,z):r)$

definition

$irrefl :: ('a * 'a) set \Rightarrow bool$ **where**
 $irrefl\ r \equiv \forall x. (x,x) \notin r$

definition

$total-on :: 'a set \Rightarrow ('a * 'a) set \Rightarrow bool$ **where**
 $total-on\ A\ r \equiv \forall x \in A. \forall y \in A. x \neq y \longrightarrow (x,y) \in r \vee (y,x) \in r$

abbreviation $total \equiv total-on\ UNIV$

definition

$single-valued :: ('a * 'b) set \Rightarrow bool$ **where**
 $single-valued\ r == ALL\ x\ y. (x,y):r \longrightarrow (ALL\ z. (x,z):r \longrightarrow y=z)$

definition

$inv-image :: ('b * 'b) set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a * 'a) set$ **where**
 $inv-image\ r\ f == \{(x, y). (f\ x, f\ y) : r\}$

18.2 The identity relation

lemma *IdI* [*intro*]: $(a, a) : Id$
 $\langle proof \rangle$

lemma *IdE* [*elim!*]: $p : Id \implies (!x. p = (x, x) \implies P) \implies P$
 $\langle proof \rangle$

lemma *pair-in-Id-conv* [*iff*]: $((a, b) : Id) = (a = b)$
 $\langle proof \rangle$

lemma *refl-Id*: *refl Id*
 $\langle proof \rangle$

lemma *antisym-Id*: *antisym Id*
 — A strange result, since *Id* is also symmetric.
 $\langle proof \rangle$

lemma *sym-Id*: *sym Id*
 $\langle proof \rangle$

lemma *trans-Id*: *trans Id*
 $\langle proof \rangle$

18.3 Diagonal: identity over a set

lemma *Id-on-empty* [*simp*]: *Id-on* $\{\} = \{\}$
 $\langle proof \rangle$

lemma *Id-on-eqI*: $a = b \implies a : A \implies (a, b) : Id-on A$
 $\langle proof \rangle$

lemma *Id-onI* [*intro!*, *noatp*]: $a : A \implies (a, a) : Id-on A$
 $\langle proof \rangle$

lemma *Id-onE* [*elim!*]:
 $c : Id-on A \implies (!x. x : A \implies c = (x, x) \implies P) \implies P$
 — The general elimination rule.
 $\langle proof \rangle$

lemma *Id-on-iff*: $((x, y) : Id-on A) = (x = y \ \& \ x : A)$
 $\langle proof \rangle$

lemma *Id-on-subset-Times*: *Id-on* $A \subseteq A \times A$
 $\langle proof \rangle$

18.4 Composition of two relations

lemma *rel-compI* [*intro*]:
 $(a, b) : s \implies (b, c) : r \implies (a, c) : r \circ s$

$\langle \text{proof} \rangle$

lemma *rel-compE* [*elim!*]: $xz : r \ O \ s \implies$
 $(!!x \ y \ z. \ xz = (x, z) \implies (x, y) : s \implies (y, z) : r \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *rel-compEpair*:
 $(a, c) : r \ O \ s \implies (!!y. (a, y) : s \implies (y, c) : r \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *R-O-Id* [*simp*]: $R \ O \ Id = R$
 $\langle \text{proof} \rangle$

lemma *Id-O-R* [*simp*]: $Id \ O \ R = R$
 $\langle \text{proof} \rangle$

lemma *rel-comp-empty1* [*simp*]: $\{\} \ O \ R = \{\}$
 $\langle \text{proof} \rangle$

lemma *rel-comp-empty2* [*simp*]: $R \ O \ \{\} = \{\}$
 $\langle \text{proof} \rangle$

lemma *O-assoc*: $(R \ O \ S) \ O \ T = R \ O \ (S \ O \ T)$
 $\langle \text{proof} \rangle$

lemma *trans-O-subset*: $\text{trans } r \implies r \ O \ r \subseteq r$
 $\langle \text{proof} \rangle$

lemma *rel-comp-mono*: $r' \subseteq r \implies s' \subseteq s \implies (r' \ O \ s') \subseteq (r \ O \ s)$
 $\langle \text{proof} \rangle$

lemma *rel-comp-subset-Sigma*:
 $s \subseteq A \times B \implies r \subseteq B \times C \implies (r \ O \ s) \subseteq A \times C$
 $\langle \text{proof} \rangle$

lemma *rel-comp-distrib* [*simp*]: $R \ O \ (S \cup T) = (R \ O \ S) \cup (R \ O \ T)$
 $\langle \text{proof} \rangle$

lemma *rel-comp-distrib2* [*simp*]: $(S \cup T) \ O \ R = (S \ O \ R) \cup (T \ O \ R)$
 $\langle \text{proof} \rangle$

18.5 Reflexivity

lemma *refl-onI*: $r \subseteq A \times A \implies (!!x. x : A \implies (x, x) : r) \implies \text{refl-on } A \ r$
 $\langle \text{proof} \rangle$

lemma *refl-onD*: $\text{refl-on } A \ r \implies a : A \implies (a, a) : r$
 $\langle \text{proof} \rangle$

lemma *refl-onD1*: $\text{refl-on } A \ r \implies (x, y) : r \implies x : A$
 $\langle \text{proof} \rangle$

lemma *refl-onD2*: $\text{refl-on } A \ r \implies (x, y) : r \implies y : A$
 $\langle \text{proof} \rangle$

lemma *refl-on-Int*: $\text{refl-on } A \ r \implies \text{refl-on } B \ s \implies \text{refl-on } (A \cap B) \ (r \cap s)$
 $\langle \text{proof} \rangle$

lemma *refl-on-Un*: $\text{refl-on } A \ r \implies \text{refl-on } B \ s \implies \text{refl-on } (A \cup B) \ (r \cup s)$
 $\langle \text{proof} \rangle$

lemma *refl-on-INTER*:
 $\text{ALL } x:S. \text{refl-on } (A \ x) \ (r \ x) \implies \text{refl-on } (\text{INTER } S \ A) \ (\text{INTER } S \ r)$
 $\langle \text{proof} \rangle$

lemma *refl-on-UNION*:
 $\text{ALL } x:S. \text{refl-on } (A \ x) \ (r \ x) \implies \text{refl-on } (\text{UNION } S \ A) \ (\text{UNION } S \ r)$
 $\langle \text{proof} \rangle$

lemma *refl-on-empty[simp]*: $\text{refl-on } \{ \} \ \{ \}$
 $\langle \text{proof} \rangle$

lemma *refl-on-Id-on*: $\text{refl-on } A \ (\text{Id-on } A)$
 $\langle \text{proof} \rangle$

18.6 Antisymmetry

lemma *antisymI*:
 $(\forall x \ y. (x, y) : r \implies (y, x) : r \implies x=y) \implies \text{antisym } r$
 $\langle \text{proof} \rangle$

lemma *antisymD*: $\text{antisym } r \implies (a, b) : r \implies (b, a) : r \implies a = b$
 $\langle \text{proof} \rangle$

lemma *antisym-subset*: $r \subseteq s \implies \text{antisym } s \implies \text{antisym } r$
 $\langle \text{proof} \rangle$

lemma *antisym-empty [simp]*: $\text{antisym } \{ \}$
 $\langle \text{proof} \rangle$

lemma *antisym-Id-on [simp]*: $\text{antisym } (\text{Id-on } A)$
 $\langle \text{proof} \rangle$

18.7 Symmetry

lemma *symI*: $(\forall a \ b. (a, b) : r \implies (b, a) : r) \implies \text{sym } r$
 $\langle \text{proof} \rangle$

lemma *symD*: $\text{sym } r \implies (a, b) : r \implies (b, a) : r$

$\langle proof \rangle$

lemma *sym-Int*: $sym\ r ==> sym\ s ==> sym\ (r \cap s)$
 $\langle proof \rangle$

lemma *sym-Un*: $sym\ r ==> sym\ s ==> sym\ (r \cup s)$
 $\langle proof \rangle$

lemma *sym-INTER*: $ALL\ x:S.\ sym\ (r\ x) ==> sym\ (INTER\ S\ r)$
 $\langle proof \rangle$

lemma *sym-UNION*: $ALL\ x:S.\ sym\ (r\ x) ==> sym\ (UNION\ S\ r)$
 $\langle proof \rangle$

lemma *sym-Id-on [simp]*: $sym\ (Id-on\ A)$
 $\langle proof \rangle$

18.8 Transitivity

lemma *transI*:
 $(!!x\ y\ z.\ (x,\ y) : r ==> (y,\ z) : r ==> (x,\ z) : r) ==> trans\ r$
 $\langle proof \rangle$

lemma *transD*: $trans\ r ==> (a,\ b) : r ==> (b,\ c) : r ==> (a,\ c) : r$
 $\langle proof \rangle$

lemma *trans-Int*: $trans\ r ==> trans\ s ==> trans\ (r \cap s)$
 $\langle proof \rangle$

lemma *trans-INTER*: $ALL\ x:S.\ trans\ (r\ x) ==> trans\ (INTER\ S\ r)$
 $\langle proof \rangle$

lemma *trans-Id-on [simp]*: $trans\ (Id-on\ A)$
 $\langle proof \rangle$

lemma *trans-diff-Id*: $trans\ r ==> antisym\ r ==> trans\ (r - Id)$
 $\langle proof \rangle$

18.9 Irreflexivity

lemma *irrefl-diff-Id [simp]*: $irrefl\ (r - Id)$
 $\langle proof \rangle$

18.10 Totality

lemma *total-on-empty [simp]*: $total-on\ \{\}\ r$
 $\langle proof \rangle$

lemma *total-on-diff-Id [simp]*: $total-on\ A\ (r - Id) = total-on\ A\ r$
 $\langle proof \rangle$

18.11 Converse

lemma *converse-iff* [iff]: $((a,b): r^{-1}) = ((b,a): r)$
 $\langle proof \rangle$

lemma *converseI*[sym]: $(a, b): r ==> (b, a): r^{-1}$
 $\langle proof \rangle$

lemma *converseD*[sym]: $(a,b): r^{-1} ==> (b, a): r$
 $\langle proof \rangle$

lemma *converseE* [elim!]:
 $yx: r^{-1} ==> (!x y. yx = (y, x) ==> (x, y): r ==> P) ==> P$
 — More general than *converseD*, as it “splits” the member of the relation.
 $\langle proof \rangle$

lemma *converse-converse* [simp]: $(r^{-1})^{-1} = r$
 $\langle proof \rangle$

lemma *converse-rel-comp*: $(r \ O \ s)^{-1} = s^{-1} \ O \ r^{-1}$
 $\langle proof \rangle$

lemma *converse-Int*: $(r \cap s)^{-1} = r^{-1} \cap s^{-1}$
 $\langle proof \rangle$

lemma *converse-Un*: $(r \cup s)^{-1} = r^{-1} \cup s^{-1}$
 $\langle proof \rangle$

lemma *converse-INTER*: $(INTER \ S \ r)^{-1} = (INT \ x:S. (r \ x)^{-1})$
 $\langle proof \rangle$

lemma *converse-UNION*: $(UNION \ S \ r)^{-1} = (UN \ x:S. (r \ x)^{-1})$
 $\langle proof \rangle$

lemma *converse-Id* [simp]: $Id^{-1} = Id$
 $\langle proof \rangle$

lemma *converse-Id-on* [simp]: $(Id-on \ A)^{-1} = Id-on \ A$
 $\langle proof \rangle$

lemma *refl-on-converse* [simp]: $refl-on \ A \ (converse \ r) = refl-on \ A \ r$
 $\langle proof \rangle$

lemma *sym-converse* [simp]: $sym \ (converse \ r) = sym \ r$
 $\langle proof \rangle$

lemma *antisym-converse* [simp]: $antisym \ (converse \ r) = antisym \ r$
 $\langle proof \rangle$

lemma *trans-converse* [simp]: $trans \ (converse \ r) = trans \ r$

$\langle proof \rangle$

lemma *sym-conv-converse-eq*: $sym\ r = (r^{\wedge}-1 = r)$
 $\langle proof \rangle$

lemma *sym-Un-converse*: $sym\ (r \cup r^{\wedge}-1)$
 $\langle proof \rangle$

lemma *sym-Int-converse*: $sym\ (r \cap r^{\wedge}-1)$
 $\langle proof \rangle$

lemma *total-on-converse[simp]*: $total-on\ A\ (r^{\wedge}-1) = total-on\ A\ r$
 $\langle proof \rangle$

18.12 Domain

declare *Domain-def* [noatp]

lemma *Domain-iff*: $(a : Domain\ r) = (EX\ y.\ (a, y) : r)$
 $\langle proof \rangle$

lemma *DomainI* [intro]: $(a, b) : r ==> a : Domain\ r$
 $\langle proof \rangle$

lemma *DomainE* [elim!]:
 $a : Domain\ r ==> (!y.\ (a, y) : r ==> P) ==> P$
 $\langle proof \rangle$

lemma *Domain-empty* [simp]: $Domain\ \{\} = \{\}$
 $\langle proof \rangle$

lemma *Domain-insert*: $Domain\ (insert\ (a, b)\ r) = insert\ a\ (Domain\ r)$
 $\langle proof \rangle$

lemma *Domain-Id* [simp]: $Domain\ Id = UNIV$
 $\langle proof \rangle$

lemma *Domain-Id-on* [simp]: $Domain\ (Id-on\ A) = A$
 $\langle proof \rangle$

lemma *Domain-Un-eq*: $Domain(A \cup B) = Domain(A) \cup Domain(B)$
 $\langle proof \rangle$

lemma *Domain-Int-subset*: $Domain(A \cap B) \subseteq Domain(A) \cap Domain(B)$
 $\langle proof \rangle$

lemma *Domain-Diff-subset*: $Domain(A) - Domain(B) \subseteq Domain(A - B)$
 $\langle proof \rangle$

lemma *Domain-Union*: $\text{Domain } (\text{Union } S) = (\bigcup A \in S. \text{Domain } A)$
 $\langle \text{proof} \rangle$

lemma *Domain-converse* [simp]: $\text{Domain}(r^{-1}) = \text{Range } r$
 $\langle \text{proof} \rangle$

lemma *Domain-mono*: $r \subseteq s \implies \text{Domain } r \subseteq \text{Domain } s$
 $\langle \text{proof} \rangle$

lemma *fst-eq-Domain*: $\text{fst } R = \text{Domain } R$
 $\langle \text{proof} \rangle$

lemma *Domain-dprod* [simp]: $\text{Domain } (\text{dprod } r \ s) = \text{uprod } (\text{Domain } r) (\text{Domain } s)$
 $\langle \text{proof} \rangle$

lemma *Domain-dsum* [simp]: $\text{Domain } (\text{dsum } r \ s) = \text{usum } (\text{Domain } r) (\text{Domain } s)$
 $\langle \text{proof} \rangle$

18.13 Range

lemma *Range-iff*: $(a : \text{Range } r) = (\exists y. (y, a) : r)$
 $\langle \text{proof} \rangle$

lemma *RangeI* [intro]: $(a, b) : r \implies b : \text{Range } r$
 $\langle \text{proof} \rangle$

lemma *RangeE* [elim!]: $b : \text{Range } r \implies (!x. (x, b) : r \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *Range-empty* [simp]: $\text{Range } \{\} = \{\}$
 $\langle \text{proof} \rangle$

lemma *Range-insert*: $\text{Range } (\text{insert } (a, b) \ r) = \text{insert } b \ (\text{Range } r)$
 $\langle \text{proof} \rangle$

lemma *Range-Id* [simp]: $\text{Range } \text{Id} = \text{UNIV}$
 $\langle \text{proof} \rangle$

lemma *Range-Id-on* [simp]: $\text{Range } (\text{Id-on } A) = A$
 $\langle \text{proof} \rangle$

lemma *Range-Un-eq*: $\text{Range}(A \cup B) = \text{Range}(A) \cup \text{Range}(B)$
 $\langle \text{proof} \rangle$

lemma *Range-Int-subset*: $\text{Range}(A \cap B) \subseteq \text{Range}(A) \cap \text{Range}(B)$
 $\langle \text{proof} \rangle$

lemma *Range-Diff-subset*: $\text{Range}(A) - \text{Range}(B) \subseteq \text{Range}(A - B)$
 $\langle \text{proof} \rangle$

lemma *Range-Union*: $\text{Range} (\text{Union } S) = (\bigcup A \in S. \text{Range } A)$
 $\langle \text{proof} \rangle$

lemma *Range-converse[simp]*: $\text{Range}(r^{-1}) = \text{Domain } r$
 $\langle \text{proof} \rangle$

lemma *snd-eq-Range*: $\text{snd} \circ R = \text{Range } R$
 $\langle \text{proof} \rangle$

18.14 Field

lemma *mono-Field*: $r \subseteq s \implies \text{Field } r \subseteq \text{Field } s$
 $\langle \text{proof} \rangle$

lemma *Field-empty[simp]*: $\text{Field } \{\} = \{\}$
 $\langle \text{proof} \rangle$

lemma *Field-insert[simp]*: $\text{Field} (\text{insert } (a, b) r) = \{a, b\} \cup \text{Field } r$
 $\langle \text{proof} \rangle$

lemma *Field-Un[simp]*: $\text{Field} (r \cup s) = \text{Field } r \cup \text{Field } s$
 $\langle \text{proof} \rangle$

lemma *Field-Union[simp]*: $\text{Field} (\bigcup R) = \bigcup (\text{Field } R)$
 $\langle \text{proof} \rangle$

lemma *Field-converse[simp]*: $\text{Field}(r^{-1}) = \text{Field } r$
 $\langle \text{proof} \rangle$

18.15 Image of a set under a relation

declare *Image-def* [noatp]

lemma *Image-iff*: $(b : r \circ A) = (\exists x : A. (x, b) : r)$
 $\langle \text{proof} \rangle$

lemma *Image-singleton*: $r \circ \{a\} = \{b. (a, b) : r\}$
 $\langle \text{proof} \rangle$

lemma *Image-singleton-iff [iff]*: $(b : r \circ \{a\}) = ((a, b) : r)$
 $\langle \text{proof} \rangle$

lemma *ImageI [intro, noatp]*: $(a, b) : r \implies a : A \implies b : r \circ A$
 $\langle \text{proof} \rangle$

lemma *ImageE [elim!]*:
 $b : r \circ A \implies (!x. (x, b) : r \implies x : A \implies P) \implies P$

$\langle proof \rangle$

lemma *rev-ImageI*: $a : A \implies (a, b) : r \implies b : r \text{ “ } A$

— This version’s more effective when we already have the required a

$\langle proof \rangle$

lemma *Image-empty* [simp]: $R \text{ “ } \{\} = \{\}$

$\langle proof \rangle$

lemma *Image-Id* [simp]: $Id \text{ “ } A = A$

$\langle proof \rangle$

lemma *Image-Id-on* [simp]: $Id\text{-on } A \text{ “ } B = A \cap B$

$\langle proof \rangle$

lemma *Image-Int-subset*: $R \text{ “ } (A \cap B) \subseteq R \text{ “ } A \cap R \text{ “ } B$

$\langle proof \rangle$

lemma *Image-Int-eq*:

single-valued (*converse* R) $\implies R \text{ “ } (A \cap B) = R \text{ “ } A \cap R \text{ “ } B$

$\langle proof \rangle$

lemma *Image-Un*: $R \text{ “ } (A \cup B) = R \text{ “ } A \cup R \text{ “ } B$

$\langle proof \rangle$

lemma *Un-Image*: $(R \cup S) \text{ “ } A = R \text{ “ } A \cup S \text{ “ } A$

$\langle proof \rangle$

lemma *Image-subset*: $r \subseteq A \times B \implies r \text{ “ } C \subseteq B$

$\langle proof \rangle$

lemma *Image-eq-UN*: $r \text{ “ } B = (\bigcup y \in B. r \text{ “ } \{y\})$

— NOT suitable for rewriting

$\langle proof \rangle$

lemma *Image-mono*: $r' \subseteq r \implies A' \subseteq A \implies (r' \text{ “ } A') \subseteq (r \text{ “ } A)$

$\langle proof \rangle$

lemma *Image-UN*: $(r \text{ “ } (UNION A B)) = (\bigcup x \in A. r \text{ “ } (B x))$

$\langle proof \rangle$

lemma *Image-INT-subset*: $(r \text{ “ } INTER A B) \subseteq (\bigcap x \in A. r \text{ “ } (B x))$

$\langle proof \rangle$

Converse inclusion requires some assumptions

lemma *Image-INT-eq*:

$[[single\text{-valued } (r^{-1}); A \neq \{\}]] \implies r \text{ “ } INTER A B = (\bigcap x \in A. r \text{ “ } B x)$

$\langle proof \rangle$

lemma *Image-subset-eq*: $(r \text{“} A \subseteq B \text{”) = } (A \subseteq - ((r \hat{-} 1) \text{“} (-B)))$
 $\langle \text{proof} \rangle$

18.16 Single valued relations

lemma *single-valuedI*:

$ALL\ x\ y.\ (x,y):r \dashrightarrow (ALL\ z.\ (x,z):r \dashrightarrow y=z) \implies \text{single-valued } r$
 $\langle \text{proof} \rangle$

lemma *single-valuedD*:

$\text{single-valued } r \implies (x, y) : r \implies (x, z) : r \implies y = z$
 $\langle \text{proof} \rangle$

lemma *single-valued-rel-comp*:

$\text{single-valued } r \implies \text{single-valued } s \implies \text{single-valued } (r \circ s)$
 $\langle \text{proof} \rangle$

lemma *single-valued-subset*:

$r \subseteq s \implies \text{single-valued } s \implies \text{single-valued } r$
 $\langle \text{proof} \rangle$

lemma *single-valued-Id* [simp]: *single-valued Id*

$\langle \text{proof} \rangle$

lemma *single-valued-Id-on* [simp]: *single-valued (Id-on A)*

$\langle \text{proof} \rangle$

18.17 Graphs given by Collect

lemma *Domain-Collect-split* [simp]: $\text{Domain}\{(x,y). P\ x\ y\} = \{x. EX\ y. P\ x\ y\}$

$\langle \text{proof} \rangle$

lemma *Range-Collect-split* [simp]: $\text{Range}\{(x,y). P\ x\ y\} = \{y. EX\ x. P\ x\ y\}$

$\langle \text{proof} \rangle$

lemma *Image-Collect-split* [simp]: $\{(x,y). P\ x\ y\} \text{“} A = \{y. EX\ x:A. P\ x\ y\}$

$\langle \text{proof} \rangle$

18.18 Inverse image

lemma *sym-inv-image*: $\text{sym } r \implies \text{sym } (\text{inv-image } r\ f)$

$\langle \text{proof} \rangle$

lemma *trans-inv-image*: $\text{trans } r \implies \text{trans } (\text{inv-image } r\ f)$

$\langle \text{proof} \rangle$

18.19 Finiteness

lemma *finite-converse* [iff]: $\text{finite } (r \hat{-} 1) = \text{finite } r$

$\langle \text{proof} \rangle$

Finiteness of transitive closure (Thanks to Sidi Ehmety)

lemma *finite-Field*: *finite* $r \implies \text{finite } (\text{Field } r)$

— A finite relation has a finite field (= *domain* \cup *range*).

$\langle \text{proof} \rangle$

18.20 Version of *lfp-induct* for binary relations

lemmas *lfp-induct2* =

lfp-induct-set [*of* (*a*, *b*), *split-format* (*complete*)]

end

19 Predicate: Predicates as relations and enumerations

theory *Predicate*

imports *Inductive Relation*

begin

notation

inf (**infixl** \sqcap 70) **and**

sup (**infixl** \sqcup 65) **and**

Inf (\bigcap - [900] 900) **and**

Sup (\bigcup - [900] 900) **and**

top (\top) **and**

bot (\perp)

19.1 Predicates as (complete) lattices

19.1.1 $op \sqcup$ on *bool*

lemma *sup-boolI1*:

$P \implies P \sqcup Q$

$\langle \text{proof} \rangle$

lemma *sup-boolI2*:

$Q \implies P \sqcup Q$

$\langle \text{proof} \rangle$

lemma *sup-boolE*:

$P \sqcup Q \implies (P \implies R) \implies (Q \implies R) \implies R$

$\langle \text{proof} \rangle$

19.1.2 Equality and Subsets

lemma *pred-equals-eq*: $((\lambda x. x \in R) = (\lambda x. x \in S)) = (R = S)$

$\langle \text{proof} \rangle$

lemma *pred-equals-eq2* [*pred-set-conv*]: $((\lambda x y. (x, y) \in R) = (\lambda x y. (x, y) \in S)) = (R = S)$
 $\langle proof \rangle$

lemma *pred-subset-eq*: $((\lambda x. x \in R) \leq (\lambda x. x \in S)) = (R \leq S)$
 $\langle proof \rangle$

lemma *pred-subset-eq2* [*pred-set-conv*]: $((\lambda x y. (x, y) \in R) \leq (\lambda x y. (x, y) \in S)) = (R \leq S)$
 $\langle proof \rangle$

19.1.3 Top and bottom elements

lemma *top1I* [*intro!*]: $top\ x$
 $\langle proof \rangle$

lemma *top2I* [*intro!*]: $top\ x\ y$
 $\langle proof \rangle$

lemma *bot1E* [*elim!*]: $bot\ x \implies P$
 $\langle proof \rangle$

lemma *bot2E* [*elim!*]: $bot\ x\ y \implies P$
 $\langle proof \rangle$

19.1.4 The empty set

lemma *bot-empty-eq*: $bot = (\lambda x. x \in \{\})$
 $\langle proof \rangle$

lemma *bot-empty-eq2*: $bot = (\lambda x y. (x, y) \in \{\})$
 $\langle proof \rangle$

19.1.5 Binary union

lemma *sup1-iff* [*simp*]: $sup\ A\ B\ x \longleftrightarrow A\ x \mid B\ x$
 $\langle proof \rangle$

lemma *sup2-iff* [*simp*]: $sup\ A\ B\ x\ y \longleftrightarrow A\ x\ y \mid B\ x\ y$
 $\langle proof \rangle$

lemma *sup-Un-eq* [*pred-set-conv*]: $sup\ (\lambda x. x \in R)\ (\lambda x. x \in S) = (\lambda x. x \in R \cup S)$
 $\langle proof \rangle$

lemma *sup-Un-eq2* [*pred-set-conv*]: $sup\ (\lambda x y. (x, y) \in R)\ (\lambda x y. (x, y) \in S) = (\lambda x y. (x, y) \in R \cup S)$
 $\langle proof \rangle$

lemma *sup1I1* [*elim?*]: $A\ x \implies sup\ A\ B\ x$

$\langle \text{proof} \rangle$

lemma *sup2I1* [*elim?*]: $A\ x\ y \implies \text{sup}\ A\ B\ x\ y$
 $\langle \text{proof} \rangle$

lemma *sup1I2* [*elim?*]: $B\ x \implies \text{sup}\ A\ B\ x$
 $\langle \text{proof} \rangle$

lemma *sup2I2* [*elim?*]: $B\ x\ y \implies \text{sup}\ A\ B\ x\ y$
 $\langle \text{proof} \rangle$

Classical introduction rule: no commitment to A vs B .

lemma *sup1CI* [*intro!*]: $(\sim B\ x \implies A\ x) \implies \text{sup}\ A\ B\ x$
 $\langle \text{proof} \rangle$

lemma *sup2CI* [*intro!*]: $(\sim B\ x\ y \implies A\ x\ y) \implies \text{sup}\ A\ B\ x\ y$
 $\langle \text{proof} \rangle$

lemma *sup1E* [*elim!*]: $\text{sup}\ A\ B\ x \implies (A\ x \implies P) \implies (B\ x \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *sup2E* [*elim!*]: $\text{sup}\ A\ B\ x\ y \implies (A\ x\ y \implies P) \implies (B\ x\ y \implies P) \implies P$
 $\langle \text{proof} \rangle$

19.1.6 Binary intersection

lemma *inf1-iff* [*simp*]: $\text{inf}\ A\ B\ x \longleftrightarrow A\ x \wedge B\ x$
 $\langle \text{proof} \rangle$

lemma *inf2-iff* [*simp*]: $\text{inf}\ A\ B\ x\ y \longleftrightarrow A\ x\ y \wedge B\ x\ y$
 $\langle \text{proof} \rangle$

lemma *inf-Int-eq* [*pred-set-conv*]: $\text{inf}\ (\lambda x. x \in R)\ (\lambda x. x \in S) = (\lambda x. x \in R \cap S)$
 $\langle \text{proof} \rangle$

lemma *inf-Int-eq2* [*pred-set-conv*]: $\text{inf}\ (\lambda x\ y. (x, y) \in R)\ (\lambda x\ y. (x, y) \in S) = (\lambda x\ y. (x, y) \in R \cap S)$
 $\langle \text{proof} \rangle$

lemma *inf1I* [*intro!*]: $A\ x \implies B\ x \implies \text{inf}\ A\ B\ x$
 $\langle \text{proof} \rangle$

lemma *inf2I* [*intro!*]: $A\ x\ y \implies B\ x\ y \implies \text{inf}\ A\ B\ x\ y$
 $\langle \text{proof} \rangle$

lemma *inf1D1*: $\text{inf}\ A\ B\ x \implies A\ x$
 $\langle \text{proof} \rangle$

lemma *inf2D1*: $\inf A B x y \implies A x y$
 $\langle \text{proof} \rangle$

lemma *inf1D2*: $\inf A B x \implies B x$
 $\langle \text{proof} \rangle$

lemma *inf2D2*: $\inf A B x y \implies B x y$
 $\langle \text{proof} \rangle$

lemma *inf1E* [*elim!*]: $\inf A B x \implies (A x \implies B x \implies P) \implies P$
 $\langle \text{proof} \rangle$

lemma *inf2E* [*elim!*]: $\inf A B x y \implies (A x y \implies B x y \implies P) \implies P$
 $\langle \text{proof} \rangle$

19.1.7 Unions of families

lemma *SUP1-iff* [*simp*]: $(\text{SUP } x:A. B x) b = (\text{EX } x:A. B x b)$
 $\langle \text{proof} \rangle$

lemma *SUP2-iff* [*simp*]: $(\text{SUP } x:A. B x) b c = (\text{EX } x:A. B x b c)$
 $\langle \text{proof} \rangle$

lemma *SUP1-I* [*intro*]: $a : A \implies B a b \implies (\text{SUP } x:A. B x) b$
 $\langle \text{proof} \rangle$

lemma *SUP2-I* [*intro*]: $a : A \implies B a b c \implies (\text{SUP } x:A. B x) b c$
 $\langle \text{proof} \rangle$

lemma *SUP1-E* [*elim!*]: $(\text{SUP } x:A. B x) b \implies (!x. x : A \implies B x b \implies R) \implies R$
 $\langle \text{proof} \rangle$

lemma *SUP2-E* [*elim!*]: $(\text{SUP } x:A. B x) b c \implies (!x. x : A \implies B x b c \implies R) \implies R$
 $\langle \text{proof} \rangle$

lemma *SUP-UN-eq*: $(\text{SUP } i. (\lambda x. x \in r i)) = (\lambda x. x \in (\text{UN } i. r i))$
 $\langle \text{proof} \rangle$

lemma *SUP-UN-eq2*: $(\text{SUP } i. (\lambda x y. (x, y) \in r i)) = (\lambda x y. (x, y) \in (\text{UN } i. r i))$
 $\langle \text{proof} \rangle$

19.1.8 Intersections of families

lemma *INF1-iff* [*simp*]: $(\text{INF } x:A. B x) b = (\text{ALL } x:A. B x b)$
 $\langle \text{proof} \rangle$

lemma *INF2-iff* [*simp*]: $(\text{INF } x:A. B x) b c = (\text{ALL } x:A. B x b c)$

$\langle proof \rangle$

lemma *INF1-I* [*intro!*]: $(!!x. x : A ==> B\ x\ b) ==> (INF\ x:A. B\ x)\ b$
 $\langle proof \rangle$

lemma *INF2-I* [*intro!*]: $(!!x. x : A ==> B\ x\ b\ c) ==> (INF\ x:A. B\ x)\ b\ c$
 $\langle proof \rangle$

lemma *INF1-D* [*elim*]: $(INF\ x:A. B\ x)\ b ==> a : A ==> B\ a\ b$
 $\langle proof \rangle$

lemma *INF2-D* [*elim*]: $(INF\ x:A. B\ x)\ b\ c ==> a : A ==> B\ a\ b\ c$
 $\langle proof \rangle$

lemma *INF1-E* [*elim*]: $(INF\ x:A. B\ x)\ b ==> (B\ a\ b ==> R) ==> (a \sim : A ==> R) ==> R$
 $\langle proof \rangle$

lemma *INF2-E* [*elim*]: $(INF\ x:A. B\ x)\ b\ c ==> (B\ a\ b\ c ==> R) ==> (a \sim : A ==> R) ==> R$
 $\langle proof \rangle$

lemma *INF-INT-eq*: $(INF\ i. (\lambda x. x \in r\ i)) = (\lambda x. x \in (INT\ i. r\ i))$
 $\langle proof \rangle$

lemma *INF-INT-eq2*: $(INF\ i. (\lambda x\ y. (x, y) \in r\ i)) = (\lambda x\ y. (x, y) \in (INT\ i. r\ i))$
 $\langle proof \rangle$

19.2 Predicates as relations

19.2.1 Composition

inductive

pred-comp :: $['b ==> 'c ==> bool, 'a ==> 'b ==> bool] ==> 'a ==> 'c ==> bool$
 (infixr *OO* 75)

for *r* :: $'b ==> 'c ==> bool$ **and** *s* :: $'a ==> 'b ==> bool$

where

pred-compI [*intro*]: $s\ a\ b ==> r\ b\ c ==> (r\ OO\ s)\ a\ c$

inductive-cases *pred-compE* [*elim!*]: $(r\ OO\ s)\ a\ c$

lemma *pred-comp-rel-comp-eq* [*pred-set-conv*]:

$((\lambda x\ y. (x, y) \in r)\ OO\ (\lambda x\ y. (x, y) \in s)) = (\lambda x\ y. (x, y) \in r\ O\ s)$
 $\langle proof \rangle$

19.2.2 Converse

inductive

conversep :: $('a ==> 'b ==> bool) ==> 'b ==> 'a ==> bool$
 ((\sim ^ $--1$) [*1000*] *1000*)

for $r :: 'a \Rightarrow 'b \Rightarrow \text{bool}$
where
 $\text{conversepI}: r\ a\ b \implies r^{\hat{---}1}\ b\ a$

notation ($x\text{symbols}$)
 $\text{conversep}\ ((-^{1-1})\ [1000]\ 1000)$

lemma conversepD :
assumes $ab: r^{\hat{---}1}\ a\ b$
shows $r\ b\ a$ $\langle\text{proof}\rangle$

lemma conversep-iff [iff]: $r^{\hat{---}1}\ a\ b = r\ b\ a$
 $\langle\text{proof}\rangle$

lemma $\text{conversep-converse-eq}$ [pred-set-conv]:
 $(\lambda x\ y. (x, y) \in r)^{\hat{---}1} = (\lambda x\ y. (x, y) \in r^{\hat{---}1})$
 $\langle\text{proof}\rangle$

lemma $\text{conversep-conversep}$ [simp]: $(r^{\hat{---}1})^{\hat{---}1} = r$
 $\langle\text{proof}\rangle$

lemma $\text{converse-pred-comp}$: $(r\ OO\ s)^{\hat{---}1} = s^{\hat{---}1}\ OO\ r^{\hat{---}1}$
 $\langle\text{proof}\rangle$

lemma converse-meet : $(\inf\ r\ s)^{\hat{---}1} = \inf\ r^{\hat{---}1}\ s^{\hat{---}1}$
 $\langle\text{proof}\rangle$

lemma converse-join : $(\sup\ r\ s)^{\hat{---}1} = \sup\ r^{\hat{---}1}\ s^{\hat{---}1}$
 $\langle\text{proof}\rangle$

lemma conversep-noteq [simp]: $(op\ \sim) ^{\hat{---}1} = op\ \sim$
 $\langle\text{proof}\rangle$

lemma conversep-eq [simp]: $(op\ =) ^{\hat{---}1} = op\ =$
 $\langle\text{proof}\rangle$

19.2.3 Domain

inductive
 $\text{DomainP} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow \text{bool}$
for $r :: 'a \Rightarrow 'b \Rightarrow \text{bool}$
where
 $\text{DomainPI}\ [\text{intro}]: r\ a\ b \implies \text{DomainP}\ r\ a$

inductive-cases $\text{DomainPE}\ [\text{elim!}]: \text{DomainP}\ r\ a$

lemma DomainP-Domain-eq [pred-set-conv]: $\text{DomainP}\ (\lambda x\ y. (x, y) \in r) = (\lambda x. x \in \text{Domain}\ r)$
 $\langle\text{proof}\rangle$

19.2.4 Range

inductive

$\text{RangeP} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'b \Rightarrow \text{bool}$

for $r :: 'a \Rightarrow 'b \Rightarrow \text{bool}$

where

$\text{RangePI} \text{ [intro]: } r \ a \ b \ ==> \text{RangeP } r \ b$

inductive-cases $\text{RangePE} \text{ [elim!]: } \text{RangeP } r \ b$

lemma $\text{RangeP-Range-eq} \text{ [pred-set-conv]: } \text{RangeP } (\lambda x \ y. (x, y) \in r) = (\lambda x. x \in \text{Range } r)$
 $\langle \text{proof} \rangle$

19.2.5 Inverse image

definition

$\text{inv-imagep} :: ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ **where**
 $\text{inv-imagep } r \ f == \%x \ y. r \ (f \ x) \ (f \ y)$

lemma $\text{[pred-set-conv]: } \text{inv-imagep } (\lambda x \ y. (x, y) \in r) \ f = (\lambda x \ y. (x, y) \in \text{inv-image } r \ f)$
 $\langle \text{proof} \rangle$

lemma $\text{in-inv-imagep} \text{ [simp]: } \text{inv-imagep } r \ f \ x \ y = r \ (f \ x) \ (f \ y)$
 $\langle \text{proof} \rangle$

19.2.6 Powerset

definition $\text{Powp} :: ('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $\text{Powp } A == \lambda B. \forall x \in B. A \ x$

lemma $\text{Powp-Pow-eq} \text{ [pred-set-conv]: } \text{Powp } (\lambda x. x \in A) = (\lambda x. x \in \text{Pow } A)$
 $\langle \text{proof} \rangle$

lemmas $\text{Powp-mono} \text{ [mono]} = \text{Pow-mono} \text{ [to-pred pred-subset-eq]}$

19.2.7 Properties of relations

abbreviation $\text{antisymP} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**
 $\text{antisymP } r == \text{antisym } \{(x, y). r \ x \ y\}$

abbreviation $\text{transP} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**
 $\text{transP } r == \text{trans } \{(x, y). r \ x \ y\}$

abbreviation $\text{single-valuedP} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**
 $\text{single-valuedP } r == \text{single-valued } \{(x, y). r \ x \ y\}$

19.3 Predicates as enumerations

19.3.1 The type of predicate enumerations (a monad)

datatype $'a \text{ pred} = \text{Pred } 'a \Rightarrow \text{bool}$

primrec $\text{eval} :: 'a \text{ pred} \Rightarrow 'a \Rightarrow \text{bool}$ **where**
 $\text{eval-pred}: \text{eval } (\text{Pred } f) = f$

lemma $\text{Pred-eval [simp]}:$
 $\text{Pred } (\text{eval } x) = x$
 $\langle \text{proof} \rangle$

lemma $\text{eval-inject}: \text{eval } x = \text{eval } y \longleftrightarrow x = y$
 $\langle \text{proof} \rangle$

definition $\text{single} :: 'a \Rightarrow 'a \text{ pred}$ **where**
 $\text{single } x = \text{Pred } ((\text{op } =) x)$

definition $\text{bind} :: 'a \text{ pred} \Rightarrow ('a \Rightarrow 'b \text{ pred}) \Rightarrow 'b \text{ pred}$ (**infixl** $\gg=$ 70) **where**
 $P \gg= f = \text{Pred } (\lambda x. (\exists y. \text{eval } P y \wedge \text{eval } (f y) x))$

instantiation $\text{pred} :: (\text{type}) \text{ complete-lattice}$
begin

definition
 $P \leq Q \longleftrightarrow \text{eval } P \leq \text{eval } Q$

definition
 $P < Q \longleftrightarrow \text{eval } P < \text{eval } Q$

definition
 $\perp = \text{Pred } \perp$

definition
 $\top = \text{Pred } \top$

definition
 $P \sqcap Q = \text{Pred } (\text{eval } P \sqcap \text{eval } Q)$

definition
 $P \sqcup Q = \text{Pred } (\text{eval } P \sqcup \text{eval } Q)$

definition
 $\bigsqcap A = \text{Pred } (\text{INFI } A \text{ eval})$

definition
 $\bigsqcup A = \text{Pred } (\text{SUPR } A \text{ eval})$

instance $\langle \text{proof} \rangle$

end

lemma *bind-bind*:

$$(P \gg= Q) \gg= R = P \gg= (\lambda x. Q\ x \gg= R)$$

<proof>

lemma *bind-single*:

$$P \gg= \text{single} = P$$

<proof>

lemma *single-bind*:

$$\text{single}\ x \gg= P = P\ x$$

<proof>

lemma *bottom-bind*:

$$\perp \gg= P = \perp$$

<proof>

lemma *sup-bind*:

$$(P \sqcup Q) \gg= R = P \gg= R \sqcup Q \gg= R$$

<proof>

lemma *Sup-bind*: $(\sqcup A \gg= f) = \sqcup ((\lambda x. x \gg= f)\ 'A)$

<proof>

lemma *pred-iffI*:

assumes $\bigwedge x. \text{eval}\ A\ x \implies \text{eval}\ B\ x$

and $\bigwedge x. \text{eval}\ B\ x \implies \text{eval}\ A\ x$

shows $A = B$

<proof>

lemma *singleI*: $\text{eval}\ (\text{single}\ x)\ x$

<proof>

lemma *singleI-unit*: $\text{eval}\ (\text{single}\ ())\ x$

<proof>

lemma *singleE*: $\text{eval}\ (\text{single}\ x)\ y \implies (y = x \implies P) \implies P$

<proof>

lemma *singleE'*: $\text{eval}\ (\text{single}\ x)\ y \implies (x = y \implies P) \implies P$

<proof>

lemma *bindI*: $\text{eval}\ P\ x \implies \text{eval}\ (Q\ x)\ y \implies \text{eval}\ (P \gg= Q)\ y$

<proof>

lemma *bindE*: $\text{eval}\ (R \gg= Q)\ y \implies (\bigwedge x. \text{eval}\ R\ x \implies \text{eval}\ (Q\ x)\ y \implies P) \implies P$

$\langle \text{proof} \rangle$

lemma *botE*: $\text{eval } \perp x \implies P$

$\langle \text{proof} \rangle$

lemma *supI1*: $\text{eval } A x \implies \text{eval } (A \sqcup B) x$

$\langle \text{proof} \rangle$

lemma *supI2*: $\text{eval } B x \implies \text{eval } (A \sqcup B) x$

$\langle \text{proof} \rangle$

lemma *supE*: $\text{eval } (A \sqcup B) x \implies (\text{eval } A x \implies P) \implies (\text{eval } B x \implies P) \implies P$

$\langle \text{proof} \rangle$

19.3.2 Derived operations

definition *if-pred* :: $\text{bool} \Rightarrow \text{unit pred}$ **where**

if-pred-eq: $\text{if-pred } b = (\text{if } b \text{ then single } () \text{ else } \perp)$

definition *not-pred* :: $\text{unit pred} \Rightarrow \text{unit pred}$ **where**

not-pred-eq: $\text{not-pred } P = (\text{if eval } P () \text{ then } \perp \text{ else single } ())$

lemma *if-predI*: $P \implies \text{eval } (\text{if-pred } P) ()$

$\langle \text{proof} \rangle$

lemma *if-predE*: $\text{eval } (\text{if-pred } b) x \implies (b \implies x = () \implies P) \implies P$

$\langle \text{proof} \rangle$

lemma *not-predI*: $\neg P \implies \text{eval } (\text{not-pred } (\text{Pred } (\lambda u. P))) ()$

$\langle \text{proof} \rangle$

lemma *not-predI'*: $\neg \text{eval } P () \implies \text{eval } (\text{not-pred } P) ()$

$\langle \text{proof} \rangle$

lemma *not-predE*: $\text{eval } (\text{not-pred } (\text{Pred } (\lambda u. P))) x \implies (\neg P \implies \text{thesis}) \implies \text{thesis}$

$\langle \text{proof} \rangle$

lemma *not-predE'*: $\text{eval } (\text{not-pred } P) x \implies (\neg \text{eval } P x \implies \text{thesis}) \implies \text{thesis}$

$\langle \text{proof} \rangle$

19.3.3 Implementation

datatype *'a seq* = *Empty* | *Insert 'a 'a pred* | *Join 'a pred 'a seq*

primrec *pred-of-seq* :: *'a seq* \Rightarrow *'a pred* **where**

pred-of-seq Empty = \perp

| *pred-of-seq (Insert x P)* = $\text{single } x \sqcup P$

| *pred-of-seq (Join P xq)* = $P \sqcup \text{pred-of-seq } xq$

definition $Seq :: (unit \Rightarrow 'a\ seq) \Rightarrow 'a\ pred$ **where**
 $Seq\ f = pred\text{-of}\text{-seq}\ (f\ ())$

code-datatype Seq

primrec $member :: 'a\ seq \Rightarrow 'a \Rightarrow bool$ **where**
 $member\ Empty\ x \longleftrightarrow False$
 $| member\ (Insert\ y\ P)\ x \longleftrightarrow x = y \vee eval\ P\ x$
 $| member\ (Join\ P\ xq)\ x \longleftrightarrow eval\ P\ x \vee member\ xq\ x$

lemma $eval\text{-}member$:
 $member\ xq = eval\ (pred\text{-of}\text{-seq}\ xq)$
 $\langle proof \rangle$

lemma $eval\text{-}code\ [code]$: $eval\ (Seq\ f) = member\ (f\ ())$
 $\langle proof \rangle$

lemma $single\text{-}code\ [code]$:
 $single\ x = Seq\ (\lambda u. Insert\ x\ \bot)$
 $\langle proof \rangle$

primrec $apply :: ('a \Rightarrow 'b\ Predicate.pred) \Rightarrow 'a\ seq \Rightarrow 'b\ seq$ **where**
 $apply\ f\ Empty = Empty$
 $| apply\ f\ (Insert\ x\ P) = Join\ (f\ x)\ (Join\ (P \gg= f)\ Empty)$
 $| apply\ f\ (Join\ P\ xq) = Join\ (P \gg= f)\ (apply\ f\ xq)$

lemma $apply\text{-}bind$:
 $pred\text{-of}\text{-seq}\ (apply\ f\ xq) = pred\text{-of}\text{-seq}\ xq \gg= f$
 $\langle proof \rangle$

lemma $bind\text{-}code\ [code]$:
 $Seq\ g \gg= f = Seq\ (\lambda u. apply\ f\ (g\ ()))$
 $\langle proof \rangle$

lemma $bot\text{-}set\text{-}code\ [code]$:
 $\bot = Seq\ (\lambda u. Empty)$
 $\langle proof \rangle$

primrec $adjunct :: 'a\ pred \Rightarrow 'a\ seq \Rightarrow 'a\ seq$ **where**
 $adjunct\ P\ Empty = Join\ P\ Empty$
 $| adjunct\ P\ (Insert\ x\ Q) = Insert\ x\ (Q \sqcup P)$
 $| adjunct\ P\ (Join\ Q\ xq) = Join\ Q\ (adjunct\ P\ xq)$

lemma $adjunct\text{-}sup$:
 $pred\text{-of}\text{-seq}\ (adjunct\ P\ xq) = P \sqcup pred\text{-of}\text{-seq}\ xq$
 $\langle proof \rangle$

lemma $sup\text{-}code\ [code]$:
 $Seq\ f \sqcup Seq\ g = Seq\ (\lambda u. case\ f\ ())$

```

    of Empty  $\Rightarrow$  g ()
    | Insert x P  $\Rightarrow$  Insert x (P  $\sqcup$  Seq g)
    | Join P xq  $\Rightarrow$  adjunct (Seq g) (Join P xq))
  <proof>

```

```

primrec contained :: 'a seq  $\Rightarrow$  'a pred  $\Rightarrow$  bool where
  contained Empty Q  $\longleftrightarrow$  True
  | contained (Insert x P) Q  $\longleftrightarrow$  eval Q x  $\wedge$  P  $\leq$  Q
  | contained (Join P xq) Q  $\longleftrightarrow$  P  $\leq$  Q  $\wedge$  contained xq Q

```

```

lemma single-less-eq-eval:
  single x  $\leq$  P  $\longleftrightarrow$  eval P x
  <proof>

```

```

lemma contained-less-eq:
  contained xq Q  $\longleftrightarrow$  pred-of-seq xq  $\leq$  Q
  <proof>

```

```

lemma less-eq-pred-code [code]:
  Seq f  $\leq$  Q = (case f ()
    of Empty  $\Rightarrow$  True
    | Insert x P  $\Rightarrow$  eval Q x  $\wedge$  P  $\leq$  Q
    | Join P xq  $\Rightarrow$  P  $\leq$  Q  $\wedge$  contained xq Q)
  <proof>

```

```

lemma eq-pred-code [code]:
  fixes P Q :: 'a::eq pred
  shows eq-class.eq P Q  $\longleftrightarrow$  P  $\leq$  Q  $\wedge$  Q  $\leq$  P
  <proof>

```

```

lemma [code]:
  pred-case f P = f (eval P)
  <proof>

```

```

lemma [code]:
  pred-rec f P = f (eval P)
  <proof>

```

```

no-notation
  inf (infixl  $\sqcap$  70) and
  sup (infixl  $\sqcup$  65) and
  Inf ( $\sqcap$  - [900] 900) and
  Sup ( $\sqcup$  - [900] 900) and
  top ( $\top$ ) and
  bot ( $\perp$ ) and
  bind (infixl  $\gg=$  70)

```

```

hide (open) type pred seq
hide (open) const Pred eval single bind if-pred not-pred

```

Empty Insert Join Seq member pred-of-seq apply adjunct

end

20 Transitive-Closure: Reflexive and Transitive closure of a relation

theory *Transitive-Closure*
imports *Predicate*
uses *~~/src/Provers/trancl.ML*
begin

rtrancl is reflexive/transitive closure, *trancl* is transitive closure, *reflcl* is reflexive closure.

These postfix operators have *maximum priority*, forcing their operands to be atomic.

inductive-set

rtrancl :: ('a × 'a) set ⇒ ('a × 'a) set ((-^*) [1000] 999)
for *r* :: ('a × 'a) set

where

rtrancl-refl [intro!, Pure.intro!, simp]: (a, a) : r^*
 | *rtrancl-into-rtrancl* [Pure.intro]: (a, b) : r^* ==> (b, c) : r ==> (a, c) : r^*

inductive-set

trancl :: ('a × 'a) set ⇒ ('a × 'a) set ((-^+) [1000] 999)
for *r* :: ('a × 'a) set

where

r-into-trancl [intro, Pure.intro]: (a, b) : r ==> (a, b) : r^+
 | *trancl-into-trancl* [Pure.intro]: (a, b) : r^+ ==> (b, c) : r ==> (a, c) : r^+

notation

rtranclp ((-^**) [1000] 1000) **and**
tranclp ((-^++) [1000] 1000)

abbreviation

reflclp :: ('a ==> 'a ==> bool) ==> 'a ==> 'a ==> bool ((-^==) [1000] 1000)

where

r^== == sup *r op* =

abbreviation

reflcl :: ('a × 'a) set ==> ('a × 'a) set ((-^=) [1000] 999) **where**
r^= == *r* ∪ *Id*

notation (*xsymbols*)

rtranclp ((-**) [1000] 1000) **and**
tranclp ((-++) [1000] 1000) **and**
reflclp ((-==) [1000] 1000) **and**

rtrancl $((-^*) [1000] 999)$ and
trancl $((-^+) [1000] 999)$ and
reflcl $((-^=) [1000] 999)$

notation (*HTML output*)

rtranclp $((-^{**}) [1000] 1000)$ and
tranclp $((-^{++}) [1000] 1000)$ and
reflclp $((-^{==}) [1000] 1000)$ and
rtrancl $((-^*) [1000] 999)$ and
trancl $((-^+) [1000] 999)$ and
reflcl $((-^=) [1000] 999)$

20.1 Reflexive closure

lemma *refl-reflcl[simp]*: $\text{refl}(r^=)$
 $\langle \text{proof} \rangle$

lemma *antisym-reflcl[simp]*: $\text{antisym}(r^=) = \text{antisym } r$
 $\langle \text{proof} \rangle$

lemma *trans-reflclI[simp]*: $\text{trans } r \implies \text{trans}(r^=)$
 $\langle \text{proof} \rangle$

20.2 Reflexive-transitive closure

lemma *reflcl-set-eq [pred-set-conv]*: $(\sup (\lambda x y. (x, y) \in r) \text{ op } =) = (\lambda x y. (x, y) \in r \text{ Un Id})$
 $\langle \text{proof} \rangle$

lemma *r-into-rtrancl [intro]*: $!!p. p \in r \implies p \in r^{^*}$
 — *rtrancl* of *r* contains *r*
 $\langle \text{proof} \rangle$

lemma *r-into-rtranclp [intro]*: $r \ x \ y \implies r^{^{**}} \ x \ y$
 — *rtrancl* of *r* contains *r*
 $\langle \text{proof} \rangle$

lemma *rtranclp-mono*: $r \leq s \implies r^{^{**}} \leq s^{^{**}}$
 — monotonicity of *rtrancl*
 $\langle \text{proof} \rangle$

lemmas *rtrancl-mono* = *rtranclp-mono* [*to-set*]

theorem *rtranclp-induct [consumes 1, case-names base step, induct set: rtranclp]*:
 assumes *a*: $r^{^{**}} \ a \ b$
 and cases: $P \ a \ !!y \ z. [\![\ r^{^{**}} \ a \ y; \ r \ y \ z; \ P \ y \]\!] \implies P \ z$
 shows $P \ b$
 $\langle \text{proof} \rangle$

lemmas *rtrancl-induct [induct set: rtrancl]* = *rtranclp-induct [to-set]*

lemmas *rtranclp-induct2* =
rtranclp-induct[*of* - (*ax,ay*) (*bx,by*), *split-rule*,
consumes 1, *case-names refl step*]

lemmas *rtrancl-induct2* =
rtrancl-induct[*of* (*ax,ay*) (*bx,by*), *split-format (complete)*,
consumes 1, *case-names refl step*]

lemma *refl-rtrancl*: *refl* (r^*)
 $\langle proof \rangle$

Transitivity of transitive closure.

lemma *trans-rtrancl*: *trans* (r^*)
 $\langle proof \rangle$

lemmas *rtrancl-trans* = *trans-rtrancl* [*THEN transD*, *standard*]

lemma *rtranclp-trans*:
assumes *xy*: $r^{**} x y$
and *yz*: $r^{**} y z$
shows $r^{**} x z$ $\langle proof \rangle$

lemma *rtranclE* [*cases set: rtrancl*]:
assumes *major*: (*a*::'*a*, *b*) : r^*
obtains
 (*base*) $a = b$
 | (*step*) *y* **where** (*a*, *y*) : r^* **and** (*y*, *b*) : r
 — elimination of *rtrancl* – by induction on a special formula
 $\langle proof \rangle$

lemma *rtrancl-Int-subset*: [$Id \subseteq s$; $r \circ (r^* \cap s) \subseteq s$] $\implies r^* \subseteq s$
 $\langle proof \rangle$

lemma *converse-rtranclp-into-rtranclp*:
 $r a b \implies r^{**} b c \implies r^{**} a c$
 $\langle proof \rangle$

lemmas *converse-rtrancl-into-rtrancl* = *converse-rtranclp-into-rtranclp* [*to-set*]

More r^* equations and inclusions.

lemma *rtranclp-idemp* [*simp*]: $(r^{**})^{**} = r^{**}$
 $\langle proof \rangle$

lemmas *rtrancl-idemp* [*simp*] = *rtranclp-idemp* [*to-set*]

lemma *rtrancl-idemp-self-comp* [*simp*]: $R^* \circ R^* = R^*$
 $\langle proof \rangle$

lemma *rtrancl-subset-rtrancl*: $r \subseteq s^{\wedge*} \implies r^{\wedge*} \subseteq s^{\wedge*}$
 ⟨proof⟩

lemma *rtranclp-subset*: $R \leq S \implies S \leq R^{\wedge**} \implies S^{\wedge**} = R^{\wedge**}$
 ⟨proof⟩

lemmas *rtrancl-subset = rtranclp-subset* [to-set]

lemma *rtranclp-sup-rtranclp*: $(\sup (R^{\wedge**}) (S^{\wedge**}))^{\wedge**} = (\sup R S)^{\wedge**}$
 ⟨proof⟩

lemmas *rtrancl-Un-rtrancl = rtranclp-sup-rtranclp* [to-set]

lemma *rtranclp-reflcl* [simp]: $(R^{\wedge==})^{\wedge**} = R^{\wedge**}$
 ⟨proof⟩

lemmas *rtrancl-reflcl* [simp] = *rtranclp-reflcl* [to-set]

lemma *rtrancl-r-diff-Id*: $(r - Id)^{\wedge*} = r^{\wedge*}$
 ⟨proof⟩

lemma *rtranclp-r-diff-Id*: $(\inf r \text{ op } \sim)^{\wedge**} = r^{\wedge**}$
 ⟨proof⟩

theorem *rtranclp-converseD*:
 assumes $r: (r^{\wedge}-1)^{\wedge**} x y$
 shows $r^{\wedge**} y x$
 ⟨proof⟩

lemmas *rtrancl-converseD = rtranclp-converseD* [to-set]

theorem *rtranclp-converseI*:
 assumes $r^{\wedge**} y x$
 shows $(r^{\wedge}-1)^{\wedge**} x y$
 ⟨proof⟩

lemmas *rtrancl-converseI = rtranclp-converseI* [to-set]

lemma *rtrancl-converse*: $(r^{\wedge}-1)^{\wedge*} = (r^{\wedge*})^{\wedge}-1$
 ⟨proof⟩

lemma *sym-rtrancl*: $\text{sym } r \implies \text{sym } (r^{\wedge*})$
 ⟨proof⟩

theorem *converse-rtranclp-induct*[consumes 1]:
 assumes major: $r^{\wedge**} a b$
 and cases: $P b !!y z. [\text{ } r y z; r^{\wedge**} z b; P z] \implies P y$
 shows $P a$
 ⟨proof⟩

lemmas *converse-rtrancl-induct* = *converse-rtranclp-induct* [to-set]

lemmas *converse-rtranclp-induct2* =
converse-rtranclp-induct [of - (ax,ay) (bx,by), split-rule,
 consumes 1, case-names refl step]

lemmas *converse-rtrancl-induct2* =
converse-rtrancl-induct [of (ax,ay) (bx,by), split-format (complete),
 consumes 1, case-names refl step]

lemma *converse-rtranclpE*:
 assumes major: $r^{**} x z$
 and cases: $x=z \implies P$
 !!y. [$r x y$; $r^{**} y z$] $\implies P$
 shows P
 <proof>

lemmas *converse-rtranclE* = *converse-rtranclpE* [to-set]

lemmas *converse-rtranclpE2* = *converse-rtranclpE* [of - (xa,xb) (za,zb), split-rule]

lemmas *converse-rtranclE2* = *converse-rtranclE* [of (xa,xb) (za,zb), split-rule]

lemma *r-comp-rtrancl-eq*: $r \circ r^* = r^* \circ r$
 <proof>

lemma *rtrancl-unfold*: $r^* = Id \cup r \circ r^*$
 <proof>

20.3 Transitive closure

lemma *trancl-mono*: !!p. $p \in r^+ \implies r \subseteq s \implies p \in s^+$
 <proof>

lemma *r-into-trancl'*: !!p. $p : r \implies p : r^+$
 <proof>

Conversions between *trancl* and *rtrancl*.

lemma *tranclp-into-rtranclp*: $r^{++} a b \implies r^{**} a b$
 <proof>

lemmas *trancl-into-rtrancl* = *tranclp-into-rtranclp* [to-set]

lemma *rtranclp-into-tranclp1*: assumes $r: r^{**} a b$
 shows !!c. $r b c \implies r^{++} a c$ <proof>

lemmas *rtrancl-into-trancl1* = *rtranclp-into-tranclp1* [to-set]

lemma *rtrancp-into-trancp2*: $[[\ r\ a\ b;\ r^{\wedge**}\ b\ c\]]\implies r^{\wedge++}\ a\ c$
 — intro rule from *r* and *rtrancp*
 $\langle proof \rangle$

lemmas *rtrancp-into-trancp2* = *rtrancp-into-trancp2* [to-set]

Nice induction rule for *trancp*

lemma *trancp-induct* [consumes 1, case-names base step, induct pred: *trancp*]:
assumes $r^{\wedge++}\ a\ b$
and cases: $!!y.\ r\ a\ y \implies P\ y$
 $!!y\ z.\ r^{\wedge++}\ a\ y \implies r\ y\ z \implies P\ y \implies P\ z$
shows $P\ b$
 $\langle proof \rangle$

lemmas *trancp-induct* [induct set: *trancp*] = *trancp-induct* [to-set]

lemmas *trancp-induct2* =
trancp-induct [of - (*ax,ay*) (*bx,by*), split-rule,
 consumes 1, case-names base step]

lemmas *trancp-induct2* =
trancp-induct [of (*ax,ay*) (*bx,by*), split-format (complete),
 consumes 1, case-names base step]

lemma *trancp-trans-induct*:
assumes *major*: $r^{\wedge++}\ x\ y$
and cases: $!!x\ y.\ r\ x\ y \implies P\ x\ y$
 $!!x\ y\ z.\ [[\ r^{\wedge++}\ x\ y;\ P\ x\ y;\ r^{\wedge++}\ y\ z;\ P\ y\ z\]]\implies P\ x\ z$
shows $P\ x\ y$
 — Another induction rule for *trancp*, incorporating transitivity
 $\langle proof \rangle$

lemmas *trancp-trans-induct* = *trancp-trans-induct* [to-set]

lemma *trancpE* [cases set: *trancp*]:
assumes $(a,\ b) : r^{\wedge++}$
obtains
 (base) $(a,\ b) : r$
 | (step) c **where** $(a,\ c) : r^{\wedge++}$ **and** $(c,\ b) : r$
 $\langle proof \rangle$

lemma *trancp-Int-subset*: $[[\ r \subseteq s;\ r\ O\ (r^{\wedge++} \cap s) \subseteq s\]]\implies r^{\wedge++} \subseteq s$
 $\langle proof \rangle$

lemma *trancp-unfold*: $r^{\wedge++} = r\ Un\ r\ O\ r^{\wedge++}$
 $\langle proof \rangle$

Transitivity of r^+

lemma *trans-trancp* [simp]: $trans\ (r^{\wedge++})$

$\langle \text{proof} \rangle$

lemmas $\text{trancl-trans} = \text{trans-trancl}$ $[\text{THEN } \text{transD}, \text{standard}]$

lemma tranclp-trans :

assumes $xy: r^{++} x y$

and $yz: r^{++} y z$

shows $r^{++} x z$ $\langle \text{proof} \rangle$

lemma trancl-id $[\text{simp}]$: $\text{trans } r \implies r^+ = r$

$\langle \text{proof} \rangle$

lemma $\text{rtranclp-tranclp-tranclp}$:

assumes $r^{**} x y$

shows $!!z. r^{++} y z \implies r^{++} x z$ $\langle \text{proof} \rangle$

lemmas $\text{rtrancl-trancl-trancl} = \text{rtranclp-tranclp-tranclp}$ $[\text{to-set}]$

lemma $\text{tranclp-into-tranclp2}$: $r a b \implies r^{++} b c \implies r^{++} a c$

$\langle \text{proof} \rangle$

lemmas $\text{trancl-into-trancl2} = \text{tranclp-into-tranclp2}$ $[\text{to-set}]$

lemma trancl-insert :

$(\text{insert } (y, x) r)^+ = r^+ \cup \{(a, b). (a, y) \in r^* \wedge (x, b) \in r^*\}$

— primitive recursion for trancl over finite relations

$\langle \text{proof} \rangle$

lemma tranclp-converseI : $(r^{++})^{--1} x y \implies (r^{--1})^{++} x y$

$\langle \text{proof} \rangle$

lemmas $\text{trancl-converseI} = \text{tranclp-converseI}$ $[\text{to-set}]$

lemma tranclp-converseD : $(r^{--1})^{++} x y \implies (r^{++})^{--1} x y$

$\langle \text{proof} \rangle$

lemmas $\text{trancl-converseD} = \text{tranclp-converseD}$ $[\text{to-set}]$

lemma tranclp-converse : $(r^{--1})^{++} = (r^{++})^{--1}$

$\langle \text{proof} \rangle$

lemmas $\text{trancl-converse} = \text{tranclp-converse}$ $[\text{to-set}]$

lemma sym-trancl : $\text{sym } r \implies \text{sym } (r^+)$

$\langle \text{proof} \rangle$

lemma $\text{converse-tranclp-induct}$:

assumes $\text{major}: r^{++} a b$

and cases: $!!y. r y b \implies P(y)$

!!y z. [r y z; r⁺⁺ z b; P(z)] ==> P(y)
 shows P a
 <proof>

lemmas converse-trancl-induct = converse-tranclp-induct [to-set]

lemma tranclpD: R⁺⁺ x y ==> EX z. R x z ∧ R^{**} z y
 <proof>

lemmas tranclD = tranclpD [to-set]

lemma tranclD2:
 (x, y) ∈ R⁺ ==> ∃ z. (x, z) ∈ R^{*} ∧ (z, y) ∈ R
 <proof>

lemma irrefl-tranclI: r⁻¹ ∩ r^{*} = { } ==> (x, x) ∉ r⁺
 <proof>

lemma irrefl-trancl-rD: !!X. ALL x. (x, x) ∉ r⁺ ==> (x, y) ∈ r ==> x ≠ y
 <proof>

lemma trancl-subset-Sigma-aux:
 (a, b) ∈ r^{*} ==> r ⊆ A × A ==> a = b ∨ a ∈ A
 <proof>

lemma trancl-subset-Sigma: r ⊆ A × A ==> r⁺ ⊆ A × A
 <proof>

lemma reflcl-tranclp [simp]: (r⁺⁺)⁺ = r^{**}
 <proof>

lemmas reflcl-trancl [simp] = reflcl-tranclp [to-set]

lemma trancl-reflcl [simp]: (r⁼)⁺ = r^{*}
 <proof>

lemma trancl-empty [simp]: { }⁺ = { }
 <proof>

lemma rtrancl-empty [simp]: { }^{*} = Id
 <proof>

lemma rtranclpD: R^{**} a b ==> a = b ∨ a ≠ b ∧ R⁺⁺ a b
 <proof>

lemmas rtranclD = rtranclpD [to-set]

lemma rtrancl-eq-or-trancl:
 (x, y) ∈ R^{*} = (x = y ∨ x ≠ y ∧ (x, y) ∈ R⁺)

$\langle \text{proof} \rangle$

Domain and Range

lemma *Domain-rtrancl* [simp]: $\text{Domain } (R^*) = \text{UNIV}$
 $\langle \text{proof} \rangle$

lemma *Range-rtrancl* [simp]: $\text{Range } (R^*) = \text{UNIV}$
 $\langle \text{proof} \rangle$

lemma *rtrancl-Un-subset*: $(R^* \cup S^*) \subseteq (R \cup S)^*$
 $\langle \text{proof} \rangle$

lemma *in-rtrancl-UnI*: $x \in R^* \vee x \in S^* \implies x \in (R \cup S)^*$
 $\langle \text{proof} \rangle$

lemma *trancl-domain* [simp]: $\text{Domain } (r^+) = \text{Domain } r$
 $\langle \text{proof} \rangle$

lemma *trancl-range* [simp]: $\text{Range } (r^+) = \text{Range } r$
 $\langle \text{proof} \rangle$

lemma *Not-Domain-rtrancl*:
 $x \sim: \text{Domain } R \implies ((x, y) : R^*) = (x = y)$
 $\langle \text{proof} \rangle$

lemma *trancl-subset-Field2*: $r^+ \leq \text{Field } r \times \text{Field } r$
 $\langle \text{proof} \rangle$

lemma *finite-trancl*: $\text{finite } (r^+) = \text{finite } r$
 $\langle \text{proof} \rangle$

More about converse *rtrancl* and *trancl*, should be merged with main body.

lemma *single-valued-confluent*:
 $\llbracket \text{single-valued } r; (x, y) \in r^*; (x, z) \in r^* \rrbracket$
 $\implies (y, z) \in r^* \vee (z, y) \in r^*$
 $\langle \text{proof} \rangle$

lemma *r-r-into-trancl*: $(a, b) \in R \implies (b, c) \in R \implies (a, c) \in R^+$
 $\langle \text{proof} \rangle$

lemma *trancl-into-trancl* [rule-format]:
 $(a, b) \in r^+ \implies (b, c) \in r \dashrightarrow (a, c) \in r^+$
 $\langle \text{proof} \rangle$

lemma *tranclp-rtranclp-tranclp*:
 $r^{++} a b \implies r^{**} b c \implies r^{++} a c$
 $\langle \text{proof} \rangle$

lemmas *trancl-rtrancl-trancl* = *tranclp-rtranclp-tranclp* [to-set]

```

lemmas transitive-closure-trans [trans] =
  r-r-into-trancl trancl-trans rtrancl-trans
  trancl.trancl-into-trancl trancl-into-trancl2
  rtrancl.rtrancl-into-rtrancl converse-rtrancl-into-rtrancl
  rtrancl-trancl-trancl trancl-rtrancl-trancl

lemmas transitive-closurep-trans' [trans] =
  tranclp-trans rtranclp-trans
  tranclp.trancl-into-trancl tranclp-into-tranclp2
  rtranclp.rtrancl-into-rtrancl converse-rtranclp-into-rtranclp
  rtranclp-tranclp-tranclp tranclp-rtranclp-tranclp

declare trancl-into-rtrancl [elim]

```

20.4 Setup of transitivity reasoner

$\langle ML \rangle$

end

21 Wellfounded: Well-founded Recursion

```

theory Wellfounded
imports Finite-Set Transitive-Closure Nat
uses (Tools/function-package/size.ML)
begin

```

21.1 Basic Definitions

```

inductive
  wfrec-rel :: ('a * 'a) set => (('a => 'b) => 'a => 'b) => 'a => 'b => bool
  for R :: ('a * 'a) set
  and F :: ('a => 'b) => 'a => 'b
where
  wfrecI: ALL z. (z, x) : R --> wfrec-rel R F z (g z) ==>
    wfrec-rel R F x (F g x)

constdefs
  wf :: ('a * 'a) set => bool
  wf(r) == (!P. (!x. (!y. (y,x):r --> P(y)) --> P(x)) --> (!x. P(x)))

  wfP :: ('a => 'a => bool) => bool
  wfP r == wf {(x, y). r x y}

  acyclic :: ('a * 'a) set => bool
  acyclic r == !x. (x,x) ~: r+

```

cut :: ('a => 'b) => ('a * 'a) set => 'a => 'a => 'b
cut f r x == (%y. if (y,x):r then f y else undefined)

adm-wf :: ('a * 'a) set => (('a => 'b) => 'a => 'b) => bool
adm-wf R F == ALL f g x.
 (ALL z. (z, x) : R --> f z = g z) --> F f x = F g x

wfrec :: ('a * 'a) set => (('a => 'b) => 'a => 'b) => 'a => 'b
 [code del]: *wfrec R F* == %x. THE y. *wfrec-rel* R (%f x. F (cut f R x) x) x y

abbreviation *acyclicP* :: ('a => 'a => bool) => bool **where**
acyclicP r == *acyclic* {(x, y). r x y}

lemma *wfP-wf-eq* [*pred-set-conv*]: *wfP* (λx y. (x, y) ∈ r) = *wf r*
 ⟨*proof*⟩

lemma *wfUNIVI*:
 (!!P x. (ALL x. (ALL y. (y,x) : r --> P(y)) --> P(x)) ==> P(x)) ==>
wf(r)
 ⟨*proof*⟩

lemmas *wfPUNIVI* = *wfUNIVI* [*to-pred*]

Restriction to domain *A* and range *B*. If *r* is well-founded over their intersection, then *wf r*

lemma *wfI*:
 [| r ⊆ A <*> B;
 !!x P. [|∀ x. (∀ y. (y,x) : r --> P y) --> P x; x : A; x : B |] ==> P x |]
 ==> *wf r*
 ⟨*proof*⟩

lemma *wf-induct*:
 [| *wf(r)*;
 !!x. [| ALL y. (y,x): r --> P(y) |] ==> P(x)
 |] ==> P(a)
 ⟨*proof*⟩

lemmas *wfP-induct* = *wf-induct* [*to-pred*]

lemmas *wf-induct-rule* = *wf-induct* [*rule-format*, *consumes 1*, *case-names less*,
induct set: wf]

lemmas *wfP-induct-rule* = *wf-induct-rule* [*to-pred*, *induct set: wfP*]

lemma *wf-not-sym*: *wf r* ==> (a, x) : r ==> (x, a) ~: r
 ⟨*proof*⟩

lemmas *wf-asym* = *wf-not-sym* [*elim-format*]

lemma *wf-not-refl* [*simp*]: $wf\ r \implies (a, a) \sim: r$
 $\langle proof \rangle$

lemmas *wf-irrefl* = *wf-not-refl* [*elim-format*]

lemma *wf-wellorderI*:
assumes *wf*: $wf\ \{(x::'a::ord, y). x < y\}$
assumes *lin*: *OFCLASS*('a::ord, *linorder-class*)
shows *OFCLASS*('a::ord, *wellorder-class*)
 $\langle proof \rangle$

lemma (*in wellorder*) *wf*:
 $wf\ \{(x, y). x < y\}$
 $\langle proof \rangle$

21.2 Basic Results

transitive closure of a well-founded relation is well-founded!

lemma *wf-trancl*:
assumes *wf* *r*
shows *wf* (r^+)
 $\langle proof \rangle$

lemmas *wfP-trancl* = *wf-trancl* [*to-pred*]

lemma *wf-converse-trancl*: $wf\ (r^-1) \implies wf\ ((r^+)^-1)$
 $\langle proof \rangle$

Minimal-element characterization of well-foundedness

lemma *wf-eq-minimal*: $wf\ r = (\forall Q\ x. x \in Q \implies (\exists z \in Q. \forall y. (y, z) \in r \implies y \notin Q))$
 $\langle proof \rangle$

lemma *wfE-min*:
assumes *wf* *R* $x \in Q$
obtains *z* **where** $z \in Q \wedge y. (y, z) \in R \implies y \notin Q$
 $\langle proof \rangle$

lemma *wfI-min*:
 $(\bigwedge x\ Q. x \in Q \implies \exists z \in Q. \forall y. (y, z) \in R \implies y \notin Q)$
 $\implies wf\ R$
 $\langle proof \rangle$

lemmas *wfP-eq-minimal* = *wf-eq-minimal* [*to-pred*]

Well-foundedness of subsets

lemma *wf-subset*: $[\mid wf(r);\ p \leq r \mid] \implies wf(p)$

$\langle proof \rangle$

lemmas *wfP-subset* = *wf-subset* [*to-pred*]

Well-foundedness of the empty relation

lemma *wf-empty* [*iff*]: *wf*({})

$\langle proof \rangle$

lemmas *wfP-empty* [*iff*] =
wf-empty [*to-pred bot-empty-eq2, simplified bot-fun-eq bot-bool-eq*]

lemma *wf-Int1*: *wf* *r* ==> *wf* (*r* Int *r'*)

$\langle proof \rangle$

lemma *wf-Int2*: *wf* *r* ==> *wf* (*r'* Int *r*)

$\langle proof \rangle$

Well-foundedness of insert

lemma *wf-insert* [*iff*]: *wf*(*insert* (*y*,*x*) *r*) = (*wf*(*r*) & (*x*,*y*) ~: *r*^*)

$\langle proof \rangle$

Well-foundedness of image

lemma *wf-prod-fun-image*: [*wf* *r*; *inj* *f*] ==> *wf*(*prod-fun* *f* *f* ' *r*)

$\langle proof \rangle$

21.3 Well-Foundedness Results for Unions

lemma *wf-union-compatible*:

assumes *wf* *R* *wf* *S*

assumes *S* *O* *R* \subseteq *R*

shows *wf* (*R* \cup *S*)

$\langle proof \rangle$

Well-foundedness of indexed union with disjoint domains and ranges

lemma *wf-UN*: [*ALL* *i*:*I*. *wf*(*r* *i*);

ALL *i*:*I*. *ALL* *j*:*I*. *r* *i* ~ = *r* *j* --> *Domain*(*r* *i*) Int *Range*(*r* *j*) = {}

] ==> *wf*(*UN* *i*:*I*. *r* *i*)

$\langle proof \rangle$

lemmas *wfP-SUP* = *wf-UN* [**where** *I*=*UNIV* **and** *r*= $\lambda i. \{(x, y). r\ i\ x\ y\}$,
to-pred SUP-UN-eq2 bot-empty-eq pred-equals-eq, simplified, standard]

lemma *wf-Union*:

[*ALL* *r*:*R*. *wf* *r*;

ALL *r*:*R*. *ALL* *s*:*R*. *r* ~ = *s* --> *Domain* *r* Int *Range* *s* = {}

] ==> *wf*(*Union* *R*)

$\langle proof \rangle$

lemma *wf-Un*:

$\llbracket \text{wf } r; \text{wf } s; \text{Domain } r \text{ Int Range } s = \{\} \rrbracket \implies \text{wf}(r \text{ Un } s)$
 $\langle \text{proof} \rangle$

lemma *wf-union-merge*:

$\text{wf}(R \cup S) = \text{wf}(R \circ R \cup R \circ S \cup S)$ (is $\text{wf } ?A = \text{wf } ?B$)
 $\langle \text{proof} \rangle$

lemma *wf-comp-self*: $\text{wf } R = \text{wf}(R \circ R)$ — special case

$\langle \text{proof} \rangle$

21.3.1 acyclic

lemma *acyclicI*: $\text{ALL } x. (x, x) \sim: r^+ \implies \text{acyclic } r$

$\langle \text{proof} \rangle$

lemma *wf-acyclic*: $\text{wf } r \implies \text{acyclic } r$

$\langle \text{proof} \rangle$

lemmas *wfP-acyclicP* = *wf-acyclic* [to-pred]

lemma *acyclic-insert* [iff]:

$\text{acyclic}(\text{insert}(y, x) \ r) = (\text{acyclic } r \ \& \ (x, y) \sim: r^*)$
 $\langle \text{proof} \rangle$

lemma *acyclic-converse* [iff]: $\text{acyclic}(r^{-1}) = \text{acyclic } r$

$\langle \text{proof} \rangle$

lemmas *acyclicP-converse* [iff] = *acyclic-converse* [to-pred]

lemma *acyclic-impl-antisym-rtrancl*: $\text{acyclic } r \implies \text{antisym}(r^*)$

$\langle \text{proof} \rangle$

lemma *acyclic-subset*: $\llbracket \text{acyclic } s; r \leq s \rrbracket \implies \text{acyclic } r$

$\langle \text{proof} \rangle$

Wellfoundedness of finite acyclic relations

lemma *finite-acyclic-wf* [rule-format]: $\text{finite } r \implies \text{acyclic } r \dashrightarrow \text{wf } r$

$\langle \text{proof} \rangle$

lemma *finite-acyclic-wf-converse*: $\llbracket \text{finite } r; \text{acyclic } r \rrbracket \implies \text{wf}(r^{-1})$

$\langle \text{proof} \rangle$

lemma *wf-iff-acyclic-if-finite*: $\text{finite } r \implies \text{wf } r = \text{acyclic } r$

$\langle \text{proof} \rangle$

21.4 Well-Founded Recursion

cut

lemma *cuts-eq*: $(cut\ f\ r\ x = cut\ g\ r\ x) = (ALL\ y.\ (y,x):r \dashrightarrow f(y)=g(y))$
 $\langle proof \rangle$

lemma *cut-apply*: $(x,a):r \implies (cut\ f\ r\ a)(x) = f(x)$
 $\langle proof \rangle$

Inductive characterization of wfrec combinator; for details see: John Harrison, “Inductive definitions: automation and application”

lemma *wfrec-unique*: $[adm\text{-}wf\ R\ F; wf\ R] \implies EX! y.\ wfrec\text{-}rel\ R\ F\ x\ y$
 $\langle proof \rangle$

lemma *adm-lemma*: $adm\text{-}wf\ R\ (\%f\ x.\ F\ (cut\ f\ R\ x)\ x)$
 $\langle proof \rangle$

lemma *wfrec*: $wf(r) \implies wfrec\ r\ H\ a = H\ (cut\ (wfrec\ r\ H)\ r\ a)\ a$
 $\langle proof \rangle$

21.5 Code generator setup

consts-code

```
wfrec  (<module>wfrec?)
attach <<
fun wfrec f x = f (wfrec f) x;
>>
```

21.6 nat is well-founded

lemma *less-nat-rel*: $op < = (\lambda m\ n.\ n = Suc\ m)^\wedge{++}$
 $\langle proof \rangle$

definition

```
pred-nat :: (nat * nat) set where
pred-nat = {(m, n). n = Suc m}
```

definition

```
less-than :: (nat * nat) set where
less-than = pred-nat^+
```

lemma *less-eq*: $(m, n) \in pred\text{-}nat^\wedge{+} \longleftrightarrow m < n$
 $\langle proof \rangle$

lemma *pred-nat-trancl-eq-le*:
 $(m, n) \in pred\text{-}nat^\wedge{*} \longleftrightarrow m \leq n$
 $\langle proof \rangle$

lemma *wf-pred-nat*: $wf\ pred\text{-}nat$

$\langle proof \rangle$

lemma *wf-less-than* [iff]: *wf less-than*
 $\langle proof \rangle$

lemma *trans-less-than* [iff]: *trans less-than*
 $\langle proof \rangle$

lemma *less-than-iff* [iff]: $((x,y): less-than) = (x < y)$
 $\langle proof \rangle$

lemma *wf-less*: *wf* $\{(x, y::nat). x < y\}$
 $\langle proof \rangle$

21.7 Accessible Part

Inductive definition of the accessible part *acc r* of a relation; see also [?].

inductive-set

acc :: $('a * 'a) set \Rightarrow 'a set$

for *r* :: $('a * 'a) set$

where

accI: $(!!y. (y, x) : r \Rightarrow y : acc\ r) \Rightarrow x : acc\ r$

abbreviation

termip :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$ **where**

termip *r* == *accp* (r^{-1-1})

abbreviation

termi :: $('a * 'a) set \Rightarrow 'a set$ **where**

termi *r* == *acc* (r^{-1})

lemmas *accpI* = *accp.accI*

Induction rules

theorem *accp-induct*:

assumes *major*: *accp r a*

assumes *hyp*: $!!x. accp\ r\ x \Rightarrow \forall y. r\ y\ x \longrightarrow P\ y \Rightarrow P\ x$

shows *P a*

$\langle proof \rangle$

theorems *accp-induct-rule* = *accp-induct* [rule-format, induct set: *accp*]

theorem *accp-downward*: *accp r b* $\Rightarrow r\ a\ b \Rightarrow accp\ r\ a$

$\langle proof \rangle$

lemma *not-accp-down*:

assumes *na*: $\neg accp\ R\ x$

obtains *z* **where** *R z x* **and** $\neg accp\ R\ z$

$\langle proof \rangle$

lemma *accp-downwards-aux*: $r^{**} b a \implies accp\ r\ a \dashv\dashv accp\ r\ b$
 $\langle proof \rangle$

theorem *accp-downwards*: $accp\ r\ a \implies r^{**} b a \implies accp\ r\ b$
 $\langle proof \rangle$

theorem *accp-wfPI*: $\forall x. accp\ r\ x \implies wfP\ r$
 $\langle proof \rangle$

theorem *accp-wfPD*: $wfP\ r \implies accp\ r\ x$
 $\langle proof \rangle$

theorem *wfP-accp-iff*: $wfP\ r = (\forall x. accp\ r\ x)$
 $\langle proof \rangle$

Smaller relations have bigger accessible parts:

lemma *accp-subset*:
assumes *sub*: $R1 \leq R2$
shows $accp\ R2 \leq accp\ R1$
 $\langle proof \rangle$

This is a generalized induction theorem that works on subsets of the accessible part.

lemma *accp-subset-induct*:
assumes *subset*: $D \leq accp\ R$
and *dcl*: $\bigwedge x z. \llbracket D\ x; R\ z\ x \rrbracket \implies D\ z$
and $D\ x$
and *istep*: $\bigwedge x. \llbracket D\ x; (\bigwedge z. R\ z\ x \implies P\ z) \rrbracket \implies P\ x$
shows $P\ x$
 $\langle proof \rangle$

Set versions of the above theorems

lemmas *acc-induct* = *accp-induct* [*to-set*]

lemmas *acc-induct-rule* = *acc-induct* [*rule-format*, *induct set*: *acc*]

lemmas *acc-downward* = *accp-downward* [*to-set*]

lemmas *not-acc-down* = *not-accp-down* [*to-set*]

lemmas *acc-downwards-aux* = *accp-downwards-aux* [*to-set*]

lemmas *acc-downwards* = *accp-downwards* [*to-set*]

lemmas *acc-wfI* = *accp-wfPI* [*to-set*]

lemmas *acc-wfD* = *accp-wfPD* [*to-set*]

lemmas $wf\text{-}acc\text{-}iff = wfP\text{-}accp\text{-}iff$ $[to\text{-}set]$

lemmas $acc\text{-}subset = accp\text{-}subset$ $[to\text{-}set\ pred\text{-}subset\text{-}eq]$

lemmas $acc\text{-}subset\text{-}induct = accp\text{-}subset\text{-}induct$ $[to\text{-}set\ pred\text{-}subset\text{-}eq]$

21.8 Tools for building wellfounded relations

Inverse Image

lemma $wf\text{-}inv\text{-}image$ $[simp,intro!]$: $wf(r) ==> wf(inv\text{-}image\ r\ (f::'a==>'b))$
 $\langle proof \rangle$

lemma $in\text{-}inv\text{-}image[simp]$: $((x,y) : inv\text{-}image\ r\ f) = ((f\ x, f\ y) : r)$
 $\langle proof \rangle$

Measure functions into nat

definition $measure :: ('a ==> nat) ==> ('a * 'a) set$
where $measure == inv\text{-}image\ less\text{-}than$

lemma $in\text{-}measure[simp]$: $((x,y) : measure\ f) = (f\ x < f\ y)$
 $\langle proof \rangle$

lemma $wf\text{-}measure$ $[iff]$: $wf\ (measure\ f)$
 $\langle proof \rangle$

Lexicographic combinations

definition
 $lex\text{-}prod :: (('a * 'a) set, ('b * 'b) set) ==> (('a * 'b) * ('a * 'b)) set$
 $(\mathbf{infixr} <*lex*> 80)$

where

$ra <*lex*> rb == \{((a,b),(a',b')). (a,a') : ra \mid a=a' \ \& \ (b,b') : rb\}$

lemma $wf\text{-}lex\text{-}prod$ $[intro!]$: $[| wf(ra); wf(rb) |] ==> wf(ra <*lex*> rb)$
 $\langle proof \rangle$

lemma $in\text{-}lex\text{-}prod[simp]$:
 $((a,b),(a',b')) : r <*lex*> s = ((a,a') : r \vee (a = a' \wedge (b, b') : s))$
 $\langle proof \rangle$

$op <*lex*>$ preserves transitivity

lemma $trans\text{-}lex\text{-}prod$ $[intro!]$:
 $[| trans\ R1; trans\ R2 |] ==> trans\ (R1 <*lex*> R2)$
 $\langle proof \rangle$

lexicographic combinations with measure functions

definition
 $mlex\text{-}prod :: ('a ==> nat) ==> ('a \times 'a) set ==> ('a \times 'a) set$ $(\mathbf{infixr} <*mlex*> 80)$
where

$f < *mlex* > R = \text{inv-image } (\text{less-than } < *lex* > R) (\%x. (f\ x, x))$

lemma *wf-mlex*: $wf\ R \implies wf\ (f < *mlex* > R)$
 $\langle \text{proof} \rangle$

lemma *mlex-less*: $f\ x < f\ y \implies (x, y) \in f < *mlex* > R$
 $\langle \text{proof} \rangle$

lemma *mlex-leq*: $f\ x \leq f\ y \implies (x, y) \in R \implies (x, y) \in f < *mlex* > R$
 $\langle \text{proof} \rangle$

proper subset relation on finite sets

definition *finite-psubset* :: $('a\ \text{set} \times 'a\ \text{set})\ \text{set}$
where *finite-psubset* == $\{(A, B). A < B \ \& \ \text{finite } B\}$

lemma *wf-finite-psubset[simp]*: $wf(\text{finite-psubset})$
 $\langle \text{proof} \rangle$

lemma *trans-finite-psubset*: $\text{trans } \text{finite-psubset}$
 $\langle \text{proof} \rangle$

lemma *in-finite-psubset[simp]*: $(A, B) \in \text{finite-psubset} = (A < B \ \& \ \text{finite } B)$
 $\langle \text{proof} \rangle$

max- and min-extension of order to finite sets

inductive-set *max-ext* :: $('a \times 'a)\ \text{set} \Rightarrow ('a\ \text{set} \times 'a\ \text{set})\ \text{set}$
for $R :: ('a \times 'a)\ \text{set}$

where

max-extI[intro]: $\text{finite } X \implies \text{finite } Y \implies Y \neq \{\} \implies (\bigwedge x. x \in X \implies \exists y \in Y. (x, y) \in R) \implies (X, Y) \in \text{max-ext } R$

lemma *max-ext-wf*:
assumes *wf*: $wf\ r$
shows $wf\ (\text{max-ext } r)$
 $\langle \text{proof} \rangle$

lemma *max-ext-additive*:
 $(A, B) \in \text{max-ext } R \implies (C, D) \in \text{max-ext } R \implies$
 $(A \cup C, B \cup D) \in \text{max-ext } R$
 $\langle \text{proof} \rangle$

definition

min-ext :: $('a \times 'a)\ \text{set} \Rightarrow ('a\ \text{set} \times 'a\ \text{set})\ \text{set}$

where

[code del]: $\text{min-ext } r = \{(X, Y) \mid X\ Y. X \neq \{\} \wedge (\forall y \in Y. (\exists x \in X. (x, y) \in r))\}$

lemma *min-ext-wf*:

assumes *wf r*
shows *wf (min-ext r)*
 ⟨*proof*⟩

Wellfoundedness of *same-fst*

definition

same-fst :: (*'a* => *bool*) => (*'a* => (*'b* * *'b*)*set*) => ((*'a***'b*)*(*'a***'b*))*set*

where

same-fst P R == {((*x',y'*),(*x,y*)) . *x'=x* & *P x* & (*y',y*) : *R x*}

— For *rec-def* declarations where the first *n* parameters stay unchanged in the recursive call.

lemma *same-fstI [intro!]*:

[| *P x*; (*y',y*) : *R x* |] ==> ((*x,y'*),(*x,y*)) : *same-fst P R*
 ⟨*proof*⟩

lemma *wf-same-fst*:

assumes *prem*: (!*x*. *P x* ==> *wf (R x)*)
shows *wf (same-fst P R)*
 ⟨*proof*⟩

21.9 Weakly decreasing sequences (w.r.t. some well-founded order) stabilize.

This material does not appear to be used any longer.

lemma *sequence-trans*: [| *ALL i*. (*f (Suc i)*, *f i*) : *r^** |] ==> (*f (i+k)*, *f i*) : *r^**
 ⟨*proof*⟩

lemma *wf-weak-decr-stable*:

assumes *as*: *ALL i*. (*f (Suc i)*, *f i*) : *r^** *wf (r^+)*
shows *EX i*. *ALL k*. *f (i+k) = f i*
 ⟨*proof*⟩

lemma *weak-decr-stable*:

ALL i. *f (Suc i) <= ((f i)::nat)* ==> *EX i*. *ALL k*. *f (i+k) = f i*
 ⟨*proof*⟩

21.10 size of a datatype value

⟨*ML*⟩

lemma *size-bool [code]*:

size (b::bool) = 0 ⟨*proof*⟩

lemma *nat-size [simp, code]*: *size (n::nat) = n*

⟨*proof*⟩


```

declare prod.size [noatp]

lemma [code]:
  size (P :: 'a Predicate.pred) = 0 ⟨proof⟩

lemma [code]:
  pred-size f P = 0 ⟨proof⟩

end

```

22 FunDef: Function Definitions and Termination Proofs

```

theory FunDef
imports Wellfounded
uses
  Tools/prop-logic.ML
  Tools/sat-solver.ML
  (Tools/function-package/fundef-lib.ML)
  (Tools/function-package/fundef-common.ML)
  (Tools/function-package/inductive-wrap.ML)
  (Tools/function-package/context-tree.ML)
  (Tools/function-package/fundef-core.ML)
  (Tools/function-package/sum-tree.ML)
  (Tools/function-package/mutual.ML)
  (Tools/function-package/pattern-split.ML)
  (Tools/function-package/fundef-package.ML)
  (Tools/function-package/auto-term.ML)
  (Tools/function-package/measure-functions.ML)
  (Tools/function-package/lexicographic-order.ML)
  (Tools/function-package/fundef-datatype.ML)
  (Tools/function-package/induction-scheme.ML)
  (Tools/function-package/termination.ML)
  (Tools/function-package/decompose.ML)
  (Tools/function-package/descent.ML)
  (Tools/function-package/scnp-solve.ML)
  (Tools/function-package/scnp-reconstruct.ML)
begin

```

22.1 Definitions with default value.

```

definition
  THE-default :: 'a  $\Rightarrow$  ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a where
  THE-default d P = (if ( $\exists !x. P\ x$ ) then (THE x. P x) else d)

lemma THE-defaultI':  $\exists !x. P\ x \Longrightarrow P\ (THE\text{-default}\ d\ P)$ 
  ⟨proof⟩

```

lemma *THE-default1-equality*:

$\llbracket \exists !x. P\ x; P\ a \rrbracket \implies \text{THE-default } d\ P = a$
 $\langle \text{proof} \rangle$

lemma *THE-default-none*:

$\neg(\exists !x. P\ x) \implies \text{THE-default } d\ P = d$
 $\langle \text{proof} \rangle$

lemma *fundef-ex1-existence*:

assumes *f-def*: $f == (\lambda x::'a. \text{THE-default } (d\ x) (\lambda y. G\ x\ y))$
assumes *ex1*: $\exists !y. G\ x\ y$
shows $G\ x\ (f\ x)$
 $\langle \text{proof} \rangle$

lemma *fundef-ex1-uniqueness*:

assumes *f-def*: $f == (\lambda x::'a. \text{THE-default } (d\ x) (\lambda y. G\ x\ y))$
assumes *ex1*: $\exists !y. G\ x\ y$
assumes *elm*: $G\ x\ (h\ x)$
shows $h\ x = f\ x$
 $\langle \text{proof} \rangle$

lemma *fundef-ex1-iff*:

assumes *f-def*: $f == (\lambda x::'a. \text{THE-default } (d\ x) (\lambda y. G\ x\ y))$
assumes *ex1*: $\exists !y. G\ x\ y$
shows $(G\ x\ y) = (f\ x = y)$
 $\langle \text{proof} \rangle$

lemma *fundef-default-value*:

assumes *f-def*: $f == (\lambda x::'a. \text{THE-default } (d\ x) (\lambda y. G\ x\ y))$
assumes *graph*: $\bigwedge x\ y. G\ x\ y \implies D\ x$
assumes $\neg D\ x$
shows $f\ x = d\ x$
 $\langle \text{proof} \rangle$

definition *in-rel-def[simp]*:

$\text{in-rel } R\ x\ y == (x, y) \in R$

lemma *wf-in-rel*:

$\text{wf } R \implies \text{wfP } (\text{in-rel } R)$
 $\langle \text{proof} \rangle$

$\langle ML \rangle$

22.2 Measure Functions

inductive *is-measure* :: $('a \Rightarrow \text{nat}) \Rightarrow \text{bool}$

where *is-measure-trivial*: $\text{is-measure } f$

$\langle ML \rangle$

lemma *measure-size*[*measure-function*]: *is-measure size*
 $\langle proof \rangle$

lemma *measure-fst*[*measure-function*]: *is-measure f* \implies *is-measure* ($\lambda p. f (fst p)$)
 $\langle proof \rangle$

lemma *measure-snd*[*measure-function*]: *is-measure f* \implies *is-measure* ($\lambda p. f (snd p)$)
 $\langle proof \rangle$

$\langle ML \rangle$

22.3 Congruence Rules

lemma *let-cong* [*fundef-cong*]:
 $M = N \implies (\bigwedge x. x = N \implies f x = g x) \implies Let M f = Let N g$
 $\langle proof \rangle$

lemmas [*fundef-cong*] =
if-cong image-cong INT-cong UN-cong
be-cong ball-cong imp-cong

lemma *split-cong* [*fundef-cong*]:
 $(\bigwedge x y. (x, y) = q \implies f x y = g x y) \implies p = q$
 $\implies split f p = split g q$
 $\langle proof \rangle$

lemma *comp-cong* [*fundef-cong*]:
 $f (g x) = f' (g' x') \implies (f \circ g) x = (f' \circ g') x'$
 $\langle proof \rangle$

22.4 Simp rules for termination proofs

lemma *termination-basic-simps*[*termination-simp*]:
 $x < (y :: nat) \implies x < y + z$
 $x < z \implies x < y + z$
 $x \leq y \implies x \leq y + (z :: nat)$
 $x \leq z \implies x \leq y + (z :: nat)$
 $x < y \implies x \leq (y :: nat)$
 $\langle proof \rangle$

declare *le-imp-less-Suc*[*termination-simp*]

lemma *prod-size-simp*[*termination-simp*]:
 $prod-size f g p = f (fst p) + g (snd p) + Suc 0$
 $\langle proof \rangle$

22.5 Decomposition

lemma *less-by-empty*:

$$A = \{\} \implies A \subseteq B$$

and *union-comp-emptyL*:

$$\llbracket A \ O \ C = \{\}; B \ O \ C = \{\} \rrbracket \implies (A \cup B) \ O \ C = \{\}$$

and *union-comp-emptyR*:

$$\llbracket A \ O \ B = \{\}; A \ O \ C = \{\} \rrbracket \implies A \ O \ (B \cup C) = \{\}$$

and *wf-no-loop*:

$$R \ O \ R = \{\} \implies wf \ R$$

<proof>

22.6 Reduction Pairs

definition

$$reduction\text{-}pair \ P = (wf \ (fst \ P) \wedge snd \ P \ O \ fst \ P \subseteq fst \ P)$$

lemma *reduction-pairI[intro]*: $wf \ R \implies S \ O \ R \subseteq R \implies reduction\text{-}pair \ (R, S)$

<proof>

lemma *reduction-pair-lemma*:

assumes *rp*: *reduction-pair* *P*

assumes $R \subseteq fst \ P$

assumes $S \subseteq snd \ P$

assumes *wf* *S*

shows *wf* $(R \cup S)$

<proof>

definition

$$rp\text{-}inv\text{-}image = (\lambda(R,S) \ f. \ (inv\text{-}image \ R \ f, \ inv\text{-}image \ S \ f))$$

lemma *rp-inv-image-rp*:

$$reduction\text{-}pair \ P \implies reduction\text{-}pair \ (rp\text{-}inv\text{-}image \ P \ f)$$

<proof>

22.7 Concrete orders for SCNP termination proofs

definition *pair-less* = *less-than* *<*lex*>* *less-than*

definition [code del]: *pair-leq* = *pair-less* $\hat{=}$

definition *max-strict* = *max-ext* *pair-less*

definition [code del]: *max-weak* = *max-ext* *pair-leq* $\cup \{(\{\}, \{\})\}$

definition [code del]: *min-strict* = *min-ext* *pair-less*

definition [code del]: *min-weak* = *min-ext* *pair-leq* $\cup \{(\{\}, \{\})\}$

lemma *wf-pair-less[simp]*: *wf* *pair-less*

<proof>

Introduction rules for *pair-less*/*pair-leq*

lemma *pair-leqI1*: $a < b \implies ((a, s), (b, t)) \in pair\text{-}leq$

and *pair-leqI2*: $a \leq b \implies s \leq t \implies ((a, s), (b, t)) \in pair\text{-}leq$

and *pair-lessI1*: $a < b \implies ((a, s), (b, t)) \in \text{pair-less}$
and *pair-lessI2*: $a \leq b \implies s < t \implies ((a, s), (b, t)) \in \text{pair-less}$
 ⟨*proof*⟩

Introduction rules for max

lemma *smax-emptyI*:
 $\text{finite } Y \implies Y \neq \{\} \implies (\{\}, Y) \in \text{max-strict}$
and *smax-insertI*:
 $\llbracket y \in Y; (x, y) \in \text{pair-less}; (X, Y) \in \text{max-strict} \rrbracket \implies (\text{insert } x \ X, Y) \in \text{max-strict}$
and *wmax-emptyI*:
 $\text{finite } X \implies (\{\}, X) \in \text{max-weak}$
and *wmax-insertI*:
 $\llbracket y \in YS; (x, y) \in \text{pair-leq}; (XS, YS) \in \text{max-weak} \rrbracket \implies (\text{insert } x \ XS, YS) \in \text{max-weak}$
 ⟨*proof*⟩

Introduction rules for min

lemma *smin-emptyI*:
 $X \neq \{\} \implies (X, \{\}) \in \text{min-strict}$
and *smin-insertI*:
 $\llbracket x \in XS; (x, y) \in \text{pair-less}; (XS, YS) \in \text{min-strict} \rrbracket \implies (XS, \text{insert } y \ YS) \in \text{min-strict}$
and *wmin-emptyI*:
 $(X, \{\}) \in \text{min-weak}$
and *wmin-insertI*:
 $\llbracket x \in XS; (x, y) \in \text{pair-leq}; (XS, YS) \in \text{min-weak} \rrbracket \implies (XS, \text{insert } y \ YS) \in \text{min-weak}$
 ⟨*proof*⟩

Reduction Pairs

lemma *max-ext-compat*:
assumes $S \ O \ R \subseteq R$
shows $(\text{max-ext } S \cup \{(\{\}, \{\})\}) \ O \ \text{max-ext } R \subseteq \text{max-ext } R$
 ⟨*proof*⟩

lemma *max-rpair-set*: *reduction-pair* (*max-strict*, *max-weak*)
 ⟨*proof*⟩

lemma *min-ext-compat*:
assumes $S \ O \ R \subseteq R$
shows $(\text{min-ext } S \cup \{(\{\}, \{\})\}) \ O \ \text{min-ext } R \subseteq \text{min-ext } R$
 ⟨*proof*⟩

lemma *min-rpair-set*: *reduction-pair* (*min-strict*, *min-weak*)
 ⟨*proof*⟩

22.8 Tool setup

⟨*ML*⟩

end

23 Record: Extensible records with structural subtyping

```
theory Record
imports Product-Type
uses (Tools/record-package.ML)
begin
```

```
lemma prop-subst:  $s = t \implies PROP\ P\ t \implies PROP\ P\ s$ 
  <proof>
```

```
lemma rec-UNIV-I:  $\bigwedge x. x \in UNIV \equiv True$ 
  <proof>
```

```
lemma rec-True-simp:  $(True \implies PROP\ P) \equiv PROP\ P$ 
  <proof>
```

```
lemma K-record-comp:  $(\lambda x. c) \circ f = (\lambda x. c)$ 
  <proof>
```

23.1 Concrete record syntax

nonterminals

ident field-type field-types field fields update updates

syntax

```
-constify      ::  $id \Rightarrow ident$                 (-)
-constify      ::  $longid \Rightarrow ident$              (-)

-field-type    ::  $[ident, type] \Rightarrow field-type$    ((2- ::/ -))
               ::  $field-type \Rightarrow field-types$     (-)
-field-types   ::  $[field-type, field-types] \Rightarrow field-types$  (-,/ -)
-record-type   ::  $field-types \Rightarrow type$           ((3'(| - |'))
-record-type-scheme ::  $[field-types, type] \Rightarrow type$  ((3'(| -,/ (2... ::/ -) |'))

-field         ::  $[ident, 'a] \Rightarrow field$           ((2- =/ -))
               ::  $field \Rightarrow fields$               (-)
-fields        ::  $[field, fields] \Rightarrow fields$     (-,/ -)
-record        ::  $fields \Rightarrow 'a$                  ((3'(| - |'))
-record-scheme ::  $[fields, 'a] \Rightarrow 'a$            ((3'(| -,/ (2... =/ -) |'))

-update-name   ::  $idt$ 
-update        ::  $[ident, 'a] \Rightarrow update$         ((2- :=/ -))
               ::  $update \Rightarrow updates$             (-)
-updates       ::  $[update, updates] \Rightarrow updates$  (-,/ -)
```

```

-record-update    :: ['a, updates] => 'b                (-(/ (3'(| - |')) [900,0] 900))

syntax (xsymbols)
-record-type      :: field-types => type                ((3(|-)))
-record-type-scheme :: [field-types, type] => type      ((3(|-, / (2... ::/ -)))
-record          :: fields => 'a                        ((3(|-)))
-record-scheme    :: [fields, 'a] => 'a                ((3(|-, / (2... =/ -)))
-record-update    :: ['a, updates] => 'b                (-(/ (3(|-)) [900,0] 900))

⟨ML⟩

end

```

24 Option: Datatype option

```

theory Option
imports Datatype Finite-Set
begin

```

```

datatype 'a option = None | Some 'a

```

```

lemma not-None-eq [iff]: (x ~ = None) = (EX y. x = Some y)
  ⟨proof⟩

```

```

lemma not-Some-eq [iff]: (ALL y. x ~ = Some y) = (x = None)
  ⟨proof⟩

```

Although it may appear that both of these equalities are helpful only when applied to assumptions, in practice it seems better to give them the uniform iff attribute.

```

lemma option-caseE:
  assumes c: (case x of None => P | Some y => Q y)
  obtains
    (None) x = None and P
  | (Some) y where x = Some y and Q y
  ⟨proof⟩

```

```

lemma insert-None-conv-UNIV: insert None (range Some) = UNIV
  ⟨proof⟩

```

```

instance option :: (finite) finite ⟨proof⟩

```

```

lemma inj-Some [simp]: inj-on Some A
  ⟨proof⟩

```

24.0.1 Operations

```

primrec the :: 'a option => 'a where

```

the (*Some* *x*) = *x*

primrec *set* :: 'a option => 'a set **where**
set *None* = {} |
set (*Some* *x*) = {*x*}

lemma *ospec* [*dest*]: (*ALL* *x*:*set* *A*. *P* *x*) ==> *A* = *Some* *x* ==> *P* *x*
 <*proof*>

<*ML*>

lemma *elem-set* [*iff*]: (*x* : *set* *xo*) = (*xo* = *Some* *x*)
 <*proof*>

lemma *set-empty-eq* [*simp*]: (*set* *xo* = {}) = (*xo* = *None*)
 <*proof*>

definition

map :: ('a => 'b) => 'a option => 'b option

where

[*code del*]: *map* = (%*f* *y*. case *y* of *None* => *None* | *Some* *x* => *Some* (*f* *x*))

lemma *option-map-None* [*simp*, *code*]: *map* *f* *None* = *None*
 <*proof*>

lemma *option-map-Some* [*simp*, *code*]: *map* *f* (*Some* *x*) = *Some* (*f* *x*)
 <*proof*>

lemma *option-map-is-None* [*iff*]:
 (*map* *f* *opt* = *None*) = (*opt* = *None*)
 <*proof*>

lemma *option-map-eq-Some* [*iff*]:
 (*map* *f* *xo* = *Some* *y*) = (*EX* *z*. *xo* = *Some* *z* & *f* *z* = *y*)
 <*proof*>

lemma *option-map-comp*:
map *f* (*map* *g* *opt*) = *map* (*f* o *g*) *opt*
 <*proof*>

lemma *option-map-o-sum-case* [*simp*]:
map *f* o *sum-case* *g* *h* = *sum-case* (*map* *f* o *g*) (*map* *f* o *h*)
 <*proof*>

hide (**open**) *const* *set* *map*

24.0.2 Code generator setup**definition**

is-none :: 'a option \Rightarrow bool **where**
is-none-none [code post, symmetric, code inline]: *is-none* *x* \longleftrightarrow *x* = None

lemma *is-none-code* [code]:

shows *is-none* None \longleftrightarrow True
and *is-none* (Some *x*) \longleftrightarrow False
 ⟨proof⟩

hide (open) const *is-none***code-type** *option*

(SML - option)
 (OCaml - option)
 (Haskell Maybe -)

code-const None and Some

(SML NONE and SOME)
 (OCaml None and Some -)
 (Haskell Nothing and Just)

code-instance *option* :: eq

(Haskell -)

code-const *eq-class.eq* :: 'a::eq option \Rightarrow 'a option \Rightarrow bool

(Haskell infixl 4 ==)

code-reserved SML*option* NONE SOME**code-reserved** OCaml*option* None Some**end****25 Extraction: Program extraction for HOL****theory** *Extraction***imports** *Option***uses** *Tools/rewrite-hol-proof.ML***begin****25.1 Setup**

⟨ML⟩

lemmas [extraction-expand] =
meta-spec atomize-eq atomize-all atomize-imp atomize-conj
allE rev-mp conjE Eq-TrueI Eq-FalseI eqTrueI eqTrueE eq-cong2
notE' impE' impE iffE imp-cong simp-thms eq-True eq-False
induct-forall-eq induct-implies-eq induct-equal-eq induct-conj-eq
induct-forall-def induct-implies-def induct-equal-def induct-conj-def
induct-atomize induct-rulify induct-rulify-fallback
True-implies-equals TrueE

datatype *sumbool* = *Left* | *Right*

25.2 Type of extracted program

extract-type

typeof (*Trueprop* *P*) \equiv *typeof* *P*

typeof *P* \equiv *Type* (*TYPE*(*Null*)) \implies *typeof* *Q* \equiv *Type* (*TYPE*('Q)) \implies
typeof (*P* \longrightarrow *Q*) \equiv *Type* (*TYPE*('Q))

typeof *Q* \equiv *Type* (*TYPE*(*Null*)) \implies *typeof* (*P* \longrightarrow *Q*) \equiv *Type* (*TYPE*(*Null*))

typeof *P* \equiv *Type* (*TYPE*('P)) \implies *typeof* *Q* \equiv *Type* (*TYPE*('Q)) \implies
typeof (*P* \longrightarrow *Q*) \equiv *Type* (*TYPE*('P \Rightarrow 'Q))

($\lambda x. \text{typeof } (P\ x)$) \equiv ($\lambda x. \text{Type } (\text{TYPE}(\text{Null}))$) \implies
typeof ($\forall x. P\ x$) \equiv *Type* (*TYPE*(*Null*))

($\lambda x. \text{typeof } (P\ x)$) \equiv ($\lambda x. \text{Type } (\text{TYPE}('P))$) \implies
typeof ($\forall x::'a. P\ x$) \equiv *Type* (*TYPE*('a \Rightarrow 'P))

($\lambda x. \text{typeof } (P\ x)$) \equiv ($\lambda x. \text{Type } (\text{TYPE}(\text{Null}))$) \implies
typeof ($\exists x::'a. P\ x$) \equiv *Type* (*TYPE*('a))

($\lambda x. \text{typeof } (P\ x)$) \equiv ($\lambda x. \text{Type } (\text{TYPE}('P))$) \implies
typeof ($\exists x::'a. P\ x$) \equiv *Type* (*TYPE*('a \times 'P))

typeof *P* \equiv *Type* (*TYPE*(*Null*)) \implies *typeof* *Q* \equiv *Type* (*TYPE*(*Null*)) \implies
typeof (*P* \vee *Q*) \equiv *Type* (*TYPE*(*sumbool*))

typeof *P* \equiv *Type* (*TYPE*(*Null*)) \implies *typeof* *Q* \equiv *Type* (*TYPE*('Q)) \implies
typeof (*P* \vee *Q*) \equiv *Type* (*TYPE*('Q option))

typeof *P* \equiv *Type* (*TYPE*('P)) \implies *typeof* *Q* \equiv *Type* (*TYPE*(*Null*)) \implies
typeof (*P* \vee *Q*) \equiv *Type* (*TYPE*('P option))

typeof *P* \equiv *Type* (*TYPE*('P)) \implies *typeof* *Q* \equiv *Type* (*TYPE*('Q)) \implies
typeof (*P* \vee *Q*) \equiv *Type* (*TYPE*('P + 'Q))

typeof *P* \equiv *Type* (*TYPE*(*Null*)) \implies *typeof* *Q* \equiv *Type* (*TYPE*('Q)) \implies

$$\text{typeof } (P \wedge Q) \equiv \text{Type } (\text{TYPE}('Q))$$

$$\begin{aligned} \text{typeof } P \equiv \text{Type } (\text{TYPE}('P)) &\implies \text{typeof } Q \equiv \text{Type } (\text{TYPE}(\text{Null})) \implies \\ \text{typeof } (P \wedge Q) &\equiv \text{Type } (\text{TYPE}('P)) \end{aligned}$$

$$\begin{aligned} \text{typeof } P \equiv \text{Type } (\text{TYPE}('P)) &\implies \text{typeof } Q \equiv \text{Type } (\text{TYPE}('Q)) \implies \\ \text{typeof } (P \wedge Q) &\equiv \text{Type } (\text{TYPE}('P \times 'Q)) \end{aligned}$$

$$\text{typeof } (P = Q) \equiv \text{typeof } ((P \longrightarrow Q) \wedge (Q \longrightarrow P))$$

$$\text{typeof } (x \in P) \equiv \text{typeof } P$$

25.3 Realizability

realizability

$$(\text{realizes } t \text{ (Trueprop } P)) \equiv (\text{Trueprop } (\text{realizes } t \text{ } P))$$

$$\begin{aligned} (\text{typeof } P) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (P \longrightarrow Q)) &\equiv (\text{realizes } \text{Null } P \longrightarrow \text{realizes } t \text{ } Q) \end{aligned}$$

$$\begin{aligned} (\text{typeof } P) \equiv (\text{Type } (\text{TYPE}('P))) &\implies \\ (\text{typeof } Q) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (P \longrightarrow Q)) &\equiv (\forall x :: 'P. \text{realizes } x \text{ } P \longrightarrow \text{realizes } \text{Null } Q) \end{aligned}$$

$$(\text{realizes } t \text{ } (P \longrightarrow Q)) \equiv (\forall x. \text{realizes } x \text{ } P \longrightarrow \text{realizes } (t \text{ } x) \text{ } Q)$$

$$\begin{aligned} (\lambda x. \text{typeof } (P \text{ } x)) \equiv (\lambda x. \text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (\forall x. P \text{ } x)) &\equiv (\forall x. \text{realizes } \text{Null } (P \text{ } x)) \end{aligned}$$

$$(\text{realizes } t \text{ } (\forall x. P \text{ } x)) \equiv (\forall x. \text{realizes } (t \text{ } x) \text{ } (P \text{ } x))$$

$$\begin{aligned} (\lambda x. \text{typeof } (P \text{ } x)) \equiv (\lambda x. \text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (\exists x. P \text{ } x)) &\equiv (\text{realizes } \text{Null } (P \text{ } t)) \end{aligned}$$

$$(\text{realizes } t \text{ } (\exists x. P \text{ } x)) \equiv (\text{realizes } (\text{snd } t) \text{ } (P \text{ } (\text{fst } t)))$$

$$\begin{aligned} (\text{typeof } P) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{typeof } Q) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (P \vee Q)) &\equiv \\ (\text{case } t \text{ of Left } \Rightarrow \text{realizes } \text{Null } P \mid \text{Right } \Rightarrow \text{realizes } \text{Null } Q) \end{aligned}$$

$$\begin{aligned} (\text{typeof } P) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (P \vee Q)) &\equiv \\ (\text{case } t \text{ of None } \Rightarrow \text{realizes } \text{Null } P \mid \text{Some } q \Rightarrow \text{realizes } q \text{ } Q) \end{aligned}$$

$$\begin{aligned} (\text{typeof } Q) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) &\implies \\ (\text{realizes } t \text{ } (P \vee Q)) &\equiv \\ (\text{case } t \text{ of None } \Rightarrow \text{realizes } \text{Null } Q \mid \text{Some } p \Rightarrow \text{realizes } p \text{ } P) \end{aligned}$$

$$\begin{aligned}
& (\text{realizes } t \ (P \vee Q)) \equiv \\
& \quad (\text{case } t \text{ of } \text{Inl } p \Rightarrow \text{realizes } p \ P \mid \text{Inr } q \Rightarrow \text{realizes } q \ Q) \\
\\
& (\text{typeof } P) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) \implies \\
& \quad (\text{realizes } t \ (P \wedge Q)) \equiv (\text{realizes } \text{Null } P \wedge \text{realizes } t \ Q) \\
\\
& (\text{typeof } Q) \equiv (\text{Type } (\text{TYPE}(\text{Null}))) \implies \\
& \quad (\text{realizes } t \ (P \wedge Q)) \equiv (\text{realizes } t \ P \wedge \text{realizes } \text{Null } Q) \\
\\
& (\text{realizes } t \ (P \wedge Q)) \equiv (\text{realizes } (\text{fst } t) \ P \wedge \text{realizes } (\text{snd } t) \ Q) \\
\\
& \text{typeof } P \equiv \text{Type } (\text{TYPE}(\text{Null})) \implies \\
& \quad \text{realizes } t \ (\neg P) \equiv \neg \text{realizes } \text{Null } P \\
\\
& \text{typeof } P \equiv \text{Type } (\text{TYPE}('P)) \implies \\
& \quad \text{realizes } t \ (\neg P) \equiv (\forall x::'P. \neg \text{realizes } x \ P) \\
\\
& \text{typeof } (P::\text{bool}) \equiv \text{Type } (\text{TYPE}(\text{Null})) \implies \\
& \quad \text{typeof } Q \equiv \text{Type } (\text{TYPE}(\text{Null})) \implies \\
& \quad \text{realizes } t \ (P = Q) \equiv \text{realizes } \text{Null } P = \text{realizes } \text{Null } Q \\
\\
& (\text{realizes } t \ (P = Q)) \equiv (\text{realizes } t \ ((P \longrightarrow Q) \wedge (Q \longrightarrow P)))
\end{aligned}$$

25.4 Computational content of basic inference rules

theorem *disjE-realizer*:

assumes r : $\text{case } x \text{ of } \text{Inl } p \Rightarrow P \ p \mid \text{Inr } q \Rightarrow Q \ q$
and $r1$: $\bigwedge p. P \ p \implies R \ (f \ p)$ **and** $r2$: $\bigwedge q. Q \ q \implies R \ (g \ q)$
shows $R \ (\text{case } x \text{ of } \text{Inl } p \Rightarrow f \ p \mid \text{Inr } q \Rightarrow g \ q)$
 $\langle \text{proof} \rangle$

theorem *disjE-realizer2*:

assumes r : $\text{case } x \text{ of } \text{None} \Rightarrow P \mid \text{Some } q \Rightarrow Q \ q$
and $r1$: $P \implies R \ f$ **and** $r2$: $\bigwedge q. Q \ q \implies R \ (g \ q)$
shows $R \ (\text{case } x \text{ of } \text{None} \Rightarrow f \mid \text{Some } q \Rightarrow g \ q)$
 $\langle \text{proof} \rangle$

theorem *disjE-realizer3*:

assumes r : $\text{case } x \text{ of } \text{Left} \Rightarrow P \mid \text{Right} \Rightarrow Q$
and $r1$: $P \implies R \ f$ **and** $r2$: $Q \implies R \ g$
shows $R \ (\text{case } x \text{ of } \text{Left} \Rightarrow f \mid \text{Right} \Rightarrow g)$
 $\langle \text{proof} \rangle$

theorem *conjI-realizer*:

$P \ p \implies Q \ q \implies P \ (\text{fst } (p, q)) \wedge Q \ (\text{snd } (p, q))$
 $\langle \text{proof} \rangle$

theorem *exI-realizer*:

$P \ y \ x \implies P \ (\text{snd } (x, y)) \ (\text{fst } (x, y)) \ \langle \text{proof} \rangle$

theorem *exE-realizer*: $P \text{ (snd } p) \text{ (fst } p) \implies$
 $(\bigwedge x y. P y x \implies Q (f x y)) \implies Q (\text{let } (x, y) = p \text{ in } f x y)$
 $\langle \text{proof} \rangle$

theorem *exE-realizer'*: $P \text{ (snd } p) \text{ (fst } p) \implies$
 $(\bigwedge x y. P y x \implies Q) \implies Q \langle \text{proof} \rangle$

$\langle ML \rangle$

realizers

impI (P, Q): $\lambda pq. pq$
 $\Lambda P Q pq (h: -). \text{allI } \cdot \cdot \cdot (\Lambda x. \text{impI } \cdot \cdot \cdot \cdot (h \cdot x))$

impI (P): *Null*
 $\Lambda P Q (h: -). \text{allI } \cdot \cdot \cdot (\Lambda x. \text{impI } \cdot \cdot \cdot \cdot (h \cdot x))$

impI (Q): $\lambda q. q \Lambda P Q q. \text{impI } \cdot \cdot \cdot \cdot$

impI: *Null impI*

mp (P, Q): $\lambda pq. pq$
 $\Lambda P Q pq (h: -) p. mp \cdot \cdot \cdot \cdot (\text{spec } \cdot \cdot \cdot p \cdot h)$

mp (P): *Null*
 $\Lambda P Q (h: -) p. mp \cdot \cdot \cdot \cdot (\text{spec } \cdot \cdot \cdot p \cdot h)$

mp (Q): $\lambda q. q \Lambda P Q q. mp \cdot \cdot \cdot \cdot$

mp: *Null mp*

allI (P): $\lambda p. p \Lambda P p. \text{allI } \cdot \cdot$

allI: *Null allI*

spec (P): $\lambda x p. p x \Lambda P x p. \text{spec } \cdot \cdot \cdot x$

spec: *Null spec*

exI (P): $\lambda x p. (x, p) \Lambda P x p. \text{exI-realizer } \cdot P \cdot p \cdot x$

exI: $\lambda x. x \Lambda P x (h: -). h$

exE (P, Q): $\lambda p pq. \text{let } (x, y) = p \text{ in } pq x y$
 $\Lambda P Q p (h: -) pq. \text{exE-realizer } \cdot P \cdot p \cdot Q \cdot pq \cdot h$

exE (P): *Null*
 $\Lambda P Q p. \text{exE-realizer}' \cdot \cdot \cdot \cdot \cdot$

$exE \ (Q): \lambda x \ pq. \ pq \ x$
 $\Lambda \ P \ Q \ x \ (h1: -) \ pq \ (h2: -). \ h2 \cdot x \cdot h1$

$exE: \text{Null}$
 $\Lambda \ P \ Q \ x \ (h1: -) \ (h2: -). \ h2 \cdot x \cdot h1$

$conjI \ (P, Q): \text{Pair}$
 $\Lambda \ P \ Q \ p \ (h: -) \ q. \ conjI\text{-realizer} \cdot P \cdot p \cdot Q \cdot q \cdot h$

$conjI \ (P): \lambda p. \ p$
 $\Lambda \ P \ Q \ p. \ conjI \cdot - \cdot - \cdot -$

$conjI \ (Q): \lambda q. \ q$
 $\Lambda \ P \ Q \ (h: -) \ q. \ conjI \cdot - \cdot - \cdot - \cdot h$

$conjI: \text{Null } conjI$

$conjunct1 \ (P, Q): \text{fst}$
 $\Lambda \ P \ Q \ pq. \ conjunct1 \cdot - \cdot - \cdot -$

$conjunct1 \ (P): \lambda p. \ p$
 $\Lambda \ P \ Q \ p. \ conjunct1 \cdot - \cdot - \cdot -$

$conjunct1 \ (Q): \text{Null}$
 $\Lambda \ P \ Q \ q. \ conjunct1 \cdot - \cdot - \cdot -$

$conjunct1: \text{Null } conjunct1$

$conjunct2 \ (P, Q): \text{snd}$
 $\Lambda \ P \ Q \ pq. \ conjunct2 \cdot - \cdot - \cdot -$

$conjunct2 \ (P): \text{Null}$
 $\Lambda \ P \ Q \ p. \ conjunct2 \cdot - \cdot - \cdot -$

$conjunct2 \ (Q): \lambda p. \ p$
 $\Lambda \ P \ Q \ p. \ conjunct2 \cdot - \cdot - \cdot -$

$conjunct2: \text{Null } conjunct2$

$disjI1 \ (P, Q): \text{Inl}$
 $\Lambda \ P \ Q \ p. \ iffD2 \cdot - \cdot - \cdot - \cdot (sum.cases-1 \cdot P \cdot - \cdot p)$

$disjI1 \ (P): \text{Some}$
 $\Lambda \ P \ Q \ p. \ iffD2 \cdot - \cdot - \cdot - \cdot (option.cases-2 \cdot - \cdot P \cdot p)$

$disjI1 \ (Q): \text{None}$
 $\Lambda \ P \ Q. \ iffD2 \cdot - \cdot - \cdot - \cdot (option.cases-1 \cdot - \cdot - \cdot -)$

$disjI1: \text{Left}$

$$\Lambda P Q. \text{iffD2} \cdot \cdot \cdot \cdot (\text{sumbool.cases-1} \cdot \cdot \cdot -)$$

$$\text{disjI2 } (P, Q): \text{Inr} \\ \Lambda Q P q. \text{iffD2} \cdot \cdot \cdot \cdot (\text{sum.cases-2} \cdot \cdot \cdot Q \cdot q)$$

$$\text{disjI2 } (P): \text{None} \\ \Lambda Q P. \text{iffD2} \cdot \cdot \cdot \cdot (\text{option.cases-1} \cdot \cdot \cdot -)$$

$$\text{disjI2 } (Q): \text{Some} \\ \Lambda Q P q. \text{iffD2} \cdot \cdot \cdot \cdot (\text{option.cases-2} \cdot \cdot \cdot Q \cdot q)$$

$$\text{disjI2}: \text{Right} \\ \Lambda Q P. \text{iffD2} \cdot \cdot \cdot \cdot (\text{sumbool.cases-2} \cdot \cdot \cdot -)$$

$$\text{disjE } (P, Q, R): \lambda pq \text{ pr } qr. \\ (\text{case } pq \text{ of } \text{Inl } p \Rightarrow \text{pr } p \mid \text{Inr } q \Rightarrow \text{qr } q) \\ \Lambda P Q R pq (h1: -) \text{ pr } (h2: -) \text{ qr}. \\ \text{disjE-realizer} \cdot \cdot \cdot \cdot pq \cdot R \cdot \text{pr} \cdot \text{qr} \cdot h1 \cdot h2$$

$$\text{disjE } (Q, R): \lambda pq \text{ pr } qr. \\ (\text{case } pq \text{ of } \text{None} \Rightarrow \text{pr} \mid \text{Some } q \Rightarrow \text{qr } q) \\ \Lambda P Q R pq (h1: -) \text{ pr } (h2: -) \text{ qr}. \\ \text{disjE-realizer2} \cdot \cdot \cdot \cdot pq \cdot R \cdot \text{pr} \cdot \text{qr} \cdot h1 \cdot h2$$

$$\text{disjE } (P, R): \lambda pq \text{ pr } qr. \\ (\text{case } pq \text{ of } \text{None} \Rightarrow \text{qr} \mid \text{Some } p \Rightarrow \text{pr } p) \\ \Lambda P Q R pq (h1: -) \text{ pr } (h2: -) \text{ qr } (h3: -). \\ \text{disjE-realizer2} \cdot \cdot \cdot \cdot pq \cdot R \cdot \text{qr} \cdot \text{pr} \cdot h1 \cdot h3 \cdot h2$$

$$\text{disjE } (R): \lambda pq \text{ pr } qr. \\ (\text{case } pq \text{ of } \text{Left} \Rightarrow \text{pr} \mid \text{Right} \Rightarrow \text{qr}) \\ \Lambda P Q R pq (h1: -) \text{ pr } (h2: -) \text{ qr}. \\ \text{disjE-realizer3} \cdot \cdot \cdot \cdot pq \cdot R \cdot \text{pr} \cdot \text{qr} \cdot h1 \cdot h2$$

$$\text{disjE } (P, Q): \text{Null} \\ \Lambda P Q R pq. \text{disjE-realizer} \cdot \cdot \cdot \cdot pq \cdot (\lambda x. R) \cdot \cdot \cdot$$

$$\text{disjE } (Q): \text{Null} \\ \Lambda P Q R pq. \text{disjE-realizer2} \cdot \cdot \cdot \cdot pq \cdot (\lambda x. R) \cdot \cdot \cdot$$

$$\text{disjE } (P): \text{Null} \\ \Lambda P Q R pq (h1: -) (h2: -) (h3: -). \\ \text{disjE-realizer2} \cdot \cdot \cdot \cdot pq \cdot (\lambda x. R) \cdot \cdot \cdot \cdot h1 \cdot h3 \cdot h2$$

$$\text{disjE}: \text{Null} \\ \Lambda P Q R pq. \text{disjE-realizer3} \cdot \cdot \cdot \cdot pq \cdot (\lambda x. R) \cdot \cdot \cdot$$

$$\text{FalseE } (P): \text{default} \\ \Lambda P. \text{FalseE} \cdot -$$

FalseE: *Null FalseE*

notI (*P*): *Null*
 $\Lambda P (h: -). \text{allI} \cdot - \cdot (\Lambda x. \text{notI} \cdot - \cdot (h \cdot x))$

notI: *Null notI*

notE (*P*, *R*): $\lambda p. \text{default}$
 $\Lambda P R (h: -) p. \text{notE} \cdot - \cdot - \cdot (\text{spec} \cdot - \cdot p \cdot h)$

notE (*P*): *Null*
 $\Lambda P R (h: -) p. \text{notE} \cdot - \cdot - \cdot (\text{spec} \cdot - \cdot p \cdot h)$

notE (*R*): *default*
 $\Lambda P R. \text{notE} \cdot - \cdot -$

notE: *Null notE*

subst (*P*): $\lambda s \ t \ ps. ps$
 $\Lambda s \ t \ P (h: -) ps. \text{subst} \cdot s \cdot t \cdot P \ ps \cdot h$

subst: *Null subst*

iffD1 (*P*, *Q*): *fst*
 $\Lambda Q \ P \ pq (h: -) p.$
 $mp \cdot - \cdot - \cdot (\text{spec} \cdot - \cdot p \cdot (\text{conjunct1} \cdot - \cdot - \cdot h))$

iffD1 (*P*): $\lambda p. p$
 $\Lambda Q \ P \ p (h: -). mp \cdot - \cdot - \cdot (\text{conjunct1} \cdot - \cdot - \cdot h)$

iffD1 (*Q*): *Null*
 $\Lambda Q \ P \ q1 (h: -) q2.$
 $mp \cdot - \cdot - \cdot (\text{spec} \cdot - \cdot q2 \cdot (\text{conjunct1} \cdot - \cdot - \cdot h))$

iffD1: *Null iffD1*

iffD2 (*P*, *Q*): *snd*
 $\Lambda P \ Q \ pq (h: -) q.$
 $mp \cdot - \cdot - \cdot (\text{spec} \cdot - \cdot q \cdot (\text{conjunct2} \cdot - \cdot - \cdot h))$

iffD2 (*P*): $\lambda p. p$
 $\Lambda P \ Q \ p (h: -). mp \cdot - \cdot - \cdot (\text{conjunct2} \cdot - \cdot - \cdot h)$

iffD2 (*Q*): *Null*
 $\Lambda P \ Q \ q1 (h: -) q2.$
 $mp \cdot - \cdot - \cdot (\text{spec} \cdot - \cdot q2 \cdot (\text{conjunct2} \cdot - \cdot - \cdot h))$

iffD2: *Null iffD2*


```

iffI (P, Q): Pair
  Λ P Q pq (h1 : -) qp (h2 : -). conjI-realizer ·
    (λpq. ∀ x. P x → Q (pq x)) · pq ·
    (λqp. ∀ x. Q x → P (qp x)) · qp ·
    (allI · - · (Λ x. impI · - · - · (h1 · x))) ·
    (allI · - · (Λ x. impI · - · - · (h2 · x)))

```

```

iffI (P): λp. p
  Λ P Q (h1 : -) p (h2 : -). conjI · - · - ·
    (allI · - · (Λ x. impI · - · - · (h1 · x))) ·
    (impI · - · - · h2)

```

```

iffI (Q): λq. q
  Λ P Q q (h1 : -) (h2 : -). conjI · - · - ·
    (impI · - · - · h1) ·
    (allI · - · (Λ x. impI · - · - · (h2 · x)))

```

```

iffI: Null iffI

```

⟨ML⟩

end

26 Divides: The division operators div and mod

```

theory Divides
imports Nat Power Product-Type
uses ~~/src/Provers/Arith/cancel-div-mod.ML
begin

```

26.1 Syntactic division operations

```

class div = dvd +
  fixes div :: 'a ⇒ 'a ⇒ 'a (infixl div 70)
  and mod :: 'a ⇒ 'a ⇒ 'a (infixl mod 70)

```

26.2 Abstract division in commutative semirings.

```

class semiring-div = comm-semiring-1-cancel + div +
  assumes mod-div-equality: a div b * b + a mod b = a
  and div-by-0 [simp]: a div 0 = 0
  and div-0 [simp]: 0 div a = 0
  and div-mult-self1 [simp]: b ≠ 0 ⇒ (a + c * b) div b = c + a div b
begin

```

op div and *op mod*

lemma *mod-div-equality2*: $b * (a \text{ div } b) + a \text{ mod } b = a$
 $\langle \text{proof} \rangle$

lemma *mod-div-equality'*: $a \text{ mod } b + a \text{ div } b * b = a$
 $\langle \text{proof} \rangle$

lemma *div-mod-equality*: $((a \text{ div } b) * b + a \text{ mod } b) + c = a + c$
 $\langle \text{proof} \rangle$

lemma *div-mod-equality2*: $(b * (a \text{ div } b) + a \text{ mod } b) + c = a + c$
 $\langle \text{proof} \rangle$

lemma *mod-by-0* [simp]: $a \text{ mod } 0 = a$
 $\langle \text{proof} \rangle$

lemma *mod-0* [simp]: $0 \text{ mod } a = 0$
 $\langle \text{proof} \rangle$

lemma *div-mult-self2* [simp]:
 assumes $b \neq 0$
 shows $(a + b * c) \text{ div } b = c + a \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *mod-mult-self1* [simp]: $(a + c * b) \text{ mod } b = a \text{ mod } b$
 $\langle \text{proof} \rangle$

lemma *mod-mult-self2* [simp]: $(a + b * c) \text{ mod } b = a \text{ mod } b$
 $\langle \text{proof} \rangle$

lemma *div-mult-self1-is-id* [simp]: $b \neq 0 \implies b * a \text{ div } b = a$
 $\langle \text{proof} \rangle$

lemma *div-mult-self2-is-id* [simp]: $b \neq 0 \implies a * b \text{ div } b = a$
 $\langle \text{proof} \rangle$

lemma *mod-mult-self1-is-0* [simp]: $b * a \text{ mod } b = 0$
 $\langle \text{proof} \rangle$

lemma *mod-mult-self2-is-0* [simp]: $a * b \text{ mod } b = 0$
 $\langle \text{proof} \rangle$

lemma *div-by-1* [simp]: $a \text{ div } 1 = a$
 $\langle \text{proof} \rangle$

lemma *mod-by-1* [simp]: $a \text{ mod } 1 = 0$
 $\langle \text{proof} \rangle$

lemma *mod-self* [simp]: $a \text{ mod } a = 0$

$\langle proof \rangle$

lemma *div-self* [*simp*]: $a \neq 0 \implies a \text{ div } a = 1$
 $\langle proof \rangle$

lemma *div-add-self1* [*simp*]:
assumes $b \neq 0$
shows $(b + a) \text{ div } b = a \text{ div } b + 1$
 $\langle proof \rangle$

lemma *div-add-self2* [*simp*]:
assumes $b \neq 0$
shows $(a + b) \text{ div } b = a \text{ div } b + 1$
 $\langle proof \rangle$

lemma *mod-add-self1* [*simp*]:
 $(b + a) \text{ mod } b = a \text{ mod } b$
 $\langle proof \rangle$

lemma *mod-add-self2* [*simp*]:
 $(a + b) \text{ mod } b = a \text{ mod } b$
 $\langle proof \rangle$

lemma *mod-div-decomp*:
fixes $a \ b$
obtains $q \ r$ **where** $q = a \text{ div } b$ **and** $r = a \text{ mod } b$
and $a = q * b + r$
 $\langle proof \rangle$

lemma *dvd-eq-mod-eq-0* [*code unfold*]: $a \text{ dvd } b \iff b \text{ mod } a = 0$
 $\langle proof \rangle$

lemma *mod-div-trivial* [*simp*]: $a \text{ mod } b \text{ div } b = 0$
 $\langle proof \rangle$

lemma *mod-mod-trivial* [*simp*]: $a \text{ mod } b \text{ mod } b = a \text{ mod } b$
 $\langle proof \rangle$

lemma *dvd-imp-mod-0*: $a \text{ dvd } b \implies b \text{ mod } a = 0$
 $\langle proof \rangle$

lemma *dvd-div-mult-self*: $a \text{ dvd } b \implies (b \text{ div } a) * a = b$
 $\langle proof \rangle$

lemma *dvd-div-mult*: $a \text{ dvd } b \implies (b \text{ div } a) * c = b * c \text{ div } a$
 $\langle proof \rangle$

lemma *div-dvd-div* [*simp*]:
 $a \text{ dvd } b \implies a \text{ dvd } c \implies (b \text{ div } a \text{ dvd } c \text{ div } a) = (b \text{ dvd } c)$

$\langle proof \rangle$

lemma *dvd-mod-imp-dvd*: $[[k \text{ dvd } m \text{ mod } n; k \text{ dvd } n]] \implies k \text{ dvd } m$
 $\langle proof \rangle$

Addition respects modular equivalence.

lemma *mod-add-left-eq*: $(a + b) \text{ mod } c = (a \text{ mod } c + b) \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-add-right-eq*: $(a + b) \text{ mod } c = (a + b \text{ mod } c) \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-add-eq*: $(a + b) \text{ mod } c = (a \text{ mod } c + b \text{ mod } c) \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-add-cong*:
 assumes $a \text{ mod } c = a' \text{ mod } c$
 assumes $b \text{ mod } c = b' \text{ mod } c$
 shows $(a + b) \text{ mod } c = (a' + b') \text{ mod } c$
 $\langle proof \rangle$

lemma *div-add[simp]*: $z \text{ dvd } x \implies z \text{ dvd } y$
 $\implies (x + y) \text{ div } z = x \text{ div } z + y \text{ div } z$
 $\langle proof \rangle$

Multiplication respects modular equivalence.

lemma *mod-mult-left-eq*: $(a * b) \text{ mod } c = ((a \text{ mod } c) * b) \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-mult-right-eq*: $(a * b) \text{ mod } c = (a * (b \text{ mod } c)) \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-mult-eq*: $(a * b) \text{ mod } c = ((a \text{ mod } c) * (b \text{ mod } c)) \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-mult-cong*:
 assumes $a \text{ mod } c = a' \text{ mod } c$
 assumes $b \text{ mod } c = b' \text{ mod } c$
 shows $(a * b) \text{ mod } c = (a' * b') \text{ mod } c$
 $\langle proof \rangle$

lemma *mod-mod-cancel*:
 assumes $c \text{ dvd } b$
 shows $a \text{ mod } b \text{ mod } c = a \text{ mod } c$
 $\langle proof \rangle$

end

lemma *div-mult-div-if-dvd*: $(y::'a::\{\text{semiring-div,no-zero-divisors}\}) \text{ dvd } x \implies$

$z \text{ dvd } w \implies (x \text{ div } y) * (w \text{ div } z) = (x * w) \text{ div } (y * z)$
 $\langle \text{proof} \rangle$

lemma *div-power*: $(y::'a::\{\text{semiring-div}, \text{no-zero-divisors}, \text{recpower}\}) \text{ dvd } x \implies$
 $(x \text{ div } y)^{\wedge n} = x^{\wedge n} \text{ div } y^{\wedge n}$
 $\langle \text{proof} \rangle$

class *ring-div* = *semiring-div* + *comm-ring-1*
begin

Negation respects modular equivalence.

lemma *mod-minus-eq*: $(- a) \text{ mod } b = (- (a \text{ mod } b)) \text{ mod } b$
 $\langle \text{proof} \rangle$

lemma *mod-minus-cong*:
assumes $a \text{ mod } b = a' \text{ mod } b$
shows $(- a) \text{ mod } b = (- a') \text{ mod } b$
 $\langle \text{proof} \rangle$

Subtraction respects modular equivalence.

lemma *mod-diff-left-eq*: $(a - b) \text{ mod } c = (a \text{ mod } c - b) \text{ mod } c$
 $\langle \text{proof} \rangle$

lemma *mod-diff-right-eq*: $(a - b) \text{ mod } c = (a - b \text{ mod } c) \text{ mod } c$
 $\langle \text{proof} \rangle$

lemma *mod-diff-eq*: $(a - b) \text{ mod } c = (a \text{ mod } c - b \text{ mod } c) \text{ mod } c$
 $\langle \text{proof} \rangle$

lemma *mod-diff-cong*:
assumes $a \text{ mod } c = a' \text{ mod } c$
assumes $b \text{ mod } c = b' \text{ mod } c$
shows $(a - b) \text{ mod } c = (a' - b') \text{ mod } c$
 $\langle \text{proof} \rangle$

lemma *dvd-neg-div*: $y \text{ dvd } x \implies -x \text{ div } y = - (x \text{ div } y)$
 $\langle \text{proof} \rangle$

lemma *dvd-div-neg*: $y \text{ dvd } x \implies x \text{ div } -y = - (x \text{ div } y)$
 $\langle \text{proof} \rangle$

end

26.3 Division on *nat*

We define *op div* and *op mod* on *nat* by means of a characteristic relation with two input arguments *m*, *n* and two output arguments *q*(uotient) and *r*(emainder).

definition *divmod-rel* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *bool* **where**

divmod-rel *m n q r* $\longleftrightarrow m = q * n + r \wedge (\text{if } n > 0 \text{ then } 0 \leq r \wedge r < n \text{ else } q = 0)$

divmod-rel is total:

lemma *divmod-rel-ex*:

obtains *q r* **where** *divmod-rel m n q r*
 $\langle \text{proof} \rangle$

divmod-rel is injective:

lemma *divmod-rel-unique-div*:

assumes *divmod-rel m n q r*
and *divmod-rel m n q' r'*
shows *q = q'*
 $\langle \text{proof} \rangle$

lemma *divmod-rel-unique-mod*:

assumes *divmod-rel m n q r*
and *divmod-rel m n q' r'*
shows *r = r'*
 $\langle \text{proof} \rangle$

We instantiate divisibility on the natural numbers by means of *divmod-rel*:

instantiation *nat* :: *semiring-div*
begin

definition *divmod* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \times *nat* **where**

$[\text{code del}]: \text{divmod } m \ n = (\text{THE } (q, r). \text{divmod-rel } m \ n \ q \ r)$

definition *div-nat* **where**

m div n = fst (divmod m n)

definition *mod-nat* **where**

m mod n = snd (divmod m n)

lemma *divmod-div-mod*:

divmod m n = (m div n, m mod n)
 $\langle \text{proof} \rangle$

lemma *divmod-eq*:

assumes *divmod-rel m n q r*
shows *divmod m n = (q, r)*
 $\langle \text{proof} \rangle$

lemma *div-eq*:

assumes *divmod-rel m n q r*
shows *m div n = q*
 $\langle \text{proof} \rangle$

lemma *mod-eq*:

assumes *divmod-rel* $m\ n\ q\ r$

shows $m \bmod n = r$

$\langle proof \rangle$

lemma *divmod-rel*: *divmod-rel* $m\ n\ (m \div n)\ (m \bmod n)$

$\langle proof \rangle$

lemma *divmod-zero*:

divmod $m\ 0 = (0, m)$

$\langle proof \rangle$

lemma *divmod-base*:

assumes $m < n$

shows *divmod* $m\ n = (0, m)$

$\langle proof \rangle$

lemma *divmod-step*:

assumes $0 < n$ **and** $n \leq m$

shows *divmod* $m\ n = (Suc\ ((m - n) \div n), (m - n) \bmod n)$

$\langle proof \rangle$

The “recursion” equations for *op div* and *op mod*

lemma *div-less* [*simp*]:

fixes $m\ n :: nat$

assumes $m < n$

shows $m \div n = 0$

$\langle proof \rangle$

lemma *le-div-geq*:

fixes $m\ n :: nat$

assumes $0 < n$ **and** $n \leq m$

shows $m \div n = Suc\ ((m - n) \div n)$

$\langle proof \rangle$

lemma *mod-less* [*simp*]:

fixes $m\ n :: nat$

assumes $m < n$

shows $m \bmod n = m$

$\langle proof \rangle$

lemma *le-mod-geq*:

fixes $m\ n :: nat$

assumes $n \leq m$

shows $m \bmod n = (m - n) \bmod n$

$\langle proof \rangle$

instance $\langle proof \rangle$

end

Simproc for cancelling *op div* and *op mod*

$\langle ML \rangle$

code generator setup

lemma *divmod-if* [code]: $\text{divmod } m \ n = (\text{if } n = 0 \ \vee \ m < n \text{ then } (0, m) \text{ else } \text{let } (q, r) = \text{divmod } (m - n) \ n \text{ in } (\text{Suc } q, r))$
 $\langle \text{proof} \rangle$

code-modulename *SML*

Divides Nat

code-modulename *OCaml*

Divides Nat

code-modulename *Haskell*

Divides Nat

26.3.1 Quotient

lemma *div-geq*: $0 < n \implies \neg m < n \implies m \text{ div } n = \text{Suc } ((m - n) \text{ div } n)$
 $\langle \text{proof} \rangle$

lemma *div-if*: $0 < n \implies m \text{ div } n = (\text{if } m < n \text{ then } 0 \text{ else } \text{Suc } ((m - n) \text{ div } n))$
 $\langle \text{proof} \rangle$

lemma *div-mult-self-is-m* [simp]: $0 < n \implies (m * n) \text{ div } n = (m :: \text{nat})$
 $\langle \text{proof} \rangle$

lemma *div-mult-self1-is-m* [simp]: $0 < n \implies (n * m) \text{ div } n = (m :: \text{nat})$
 $\langle \text{proof} \rangle$

26.3.2 Remainder

lemma *mod-less-divisor* [simp]:

fixes $m \ n :: \text{nat}$

assumes $n > 0$

shows $m \text{ mod } n < (n :: \text{nat})$

$\langle \text{proof} \rangle$

lemma *mod-less-eq-dividend* [simp]:

fixes $m \ n :: \text{nat}$

shows $m \text{ mod } n \leq m$

$\langle \text{proof} \rangle$

lemma *mod-geq*: $\neg m < (n :: \text{nat}) \implies m \text{ mod } n = (m - n) \text{ mod } n$
 $\langle \text{proof} \rangle$

lemma *mod-if*: $m \text{ mod } (n :: \text{nat}) = (\text{if } m < n \text{ then } m \text{ else } (m - n) \text{ mod } n)$

$\langle proof \rangle$

lemma *mod-1* [*simp*]: $m \text{ mod } \text{Suc } 0 = 0$
 $\langle proof \rangle$

lemma *mod-mult-distrib*: $(m \text{ mod } n) * (k::nat) = (m * k) \text{ mod } (n * k)$
 $\langle proof \rangle$

lemma *mod-mult-distrib2*: $(k::nat) * (m \text{ mod } n) = (k*m) \text{ mod } (k*n)$
 $\langle proof \rangle$

lemma *mult-div-cancel*: $(n::nat) * (m \text{ div } n) = m - (m \text{ mod } n)$
 $\langle proof \rangle$

lemma *mod-le-divisor*[*simp*]: $0 < n \implies m \text{ mod } n \leq (n::nat)$
 $\langle proof \rangle$

26.3.3 Quotient and Remainder

lemma *divmod-rel-mult1-eq*:
 $[[\text{divmod-rel } b \ c \ q \ r; \ c > 0 \]]$
 $\implies \text{divmod-rel } (a*b) \ c \ (a*q + a*r \text{ div } c) \ (a*r \text{ mod } c)$
 $\langle proof \rangle$

lemma *div-mult1-eq*: $(a*b) \text{ div } c = a*(b \text{ div } c) + a*(b \text{ mod } c) \text{ div } (c::nat)$
 $\langle proof \rangle$

lemma *divmod-rel-add1-eq*:
 $[[\text{divmod-rel } a \ c \ aq \ ar; \text{divmod-rel } b \ c \ bq \ br; \ c > 0 \]]$
 $\implies \text{divmod-rel } (a + b) \ c \ (aq + bq + (ar+br) \text{ div } c) \ ((ar + br) \text{ mod } c)$
 $\langle proof \rangle$

lemma *div-add1-eq*:
 $(a+b) \text{ div } (c::nat) = a \text{ div } c + b \text{ div } c + ((a \text{ mod } c + b \text{ mod } c) \text{ div } c)$
 $\langle proof \rangle$

lemma *mod-lemma*: $[[(0::nat) < c; \ r < b \]] \implies b * (q \text{ mod } c) + r < b * c$
 $\langle proof \rangle$

lemma *divmod-rel-mult2-eq*: $[[\text{divmod-rel } a \ b \ q \ r; \ 0 < b; \ 0 < c \]]$
 $\implies \text{divmod-rel } a \ (b*c) \ (q \text{ div } c) \ (b*(q \text{ mod } c) + r)$
 $\langle proof \rangle$

lemma *div-mult2-eq*: $a \text{ div } (b*c) = (a \text{ div } b) \text{ div } (c::nat)$
 $\langle proof \rangle$

lemma *mod-mult2-eq*: $a \text{ mod } (b*c) = b*(a \text{ div } b \text{ mod } c) + a \text{ mod } (b::nat)$

$\langle \text{proof} \rangle$

26.3.4 Cancellation of Common Factors in Division

lemma *div-mult-mult-lemma*:

$\llbracket (0::\text{nat}) < b; \ 0 < c \rrbracket \implies (c*a) \text{ div } (c*b) = a \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *div-mult-mult1* [simp]: $(0::\text{nat}) < c \implies (c*a) \text{ div } (c*b) = a \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *div-mult-mult2* [simp]: $(0::\text{nat}) < c \implies (a*c) \text{ div } (b*c) = a \text{ div } b$
 $\langle \text{proof} \rangle$

26.3.5 Further Facts about Quotient and Remainder

lemma *div-1* [simp]: $m \text{ div } \text{Suc } 0 = m$
 $\langle \text{proof} \rangle$

lemma *div-le-mono* [rule-format]:
 $\forall m::\text{nat}. m \leq n \longrightarrow (m \text{ div } k) \leq (n \text{ div } k)$
 $\langle \text{proof} \rangle$

lemma *div-le-mono2*: $\llbracket 0 < m; m \leq n \rrbracket \implies (k \text{ div } n) \leq (k \text{ div } m)$
 $\langle \text{proof} \rangle$

lemma *div-le-dividend* [simp]: $m \text{ div } n \leq (m::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *div-less-dividend* [rule-format]:
 $\llbracket n::\text{nat}. 1 < n \rrbracket \implies 0 < m \longrightarrow m \text{ div } n < m$
 $\langle \text{proof} \rangle$

lemma *nat-div-eq-0* [simp]: $(n::\text{nat}) > 0 \implies ((m \text{ div } n) = 0) = (m < n)$
 $\langle \text{proof} \rangle$

lemma *nat-div-gt-0* [simp]: $(n::\text{nat}) > 0 \implies ((m \text{ div } n) > 0) = (m \geq n)$
 $\langle \text{proof} \rangle$

declare *div-less-dividend* [simp]

A fact for the mutilated chess board

lemma *mod-Suc*: $\text{Suc}(m) \text{ mod } n = (\text{if } \text{Suc}(m \text{ mod } n) = n \text{ then } 0 \text{ else } \text{Suc}(m \text{ mod } n))$
 $\langle \text{proof} \rangle$

26.3.6 The Divides Relation

lemma *dvd-1-left* [*iff*]: $Suc\ 0\ dvd\ k$
 $\langle proof \rangle$

lemma *dvd-1-iff-1* [*simp*]: $(m\ dvd\ Suc\ 0) = (m = Suc\ 0)$
 $\langle proof \rangle$

lemma *nat-dvd-1-iff-1* [*simp*]: $m\ dvd\ (1::nat) \longleftrightarrow m = 1$
 $\langle proof \rangle$

lemma *dvd-anti-sym*: $[| m\ dvd\ n; n\ dvd\ m |] ==> m = (n::nat)$
 $\langle proof \rangle$

op dvd is a partial order

interpretation *dvd*: *order op dvd* $\lambda n\ m :: nat. n\ dvd\ m \wedge \neg m\ dvd\ n$
 $\langle proof \rangle$

lemma *nat-dvd-diff* [*simp*]: $[| k\ dvd\ m; k\ dvd\ n |] ==> k\ dvd\ (m-n :: nat)$
 $\langle proof \rangle$

lemma *dvd-diffD*: $[| k\ dvd\ m-n; k\ dvd\ n; n \leq m |] ==> k\ dvd\ (m::nat)$
 $\langle proof \rangle$

lemma *dvd-diffD1*: $[| k\ dvd\ m-n; k\ dvd\ m; n \leq m |] ==> k\ dvd\ (n::nat)$
 $\langle proof \rangle$

lemma *dvd-reduce*: $(k\ dvd\ n + k) = (k\ dvd\ (n::nat))$
 $\langle proof \rangle$

lemma *dvd-mod*: $!!n::nat. [| f\ dvd\ m; f\ dvd\ n |] ==> f\ dvd\ m\ mod\ n$
 $\langle proof \rangle$

lemma *dvd-mod-iff*: $k\ dvd\ n ==> ((k::nat)\ dvd\ m\ mod\ n) = (k\ dvd\ m)$
 $\langle proof \rangle$

lemma *dvd-mult-cancel*: $!!k::nat. [| k*m\ dvd\ k*n; 0 < k |] ==> m\ dvd\ n$
 $\langle proof \rangle$

lemma *dvd-mult-cancel1*: $0 < m ==> (m*n\ dvd\ m) = (n = (1::nat))$
 $\langle proof \rangle$

lemma *dvd-mult-cancel2*: $0 < m ==> (n*m\ dvd\ m) = (n = (1::nat))$
 $\langle proof \rangle$

lemma *dvd-imp-le*: $[| k\ dvd\ n; 0 < n |] ==> k \leq (n::nat)$
 $\langle proof \rangle$

lemma *nat-dvd-not-less*: $(0::nat) < m \implies m < n \implies \neg n\ dvd\ m$
 $\langle proof \rangle$

lemma *dvd-mult-div-cancel*: $n \text{ dvd } m \implies n * (m \text{ div } n) = (m::nat)$
 ⟨proof⟩

lemma *nat-zero-less-power-iff* [simp]: $(x^n > 0) = (x > (0::nat) \mid n=0)$
 ⟨proof⟩

lemma *power-dvd-imp-le*: $[|i^n \text{ dvd } i^n; (1::nat) < i|] \implies m \leq n$
 ⟨proof⟩

lemma *mod-eq-0-iff*: $(m \bmod d = 0) = (\exists q::nat. m = d*q)$
 ⟨proof⟩

lemmas *mod-eq-0D* [dest!] = *mod-eq-0-iff* [THEN iffD1]

lemma *mod-eqD*: $(m \bmod d = r) \implies \exists q::nat. m = r + q*d$
 ⟨proof⟩

lemma *split-div*:
 $P(n \text{ div } k :: nat) =$
 $((k = 0 \longrightarrow P\ 0) \wedge (k \neq 0 \longrightarrow (!i. !j < k. n = k*i + j \longrightarrow P\ i)))$
 $(\text{is } ?P = ?Q \text{ is } - = (- \wedge (- \longrightarrow ?R)))$
 ⟨proof⟩

lemma *split-div-lemma*:
 assumes $0 < n$
 shows $n * q \leq m \wedge m < n * \text{Suc } q \longleftrightarrow q = ((m::nat) \text{ div } n) (\text{is } ?lhs \longleftrightarrow ?rhs)$
 ⟨proof⟩

theorem *split-div'*:
 $P((m::nat) \text{ div } n) = ((n = 0 \wedge P\ 0) \vee$
 $(\exists q. (n * q \leq m \wedge m < n * (\text{Suc } q)) \wedge P\ q))$
 ⟨proof⟩

lemma *split-mod*:
 $P(n \bmod k :: nat) =$
 $((k = 0 \longrightarrow P\ n) \wedge (k \neq 0 \longrightarrow (!i. !j < k. n = k*i + j \longrightarrow P\ j)))$
 $(\text{is } ?P = ?Q \text{ is } - = (- \wedge (- \longrightarrow ?R)))$
 ⟨proof⟩

theorem *mod-div-equality'*: $(m::nat) \bmod n = m - (m \text{ div } n) * n$
 ⟨proof⟩

lemma *div-mod-equality'*:
 fixes $m\ n :: nat$
 shows $m \text{ div } n * n = m - m \bmod n$
 ⟨proof⟩

26.3.7 An “induction” law for modulus arithmetic.

lemma *mod-induct-0*:
 assumes *step*: $\forall i < p. P\ i \longrightarrow P\ ((Suc\ i)\ mod\ p)$
 and *base*: $P\ i$ and $i: i < p$
 shows $P\ 0$
 $\langle proof \rangle$

lemma *mod-induct*:
 assumes *step*: $\forall i < p. P\ i \longrightarrow P\ ((Suc\ i)\ mod\ p)$
 and *base*: $P\ i$ and $i: i < p$ and $j: j < p$
 shows $P\ j$
 $\langle proof \rangle$

end

27 Plain: Plain HOL

theory *Plain*
imports *Datatype FunDef Record Extraction Divides*
begin

Plain bootstrap of fundamental HOL tools and packages; does not include *Hilbert-Choice*.

$\langle ML \rangle$

end

28 Relation-Power: Powers of Relations and Functions

theory *Relation-Power*
imports *Power Transitive-Closure Plain*
begin

instance
fun :: $(type, type)\ power\ \langle proof \rangle$

overloading
 $relpow \equiv power :: ('a \times 'a)\ set \Rightarrow nat \Rightarrow ('a \times 'a)\ set\ \text{(unchecked)}$
begin

$R \wedge n = R\ O\ \dots\ O\ R$, the n -fold composition of R

primrec *relpow* **where**
 $(R :: ('a \times 'a)\ set) \wedge 0 = Id$
 $| (R :: ('a \times 'a)\ set) \wedge Suc\ n = R\ O\ (R \wedge n)$

end

overloading

$\text{funpow} \equiv \text{power} :: ('a \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a$ (**unchecked**)

begin

$f \wedge n = f \circ \dots \circ f$, the n -fold composition of f

primrec funpow where

$(f :: 'a \Rightarrow 'a) \wedge 0 = \text{id}$
 $| (f :: 'a \Rightarrow 'a) \wedge \text{Suc } n = f \circ (f \wedge n)$

end

WARNING: due to the limits of Isabelle’s type classes, exponentiation on functions and relations has too general a domain, namely $('a \times 'b)$ *set* and $'a \Rightarrow 'b$. Explicit type constraints may therefore be necessary. For example, $\text{range } (f \wedge n) = A$ and $\text{Range } (R \wedge n) = B$ need constraints.

Circumvent this problem for code generation:

primrec

$\text{fun-pow} :: \text{nat} \Rightarrow ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a$

where

$\text{fun-pow } 0 f = \text{id}$
 $| \text{fun-pow } (\text{Suc } n) f = f \circ \text{fun-pow } n f$

lemma funpow-fun-pow [*code unfold*]: $f \wedge n = \text{fun-pow } n f$
 $\langle \text{proof} \rangle$

lemma funpow-add: $f \wedge (m+n) = f \wedge m \circ f \wedge n$
 $\langle \text{proof} \rangle$

lemma funpow-swap1: $f((f \wedge n) x) = (f \wedge n)(f x)$
 $\langle \text{proof} \rangle$

lemma rel-pow-1 [*simp*]:

fixes $R :: ('a * 'a)_{\text{set}}$

shows $R \wedge 1 = R$

$\langle \text{proof} \rangle$

lemma rel-pow-0-I: $(x, x) : R \wedge 0$
 $\langle \text{proof} \rangle$

lemma rel-pow-Suc-I: $[(x, y) : R \wedge n; (y, z) : R] \implies (x, z) : R \wedge (\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma rel-pow-Suc-I2:

$(x, y) : R \implies (y, z) : R \wedge n \implies (x, z) : R \wedge (\text{Suc } n)$

$\langle \text{proof} \rangle$

lemma *rel-pow-0-E*: $\llbracket (x,y) : R^{\wedge}0; x=y \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *rel-pow-Suc-E*:
 $\llbracket (x,z) : R^{\wedge}(Suc\ n); !!y. \llbracket (x,y) : R^{\wedge}n; (y,z) : R \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *rel-pow-E*:
 $\llbracket (x,z) : R^{\wedge}n; \llbracket n=0; x=z \rrbracket \implies P; !!y\ m. \llbracket n = Suc\ m; (x,y) : R^{\wedge}m; (y,z) : R \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *rel-pow-Suc-D2*:
 $(x, z) : R^{\wedge}(Suc\ n) \implies (\exists y. (x,y) : R \ \& \ (y,z) : R^{\wedge}n)$
 $\langle proof \rangle$

lemma *rel-pow-Suc-D2'*:
 $\forall x\ y\ z. (x,y) : R^{\wedge}n \ \& \ (y,z) : R \dashrightarrow (\exists w. (x,w) : R \ \& \ (w,z) : R^{\wedge}n)$
 $\langle proof \rangle$

lemma *rel-pow-E2*:
 $\llbracket (x,z) : R^{\wedge}n; \llbracket n=0; x=z \rrbracket \implies P; !!y\ m. \llbracket n = Suc\ m; (x,y) : R; (y,z) : R^{\wedge}m \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *rtrancl-imp-UN-rel-pow*: $!!p. p : R^{\wedge}* \implies p : (UN\ n. R^{\wedge}n)$
 $\langle proof \rangle$

lemma *rel-pow-imp-rtrancl*: $!!p. p : R^{\wedge}n \implies p : R^{\wedge}*$
 $\langle proof \rangle$

lemma *rtrancl-is-UN-rel-pow*: $R^{\wedge}* = (UN\ n. R^{\wedge}n)$
 $\langle proof \rangle$

lemma *tranc1-power*:
 $x \in r^{\wedge}+ = (\exists n > 0. x \in r^{\wedge}n)$
 $\langle proof \rangle$

lemma *single-valued-rel-pow*:
 $!!r::('a * 'a) set. single-valued\ r \implies single-valued\ (r^{\wedge}n)$
 $\langle proof \rangle$

$\langle ML \rangle$

end

29 Equiv-Relations: Equivalence Relations in Higher-Order Set Theory

theory *Equiv-Relations*
imports *Finite-Set Relation Plain*
begin

29.1 Equivalence relations

locale *equiv* =
fixes *A* **and** *r*
assumes *refl-on*: *refl-on A r*
and *sym*: *sym r*
and *trans*: *trans r*

Suppes, Theorem 70: *r* is an equiv relation iff $r^{-1} \circ r = r$.

First half: *equiv A r* $\implies r^{-1} \circ r = r$.

lemma *sym-trans-comp-subset*:
 $\text{sym } r \implies \text{trans } r \implies r^{-1} \circ r \subseteq r$
 $\langle \text{proof} \rangle$

lemma *refl-on-comp-subset*: *refl-on A r* $\implies r \subseteq r^{-1} \circ r$
 $\langle \text{proof} \rangle$

lemma *equiv-comp-eq*: *equiv A r* $\implies r^{-1} \circ r = r$
 $\langle \text{proof} \rangle$

Second half.

lemma *comp-equivI*:
 $r^{-1} \circ r = r \implies \text{Domain } r = A \implies \text{equiv } A r$
 $\langle \text{proof} \rangle$

29.2 Equivalence classes

lemma *equiv-class-subset*:
 $\text{equiv } A r \implies (a, b) \in r \implies r''\{a\} \subseteq r''\{b\}$
 — lemma for the next result
 $\langle \text{proof} \rangle$

theorem *equiv-class-eq*: *equiv A r* $\implies (a, b) \in r \implies r''\{a\} = r''\{b\}$
 $\langle \text{proof} \rangle$

lemma *equiv-class-self*: *equiv A r* $\implies a \in A \implies a \in r''\{a\}$
 $\langle \text{proof} \rangle$

lemma *subset-equiv-class*:
 $\text{equiv } A r \implies r''\{b\} \subseteq r''\{a\} \implies b \in A \implies (a, b) \in r$
 — lemma for the next result
 $\langle \text{proof} \rangle$

lemma *eq-equiv-class*:

$$r^{\{\{a\}\}} = r^{\{\{b\}\}} \implies \text{equiv } A \ r \implies b \in A \implies (a, b) \in r$$

<proof>

lemma *equiv-class-nondisjoint*:

$$\text{equiv } A \ r \implies x \in (r^{\{\{a\}\}} \cap r^{\{\{b\}\}}) \implies (a, b) \in r$$

<proof>

lemma *equiv-type*: $\text{equiv } A \ r \implies r \subseteq A \times A$

<proof>

theorem *equiv-class-eq-iff*:

$$\text{equiv } A \ r \implies ((x, y) \in r) = (r^{\{\{x\}\}} = r^{\{\{y\}\}} \ \& \ x \in A \ \& \ y \in A)$$

<proof>

theorem *eq-equiv-class-iff*:

$$\text{equiv } A \ r \implies x \in A \implies y \in A \implies (r^{\{\{x\}\}} = r^{\{\{y\}\}}) = ((x, y) \in r)$$

<proof>

29.3 Quotients

definition *quotient* :: $'a \text{ set} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow 'a \text{ set set}$ (**infixl** $'/'/$ 90) **where**
 $[code \ del]: A//r = (\bigcup x \in A. \{r^{\{\{x\}\}}\})$ — set of equiv classes

lemma *quotientI*: $x \in A \implies r^{\{\{x\}\}} \in A//r$

<proof>

lemma *quotientE*:

$$X \in A//r \implies (!x. X = r^{\{\{x\}\}} \implies x \in A \implies P) \implies P$$

<proof>

lemma *Union-quotient*: $\text{equiv } A \ r \implies \text{Union } (A//r) = A$

<proof>

lemma *quotient-disj*:

$$\text{equiv } A \ r \implies X \in A//r \implies Y \in A//r \implies X = Y \mid (X \cap Y = \{\})$$

<proof>

lemma *quotient-eqI*:

$$[\text{equiv } A \ r; X \in A//r; Y \in A//r; x \in X; y \in Y; (x, y) \in r] \implies X = Y$$

<proof>

lemma *quotient-eq-iff*:

$$[\text{equiv } A \ r; X \in A//r; Y \in A//r; x \in X; y \in Y] \implies (X = Y) = ((x, y) \in r)$$

<proof>

lemma *eq-equiv-class-iff2*:

$\llbracket \text{equiv } A \ r; x \in A; y \in A \rrbracket \implies (\{x\}/r = \{y\}/r) = ((x,y) : r)$
 $\langle \text{proof} \rangle$

lemma *quotient-empty* [simp]: $\{\}/r = \{\}$
 $\langle \text{proof} \rangle$

lemma *quotient-is-empty* [iff]: $(A/r = \{\}) = (A = \{\})$
 $\langle \text{proof} \rangle$

lemma *quotient-is-empty2* [iff]: $(\{\} = A/r) = (A = \{\})$
 $\langle \text{proof} \rangle$

lemma *singleton-quotient*: $\{x\}/r = \{r \text{ `` } \{x\}\}$
 $\langle \text{proof} \rangle$

lemma *quotient-diff1*:
 $\llbracket \text{inj-on } (\%a. \{a\}/r) \ A; a \in A \rrbracket \implies (A - \{a\})/r = A/r - \{a\}/r$
 $\langle \text{proof} \rangle$

29.4 Defining unary operations upon equivalence classes

A congruence-preserving function

locale *congruent* =
 fixes r and f
 assumes *congruent*: $(y,z) \in r \implies f\ y = f\ z$

abbreviation
 $\text{RESPECTS} :: ('a \implies 'b) \implies ('a * 'a) \text{ set} \implies \text{bool}$
 (infixr *respects* 80) **where**
 $f \text{ respects } r == \text{congruent } r\ f$

lemma *UN-constant-eq*: $a \in A \implies \forall y \in A. f\ y = c \implies (\bigcup y \in A. f(y)) = c$
 — lemma required to prove *UN-equiv-class*
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class*:
 $\text{equiv } A \ r \implies f \text{ respects } r \implies a \in A$
 $\implies (\bigcup x \in r \text{ `` } \{a\}. f\ x) = f\ a$
 — Conversion rule
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class-type*:
 $\text{equiv } A \ r \implies f \text{ respects } r \implies X \in A/r \implies$
 $(!!x. x \in A \implies f\ x \in B) \implies (\bigcup x \in X. f\ x) \in B$
 $\langle \text{proof} \rangle$

Sufficient conditions for injectiveness. Could weaken premises! major premise could be an inclusion; bcong could be $!!y. y \in A \implies f y \in B$.

lemma *UN-equiv-class-inject*:

equiv $A \ r \implies f \text{ respects } r \implies$
 $(\bigcup x \in X. f x) = (\bigcup y \in Y. f y) \implies X \in A//r \implies Y \in A//r$
 $\implies (!!x y. x \in A \implies y \in A \implies f x = f y \implies (x, y) \in r)$
 $\implies X = Y$
 $\langle \text{proof} \rangle$

29.5 Defining binary operations upon equivalence classes

A congruence-preserving function of two arguments

locale *congruent2* =

fixes $r1$ **and** $r2$ **and** f

assumes *congruent2*:

$(y1, z1) \in r1 \implies (y2, z2) \in r2 \implies f y1 y2 = f z1 z2$

Abbreviation for the common case where the relations are identical

abbreviation

RESPECTS2:: $['a \implies 'a \implies 'b, ('a * 'a) \text{ set}] \implies \text{bool}$

(infixr *respects2* 80) **where**

$f \text{ respects2 } r == \text{congruent2 } r \ r \ f$

lemma *congruent2-implies-congruent*:

equiv $A \ r1 \implies \text{congruent2 } r1 \ r2 \ f \implies a \in A \implies \text{congruent } r2 \ (f a)$
 $\langle \text{proof} \rangle$

lemma *congruent2-implies-congruent-UN*:

equiv $A1 \ r1 \implies \text{equiv } A2 \ r2 \implies \text{congruent2 } r1 \ r2 \ f \implies a \in A2 \implies$
 $\text{congruent } r1 \ (\lambda x1. \bigcup x2 \in r2. \{a\}. f x1 x2)$
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class2*:

equiv $A1 \ r1 \implies \text{equiv } A2 \ r2 \implies \text{congruent2 } r1 \ r2 \ f \implies a1 \in A1 \implies a2$
 $\in A2$
 $\implies (\bigcup x1 \in r1. \{a1\}. \bigcup x2 \in r2. \{a2\}. f x1 x2) = f a1 a2$
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class-type2*:

equiv $A1 \ r1 \implies \text{equiv } A2 \ r2 \implies \text{congruent2 } r1 \ r2 \ f$
 $\implies X1 \in A1//r1 \implies X2 \in A2//r2$
 $\implies (!!x1 x2. x1 \in A1 \implies x2 \in A2 \implies f x1 x2 \in B)$
 $\implies (\bigcup x1 \in X1. \bigcup x2 \in X2. f x1 x2) \in B$
 $\langle \text{proof} \rangle$

lemma *UN-UN-split-split-eq*:

$(\bigcup (x1, x2) \in X. \bigcup (y1, y2) \in Y. A \ x1 \ x2 \ y1 \ y2) =$

$(\bigcup x \in X. \bigcup y \in Y. (\lambda(x1, x2). (\lambda(y1, y2). A\ x1\ x2\ y1\ y2)\ y)\ x)$
 — Allows a natural expression of binary operators,
 — without explicit calls to *split*
 $\langle proof \rangle$

lemma *congruent2I*:

equiv A1 r1 ==> equiv A2 r2
 $==> (!y\ z\ w. w \in A2 ==> (y, z) \in r1 ==> f\ y\ w = f\ z\ w)$
 $==> (!y\ z\ w. w \in A1 ==> (y, z) \in r2 ==> f\ w\ y = f\ w\ z)$
 $==> congruent2\ r1\ r2\ f$
 — Suggested by John Harrison – the two subproofs may be
 — *much* simpler than the direct proof.
 $\langle proof \rangle$

lemma *congruent2-commuteI*:

assumes equivA: equiv A r
and commute: !y z. y ∈ A ==> z ∈ A ==> f y z = f z y
and cong: !y z w. w ∈ A ==> (y, z) ∈ r ==> f w y = f w z
shows f respects2 r
 $\langle proof \rangle$

29.6 Quotients and finiteness

Suggested by Florian Kammüller

lemma *finite-quotient*: *finite A ==> r ⊆ A × A ==> finite (A//r)*
 — recall *equiv ?A ?r ==> ?r ⊆ ?A × ?A*
 $\langle proof \rangle$

lemma *finite-equiv-class*:

finite A ==> r ⊆ A × A ==> X ∈ A//r ==> finite X
 $\langle proof \rangle$

lemma *equiv-imp-dvd-card*:

finite A ==> equiv A r ==> ∀ X ∈ A//r. k dvd card X
 $==> k\ dvd\ card\ A$
 $\langle proof \rangle$

lemma *card-quotient-disjoint*:

$\llbracket finite\ A;\ inj-on\ (\lambda x. \{x\}\ /\ /\ r)\ A \rrbracket ==> card(A//r) = card\ A$
 $\langle proof \rangle$

end

30 Int: The Integers as Equivalence Classes over Pairs of Natural Numbers

theory *Int*

```

imports Equiv-Relations Nat Wellfounded
uses
  (Tools/numeral.ML)
  (Tools/numeral-syntax.ML)
  ~~/src/Provers/Arith/assoc-fold.ML
  ~~/src/Provers/Arith/cancel-numerals.ML
  ~~/src/Provers/Arith/combine-numerals.ML
  (Tools/int-arith.ML)
begin

```

30.1 The equivalence relation underlying the integers

definition *intrel* :: $((nat \times nat) \times (nat \times nat))$ set **where**
 $[code\ del]: intrel = \{(x, y), (u, v) \mid x\ y\ u\ v.\ x + v = u + y\}$

typedef (*Integ*)
int = *UNIV* // *intrel*
 $\langle proof \rangle$

instantiation *int* :: $\{zero, one, plus, minus, uminus, times, ord, abs, sgn\}$
begin

definition
Zero-int-def $[code\ del]: 0 = Abs-Integ\ (intrel\ \{(0, 0)\})$

definition
One-int-def $[code\ del]: 1 = Abs-Integ\ (intrel\ \{(1, 0)\})$

definition
add-int-def $[code\ del]: z + w = Abs-Integ$
 $(\bigcup (x, y) \in Rep-Integ\ z.\ \bigcup (u, v) \in Rep-Integ\ w.$
 $intrel\ \{(x + u, y + v)\})$

definition
minus-int-def $[code\ del]:$
 $- z = Abs-Integ\ (\bigcup (x, y) \in Rep-Integ\ z.\ intrel\ \{(y, x)\})$

definition
diff-int-def $[code\ del]: z - w = z + (-w :: int)$

definition
mult-int-def $[code\ del]: z * w = Abs-Integ$
 $(\bigcup (x, y) \in Rep-Integ\ z.\ \bigcup (u, v) \in Rep-Integ\ w.$
 $intrel\ \{(x*u + y*v, x*v + y*u)\})$

definition
le-int-def $[code\ del]:$
 $z \leq w \iff (\exists x\ y\ u\ v.\ x + v \leq u + y \wedge (x, y) \in Rep-Integ\ z \wedge (u, v) \in Rep-Integ\ w)$

definition

less-int-def [code del]: $(z::int) < w \longleftrightarrow z \leq w \wedge z \neq w$

definition

zabs-def: $|i::int| = (if\ i < 0\ then\ -\ i\ else\ i)$

definition

zsgn-def: $sgn\ (i::int) = (if\ i=0\ then\ 0\ else\ if\ 0<i\ then\ 1\ else\ -\ 1)$

instance $\langle proof \rangle$

end

30.2 Construction of the Integers

lemma *intrel-iff* [simp]: $((x,y),(u,v)) \in intrel = (x+v = u+y)$
 $\langle proof \rangle$

lemma *equiv-intrel*: *equiv UNIV intrel*
 $\langle proof \rangle$

Reduces equality of equivalence classes to the *intrel* relation: $(intrel\ \{\!\{x\}\!\} = intrel\ \{\!\{y\}\!\}) = ((x, y) \in intrel)$

lemmas *equiv-intrel-iff* [simp] = *eq-equiv-class-iff* [OF *equiv-intrel UNIV-I UNIV-I*]

All equivalence classes belong to set of representatives

lemma [simp]: $intrel\ \{\!\{(x,y)\}\!\} \in Integ$
 $\langle proof \rangle$

Reduces equality on abstractions to equality on representatives: $\llbracket x \in Integ; y \in Integ \rrbracket \implies (Abs-Integ\ x = Abs-Integ\ y) = (x = y)$

declare *Abs-Integ-inject* [simp,noatp] *Abs-Integ-inverse* [simp,noatp]

Case analysis on the representation of an integer as an equivalence class of pairs of naturals.

lemma *eq-Abs-Integ* [case-names *Abs-Integ*, cases type: *int*]:
 $(!!x\ y.\ z = Abs-Integ(intrel\ \{\!\{(x,y)\}\!\}) \implies P) \implies P$
 $\langle proof \rangle$

30.3 Arithmetic Operations

lemma *minus*: $- Abs-Integ(intrel\ \{\!\{(x,y)\}\!\}) = Abs-Integ(intrel\ \{\!\{(y,x)\}\!\})$
 $\langle proof \rangle$

lemma *add*:

$Abs-Integ\ (intrel\ \{\!\{(x,y)\}\!\}) + Abs-Integ\ (intrel\ \{\!\{(u,v)\}\!\}) =$
 $Abs-Integ\ (intrel\ \{\!\{(x+u, y+v)\}\!\})$

$\langle proof \rangle$

Congruence property for multiplication

lemma *mult-congruent2*:

$(\%p1\ p2. (\%(x,y). (\%(u,v). \text{intrel}''\{(x*u + y*v, x*v + y*u)\})\ p2)\ p1)$
respects2 intrel

$\langle proof \rangle$

lemma *mult*:

$\text{Abs-Integ}((\text{intrel}''\{(x,y)\})) * \text{Abs-Integ}((\text{intrel}''\{(u,v)\})) =$
 $\text{Abs-Integ}(\text{intrel}''\{(x*u + y*v, x*v + y*u)\})$

$\langle proof \rangle$

The integers form a *comm-ring-1*

instance *int :: comm-ring-1*

$\langle proof \rangle$

lemma *int-def*: $of\text{-}nat\ m = \text{Abs-Integ}(\text{intrel}''\{(m, 0)\})$

$\langle proof \rangle$

30.4 The \leq Ordering

lemma *le*:

$(\text{Abs-Integ}(\text{intrel}''\{(x,y)\}) \leq \text{Abs-Integ}(\text{intrel}''\{(u,v)\})) = (x+v \leq u+y)$

$\langle proof \rangle$

lemma *less*:

$(\text{Abs-Integ}(\text{intrel}''\{(x,y)\}) < \text{Abs-Integ}(\text{intrel}''\{(u,v)\})) = (x+v < u+y)$

$\langle proof \rangle$

instance *int :: linorder*

$\langle proof \rangle$

instantiation *int :: distrib-lattice*

begin

definition

$(inf :: int \Rightarrow int \Rightarrow int) = min$

definition

$(sup :: int \Rightarrow int \Rightarrow int) = max$

instance

$\langle proof \rangle$

end

instance *int :: pordered-cancel-ab-semigroup-add*

$\langle proof \rangle$

Strict Monotonicity of Multiplication

strict, in 1st argument; proof is by induction on $k \geq 0$

lemma *zmult-zless-mono2-lemma*:

$(i :: \text{int}) < j \implies 0 < k \implies \text{of-nat } k * i < \text{of-nat } k * j$
 $\langle \text{proof} \rangle$

lemma *zero-le-imp-eq-int*: $(0 :: \text{int}) \leq k \implies \exists n. k = \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *zero-less-imp-eq-int*: $(0 :: \text{int}) < k \implies \exists n > 0. k = \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *zmult-zless-mono2*: $[\mid i < j; (0 :: \text{int}) < k \mid] \implies k * i < k * j$
 $\langle \text{proof} \rangle$

The integers form an ordered integral domain

instance *int :: ordered-idom*
 $\langle \text{proof} \rangle$

instance *int :: lordered-ring*
 $\langle \text{proof} \rangle$

lemma *zless-imp-add1-zle*: $w < z \implies w + (1 :: \text{int}) \leq z$
 $\langle \text{proof} \rangle$

lemma *zless-iff-Suc-zadd*:
 $(w :: \text{int}) < z \iff (\exists n. z = w + \text{of-nat } (\text{Suc } n))$
 $\langle \text{proof} \rangle$

lemmas *int-distrib* =
left-distrib [*of z1 :: int z2 w, standard*]
right-distrib [*of w :: int z1 z2, standard*]
left-diff-distrib [*of z1 :: int z2 w, standard*]
right-diff-distrib [*of w :: int z1 z2, standard*]

30.5 Embedding of the Integers into any *ring-1*: *of-int*

context *ring-1*
begin

definition

of-int :: *int* \Rightarrow 'a

where

[code del]: *of-int* *z* = *contents* ($\bigcup (i, j) \in \text{Rep-Integ } z. \{ \text{of-nat } i - \text{of-nat } j \}$)

lemma *of-int*: *of-int* (*Abs-Integ* (*intrel* “ $\{(i, j)\}$ ”)) = *of-nat* *i* - *of-nat* *j*
 $\langle \text{proof} \rangle$

lemma *of-int-0* [*simp*]: *of-int* 0 = 0
 ⟨*proof*⟩

lemma *of-int-1* [*simp*]: *of-int* 1 = 1
 ⟨*proof*⟩

lemma *of-int-add* [*simp*]: *of-int* (w+z) = *of-int* w + *of-int* z
 ⟨*proof*⟩

lemma *of-int-minus* [*simp*]: *of-int* (−z) = − (*of-int* z)
 ⟨*proof*⟩

lemma *of-int-diff* [*simp*]: *of-int* (w − z) = *of-int* w − *of-int* z
 ⟨*proof*⟩

lemma *of-int-mult* [*simp*]: *of-int* (w*z) = *of-int* w * *of-int* z
 ⟨*proof*⟩

Collapse nested embeddings

lemma *of-int-of-nat-eq* [*simp*]: *of-int* (*of-nat* n) = *of-nat* n
 ⟨*proof*⟩

end

context *ordered-idom*
begin

lemma *of-int-le-iff* [*simp*]:
of-int w ≤ *of-int* z \longleftrightarrow w ≤ z
 ⟨*proof*⟩

Special cases where either operand is zero

lemmas *of-int-0-le-iff* [*simp*] = *of-int-le-iff* [*of* 0, *simplified*]
lemmas *of-int-le-0-iff* [*simp*] = *of-int-le-iff* [*of* - 0, *simplified*]

lemma *of-int-less-iff* [*simp*]:
of-int w < *of-int* z \longleftrightarrow w < z
 ⟨*proof*⟩

Special cases where either operand is zero

lemmas *of-int-0-less-iff* [*simp*] = *of-int-less-iff* [*of* 0, *simplified*]
lemmas *of-int-less-0-iff* [*simp*] = *of-int-less-iff* [*of* - 0, *simplified*]

end

Class for unital rings with characteristic zero. Includes non-ordered rings like the complex numbers.

class *ring-char-0* = *ring-1* + *semiring-char-0*

begin

lemma *of-int-eq-iff* [simp]:
 $of\text{-}int\ w = of\text{-}int\ z \longleftrightarrow w = z$
 ⟨proof⟩

Special cases where either operand is zero

lemmas *of-int-0-eq-iff* [simp] = *of-int-eq-iff* [of 0, simplified]
lemmas *of-int-eq-0-iff* [simp] = *of-int-eq-iff* [of - 0, simplified]

end

Every *ordered-idom* has characteristic zero.

subclass (in *ordered-idom*) *ring-char-0* ⟨proof⟩

lemma *of-int-eq-id* [simp]: $of\text{-}int = id$
 ⟨proof⟩

30.6 Magnitude of an Integer, as a Natural Number: *nat*

definition

$nat :: int \Rightarrow nat$

where

[code del]: $nat\ z = contents\ (\bigcup (x, y) \in Rep\text{-}Integ\ z. \{x - y\})$

lemma *nat*: $nat\ (Abs\text{-}Integ\ (intrel''\{(x, y)\})) = x - y$
 ⟨proof⟩

lemma *nat-int* [simp]: $nat\ (of\text{-}nat\ n) = n$
 ⟨proof⟩

lemma *nat-zero* [simp]: $nat\ 0 = 0$
 ⟨proof⟩

lemma *int-nat-eq* [simp]: $of\text{-}nat\ (nat\ z) = (if\ 0 \leq z\ then\ z\ else\ 0)$
 ⟨proof⟩

corollary *nat-0-le*: $0 \leq z ==> of\text{-}nat\ (nat\ z) = z$
 ⟨proof⟩

lemma *nat-le-0* [simp]: $z \leq 0 ==> nat\ z = 0$
 ⟨proof⟩

lemma *nat-le-eq-zle*: $0 < w \mid 0 \leq z ==> (nat\ w \leq nat\ z) = (w \leq z)$
 ⟨proof⟩

An alternative condition is $(0::'a) \leq w$

corollary *nat-mono-iff*: $0 < z ==> (nat\ w < nat\ z) = (w < z)$
 ⟨proof⟩

corollary *nat-less-eq-zless*: $0 \leq w \implies (\text{nat } w < \text{nat } z) = (w < z)$
 $\langle \text{proof} \rangle$

lemma *zless-nat-conj* [*simp*]: $(\text{nat } w < \text{nat } z) = (0 < z \ \& \ w < z)$
 $\langle \text{proof} \rangle$

lemma *nonneg-eq-int*:
fixes $z :: \text{int}$
assumes $0 \leq z$ **and** $\bigwedge m. z = \text{of-nat } m \implies P$
shows P
 $\langle \text{proof} \rangle$

lemma *nat-eq-iff*: $(\text{nat } w = m) = (\text{if } 0 \leq w \text{ then } w = \text{of-nat } m \text{ else } m=0)$
 $\langle \text{proof} \rangle$

corollary *nat-eq-iff2*: $(m = \text{nat } w) = (\text{if } 0 \leq w \text{ then } w = \text{of-nat } m \text{ else } m=0)$
 $\langle \text{proof} \rangle$

lemma *nat-less-iff*: $0 \leq w \implies (\text{nat } w < m) = (w < \text{of-nat } m)$
 $\langle \text{proof} \rangle$

lemma *nat-0-iff* [*simp*]: $\text{nat}(i::\text{int}) = 0 \longleftrightarrow i \leq 0$
 $\langle \text{proof} \rangle$

lemma *int-eq-iff*: $(\text{of-nat } m = z) = (m = \text{nat } z \ \& \ 0 \leq z)$
 $\langle \text{proof} \rangle$

lemma *zero-less-nat-eq* [*simp*]: $(0 < \text{nat } z) = (0 < z)$
 $\langle \text{proof} \rangle$

lemma *nat-add-distrib*:
 $\llbracket (0::\text{int}) \leq z; \ 0 \leq z' \rrbracket \implies \text{nat } (z+z') = \text{nat } z + \text{nat } z'$
 $\langle \text{proof} \rangle$

lemma *nat-diff-distrib*:
 $\llbracket (0::\text{int}) \leq z'; \ z' \leq z \rrbracket \implies \text{nat } (z-z') = \text{nat } z - \text{nat } z'$
 $\langle \text{proof} \rangle$

lemma *nat-zminus-int* [*simp*]: $\text{nat } (- (\text{of-nat } n)) = 0$
 $\langle \text{proof} \rangle$

lemma *zless-nat-eq-int-zless*: $(m < \text{nat } z) = (\text{of-nat } m < z)$
 $\langle \text{proof} \rangle$

context *ring-1*
begin

lemma *of-nat-nat*: $0 \leq z \implies \text{of-nat } (\text{nat } z) = \text{of-int } z$

$\langle \text{proof} \rangle$

end

For termination proofs:

lemma *measure-function-int*[*measure-function*]: *is-measure* (*nat o abs*) $\langle \text{proof} \rangle$

30.7 Lemmas about the Function *of-nat* and Orderings

lemma *negative-zless-0*: $-(\text{of-nat } (\text{Suc } n)) < (0 :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *negative-zless* [*iff*]: $-(\text{of-nat } (\text{Suc } n)) < (\text{of-nat } m :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *negative-zle-0*: $-\text{of-nat } n \leq (0 :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *negative-zle* [*iff*]: $-\text{of-nat } n \leq (\text{of-nat } m :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *not-zle-0-negative* [*simp*]: $\sim (0 \leq -(\text{of-nat } (\text{Suc } n)) :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *int-zle-neg*: $((\text{of-nat } n :: \text{int}) \leq -\text{of-nat } m) = (n = 0 \ \& \ m = 0)$
 $\langle \text{proof} \rangle$

lemma *not-int-zless-negative* [*simp*]: $\sim ((\text{of-nat } n :: \text{int}) < -\text{of-nat } m)$
 $\langle \text{proof} \rangle$

lemma *negative-eq-positive* [*simp*]: $((-\text{of-nat } n :: \text{int}) = \text{of-nat } m) = (n = 0 \ \& \ m = 0)$
 $\langle \text{proof} \rangle$

lemma *zle-iff-zadd*: $(w :: \text{int}) \leq z \longleftrightarrow (\exists n. z = w + \text{of-nat } n)$
 $\langle \text{proof} \rangle$

lemma *zadd-int-left*: $\text{of-nat } m + (\text{of-nat } n + z) = \text{of-nat } (m + n) + (z :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *int-Suc0-eq-1*: $\text{of-nat } (\text{Suc } 0) = (1 :: \text{int})$
 $\langle \text{proof} \rangle$

This version is proved for all ordered rings, not just integers! It is proved here because attribute *arith-split* is not available in theory *Ring-and-Field*. But is it really better than just rewriting with *abs-if*?

lemma *abs-split* [*arith-split, noatp*]:

$P(\text{abs}(a :: 'a :: \text{ordered-idom})) = ((0 \leq a \longrightarrow P a) \ \& \ (a < 0 \longrightarrow P(-a)))$
 $\langle \text{proof} \rangle$

lemma *negD*: $(x :: \text{int}) < 0 \implies \exists n. x = - (\text{of-nat } (\text{Suc } n))$
 $\langle \text{proof} \rangle$

30.8 Cases and induction

Now we replace the case analysis rule by a more conventional one: whether an integer is negative or not.

theorem *int-cases* [*cases type: int, case-names nonneg neg*]:
 $[[!! n. (z :: \text{int}) = \text{of-nat } n \implies P; !! n. z = - (\text{of-nat } (\text{Suc } n)) \implies P] \implies P$
 $\langle \text{proof} \rangle$

theorem *int-induct* [*induct type: int, case-names nonneg neg*]:
 $[[!! n. P (\text{of-nat } n :: \text{int}); !! n. P (- (\text{of-nat } (\text{Suc } n)))] \implies P z$
 $\langle \text{proof} \rangle$

Contributed by Brian Huffman

theorem *int-diff-cases*:
obtains $(\text{diff}) m n$ **where** $(z :: \text{int}) = \text{of-nat } m - \text{of-nat } n$
 $\langle \text{proof} \rangle$

30.9 Binary representation

This formalization defines binary arithmetic in terms of the integers rather than using a datatype. This avoids multiple representations (leading zeroes, etc.) See *ZF/Tools/twos-compl.ML*, function *int-of-binary*, for the numerical interpretation.

The representation expects that $(m \bmod 2)$ is 0 or 1, even if m is negative; For instance, $-5 \text{ div } 2 = -3$ and $-5 \bmod 2 = 1$; thus $-5 = (-3)*2 + 1$.

This two’s complement binary representation derives from the paper “An Efficient Representation of Arithmetic for Term Rewriting” by Dave Cohen and Phil Watson, *Rewriting Techniques and Applications*, Springer LNCS 488 (240-251), 1991.

30.9.1 The constructors *Bit0*, *Bit1*, *Pls* and *Min*

definition
 $\text{Pls} :: \text{int}$ **where**
 $[\text{code del}]: \text{Pls} = 0$

definition
 $\text{Min} :: \text{int}$ **where**
 $[\text{code del}]: \text{Min} = - 1$

definition

Bit0 :: *int* \Rightarrow *int* **where**
 [code del]: *Bit0* *k* = *k* + *k*

definition

Bit1 :: *int* \Rightarrow *int* **where**
 [code del]: *Bit1* *k* = 1 + *k* + *k*

class *number* = — for numeric types: nat, int, real, ...
fixes *number-of* :: *int* \Rightarrow 'a

$\langle ML \rangle$

syntax

-*Numeral* :: *num-const* \Rightarrow 'a (-)

$\langle ML \rangle$

abbreviation

Numeral0 \equiv *number-of* *Pls*

abbreviation

Numeral1 \equiv *number-of* (*Bit1* *Pls*)

lemma *Let-number-of* [simp]: *Let* (*number-of* *v*) *f* = *f* (*number-of* *v*)
 — Unfold all *lets* involving constants
 $\langle proof \rangle$

definition

succ :: *int* \Rightarrow *int* **where**
 [code del]: *succ* *k* = *k* + 1

definition

pred :: *int* \Rightarrow *int* **where**
 [code del]: *pred* *k* = *k* - 1

lemmas

max-number-of [simp] = *max-def*
 [of *number-of* *u* *number-of* *v*, *standard*, *simp*]

and

min-number-of [simp] = *min-def*
 [of *number-of* *u* *number-of* *v*, *standard*, *simp*]
 — unfolding *minx* and *max* on numerals

lemmas *numeral-simps* =

succ-def *pred-def* *Pls-def* *Min-def* *Bit0-def* *Bit1-def*

Removal of leading zeroes

lemma *Bit0-Pls* [simp, code post]:
Bit0 *Pls* = *Pls*

$\langle \text{proof} \rangle$

lemma *Bit1-Min* [*simp*, *code post*]:
 $\text{Bit1 } \text{Min} = \text{Min}$
 $\langle \text{proof} \rangle$

lemmas *normalize-bin-simps* =
 $\text{Bit0-Pls } \text{Bit1-Min}$

30.9.2 Successor and predecessor functions

Successor

lemma *succ-Pls*:
 $\text{succ } \text{Pls} = \text{Bit1 } \text{Pls}$
 $\langle \text{proof} \rangle$

lemma *succ-Min*:
 $\text{succ } \text{Min} = \text{Pls}$
 $\langle \text{proof} \rangle$

lemma *succ-Bit0*:
 $\text{succ } (\text{Bit0 } k) = \text{Bit1 } k$
 $\langle \text{proof} \rangle$

lemma *succ-Bit1*:
 $\text{succ } (\text{Bit1 } k) = \text{Bit0 } (\text{succ } k)$
 $\langle \text{proof} \rangle$

lemmas *succ-bin-simps* [*simp*] =
 $\text{succ-Pls } \text{succ-Min } \text{succ-Bit0 } \text{succ-Bit1}$

Predecessor

lemma *pred-Pls*:
 $\text{pred } \text{Pls} = \text{Min}$
 $\langle \text{proof} \rangle$

lemma *pred-Min*:
 $\text{pred } \text{Min} = \text{Bit0 } \text{Min}$
 $\langle \text{proof} \rangle$

lemma *pred-Bit0*:
 $\text{pred } (\text{Bit0 } k) = \text{Bit1 } (\text{pred } k)$
 $\langle \text{proof} \rangle$

lemma *pred-Bit1*:
 $\text{pred } (\text{Bit1 } k) = \text{Bit0 } k$
 $\langle \text{proof} \rangle$

lemmas *pred-bin-simps* [*simp*] =

pred-Pls pred-Min pred-Bit0 pred-Bit1

30.9.3 Binary arithmetic

Addition

lemma *add-Pls*:

$$Pls + k = k$$

<proof>

lemma *add-Min*:

$$Min + k = pred\ k$$

<proof>

lemma *add-Bit0-Bit0*:

$$(Bit0\ k) + (Bit0\ l) = Bit0\ (k + l)$$

<proof>

lemma *add-Bit0-Bit1*:

$$(Bit0\ k) + (Bit1\ l) = Bit1\ (k + l)$$

<proof>

lemma *add-Bit1-Bit0*:

$$(Bit1\ k) + (Bit0\ l) = Bit1\ (k + l)$$

<proof>

lemma *add-Bit1-Bit1*:

$$(Bit1\ k) + (Bit1\ l) = Bit0\ (k + succ\ l)$$

<proof>

lemma *add-Pls-right*:

$$k + Pls = k$$

<proof>

lemma *add-Min-right*:

$$k + Min = pred\ k$$

<proof>

lemmas *add-bin-simps* [*simp*] =

add-Pls add-Min add-Pls-right add-Min-right

add-Bit0-Bit0 add-Bit0-Bit1 add-Bit1-Bit0 add-Bit1-Bit1

Negation

lemma *minus-Pls*:

$$- Pls = Pls$$

<proof>

lemma *minus-Min*:

$$- Min = Bit1\ Pls$$

<proof>

lemma *minus-Bit0*:

$$- (Bit0\ k) = Bit0\ (-\ k)$$

<proof>

lemma *minus-Bit1*:

$$- (Bit1\ k) = Bit1\ (pred\ (-\ k))$$

<proof>

lemmas *minus-bin-simps* [simp] =
minus-Pls minus-Min minus-Bit0 minus-Bit1

Subtraction

lemma *diff-bin-simps* [simp]:

$$\begin{aligned} k - Pls &= k \\ k - Min &= succ\ k \\ Pls - (Bit0\ l) &= Bit0\ (Pls - l) \\ Pls - (Bit1\ l) &= Bit1\ (Min - l) \\ Min - (Bit0\ l) &= Bit1\ (Min - l) \\ Min - (Bit1\ l) &= Bit0\ (Min - l) \\ (Bit0\ k) - (Bit0\ l) &= Bit0\ (k - l) \\ (Bit0\ k) - (Bit1\ l) &= Bit1\ (pred\ k - l) \\ (Bit1\ k) - (Bit0\ l) &= Bit1\ (k - l) \\ (Bit1\ k) - (Bit1\ l) &= Bit0\ (k - l) \end{aligned}$$

<proof>

Multiplication

lemma *mult-Pls*:

$$Pls * w = Pls$$

<proof>

lemma *mult-Min*:

$$Min * k = -\ k$$

<proof>

lemma *mult-Bit0*:

$$(Bit0\ k) * l = Bit0\ (k * l)$$

<proof>

lemma *mult-Bit1*:

$$(Bit1\ k) * l = (Bit0\ (k * l)) + l$$

<proof>

lemmas *mult-bin-simps* [simp] =
mult-Pls mult-Min mult-Bit0 mult-Bit1

30.9.4 Binary comparisons

Preliminaries

lemma *even-less-0-iff*:

$a + a < 0 \iff a < (0::'a::ordered-idom)$
 $\langle proof \rangle$

lemma *le-imp-0-less*:

assumes $le: 0 \leq z$
shows $(0::int) < 1 + z$
 $\langle proof \rangle$

lemma *odd-less-0-iff*:

$(1 + z + z < 0) = (z < (0::int))$
 $\langle proof \rangle$

lemma *bin-less-0-simps*:

$Pls < 0 \iff False$
 $Min < 0 \iff True$
 $Bit0\ w < 0 \iff w < 0$
 $Bit1\ w < 0 \iff w < 0$
 $\langle proof \rangle$

lemma *less-bin-lemma*: $k < l \iff k - l < (0::int)$

$\langle proof \rangle$

lemma *le-iff-pred-less*: $k \leq l \iff pred\ k < l$

$\langle proof \rangle$

lemma *succ-pred*: $succ\ (pred\ x) = x$

$\langle proof \rangle$

Less-than

lemma *less-bin-simps* [*simp*]:

$Pls < Pls \iff False$
 $Pls < Min \iff False$
 $Pls < Bit0\ k \iff Pls < k$
 $Pls < Bit1\ k \iff Pls \leq k$
 $Min < Pls \iff True$
 $Min < Min \iff False$
 $Min < Bit0\ k \iff Min < k$
 $Min < Bit1\ k \iff Min < k$
 $Bit0\ k < Pls \iff k < Pls$
 $Bit0\ k < Min \iff k \leq Min$
 $Bit1\ k < Pls \iff k < Pls$
 $Bit1\ k < Min \iff k < Min$
 $Bit0\ k < Bit0\ l \iff k < l$
 $Bit0\ k < Bit1\ l \iff k \leq l$
 $Bit1\ k < Bit0\ l \iff k < l$
 $Bit1\ k < Bit1\ l \iff k < l$
 $\langle proof \rangle$

Less-than-or-equal

lemma *le-bin-simps* [*simp*]:
 $Pls \leq Pls \longleftrightarrow True$
 $Pls \leq Min \longleftrightarrow False$
 $Pls \leq Bit0\ k \longleftrightarrow Pls \leq k$
 $Pls \leq Bit1\ k \longleftrightarrow Pls \leq k$
 $Min \leq Pls \longleftrightarrow True$
 $Min \leq Min \longleftrightarrow True$
 $Min \leq Bit0\ k \longleftrightarrow Min < k$
 $Min \leq Bit1\ k \longleftrightarrow Min \leq k$
 $Bit0\ k \leq Pls \longleftrightarrow k \leq Pls$
 $Bit0\ k \leq Min \longleftrightarrow k \leq Min$
 $Bit1\ k \leq Pls \longleftrightarrow k < Pls$
 $Bit1\ k \leq Min \longleftrightarrow k \leq Min$
 $Bit0\ k \leq Bit0\ l \longleftrightarrow k \leq l$
 $Bit0\ k \leq Bit1\ l \longleftrightarrow k \leq l$
 $Bit1\ k \leq Bit0\ l \longleftrightarrow k < l$
 $Bit1\ k \leq Bit1\ l \longleftrightarrow k \leq l$
 ⟨*proof*⟩

Equality

lemma *eq-bin-simps* [*simp*]:
 $Pls = Pls \longleftrightarrow True$
 $Pls = Min \longleftrightarrow False$
 $Pls = Bit0\ l \longleftrightarrow Pls = l$
 $Pls = Bit1\ l \longleftrightarrow False$
 $Min = Pls \longleftrightarrow False$
 $Min = Min \longleftrightarrow True$
 $Min = Bit0\ l \longleftrightarrow False$
 $Min = Bit1\ l \longleftrightarrow Min = l$
 $Bit0\ k = Pls \longleftrightarrow k = Pls$
 $Bit0\ k = Min \longleftrightarrow False$
 $Bit1\ k = Pls \longleftrightarrow False$
 $Bit1\ k = Min \longleftrightarrow k = Min$
 $Bit0\ k = Bit0\ l \longleftrightarrow k = l$
 $Bit0\ k = Bit1\ l \longleftrightarrow False$
 $Bit1\ k = Bit0\ l \longleftrightarrow False$
 $Bit1\ k = Bit1\ l \longleftrightarrow k = l$
 ⟨*proof*⟩

30.10 Converting Numerals to Rings: *number-of*

class *number-ring* = *number* + *comm-ring-1* +
assumes *number-of-eq*: *number-of* *k* = *of-int* *k*

self-embedding of the integers

instantiation *int* :: *number-ring*
begin

definition *int-number-of-def* [*code del*]:

$number-of\ w = (of-int\ w :: int)$

instance $\langle proof \rangle$

end

lemma *number-of-is-id*:

$number-of\ (k :: int) = k$
 $\langle proof \rangle$

lemma *number-of-succ*:

$number-of\ (succ\ k) = (1 + number-of\ k :: 'a::number-ring)$
 $\langle proof \rangle$

lemma *number-of-pred*:

$number-of\ (pred\ w) = (-\ 1 + number-of\ w :: 'a::number-ring)$
 $\langle proof \rangle$

lemma *number-of-minus*:

$number-of\ (uminus\ w) = (-\ (number-of\ w) :: 'a::number-ring)$
 $\langle proof \rangle$

lemma *number-of-add*:

$number-of\ (v + w) = (number-of\ v + number-of\ w :: 'a::number-ring)$
 $\langle proof \rangle$

lemma *number-of-diff*:

$number-of\ (v - w) = (number-of\ v - number-of\ w :: 'a::number-ring)$
 $\langle proof \rangle$

lemma *number-of-mult*:

$number-of\ (v * w) = (number-of\ v * number-of\ w :: 'a::number-ring)$
 $\langle proof \rangle$

The correctness of shifting. But it doesn't seem to give a measurable speed-up.

lemma *double-number-of-Bit0*:

$(1 + 1) * number-of\ w = (number-of\ (Bit0\ w) :: 'a::number-ring)$
 $\langle proof \rangle$

Converting numerals 0 and 1 to their abstract versions.

lemma *numeral-0-eq-0* [simp]:

$N numeral0 = (0 :: 'a::number-ring)$
 $\langle proof \rangle$

lemma *numeral-1-eq-1* [simp]:

$N numeral1 = (1 :: 'a::number-ring)$
 $\langle proof \rangle$

Special-case simplification for small constants.

Unary minus for the abstract constant 1. Cannot be inserted as a simplrule until later: it is *number-of-Min* re-oriented!

lemma *numeral-m1-eq-minus-1*:
 $(-1 :: 'a :: \text{number-ring}) = -\ 1$
 $\langle \text{proof} \rangle$

lemma *mult-minus1 [simp]*:
 $-1 * z = -(z :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *mult-minus1-right [simp]*:
 $z * -1 = -(z :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *minus-number-of-mult [simp]*:
 $-(\text{number-of } w) * z = \text{number-of } (\text{uminus } w) * (z :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

Subtraction

lemma *diff-number-of-eq*:
 $\text{number-of } v - \text{number-of } w =$
 $(\text{number-of } (v + \text{uminus } w) :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *number-of-Pls*:
 $\text{number-of } \text{Pls} = (0 :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *number-of-Min*:
 $\text{number-of } \text{Min} = (-\ 1 :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *number-of-Bit0*:
 $\text{number-of } (\text{Bit0 } w) = (0 :: 'a :: \text{number-ring}) + (\text{number-of } w) + (\text{number-of } w)$
 $\langle \text{proof} \rangle$

lemma *number-of-Bit1*:
 $\text{number-of } (\text{Bit1 } w) = (1 :: 'a :: \text{number-ring}) + (\text{number-of } w) + (\text{number-of } w)$
 $\langle \text{proof} \rangle$

30.10.1 Equality of Binary Numbers

First version by Norbert Voelker

definition
 $\text{iszero} :: 'a :: \text{semiring-1} \Rightarrow \text{bool}$

where

$$\text{iszero } z \longleftrightarrow z = 0$$

lemma *iszero-0*: *iszero 0*

<proof>

lemma *not-iszero-1*: $\sim \text{iszero } 1$

<proof>

lemma *eq-number-of-eq*:

$$((\text{number-of } x :: 'a :: \text{number-ring}) = \text{number-of } y) = \\ \text{iszero } (\text{number-of } (x + \text{uminus } y) :: 'a)$$

<proof>

lemma *iszero-number-of-Pls*:

$$\text{iszero } ((\text{number-of } \text{Pls}) :: 'a :: \text{number-ring})$$

<proof>

lemma *nonzero-number-of-Min*:

$$\sim \text{iszero } ((\text{number-of } \text{Min}) :: 'a :: \text{number-ring})$$

<proof>

30.10.2 Comparisons, for Ordered Rings

lemmas *double-eq-0-iff* = *double-zero*

lemma *odd-nonzero*:

$$1 + z + z \neq (0 :: \text{int})$$

<proof>

lemma *iszero-number-of-Bit0*:

$$\text{iszero } (\text{number-of } (\text{Bit0 } w) :: 'a) = \\ \text{iszero } (\text{number-of } w :: 'a :: \{\text{ring-char-0}, \text{number-ring}\})$$

<proof>

lemma *iszero-number-of-Bit1*:

$$\sim \text{iszero } (\text{number-of } (\text{Bit1 } w) :: 'a :: \{\text{ring-char-0}, \text{number-ring}\})$$

<proof>

lemmas *iszero-simps* =

$$\text{iszero-0 } \text{not-iszero-1} \\ \text{iszero-number-of-Pls } \text{nonzero-number-of-Min} \\ \text{iszero-number-of-Bit0 } \text{iszero-number-of-Bit1}$$

30.10.3 The Less-Than Relation

lemma *double-less-0-iff*:

$$(a + a < 0) = (a < (0 :: 'a :: \text{ordered-idom}))$$

<proof>

lemma *odd-less-0*:

$(1 + z + z < 0) = (z < (0::int))$
 $\langle proof \rangle$

Less-Than or Equals

Reduces $a \leq b$ to $\neg b < a$ for ALL numerals.

lemmas *le-number-of-eq-not-less* =

linorder-not-less [of number-of w number-of v, symmetric,
 standard]

Absolute value (*abs*)

lemma *abs-number-of*:

$abs(number-of\ x::'a::\{\text{ordered-idom}, \text{number-ring}\}) =$
 $(if\ number-of\ x < (0::'a)\ then\ -number-of\ x\ else\ number-of\ x)$
 $\langle proof \rangle$

Re-orientation of the equation $nnn=x$

lemma *number-of-reorient*:

$(number-of\ w = x) = (x = number-of\ w)$
 $\langle proof \rangle$

30.10.4 Simplification of arithmetic operations on integer constants.

lemmas *arith-extra-simps* [standard, simp] =

number-of-add [symmetric]
number-of-minus [symmetric]
numeral-m1-eq-minus-1 [symmetric]
number-of-mult [symmetric]
diff-number-of-eq abs-number-of

For making a minimal simpset, one must include these default simprules.
 Also include *simp-thms*.

lemmas *arith-simps* =

normalize-bin-simps pred-bin-simps succ-bin-simps
add-bin-simps minus-bin-simps mult-bin-simps
abs-zero abs-one arith-extra-simps

Simplification of relational operations

lemma *less-number-of* [simp]:

$(number-of\ x::'a::\{\text{ordered-idom}, \text{number-ring}\}) < number-of\ y \longleftrightarrow x < y$
 $\langle proof \rangle$

lemma *le-number-of* [simp]:

$(number-of\ x::'a::\{\text{ordered-idom}, \text{number-ring}\}) \leq number-of\ y \longleftrightarrow x \leq y$
 $\langle proof \rangle$

lemma *eq-number-of* [simp]:
 $(\text{number-of } x :: 'a :: \{\text{ring-char-0}, \text{number-ring}\}) = \text{number-of } y \longleftrightarrow x = y$
 ⟨proof⟩

lemmas *rel-simps* [simp] =
less-number-of less-bin-simps
le-number-of le-bin-simps
eq-number-of-eq eq-bin-simps
iszero-simps

30.10.5 Simplification of arithmetic when nested to the right.

lemma *add-number-of-left* [simp]:
 $\text{number-of } v + (\text{number-of } w + z) =$
 $(\text{number-of } (v + w) + z :: 'a :: \text{number-ring})$
 ⟨proof⟩

lemma *mult-number-of-left* [simp]:
 $\text{number-of } v * (\text{number-of } w * z) =$
 $(\text{number-of } (v * w) * z :: 'a :: \text{number-ring})$
 ⟨proof⟩

lemma *add-number-of-diff1*:
 $\text{number-of } v + (\text{number-of } w - c) =$
 $\text{number-of } (v + w) - (c :: 'a :: \text{number-ring})$
 ⟨proof⟩

lemma *add-number-of-diff2* [simp]:
 $\text{number-of } v + (c - \text{number-of } w) =$
 $\text{number-of } (v + \text{uminus } w) + (c :: 'a :: \text{number-ring})$
 ⟨proof⟩

30.11 The Set of Integers

context *ring-1*
begin

definition *Ints* :: 'a set **where**
 [code del]: *Ints* = range of-int

end

notation (*xsymbols*)
Ints (\mathbb{Z})

context *ring-1*
begin

lemma *Ints-0* [simp]: $0 \in \mathbb{Z}$
 ⟨proof⟩

lemma *Ints-1* [*simp*]: $1 \in \mathbb{Z}$

$\langle \text{proof} \rangle$

lemma *Ints-add* [*simp*]: $a \in \mathbb{Z} \implies b \in \mathbb{Z} \implies a + b \in \mathbb{Z}$

$\langle \text{proof} \rangle$

lemma *Ints-minus* [*simp*]: $a \in \mathbb{Z} \implies -a \in \mathbb{Z}$

$\langle \text{proof} \rangle$

lemma *Ints-mult* [*simp*]: $a \in \mathbb{Z} \implies b \in \mathbb{Z} \implies a * b \in \mathbb{Z}$

$\langle \text{proof} \rangle$

lemma *Ints-cases* [*cases set: Ints*]:

assumes $q \in \mathbb{Z}$

obtains (*of-int*) z **where** $q = \text{of-int } z$

$\langle \text{proof} \rangle$

lemma *Ints-induct* [*case-names of-int, induct set: Ints*]:

$q \in \mathbb{Z} \implies (\bigwedge z. P (\text{of-int } z)) \implies P q$

$\langle \text{proof} \rangle$

end

lemma *Ints-diff* [*simp*]: $a \in \mathbb{Z} \implies b \in \mathbb{Z} \implies a - b \in \mathbb{Z}$

$\langle \text{proof} \rangle$

The premise involving \mathbb{Z} prevents $a = (1::'a) / (2::'a)$.

lemma *Ints-double-eq-0-iff*:

assumes *in-Ints*: $a \in \text{Ints}$

shows $(a + a = 0) = (a = (0::'a::\text{ring-char-0}))$

$\langle \text{proof} \rangle$

lemma *Ints-odd-nonzero*:

assumes *in-Ints*: $a \in \text{Ints}$

shows $1 + a + a \neq (0::'a::\text{ring-char-0})$

$\langle \text{proof} \rangle$

lemma *Ints-number-of*:

$(\text{number-of } w :: 'a::\text{number-ring}) \in \text{Ints}$

$\langle \text{proof} \rangle$

lemma *Ints-odd-less-0*:

assumes *in-Ints*: $a \in \text{Ints}$

shows $(1 + a + a < 0) = (a < (0::'a::\text{ordered-idom}))$

$\langle \text{proof} \rangle$

30.12 *setsum and setprod*

lemma *of-nat-setsum*: $\text{of-nat } (\text{setsum } f \ A) = (\sum x \in A. \ \text{of-nat}(f \ x))$
 $\langle \text{proof} \rangle$

lemma *of-int-setsum*: $\text{of-int } (\text{setsum } f \ A) = (\sum x \in A. \ \text{of-int}(f \ x))$
 $\langle \text{proof} \rangle$

lemma *of-nat-setprod*: $\text{of-nat } (\text{setprod } f \ A) = (\prod x \in A. \ \text{of-nat}(f \ x))$
 $\langle \text{proof} \rangle$

lemma *of-int-setprod*: $\text{of-int } (\text{setprod } f \ A) = (\prod x \in A. \ \text{of-int}(f \ x))$
 $\langle \text{proof} \rangle$

lemmas *int-setsum* = *of-nat-setsum* [where 'a=int]

lemmas *int-setprod* = *of-nat-setprod* [where 'a=int]

30.13 Inequality Reasoning for the Arithmetic Simproc

lemma *add-numeral-0*: $\text{Numeral0} + a = (a::'a::\text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *add-numeral-0-right*: $a + \text{Numeral0} = (a::'a::\text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *mult-numeral-1*: $\text{Numeral1} * a = (a::'a::\text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *mult-numeral-1-right*: $a * \text{Numeral1} = (a::'a::\text{number-ring})$
 $\langle \text{proof} \rangle$

lemma *divide-numeral-1*: $a / \text{Numeral1} = (a::'a::\{\text{number-ring}, \text{field}\})$
 $\langle \text{proof} \rangle$

lemma *inverse-numeral-1*:
 $\text{inverse } \text{Numeral1} = (\text{Numeral1}::'a::\{\text{number-ring}, \text{field}\})$
 $\langle \text{proof} \rangle$

Theorem lists for the cancellation simprocs. The use of binary numerals for 0 and 1 reduces the number of special cases.

lemmas *add-0s* = *add-numeral-0 add-numeral-0-right*

lemmas *mult-1s* = *mult-numeral-1 mult-numeral-1-right*
mult-minus1 mult-minus1-right

30.14 Special Arithmetic Rules for Abstract 0 and 1

Arithmetic computations are defined for binary literals, which leaves 0 and 1 as special cases. Addition already has rules for 0, but not 1. Multiplication and unary minus already have rules for both 0 and 1.

lemma *binop-eq*: $[[f\ x\ y = g\ x\ y; x = x'; y = y']] ==> f\ x'\ y' = g\ x'\ y'$
 $\langle proof \rangle$

lemmas *add-number-of-eq* = *number-of-add* [*symmetric*]

Allow 1 on either or both sides

lemma *one-add-one-is-two*: $1 + 1 = (2::'a::number-ring)$
 $\langle proof \rangle$

lemmas *add-special* =
one-add-one-is-two
binop-eq [*of op* +, *OF add-number-of-eq numeral-1-eq-1 refl, standard*]
binop-eq [*of op* +, *OF add-number-of-eq refl numeral-1-eq-1, standard*]

Allow 1 on either or both sides (1-1 already simplifies to 0)

lemmas *diff-special* =
binop-eq [*of op* −, *OF diff-number-of-eq numeral-1-eq-1 refl, standard*]
binop-eq [*of op* −, *OF diff-number-of-eq refl numeral-1-eq-1, standard*]

Allow 0 or 1 on either side with a binary numeral on the other

lemmas *eq-special* =
binop-eq [*of op* =, *OF eq-number-of-eq numeral-0-eq-0 refl, standard*]
binop-eq [*of op* =, *OF eq-number-of-eq numeral-1-eq-1 refl, standard*]
binop-eq [*of op* =, *OF eq-number-of-eq refl numeral-0-eq-0, standard*]
binop-eq [*of op* =, *OF eq-number-of-eq refl numeral-1-eq-1, standard*]

Allow 0 or 1 on either side with a binary numeral on the other

lemmas *less-special* =
binop-eq [*of op* <, *OF less-number-of numeral-0-eq-0 refl, standard*]
binop-eq [*of op* <, *OF less-number-of numeral-1-eq-1 refl, standard*]
binop-eq [*of op* <, *OF less-number-of refl numeral-0-eq-0, standard*]
binop-eq [*of op* <, *OF less-number-of refl numeral-1-eq-1, standard*]

Allow 0 or 1 on either side with a binary numeral on the other

lemmas *le-special* =
binop-eq [*of op* ≤, *OF le-number-of numeral-0-eq-0 refl, standard*]
binop-eq [*of op* ≤, *OF le-number-of numeral-1-eq-1 refl, standard*]
binop-eq [*of op* ≤, *OF le-number-of refl numeral-0-eq-0, standard*]
binop-eq [*of op* ≤, *OF le-number-of refl numeral-1-eq-1, standard*]

lemmas *arith-special*[*simp*] =
add-special *diff-special* *eq-special* *less-special* *le-special*

lemma *min-max-01*: $\min\ (0::int)\ 1 = 0 \ \&\ \min\ (1::int)\ 0 = 0 \ \&$
 $\max\ (0::int)\ 1 = 1 \ \&\ \max\ (1::int)\ 0 = 1$
 $\langle proof \rangle$

```

lemmas min-max-special[simp] =
  min-max-01
  max-def[of 0::int number-of v, standard, simp]
  min-def[of 0::int number-of v, standard, simp]
  max-def[of number-of u 0::int, standard, simp]
  min-def[of number-of u 0::int, standard, simp]
  max-def[of 1::int number-of v, standard, simp]
  min-def[of 1::int number-of v, standard, simp]
  max-def[of number-of u 1::int, standard, simp]
  min-def[of number-of u 1::int, standard, simp]

```

Legacy theorems

```

lemmas zle-int = of-nat-le-iff [where 'a=int]
lemmas int-int-eq = of-nat-eq-iff [where 'a=int]

```

30.15 Setting up simplification procedures

```

lemmas int-arith-rules =
  neg-le-iff-le numeral-0-eq-0 numeral-1-eq-1
  minus-zero diff-minus left-minus right-minus
  mult-zero-left mult-zero-right mult-Bit1 mult-1-right
  mult-minus-left mult-minus-right
  minus-add-distrib minus-minus mult-assoc
  of-nat-0 of-nat-1 of-nat-Suc of-nat-add of-nat-mult
  of-int-0 of-int-1 of-int-add of-int-mult

```

$\langle ML \rangle$

30.16 Lemmas About Small Numerals

```

lemma of-int-m1 [simp]: of-int -1 = (-1 :: 'a :: number-ring)
  <proof>

```

```

lemma abs-minus-one [simp]: abs (-1) = (1::'a::{ordered-idom,number-ring})
  <proof>

```

```

lemma abs-power-minus-one [simp]:
  abs(-1 ^ n) = (1::'a::{ordered-idom,number-ring,recpower})
  <proof>

```

```

lemma of-int-number-of-eq [simp]:
  of-int (number-of v) = (number-of v :: 'a :: number-ring)
  <proof>

```

Lemmas for specialist use, NOT as default simprules

```

lemma mult-2: 2 * z = (z+z::'a::number-ring)
  <proof>

```

lemma *mult-2-right*: $z * 2 = (z + z :: 'a :: \text{number-ring})$
 $\langle \text{proof} \rangle$

30.17 More Inequality Reasoning

lemma *zless-add1-eq*: $(w < z + (1 :: \text{int})) = (w < z \mid w = z)$
 $\langle \text{proof} \rangle$

lemma *add1-zle-eq*: $(w + (1 :: \text{int}) \leq z) = (w < z)$
 $\langle \text{proof} \rangle$

lemma *zle-diff1-eq* [*simp*]: $(w \leq z - (1 :: \text{int})) = (w < z)$
 $\langle \text{proof} \rangle$

lemma *zle-add1-eq-le* [*simp*]: $(w < z + (1 :: \text{int})) = (w \leq z)$
 $\langle \text{proof} \rangle$

lemma *int-one-le-iff-zero-less*: $((1 :: \text{int}) \leq z) = (0 < z)$
 $\langle \text{proof} \rangle$

30.18 The functions *nat* and *int*

Simplify the terms *int* $(0 :: 'a)$, *int* $(\text{Suc } 0)$ and $w + - z$

declare *Zero-int-def* [*symmetric*, *simp*]

declare *One-int-def* [*symmetric*, *simp*]

lemmas *diff-int-def-symmetric* = *diff-int-def* [*symmetric*, *simp*]

lemma *nat-0*: $\text{nat } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *nat-1*: $\text{nat } 1 = \text{Suc } 0$
 $\langle \text{proof} \rangle$

lemma *nat-2*: $\text{nat } 2 = \text{Suc } (\text{Suc } 0)$
 $\langle \text{proof} \rangle$

lemma *one-less-nat-eq* [*simp*]: $(\text{Suc } 0 < \text{nat } z) = (1 < z)$
 $\langle \text{proof} \rangle$

This simplifies expressions of the form *int* $n = z$ where z is an integer literal.

lemmas *int-eq-iff-number-of* [*simp*] = *int-eq-iff* [*of - number-of v, standard*]

lemma *split-nat* [*arith-split*]:

$P(\text{nat}(i :: \text{int})) = ((\forall n. i = \text{of-nat } n \longrightarrow P n) \ \& \ (i < 0 \longrightarrow P 0))$
 $(\text{is } ?P = (?L \ \& \ ?R))$

$\langle \text{proof} \rangle$

context *ring-1*

begin

lemma *of-int-of-nat* [nitpick-const-simp]:

of-int $k = (\text{if } k < 0 \text{ then } - \text{of-nat } (\text{nat } (-k)) \text{ else } \text{of-nat } (\text{nat } k))$
 $\langle \text{proof} \rangle$

end

lemma *nat-mult-distrib*:

fixes $z\ z' :: \text{int}$
assumes $0 \leq z$
shows $\text{nat } (z * z') = \text{nat } z * \text{nat } z'$
 $\langle \text{proof} \rangle$

lemma *nat-mult-distrib-neg*: $z \leq (0 :: \text{int}) \implies \text{nat}(z * z') = \text{nat}(-z) * \text{nat}(-z')$
 $\langle \text{proof} \rangle$

lemma *nat-abs-mult-distrib*: $\text{nat } (\text{abs } (w * z)) = \text{nat } (\text{abs } w) * \text{nat } (\text{abs } z)$
 $\langle \text{proof} \rangle$

30.19 Induction principles for int

Well-founded segments of the integers

definition

int-ge-less-than $:: \text{int} \Rightarrow (\text{int} * \text{int}) \text{ set}$
where
int-ge-less-than $d = \{(z', z). d \leq z' \ \& \ z' < z\}$

theorem *wf-int-ge-less-than*: $\text{wf } (\text{int-ge-less-than } d)$
 $\langle \text{proof} \rangle$

This variant looks odd, but is typical of the relations suggested by Rank-Finder.

definition

int-ge-less-than2 $:: \text{int} \Rightarrow (\text{int} * \text{int}) \text{ set}$
where
int-ge-less-than2 $d = \{(z', z). d \leq z \ \& \ z' < z\}$

theorem *wf-int-ge-less-than2*: $\text{wf } (\text{int-ge-less-than2 } d)$
 $\langle \text{proof} \rangle$

abbreviation

int $:: \text{nat} \Rightarrow \text{int}$
where
int $\equiv \text{of-nat}$

theorem *int-ge-induct* [case-names base step, induct set: int]:

fixes $i :: \text{int}$
assumes $ge: k \leq i$ **and**
 $base: P\ k$ **and**
 $step: \bigwedge i. k \leq i \implies P\ i \implies P\ (i + 1)$
shows $P\ i$
 $\langle proof \rangle$

theorem *int-gr-induct* [*case-names base step, induct set: int*]:
assumes $gr: k < (i :: \text{int})$ **and**
 $base: P(k+1)$ **and**
 $step: \bigwedge i. \llbracket k < i; P\ i \rrbracket \implies P(i+1)$
shows $P\ i$
 $\langle proof \rangle$

theorem *int-le-induct* [*consumes 1, case-names base step*]:
assumes $le: i \leq (k :: \text{int})$ **and**
 $base: P(k)$ **and**
 $step: \bigwedge i. \llbracket i \leq k; P\ i \rrbracket \implies P(i - 1)$
shows $P\ i$
 $\langle proof \rangle$

theorem *int-less-induct* [*consumes 1, case-names base step*]:
assumes $less: (i :: \text{int}) < k$ **and**
 $base: P(k - 1)$ **and**
 $step: \bigwedge i. \llbracket i < k; P\ i \rrbracket \implies P(i - 1)$
shows $P\ i$
 $\langle proof \rangle$

30.20 Intermediate value theorems

lemma *int-val-lemma*:
 $(\forall i < n :: \text{nat}. \text{abs}(f(i+1) - f\ i) \leq 1) \dashv\dashv$
 $f\ 0 \leq k \dashv\dashv k \leq f\ n \dashv\dashv (\exists i \leq n. f\ i = (k :: \text{int}))$
 $\langle proof \rangle$

lemmas *nat0-intermed-int-val* = *int-val-lemma* [*rule-format (no-asm)*]

lemma *nat-intermed-int-val*:
 $\llbracket \forall i. m \leq i \ \& \ i < n \dashv\dashv \text{abs}(f(i + 1 :: \text{nat}) - f\ i) \leq 1; m < n;$
 $f\ m \leq k; k \leq f\ n \rrbracket \implies ?\ i. m \leq i \ \& \ i \leq n \ \& \ f\ i = (k :: \text{int})$
 $\langle proof \rangle$

30.21 Products and 1, by T. M. Rasmussen

lemma *zabs-less-one-iff* [*simp*]: $(|z| < 1) = (z = (0 :: \text{int}))$
 $\langle proof \rangle$

lemma *abs-zmult-eq-1*: $(|m * n| = 1) \implies |m| = (1 :: \text{int})$
 $\langle proof \rangle$

lemma *pos-zmult-eq-1-iff-lemma*: $(m * n = 1) ==> m = (1::int) \mid m = -1$
 $\langle proof \rangle$

lemma *pos-zmult-eq-1-iff*: $0 < (m::int) ==> (m * n = 1) = (m = 1 \ \& \ n = 1)$
 $\langle proof \rangle$

lemma *zmult-eq-1-iff*: $(m*n = (1::int)) = ((m = 1 \ \& \ n = 1) \mid (m = -1 \ \& \ n = -1))$
 $\langle proof \rangle$

lemma *infinite-UNIV-int*: $\sim finite(UNIV::int \ set)$
 $\langle proof \rangle$

30.22 Integer Powers

instantiation *int* :: *recpower*
begin

primrec *power-int* **where**
 $p \ ^\wedge \ 0 = (1::int)$
 $\mid p \ ^\wedge \ (Suc \ n) = (p::int) * (p \ ^\wedge \ n)$

instance $\langle proof \rangle$

declare *power-int.simps* [*simp del*]

end

lemma *zpower-zadd-distrib*: $x \ ^\wedge \ (y + z) = ((x \ ^\wedge \ y) * (x \ ^\wedge \ z)::int)$
 $\langle proof \rangle$

lemma *zpower-zpower*: $(x \ ^\wedge \ y) \ ^\wedge \ z = (x \ ^\wedge \ (y * z)::int)$
 $\langle proof \rangle$

lemma *zero-less-zpower-abs-iff* [*simp*]:
 $(0 < abs \ x \ ^\wedge \ n) \longleftrightarrow (x \neq (0::int) \mid n = 0)$
 $\langle proof \rangle$

lemma *zero-le-zpower-abs* [*simp*]: $(0::int) \leq abs \ x \ ^\wedge \ n$
 $\langle proof \rangle$

lemma *of-int-power*:
 $of-int \ (z \ ^\wedge \ n) = (of-int \ z \ ^\wedge \ n :: 'a::\{recpower, ring-1\})$
 $\langle proof \rangle$

lemma *int-power*: $int \ (m \ ^\wedge \ n) = (int \ m) \ ^\wedge \ n$
 $\langle proof \rangle$

lemmas *zpower-int = int-power [symmetric]*

30.23 Further theorems on numerals

30.23.1 Special Simplification for Constants

These distributive laws move literals inside sums and differences.

lemmas *left-distrib-number-of [simp] =
left-distrib [of - - number-of v, standard]*

lemmas *right-distrib-number-of [simp] =
right-distrib [of number-of v, standard]*

lemmas *left-diff-distrib-number-of [simp] =
left-diff-distrib [of - - number-of v, standard]*

lemmas *right-diff-distrib-number-of [simp] =
right-diff-distrib [of number-of v, standard]*

These are actually for fields, like real: but where else to put them?

lemmas *zero-less-divide-iff-number-of [simp, noatp] =
zero-less-divide-iff [of number-of w, standard]*

lemmas *divide-less-0-iff-number-of [simp, noatp] =
divide-less-0-iff [of number-of w, standard]*

lemmas *zero-le-divide-iff-number-of [simp, noatp] =
zero-le-divide-iff [of number-of w, standard]*

lemmas *divide-le-0-iff-number-of [simp, noatp] =
divide-le-0-iff [of number-of w, standard]*

Replaces *inverse #nn* by *1/#nn*. It looks strange, but then other simprocs simplify the quotient.

lemmas *inverse-eq-divide-number-of [simp] =
inverse-eq-divide [of number-of w, standard]*

These laws simplify inequalities, moving unary minus from a term into the literal.

lemmas *less-minus-iff-number-of [simp, noatp] =
less-minus-iff [of number-of v, standard]*

lemmas *le-minus-iff-number-of [simp, noatp] =
le-minus-iff [of number-of v, standard]*

lemmas *equation-minus-iff-number-of [simp, noatp] =
equation-minus-iff [of number-of v, standard]*

lemmas *minus-less-iff-number-of* [*simp*, *noatp*] =
minus-less-iff [*of* - *number-of* *v*, *standard*]

lemmas *minus-le-iff-number-of* [*simp*, *noatp*] =
minus-le-iff [*of* - *number-of* *v*, *standard*]

lemmas *minus-equation-iff-number-of* [*simp*, *noatp*] =
minus-equation-iff [*of* - *number-of* *v*, *standard*]

To Simplify Inequalities Where One Side is the Constant 1

lemma *less-minus-iff-1* [*simp*,*noatp*]:
fixes *b::'b::{ordered-idom,number-ring}*
shows $(1 < - b) = (b < -1)$
 $\langle proof \rangle$

lemma *le-minus-iff-1* [*simp*,*noatp*]:
fixes *b::'b::{ordered-idom,number-ring}*
shows $(1 \leq - b) = (b \leq -1)$
 $\langle proof \rangle$

lemma *equation-minus-iff-1* [*simp*,*noatp*]:
fixes *b::'b::number-ring*
shows $(1 = - b) = (b = -1)$
 $\langle proof \rangle$

lemma *minus-less-iff-1* [*simp*,*noatp*]:
fixes *a::'b::{ordered-idom,number-ring}*
shows $(- a < 1) = (-1 < a)$
 $\langle proof \rangle$

lemma *minus-le-iff-1* [*simp*,*noatp*]:
fixes *a::'b::{ordered-idom,number-ring}*
shows $(- a \leq 1) = (-1 \leq a)$
 $\langle proof \rangle$

lemma *minus-equation-iff-1* [*simp*,*noatp*]:
fixes *a::'b::number-ring*
shows $(- a = 1) = (a = -1)$
 $\langle proof \rangle$

Cancellation of constant factors in comparisons ($<$ and \leq)

lemmas *mult-less-cancel-left-number-of* [*simp*, *noatp*] =
mult-less-cancel-left [*of* *number-of* *v*, *standard*]

lemmas *mult-less-cancel-right-number-of* [*simp*, *noatp*] =
mult-less-cancel-right [*of* - *number-of* *v*, *standard*]

lemmas *mult-le-cancel-left-number-of* [*simp*, *noatp*] =

mult-le-cancel-left [of number-of v , standard]

lemmas *mult-le-cancel-right-number-of* [simp, noatp] =
mult-le-cancel-right [of - number-of v , standard]

Multiplying out constant divisors in comparisons ($<$, \leq and $=$)

lemmas *le-divide-eq-number-of1* [simp] = *le-divide-eq* [of - - number-of w , standard]

lemmas *divide-le-eq-number-of1* [simp] = *divide-le-eq* [of - number-of w , standard]

lemmas *less-divide-eq-number-of1* [simp] = *less-divide-eq* [of - - number-of w , standard]

lemmas *divide-less-eq-number-of1* [simp] = *divide-less-eq* [of - number-of w , standard]

lemmas *eq-divide-eq-number-of1* [simp] = *eq-divide-eq* [of - - number-of w , standard]

lemmas *divide-eq-eq-number-of1* [simp] = *divide-eq-eq* [of - number-of w , standard]

30.23.2 Optional Simplification Rules Involving Constants

Simplify quotients that are compared with a literal constant.

lemmas *le-divide-eq-number-of* = *le-divide-eq* [of number-of w , standard]

lemmas *divide-le-eq-number-of* = *divide-le-eq* [of - - number-of w , standard]

lemmas *less-divide-eq-number-of* = *less-divide-eq* [of number-of w , standard]

lemmas *divide-less-eq-number-of* = *divide-less-eq* [of - - number-of w , standard]

lemmas *eq-divide-eq-number-of* = *eq-divide-eq* [of number-of w , standard]

lemmas *divide-eq-eq-number-of* = *divide-eq-eq* [of - - number-of w , standard]

Not good as automatic simprules because they cause case splits.

lemmas *divide-const-simps* =
le-divide-eq-number-of divide-le-eq-number-of less-divide-eq-number-of
divide-less-eq-number-of eq-divide-eq-number-of divide-eq-eq-number-of
le-divide-eq-1 divide-le-eq-1 less-divide-eq-1 divide-less-eq-1

Division By -1

lemma *divide-minus1* [simp]:
 $x / -1 = -(x :: 'a :: \{\text{field}, \text{division-by-zero}, \text{number-ring}\})$
 <proof>

lemma *minus1-divide* [simp]:
 $-1 / (x :: 'a :: \{\text{field}, \text{division-by-zero}, \text{number-ring}\}) = -(1/x)$
 <proof>

lemma *half-gt-zero-iff*:
 $(0 < r/2) = (0 < (r :: 'a :: \{\text{ordered-field}, \text{division-by-zero}, \text{number-ring}\}))$
 <proof>

lemmas *half-gt-zero* [simp] = *half-gt-zero-iff* [THEN iffD2, standard]

30.24 Configuration of the code generator

code-datatype *Pls Min Bit0 Bit1 number-of* :: *int* \Rightarrow *int*

lemmas *pred-succ-numeral-code* [*code*] =
pred-bin-simps succ-bin-simps

lemmas *plus-numeral-code* [*code*] =
add-bin-simps
arith-extra-simps(1) [**where** '*a* = *int*]

lemmas *minus-numeral-code* [*code*] =
minus-bin-simps
arith-extra-simps(2) [**where** '*a* = *int*]
arith-extra-simps(5) [**where** '*a* = *int*]

lemmas *times-numeral-code* [*code*] =
mult-bin-simps
arith-extra-simps(4) [**where** '*a* = *int*]

instantiation *int* :: *eq*
begin

definition [*code del*]: *eq-class.eq* *k l* \longleftrightarrow *k* - *l* = (0::*int*)

instance \langle *proof* \rangle

end

lemma *eq-number-of-int-code* [*code*]:
eq-class.eq (*number-of* *k* :: *int*) (*number-of* *l*) \longleftrightarrow *eq-class.eq* *k l*
 \langle *proof* \rangle

lemma *eq-int-code* [*code*]:
eq-class.eq *Int.Pls Int.Pls* \longleftrightarrow *True*
eq-class.eq *Int.Pls Int.Min* \longleftrightarrow *False*
eq-class.eq *Int.Pls (Int.Bit0 k2)* \longleftrightarrow *eq-class.eq Int.Pls k2*
eq-class.eq *Int.Pls (Int.Bit1 k2)* \longleftrightarrow *False*
eq-class.eq *Int.Min Int.Pls* \longleftrightarrow *False*
eq-class.eq *Int.Min Int.Min* \longleftrightarrow *True*
eq-class.eq *Int.Min (Int.Bit0 k2)* \longleftrightarrow *False*
eq-class.eq *Int.Min (Int.Bit1 k2)* \longleftrightarrow *eq-class.eq Int.Min k2*
eq-class.eq (*Int.Bit0 k1*) *Int.Pls* \longleftrightarrow *eq-class.eq k1 Int.Pls*
eq-class.eq (*Int.Bit1 k1*) *Int.Pls* \longleftrightarrow *False*
eq-class.eq (*Int.Bit0 k1*) *Int.Min* \longleftrightarrow *False*
eq-class.eq (*Int.Bit1 k1*) *Int.Min* \longleftrightarrow *eq-class.eq k1 Int.Min*
eq-class.eq (*Int.Bit0 k1*) (*Int.Bit0 k2*) \longleftrightarrow *eq-class.eq k1 k2*
eq-class.eq (*Int.Bit0 k1*) (*Int.Bit1 k2*) \longleftrightarrow *False*
eq-class.eq (*Int.Bit1 k1*) (*Int.Bit0 k2*) \longleftrightarrow *False*
eq-class.eq (*Int.Bit1 k1*) (*Int.Bit1 k2*) \longleftrightarrow *eq-class.eq k1 k2*

$\langle \text{proof} \rangle$

lemma *eq-int-refl* [code nbe]:

eq-class.eq (*k::int*) *k* \longleftrightarrow *True*

$\langle \text{proof} \rangle$

lemma *less-eq-number-of-int-code* [code]:

(*number-of k :: int*) \leq *number-of l* \longleftrightarrow *k* \leq *l*

$\langle \text{proof} \rangle$

lemma *less-eq-int-code* [code]:

Int.Pls \leq *Int.Pls* \longleftrightarrow *True*

Int.Pls \leq *Int.Min* \longleftrightarrow *False*

Int.Pls \leq *Int.Bit0 k* \longleftrightarrow *Int.Pls* \leq *k*

Int.Pls \leq *Int.Bit1 k* \longleftrightarrow *Int.Pls* \leq *k*

Int.Min \leq *Int.Pls* \longleftrightarrow *True*

Int.Min \leq *Int.Min* \longleftrightarrow *True*

Int.Min \leq *Int.Bit0 k* \longleftrightarrow *Int.Min* $<$ *k*

Int.Min \leq *Int.Bit1 k* \longleftrightarrow *Int.Min* \leq *k*

Int.Bit0 k \leq *Int.Pls* \longleftrightarrow *k* \leq *Int.Pls*

Int.Bit1 k \leq *Int.Pls* \longleftrightarrow *k* $<$ *Int.Pls*

Int.Bit0 k \leq *Int.Min* \longleftrightarrow *k* \leq *Int.Min*

Int.Bit1 k \leq *Int.Min* \longleftrightarrow *k* \leq *Int.Min*

Int.Bit0 k1 \leq *Int.Bit0 k2* \longleftrightarrow *k1* \leq *k2*

Int.Bit0 k1 \leq *Int.Bit1 k2* \longleftrightarrow *k1* \leq *k2*

Int.Bit1 k1 \leq *Int.Bit0 k2* \longleftrightarrow *k1* $<$ *k2*

Int.Bit1 k1 \leq *Int.Bit1 k2* \longleftrightarrow *k1* \leq *k2*

$\langle \text{proof} \rangle$

lemma *less-number-of-int-code* [code]:

(*number-of k :: int*) $<$ *number-of l* \longleftrightarrow *k* $<$ *l*

$\langle \text{proof} \rangle$

lemma *less-int-code* [code]:

Int.Pls $<$ *Int.Pls* \longleftrightarrow *False*

Int.Pls $<$ *Int.Min* \longleftrightarrow *False*

Int.Pls $<$ *Int.Bit0 k* \longleftrightarrow *Int.Pls* $<$ *k*

Int.Pls $<$ *Int.Bit1 k* \longleftrightarrow *Int.Pls* \leq *k*

Int.Min $<$ *Int.Pls* \longleftrightarrow *True*

Int.Min $<$ *Int.Min* \longleftrightarrow *False*

Int.Min $<$ *Int.Bit0 k* \longleftrightarrow *Int.Min* $<$ *k*

Int.Min $<$ *Int.Bit1 k* \longleftrightarrow *Int.Min* $<$ *k*

Int.Bit0 k $<$ *Int.Pls* \longleftrightarrow *k* $<$ *Int.Pls*

Int.Bit1 k $<$ *Int.Pls* \longleftrightarrow *k* $<$ *Int.Pls*

Int.Bit0 k $<$ *Int.Min* \longleftrightarrow *k* \leq *Int.Min*

Int.Bit1 k $<$ *Int.Min* \longleftrightarrow *k* $<$ *Int.Min*

Int.Bit0 k1 $<$ *Int.Bit0 k2* \longleftrightarrow *k1* $<$ *k2*

Int.Bit0 k1 $<$ *Int.Bit1 k2* \longleftrightarrow *k1* \leq *k2*

Int.Bit1 k1 $<$ *Int.Bit0 k2* \longleftrightarrow *k1* $<$ *k2*

Int.Bit1 $k1 < \text{Int.Bit1 } k2 \longleftrightarrow k1 < k2$
 ⟨proof⟩

definition

nat-aux :: *int* \Rightarrow *nat* \Rightarrow *nat* **where**
nat-aux *i n* = *nat i* + *n*

lemma [*code*]:

nat-aux *i n* = (if *i* \leq 0 then *n* else *nat-aux* (*i* − 1) (*Suc n*)) — tail recursive
 ⟨proof⟩

lemma [*code*]: *nat i* = *nat-aux i 0*

⟨proof⟩

hide (**open**) *const nat-aux*

lemma *zero-is-num-zero* [*code*, *code inline*, *symmetric*, *code post*]:

(0::*int*) = *Numeral0*
 ⟨proof⟩

lemma *one-is-num-one* [*code*, *code inline*, *symmetric*, *code post*]:

(1::*int*) = *Numeral1*
 ⟨proof⟩

code-modulename *SML*

Int Integer

code-modulename *OCaml*

Int Integer

code-modulename *Haskell*

Int Integer

types-code

int (*int*)

attach (*term-of*) ⟨⟨

val term-of-int = *HOLogic.mk-number HOLogic.intT*;

⟩⟩

attach (*test*) ⟨⟨

fun gen-int i =

let val j = *one-of* [*~1*, *1*] * *random-range 0 i*

in (*j*, *fn* () => *term-of-int j*) *end*;

⟩⟩

⟨ML⟩

consts-code

number-of :: *int* \Rightarrow *int* ((-))

0 :: *int* (0)

```

1 :: int (1)
uminus :: int => int (~)
op + :: int => int => int ((- +/ -))
op * :: int => int => int ((- */ -))
op ≤ :: int => int => bool ((- <=/ -))
op < :: int => int => bool ((- </ -))

```

quickcheck-params [default-type = int]

hide (**open**) *const Pls Min Bit0 Bit1 succ pred*

30.25 Legacy theorems

```

lemmas zminus-zminus = minus-minus [of z::int, standard]
lemmas zminus-0 = minus-zero [where 'a=int]
lemmas zminus-zadd-distrib = minus-add-distrib [of z::int w, standard]
lemmas zadd-commute = add-commute [of z::int w, standard]
lemmas zadd-assoc = add-assoc [of z1::int z2 z3, standard]
lemmas zadd-left-commute = add-left-commute [of x::int y z, standard]
lemmas zadd-ac = zadd-assoc zadd-commute zadd-left-commute
lemmas zmult-ac = OrderedGroup.mult-ac
lemmas zadd-0 = OrderedGroup.add-0-left [of z::int, standard]
lemmas zadd-0-right = OrderedGroup.add-0-right [of z::int, standard]
lemmas zadd-zminus-inverse2 = left-minus [of z::int, standard]
lemmas zmult-zminus = mult-minus-left [of z::int w, standard]
lemmas zmult-commute = mult-commute [of z::int w, standard]
lemmas zmult-assoc = mult-assoc [of z1::int z2 z3, standard]
lemmas zadd-zmult-distrib = left-distrib [of z1::int z2 w, standard]
lemmas zadd-zmult-distrib2 = right-distrib [of w::int z1 z2, standard]
lemmas zdiff-zmult-distrib = left-diff-distrib [of z1::int z2 w, standard]
lemmas zdiff-zmult-distrib2 = right-diff-distrib [of w::int z1 z2, standard]

```

```

lemmas zmult-1 = mult-1-left [of z::int, standard]
lemmas zmult-1-right = mult-1-right [of z::int, standard]

```

```

lemmas zle-refl = order-refl [of w::int, standard]
lemmas zle-trans = order-trans [where 'a=int and x=i and y=j and z=k,
standard]
lemmas zle-anti-sym = order-antisym [of z::int w, standard]
lemmas zle-linear = linorder-linear [of z::int w, standard]
lemmas zless-linear = linorder-less-linear [where 'a = int]

```

```

lemmas zadd-left-mono = add-left-mono [of i::int j k, standard]
lemmas zadd-strict-right-mono = add-strict-right-mono [of i::int j k, standard]
lemmas zadd-zless-mono = add-less-le-mono [of w'::int w z' z, standard]

```

```

lemmas int-0-less-1 = zero-less-one [where 'a=int]
lemmas int-0-neq-1 = zero-neq-one [where 'a=int]

```

```

lemmas inj-int = inj-of-nat [where 'a=int]
lemmas zadd-int = of-nat-add [where 'a=int, symmetric]
lemmas int-mult = of-nat-mult [where 'a=int]
lemmas zmult-int = of-nat-mult [where 'a=int, symmetric]
lemmas int-eq-0-conv = of-nat-eq-0-iff [where 'a=int and m=n, standard]
lemmas zless-int = of-nat-less-iff [where 'a=int]
lemmas int-less-0-conv = of-nat-less-0-iff [where 'a=int and m=k, standard]
lemmas zero-less-int-conv = of-nat-0-less-iff [where 'a=int]
lemmas zero-zle-int = of-nat-0-le-iff [where 'a=int]
lemmas int-le-0-conv = of-nat-le-0-iff [where 'a=int and m=n, standard]
lemmas int-0 = of-nat-0 [where 'a=int]
lemmas int-1 = of-nat-1 [where 'a=int]
lemmas int-Suc = of-nat-Suc [where 'a=int]
lemmas abs-int-eq = abs-of-nat [where 'a=int and n=m, standard]
lemmas of-int-int-eq = of-int-of-nat-eq [where 'a=int]
lemmas zdiff-int = of-nat-diff [where 'a=int, symmetric]
lemmas zless-le = less-int-def
lemmas int-eq-of-nat = TrueI

end

```

31 IntDiv: The Division Operators div and mod

```

theory IntDiv
imports Int Divides FunDef
uses
  ~~/src/Provers/Arith/cancel-numeral-factor.ML
  ~~/src/Provers/Arith/extract-common-term.ML
  (Tools/int-factor-simprocs.ML)
begin

definition divmod-rel :: int  $\Rightarrow$  int  $\Rightarrow$  int  $\times$  int  $\Rightarrow$  bool where
  — definition of quotient and remainder
  [code]: divmod-rel a b = ( $\lambda(q, r). a = b * q + r \wedge$ 
    ( $\text{if } 0 < b \text{ then } 0 \leq r \wedge r < b \text{ else } b < r \wedge r \leq 0$ ))

definition adjust :: int  $\Rightarrow$  int  $\times$  int  $\Rightarrow$  int  $\times$  int where
  — for the division algorithm
  [code]: adjust b = ( $\lambda(q, r). \text{if } 0 \leq r - b \text{ then } (2 * q + 1, r - b)$ 
    else ( $2 * q, r$ ))

algorithm for the case  $a \geq 0, b > 0$ 

function posDivAlg :: int  $\Rightarrow$  int  $\Rightarrow$  int  $\times$  int where
  posDivAlg a b = ( $\text{if } a < b \vee b \leq 0 \text{ then } (0, a)$ 
    else adjust b (posDivAlg a ( $2 * b$ )))
  <proof>
termination <proof>

```


algorithm for the case $a < 0, b > 0$

function *negDivAlg* :: *int* \Rightarrow *int* \Rightarrow *int* \times *int* **where**
 negDivAlg *a* *b* = (if $0 \leq a + b \vee b \leq 0$ then $(-1, a + b)$
 else *adjust* *b* (*negDivAlg* *a* ($2 * b$)))
<proof>
termination *<proof>*

algorithm for the general case $b \neq (0 :: 'a)$

definition *negateSnd* :: *int* \times *int* \Rightarrow *int* \times *int* **where**
 [code inline]: *negateSnd* = *apsnd uminus*

definition *divmod* :: *int* \Rightarrow *int* \Rightarrow *int* \times *int* **where**
 — The full division algorithm considers all possible signs for *a*, *b* including the special case $a=0, b<0$ because *negDivAlg* requires $a < (0 :: 'a)$.
 divmod *a* *b* = (if $0 \leq a$ then if $0 \leq b$ then *posDivAlg* *a* *b*
 else if $a = 0$ then $(0, 0)$
 else *negateSnd* (*negDivAlg* $(-a)$ $(-b)$)
 else
 if $0 < b$ then *negDivAlg* *a* *b*
 else *negateSnd* (*posDivAlg* $(-a)$ $(-b)$))

instantiation *int* :: *Divides.div*
begin

definition
div-def: $a \text{ div } b = \text{fst } (\text{divmod } a \ b)$

definition
mod-def: $a \text{ mod } b = \text{snd } (\text{divmod } a \ b)$

instance *<proof>*

end

lemma *divmod-mod-div*:
divmod *p* *q* = (*p* *div* *q*, *p* *mod* *q*)
<proof>

Here is the division algorithm in ML:

```
fun posDivAlg (a,b) =
  if a<b then (0,a)
  else let val (q,r) = posDivAlg(a, 2*b)
        in if 0<le>r-b then (2*q+1, r-b) else (2*q, r)
  end

fun negDivAlg (a,b) =
  if 0<le>a+b then (~1,a+b)
```

```

    else let val (q,r) = negDivAlg(a, 2*b)
    in if 0 < r-b then (2*q+1, r-b) else (2*q, r)
end;

fun negateSnd (q,r:int) = (q,~r);

fun divmod (a,b) = if 0 < a then
if b>0 then posDivAlg (a,b)
else if a=0 then (0,0)
else negateSnd (negDivAlg (~a,~b))
else
if 0 < b then negDivAlg (a,b)
else negateSnd (posDivAlg (~a,~b));

```

31.1 Uniqueness and Monotonicity of Quotients and Remainders

lemma *unique-quotient-lemma:*

$$[[b*q' + r' \leq b*q + r; \ 0 \leq r'; \ r' < b; \ r < b]]$$

$$\implies q' \leq (q::int)$$

<proof>

lemma *unique-quotient-lemma-neg:*

$$[[b*q' + r' \leq b*q + r; \ r \leq 0; \ b < r; \ b < r']]$$

$$\implies q \leq (q'::int)$$

<proof>

lemma *unique-quotient:*

$$[[\text{divmod-rel } a \ b \ (q, \ r); \ \text{divmod-rel } a \ b \ (q', \ r'); \ b \neq 0]]$$

$$\implies q = q'$$

<proof>

lemma *unique-remainder:*

$$[[\text{divmod-rel } a \ b \ (q, \ r); \ \text{divmod-rel } a \ b \ (q', \ r'); \ b \neq 0]]$$

$$\implies r = r'$$

<proof>

31.2 Correctness of *posDivAlg*, the Algorithm for Non-Negative Dividends

And positive divisors

lemma *adjust-eq [simp]:*

$$\text{adjust } b \ (q,r) =$$

$$(\text{let } \text{diff} = r-b \text{ in}$$

$$\text{if } 0 \leq \text{diff} \text{ then } (2*q + 1, \text{diff}))$$

else ($2*q, r$)

⟨*proof*⟩

declare *posDivAlg.simps* [*simp del*]

use with a *simproc* to avoid repeatedly proving the premise

lemma *posDivAlg-eqn*:

$0 < b \implies$
 $\text{posDivAlg } a \ b = (\text{if } a < b \text{ then } (0, a) \text{ else adjust } b \ (\text{posDivAlg } a \ (2*b)))$

⟨*proof*⟩

Correctness of *posDivAlg*: it computes quotients correctly

theorem *posDivAlg-correct*:

assumes $0 \leq a$ **and** $0 < b$
shows *divmod-rel* $a \ b \ (\text{posDivAlg } a \ b)$

⟨*proof*⟩

31.3 Correctness of *negDivAlg*, the Algorithm for Negative Dividends

And positive divisors

declare *negDivAlg.simps* [*simp del*]

use with a *simproc* to avoid repeatedly proving the premise

lemma *negDivAlg-eqn*:

$0 < b \implies$
 $\text{negDivAlg } a \ b =$
 $(\text{if } 0 \leq a + b \text{ then } (-1, a + b) \text{ else adjust } b \ (\text{negDivAlg } a \ (2*b)))$

⟨*proof*⟩

lemma *negDivAlg-correct*:

assumes $a < 0$ **and** $b > 0$
shows *divmod-rel* $a \ b \ (\text{negDivAlg } a \ b)$

⟨*proof*⟩

31.4 Existence Shown by Proving the Division Algorithm to be Correct

lemma *divmod-rel-0*: $b \neq 0 \implies \text{divmod-rel } 0 \ b \ (0, 0)$

⟨*proof*⟩

lemma *posDivAlg-0* [*simp*]: $\text{posDivAlg } 0 \ b = (0, 0)$

⟨*proof*⟩

lemma *negDivAlg-minus1* [*simp*]: $\text{negDivAlg } -1 \ b = (-1, b - 1)$

⟨*proof*⟩

lemma *negateSnd-eq* [simp]: $\text{negateSnd}(q, r) = (q, -r)$
 ⟨proof⟩

lemma *divmod-rel-neg*: $\text{divmod-rel } (-a) (-b) \text{ } qr \implies \text{divmod-rel } a \ b \ (\text{negateSnd } qr)$
 ⟨proof⟩

lemma *divmod-correct*: $b \neq 0 \implies \text{divmod-rel } a \ b \ (\text{divmod } a \ b)$
 ⟨proof⟩

Arbitrary definitions for division by zero. Useful to simplify certain equations.

lemma *DIVISION-BY-ZERO* [simp]: $a \text{ div } (0::\text{int}) = 0 \ \& \ a \text{ mod } (0::\text{int}) = a$
 ⟨proof⟩

Basic laws about division and remainder

lemma *zmod-zdiv-equality*: $(a::\text{int}) = b * (a \text{ div } b) + (a \text{ mod } b)$
 ⟨proof⟩

lemma *zdiv-zmod-equality*: $(b * (a \text{ div } b) + (a \text{ mod } b)) + k = (a::\text{int}) + k$
 ⟨proof⟩

lemma *zdiv-zmod-equality2*: $((a \text{ div } b) * b + (a \text{ mod } b)) + k = (a::\text{int}) + k$
 ⟨proof⟩

Tool setup

⟨ML⟩

lemma *pos-mod-conj* : $(0::\text{int}) < b \implies 0 \leq a \text{ mod } b \ \& \ a \text{ mod } b < b$
 ⟨proof⟩

lemmas *pos-mod-sign* [simp] = *pos-mod-conj* [THEN conjunct1, standard]
 and *pos-mod-bound* [simp] = *pos-mod-conj* [THEN conjunct2, standard]

lemma *neg-mod-conj* : $b < (0::\text{int}) \implies a \text{ mod } b \leq 0 \ \& \ b < a \text{ mod } b$
 ⟨proof⟩

lemmas *neg-mod-sign* [simp] = *neg-mod-conj* [THEN conjunct1, standard]
 and *neg-mod-bound* [simp] = *neg-mod-conj* [THEN conjunct2, standard]

31.5 General Properties of div and mod

lemma *divmod-rel-div-mod*: $b \neq 0 \implies \text{divmod-rel } a \ b \ (a \text{ div } b, a \text{ mod } b)$
 ⟨proof⟩

lemma *divmod-rel-div*: $[| \text{divmod-rel } a \ b \ (q, r); \ b \neq 0 |] \implies a \text{ div } b = q$
 ⟨proof⟩

lemma *divmod-rel-mod*: $[| \text{divmod-rel } a \ b \ (q, r); \ b \neq 0 |] \implies a \text{ mod } b = r$

$\langle proof \rangle$

lemma *div-pos-pos-trivial*: $[(0::int) \leq a; \ a < b] \implies a \text{ div } b = 0$
 $\langle proof \rangle$

lemma *div-neg-neg-trivial*: $[a \leq (0::int); \ b < a] \implies a \text{ div } b = 0$
 $\langle proof \rangle$

lemma *div-pos-neg-trivial*: $[(0::int) < a; \ a+b \leq 0] \implies a \text{ div } b = -1$
 $\langle proof \rangle$

lemma *mod-pos-pos-trivial*: $[(0::int) \leq a; \ a < b] \implies a \text{ mod } b = a$
 $\langle proof \rangle$

lemma *mod-neg-neg-trivial*: $[a \leq (0::int); \ b < a] \implies a \text{ mod } b = a$
 $\langle proof \rangle$

lemma *mod-pos-neg-trivial*: $[(0::int) < a; \ a+b \leq 0] \implies a \text{ mod } b = a+b$
 $\langle proof \rangle$

There is no *mod-neg-pos-trivial*.

lemma *zdiv-zminus-zminus* [simp]: $(-a) \text{ div } (-b) = a \text{ div } (b::int)$
 $\langle proof \rangle$

lemma *zmod-zminus-zminus* [simp]: $(-a) \text{ mod } (-b) = -(a \text{ mod } (b::int))$
 $\langle proof \rangle$

31.6 Laws for div and mod with Unary Minus

lemma *zminus1-lemma*:
 $\text{divmod-rel } a \ b \ (q, r)$
 $\implies \text{divmod-rel } (-a) \ b \ (\text{if } r=0 \text{ then } -q \text{ else } -q - 1,$
 $\text{if } r=0 \text{ then } 0 \text{ else } b-r)$
 $\langle proof \rangle$

lemma *zdiv-zminus1-eq-if*:
 $b \neq (0::int)$
 $\implies (-a) \text{ div } b =$
 $(\text{if } a \text{ mod } b = 0 \text{ then } -(a \text{ div } b) \text{ else } -(a \text{ div } b) - 1)$
 $\langle proof \rangle$

lemma *zmod-zminus1-eq-if*:
 $(-a::int) \text{ mod } b = (\text{if } a \text{ mod } b = 0 \text{ then } 0 \text{ else } b - (a \text{ mod } b))$
 $\langle proof \rangle$

lemma *zmod-zminus1-not-zero*:
fixes $k\ l :: \text{int}$
shows $-k \bmod l \neq 0 \implies k \bmod l \neq 0$
 $\langle \text{proof} \rangle$

lemma *zdiv-zminus2*: $a \text{ div } (-b) = (-a::\text{int}) \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *zmod-zminus2*: $a \bmod (-b) = -((-a::\text{int}) \bmod b)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zminus2-eq-if*:
 $b \neq (0::\text{int})$
 $\implies a \text{ div } (-b) =$
 $(\text{if } a \bmod b = 0 \text{ then } -(a \text{ div } b) \text{ else } -(a \text{ div } b) - 1)$
 $\langle \text{proof} \rangle$

lemma *zmod-zminus2-eq-if*:
 $a \bmod (-b::\text{int}) = (\text{if } a \bmod b = 0 \text{ then } 0 \text{ else } (a \bmod b) - b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zminus2-not-zero*:
fixes $k\ l :: \text{int}$
shows $k \bmod -l \neq 0 \implies k \bmod l \neq 0$
 $\langle \text{proof} \rangle$

31.7 Division of a Number by Itself

lemma *self-quotient-aux1*: $[(0::\text{int}) < a; a = r + a*q; r < a] \implies 1 \leq q$
 $\langle \text{proof} \rangle$

lemma *self-quotient-aux2*: $[(0::\text{int}) < a; a = r + a*q; 0 \leq r] \implies q \leq 1$
 $\langle \text{proof} \rangle$

lemma *self-quotient*: $[\text{divmod-rel } a\ a\ (q, r); a \neq (0::\text{int})] \implies q = 1$
 $\langle \text{proof} \rangle$

lemma *self-remainder*: $[\text{divmod-rel } a\ a\ (q, r); a \neq (0::\text{int})] \implies r = 0$
 $\langle \text{proof} \rangle$

lemma *zdiv-self [simp]*: $a \neq 0 \implies a \text{ div } a = (1::\text{int})$
 $\langle \text{proof} \rangle$

lemma *zmod-self [simp]*: $a \bmod a = (0::\text{int})$
 $\langle \text{proof} \rangle$

31.8 Computation of Division and Remainder

lemma *zdiv-zero [simp]*: $(0::\text{int}) \text{ div } b = 0$

$\langle proof \rangle$

lemma *div-eq-minus1*: $(0::int) < b \implies -1 \text{ div } b = -1$
 $\langle proof \rangle$

lemma *zmod-zero* [simp]: $(0::int) \text{ mod } b = 0$
 $\langle proof \rangle$

lemma *zmod-minus1*: $(0::int) < b \implies -1 \text{ mod } b = b - 1$
 $\langle proof \rangle$

a positive, b positive

lemma *div-pos-pos*: $[| 0 < a; 0 \leq b |] \implies a \text{ div } b = \text{fst } (\text{posDivAlg } a \ b)$
 $\langle proof \rangle$

lemma *mod-pos-pos*: $[| 0 < a; 0 \leq b |] \implies a \text{ mod } b = \text{snd } (\text{posDivAlg } a \ b)$
 $\langle proof \rangle$

a negative, b positive

lemma *div-neg-pos*: $[| a < 0; 0 < b |] \implies a \text{ div } b = \text{fst } (\text{negDivAlg } a \ b)$
 $\langle proof \rangle$

lemma *mod-neg-pos*: $[| a < 0; 0 < b |] \implies a \text{ mod } b = \text{snd } (\text{negDivAlg } a \ b)$
 $\langle proof \rangle$

a positive, b negative

lemma *div-pos-neg*:
 $[| 0 < a; b < 0 |] \implies a \text{ div } b = \text{fst } (\text{negateSnd } (\text{negDivAlg } (-a) \ (-b)))$
 $\langle proof \rangle$

lemma *mod-pos-neg*:
 $[| 0 < a; b < 0 |] \implies a \text{ mod } b = \text{snd } (\text{negateSnd } (\text{negDivAlg } (-a) \ (-b)))$
 $\langle proof \rangle$

a negative, b negative

lemma *div-neg-neg*:
 $[| a < 0; b \leq 0 |] \implies a \text{ div } b = \text{fst } (\text{negateSnd } (\text{posDivAlg } (-a) \ (-b)))$
 $\langle proof \rangle$

lemma *mod-neg-neg*:
 $[| a < 0; b \leq 0 |] \implies a \text{ mod } b = \text{snd } (\text{negateSnd } (\text{posDivAlg } (-a) \ (-b)))$
 $\langle proof \rangle$

Simplify expresions in which div and mod combine numerical constants

lemma *divmod-relI*:
 $[| a == b * q + r; \text{ if } 0 < b \text{ then } 0 \leq r \wedge r < b \text{ else } b < r \wedge r \leq 0 |]$
 $\implies \text{divmod-rel } a \ b \ (q, r)$
 $\langle proof \rangle$

lemmas *divmod-rel-div-eq* = *divmod-rell* [*THEN divmod-rel-div*, *THEN eq-reflection*]
lemmas *divmod-rel-mod-eq* = *divmod-rell* [*THEN divmod-rel-mod*, *THEN eq-reflection*]
lemmas *arithmetic-simps* =
arith-simps
add-special
OrderedGroup.add-0-left
OrderedGroup.add-0-right
mult-zero-left
mult-zero-right
mult-1-left
mult-1-right

⟨*ML*⟩

lemmas *posDivAlg-eqn-number-of* [*simp*] =
posDivAlg-eqn [*of number-of v number-of w*, *standard*]

lemmas *negDivAlg-eqn-number-of* [*simp*] =
negDivAlg-eqn [*of number-of v number-of w*, *standard*]

Special-case simplification

lemma *zmod-minus1-right* [*simp*]: $a \bmod (-1::int) = 0$
 ⟨*proof*⟩

lemma *zdiv-minus1-right* [*simp*]: $a \operatorname{div} (-1::int) = -a$
 ⟨*proof*⟩

lemmas *div-pos-pos-1-number-of* [*simp*] =
div-pos-pos [*OF int-0-less-1*, *of number-of w*, *standard*]

lemmas *div-pos-neg-1-number-of* [*simp*] =
div-pos-neg [*OF int-0-less-1*, *of number-of w*, *standard*]

lemmas *mod-pos-pos-1-number-of* [*simp*] =
mod-pos-pos [*OF int-0-less-1*, *of number-of w*, *standard*]

lemmas *mod-pos-neg-1-number-of* [*simp*] =
mod-pos-neg [*OF int-0-less-1*, *of number-of w*, *standard*]

lemmas *posDivAlg-eqn-1-number-of* [*simp*] =
posDivAlg-eqn [*of concl: 1 number-of w*, *standard*]

lemmas *negDivAlg-eqn-1-number-of* [*simp*] =
negDivAlg-eqn [*of concl: 1 number-of w*, *standard*]

31.9 Monotonicity in the First Argument (Dividend)

lemma *zdiv-mono1*: $[[a \leq a'; \ 0 < (b::int)]] \implies a \text{ div } b \leq a' \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-mono1-neg*: $[[a \leq a'; \ (b::int) < 0]] \implies a' \text{ div } b \leq a \text{ div } b$
 $\langle \text{proof} \rangle$

31.10 Monotonicity in the Second Argument (Divisor)

lemma *q-pos-lemma*:
 $[[0 \leq b'*q' + r'; \ r' < b'; \ 0 < b']] \implies 0 \leq (q'::int)$
 $\langle \text{proof} \rangle$

lemma *zdiv-mono2-lemma*:
 $[[b*q + r = b'*q' + r'; \ 0 \leq b'*q' + r';$
 $r' < b'; \ 0 \leq r; \ 0 < b'; \ b' \leq b]]$
 $\implies q \leq (q'::int)$
 $\langle \text{proof} \rangle$

lemma *zdiv-mono2*:
 $[[(0::int) \leq a; \ 0 < b'; \ b' \leq b]] \implies a \text{ div } b \leq a \text{ div } b'$
 $\langle \text{proof} \rangle$

lemma *q-neg-lemma*:
 $[[b'*q' + r' < 0; \ 0 \leq r'; \ 0 < b']] \implies q' \leq (0::int)$
 $\langle \text{proof} \rangle$

lemma *zdiv-mono2-neg-lemma*:
 $[[b*q + r = b'*q' + r'; \ b'*q' + r' < 0;$
 $r < b; \ 0 \leq r'; \ 0 < b'; \ b' \leq b]]$
 $\implies q' \leq (q::int)$
 $\langle \text{proof} \rangle$

lemma *zdiv-mono2-neg*:
 $[[a < (0::int); \ 0 < b'; \ b' \leq b]] \implies a \text{ div } b' \leq a \text{ div } b$
 $\langle \text{proof} \rangle$

31.11 More Algebraic Laws for div and mod

proving $(a*b) \text{ div } c = a * (b \text{ div } c) + a * (b \text{ mod } c)$

lemma *zmult1-lemma*:
 $[[\text{divmod-rel } b \ c \ (q, r); \ c \neq 0]]$
 $\implies \text{divmod-rel } (a * b) \ c \ (a*q + a*r \text{ div } c, a*r \text{ mod } c)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult1-eq*: $(a*b) \text{ div } c = a*(b \text{ div } c) + a*(b \text{ mod } c) \text{ div } (c::int)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult1-eq*: $(a * b) \bmod c = a * (b \bmod c) \bmod (c :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *zmod-zdiv-trivial*: $(a \bmod b) \text{ div } b = (0 :: \text{int})$
 $\langle \text{proof} \rangle$

proving $(a+b) \text{ div } c = a \text{ div } c + b \text{ div } c + ((a \bmod c + b \bmod c) \text{ div } c)$

lemma *zadd1-lemma*:
 $\llbracket \text{divmod-rel } a \ c \ (aq, ar); \text{ divmod-rel } b \ c \ (bq, br); \ c \neq 0 \rrbracket$
 $\implies \text{divmod-rel } (a+b) \ c \ (aq + bq + (ar+br) \text{ div } c, (ar+br) \bmod c)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zadd1-eq*:
 $(a+b) \text{ div } (c :: \text{int}) = a \text{ div } c + b \text{ div } c + ((a \bmod c + b \bmod c) \text{ div } c)$
 $\langle \text{proof} \rangle$

instance *int :: ring-div*
 $\langle \text{proof} \rangle$

lemma *posDivAlg-div-mod*:
assumes $k \geq 0$
and $l \geq 0$
shows $\text{posDivAlg } k \ l = (k \text{ div } l, k \bmod l)$
 $\langle \text{proof} \rangle$

lemma *negDivAlg-div-mod*:
assumes $k < 0$
and $l > 0$
shows $\text{negDivAlg } k \ l = (k \text{ div } l, k \bmod l)$
 $\langle \text{proof} \rangle$

lemma *zmod-eq-0-iff*: $(m \bmod d = 0) = (\text{EX } q :: \text{int}. m = d * q)$
 $\langle \text{proof} \rangle$

lemmas *zmod-eq-0D* $[dest!] = \text{zmod-eq-0-iff } [THEN \text{ iffD1}]$

31.12 Proving $a \text{ div } (b * c) = a \text{ div } b \text{ div } c$

first, four lemmas to bound the remainder for the cases $b \nmid 0$ and $b \mid 0$

lemma *zmult2-lemma-aux1*: $\llbracket (0 :: \text{int}) < c; \ b < r; \ r \leq 0 \rrbracket \implies b * c < b * (q \bmod c) + r$
 $\langle \text{proof} \rangle$

lemma *zmult2-lemma-aux2*:
 $\llbracket (0 :: \text{int}) < c; \ b < r; \ r \leq 0 \rrbracket \implies b * (q \bmod c) + r \leq 0$
 $\langle \text{proof} \rangle$

lemma *zmult2-lemma-aux3*: $[[(0::int) < c; \ 0 \leq r; \ r < b]] \implies 0 \leq b * (q \text{ mod } c) + r$
 $\langle \text{proof} \rangle$

lemma *zmult2-lemma-aux4*: $[[(0::int) < c; \ 0 \leq r; \ r < b]] \implies b * (q \text{ mod } c) + r < b * c$
 $\langle \text{proof} \rangle$

lemma *zmult2-lemma*: $[[\text{divmod-rel } a \ b \ (q, r); \ b \neq 0; \ 0 < c]] \implies \text{divmod-rel } a \ (b * c) \ (q \text{ div } c, b * (q \text{ mod } c) + r)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-eq*: $(0::int) < c \implies a \text{ div } (b * c) = (a \text{ div } b) \text{ div } c$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult2-eq*:
 $(0::int) < c \implies a \text{ mod } (b * c) = b * (a \text{ div } b \text{ mod } c) + a \text{ mod } b$
 $\langle \text{proof} \rangle$

31.13 Cancellation of Common Factors in div

lemma *zdiv-zmult-zmult1-aux1*:
 $[[(0::int) < b; \ c \neq 0]] \implies (c * a) \text{ div } (c * b) = a \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1-aux2*:
 $[[b < (0::int); \ c \neq 0]] \implies (c * a) \text{ div } (c * b) = a \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1*: $c \neq (0::int) \implies (c * a) \text{ div } (c * b) = a \text{ div } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1-if[simp]*:
 $(k * m) \text{ div } (k * n) = (\text{if } k = (0::int) \text{ then } 0 \text{ else } m \text{ div } n)$
 $\langle \text{proof} \rangle$

31.14 Distribution of Factors over mod

lemma *zmod-zmult-zmult1-aux1*:
 $[[(0::int) < b; \ c \neq 0]] \implies (c * a) \text{ mod } (c * b) = c * (a \text{ mod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1-aux2*:
 $[[b < (0::int); \ c \neq 0]] \implies (c * a) \text{ mod } (c * b) = c * (a \text{ mod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1*: $(c * a) \text{ mod } (c * b) = (c::int) * (a \text{ mod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult2*: $(a * c) \text{ mod } (b * c) = (a \text{ mod } b) * (c::int)$

$\langle \text{proof} \rangle$

31.15 Splitting Rules for div and mod

The proofs of the two lemmas below are essentially identical

lemma *split-pos-lemma*:

$0 < k \implies$
 $P(n \text{ div } k :: \text{int})(n \text{ mod } k) = (\forall i j. 0 \leq j \ \& \ j < k \ \& \ n = k*i + j \implies P \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *split-neg-lemma*:

$k < 0 \implies$
 $P(n \text{ div } k :: \text{int})(n \text{ mod } k) = (\forall i j. k < j \ \& \ j \leq 0 \ \& \ n = k*i + j \implies P \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *split-zdiv*:

$P(n \text{ div } k :: \text{int}) =$
 $((k = 0 \implies P \ 0) \ \&$
 $(0 < k \implies (\forall i j. 0 \leq j \ \& \ j < k \ \& \ n = k*i + j \implies P \ i)) \ \&$
 $(k < 0 \implies (\forall i j. k < j \ \& \ j \leq 0 \ \& \ n = k*i + j \implies P \ i)))$
 $\langle \text{proof} \rangle$

lemma *split-zmod*:

$P(n \text{ mod } k :: \text{int}) =$
 $((k = 0 \implies P \ n) \ \&$
 $(0 < k \implies (\forall i j. 0 \leq j \ \& \ j < k \ \& \ n = k*i + j \implies P \ j)) \ \&$
 $(k < 0 \implies (\forall i j. k < j \ \& \ j \leq 0 \ \& \ n = k*i + j \implies P \ j)))$
 $\langle \text{proof} \rangle$

declare *split-zdiv* [of - - number-of k, simplified, standard, arith-split]

declare *split-zmod* [of - - number-of k, simplified, standard, arith-split]

31.16 Speeding up the Division Algorithm with Shifting

computing div by shifting

lemma *pos-zdiv-mult-2*: $(0 :: \text{int}) \leq a \implies (1 + 2*b) \text{ div } (2*a) = b \text{ div } a$
 $\langle \text{proof} \rangle$

lemma *neg-zdiv-mult-2*: $a \leq (0 :: \text{int}) \implies (1 + 2*b) \text{ div } (2*a) = (b+1) \text{ div } a$
 $\langle \text{proof} \rangle$

lemma *zdiv-number-of-Bit0* [simp]:

$\text{number-of } (\text{Int.Bit0 } v) \text{ div } \text{number-of } (\text{Int.Bit0 } w) =$
 $\text{number-of } v \text{ div } (\text{number-of } w :: \text{int})$
 $\langle \text{proof} \rangle$

lemma *zdiv-number-of-Bit1* [simp]:

$$\begin{aligned} & \text{number-of } (\text{Int.Bit1 } v) \text{ div number-of } (\text{Int.Bit0 } w) = \\ & \quad (\text{if } (0::\text{int}) \leq \text{number-of } w \\ & \quad \text{then number-of } v \text{ div } (\text{number-of } w) \\ & \quad \text{else } (\text{number-of } v + (1::\text{int})) \text{ div } (\text{number-of } w)) \\ & \langle \text{proof} \rangle \end{aligned}$$

31.17 Computing mod by Shifting (proofs resemble those for div)

lemma *pos-zmod-mult-2*:

$$(0::\text{int}) \leq a \implies (1 + 2*b) \text{ mod } (2*a) = 1 + 2 * (b \text{ mod } a)$$
 $\langle \text{proof} \rangle$

lemma *neg-zmod-mult-2*:

$$a \leq (0::\text{int}) \implies (1 + 2*b) \text{ mod } (2*a) = 2 * ((b+1) \text{ mod } a) - 1$$
 $\langle \text{proof} \rangle$

lemma *zmod-number-of-Bit0 [simp]*:

$$\begin{aligned} & \text{number-of } (\text{Int.Bit0 } v) \text{ mod number-of } (\text{Int.Bit0 } w) = \\ & \quad (2::\text{int}) * (\text{number-of } v \text{ mod number-of } w) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *zmod-number-of-Bit1 [simp]*:

$$\begin{aligned} & \text{number-of } (\text{Int.Bit1 } v) \text{ mod number-of } (\text{Int.Bit0 } w) = \\ & \quad (\text{if } (0::\text{int}) \leq \text{number-of } w \\ & \quad \text{then } 2 * (\text{number-of } v \text{ mod number-of } w) + 1 \\ & \quad \text{else } 2 * ((\text{number-of } v + (1::\text{int})) \text{ mod number-of } w) - 1) \\ & \langle \text{proof} \rangle \end{aligned}$$

31.18 Quotients of Signs

lemma *div-neg-pos-less0*: $[[a < (0::\text{int}); \ 0 < b]] \implies a \text{ div } b < 0$
 $\langle \text{proof} \rangle$

lemma *div-nonneg-neg-le0*: $[[(0::\text{int}) \leq a; \ b < 0]] \implies a \text{ div } b \leq 0$
 $\langle \text{proof} \rangle$

lemma *div-nonpos-pos-le0*: $[[(a::\text{int}) \leq 0; \ b > 0]] \implies a \text{ div } b \leq 0$
 $\langle \text{proof} \rangle$

lemma *pos-imp-zdiv-nonneg-iff*: $(0::\text{int}) < b \implies (0 \leq a \text{ div } b) = (0 \leq a)$
 $\langle \text{proof} \rangle$

lemma *neg-imp-zdiv-nonneg-iff*:

$$b < (0::\text{int}) \implies (0 \leq a \text{ div } b) = (a \leq (0::\text{int}))$$
 $\langle \text{proof} \rangle$

lemma *pos-imp-zdiv-neg-iff*: $(0::\text{int}) < b \implies (a \text{ div } b < 0) = (a < 0)$

$\langle proof \rangle$

lemma *neg-imp-zdiv-neg-iff*: $b < (0::int) ==> (a \text{ div } b < 0) = (0 < a)$
 $\langle proof \rangle$

31.19 The Divides Relation

lemmas *zdvd-iff-zmod-eq-0-number-of* [simp] =
dvd-eq-mod-eq-0 [of number-of $x::int$ number-of $y::int$, standard]

lemma *zdvd-anti-sym*:
 $0 < m ==> 0 < n ==> m \text{ dvd } n ==> n \text{ dvd } m ==> m = (n::int)$
 $\langle proof \rangle$

lemma *zdvd-dvd-eq*: **assumes** $a \neq 0$ **and** $(a::int) \text{ dvd } b$ **and** $b \text{ dvd } a$
shows $|a| = |b|$
 $\langle proof \rangle$

lemma *zdvd-zdiffD*: $k \text{ dvd } m - n ==> k \text{ dvd } n ==> k \text{ dvd } (m::int)$
 $\langle proof \rangle$

lemma *zdvd-reduce*: $(k \text{ dvd } n + k * m) = (k \text{ dvd } (n::int))$
 $\langle proof \rangle$

lemma *zdvd-zmod*: $f \text{ dvd } m ==> f \text{ dvd } (n::int) ==> f \text{ dvd } m \text{ mod } n$
 $\langle proof \rangle$

lemma *zdvd-zmod-imp-zdvd*: $k \text{ dvd } m \text{ mod } n ==> k \text{ dvd } n ==> k \text{ dvd } (m::int)$
 $\langle proof \rangle$

lemma *zdvd-not-zless*: $0 < m ==> m < n ==> \neg n \text{ dvd } (m::int)$
 $\langle proof \rangle$

lemma *zmult-div-cancel*: $(n::int) * (m \text{ div } n) = m - (m \text{ mod } n)$
 $\langle proof \rangle$

lemma *zdvd-mult-div-cancel*: $(n::int) \text{ dvd } m \implies n * (m \text{ div } n) = m$
 $\langle proof \rangle$

lemma *zdvd-mult-cancel*: **assumes** $d:k * m \text{ dvd } k * n$ **and** $kz:k \neq (0::int)$
shows $m \text{ dvd } n$
 $\langle proof \rangle$

theorem *ex-nat*: $(\exists x::nat. P x) = (\exists x::int. 0 \leq x \wedge P (nat x))$
 $\langle proof \rangle$

theorem *zdvd-int*: $(x \text{ dvd } y) = (int x \text{ dvd } int y)$

$\langle proof \rangle$

lemma *zdvd1-eq[simp]*: $(x::int) \text{ dvd } 1 = (|x| = 1)$

$\langle proof \rangle$

lemma *zdvd-mult-cancel1*:

assumes $mp:m \neq (0::int)$ **shows** $(m * n \text{ dvd } m) = (|n| = 1)$

$\langle proof \rangle$

lemma *int-dvd-iff*: $(int\ m \text{ dvd } z) = (m \text{ dvd } nat\ (abs\ z))$

$\langle proof \rangle$

lemma *dvd-int-iff*: $(z \text{ dvd } int\ m) = (nat\ (abs\ z) \text{ dvd } m)$

$\langle proof \rangle$

lemma *nat-dvd-iff*: $(nat\ z \text{ dvd } m) = (if\ 0 \leq z\ then\ (z \text{ dvd } int\ m)\ else\ m = 0)$

$\langle proof \rangle$

lemma *zdvd-imp-le*: $[| z \text{ dvd } n; 0 < n |] ==> z \leq (n::int)$

$\langle proof \rangle$

lemma *zpower-zmod*: $((x::int) \text{ mod } m)^y \text{ mod } m = x^y \text{ mod } m$

$\langle proof \rangle$

lemma *zdiv-int*: $int\ (a \text{ div } b) = (int\ a) \text{ div } (int\ b)$

$\langle proof \rangle$

lemma *zmod-int*: $int\ (a \text{ mod } b) = (int\ a) \text{ mod } (int\ b)$

$\langle proof \rangle$

lemma *abs-div*: $(y::int) \text{ dvd } x \implies abs\ (x \text{ div } y) = abs\ x \text{ div } abs\ y$

$\langle proof \rangle$

Suggested by Matthias Daum

lemma *int-power-div-base*:

$\llbracket 0 < m; 0 < k \rrbracket \implies k^m \text{ div } k = (k::int)^m - Suc\ 0$

$\langle proof \rangle$

by Brian Huffman

lemma *zminus-zmod*: $-\ ((x::int) \text{ mod } m) \text{ mod } m = -x \text{ mod } m$

$\langle proof \rangle$

lemma *zdiff-zmod-left*: $(x \text{ mod } m - y) \text{ mod } m = (x - y) \text{ mod } (m::int)$

$\langle proof \rangle$

lemma *zdiff-zmod-right*: $(x - y \text{ mod } m) \text{ mod } m = (x - y) \text{ mod } (m::int)$

$\langle proof \rangle$

lemmas *zmod-simps* =

mod-add-left-eq [symmetric]

mod-add-right-eq [symmetric]
 $\text{IntDiv.zmod-zmult1-eq}$ [symmetric]
 mod-mult-left-eq [symmetric]
 $\text{IntDiv.zpower-zmod}$
 $\text{zminus-zmod zdiff-zmod-left zdiff-zmod-right}$

Distributive laws for function *nat*.

lemma *nat-div-distrib*: $0 \leq x \implies \text{nat } (x \text{ div } y) = \text{nat } x \text{ div nat } y$
 $\langle \text{proof} \rangle$

lemma *nat-mod-distrib*:
 $\llbracket 0 \leq x; 0 \leq y \rrbracket \implies \text{nat } (x \text{ mod } y) = \text{nat } x \text{ mod nat } y$
 $\langle \text{proof} \rangle$

Suggested by Matthias Daum

lemma *int-div-less-self*: $\llbracket 0 < x; 1 < k \rrbracket \implies x \text{ div } k < (x::\text{int})$
 $\langle \text{proof} \rangle$

code generator setup

context *ring-1*
begin

lemma *of-int-num* [code]:
 $\text{of-int } k = (\text{if } k = 0 \text{ then } 0 \text{ else if } k < 0 \text{ then}$
 $\quad - \text{of-int } (-k) \text{ else let}$
 $\quad (l, m) = \text{divmod } k \ 2;$
 $\quad l' = \text{of-int } l$
 $\quad \text{in if } m = 0 \text{ then } l' + l' \text{ else } l' + l' + 1)$
 $\langle \text{proof} \rangle$

end

lemma *zmod-eq-dvd-iff*: $(x::\text{int}) \text{ mod } n = y \text{ mod } n \longleftrightarrow n \text{ dvd } x - y$
 $\langle \text{proof} \rangle$

lemma *nat-mod-eq-lemma*: **assumes** $xy:n$: $(x::\text{nat}) \text{ mod } n = y \text{ mod } n$ **and** $xy:y \leq x$
shows $\exists q. x = y + n * q$
 $\langle \text{proof} \rangle$

lemma *nat-mod-eq-iff*: $(x::\text{nat}) \text{ mod } n = y \text{ mod } n \longleftrightarrow (\exists q1 \ q2. x + n * q1 = y + n * q2)$
 $(\text{is } ?lhs = ?rhs)$
 $\langle \text{proof} \rangle$

31.20 Simproc setup

$\langle ML \rangle$

31.21 Code generation

definition $pdivmod :: int \Rightarrow int \Rightarrow int \times int$ **where**

$pdivmod\ k\ l = (|k| \div |l|, |k| \bmod |l|)$

lemma $pdivmod\text{-}posDivAlg$ [code]:

$pdivmod\ k\ l = (if\ l = 0\ then\ (0, |k|)\ else\ posDivAlg\ |k|\ |l|)$

$\langle proof \rangle$

lemma $divmod\text{-}pdivmod$: $divmod\ k\ l = (if\ k = 0\ then\ (0, 0)\ else\ if\ l = 0\ then\ (0, k)\ else$

$apsnd\ ((op\ *)\ (sgn\ l))\ (if\ 0 < l \wedge 0 \leq k \vee l < 0 \wedge k < 0$
 $then\ pdivmod\ k\ l$

$else\ (let\ (r, s) = pdivmod\ k\ l\ in$

$if\ s = 0\ then\ (-\ r, 0)\ else\ (-\ r - 1, |l| - s))))$

$\langle proof \rangle$

lemma $divmod\text{-}code$ [code]: $divmod\ k\ l = (if\ k = 0\ then\ (0, 0)\ else\ if\ l = 0\ then\ (0, k)\ else$

$apsnd\ ((op\ *)\ (sgn\ l))\ (if\ sgn\ k = sgn\ l$
 $then\ pdivmod\ k\ l$

$else\ (let\ (r, s) = pdivmod\ k\ l\ in$

$if\ s = 0\ then\ (-\ r, 0)\ else\ (-\ r - 1, |l| - s))))$

$\langle proof \rangle$

code-modulename *SML*

IntDiv Integer

code-modulename *OCaml*

IntDiv Integer

code-modulename *Haskell*

IntDiv Integer

end

32 NatBin: Binary arithmetic for the natural numbers

theory *NatBin*

imports *IntDiv*

uses (*Tools/nat-simprocs.ML*)

begin

Arithmetic for naturals is reduced to that for the non-negative integers.

instantiation *nat :: number*

begin

definition

nat-number-of-def [*code inline, code del*]: $\text{number-of } v = \text{nat } (\text{number-of } v)$

instance $\langle \text{proof} \rangle$

end

lemma [*code post*]:

$\text{nat } (\text{number-of } v) = \text{number-of } v$
 $\langle \text{proof} \rangle$

abbreviation (*xsymbols*)

$\text{power2} :: 'a :: \text{power} \Rightarrow 'a \ ((-^2) [1000] 999) \text{ where}$
 $x^2 == x^{\wedge}2$

notation (*latex output*)

$\text{power2} \ ((-^2) [1000] 999)$

notation (*HTML output*)

$\text{power2} \ ((-^2) [1000] 999)$

32.1 Predicate for negative binary numbers

definition $\text{neg} :: \text{int} \Rightarrow \text{bool}$ **where**

$\text{neg } Z \longleftrightarrow Z < 0$

lemma *not-neg-int* [*simp*]: $\sim \text{neg } (\text{of-nat } n)$

$\langle \text{proof} \rangle$

lemma *neg-zminus-int* [*simp*]: $\text{neg } (- (\text{of-nat } (\text{Suc } n)))$

$\langle \text{proof} \rangle$

lemmas $\text{neg-eq-less-0} = \text{neg-def}$

lemma *not-neg-eq-ge-0*: $(\sim \text{neg } x) = (0 \leq x)$

$\langle \text{proof} \rangle$

To simplify inequalities when `Numerals` can get simplified to 1

lemma *not-neg-0*: $\sim \text{neg } 0$

$\langle \text{proof} \rangle$

lemma *not-neg-1*: $\sim \text{neg } 1$

$\langle \text{proof} \rangle$

lemma *neg-nat*: $\text{neg } z ==> \text{nat } z = 0$

$\langle \text{proof} \rangle$

lemma *not-neg-nat*: $\sim \text{neg } z ==> \text{of-nat } (\text{nat } z) = z$

$\langle \text{proof} \rangle$

If *Numeral0* is rewritten to 0 then this rule can’t be applied: *Numeral0* IS *number-of Pls*

lemma *not-neg-number-of-Pls*: $\sim \text{neg } (\text{number-of Int.Pls})$
 $\langle \text{proof} \rangle$

lemma *neg-number-of-Min*: $\text{neg } (\text{number-of Int.Min})$
 $\langle \text{proof} \rangle$

lemma *neg-number-of-Bit0*:
 $\text{neg } (\text{number-of } (\text{Int.Bit0 } w)) = \text{neg } (\text{number-of } w)$
 $\langle \text{proof} \rangle$

lemma *neg-number-of-Bit1*:
 $\text{neg } (\text{number-of } (\text{Int.Bit1 } w)) = \text{neg } (\text{number-of } w)$
 $\langle \text{proof} \rangle$

lemmas *neg-simps* [simp] =
not-neg-0 not-neg-1
not-neg-number-of-Pls neg-number-of-Min
neg-number-of-Bit0 neg-number-of-Bit1

32.2 Function *nat*: Coercion from Type *int* to *nat*

declare *nat-0* [simp] *nat-1* [simp]

lemma *nat-number-of* [simp]: $\text{nat } (\text{number-of } w) = \text{number-of } w$
 $\langle \text{proof} \rangle$

lemma *nat-numeral-0-eq-0* [simp]: $\text{Numeral0} = (0::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *nat-numeral-1-eq-1* [simp]: $\text{Numeral1} = (1::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *numeral-1-eq-Suc-0*: $\text{Numeral1} = \text{Suc } 0$
 $\langle \text{proof} \rangle$

lemma *numeral-2-eq-2*: $2 = \text{Suc } (\text{Suc } 0)$
 $\langle \text{proof} \rangle$

32.3 Function *int*: Coercion from Type *nat* to *int*

lemma *int-nat-number-of* [simp]:
 $\text{int } (\text{number-of } v) =$
 $(\text{if } \text{neg } (\text{number-of } v :: \text{int}) \text{ then } 0$
 $\text{else } (\text{number-of } v :: \text{int}))$
 $\langle \text{proof} \rangle$

32.3.1 Successor

lemma *Suc-nat-eq-nat-zadd1*: $(0::int) \leq z \implies \text{Suc} (\text{nat } z) = \text{nat } (1 + z)$
 $\langle \text{proof} \rangle$

lemma *Suc-nat-number-of-add*:

$\text{Suc} (\text{number-of } v + n) =$
 $(\text{if } \text{neg} (\text{number-of } v :: \text{int}) \text{ then } 1+n \text{ else } \text{number-of } (\text{Int.succ } v) + n)$
 $\langle \text{proof} \rangle$

lemma *Suc-nat-number-of [simp]*:

$\text{Suc} (\text{number-of } v) =$
 $(\text{if } \text{neg} (\text{number-of } v :: \text{int}) \text{ then } 1 \text{ else } \text{number-of } (\text{Int.succ } v))$
 $\langle \text{proof} \rangle$

32.3.2 Addition

lemma *add-nat-number-of [simp]*:

$(\text{number-of } v :: \text{nat}) + \text{number-of } v' =$
 $(\text{if } v < \text{Int.Pl} \text{ then } \text{number-of } v'$
 $\text{else if } v' < \text{Int.Pl} \text{ then } \text{number-of } v$
 $\text{else } \text{number-of } (v + v'))$
 $\langle \text{proof} \rangle$

lemma *nat-number-of-add-1 [simp]*:

$\text{number-of } v + (1::\text{nat}) =$
 $(\text{if } v < \text{Int.Pl} \text{ then } 1 \text{ else } \text{number-of } (\text{Int.succ } v))$
 $\langle \text{proof} \rangle$

lemma *nat-1-add-number-of [simp]*:

$(1::\text{nat}) + \text{number-of } v =$
 $(\text{if } v < \text{Int.Pl} \text{ then } 1 \text{ else } \text{number-of } (\text{Int.succ } v))$
 $\langle \text{proof} \rangle$

lemma *nat-1-add-1 [simp]*: $1 + 1 = (2::\text{nat})$

$\langle \text{proof} \rangle$

32.3.3 Subtraction

lemma *diff-nat-eq-if*:

$\text{nat } z - \text{nat } z' =$
 $(\text{if } \text{neg } z' \text{ then } \text{nat } z$
 $\text{else let } d = z - z' \text{ in}$
 $\text{if } \text{neg } d \text{ then } 0 \text{ else } \text{nat } d)$

$\langle \text{proof} \rangle$

lemma *diff-nat-number-of [simp]*:

$(\text{number-of } v :: \text{nat}) - \text{number-of } v' =$
 $(\text{if } v' < \text{Int.Pl} \text{ then } \text{number-of } v$

$\text{else let } d = \text{number-of } (v + \text{uminus } v') \text{ in}$
 $\text{if neg } d \text{ then } 0 \text{ else nat } d)$
 $\langle \text{proof} \rangle$

lemma *nat-number-of-diff-1* [simp]:
 $\text{number-of } v - (1::\text{nat}) =$
 $(\text{if } v \leq \text{Int.Plus} \text{ then } 0 \text{ else number-of } (\text{Int.pred } v))$
 $\langle \text{proof} \rangle$

32.3.4 Multiplication

lemma *mult-nat-number-of* [simp]:
 $(\text{number-of } v :: \text{nat}) * \text{number-of } v' =$
 $(\text{if } v < \text{Int.Plus} \text{ then } 0 \text{ else number-of } (v * v'))$
 $\langle \text{proof} \rangle$

32.3.5 Quotient

lemma *div-nat-number-of* [simp]:
 $(\text{number-of } v :: \text{nat}) \text{ div number-of } v' =$
 $(\text{if neg } (\text{number-of } v :: \text{int}) \text{ then } 0$
 $\text{else nat } (\text{number-of } v \text{ div number-of } v'))$
 $\langle \text{proof} \rangle$

lemma *one-div-nat-number-of* [simp]:
 $\text{Suc } 0 \text{ div number-of } v' = \text{nat } (1 \text{ div number-of } v')$
 $\langle \text{proof} \rangle$

32.3.6 Remainder

lemma *mod-nat-number-of* [simp]:
 $(\text{number-of } v :: \text{nat}) \text{ mod number-of } v' =$
 $(\text{if neg } (\text{number-of } v :: \text{int}) \text{ then } 0$
 $\text{else if neg } (\text{number-of } v' :: \text{int}) \text{ then number-of } v$
 $\text{else nat } (\text{number-of } v \text{ mod number-of } v'))$
 $\langle \text{proof} \rangle$

lemma *one-mod-nat-number-of* [simp]:
 $\text{Suc } 0 \text{ mod number-of } v' =$
 $(\text{if neg } (\text{number-of } v' :: \text{int}) \text{ then Suc } 0$
 $\text{else nat } (1 \text{ mod number-of } v'))$
 $\langle \text{proof} \rangle$

32.3.7 Divisibility

lemmas *dvd-eq-mod-eq-0-number-of* =
 $\text{dvd-eq-mod-eq-0} \text{ [of number-of } x \text{ number-of } y, \text{ standard}]$

declare *dvd-eq-mod-eq-0-number-of* [simp]

$\langle ML \rangle$

32.4 Comparisons

32.4.1 Equals (=)

lemma *eq-nat-nat-iff*:

$$[| (0::int) \leq z; \ 0 \leq z' |] \implies (nat\ z = nat\ z') = (z=z')$$

 $\langle proof \rangle$

lemma *eq-nat-number-of [simp]*:

$$\begin{aligned} & ((number-of\ v :: nat) = number-of\ v') = \\ & (if\ neg\ (number-of\ v :: int)\ then\ (number-of\ v' :: int) \leq 0 \\ & \quad else\ if\ neg\ (number-of\ v' :: int)\ then\ (number-of\ v :: int) = 0 \\ & \quad else\ v = v') \end{aligned}$$

 $\langle proof \rangle$

32.4.2 Less-than (<)

lemma *less-nat-number-of [simp]*:

$$\begin{aligned} & (number-of\ v :: nat) < number-of\ v' \iff \\ & (if\ v < v'\ then\ Int.Pl\ < v'\ else\ False) \end{aligned}$$

 $\langle proof \rangle$

32.4.3 Less-than-or-equal

lemma *le-nat-number-of [simp]*:

$$\begin{aligned} & (number-of\ v :: nat) \leq number-of\ v' \iff \\ & (if\ v \leq v'\ then\ True\ else\ v \leq Int.Pl) \end{aligned}$$

 $\langle proof \rangle$

lemmas *numerals* = *nat-numeral-0-eq-0* *nat-numeral-1-eq-1* *numeral-2-eq-2*

32.5 Powers with Numeric Exponents

We cannot refer to the number $2::'a$ in *Ring-and-Field.thy*. We cannot prove general results about the numeral $-1::'a$, so we have to use $-(1::'a)$ instead.

lemma *power2-eq-square*: $(a::'a::recpower)^2 = a * a$
 $\langle proof \rangle$

lemma *zero-power2 [simp]*: $(0::'a::\{semiring-1,recpower\})^2 = 0$
 $\langle proof \rangle$

lemma *one-power2 [simp]*: $(1::'a::\{semiring-1,recpower\})^2 = 1$
 $\langle proof \rangle$

lemma *power3-eq-cube*: $(x::'a::recpower) ^ 3 = x * x * x$

$\langle \text{proof} \rangle$

Squares of literal numerals will be evaluated.

lemmas *power2-eq-square-number-of* =
 power2-eq-square [of number-of w, standard]
declare *power2-eq-square-number-of* [simp]

lemma *zero-le-power2*[simp]: $0 \leq (a^2 :: 'a :: \{\text{ordered-idom}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

lemma *zero-less-power2*[simp]:
 $(0 < a^2) = (a \neq (0 :: 'a :: \{\text{ordered-idom}, \text{recpower}\}))$
 $\langle \text{proof} \rangle$

lemma *power2-less-0*[simp]:
fixes $a :: 'a :: \{\text{ordered-idom}, \text{recpower}\}$
shows $\sim (a^2 < 0)$
 $\langle \text{proof} \rangle$

lemma *zero-eq-power2*[simp]:
 $(a^2 = 0) = (a = (0 :: 'a :: \{\text{ordered-idom}, \text{recpower}\}))$
 $\langle \text{proof} \rangle$

lemma *abs-power2*[simp]:
 $\text{abs}(a^2) = (a^2 :: 'a :: \{\text{ordered-idom}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

lemma *power2-abs*[simp]:
 $(\text{abs } a)^2 = (a^2 :: 'a :: \{\text{ordered-idom}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

lemma *power2-minus*[simp]:
 $(- a)^2 = (a^2 :: 'a :: \{\text{comm-ring-1}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

lemma *power2-le-imp-le*:
fixes $x y :: 'a :: \{\text{ordered-semidom}, \text{recpower}\}$
shows $\llbracket x^2 \leq y^2; 0 \leq y \rrbracket \implies x \leq y$
 $\langle \text{proof} \rangle$

lemma *power2-less-imp-less*:
fixes $x y :: 'a :: \{\text{ordered-semidom}, \text{recpower}\}$
shows $\llbracket x^2 < y^2; 0 \leq y \rrbracket \implies x < y$
 $\langle \text{proof} \rangle$

lemma *power2-eq-imp-eq*:
fixes $x y :: 'a :: \{\text{ordered-semidom}, \text{recpower}\}$
shows $\llbracket x^2 = y^2; 0 \leq x; 0 \leq y \rrbracket \implies x = y$

$\langle \text{proof} \rangle$

lemma *power-minus1-even*[simp]: $(-1) \wedge (2*n) = (1::'a::\{\text{comm-ring-1}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

lemma *power-minus1-odd*: $(-1) \wedge \text{Suc}(2*n) = -(1::'a::\{\text{comm-ring-1}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

lemma *power-even-eq*: $(a::'a::\text{recpower}) \wedge (2*n) = (a \wedge n) \wedge 2$
 $\langle \text{proof} \rangle$

lemma *power-odd-eq*: $(a::\text{int}) \wedge \text{Suc}(2*n) = a * (a \wedge n) \wedge 2$
 $\langle \text{proof} \rangle$

lemma *power-minus-even* [simp]:
 $(-a) \wedge (2*n) = (a::'a::\{\text{comm-ring-1}, \text{recpower}\}) \wedge (2*n)$
 $\langle \text{proof} \rangle$

lemma *zero-le-even-power'*[simp]:
 $0 \leq (a::'a::\{\text{ordered-idom}, \text{recpower}\}) \wedge (2*n)$
 $\langle \text{proof} \rangle$

lemma *odd-power-less-zero*:
 $(a::'a::\{\text{ordered-idom}, \text{recpower}\}) < 0 \implies a \wedge \text{Suc}(2*n) < 0$
 $\langle \text{proof} \rangle$

lemma *odd-0-le-power-imp-0-le*:
 $0 \leq a \wedge \text{Suc}(2*n) \implies 0 \leq (a::'a::\{\text{ordered-idom}, \text{recpower}\})$
 $\langle \text{proof} \rangle$

Simprules for comparisons where common factors can be cancelled.

lemmas *zero-compare-simps* =
add-strict-increasing add-strict-increasing2 add-increasing
zero-le-mult-iff zero-le-divide-iff
zero-less-mult-iff zero-less-divide-iff
mult-le-0-iff divide-le-0-iff
mult-less-0-iff divide-less-0-iff
zero-le-power2 power2-less-0

32.5.1 Nat

lemma *Suc-pred'*: $0 < n \implies n = \text{Suc}(n - 1)$
 $\langle \text{proof} \rangle$

lemmas *expand-Suc* = *Suc-pred'* [of number-of v, standard]

32.5.2 Arith

lemma *Suc-eq-add-numeral-1*: $\text{Suc } n = n + 1$

$\langle proof \rangle$

lemma *Suc-eq-add-numeral-1-left*: $Suc\ n = 1 + n$
 $\langle proof \rangle$

lemma *add-eq-if*: $(m::nat) + n = (if\ m=0\ then\ n\ else\ Suc\ ((m - 1) + n))$
 $\langle proof \rangle$

lemma *mult-eq-if*: $(m::nat) * n = (if\ m=0\ then\ 0\ else\ n + ((m - 1) * n))$
 $\langle proof \rangle$

lemma *power-eq-if*: $(p \wedge m :: nat) = (if\ m=0\ then\ 1\ else\ p * (p \wedge (m - 1)))$
 $\langle proof \rangle$

32.6 Comparisons involving $(0::nat)$

Simplification already does $n < (0::'a)$, $n \leq (0::'a)$ and $(0::'a) \leq n$.

lemma *eq-number-of-0* [simp]:
 $number-of\ v = (0::nat) \longleftrightarrow v \leq Int.Pl$
 $\langle proof \rangle$

lemma *eq-0-number-of* [simp]:
 $(0::nat) = number-of\ v \longleftrightarrow v \leq Int.Pl$
 $\langle proof \rangle$

lemma *less-0-number-of* [simp]:
 $(0::nat) < number-of\ v \longleftrightarrow Int.Pl < v$
 $\langle proof \rangle$

lemma *neg-imp-number-of-eq-0*: $neg\ (number-of\ v :: int) ==> number-of\ v = (0::nat)$
 $\langle proof \rangle$

32.7 Comparisons involving *Suc*

lemma *eq-number-of-Suc* [simp]:
 $(number-of\ v = Suc\ n) =$
 $(let\ pv = number-of\ (Int.pred\ v)\ in$
 $if\ neg\ pv\ then\ False\ else\ nat\ pv = n)$
 $\langle proof \rangle$

lemma *Suc-eq-number-of* [simp]:
 $(Suc\ n = number-of\ v) =$
 $(let\ pv = number-of\ (Int.pred\ v)\ in$
 $if\ neg\ pv\ then\ False\ else\ nat\ pv = n)$
 $\langle proof \rangle$

lemma *less-number-of-Suc* [simp]:
 (number-of $v < \text{Suc } n$) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then *True* else $\text{nat } pv < n$)
 <proof>

lemma *less-Suc-number-of* [simp]:
 ($\text{Suc } n < \text{number-of } v$) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then *False* else $n < \text{nat } pv$)
 <proof>

lemma *le-number-of-Suc* [simp]:
 (number-of $v \leq \text{Suc } n$) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then *True* else $\text{nat } pv \leq n$)
 <proof>

lemma *le-Suc-number-of* [simp]:
 ($\text{Suc } n \leq \text{number-of } v$) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then *False* else $n \leq \text{nat } pv$)
 <proof>

lemma *eq-number-of-Pls-Min*: (*Natural0* :: int) $\sim = \text{number-of Int.Min}$
 <proof>

32.8 Max and Min Combined with *Suc*

lemma *max-number-of-Suc* [simp]:
 max ($\text{Suc } n$) (number-of v) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then $\text{Suc } n$ else $\text{Suc}(\text{max } n (\text{nat } pv))$)
 <proof>

lemma *max-Suc-number-of* [simp]:
 max (number-of v) ($\text{Suc } n$) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then $\text{Suc } n$ else $\text{Suc}(\text{max } (\text{nat } pv) n)$)
 <proof>

lemma *min-number-of-Suc* [simp]:
 min ($\text{Suc } n$) (number-of v) =
 (let $pv = \text{number-of } (\text{Int.pred } v)$ in
 if $\text{neg } pv$ then 0 else $\text{Suc}(\text{min } n (\text{nat } pv))$)
 <proof>

lemma *min-Suc-number-of* [simp]:

```

min (number-of v) (Suc n) =
  (let pv = number-of (Int.pred v) in
   if neg pv then 0 else Suc(min (nat pv) n))
⟨proof⟩

```

32.9 Literal arithmetic involving powers

lemma *nat-power-eq*: $(0::int) \leq z \implies \text{nat } (z^n) = \text{nat } z ^ n$
 ⟨proof⟩

lemma *power-nat-number-of*:
 $(\text{number-of } v :: \text{nat}) ^ n =$
 $(\text{if neg } (\text{number-of } v :: \text{int}) \text{ then } 0^n \text{ else } \text{nat } ((\text{number-of } v :: \text{int}) ^ n))$
 ⟨proof⟩

lemmas *power-nat-number-of-number-of* = *power-nat-number-of* [of - number-of
 w, standard]
declare *power-nat-number-of-number-of* [simp]

For arbitrary rings

lemma *power-number-of-even*:
fixes $z :: 'a::\{\text{number-ring}, \text{recpower}\}$
shows $z ^ \text{number-of } (\text{Int.Bit0 } w) = (\text{let } w = z ^ (\text{number-of } w) \text{ in } w * w)$
 ⟨proof⟩

lemma *power-number-of-odd*:
fixes $z :: 'a::\{\text{number-ring}, \text{recpower}\}$
shows $z ^ \text{number-of } (\text{Int.Bit1 } w) = (\text{if } (0::int) \leq \text{number-of } w$
 $\text{then } (\text{let } w = z ^ (\text{number-of } w) \text{ in } z * w * w) \text{ else } 1)$
 ⟨proof⟩

lemmas *zpower-number-of-even* = *power-number-of-even* [where 'a=int]
lemmas *zpower-number-of-odd* = *power-number-of-odd* [where 'a=int]

lemmas *power-number-of-even-number-of* [simp] =
power-number-of-even [of number-of v, standard]

lemmas *power-number-of-odd-number-of* [simp] =
power-number-of-odd [of number-of v, standard]

⟨ML⟩

declare *split-div*[of - - number-of k, standard, arith-split]
declare *split-mod*[of - - number-of k, standard, arith-split]

lemma *nat-number-of-Pls*: $\text{Numeral0} = (0::\text{nat})$
 ⟨proof⟩

lemma *nat-number-of-Min*: $\text{number-of Int.Min} = (0::\text{nat})$
 ⟨proof⟩

lemma *nat-number-of-Bit0*:
 $\text{number-of (Int.Bit0 } w) = (\text{let } n::\text{nat} = \text{number-of } w \text{ in } n + n)$
 ⟨proof⟩

lemma *nat-number-of-Bit1*:
 $\text{number-of (Int.Bit1 } w) =$
 (if $\text{neg (number-of } w :: \text{int})$ then 0
 else $\text{let } n = \text{number-of } w \text{ in Suc (} n + n))$
 ⟨proof⟩

lemmas *nat-number* =
nat-number-of-Pls nat-number-of-Min
nat-number-of-Bit0 nat-number-of-Bit1

lemma *Let-Suc [simp]*: $\text{Let (Suc } n) f == f (\text{Suc } n)$
 ⟨proof⟩

lemma *power-m1-even*: $(-1) ^ (2*n) = (1::'a::\{\text{number-ring}, \text{recpower}\})$
 ⟨proof⟩

lemma *power-m1-odd*: $(-1) ^ \text{Suc}(2*n) = (-1::'a::\{\text{number-ring}, \text{recpower}\})$
 ⟨proof⟩

32.10 Literal arithmetic and of-nat

lemma *of-nat-double*:
 $0 \leq x ==> \text{of-nat (nat (2 * x))} = \text{of-nat (nat } x) + \text{of-nat (nat } x)$
 ⟨proof⟩

lemma *nat-numeral-m1-eq-0*: $-1 = (0::\text{nat})$
 ⟨proof⟩

lemma *of-nat-number-of-lemma*:
 $\text{of-nat (number-of } v :: \text{nat}) =$
 (if $0 \leq (\text{number-of } v :: \text{int})$
 then $(\text{number-of } v :: 'a :: \text{number-ring})$
 else 0)
 ⟨proof⟩

lemma *of-nat-number-of-eq [simp]*:
 $\text{of-nat (number-of } v :: \text{nat}) =$
 (if $\text{neg (number-of } v :: \text{int})$ then 0
 else $(\text{number-of } v :: 'a :: \text{number-ring}))$

$\langle proof \rangle$

32.11 Lemmas for the Combination and Cancellation Sim-procs

lemma *nat-number-of-add-left*:

$number-of\ v + (number-of\ v' + (k::nat)) =$
 $(if\ neg\ (number-of\ v :: int)\ then\ number-of\ v' + k$
 $else\ if\ neg\ (number-of\ v' :: int)\ then\ number-of\ v + k$
 $else\ number-of\ (v + v') + k)$

$\langle proof \rangle$

lemma *nat-number-of-mult-left*:

$number-of\ v * (number-of\ v' * (k::nat)) =$
 $(if\ v < Int.Plus\ then\ 0$
 $else\ number-of\ (v * v') * k)$

$\langle proof \rangle$

32.11.1 For combine-numerals

lemma *left-add-mult-distrib*: $i*u + (j*u + k) = (i+j)*u + (k::nat)$

$\langle proof \rangle$

32.11.2 For cancel-numerals

lemma *nat-diff-add-eq1*:

$j <= (i::nat) ==> ((i*u + m) - (j*u + n)) = (((i-j)*u + m) - n)$

$\langle proof \rangle$

lemma *nat-diff-add-eq2*:

$i <= (j::nat) ==> ((i*u + m) - (j*u + n)) = (m - ((j-i)*u + n))$

$\langle proof \rangle$

lemma *nat-eq-add-iff1*:

$j <= (i::nat) ==> (i*u + m = j*u + n) = ((i-j)*u + m = n)$

$\langle proof \rangle$

lemma *nat-eq-add-iff2*:

$i <= (j::nat) ==> (i*u + m = j*u + n) = (m = (j-i)*u + n)$

$\langle proof \rangle$

lemma *nat-less-add-iff1*:

$j <= (i::nat) ==> (i*u + m < j*u + n) = ((i-j)*u + m < n)$

$\langle proof \rangle$

lemma *nat-less-add-iff2*:

$i <= (j::nat) ==> (i*u + m < j*u + n) = (m < (j-i)*u + n)$

$\langle proof \rangle$

lemma *nat-le-add-iff1*:

$j \leq (i::nat) \implies (i*u + m \leq j*u + n) = ((i-j)*u + m \leq n)$
 $\langle proof \rangle$

lemma *nat-le-add-iff2*:

$i \leq (j::nat) \implies (i*u + m \leq j*u + n) = (m \leq (j-i)*u + n)$
 $\langle proof \rangle$

32.11.3 For cancel-numeral-factors

lemma *nat-mult-le-cancel1*: $(0::nat) < k \implies (k*m \leq k*n) = (m \leq n)$
 $\langle proof \rangle$

lemma *nat-mult-less-cancel1*: $(0::nat) < k \implies (k*m < k*n) = (m < n)$
 $\langle proof \rangle$

lemma *nat-mult-eq-cancel1*: $(0::nat) < k \implies (k*m = k*n) = (m = n)$
 $\langle proof \rangle$

lemma *nat-mult-div-cancel1*: $(0::nat) < k \implies (k*m) \text{ div } (k*n) = (m \text{ div } n)$
 $\langle proof \rangle$

lemma *nat-mult-dvd-cancel-disj[simp]*:
 $(k*m) \text{ dvd } (k*n) = (k=0 \mid m \text{ dvd } (n::nat))$
 $\langle proof \rangle$

lemma *nat-mult-dvd-cancel1*: $0 < k \implies (k*m) \text{ dvd } (k*n::nat) = (m \text{ dvd } n)$
 $\langle proof \rangle$

32.11.4 For cancel-factor

lemma *nat-mult-le-cancel-disj*: $(k*m \leq k*n) = ((0::nat) < k \implies m \leq n)$
 $\langle proof \rangle$

lemma *nat-mult-less-cancel-disj*: $(k*m < k*n) = ((0::nat) < k \ \& \ m < n)$
 $\langle proof \rangle$

lemma *nat-mult-eq-cancel-disj*: $(k*m = k*n) = (k = (0::nat) \mid m = n)$
 $\langle proof \rangle$

lemma *nat-mult-div-cancel-disj[simp]*:
 $(k*m) \text{ div } (k*n) = (\text{if } k = (0::nat) \text{ then } 0 \text{ else } m \text{ div } n)$
 $\langle proof \rangle$

32.12 Simprocs for the Naturals

$\langle ML \rangle$

32.12.1 For simplifying $Suc\ m - K$ and $K - Suc\ m$

Where K above is a literal

lemma *Suc-diff-eq-diff-pred*: $\text{Numeral0} < n \implies \text{Suc } m - n = m - (n - \text{Numeral1})$
 $\langle \text{proof} \rangle$

Now just instantiating n to *number-of* v does the right simplification, but with some redundant inequality tests.

lemma *neg-number-of-pred-iff-0*:
 $\text{neg } (\text{number-of } (\text{Int.pred } v)::\text{int}) = (\text{number-of } v = (0::\text{nat}))$
 $\langle \text{proof} \rangle$

No longer required as a *simprule* because of the *inverse-fold* *simproc*

lemma *Suc-diff-number-of*:
 $\text{Int.Plus} < v \implies$
 $\text{Suc } m - (\text{number-of } v) = m - (\text{number-of } (\text{Int.pred } v))$
 $\langle \text{proof} \rangle$

lemma *diff-Suc-eq-diff-pred*: $m - \text{Suc } n = (m - 1) - n$
 $\langle \text{proof} \rangle$

32.12.2 For *nat-case* and *nat-rec*

lemma *nat-case-number-of* [simp]:
 $\text{nat-case } a \ f \ (\text{number-of } v) =$
 $(\text{let } pv = \text{number-of } (\text{Int.pred } v) \text{ in}$
 $\text{if neg } pv \text{ then } a \text{ else } f \ (\text{nat } pv))$
 $\langle \text{proof} \rangle$

lemma *nat-case-add-eq-if* [simp]:
 $\text{nat-case } a \ f \ ((\text{number-of } v) + n) =$
 $(\text{let } pv = \text{number-of } (\text{Int.pred } v) \text{ in}$
 $\text{if neg } pv \text{ then } \text{nat-case } a \ f \ n \text{ else } f \ (\text{nat } pv + n))$
 $\langle \text{proof} \rangle$

lemma *nat-rec-number-of* [simp]:
 $\text{nat-rec } a \ f \ (\text{number-of } v) =$
 $(\text{let } pv = \text{number-of } (\text{Int.pred } v) \text{ in}$
 $\text{if neg } pv \text{ then } a \text{ else } f \ (\text{nat } pv) \ (\text{nat-rec } a \ f \ (\text{nat } pv)))$
 $\langle \text{proof} \rangle$

lemma *nat-rec-add-eq-if* [simp]:
 $\text{nat-rec } a \ f \ (\text{number-of } v + n) =$
 $(\text{let } pv = \text{number-of } (\text{Int.pred } v) \text{ in}$
 $\text{if neg } pv \text{ then } \text{nat-rec } a \ f \ n$
 $\text{else } f \ (\text{nat } pv + n) \ (\text{nat-rec } a \ f \ (\text{nat } pv + n)))$
 $\langle \text{proof} \rangle$

32.12.3 Various Other Lemmas

Evens and Odds, for Mutilated Chess Board

Lemmas for specialist use, NOT as default simprules

lemma *nat-mult-2*: $2 * z = (z + z :: nat)$
 $\langle proof \rangle$

lemma *nat-mult-2-right*: $z * 2 = (z + z :: nat)$
 $\langle proof \rangle$

Case analysis on $n < (2 :: 'a)$

lemma *less-2-cases*: $(n :: nat) < 2 ==> n = 0 \mid n = Suc\ 0$
 $\langle proof \rangle$

lemma *div2-Suc-Suc* [simp]: $Suc(Suc\ m) \div 2 = Suc\ (m \div 2)$
 $\langle proof \rangle$

lemma *add-self-div-2* [simp]: $(m + m) \div 2 = (m :: nat)$
 $\langle proof \rangle$

lemma *mod2-Suc-Suc* [simp]: $Suc(Suc\ m) \bmod 2 = m \bmod 2$
 $\langle proof \rangle$

lemma *mod2-gr-0* [simp]: $!!m :: nat. (0 < m \bmod 2) = (m \bmod 2 = 1)$
 $\langle proof \rangle$

Removal of Small Numerals: 0, 1 and (in additive positions) 2

lemma *add-2-eq-Suc* [simp]: $2 + n = Suc\ (Suc\ n)$
 $\langle proof \rangle$

lemma *add-2-eq-Suc'* [simp]: $n + 2 = Suc\ (Suc\ n)$
 $\langle proof \rangle$

Can be used to eliminate long strings of Sucs, but not by default

lemma *Suc3-eq-add-3*: $Suc\ (Suc\ (Suc\ n)) = 3 + n$
 $\langle proof \rangle$

These lemmas collapse some needless occurrences of Suc: at least three Sucs, since two and fewer are rewritten back to Suc again! We already have some rules to simplify operands smaller than 3.

lemma *div-Suc-eq-div-add3* [simp]: $m \div (Suc\ (Suc\ (Suc\ n))) = m \div (3 + n)$
 $\langle proof \rangle$

lemma *mod-Suc-eq-mod-add3* [simp]: $m \bmod (Suc\ (Suc\ (Suc\ n))) = m \bmod (3 + n)$
 $\langle proof \rangle$

lemma *Suc-div-eq-add3-div*: $(Suc\ (Suc\ (Suc\ m))) \div n = (3 + m) \div n$
 $\langle proof \rangle$

lemma *Suc-mod-eq-add3-mod*: $(Suc\ (Suc\ (Suc\ m))) \bmod n = (3 + m) \bmod n$
 $\langle proof \rangle$


```

lemmas Suc-div-eq-add3-div-number-of =
  Suc-div-eq-add3-div [of - number-of v, standard]
declare Suc-div-eq-add3-div-number-of [simp]

lemmas Suc-mod-eq-add3-mod-number-of =
  Suc-mod-eq-add3-mod [of - number-of v, standard]
declare Suc-mod-eq-add3-mod-number-of [simp]

end

```

33 Groebner-Basis: Semiring normalization and Groebner Bases

```

theory Groebner-Basis
imports NatBin
uses
  Tools/Groebner-Basis/misc.ML
  Tools/Groebner-Basis/normalizer-data.ML
  (Tools/Groebner-Basis/normalizer.ML)
  (Tools/Groebner-Basis/groebner.ML)
begin

```

33.1 Semiring normalization

$\langle ML \rangle$

```

locale gb-semiring =
  fixes add mul pwr r0 r1
  assumes add-a: $(add\ x\ (add\ y\ z) = add\ (add\ x\ y)\ z)$ 
    and add-c: $add\ x\ y = add\ y\ x$  and add-0: $add\ r0\ x = x$ 
    and mul-a: $mul\ x\ (mul\ y\ z) = mul\ (mul\ x\ y)\ z$  and mul-c: $mul\ x\ y = mul\ y\ x$ 
    and mul-1: $mul\ r1\ x = x$  and mul-0: $mul\ r0\ x = r0$ 
    and mul-d: $mul\ x\ (add\ y\ z) = add\ (mul\ x\ y)\ (mul\ x\ z)$ 
    and pwr-0: $pwr\ x\ 0 = r1$  and pwr-Suc: $pwr\ x\ (Suc\ n) = mul\ x\ (pwr\ x\ n)$ 
begin

```

```

lemma mul-pwr: $mul\ (pwr\ x\ p)\ (pwr\ x\ q) = pwr\ x\ (p + q)$ 
 $\langle proof \rangle$ 

```

```

lemma pwr-mul: $pwr\ (mul\ x\ y)\ q = mul\ (pwr\ x\ q)\ (pwr\ y\ q)$ 
 $\langle proof \rangle$ 

```

```

lemma pwr-pwr: $pwr\ (pwr\ x\ p)\ q = pwr\ x\ (p * q)$ 
 $\langle proof \rangle$ 

```

33.1.1 Declaring the abstract theory**lemma** *semiring-ops*:

shows *TERM* (*add* *x* *y*) **and** *TERM* (*mul* *x* *y*) **and** *TERM* (*pwr* *x* *n*)
and *TERM* *r0* **and** *TERM* *r1* *<proof>*

lemma *semiring-rules*:

add (*mul* *a* *m*) (*mul* *b* *m*) = *mul* (*add* *a* *b*) *m*
add (*mul* *a* *m*) *m* = *mul* (*add* *a* *r1*) *m*
add *m* (*mul* *a* *m*) = *mul* (*add* *a* *r1*) *m*
add *m* *m* = *mul* (*add* *r1* *r1*) *m*
add *r0* *a* = *a*
add *a* *r0* = *a*
mul *a* *b* = *mul* *b* *a*
mul (*add* *a* *b*) *c* = *add* (*mul* *a* *c*) (*mul* *b* *c*)
mul *r0* *a* = *r0*
mul *a* *r0* = *r0*
mul *r1* *a* = *a*
mul *a* *r1* = *a*
mul (*mul* *lx* *ly*) (*mul* *rx* *ry*) = *mul* (*mul* *lx* *rx*) (*mul* *ly* *ry*)
mul (*mul* *lx* *ly*) (*mul* *rx* *ry*) = *mul* *lx* (*mul* *ly* (*mul* *rx* *ry*))
mul (*mul* *lx* *ly*) (*mul* *rx* *ry*) = *mul* *rx* (*mul* (*mul* *lx* *ly*) *ry*)
mul (*mul* *lx* *ly*) *rx* = *mul* (*mul* *lx* *rx*) *ly*
mul (*mul* *lx* *ly*) *rx* = *mul* *lx* (*mul* *ly* *rx*)
mul *lx* (*mul* *rx* *ry*) = *mul* (*mul* *lx* *rx*) *ry*
mul *lx* (*mul* *rx* *ry*) = *mul* *rx* (*mul* *lx* *ry*)
add (*add* *a* *b*) (*add* *c* *d*) = *add* (*add* *a* *c*) (*add* *b* *d*)
add (*add* *a* *b*) *c* = *add* *a* (*add* *b* *c*)
add *a* (*add* *c* *d*) = *add* *c* (*add* *a* *d*)
add (*add* *a* *b*) *c* = *add* (*add* *a* *c*) *b*
add *a* *c* = *add* *c* *a*
add *a* (*add* *c* *d*) = *add* (*add* *a* *c*) *d*
mul (*pwr* *x* *p*) (*pwr* *x* *q*) = *pwr* *x* (*p* + *q*)
mul *x* (*pwr* *x* *q*) = *pwr* *x* (*Suc* *q*)
mul (*pwr* *x* *q*) *x* = *pwr* *x* (*Suc* *q*)
mul *x* *x* = *pwr* *x* 2
pwr (*mul* *x* *y*) *q* = *mul* (*pwr* *x* *q*) (*pwr* *y* *q*)
pwr (*pwr* *x* *p*) *q* = *pwr* *x* (*p* * *q*)
pwr *x* 0 = *r1*
pwr *x* 1 = *x*
mul *x* (*add* *y* *z*) = *add* (*mul* *x* *y*) (*mul* *x* *z*)
pwr *x* (*Suc* *q*) = *mul* *x* (*pwr* *x* *q*)
pwr *x* (2 * *n*) = *mul* (*pwr* *x* *n*) (*pwr* *x* *n*)
pwr *x* (*Suc* (2 * *n*)) = *mul* *x* (*mul* (*pwr* *x* *n*) (*pwr* *x* *n*))
<proof>

lemmas *gb-semiring-axioms'* =

gb-semiring-axioms [normalizer
semiring ops: *semiring-ops*

```

    semiring rules: semiring-rules]

end

interpretation class-semiring: gb-semiring
  op + op * op ^ 0::'a::{comm-semiring-1, recpower} 1
  <proof>

lemmas nat-arith =
  add-nat-number-of
  diff-nat-number-of
  mult-nat-number-of
  eq-nat-number-of
  less-nat-number-of

lemma not-iszero-Numeral1:  $\neg$  iszero (Numeral1::'a::number-ring)
  <proof>

lemmas comp-arith =
  Let-def arith-simps nat-arith rel-simps neg-simps if-False
  if-True add-0 add-Suc add-number-of-left mult-number-of-left
  numeral-1-eq-1[symmetric] Suc-eq-add-numeral-1
  numeral-0-eq-0[symmetric] numerals[symmetric]
  iszero-simps not-iszero-Numeral1

lemmas semiring-norm = comp-arith

<ML>

locale gb-ring = gb-semiring +
  fixes sub :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a
  and neg :: 'a  $\Rightarrow$  'a
  assumes neg-mul: neg x = mul (neg r1) x
  and sub-add: sub x y = add x (neg y)
begin

lemma ring-ops: shows TERM (sub x y) and TERM (neg x) <proof>

lemmas ring-rules = neg-mul sub-add

lemmas gb-ring-axioms' =
  gb-ring-axioms [normalizer
  semiring ops: semiring-ops
  semiring rules: semiring-rules
  ring ops: ring-ops
  ring rules: ring-rules]

end

```

interpretation *class-ring*: *gb-ring* *op* + *op* * *op* ^
 0::'a::{*comm-semiring-1*,*recpower*,*number-ring*} 1 *op* - *uminus*
 <proof>

<ML>

locale *gb-field* = *gb-ring* +
fixes *divide* :: 'a ⇒ 'a ⇒ 'a
and *inverse*:: 'a ⇒ 'a
assumes *divide-inverse*: *divide* *x* *y* = *mul* *x* (*inverse* *y*)
and *inverse-divide*: *inverse* *x* = *divide* *r1* *x*
begin

lemma *field-ops*: **shows** *TERM* (*divide* *x* *y*) **and** *TERM* (*inverse* *x*) <proof>

lemmas *field-rules* = *divide-inverse* *inverse-divide*

lemmas *gb-field-axioms'* =
gb-field-axioms [*normalizer*
semiring ops: *semiring-ops*
semiring rules: *semiring-rules*
ring ops: *ring-ops*
ring rules: *ring-rules*
field ops: *field-ops*
field rules: *field-rules*]

end

33.2 Groebner Bases

locale *semiringb* = *gb-semiring* +
assumes *add-cancel*: *add* (*x*::'a) *y* = *add* *x* *z* \longleftrightarrow *y* = *z*
and *add-mul-solve*: *add* (*mul* *w* *y*) (*mul* *x* *z*) =
add (*mul* *w* *z*) (*mul* *x* *y*) \longleftrightarrow *w* = *x* \vee *y* = *z*
begin

lemma *noteq-reduce*: *a* \neq *b* \wedge *c* \neq *d* \longleftrightarrow *add* (*mul* *a* *c*) (*mul* *b* *d*) \neq *add* (*mul* *a*
d) (*mul* *b* *c*)
 <proof>

lemma *add-scale-eq-noteq*: $\llbracket r \neq r0 ; (a = b) \wedge \sim(c = d) \rrbracket$
 \implies *add* *a* (*mul* *r* *c*) \neq *add* *b* (*mul* *r* *d*)
 <proof>

lemma *add-r0-iff*: *x* = *add* *x* *a* \longleftrightarrow *a* = *r0*

$\langle proof \rangle$

declare *gb-semiring-axioms'* [*normalizer del*]

lemmas *semiringb-axioms'* = *semiringb-axioms* [*normalizer*
semiring ops: semiring-ops
semiring rules: semiring-rules
idom rules: noteq-reduce add-scale-eq-noteq]

end

locale *ringb* = *semiringb* + *gb-ring* +
assumes *subr0-iff*: $sub\ x\ y = r0 \longleftrightarrow x = y$
begin

declare *gb-ring-axioms'* [*normalizer del*]

lemmas *ringb-axioms'* = *ringb-axioms* [*normalizer*
semiring ops: semiring-ops
semiring rules: semiring-rules
ring ops: ring-ops
ring rules: ring-rules
idom rules: noteq-reduce add-scale-eq-noteq
ideal rules: subr0-iff add-r0-iff]

end

lemma *no-zero-divisors-neq0*:
assumes *az*: (*a*::'*a*::no-zero-divisors) $\neq 0$
and *ab*: $a*b = 0$ **shows** $b = 0$
 $\langle proof \rangle$

interpretation *class-ringb*: *ringb*
op + *op* * *op* ^ *0*::'*a*::{*idom*,*recpower*,*number-ring*} 1 *op* – *uminus*
 $\langle proof \rangle$

$\langle ML \rangle$

interpretation *natgb*: *semiringb*
op + *op* * *op* ^ *0*::*nat* 1
 $\langle proof \rangle$

$\langle ML \rangle$

locale *fieldgb* = *ringb* + *gb-field*
begin

declare *gb-field-axioms'* [*normalizer del*]

```

lemmas fieldgb-axioms' = fieldgb-axioms [normalizer
  semiring ops: semiring-ops
  semiring rules: semiring-rules
  ring ops: ring-ops
  ring rules: ring-rules
  field ops: field-ops
  field rules: field-rules
  idom rules: noteq-reduce add-scale-eq-noteq
  ideal rules: subr0-iff add-r0-iff]

```

end

```

lemmas bool-simps = simp-thms(1-34)

```

```

lemma dnf:

```

$$(P \& (Q \mid R)) = ((P\&Q) \mid (P\&R)) \quad ((Q \mid R) \& P) = ((Q\&P) \mid (R\&P))$$

$$(P \wedge Q) = (Q \wedge P) \quad (P \vee Q) = (Q \vee P)$$

<proof>

```

lemmas weak-dnf-simps = dnf bool-simps

```

```

lemma nnf-simps:

```

$$(\neg(P \wedge Q)) = (\neg P \vee \neg Q) \quad (\neg(P \vee Q)) = (\neg P \wedge \neg Q) \quad (P \longrightarrow Q) = (\neg P \vee Q)$$

$$(P = Q) = ((P \wedge Q) \vee (\neg P \wedge \neg Q)) \quad (\neg \neg(P)) = P$$

<proof>

```

lemma PFalse:

```

$$P \equiv \text{False} \implies \neg P$$

$$\neg P \implies (P \equiv \text{False})$$

<proof>

<ML>

```

declare dvd-def[algebra]
declare dvd-eq-mod-eq-0[symmetric, algebra]
declare mod-div-trivial[algebra]
declare mod-mod-trivial[algebra]
declare conjunct1[OF DIVISION-BY-ZERO, algebra]
declare conjunct2[OF DIVISION-BY-ZERO, algebra]
declare zmod-zdiv-equality[symmetric, algebra]
declare zdiv-zmod-equality[symmetric, algebra]
declare zdiv-zminus-zminus[algebra]
declare zmod-zminus-zminus[algebra]
declare zdiv-zminus2[algebra]
declare zmod-zminus2[algebra]
declare zdiv-zero[algebra]
declare zmod-zero[algebra]
declare mod-by-1[algebra]
declare div-by-1[algebra]
declare zmod-minus1-right[algebra]

```

```

declare zdiv-minus1-right[algebra]
declare mod-div-trivial[algebra]
declare mod-mod-trivial[algebra]
declare mod-mult-self2-is-0[algebra]
declare mod-mult-self1-is-0[algebra]
declare zmod-eq-0-iff[algebra]
declare dvd-0-left-iff[algebra]
declare zdvd1-eq[algebra]
declare zmod-eq-dvd-iff[algebra]
declare nat-mod-eq-iff[algebra]

```

33.3 Groebner Bases for fields

interpretation class-fieldgb:

```

  fieldgb op + op * op ^ 0::'a::{field,recpower,number-ring} 1 op - uminus op /
  inverse <proof>

```

lemma divide-Numeral1: $(x::'a::{field,number-ring}) / \text{Numeral1} = x$ <proof>

lemma divide-Numeral0: $(x::'a::{field,number-ring, division-by-zero}) / \text{Numeral0} = 0$
 <proof>

lemma mult-frac-frac: $((x::'a::{field,division-by-zero}) / y) * (z / w) = (x*z) / (y*w)$
 <proof>

lemma mult-frac-num: $((x::'a::{field, division-by-zero}) / y) * z = (x*z) / y$
 <proof>

lemma mult-num-frac: $((x::'a::{field, division-by-zero}) / y) * z = (x*z) / y$
 <proof>

lemma Numeral1-eq1-nat: $(1::nat) = \text{Numeral1}$ <proof>

lemma add-frac-num: $y \neq 0 \implies (x::'a::{field, division-by-zero}) / y + z = (x + z*y) / y$
 <proof>

lemma add-num-frac: $y \neq 0 \implies z + (x::'a::{field, division-by-zero}) / y = (x + z*y) / y$
 <proof>
 <ML>

end

34 SetInterval: Set intervals

theory SetInterval

imports Int

begin

context ord

begin**definition**

lessThan :: 'a => 'a set ((1{..<})) **where**
 $\{..$

definition

atMost :: 'a => 'a set ((1{..})) **where**
 $\{..u\} == \{x. x \leq u\}$

definition

greaterThan :: 'a => 'a set ((1{<..})) **where**
 $\{l<..\} == \{x. l < x\}$

definition

atLeast :: 'a => 'a set ((1{>..})) **where**
 $\{l>..\} == \{x. l < x\}$

definition

greaterThanLessThan :: 'a => 'a => 'a set ((1{<..<})) **where**
 $\{l<..$

definition

atLeastLessThan :: 'a => 'a => 'a set ((1{>..<})) **where**
 $\{l>..$

definition

greaterThanAtMost :: 'a => 'a => 'a set ((1{<..})) **where**
 $\{l<..u\} == \{l<..\} \text{ Int } \{..u\}$

definition

atLeastAtMost :: 'a => 'a => 'a set ((1{>..})) **where**
 $\{l>..u\} == \{l>..\} \text{ Int } \{..u\}$

end

A note of warning when using $\{.. on type *nat*: it is equivalent to $\{0.. but some lemmas involving $\{m.. may not exist in $\{..-form as well.$$$$

syntax

@UNION-le :: 'a => 'a => 'b set => 'b set ((3UN -<= ./ -) 10)
 @UNION-less :: 'a => 'a => 'b set => 'b set ((3UN -< ./ -) 10)
 @INTER-le :: 'a => 'a => 'b set => 'b set ((3INT -<= ./ -) 10)
 @INTER-less :: 'a => 'a => 'b set => 'b set ((3INT -< ./ -) 10)

syntax (*xsymbols*)

@UNION-le :: 'a => 'a => 'b set => 'b set ((3 \bigcup - \leq ./ -) 10)
 @UNION-less :: 'a => 'a => 'b set => 'b set ((3 \bigcup -< ./ -) 10)
 @INTER-le :: 'a => 'a => 'b set => 'b set ((3 \bigcap - \leq ./ -) 10)
 @INTER-less :: 'a => 'a => 'b set => 'b set ((3 \bigcap -< ./ -) 10)

syntax (*latex output*)

$\text{@UNION-le} :: 'a \Rightarrow 'a \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \bigcup (00- \leq -) / -) 10)$
 $\text{@UNION-less} :: 'a \Rightarrow 'a \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \bigcup (00- < -) / -) 10)$
 $\text{@INTER-le} :: 'a \Rightarrow 'a \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \bigcap (00- \leq -) / -) 10)$
 $\text{@INTER-less} :: 'a \Rightarrow 'a \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set} \quad ((\exists \bigcap (00- < -) / -) 10)$

translations

$UN\ i \leq n. A == UN\ i: \{..n\}. A$
 $UN\ i < n. A == UN\ i: \{..<n\}. A$
 $INT\ i \leq n. A == INT\ i: \{..n\}. A$
 $INT\ i < n. A == INT\ i: \{..<n\}. A$

34.1 Various equivalences

lemma (*in ord*) *lessThan-iff* [*iff*]: $(i: \text{lessThan } k) = (i < k)$
<proof>

lemma *Compl-lessThan* [*simp*]:
 $!!k:: 'a::\text{linorder}. \neg \text{lessThan } k = \text{atLeast } k$
<proof>

lemma *single-Diff-lessThan* [*simp*]: $!!k:: 'a::\text{order}. \{k\} - \text{lessThan } k = \{k\}$
<proof>

lemma (*in ord*) *greaterThan-iff* [*iff*]: $(i: \text{greaterThan } k) = (k < i)$
<proof>

lemma *Compl-greaterThan* [*simp*]:
 $!!k:: 'a::\text{linorder}. \neg \text{greaterThan } k = \text{atMost } k$
<proof>

lemma *Compl-atMost* [*simp*]: $!!k:: 'a::\text{linorder}. \neg \text{atMost } k = \text{greaterThan } k$
<proof>

lemma (*in ord*) *atLeast-iff* [*iff*]: $(i: \text{atLeast } k) = (k \leq i)$
<proof>

lemma *Compl-atLeast* [*simp*]:
 $!!k:: 'a::\text{linorder}. \neg \text{atLeast } k = \text{lessThan } k$
<proof>

lemma (*in ord*) *atMost-iff* [*iff*]: $(i: \text{atMost } k) = (i \leq k)$
<proof>

lemma *atMost-Int-atLeast*: $!!n:: 'a::\text{order}. \text{atMost } n \text{ Int } \text{atLeast } n = \{n\}$
<proof>

34.2 Logical Equivalences for Set Inclusion and Equality

lemma *atLeast-subset-iff* [*iff*]:

$(atLeast\ x \subseteq atLeast\ y) = (y \leq (x::'a::order))$
 $\langle proof \rangle$

lemma *atLeast-eq-iff* [iff]:
 $(atLeast\ x = atLeast\ y) = (x = (y::'a::linorder))$
 $\langle proof \rangle$

lemma *greaterThan-subset-iff* [iff]:
 $(greaterThan\ x \subseteq greaterThan\ y) = (y \leq (x::'a::linorder))$
 $\langle proof \rangle$

lemma *greaterThan-eq-iff* [iff]:
 $(greaterThan\ x = greaterThan\ y) = (x = (y::'a::linorder))$
 $\langle proof \rangle$

lemma *atMost-subset-iff* [iff]: $(atMost\ x \subseteq atMost\ y) = (x \leq (y::'a::order))$
 $\langle proof \rangle$

lemma *atMost-eq-iff* [iff]: $(atMost\ x = atMost\ y) = (x = (y::'a::linorder))$
 $\langle proof \rangle$

lemma *lessThan-subset-iff* [iff]:
 $(lessThan\ x \subseteq lessThan\ y) = (x \leq (y::'a::linorder))$
 $\langle proof \rangle$

lemma *lessThan-eq-iff* [iff]:
 $(lessThan\ x = lessThan\ y) = (x = (y::'a::linorder))$
 $\langle proof \rangle$

34.3 Two-sided intervals

context *ord*

begin

lemma *greaterThanLessThan-iff* [simp,noatp]:
 $(i : \{l <..<u\}) = (l < i \ \& \ i < u)$
 $\langle proof \rangle$

lemma *atLeastLessThan-iff* [simp,noatp]:
 $(i : \{l..<u\}) = (l \leq i \ \& \ i < u)$
 $\langle proof \rangle$

lemma *greaterThanAtMost-iff* [simp,noatp]:
 $(i : \{l <..u\}) = (l < i \ \& \ i \leq u)$
 $\langle proof \rangle$

lemma *atLeastAtMost-iff* [simp,noatp]:
 $(i : \{l..u\}) = (l \leq i \ \& \ i \leq u)$
 $\langle proof \rangle$

The above four lemmas could be declared as *iffs*. If we do so, a call to *blast* in *Hyperreal/Star.ML*, lemma *STAR-Int* seems to take forever (more than one hour).

end

34.3.1 Emptiness and singletons

context *order*

begin

lemma *atLeastAtMost-empty* [simp]: $n < m \implies \{m..n\} = \{\}$
<proof>

lemma *atLeastLessThan-empty*[simp]: $n \leq m \implies \{m.. $n\} = \{\}$
<proof>$

lemma *greaterThanAtMost-empty*[simp]: $l \leq k \implies \{k<.. $l\} = \{\}$
<proof>$

lemma *greaterThanLessThan-empty*[simp]: $l \leq k \implies \{k<.. $l\} = \{\}$
<proof>$

lemma *atLeastAtMost-singleton* [simp]: $\{a..a\} = \{a\}$
<proof>

end

34.4 Intervals of natural numbers

34.4.1 The Constant *lessThan*

lemma *lessThan-0* [simp]: $\text{lessThan } (0::\text{nat}) = \{\}$
<proof>

lemma *lessThan-Suc*: $\text{lessThan } (\text{Suc } k) = \text{insert } k (\text{lessThan } k)$
<proof>

lemma *lessThan-Suc-atMost*: $\text{lessThan } (\text{Suc } k) = \text{atMost } k$
<proof>

lemma *UN-lessThan-UNIV*: $(\text{UN } m::\text{nat}. \text{lessThan } m) = \text{UNIV}$
<proof>

34.4.2 The Constant *greaterThan*

lemma *greaterThan-0* [simp]: $\text{greaterThan } 0 = \text{range } \text{Suc}$
<proof>

lemma *greaterThan-Suc*: $\text{greaterThan } (\text{Suc } k) = \text{greaterThan } k - \{\text{Suc } k\}$

$\langle proof \rangle$

lemma *INT-greaterThan-UNIV*: $(INT\ m::nat.\ greaterThan\ m) = \{\}$
 $\langle proof \rangle$

34.4.3 The Constant *atLeast*

lemma *atLeast-0 [simp]*: $atLeast\ (0::nat) = UNIV$
 $\langle proof \rangle$

lemma *atLeast-Suc*: $atLeast\ (Suc\ k) = atLeast\ k - \{k\}$
 $\langle proof \rangle$

lemma *atLeast-Suc-greaterThan*: $atLeast\ (Suc\ k) = greaterThan\ k$
 $\langle proof \rangle$

lemma *UN-atLeast-UNIV*: $(UN\ m::nat.\ atLeast\ m) = UNIV$
 $\langle proof \rangle$

34.4.4 The Constant *atMost*

lemma *atMost-0 [simp]*: $atMost\ (0::nat) = \{0\}$
 $\langle proof \rangle$

lemma *atMost-Suc*: $atMost\ (Suc\ k) = insert\ (Suc\ k)\ (atMost\ k)$
 $\langle proof \rangle$

lemma *UN-atMost-UNIV*: $(UN\ m::nat.\ atMost\ m) = UNIV$
 $\langle proof \rangle$

34.4.5 The Constant *atLeastLessThan*

The orientation of the following 2 rules is tricky. The lhs is defined in terms of the rhs. Hence the chosen orientation makes sense in this theory — the reverse orientation complicates proofs (eg nontermination). But outside, when the definition of the lhs is rarely used, the opposite orientation seems preferable because it reduces a specific concept to a more general one.

lemma *atLeast0LessThan*: $\{0::nat..<n\} = \{..
 $\langle proof \rangle$$

lemma *atLeast0AtMost*: $\{0..n::nat\} = \{..n\}$
 $\langle proof \rangle$

declare *atLeast0LessThan*[*symmetric, code unfold*]
atLeast0AtMost[*symmetric, code unfold*]

lemma *atLeastLessThan0*: $\{m..<0::nat\} = \{\}$
 $\langle proof \rangle$

34.4.6 Intervals of nats with *Suc*

Not a simplrule because the RHS is too messy.

lemma *atLeastLessThanSuc*:

$\{m..<Suc\ n\} = (if\ m \leq n\ then\ insert\ n\ \{m..<n\}\ else\ \{\})$
 $\langle proof \rangle$

lemma *atLeastLessThan-singleton [simp]*: $\{m..<Suc\ m\} = \{m\}$

$\langle proof \rangle$

lemma *atLeastLessThanSuc-atLeastAtMost*: $\{l..<Suc\ u\} = \{l..u\}$

$\langle proof \rangle$

lemma *atLeastSucAtMost-greaterThanAtMost*: $\{Suc\ l..u\} = \{l<..u\}$

$\langle proof \rangle$

lemma *atLeastSucLessThan-greaterThanLessThan*: $\{Suc\ l..<u\} = \{l<..<u\}$

$\langle proof \rangle$

lemma *atLeastAtMostSuc-conv*: $m \leq Suc\ n \implies \{m..Suc\ n\} = insert\ (Suc\ n)\ \{m..n\}$

$\langle proof \rangle$

34.4.7 Image

lemma *image-add-atLeastAtMost*:

$(\%n::nat.\ n+k)\ ' \{i..j\} = \{i+k..j+k\}\ (is\ ?A = ?B)$
 $\langle proof \rangle$

lemma *image-add-atLeastLessThan*:

$(\%n::nat.\ n+k)\ ' \{i..<j\} = \{i+k..<j+k\}\ (is\ ?A = ?B)$
 $\langle proof \rangle$

corollary *image-Suc-atLeastAtMost[simp]*:

$Suc\ ' \{i..j\} = \{Suc\ i..Suc\ j\}$
 $\langle proof \rangle$

corollary *image-Suc-atLeastLessThan[simp]*:

$Suc\ ' \{i..<j\} = \{Suc\ i..<Suc\ j\}$
 $\langle proof \rangle$

lemma *image-add-int-atLeastLessThan*:

$(\%x.\ x + (l::int))\ ' \{0..<u-l\} = \{l..<u\}$
 $\langle proof \rangle$

34.4.8 Finiteness

lemma *finite-lessThan [iff]*: **fixes** $k :: nat$ **shows** *finite* $\{..<k\}$

$\langle proof \rangle$

lemma *finite-atMost* [iff]: **fixes** $k :: nat$ **shows** *finite* $\{..k\}$
 ⟨proof⟩

lemma *finite-greaterThanLessThan* [iff]:
fixes $l :: nat$ **shows** *finite* $\{l < .. < u\}$
 ⟨proof⟩

lemma *finite-atLeastLessThan* [iff]:
fixes $l :: nat$ **shows** *finite* $\{l .. < u\}$
 ⟨proof⟩

lemma *finite-greaterThanAtMost* [iff]:
fixes $l :: nat$ **shows** *finite* $\{l < .. u\}$
 ⟨proof⟩

lemma *finite-atLeastAtMost* [iff]:
fixes $l :: nat$ **shows** *finite* $\{l .. u\}$
 ⟨proof⟩

A bounded set of natural numbers is finite.

lemma *bounded-nat-set-is-finite*:
 (ALL $i:N. i < (n::nat)$) ==> *finite* N
 ⟨proof⟩

lemma *finite-less-ub*:
 !! $f::nat=>nat. (!n. n \leq f n) ==> finite \{n. f n \leq u\}$
 ⟨proof⟩

Any subset of an interval of natural numbers the size of the subset is exactly that interval.

lemma *subset-card-intvl-is-intvl*:
 $A \leq \{k..<k+card\ A\} \implies A = \{k..<k+card\ A\}$ (is PROP ?P)
 ⟨proof⟩

34.4.9 Cardinality

lemma *card-lessThan* [simp]: *card* $\{..<u\} = u$
 ⟨proof⟩

lemma *card-atMost* [simp]: *card* $\{..u\} = Suc\ u$
 ⟨proof⟩

lemma *card-atLeastLessThan* [simp]: *card* $\{l..<u\} = u - l$
 ⟨proof⟩

lemma *card-atLeastAtMost* [simp]: *card* $\{l..u\} = Suc\ u - l$
 ⟨proof⟩

lemma *card-greaterThanAtMost* [simp]: $\text{card } \{l < .. u\} = u - l$
 ⟨proof⟩

lemma *card-greaterThanLessThan* [simp]: $\text{card } \{l < .. < u\} = u - \text{Suc } l$
 ⟨proof⟩

lemma *ex-bij-betw-nat-finite*:
 $\text{finite } M \implies \exists h. \text{bij-betw } h \{0 .. < \text{card } M\} M$
 ⟨proof⟩

lemma *ex-bij-betw-finite-nat*:
 $\text{finite } M \implies \exists h. \text{bij-betw } h M \{0 .. < \text{card } M\}$
 ⟨proof⟩

34.5 Intervals of integers

lemma *atLeastLessThanPlusOne-atLeastAtMost-int*: $\{l .. < u + 1\} = \{l .. (u :: \text{int})\}$
 ⟨proof⟩

lemma *atLeastPlusOneAtMost-greaterThanAtMost-int*: $\{l + 1 .. u\} = \{l < .. (u :: \text{int})\}$
 ⟨proof⟩

lemma *atLeastPlusOneLessThan-greaterThanLessThan-int*:
 $\{l + 1 .. < u\} = \{l < .. < u :: \text{int}\}$
 ⟨proof⟩

34.5.1 Finiteness

lemma *image-atLeastZeroLessThan-int*: $0 \leq u \implies$
 $\{(0 :: \text{int}) .. < u\} = \text{int } ` \{.. < \text{nat } u\}$
 ⟨proof⟩

lemma *finite-atLeastZeroLessThan-int*: $\text{finite } \{(0 :: \text{int}) .. < u\}$
 ⟨proof⟩

lemma *finite-atLeastLessThan-int* [iff]: $\text{finite } \{l .. < u :: \text{int}\}$
 ⟨proof⟩

lemma *finite-atLeastAtMost-int* [iff]: $\text{finite } \{l .. (u :: \text{int})\}$
 ⟨proof⟩

lemma *finite-greaterThanAtMost-int* [iff]: $\text{finite } \{l < .. (u :: \text{int})\}$
 ⟨proof⟩

lemma *finite-greaterThanLessThan-int* [iff]: $\text{finite } \{l < .. < u :: \text{int}\}$
 ⟨proof⟩

34.5.2 Cardinality

lemma *card-atLeastZeroLessThan-int*: $\text{card } \{(0::\text{int})..<u\} = \text{nat } u$
 $\langle \text{proof} \rangle$

lemma *card-atLeastLessThan-int* [simp]: $\text{card } \{l..<u\} = \text{nat } (u - l)$
 $\langle \text{proof} \rangle$

lemma *card-atLeastAtMost-int* [simp]: $\text{card } \{l..u\} = \text{nat } (u - l + 1)$
 $\langle \text{proof} \rangle$

lemma *card-greaterThanAtMost-int* [simp]: $\text{card } \{l<..u\} = \text{nat } (u - l)$
 $\langle \text{proof} \rangle$

lemma *card-greaterThanLessThan-int* [simp]: $\text{card } \{l<..
 $\langle \text{proof} \rangle$$

lemma *finite-M-bounded-by-nat*: $\text{finite } \{k. P\ k \wedge k < (i::\text{nat})\}$
 $\langle \text{proof} \rangle$

lemma *card-less*:
assumes *zero-in-M*: $0 \in M$
shows $\text{card } \{k \in M. k < \text{Suc } i\} \neq 0$
 $\langle \text{proof} \rangle$

lemma *card-less-Suc2*: $0 \notin M \implies \text{card } \{k. \text{Suc } k \in M \wedge k < i\} = \text{card } \{k \in M. k < \text{Suc } i\}$
 $\langle \text{proof} \rangle$

lemma *card-less-Suc*:
assumes *zero-in-M*: $0 \in M$
shows $\text{Suc } (\text{card } \{k. \text{Suc } k \in M \wedge k < i\}) = \text{card } \{k \in M. k < \text{Suc } i\}$
 $\langle \text{proof} \rangle$

34.6 Lemmas useful with the summation operator setsum

For examples, see Algebra/poly/UnivPoly2.thy

34.6.1 Disjoint Unions

Singletons and open intervals

lemma *ivl-disj-un-singleton*:
 $\{l::'a::\text{linorder}\} \text{ Un } \{l<..\} = \{l..\}$
 $\{..
 $(l::'a::\text{linorder}) < u \implies \{l\} \text{ Un } \{l<..
 $(l::'a::\text{linorder}) < u \implies \{l<..
 $(l::'a::\text{linorder}) \leq u \implies \{l\} \text{ Un } \{l<..u\} = \{l..u\}$
 $(l::'a::\text{linorder}) \leq u \implies \{l..
 $\langle \text{proof} \rangle$$$$$

One- and two-sided intervals

lemma *ivl-disj-un-one*:

$$\begin{aligned}
(l::'a::\text{linorder}) < u &\implies \{..l\} \text{ Un } \{l<..\} = \{..\} \\
(l::'a::\text{linorder}) <= u &\implies \{..\} \text{ Un } \{l..\} = \{..\} \\
(l::'a::\text{linorder}) < u &\implies \{..l\} \text{ Un } \{l<..\} = \{..\} \\
(l::'a::\text{linorder}) <= u &\implies \{..\} \text{ Un } \{l..\} = \{..\} \\
(l::'a::\text{linorder}) < u &\implies \{l<..\} \text{ Un } \{u<..\} = \{l<..\} \\
(l::'a::\text{linorder}) < u &\implies \{l<..\} \text{ Un } \{u..\} = \{l<..\} \\
(l::'a::\text{linorder}) <= u &\implies \{l..\} \text{ Un } \{u<..\} = \{l..\} \\
(l::'a::\text{linorder}) <= u &\implies \{l..\} \text{ Un } \{u..\} = \{l..\}
\end{aligned}$$

<proof>

Two- and two-sided intervals

lemma *ivl-disj-un-two*:

$$\begin{aligned}
\llbracket (l::'a::\text{linorder}) < m; m <= u \rrbracket &\implies \{l<..\} \text{ Un } \{m..\} = \{l<..\} \\
\llbracket (l::'a::\text{linorder}) <= m; m < u \rrbracket &\implies \{l<..\} \text{ Un } \{m<..\} = \{l<..\} \\
\llbracket (l::'a::\text{linorder}) <= m; m <= u \rrbracket &\implies \{l<..\} \text{ Un } \{m..\} = \{l<..\} \\
\llbracket (l::'a::\text{linorder}) <= m; m < u \rrbracket &\implies \{l..\} \text{ Un } \{m<..\} = \{l..\} \\
\llbracket (l::'a::\text{linorder}) < m; m <= u \rrbracket &\implies \{l<..\} \text{ Un } \{m..\} = \{l<..\} \\
\llbracket (l::'a::\text{linorder}) <= m; m <= u \rrbracket &\implies \{l<..\} \text{ Un } \{m<..\} = \{l<..\} \\
\llbracket (l::'a::\text{linorder}) <= m; m <= u \rrbracket &\implies \{l<..\} \text{ Un } \{m..\} = \{l<..\} \\
\llbracket (l::'a::\text{linorder}) <= m; m <= u \rrbracket &\implies \{l..\} \text{ Un } \{m<..\} = \{l..\}
\end{aligned}$$

<proof>

lemmas *ivl-disj-un = ivl-disj-un-singleton ivl-disj-un-one ivl-disj-un-two*

34.6.2 Disjoint Intersections

Singletons and open intervals

lemma *ivl-disj-int-singleton*:

$$\begin{aligned}
\{l::'a::\text{order}\} \text{ Int } \{l<..\} &= \{\} \\
\{..\} \text{ Int } \{u\} &= \{\} \\
\{l\} \text{ Int } \{l<..\} &= \{\} \\
\{l<..\} \text{ Int } \{u\} &= \{\} \\
\{l\} \text{ Int } \{l<..\} &= \{\} \\
\{l<..\} \text{ Int } \{u\} &= \{\}
\end{aligned}$$

<proof>

One- and two-sided intervals

lemma *ivl-disj-int-one*:

$$\begin{aligned}
\{..l::'a::\text{order}\} \text{ Int } \{l<..\} &= \{\} \\
\{..\} \text{ Int } \{l..\} &= \{\} \\
\{..l\} \text{ Int } \{l<..\} &= \{\} \\
\{..\} \text{ Int } \{l..\} &= \{\} \\
\{l<..\} \text{ Int } \{u<..\} &= \{\} \\
\{l<..\} \text{ Int } \{u..\} &= \{\} \\
\{l..\} \text{ Int } \{u<..\} &= \{\} \\
\{l..\} \text{ Int } \{u..\} &= \{\}
\end{aligned}$$

$\langle proof \rangle$

Two- and two-sided intervals

lemma *ivl-disj-int-two*:

$\{l::'a::order<.. m \} \text{ Int } \{m.. u \} = \{\}$
 $\{l<.. m \} \text{ Int } \{m<.. u \} = \{\}$
 $\{l.. m \} \text{ Int } \{m.. u \} = \{\}$
 $\{l.. m \} \text{ Int } \{m<.. u \} = \{\}$
 $\{l<.. m \} \text{ Int } \{m.. u \} = \{\}$
 $\{l<.. m \} \text{ Int } \{m<.. u \} = \{\}$
 $\{l.. m \} \text{ Int } \{m.. u \} = \{\}$
 $\{l.. m \} \text{ Int } \{m<.. u \} = \{\}$
 $\langle proof \rangle$

lemmas *ivl-disj-int* = *ivl-disj-int-singleton ivl-disj-int-one ivl-disj-int-two*

34.6.3 Some Differences

lemma *ivl-diff[simp]*:

$i \leq n \implies \{i.. m \} - \{i.. n \} = \{n.. $(m::'a::linorder)$ \}$
 $\langle proof \rangle$

34.6.4 Some Subset Conditions

lemma *ivl-subset [simp,noatp]*:

$(\{i.. j \} \subseteq \{m.. n \}) = (j \leq i \mid m \leq i \ \& \ j \leq (n::'a::linorder))$
 $\langle proof \rangle$

34.7 Summation indexed over intervals

syntax

$-from-to-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((SUM - = \dots / -) [0,0,0,10] \ 10)$
 $-from-upto-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((SUM - = \dots < - / -) [0,0,0,10] \ 10)$
 $-upt-setsum :: idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((SUM - < - / -) [0,0,10] \ 10)$
 $-upto-setsum :: idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((SUM - < = - / -) [0,0,10] \ 10)$

syntax (*xsymbols*)

$-from-to-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - = \dots / -) [0,0,0,10] \ 10)$
 $-from-upto-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - = \dots < - / -) [0,0,0,10] \ 10)$
 $-upt-setsum :: idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - < - / -) [0,0,10] \ 10)$
 $-upto-setsum :: idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - < = - / -) [0,0,10] \ 10)$

syntax (*HTML output*)

$-from-to-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - = \dots / -) [0,0,0,10] \ 10)$
 $-from-upto-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - = \dots < - / -) [0,0,0,10] \ 10)$
 $-upt-setsum :: idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - < - / -) [0,0,10] \ 10)$
 $-upto-setsum :: idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathfrak{S}\sum - < = - / -) [0,0,10] \ 10)$

syntax (*latex-sum output*)

$-from-to-setsum :: idt \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$

$((\mathcal{S}\Sigma^- = - -) [0,0,0,10] 10)$
 $\text{-from-upto-setsum} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{S}\Sigma^< = - -) [0,0,0,10] 10)$
 $\text{-upt-setsum} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{S}\Sigma_- < - -) [0,0,10] 10)$
 $\text{-upto-setsum} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{S}\Sigma_- \leq - -) [0,0,10] 10)$

translations

$\sum x=a..b. t == \text{CONST setsum } (\%x. t) \{a..b\}$
 $\sum x=a..<b. t == \text{CONST setsum } (\%x. t) \{a..<b\}$
 $\sum i \leq n. t == \text{CONST setsum } (\lambda i. t) \{..n\}$
 $\sum i < n. t == \text{CONST setsum } (\lambda i. t) \{..<n\}$

The above introduces some pretty alternative syntaxes for summation over intervals:

Old	New	L ^A T _E X
$\sum x \in \{a..b\}. e$	$\sum x = a..b. e$	$\sum_{x=a}^b e$
$\sum x \in \{a..<b\}. e$	$\sum x = a..<b. e$	$\sum_{x=a}^{<b} e$
$\sum x \in \{..b\}. e$	$\sum x \leq b. e$	$\sum_{x \leq b} e$
$\sum x \in \{..<b\}. e$	$\sum x < b. e$	$\sum_{x < b} e$

The left column shows the term before introduction of the new syntax, the middle column shows the new (default) syntax, and the right column shows a special syntax. The latter is only meaningful for latex output and has to be activated explicitly by setting the print mode to *latex-sum* (e.g. via *mode = latex-sum* in antiquotations). It is not the default L^AT_EX output because it only works well with italic-style formulae, not tt-style.

Note that for uniformity on *nat* it is better to use $\sum x = 0..<n. e$ rather than $\sum x < n. e$: *setsum* may not provide all lemmas available for $\{m..<n\}$ also in the special form for $\{..<n\}$.

This congruence rule should be used for sums over intervals as the standard theorem *setsum-cong* does not work well with the simplifier who adds the unsimplified premise $x \in B$ to the context.

lemma *setsum-ivl-cong*:

$\llbracket a = c; b = d; !!x. \llbracket c \leq x; x < d \rrbracket \implies f x = g x \rrbracket \implies$
 $\text{setsum } f \{a..<b\} = \text{setsum } g \{c..<d\}$
 $\langle \text{proof} \rangle$

lemma *setsum-atMost-Suc[simp]*: $(\sum i \leq \text{Suc } n. f i) = (\sum i \leq n. f i) + f(\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma *setsum-lessThan-Suc[simp]*: $(\sum i < \text{Suc } n. f i) = (\sum i < n. f i) + f n$

$\langle \text{proof} \rangle$

lemma *setsum-cl-ivl-Suc*[simp]:

$\text{setsum } f \{m..Suc\ n\} = (\text{if } Suc\ n < m \text{ then } 0 \text{ else } \text{setsum } f \{m..n\} + f(Suc\ n))$

$\langle \text{proof} \rangle$

lemma *setsum-op-ivl-Suc*[simp]:

$\text{setsum } f \{m..<Suc\ n\} = (\text{if } n < m \text{ then } 0 \text{ else } \text{setsum } f \{m..<n\} + f(n))$

$\langle \text{proof} \rangle$

lemma *setsum-head*:

fixes $n :: nat$

assumes $mn: m \leq n$

shows $(\sum x \in \{m..n\}. P\ x) = P\ m + (\sum x \in \{m < .. n\}. P\ x)$ (**is** ?lhs = ?rhs)

$\langle \text{proof} \rangle$

lemma *setsum-head-Suc*:

$m \leq n \implies \text{setsum } f \{m..n\} = f\ m + \text{setsum } f \{Suc\ m..n\}$

$\langle \text{proof} \rangle$

lemma *setsum-head-upt-Suc*:

$m < n \implies \text{setsum } f \{m..<n\} = f\ m + \text{setsum } f \{Suc\ m..<n\}$

$\langle \text{proof} \rangle$

lemma *setsum-add-nat-ivl*: $\llbracket m \leq n; n \leq p \rrbracket \implies$

$\text{setsum } f \{m..<n\} + \text{setsum } f \{n..<p\} = \text{setsum } f \{m..<p::nat\}$

$\langle \text{proof} \rangle$

lemma *setsum-diff-nat-ivl*:

fixes $f :: nat \Rightarrow 'a::ab-group-add$

shows $\llbracket m \leq n; n \leq p \rrbracket \implies$

$\text{setsum } f \{m..<p\} - \text{setsum } f \{m..<n\} = \text{setsum } f \{n..<p\}$

$\langle \text{proof} \rangle$

34.8 Shifting bounds

lemma *setsum-shift-bounds-nat-ivl*:

$\text{setsum } f \{m+k..<n+k\} = \text{setsum } (\%i. f(i + k))\{m..<n::nat\}$

$\langle \text{proof} \rangle$

lemma *setsum-shift-bounds-cl-nat-ivl*:

$\text{setsum } f \{m+k..n+k\} = \text{setsum } (\%i. f(i + k))\{m..n::nat\}$

$\langle \text{proof} \rangle$

corollary *setsum-shift-bounds-cl-Suc-ivl*:

$\text{setsum } f \{Suc\ m..Suc\ n\} = \text{setsum } (\%i. f(Suc\ i))\{m..n\}$

$\langle \text{proof} \rangle$

corollary *setsum-shift-bounds-Suc-ivl:*

$$\text{setsum } f \{ \text{Suc } m..<\text{Suc } n \} = \text{setsum } (\%i. f(\text{Suc } i)) \{ m..<n \}$$

<proof>

lemma *setsum-shift-lb-Suc0-0:*

$$f(0::\text{nat}) = (0::\text{nat}) \implies \text{setsum } f \{ \text{Suc } 0..k \} = \text{setsum } f \{ 0..k \}$$

<proof>

lemma *setsum-shift-lb-Suc0-0-upt:*

$$f(0::\text{nat}) = 0 \implies \text{setsum } f \{ \text{Suc } 0..<k \} = \text{setsum } f \{ 0..<k \}$$

<proof>

34.9 The formula for geometric sums

lemma *geometric-sum:*

$$x \sim 1 \implies \left(\sum_{i=0..<n} x^i \right) = (x^n - 1) / (x - 1::'a::\{\text{field}, \text{recpower}\})$$

<proof>

34.10 The formula for arithmetic sums

lemma *gauss-sum:*

$$((1::'a::\text{comm-semiring-1}) + 1) * (\sum_{i \in \{1..n\}} \text{of-nat } i) = \text{of-nat } n * ((\text{of-nat } n) + 1)$$

<proof>

theorem *arith-series-general:*

$$((1::'a::\text{comm-semiring-1}) + 1) * (\sum_{i \in \{..<n\}} a + \text{of-nat } i * d) = \text{of-nat } n * (a + (a + \text{of-nat}(n - 1) * d))$$

<proof>

lemma *arith-series-nat:*

$$\text{Suc } (\text{Suc } 0) * (\sum_{i \in \{..<n\}} a + i * d) = n * (a + (a + (n - 1) * d))$$

<proof>

lemma *arith-series-int:*

$$(2::\text{int}) * (\sum_{i \in \{..<n\}} a + \text{of-nat } i * d) = \text{of-nat } n * (a + (a + \text{of-nat}(n - 1) * d))$$

<proof>

lemma *sum-diff-distrib:*

fixes $P::\text{nat} \Rightarrow \text{nat}$

shows

$$\forall x. Q \ x \leq P \ x \implies (\sum_{x < n} P \ x) - (\sum_{x < n} Q \ x) = (\sum_{x < n} P \ x - Q \ x)$$

<proof>

34.11 Products indexed over intervals

syntax

$\text{-from-to-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((PROD \ - \ = \ \dots \ / \ -) \ [0,0,0,10] \ 10)$
 $\text{-from-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((PROD \ - \ = \ \dots \ < \ / \ -) \ [0,0,0,10] \ 10)$
 $\text{-upt-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((PROD \ - \ < \ / \ -) \ [0,0,10] \ 10)$
 $\text{-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((PROD \ - \ < \ = \ / \ -) \ [0,0,10] \ 10)$

syntax (xsymbols)

$\text{-from-to-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ = \ \dots \ / \ -) \ [0,0,0,10] \ 10)$
 $\text{-from-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ = \ \dots \ < \ / \ -) \ [0,0,0,10] \ 10)$
 $\text{-upt-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ < \ / \ -) \ [0,0,10] \ 10)$
 $\text{-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ < \ = \ / \ -) \ [0,0,10] \ 10)$

syntax (HTML output)

$\text{-from-to-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ = \ \dots \ / \ -) \ [0,0,0,10] \ 10)$
 $\text{-from-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ = \ \dots \ < \ / \ -) \ [0,0,0,10] \ 10)$
 $\text{-upt-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ < \ / \ -) \ [0,0,10] \ 10)$
 $\text{-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\mathcal{P}\prod \ - \ < \ = \ / \ -) \ [0,0,10] \ 10)$

syntax (latex-prod output)

$\text{-from-to-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{P}\prod \ - \ = \ -) \ [0,0,0,10] \ 10)$
 $\text{-from-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{P}\prod \ - \ = \ -) \ [0,0,0,10] \ 10)$
 $\text{-upt-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{P}\prod \ - \ < \ -) \ [0,0,10] \ 10)$
 $\text{-upto-setprod} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b$
 $((\mathcal{P}\prod \ - \ \leq \ -) \ [0,0,10] \ 10)$

translations

$\prod x=a..b. t == \text{CONST setprod } (\%x. t) \ \{a..b\}$
 $\prod x=a..<b. t == \text{CONST setprod } (\%x. t) \ \{a..<b\}$
 $\prod i \leq n. t == \text{CONST setprod } (\lambda i. t) \ \{..n\}$
 $\prod i < n. t == \text{CONST setprod } (\lambda i. t) \ \{..<n\}$

end

35 Presburger: Decision Procedure for Presburger Arithmetic

theory Presburger

imports Groebner-Basis SetInterval

uses

Tools/Qelim/qelim.ML
 Tools/Qelim/cooper-data.ML
 Tools/Qelim/generated-cooper.ML
 (Tools/Qelim/cooper.ML)

(Tools/Qelim/presburger.ML)
begin

⟨ML⟩

35.1 The $-\infty$ and $+\infty$ Properties

lemma *minf*:

$$\begin{aligned} & \llbracket \exists (z :: 'a::\text{linorder}). \forall x < z. P\ x = P'\ x; \exists z. \forall x < z. Q\ x = Q'\ x \rrbracket \\ & \implies \exists z. \forall x < z. (P\ x \wedge Q\ x) = (P'\ x \wedge Q'\ x) \\ & \llbracket \exists (z :: 'a::\text{linorder}). \forall x < z. P\ x = P'\ x; \exists z. \forall x < z. Q\ x = Q'\ x \rrbracket \\ & \implies \exists z. \forall x < z. (P\ x \vee Q\ x) = (P'\ x \vee Q'\ x) \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x < z. (x = t) = \text{False} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x < z. (x \neq t) = \text{True} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x < z. (x < t) = \text{True} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x < z. (x \leq t) = \text{True} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x < z. (x > t) = \text{False} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x < z. (x \geq t) = \text{False} \\ & \exists z. \forall (x :: 'a::\{\text{linorder}, \text{plus}, \text{Ring-and-Field.dvd}\}) < z. (d\ \text{dvd}\ x + s) = (d\ \text{dvd}\ x + s) \\ & \exists z. \forall (x :: 'a::\{\text{linorder}, \text{plus}, \text{Ring-and-Field.dvd}\}) < z. (\neg d\ \text{dvd}\ x + s) = (\neg d\ \text{dvd}\ x + s) \\ & \exists z. \forall x < z. F = F \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *pinf*:

$$\begin{aligned} & \llbracket \exists (z :: 'a::\text{linorder}). \forall x > z. P\ x = P'\ x; \exists z. \forall x > z. Q\ x = Q'\ x \rrbracket \\ & \implies \exists z. \forall x > z. (P\ x \wedge Q\ x) = (P'\ x \wedge Q'\ x) \\ & \llbracket \exists (z :: 'a::\text{linorder}). \forall x > z. P\ x = P'\ x; \exists z. \forall x > z. Q\ x = Q'\ x \rrbracket \\ & \implies \exists z. \forall x > z. (P\ x \vee Q\ x) = (P'\ x \vee Q'\ x) \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x > z. (x = t) = \text{False} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x > z. (x \neq t) = \text{True} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x > z. (x < t) = \text{False} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x > z. (x \leq t) = \text{False} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x > z. (x > t) = \text{True} \\ & \exists (z :: 'a::\{\text{linorder}\}). \forall x > z. (x \geq t) = \text{True} \\ & \exists z. \forall (x :: 'a::\{\text{linorder}, \text{plus}, \text{Ring-and-Field.dvd}\}) > z. (d\ \text{dvd}\ x + s) = (d\ \text{dvd}\ x + s) \\ & \exists z. \forall (x :: 'a::\{\text{linorder}, \text{plus}, \text{Ring-and-Field.dvd}\}) > z. (\neg d\ \text{dvd}\ x + s) = (\neg d\ \text{dvd}\ x + s) \\ & \exists z. \forall x > z. F = F \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *inf-period*:

$$\begin{aligned} & \llbracket \forall x\ k. P\ x = P\ (x - k * D); \forall x\ k. Q\ x = Q\ (x - k * D) \rrbracket \\ & \implies \forall x\ k. (P\ x \wedge Q\ x) = (P\ (x - k * D) \wedge Q\ (x - k * D)) \\ & \llbracket \forall x\ k. P\ x = P\ (x - k * D); \forall x\ k. Q\ x = Q\ (x - k * D) \rrbracket \\ & \implies \forall x\ k. (P\ x \vee Q\ x) = (P\ (x - k * D) \vee Q\ (x - k * D)) \\ & (d :: 'a::\{\text{comm-ring}, \text{Ring-and-Field.dvd}\})\ \text{dvd}\ D \implies \forall x\ k. (d\ \text{dvd}\ x + t) = (d\ \text{dvd}\ x + t) \end{aligned}$$

$dvd (x - k * D) + t$
 $(d :: 'a :: \{comm-ring, Ring-and-Field.dvd\}) \quad dvd \ D \implies \forall x \ k. (\neg d \ dvd \ x + t) = (\neg d \ dvd \ (x - k * D) + t)$
 $\forall x \ k. F = F$
 $\langle proof \rangle$

35.2 The A and B sets

lemma *bset*:

$\llbracket \forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow P \ x \longrightarrow P(x - D) ;$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow Q \ x \longrightarrow Q(x - D) \rrbracket \implies$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (P \ x \wedge Q \ x) \longrightarrow (P(x - D) \wedge Q(x - D))$
 $\llbracket \forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow P \ x \longrightarrow P(x - D) ;$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow Q \ x \longrightarrow Q(x - D) \rrbracket \implies$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (P \ x \vee Q \ x) \longrightarrow (P(x - D) \vee Q(x - D))$
 $\llbracket D > 0; t - 1 \in B \rrbracket \implies (\forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (x = t) \longrightarrow (x - D = t))$
 $\llbracket D > 0 ; t \in B \rrbracket \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (x \neq t) \longrightarrow (x - D \neq t))$
 $D > 0 \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (x < t) \longrightarrow (x - D < t))$
 $D > 0 \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (x \leq t) \longrightarrow (x - D \leq t))$
 $\llbracket D > 0 ; t \in B \rrbracket \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (x > t) \longrightarrow (x - D > t))$
 $\llbracket D > 0 ; t - 1 \in B \rrbracket \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (x \geq t) \longrightarrow (x - D \geq t))$
 $d \ dvd \ D \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (d \ dvd \ x + t) \longrightarrow (d \ dvd \ (x - D) + t))$
 $d \ dvd \ D \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow (\neg d \ dvd \ x + t) \longrightarrow (\neg d \ dvd \ (x - D) + t))$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in B. x \neq b + j) \longrightarrow F \longrightarrow F$
 $\langle proof \rangle$

lemma *aset*:

$\llbracket \forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow P \ x \longrightarrow P(x + D) ;$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow Q \ x \longrightarrow Q(x + D) \rrbracket \implies$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (P \ x \wedge Q \ x) \longrightarrow (P(x + D) \wedge Q(x + D))$
 $\llbracket \forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow P \ x \longrightarrow P(x + D) ;$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow Q \ x \longrightarrow Q(x + D) \rrbracket \implies$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (P \ x \vee Q \ x) \longrightarrow (P(x + D) \vee Q(x + D))$
 $\llbracket D > 0; t + 1 \in A \rrbracket \implies (\forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (x = t) \longrightarrow (x + D = t))$
 $\llbracket D > 0 ; t \in A \rrbracket \implies (\forall (x :: int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (x \neq t) \longrightarrow (x + D \neq t))$

$\llbracket D > 0; t \in A \rrbracket \implies (\forall (x::int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (x < t) \longrightarrow (x + D < t))$
 $\llbracket D > 0; t + 1 \in A \rrbracket \implies (\forall (x::int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (x \leq t) \longrightarrow (x + D \leq t))$
 $D > 0 \implies (\forall (x::int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (x > t) \longrightarrow (x + D > t))$
 $D > 0 \implies (\forall (x::int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (x \geq t) \longrightarrow (x + D \geq t))$
 $d \text{ dvd } D \implies (\forall (x::int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (d \text{ dvd } x+t) \longrightarrow (d \text{ dvd } (x + D) + t))$
 $d \text{ dvd } D \implies (\forall (x::int). (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow (\neg d \text{ dvd } x+t) \longrightarrow (\neg d \text{ dvd } (x + D) + t))$
 $\forall x. (\forall j \in \{1 \dots D\}. \forall b \in A. x \neq b - j) \longrightarrow F \longrightarrow F$
 $\langle \text{proof} \rangle$

35.3 Cooper’s Theorem $-\infty$ and $+\infty$ Version

35.3.1 First some trivial facts about periodic sets or predicates

lemma *periodic-finite-ex*:

assumes *dpos*: $(0::int) < d$ **and** *modd*: $ALL\ x\ k. P\ x = P(x - k*d)$

shows $(EX\ x. P\ x) = (EX\ j : \{1..d\}. P\ j)$

(**is** *?LHS* = *?RHS*)

$\langle \text{proof} \rangle$

35.3.2 The $-\infty$ Version

lemma *decr-lemma*: $0 < (d::int) \implies x - (abs(x-z)+1) * d < z$

$\langle \text{proof} \rangle$

lemma *incr-lemma*: $0 < (d::int) \implies z < x + (abs(x-z)+1) * d$

$\langle \text{proof} \rangle$

theorem *int-induct*[*case-names base step1 step2*]:

assumes

base: $P(k::int)$ **and** *step1*: $\bigwedge i. \llbracket k \leq i; P\ i \rrbracket \implies P(i+1)$ **and**

step2: $\bigwedge i. \llbracket k \geq i; P\ i \rrbracket \implies P(i-1)$

shows $P\ i$

$\langle \text{proof} \rangle$

lemma *decr-mult-lemma*:

assumes *dpos*: $(0::int) < d$ **and** *minus*: $\forall x. P\ x \longrightarrow P(x - d)$ **and** *kneg*: $0 \leq k$

shows $ALL\ x. P\ x \longrightarrow P(x - k*d)$

$\langle \text{proof} \rangle$

lemma *minusinfinity*:

assumes *dpos*: $0 < d$ **and**

P1eqP1: $ALL\ x\ k. P1\ x = P1(x - k*d)$ **and** *ePeqP1*: $EX\ z::int. ALL\ x. x < z \longrightarrow (P\ x = P1\ x)$

shows $(EX\ x. P1\ x) \longrightarrow (EX\ x. P\ x)$
 $\langle proof \rangle$

lemma *cpmi*:

assumes *dp*: $0 < D$ **and** *p1*: $\exists z. \forall x < z. P\ x = P'\ x$
and *nb*: $\forall x. (\forall j \in \{1..D\}. \forall (b::int) \in B. x \neq b+j) \longrightarrow P\ (x) \longrightarrow P\ (x - D)$
and *pd*: $\forall x\ k. P'\ x = P'\ (x - k * D)$
shows $(\exists x. P\ x) = ((\exists j \in \{1..D\}. P'\ j) \mid (\exists j \in \{1..D\}. \exists b \in B. P\ (b+j)))$
(is $?L = (?R1 \vee ?R2)$
 $\langle proof \rangle$

35.3.3 The $+\infty$ Version

lemma *plusinfinity*:

assumes *dpos*: $(0::int) < d$ **and**
P1eqP1: $\forall x\ k. P'\ x = P'\ (x - k * d)$ **and** *ePeqP1*: $\exists z. \forall x > z. P\ x = P'\ x$
shows $(\exists x. P'\ x) \longrightarrow (\exists x. P\ x)$
 $\langle proof \rangle$

lemma *incr-mult-lemma*:

assumes *dpos*: $(0::int) < d$ **and** *plus*: $ALL\ x::int. P\ x \longrightarrow P\ (x + d)$ **and** *knneg*:
 $0 \leq k$
shows $ALL\ x. P\ x \longrightarrow P\ (x + k * d)$
 $\langle proof \rangle$

lemma *cpqi*:

assumes *dp*: $0 < D$ **and** *p1*: $\exists z. \forall x > z. P\ x = P'\ x$
and *nb*: $\forall x. (\forall j \in \{1..D\}. \forall (b::int) \in A. x \neq b - j) \longrightarrow P\ (x) \longrightarrow P\ (x + D)$
and *pd*: $\forall x\ k. P'\ x = P'\ (x - k * D)$
shows $(\exists x. P\ x) = ((\exists j \in \{1..D\}. P'\ j) \mid (\exists j \in \{1..D\}. \exists b \in A. P\ (b - j)))$
(is $?L = (?R1 \vee ?R2)$
 $\langle proof \rangle$

lemma *simp-from-to*: $\{i..j::int\} = (\text{if } j < i \text{ then } \{\} \text{ else insert } i\ \{i+1..j\})$
 $\langle proof \rangle$

theorem *unity-coeff-ex*: $(\exists (x::'a::\{\text{semiring-0, Ring-and-Field.dvd}\}). P\ (l * x)) \equiv$
 $(\exists x. l\ \text{dvd}\ (x + 0) \wedge P\ x)$
 $\langle proof \rangle$

lemma *zdvd-mono*: **assumes** *not0*: $(k::int) \neq 0$

shows $((m::int)\ \text{dvd}\ t) \equiv (k * m\ \text{dvd}\ k * t)$
 $\langle proof \rangle$

lemma *uminus-dvd-conv*: $(d\ \text{dvd}\ (t::int)) \equiv (-d\ \text{dvd}\ t) \wedge (d\ \text{dvd}\ (t::int)) \equiv (d\ \text{dvd}\ -t)$
 $\langle proof \rangle$

Theorems for transforming predicates on *nat* to predicates on *int*

lemma *all-nat*: $(\forall x::nat. P\ x) = (\forall x::int. 0 \leq x \longrightarrow P\ (nat\ x))$
 $\langle proof \rangle$

lemma *ex-nat*: $(\exists x::nat. P\ x) = (\exists x::int. 0 \leq x \wedge P\ (nat\ x))$
 $\langle proof \rangle$

lemma *zdiff-int-split*: $P\ (int\ (x - y)) =$
 $((y \leq x \longrightarrow P\ (int\ x - int\ y)) \wedge (x < y \longrightarrow P\ 0))$
 $\langle proof \rangle$

lemma *number-of1*: $(0::int) \leq number-of\ n \implies (0::int) \leq number-of\ (Int.Bit0\ n) \wedge (0::int) \leq number-of\ (Int.Bit1\ n)$
 $\langle proof \rangle$

lemma *number-of2*: $(0::int) \leq Numeral0$ $\langle proof \rangle$

lemma *Suc-plus1*: $Suc\ n = n + 1$ $\langle proof \rangle$

Specific instances of congruence rules, to prevent simplifier from looping.

theorem *imp-le-cong*: $(0 \leq x \implies P = P') \implies (0 \leq (x::int) \longrightarrow P) = (0 \leq x \longrightarrow P')$ $\langle proof \rangle$

theorem *conj-le-cong*: $(0 \leq x \implies P = P') \implies (0 \leq (x::int) \wedge P) = (0 \leq x \wedge P')$
 $\langle proof \rangle$

lemma *int-eq-number-of-eq*:
 $((number-of\ v)::int) = (number-of\ w) = iszero\ ((number-of\ (v + (uminus\ w)))::int)$
 $\langle proof \rangle$

declare *dvd-eq-mod-eq-0*[*symmetric, presburger*]

declare *mod-1*[*presburger*]

declare *mod-0*[*presburger*]

declare *mod-by-1*[*presburger*]

declare *zmod-zero*[*presburger*]

declare *zmod-self*[*presburger*]

declare *mod-self*[*presburger*]

declare *mod-by-0*[*presburger*]

declare *mod-div-trivial*[*presburger*]

declare *div-mod-equality2*[*presburger*]

declare *div-mod-equality*[*presburger*]

declare *mod-div-equality2*[*presburger*]

declare *mod-div-equality*[*presburger*]

declare *mod-mult-self1*[*presburger*]

declare *mod-mult-self2*[*presburger*]

declare *zdiv-zmod-equality2*[*presburger*]

declare *zdiv-zmod-equality*[*presburger*]

declare *mod2-Suc-Suc*[*presburger*]

lemma [*presburger*]: $(a::int) \div 0 = 0$ **and** [*presburger*]: $a \bmod 0 = a$

$\langle proof \rangle$

$\langle ML \rangle$

lemma [presburger, algebra]: $m \bmod 2 = (1::nat) \longleftrightarrow \neg 2 \text{ dvd } m$ $\langle proof \rangle$

lemma [presburger, algebra]: $m \bmod 2 = \text{Suc } 0 \longleftrightarrow \neg 2 \text{ dvd } m$ $\langle proof \rangle$

lemma [presburger, algebra]: $m \bmod (\text{Suc } (\text{Suc } 0)) = (1::nat) \longleftrightarrow \neg 2 \text{ dvd } m$
 $\langle proof \rangle$

lemma [presburger, algebra]: $m \bmod (\text{Suc } (\text{Suc } 0)) = \text{Suc } 0 \longleftrightarrow \neg 2 \text{ dvd } m$
 $\langle proof \rangle$

lemma [presburger, algebra]: $m \bmod 2 = (1::int) \longleftrightarrow \neg 2 \text{ dvd } m$ $\langle proof \rangle$

lemma *zdvd-period*:

fixes $a \ d :: int$

assumes *advdd*: $a \text{ dvd } d$

shows $a \text{ dvd } (x + t) \longleftrightarrow a \text{ dvd } ((x + c * d) + t)$

$\langle proof \rangle$

end

36 Recdef: TFL: recursive function definitions

theory *Recdef*

imports *FunDef Plain*

uses

(*Tools/TFL/casesplit.ML*)

(*Tools/TFL/utis.ML*)

(*Tools/TFL/usyntax.ML*)

(*Tools/TFL/dcterm.ML*)

(*Tools/TFL/thms.ML*)

(*Tools/TFL/rules.ML*)

(*Tools/TFL/thry.ML*)

(*Tools/TFL/tfl.ML*)

(*Tools/TFL/post.ML*)

(*Tools/recdef-package.ML*)

begin

* This form avoids giant explosions in proofs. NOTE USE OF ==

lemma *def-wfrec*: $[| f == wfrec \ r \ H; \ wf(r) \ |] ==> f(a) = H \ (cut \ f \ r \ a) \ a$

$\langle proof \rangle$

lemma *tfl-wf-induct*: $ALL \ R. \ wf \ R \ -->$

$(ALL \ P. \ (ALL \ x. \ (ALL \ y. \ (y,x):R \ --> \ P \ y) \ --> \ P \ x) \ --> \ (ALL \ x. \ P \ x))$

$\langle proof \rangle$

lemma *tfl-cut-apply*: $ALL\ f\ R.\ (x,a):R \dashrightarrow (cut\ f\ R\ a)(x) = f(x)$
 $\langle proof \rangle$

lemma *tfl-wfrec*:
 $ALL\ M\ R\ f.\ (f = wfrec\ R\ M) \dashrightarrow wf\ R \dashrightarrow (ALL\ x.\ f\ x = M\ (cut\ f\ R\ x)\ x)$
 $\langle proof \rangle$

lemma *tfl-eq-True*: $(x = True) \dashrightarrow x$
 $\langle proof \rangle$

lemma *tfl-rev-eq-mp*: $(x = y) \dashrightarrow y \dashrightarrow x$
 $\langle proof \rangle$

lemma *tfl-simp-thm*: $(x \dashrightarrow y) \dashrightarrow (x = x') \dashrightarrow (x' \dashrightarrow y)$
 $\langle proof \rangle$

lemma *tfl-P-imp-P-iff-True*: $P \implies P = True$
 $\langle proof \rangle$

lemma *tfl-imp-trans*: $(A \dashrightarrow B) \implies (B \dashrightarrow C) \implies (A \dashrightarrow C)$
 $\langle proof \rangle$

lemma *tfl-disj-assoc*: $(a \vee b) \vee c == a \vee (b \vee c)$
 $\langle proof \rangle$

lemma *tfl-disjE*: $P \vee Q \implies P \dashrightarrow R \implies Q \dashrightarrow R \implies R$
 $\langle proof \rangle$

lemma *tfl-exE*: $\exists x.\ P\ x \implies \forall x.\ P\ x \dashrightarrow Q \implies Q$
 $\langle proof \rangle$

$\langle ML \rangle$

lemmas [*recdef-simp*] =
inv-image-def
measure-def
lex-prod-def
same-fst-def
less-Suc-eq [*THEN iffD2*]

lemmas [*recdef-cong*] =
if-cong *let-cong* *image-cong* *INT-cong* *UN-cong* *bex-cong* *ball-cong* *imp-cong*

lemmas [*recdef-wf*] =
wf-trancl
wf-less-than
wf-lex-prod
wf-inv-image
wf-measure

```

wf-pred-nat
wf-same-fst
wf-empty

```

```
end
```

37 Hilbert-Choice: Hilbert’s Epsilon-Operator and the Axiom of Choice

```

theory Hilbert-Choice
imports Nat Wellfounded Plain
uses (Tools/meson.ML) (Tools/specification-package.ML)
begin

```

37.1 Hilbert’s epsilon

axiomatization

```
Eps :: ('a => bool) => 'a
```

where

```
someI: P x ==> P (Eps P)
```

syntax (epsilon)

```
-Eps :: [pttrn, bool] => 'a (( $\exists \epsilon$  ./ -) [0, 10] 10)
```

syntax (HOL)

```
-Eps :: [pttrn, bool] => 'a (( $\exists @$  ./ -) [0, 10] 10)
```

syntax

```
-Eps :: [pttrn, bool] => 'a (( $\exists$  SOME ./ -) [0, 10] 10)
```

translations

```
SOME x. P == CONST Eps (%x. P)
```

$\langle ML \rangle$

constdefs

```
inv :: ('a => 'b) => ('b => 'a)
```

```
inv(f :: 'a => 'b) == %y. SOME x. f x = y
```

```
Inv :: 'a set => ('a => 'b) => ('b => 'a)
```

```
Inv A f == %x. SOME y. y  $\in$  A & f y = x
```

37.2 Hilbert’s Epsilon-operator

Easier to apply than *someI* if the witness comes from an existential formula

lemma *someI-ex* [*elim?*]: $\exists x. P x \implies P (SOME x. P x)$

$\langle proof \rangle$

Easier to apply than *someI* because the conclusion has only one occurrence of *P*.

lemma *someI2*: $[[P\ a; \ !x. P\ x \implies Q\ x]] \implies Q\ (SOME\ x. P\ x)$
 $\langle proof \rangle$

Easier to apply than *someI2* if the witness comes from an existential formula

lemma *someI2-ex*: $[[\exists a. P\ a; \ !x. P\ x \implies Q\ x]] \implies Q\ (SOME\ x. P\ x)$
 $\langle proof \rangle$

lemma *some-equality* [intro]:
 $[[P\ a; \ !x. P\ x \implies x=a]] \implies (SOME\ x. P\ x) = a$
 $\langle proof \rangle$

lemma *some1-equality*: $[[EX!x. P\ x; P\ a]] \implies (SOME\ x. P\ x) = a$
 $\langle proof \rangle$

lemma *some-eq-ex*: $P\ (SOME\ x. P\ x) = (\exists x. P\ x)$
 $\langle proof \rangle$

lemma *some-eq-trivial* [simp]: $(SOME\ y. y=x) = x$
 $\langle proof \rangle$

lemma *some-sym-eq-trivial* [simp]: $(SOME\ y. x=y) = x$
 $\langle proof \rangle$

37.3 Axiom of Choice, Proved Using the Description Operator

Used in *Tools/meson.ML*

lemma *choice*: $\forall x. \exists y. Q\ x\ y \implies \exists f. \forall x. Q\ x\ (f\ x)$
 $\langle proof \rangle$

lemma *bchoice*: $\forall x \in S. \exists y. Q\ x\ y \implies \exists f. \forall x \in S. Q\ x\ (f\ x)$
 $\langle proof \rangle$

37.4 Function Inverse

lemma *inv-id* [simp]: $inv\ id = id$
 $\langle proof \rangle$

A one-to-one function has an inverse.

lemma *inv-f-f* [simp]: $inj\ f \implies inv\ f\ (f\ x) = x$
 $\langle proof \rangle$

lemma *inv-f-eq*: $[[inj\ f; f\ x = y]] \implies inv\ f\ y = x$
 $\langle proof \rangle$

lemma *inj-imp-inv-eq*: $[[inj\ f; \forall x. f\ (g\ x) = x]] \implies inv\ f = g$
 $\langle proof \rangle$

But is it useful?

lemma *inj-transfer*:

assumes *injf*: $\text{inj } f$ and *minor*: $\forall y. y \in \text{range}(f) \implies P(\text{inv } f \ y)$

shows $P \ x$

$\langle \text{proof} \rangle$

lemma *inj-iff*: $(\text{inj } f) = (\text{inv } f \circ f = \text{id})$

$\langle \text{proof} \rangle$

lemma *inv-o-cancel[simp]*: $\text{inj } f \implies \text{inv } f \circ f = \text{id}$

$\langle \text{proof} \rangle$

lemma *o-inv-o-cancel[simp]*: $\text{inj } f \implies g \circ \text{inv } f \circ f = g$

$\langle \text{proof} \rangle$

lemma *inv-image-cancel[simp]*:

$\text{inj } f \implies \text{inv } f \circ f \circ S = S$

$\langle \text{proof} \rangle$

lemma *inj-imp-surj-inv*: $\text{inj } f \implies \text{surj } (\text{inv } f)$

$\langle \text{proof} \rangle$

lemma *f-inv-f*: $y \in \text{range}(f) \implies f(\text{inv } f \ y) = y$

$\langle \text{proof} \rangle$

lemma *surj-f-inv-f*: $\text{surj } f \implies f(\text{inv } f \ y) = y$

$\langle \text{proof} \rangle$

lemma *inv-injective*:

assumes *eq*: $\text{inv } f \ x = \text{inv } f \ y$

and *x*: $x \in \text{range } f$

and *y*: $y \in \text{range } f$

shows $x=y$

$\langle \text{proof} \rangle$

lemma *inj-on-inv*: $A \subseteq \text{range}(f) \implies \text{inj-on } (\text{inv } f) \ A$

$\langle \text{proof} \rangle$

lemma *surj-imp-inj-inv*: $\text{surj } f \implies \text{inj } (\text{inv } f)$

$\langle \text{proof} \rangle$

lemma *surj-iff*: $(\text{surj } f) = (f \circ \text{inv } f = \text{id})$

$\langle \text{proof} \rangle$

lemma *surj-imp-inv-eq*: $[\text{surj } f; \forall x. g(f \ x) = x] \implies \text{inv } f = g$

$\langle \text{proof} \rangle$

lemma *bij-imp-bij-inv*: $\text{bij } f \implies \text{bij } (\text{inv } f)$

$\langle \text{proof} \rangle$

lemma *inv-equality*: $[[\text{!!}x. g (f x) = x; \text{!!}y. f (g y) = y]] \implies \text{inv } f = g$
 $\langle \text{proof} \rangle$

lemma *inv-inv-eq*: $\text{bij } f \implies \text{inv } (\text{inv } f) = f$
 $\langle \text{proof} \rangle$

lemma *o-inv-distrib*: $[[\text{bij } f; \text{bij } g]] \implies \text{inv } (f \circ g) = \text{inv } g \circ \text{inv } f$
 $\langle \text{proof} \rangle$

lemma *image-surj-f-inv-f*: $\text{surj } f \implies f ' (\text{inv } f ' A) = A$
 $\langle \text{proof} \rangle$

lemma *image-inv-f-f*: $\text{inj } f \implies (\text{inv } f) ' (f ' A) = A$
 $\langle \text{proof} \rangle$

lemma *inv-image-comp*: $\text{inj } f \implies \text{inv } f ' (f'X) = X$
 $\langle \text{proof} \rangle$

lemma *bij-image-Collect-eq*: $\text{bij } f \implies f ' \text{Collect } P = \{y. P (\text{inv } f y)\}$
 $\langle \text{proof} \rangle$

lemma *bij-vimage-eq-inv-image*: $\text{bij } f \implies f -' A = \text{inv } f ' A$
 $\langle \text{proof} \rangle$

37.5 Inverse of a PI-function (restricted domain)

lemma *Inv-f-f*: $[[\text{inj-on } f A; x \in A]] \implies \text{Inv } A f (f x) = x$
 $\langle \text{proof} \rangle$

lemma *f-Inv-f*: $y \in f'A \implies f (\text{Inv } A f y) = y$
 $\langle \text{proof} \rangle$

lemma *Inv-injective*:
 assumes *eq*: $\text{Inv } A f x = \text{Inv } A f y$
 and *x*: $x: f'A$
 and *y*: $y: f'A$
 shows $x=y$
 $\langle \text{proof} \rangle$

lemma *inj-on-Inv*: $B \leq f'A \implies \text{inj-on } (\text{Inv } A f) B$
 $\langle \text{proof} \rangle$

lemma *Inv-mem*: $[[f ' A = B; x \in B]] \implies \text{Inv } A f x \in A$
 $\langle \text{proof} \rangle$

lemma *Inv-f-eq*: $[[\text{inj-on } f \ A; f \ x = y; x \in A \] \implies \text{Inv } A \ f \ y = x]$
 $\langle \text{proof} \rangle$

lemma *Inv-comp*:
 $[[\text{inj-on } f \ (g \ ' \ A); \text{inj-on } g \ A; x \in f \ ' \ g \ ' \ A \] \implies$
 $\text{Inv } A \ (f \ o \ g) \ x = (\text{Inv } A \ g \ o \ \text{Inv } (g \ ' \ A) \ f) \ x]$
 $\langle \text{proof} \rangle$

lemma *bij-betw-Inv*: $\text{bij-betw } f \ A \ B \implies \text{bij-betw } (\text{Inv } A \ f) \ B \ A$
 $\langle \text{proof} \rangle$

37.6 Other Consequences of Hilbert’s Epsilon

Hilbert’s Epsilon and the *split* Operator

Looping simprule

lemma *split-paired-Eps*: $(\text{SOME } x. P \ x) = (\text{SOME } (a,b). P(a,b))$
 $\langle \text{proof} \rangle$

lemma *Eps-split*: $\text{Eps } (\text{split } P) = (\text{SOME } xy. P \ (\text{fst } xy) \ (\text{snd } xy))$
 $\langle \text{proof} \rangle$

lemma *Eps-split-eq [simp]*: $(@ (x',y'). x = x' \ \& \ y = y') = (x,y)$
 $\langle \text{proof} \rangle$

A relation is wellfounded iff it has no infinite descending chain

lemma *wf-iff-no-infinite-down-chain*:
 $\text{wf } r = (\sim (\exists f. \forall i. (f(\text{Suc } i), f \ i) \in r))$
 $\langle \text{proof} \rangle$

lemma *wf-no-infinite-down-chainE*:
assumes $\text{wf } r$ **obtains** k **where** $(f \ (\text{Suc } k), f \ k) \notin r$
 $\langle \text{proof} \rangle$

A dynamically-scoped fact for TFL

lemma *tfl-some*: $\forall P \ x. P \ x \dashrightarrow P \ (\text{Eps } P)$
 $\langle \text{proof} \rangle$

37.7 Least value operator

constdefs

$\text{LeastM} :: ['a \Rightarrow 'b::\text{ord}, 'a \Rightarrow \text{bool}] \Rightarrow 'a$
 $\text{LeastM } m \ P == \text{SOME } x. P \ x \ \& \ (\forall y. P \ y \dashrightarrow m \ x \leq m \ y)$

syntax

$\text{-LeastM} :: [\text{pttrn}, 'a \Rightarrow 'b::\text{ord}, \text{bool}] \Rightarrow 'a \quad (\text{LEAST } - \text{ WRT } -. - [0, 4, 10]$
 $10)$

translations

$\text{LEAST } x \text{ WRT } m. P == \text{LeastM } m \ (\%x. P)$

lemma *LeastMI2*:

$$\begin{aligned} P\ x \implies (!y. P\ y \implies m\ x \leq m\ y) \\ \implies (!x. P\ x \implies \forall y. P\ y \dashrightarrow m\ x \leq m\ y \implies Q\ x) \\ \implies Q\ (LeastM\ m\ P) \\ \langle proof \rangle \end{aligned}$$

lemma *LeastM-equality*:

$$\begin{aligned} P\ k \implies (!x. P\ x \implies m\ k \leq m\ x) \\ \implies m\ (LEAST\ x\ WRT\ m. P\ x) = (m\ k :: 'a::order) \\ \langle proof \rangle \end{aligned}$$

lemma *wf-linord-ex-has-least*:

$$\begin{aligned} wf\ r \implies \forall x\ y. ((x, y) : r^+ \wedge) = ((y, x) : r^+ \wedge) \implies P\ k \\ \implies \exists x. P\ x \ \& \ (!y. P\ y \dashrightarrow (m\ x, m\ y) : r^+ \wedge) \\ \langle proof \rangle \end{aligned}$$

lemma *ex-has-least-nat*:

$$\begin{aligned} P\ k \implies \exists x. P\ x \ \& \ (\forall y. P\ y \dashrightarrow m\ x \leq (m\ y :: nat)) \\ \langle proof \rangle \end{aligned}$$

lemma *LeastM-nat-lemma*:

$$\begin{aligned} P\ k \implies P\ (LeastM\ m\ P) \ \& \ (\forall y. P\ y \dashrightarrow m\ (LeastM\ m\ P) \leq (m\ y :: nat)) \\ \langle proof \rangle \end{aligned}$$

lemmas *LeastM-natI* = *LeastM-nat-lemma* [*THEN* *conjunct1*, *standard*]

lemma *LeastM-nat-le*: $P\ x \implies m\ (LeastM\ m\ P) \leq (m\ x :: nat)$

$\langle proof \rangle$

37.8 Greatest value operator

constdefs

$$\begin{aligned} GreatestM :: ['a \Rightarrow 'b::ord, 'a \Rightarrow bool] \Rightarrow 'a \\ GreatestM\ m\ P == SOME\ x. P\ x \ \& \ (\forall y. P\ y \dashrightarrow m\ y \leq m\ x) \end{aligned}$$

$$\begin{aligned} Greatest :: ('a::ord \Rightarrow bool) \Rightarrow 'a \quad (\mathbf{binder}\ GREATEST\ 10) \\ Greatest == GreatestM\ (\%x. x) \end{aligned}$$

syntax

$$\begin{aligned} -GreatestM :: [pttrn, 'a \Rightarrow 'b::ord, bool] \Rightarrow 'a \\ (GREATEST\ -\ WRT\ -. \ -\ [0, 4, 10]\ 10) \end{aligned}$$

translations

$$GREATEST\ x\ WRT\ m. P == GreatestM\ m\ (\%x. P)$$

lemma *GreatestMI2*:

$$\begin{aligned} P\ x \implies (!y. P\ y \implies m\ y \leq m\ x) \\ \implies (!x. P\ x \implies \forall y. P\ y \dashrightarrow m\ y \leq m\ x \implies Q\ x) \end{aligned}$$

$\implies Q \text{ (GreatestM } m \text{ } P)$
 $\langle \text{proof} \rangle$

lemma *GreatestM-equality:*

$P \text{ } k \implies (!x. P \text{ } x \implies m \text{ } x \leq m \text{ } k)$
 $\implies m \text{ (GREATEST } x \text{ WRT } m. P \text{ } x) = (m \text{ } k :: 'a :: \text{order})$
 $\langle \text{proof} \rangle$

lemma *Greatest-equality:*

$P \text{ (} k :: 'a :: \text{order}) \implies (!x. P \text{ } x \implies x \leq k) \implies (\text{GREATEST } x. P \text{ } x) = k$
 $\langle \text{proof} \rangle$

lemma *ex-has-greatest-nat-lemma:*

$P \text{ } k \implies \forall x. P \text{ } x \longrightarrow (\exists y. P \text{ } y \ \& \ \sim ((m \text{ } y :: \text{nat}) \leq m \text{ } x))$
 $\implies \exists y. P \text{ } y \ \& \ \sim (m \text{ } y < m \text{ } k + n)$
 $\langle \text{proof} \rangle$

lemma *ex-has-greatest-nat:*

$P \text{ } k \implies \forall y. P \text{ } y \longrightarrow m \text{ } y < b$
 $\implies \exists x. P \text{ } x \ \& \ (\forall y. P \text{ } y \longrightarrow (m \text{ } y :: \text{nat}) \leq m \text{ } x)$
 $\langle \text{proof} \rangle$

lemma *GreatestM-nat-lemma:*

$P \text{ } k \implies \forall y. P \text{ } y \longrightarrow m \text{ } y < b$
 $\implies P \text{ (GreatestM } m \text{ } P) \ \& \ (\forall y. P \text{ } y \longrightarrow (m \text{ } y :: \text{nat}) \leq m \text{ (GreatestM } m \text{ } P))$
 $\langle \text{proof} \rangle$

lemmas *GreatestM-natI* = *GreatestM-nat-lemma* [THEN conjunct1, standard]

lemma *GreatestM-nat-le:*

$P \text{ } x \implies \forall y. P \text{ } y \longrightarrow m \text{ } y < b$
 $\implies (m \text{ } x :: \text{nat}) \leq m \text{ (GreatestM } m \text{ } P)$
 $\langle \text{proof} \rangle$

Specialization to *GREATEST*.

lemma *GreatestI*: $P \text{ (} k :: \text{nat}) \implies \forall y. P \text{ } y \longrightarrow y < b \implies P \text{ (GREATEST } x. P \text{ } x)$
 $\langle \text{proof} \rangle$

lemma *Greatest-le:*

$P \text{ } x \implies \forall y. P \text{ } y \longrightarrow y < b \implies (x :: \text{nat}) \leq (\text{GREATEST } x. P \text{ } x)$
 $\langle \text{proof} \rangle$

37.9 The Meson proof procedure

37.9.1 Negation Normal Form

de Morgan laws

lemma *meson-not-conjD*: $\sim(P \& Q) ==> \sim P \mid \sim Q$
and *meson-not-disjD*: $\sim(P \mid Q) ==> \sim P \& \sim Q$
and *meson-not-notD*: $\sim\sim P ==> P$
and *meson-not-allD*: $!!P. \sim(\forall x. P(x)) ==> \exists x. \sim P(x)$
and *meson-not-exD*: $!!P. \sim(\exists x. P(x)) ==> \forall x. \sim P(x)$
<proof>

Removal of $-->$ and $<->$ (positive and negative occurrences)

lemma *meson-imp-to-disjD*: $P-->Q ==> \sim P \mid Q$
and *meson-not-impD*: $\sim(P-->Q) ==> P \& \sim Q$
and *meson-iff-to-disjD*: $P=Q ==> (\sim P \mid Q) \& (\sim Q \mid P)$
and *meson-not-iffD*: $\sim(P=Q) ==> (P \mid Q) \& (\sim P \mid \sim Q)$
 — Much more efficient than $P \wedge \neg Q \vee Q \wedge \neg P$ for computing CNF
and *meson-not-refl-disj-D*: $x \sim = x \mid P ==> P$
<proof>

37.9.2 Pulling out the existential quantifiers

Conjunction

lemma *meson-conj-exD1*: $!!P Q. (\exists x. P(x)) \& Q ==> \exists x. P(x) \& Q$
and *meson-conj-exD2*: $!!P Q. P \& (\exists x. Q(x)) ==> \exists x. P \& Q(x)$
<proof>

Disjunction

lemma *meson-disj-exD*: $!!P Q. (\exists x. P(x)) \mid (\exists x. Q(x)) ==> \exists x. P(x) \mid Q(x)$
 — DO NOT USE with forall-Skolemization: makes fewer schematic variables!!
 — With ex-Skolemization, makes fewer Skolem constants
and *meson-disj-exD1*: $!!P Q. (\exists x. P(x)) \mid Q ==> \exists x. P(x) \mid Q$
and *meson-disj-exD2*: $!!P Q. P \mid (\exists x. Q(x)) ==> \exists x. P \mid Q(x)$
<proof>

37.9.3 Generating clauses for the Meson Proof Procedure

Disjunctions

lemma *meson-disj-assoc*: $(P \mid Q) \mid R ==> P \mid (Q \mid R)$
and *meson-disj-comm*: $P \mid Q ==> Q \mid P$
and *meson-disj-FalseD1*: $False \mid P ==> P$
and *meson-disj-FalseD2*: $P \mid False ==> P$
<proof>

37.10 Lemmas for Meson, the Model Elimination Procedure

Generation of contrapositives

Inserts negated disjunct after removing the negation; P is a literal. Model elimination requires assuming the negation of every attempted subgoal, hence the negated disjuncts.

lemma *make-neg-rule*: $\sim P|Q \implies ((\sim P \implies P) \implies Q)$
 $\langle proof \rangle$

Version for Plaisted’s ”Postive refinement” of the Meson procedure

lemma *make-refined-neg-rule*: $\sim P|Q \implies (P \implies Q)$
 $\langle proof \rangle$

P should be a literal

lemma *make-pos-rule*: $P|Q \implies ((P \implies \sim P) \implies Q)$
 $\langle proof \rangle$

Versions of *make-neg-rule* and *make-pos-rule* that don’t insert new assumptions, for ordinary resolution.

lemmas *make-neg-rule'* = *make-refined-neg-rule*

lemma *make-pos-rule'*: $[|P|Q; \sim P|] \implies Q$
 $\langle proof \rangle$

Generation of a goal clause – put away the final literal

lemma *make-neg-goal*: $\sim P \implies ((\sim P \implies P) \implies False)$
 $\langle proof \rangle$

lemma *make-pos-goal*: $P \implies ((P \implies \sim P) \implies False)$
 $\langle proof \rangle$

37.10.1 Lemmas for Forward Proof

There is a similarity to congruence rules

lemma *conj-forward*: $[|P' \& Q'; P' \implies P; Q' \implies Q|] \implies P \& Q$
 $\langle proof \rangle$

lemma *disj-forward*: $[|P'|Q'; P' \implies P; Q' \implies Q|] \implies P|Q$
 $\langle proof \rangle$

lemma *disj-forward2*:
 $[|P'|Q'; P' \implies P; [|Q'; P \implies False|] \implies Q|] \implies P|Q$
 $\langle proof \rangle$

lemma *all-forward*: $[|\forall x. P'(x); !x. P'(x) \implies P(x)|] \implies \forall x. P(x)$
 $\langle proof \rangle$

lemma *ex-forward*: $[|\exists x. P'(x); !x. P'(x) \implies P(x)|] \implies \exists x. P(x)$
 $\langle proof \rangle$

Many of these bindings are used by the ATP linkup, and not just by legacy proof scripts.

$\langle ML \rangle$

37.11 Meson package $\langle ML \rangle$ **37.12 Specification package – Hilbertized version**

lemma *exE-some*: $[[\text{Ex } P ; c == \text{Eps } P]] ==> P \ c$
 $\langle proof \rangle$

 $\langle ML \rangle$ **end****38 ATP-Linkup: The Isabelle-ATP Linkup****theory** *ATP-Linkup***imports** *Divides Record Hilbert-Choice Plain***uses**

Tools/polyhash.ML
Tools/res-clause.ML
(Tools/res-axioms.ML)
(Tools/res-hol-clause.ML)
(Tools/res-reconstruct.ML)
(Tools/res-atp.ML)
(Tools/atp-manager.ML)
(Tools/atp-wrapper.ML)
 $\sim\sim$ */src/Tools/Metis/metis.ML*
(Tools/metis-tools.ML)

begin

definition *COMBI* :: $'a ==> 'a$
where *COMBI* $P == P$

definition *COMBK* :: $'a ==> 'b ==> 'a$
where *COMBK* $P \ Q == P$

definition *COMBB* :: $('b ==> 'c) ==> ('a ==> 'b) ==> 'a ==> 'c$
where *COMBB* $P \ Q \ R == P \ (Q \ R)$

definition *COMBC* :: $('a ==> 'b ==> 'c) ==> 'b ==> 'a ==> 'c$
where *COMBC* $P \ Q \ R == P \ R \ Q$

definition *COMBS* :: $('a ==> 'b ==> 'c) ==> ('a ==> 'b) ==> 'a ==> 'c$
where *COMBS* $P \ Q \ R == P \ R \ (Q \ R)$

definition *fequal* :: $'a ==> 'a ==> bool$
where *fequal* $X \ Y == (X=Y)$

lemma *fequal-imp-equal*: $fequal\ X\ Y ==> X=Y$
 $\langle proof \rangle$

lemma *equal-imp-fequal*: $X=Y ==> fequal\ X\ Y$
 $\langle proof \rangle$

These two represent the equivalence between Boolean equality and iff. They can't be converted to clauses automatically, as the iff would be expanded...

lemma *iff-positive*: $P \mid Q \mid P=Q$
 $\langle proof \rangle$

lemma *iff-negative*: $\sim P \mid \sim Q \mid P=Q$
 $\langle proof \rangle$

Theorems for translation to combinators

lemma *abs-S*: $(\%x. (f\ x)\ (g\ x)) == COMBS\ f\ g$
 $\langle proof \rangle$

lemma *abs-I*: $(\%x. x) == COMBI$
 $\langle proof \rangle$

lemma *abs-K*: $(\%x. y) == COMBK\ y$
 $\langle proof \rangle$

lemma *abs-B*: $(\%x. a\ (g\ x)) == COMBB\ a\ g$
 $\langle proof \rangle$

lemma *abs-C*: $(\%x. (f\ x)\ b) == COMBC\ f\ b$
 $\langle proof \rangle$

38.1 Setup of external ATPs

$\langle ML \rangle$

basic provers

$\langle ML \rangle$

provers with structured output

$\langle ML \rangle$

on some problems better results

$\langle ML \rangle$

remote provers via SystemOnTPTP

$\langle ML \rangle$

38.2 The Metis prover

$\langle ML \rangle$

end

39 List: The datatype of finite lists

```
theory List
imports Plain Relation-Power Presburger Recdef ATP-Linkup
uses Tools/string-syntax.ML
begin
```

```
datatype 'a list =
  Nil    ([])
| Cons 'a 'a list  (infixr # 65)
```

39.1 Basic list processing functions

```
consts
  filter:: ('a => bool) => 'a list => 'a list
  concat:: 'a list list => 'a list
  foldl :: ('b => 'a => 'b) => 'b => 'a list => 'b
  foldr :: ('a => 'b => 'b) => 'a list => 'b => 'b
  hd:: 'a list => 'a
  tl:: 'a list => 'a list
  last:: 'a list => 'a
  butlast :: 'a list => 'a list
  set :: 'a list => 'a set
  map :: ('a=>'b) => ('a list => 'b list)
  listsum :: 'a list => 'a::monoid-add
  list-update :: 'a list => nat => 'a => 'a list
  take:: nat => 'a list => 'a list
  drop:: nat => 'a list => 'a list
  takeWhile :: ('a => bool) => 'a list => 'a list
  dropWhile :: ('a => bool) => 'a list => 'a list
  rev :: 'a list => 'a list
  zip :: 'a list => 'b list => ('a * 'b) list
  upt :: nat => nat => nat list ((1[-.</-]))
  remdups :: 'a list => 'a list
  remove1 :: 'a => 'a list => 'a list
  removeAll :: 'a => 'a list => 'a list
  distinct:: 'a list => bool
  replicate :: nat => 'a => 'a list
  splice :: 'a list => 'a list => 'a list
```

```
nonterminals lupdbinds lupdbind
```

```
syntax
```

```
— list Enumeration
@list :: args => 'a list  ([[(-)])
```

— Special syntax for filter
 $\text{@filter} :: [\text{pttrn}, 'a \text{ list}, \text{bool}] \Rightarrow 'a \text{ list} \quad ((1[-<--./-]))$

— list update
 $\text{-lupdbind} :: ['a, 'a] \Rightarrow \text{lupdbind} \quad ((2[-:=/-]))$
 $:: \text{lupdbind} \Rightarrow \text{lupdbinds} \quad (-)$
 $\text{-lupdbinds} :: [\text{lupdbind}, \text{lupdbinds}] \Rightarrow \text{lupdbinds} \quad (-./-)$
 $\text{-LUpdate} :: ['a, \text{lupdbinds}] \Rightarrow 'a \quad (-/[(-)] [900,0] 900)$

translations

$[x, xs] == x\#[xs]$
 $[x] == x\#[\]$
 $[x < -xs \ . \ P] == \text{filter } (\%x. P) \ xs$

$\text{-LUpdate } xs \ (\text{-lupdbinds } b \ bs) == \text{-LUpdate } (\text{-LUpdate } xs \ b) \ bs$
 $xs[i:=x] == \text{list-update } xs \ i \ x$

syntax (*xsymbols*)

$\text{@filter} :: [\text{pttrn}, 'a \text{ list}, \text{bool}] \Rightarrow 'a \text{ list}((1[-<--./-]))$

syntax (*HTML output*)

$\text{@filter} :: [\text{pttrn}, 'a \text{ list}, \text{bool}] \Rightarrow 'a \text{ list}((1[-<--./-]))$

Function *size* is overloaded for all datatypes. Users may refer to the list version as *length*.

abbreviation

$\text{length} :: 'a \text{ list} \Rightarrow \text{nat} \text{ where}$
 $\text{length} == \text{size}$

primrec

$\text{hd}(x\#xs) = x$

primrec

$\text{tl}([\]) = []$
 $\text{tl}(x\#xs) = xs$

primrec

$\text{last}(x\#xs) = (\text{if } xs=[] \text{ then } x \text{ else last } xs)$

primrec

$\text{butlast } [] = []$
 $\text{butlast}(x\#xs) = (\text{if } xs=[] \text{ then } [] \text{ else } x\#\text{butlast } xs)$

primrec

$\text{set } [] = \{\}$
 $\text{set } (x\#xs) = \text{insert } x \ (\text{set } xs)$

primrec

$map\ f\ [] = []$
 $map\ f\ (x\#\!xs) = f(x)\#\!map\ f\ xs$

primrec

$append :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list\ (\textbf{infixr}\ @\ 65)$

where

$append\ Nil: []\ @\ ys = ys$
 $| append\ Cons: (x\#\!xs)\ @\ ys = x\ \#\ xs\ @\ ys$

primrec

$rev([]) = []$
 $rev(x\#\!xs) = rev(xs)\ @\ [x]$

primrec

$filter\ P\ [] = []$
 $filter\ P\ (x\#\!xs) = (if\ P\ x\ then\ x\#\!filter\ P\ xs\ else\ filter\ P\ xs)$

primrec

$foldl\ Nil: foldl\ f\ a\ [] = a$
 $foldl\ Cons: foldl\ f\ a\ (x\#\!xs) = foldl\ f\ (f\ a\ x)\ xs$

primrec

$foldr\ f\ []\ a = a$
 $foldr\ f\ (x\#\!xs)\ a = f\ x\ (foldr\ f\ xs\ a)$

primrec

$concat([]) = []$
 $concat(x\#\!xs) = x\ @\ concat(xs)$

primrec

$listsum\ [] = 0$
 $listsum\ (x\ \#\ xs) = x + listsum\ xs$

primrec

$drop\ Nil: drop\ n\ [] = []$
 $drop\ Cons: drop\ n\ (x\#\!xs) = (case\ n\ of\ 0 \Rightarrow x\#\!xs\ |\ Suc(m) \Rightarrow drop\ m\ xs)$
 — Warning: simpset does not contain this definition, but separate theorems for
 $n = 0$ and $n = Suc\ k$

primrec

$take\ Nil: take\ n\ [] = []$
 $take\ Cons: take\ n\ (x\#\!xs) = (case\ n\ of\ 0 \Rightarrow []\ |\ Suc(m) \Rightarrow x\ \#\ take\ m\ xs)$
 — Warning: simpset does not contain this definition, but separate theorems for
 $n = 0$ and $n = Suc\ k$

primrec $nth :: 'a\ list \Rightarrow nat \Rightarrow 'a\ (\textbf{infixl}\ !\ 100)$ **where**

$nth\ Cons: (x\#\!xs)!\ n = (case\ n\ of\ 0 \Rightarrow x\ |\ (Suc\ k) \Rightarrow xs!\ k)$
 — Warning: simpset does not contain this definition, but separate theorems for
 $n = 0$ and $n = Suc\ k$

primrec

$$\begin{aligned} [][i:=v] &= [] \\ (x\#xs)[i:=v] &= (\text{case } i \text{ of } 0 \Rightarrow v \# xs \mid \text{Suc } j \Rightarrow x \# xs[j:=v]) \end{aligned}$$
primrec

$$\begin{aligned} \text{takeWhile } P \ [] &= [] \\ \text{takeWhile } P \ (x\#xs) &= (\text{if } P \ x \text{ then } x\#\text{takeWhile } P \ xs \text{ else } []) \end{aligned}$$
primrec

$$\begin{aligned} \text{dropWhile } P \ [] &= [] \\ \text{dropWhile } P \ (x\#xs) &= (\text{if } P \ x \text{ then } \text{dropWhile } P \ xs \text{ else } x\#xs) \end{aligned}$$
primrec

$$\begin{aligned} \text{zip } xs \ [] &= [] \\ \text{zip-Cons: } \text{zip } xs \ (y\#ys) &= (\text{case } xs \text{ of } [] \Rightarrow [] \mid z\#zs \Rightarrow (z,y)\#\text{zip } zs \ ys) \\ \text{--- Warning: simpset does not contain this definition, but separate theorems for} \\ xs = [] \text{ and } xs = z \# zs \end{aligned}$$
primrec

$$\begin{aligned} \text{upt-0: } [i..<0] &= [] \\ \text{upt-Suc: } [i..<(\text{Suc } j)] &= (\text{if } i \leq j \text{ then } [i..<j] @ [j] \text{ else } []) \end{aligned}$$
primrec

$$\begin{aligned} \text{distinct } [] &= \text{True} \\ \text{distinct } (x\#xs) &= (x \sim: \text{set } xs \wedge \text{distinct } xs) \end{aligned}$$
primrec

$$\begin{aligned} \text{remdups } [] &= [] \\ \text{remdups } (x\#xs) &= (\text{if } x : \text{set } xs \text{ then } \text{remdups } xs \text{ else } x \# \text{remdups } xs) \end{aligned}$$
primrec

$$\begin{aligned} \text{remove1 } x \ [] &= [] \\ \text{remove1 } x \ (y\#xs) &= (\text{if } x=y \text{ then } xs \text{ else } y \# \text{remove1 } x \ xs) \end{aligned}$$
primrec

$$\begin{aligned} \text{removeAll } x \ [] &= [] \\ \text{removeAll } x \ (y\#xs) &= (\text{if } x=y \text{ then } \text{removeAll } x \ xs \text{ else } y \# \text{removeAll } x \ xs) \end{aligned}$$
primrec

$$\begin{aligned} \text{replicate-0: } \text{replicate } 0 \ x &= [] \\ \text{replicate-Suc: } \text{replicate } (\text{Suc } n) \ x &= x \# \text{replicate } n \ x \end{aligned}$$
definition

$$\begin{aligned} \text{rotate1} &:: 'a \text{ list} \Rightarrow 'a \text{ list} \text{ where} \\ \text{rotate1 } xs &= (\text{case } xs \text{ of } [] \Rightarrow [] \mid x\#xs \Rightarrow xs @ [x]) \end{aligned}$$
definition

$$\text{rotate} :: \text{nat} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \text{ where}$$

rotate *n* = *rotate1* ^{*n*}

definition

list-all2 :: ('a => 'b => bool) => 'a list => 'b list => bool **where**
~~[code del]:~~ *list-all2* *P xs ys* =
 (length *xs* = length *ys* ∧ (∀ (*x*, *y*) ∈ set (zip *xs ys*). *P x y*))

definition

sublist :: 'a list => nat set => 'a list **where**
sublist xs A = map fst (filter (λ*p*. snd *p* ∈ *A*) (zip *xs* [0..*size xs*]))

primrec

splice [] *ys* = *ys*
splice (*x#xs*) *ys* = (if *ys*=[] then *x#xs* else *x* # *hd ys* # *splice xs* (*tl ys*))
 — Warning: simpset does not contain the second eqn but a derived one.

Figure 1 shows characteristic examples that should give an intuitive understanding of the above functions.

The following simple sort functions are intended for proofs, not for efficient implementations.

context *linorder*
begin

fun *sorted* :: 'a list ⇒ bool **where**
sorted [] ⇔ True |
sorted [*x*] ⇔ True |
sorted (*x#y#zs*) ⇔ *x* ≤ *y* ∧ *sorted* (*y#zs*)

primrec *insort* :: 'a ⇒ 'a list ⇒ 'a list **where**
insort *x* [] = [*x*] |
insort *x* (*y#ys*) = (if *x* ≤ *y* then (*x#y#ys*) else *y#(insort x ys)*)

primrec *sort* :: 'a list ⇒ 'a list **where**
sort [] = [] |
sort (*x#xs*) = *insort x* (*sort xs*)

end

39.1.1 List comprehension

Input syntax for Haskell-like list comprehension notation. Typical example: [(*x*,*y*). *x* ← *xs*, *y* ← *ys*, *x* ≠ *y*], the list of all pairs of distinct elements from *xs* and *ys*. The syntax is as in Haskell, except that | becomes a dot (like in Isabelle’s set comprehension): [*e*. *x* ← *xs*, ...] rather than [*e* | *x* ← *xs*, ...].

The qualifiers after the dot are

```

[a, b] @ [c, d] = [a, b, c, d]
length [a, b, c] = 3
set [a, b, c] = {a, b, c}
map f [a, b, c] = [f a, f b, f c]
rev [a, b, c] = [c, b, a]
hd [a, b, c, d] = a
tl [a, b, c, d] = [b, c, d]
last [a, b, c, d] = d
butlast [a, b, c, d] = [a, b, c]
filter (λn::nat. n < 2) [0, 2, 1] = [0, 1]
concat [[a, b], [c, d, e], [], [f]] = [a, b, c, d, e, f]
foldl f x [a, b, c] = f (f (f x a) b) c
foldr f [a, b, c] x = f a (f b (f c x))
zip [a, b, c] [x, y, z] = [(a, x), (b, y), (c, z)]
zip [a, b] [x, y, z] = [(a, x), (b, y)]
splice [a, b, c] [x, y, z] = [a, x, b, y, c, z]
splice [a, b, c, d] [x, y] = [a, x, b, y, c, d]
take 2 [a, b, c, d] = [a, b]
take 6 [a, b, c, d] = [a, b, c, d]
drop 2 [a, b, c, d] = [c, d]
drop 6 [a, b, c, d] = []
takeWhile (λn. n < 3) [1, 2, 3, 0] = [1, 2]
dropWhile (λn. n < 3) [1, 2, 3, 0] = [3, 0]
distinct [2, 0, 1]
remdups [2, 0, 2, 1, 2] = [0, 1, 2]
remove1 2 [2, 0, 2, 1, 2] = [0, 2, 1, 2]
removeAll 2 [2, 0, 2, 1, 2] = [0, 1]
[a, b, c, d] ! 2 = c
[a, b, c, d][2 := x] = [a, b, x, d]
sublist [a, b, c, d, e] {0, 2, 3} = [a, c, d]
rotate1 [a, b, c, d] = [b, c, d, a]
rotate 3 [a, b, c, d] = [d, a, b, c]
replicate 4 a = [a, a, a, a]
[2..<5] = [2, 3, 4]
listsum [1, 2, 3] = 6

```

Figure 1: Characteristic examples

generators $p \leftarrow xs$, where p is a pattern and xs an expression of list type,
or

guards b , where b is a boolean expression.

Just like in Haskell, list comprehension is just a shorthand. To avoid misunderstandings, the translation into desugared form is not reversed upon output. Note that the translation of $[e. x \leftarrow xs]$ is optimized to $map (\lambda x. e) xs$.

It is easy to write short list comprehensions which stand for complex expressions. During proofs, they may become unreadable (and mangled). In such cases it can be advisable to introduce separate definitions for the list comprehensions in question.

nonterminals $lc\text{-}qual$ $lc\text{-}quals$

syntax

$-listcompr :: 'a \Rightarrow lc\text{-}qual \Rightarrow lc\text{-}quals \Rightarrow 'a\ list \ ([- . --)$

$-lc\text{-}gen :: 'a \Rightarrow 'a\ list \Rightarrow lc\text{-}qual \ (- <- -)$

$-lc\text{-}test :: bool \Rightarrow lc\text{-}qual \ (-)$

$-lc\text{-}end :: lc\text{-}quals \ (())$

$-lc\text{-}quals :: lc\text{-}qual \Rightarrow lc\text{-}quals \Rightarrow lc\text{-}quals \ (, --)$

$-lc\text{-}abs :: 'a \Rightarrow 'b\ list \Rightarrow 'b\ list$

syntax ($xsymbols$)

$-lc\text{-}gen :: 'a \Rightarrow 'a\ list \Rightarrow lc\text{-}qual \ (- \leftarrow -)$

syntax (*HTML output*)

$-lc\text{-}gen :: 'a \Rightarrow 'a\ list \Rightarrow lc\text{-}qual \ (- \leftarrow -)$

$\langle ML \rangle$

39.1.2 $[]$ and $op \#$

lemma *not-Cons-self* [*simp*]:

$xs \neq x \# xs$

$\langle proof \rangle$

lemmas *not-Cons-self2* [*simp*] = *not-Cons-self* [*symmetric*]

lemma *neq-Nil-conv*: $(xs \neq []) = (\exists y\ ys. xs = y \# ys)$

$\langle proof \rangle$

lemma *length-induct*:

$(\bigwedge xs. \forall ys. length\ ys < length\ xs \longrightarrow P\ ys \Longrightarrow P\ xs) \Longrightarrow P\ xs$

$\langle proof \rangle$

39.1.3 *length*

Needs to come before @ because of theorem *append-eq-append-conv*.

lemma *length-append* [simp]: $\text{length } (xs @ ys) = \text{length } xs + \text{length } ys$
 ⟨proof⟩

lemma *length-map* [simp]: $\text{length } (\text{map } f \text{ } xs) = \text{length } xs$
 ⟨proof⟩

lemma *length-rev* [simp]: $\text{length } (\text{rev } xs) = \text{length } xs$
 ⟨proof⟩

lemma *length-tl* [simp]: $\text{length } (\text{tl } xs) = \text{length } xs - 1$
 ⟨proof⟩

lemma *length-0-conv* [iff]: $(\text{length } xs = 0) = (xs = [])$
 ⟨proof⟩

lemma *length-greater-0-conv* [iff]: $(0 < \text{length } xs) = (xs \neq [])$
 ⟨proof⟩

lemma *length-pos-if-in-set*: $x : \text{set } xs \implies \text{length } xs > 0$
 ⟨proof⟩

lemma *length-Suc-conv*:
 $(\text{length } xs = \text{Suc } n) = (\exists y \text{ } ys. xs = y \# ys \wedge \text{length } ys = n)$
 ⟨proof⟩

lemma *Suc-length-conv*:
 $(\text{Suc } n = \text{length } xs) = (\exists y \text{ } ys. xs = y \# ys \wedge \text{length } ys = n)$
 ⟨proof⟩

lemma *impossible-Cons*: $\text{length } xs <= \text{length } ys \implies xs = x \# ys = \text{False}$
 ⟨proof⟩

lemma *list-induct2* [consumes 1, case-names Nil Cons]:
 $\text{length } xs = \text{length } ys \implies P [] [] \implies$
 $(\bigwedge x \text{ } xs \text{ } y \text{ } ys. \text{length } xs = \text{length } ys \implies P \text{ } xs \text{ } ys \implies P (x \# xs) (y \# ys))$
 $\implies P \text{ } xs \text{ } ys$
 ⟨proof⟩

lemma *list-induct3* [consumes 2, case-names Nil Cons]:
 $\text{length } xs = \text{length } ys \implies \text{length } ys = \text{length } zs \implies P [] [] [] \implies$
 $(\bigwedge x \text{ } xs \text{ } y \text{ } ys \text{ } z \text{ } zs. \text{length } xs = \text{length } ys \implies \text{length } ys = \text{length } zs \implies P \text{ } xs \text{ } ys \text{ } zs$
 $\implies P (x \# xs) (y \# ys) (z \# zs))$
 $\implies P \text{ } xs \text{ } ys \text{ } zs$
 ⟨proof⟩

lemma *list-induct2'*:

$$\begin{aligned} & \llbracket P \rrbracket \llbracket \rrbracket; \\ & \bigwedge x \, xs. P \, (x \# xs) \llbracket \rrbracket; \\ & \bigwedge y \, ys. P \llbracket \rrbracket (y \# ys); \\ & \bigwedge x \, xs \, y \, ys. P \, xs \, ys \implies P \, (x \# xs) \, (y \# ys) \rrbracket \\ & \implies P \, xs \, ys \\ & \langle proof \rangle
\end{aligned}$$

lemma *neq-if-length-neq*: $length \, xs \neq length \, ys \implies (xs = ys) == False$
 $\langle proof \rangle$

$\langle ML \rangle$

39.1.4 @ – append

lemma *append-assoc* [simp]: $(xs @ ys) @ zs = xs @ (ys @ zs)$
 $\langle proof \rangle$

lemma *append-Nil2* [simp]: $xs @ [] = xs$
 $\langle proof \rangle$

lemma *append-is-Nil-conv* [iff]: $(xs @ ys = []) = (xs = [] \wedge ys = [])$
 $\langle proof \rangle$

lemma *Nil-is-append-conv* [iff]: $([] = xs @ ys) = (xs = [] \wedge ys = [])$
 $\langle proof \rangle$

lemma *append-self-conv* [iff]: $(xs @ ys = xs) = (ys = [])$
 $\langle proof \rangle$

lemma *self-append-conv* [iff]: $(xs = xs @ ys) = (ys = [])$
 $\langle proof \rangle$

lemma *append-eq-append-conv* [simp, noatp]:
 $length \, xs = length \, ys \vee length \, us = length \, vs$
 $\implies (xs @ us = ys @ vs) = (xs = ys \wedge us = vs)$
 $\langle proof \rangle$

lemma *append-eq-append-conv2*: $(xs @ ys = zs @ ts) =$
 $(EX \, us. xs = zs @ us \ \& \ us @ ys = ts \mid xs @ us = zs \ \& \ ys = us @ ts)$
 $\langle proof \rangle$

lemma *same-append-eq* [iff]: $(xs @ ys = xs @ zs) = (ys = zs)$
 $\langle proof \rangle$

lemma *append1-eq-conv* [iff]: $(xs @ [x] = ys @ [y]) = (xs = ys \wedge x = y)$
 $\langle proof \rangle$

lemma *append-same-eq* [iff]: $(ys @ xs = zs @ xs) = (ys = zs)$
 $\langle proof \rangle$

lemma *append-self-conv2* [iff]: $(xs @ ys = ys) = (xs = [])$
 ⟨proof⟩

lemma *self-append-conv2* [iff]: $(ys = xs @ ys) = (xs = [])$
 ⟨proof⟩

lemma *hd-Cons-tl* [simp,noatp]: $xs \neq [] \implies hd\ xs \# tl\ xs = xs$
 ⟨proof⟩

lemma *hd-append*: $hd\ (xs @ ys) = (if\ xs = []\ then\ hd\ ys\ else\ hd\ xs)$
 ⟨proof⟩

lemma *hd-append2* [simp]: $xs \neq [] \implies hd\ (xs @ ys) = hd\ xs$
 ⟨proof⟩

lemma *tl-append*: $tl\ (xs @ ys) = (case\ xs\ of\ [] \Rightarrow tl\ ys \mid z \# zs \Rightarrow zs @ ys)$
 ⟨proof⟩

lemma *tl-append2* [simp]: $xs \neq [] \implies tl\ (xs @ ys) = tl\ xs @ ys$
 ⟨proof⟩

lemma *Cons-eq-append-conv*: $x \# xs = ys @ zs =$
 $(ys = [] \ \& \ x \# xs = zs \mid (EX\ ys'.\ x \# ys' = ys \ \& \ xs = ys' @ zs))$
 ⟨proof⟩

lemma *append-eq-Cons-conv*: $(ys @ zs = x \# xs) =$
 $(ys = [] \ \& \ zs = x \# xs \mid (EX\ ys'.\ ys = x \# ys' \ \& \ ys' @ zs = xs))$
 ⟨proof⟩

Trivial rules for solving @-equations automatically.

lemma *eq-Nil-appendI*: $xs = ys \implies xs = [] @ ys$
 ⟨proof⟩

lemma *Cons-eq-appendI*:
 $[| x \# xs1 = ys; xs = xs1 @ zs |] \implies x \# xs = ys @ zs$
 ⟨proof⟩

lemma *append-eq-appendI*:
 $[| xs @ xs1 = zs; ys = xs1 @ us |] \implies xs @ ys = zs @ us$
 ⟨proof⟩

Simplification procedure for all list equalities. Currently only tries to rearrange @ to see if - both lists end in a singleton list, - or both lists end in the same list.

⟨ML⟩

39.1.5 *map*

lemma *map-ext*: $(!!x. x : \text{set } xs \longrightarrow f\ x = g\ x) \implies \text{map } f\ xs = \text{map } g\ xs$
 $\langle \text{proof} \rangle$

lemma *map-ident* [*simp*]: $\text{map } (\lambda x. x) = (\lambda xs. xs)$
 $\langle \text{proof} \rangle$

lemma *map-append* [*simp*]: $\text{map } f\ (xs @ ys) = \text{map } f\ xs @ \text{map } f\ ys$
 $\langle \text{proof} \rangle$

lemma *map-compose*: $\text{map } (f \circ g)\ xs = \text{map } f\ (\text{map } g\ xs)$
 $\langle \text{proof} \rangle$

lemma *rev-map*: $\text{rev } (\text{map } f\ xs) = \text{map } f\ (\text{rev } xs)$
 $\langle \text{proof} \rangle$

lemma *map-eq-conv* [*simp*]: $(\text{map } f\ xs = \text{map } g\ xs) = (!x : \text{set } xs. f\ x = g\ x)$
 $\langle \text{proof} \rangle$

lemma *map-cong* [*fundef-cong*, *recdef-cong*]:
 $xs = ys \implies (!x. x : \text{set } ys \implies f\ x = g\ x) \implies \text{map } f\ xs = \text{map } g\ ys$
— a congruence rule for *map*
 $\langle \text{proof} \rangle$

lemma *map-is-Nil-conv* [*iff*]: $(\text{map } f\ xs = []) = (xs = [])$
 $\langle \text{proof} \rangle$

lemma *Nil-is-map-conv* [*iff*]: $([] = \text{map } f\ xs) = (xs = [])$
 $\langle \text{proof} \rangle$

lemma *map-eq-Cons-conv*:
 $(\text{map } f\ xs = y \# ys) = (\exists z\ zs. xs = z \# zs \wedge f\ z = y \wedge \text{map } f\ zs = ys)$
 $\langle \text{proof} \rangle$

lemma *Cons-eq-map-conv*:
 $(x \# xs = \text{map } f\ ys) = (\exists z\ zs. ys = z \# zs \wedge x = f\ z \wedge xs = \text{map } f\ zs)$
 $\langle \text{proof} \rangle$

lemmas *map-eq-Cons-D* = *map-eq-Cons-conv* [*THEN* *iffD1*]

lemmas *Cons-eq-map-D* = *Cons-eq-map-conv* [*THEN* *iffD1*]

declare *map-eq-Cons-D* [*dest!*] *Cons-eq-map-D* [*dest!*]

lemma *ex-map-conv*:
 $(EX\ xs. ys = \text{map } f\ xs) = (ALL\ y : \text{set } ys. EX\ x. y = f\ x)$
 $\langle \text{proof} \rangle$

lemma *map-eq-imp-length-eq*:
assumes $\text{map } f\ xs = \text{map } f\ ys$
shows $\text{length } xs = \text{length } ys$

$\langle proof \rangle$

lemma *map-inj-on*:

$[[\text{map } f \text{ } xs = \text{map } f \text{ } ys; \text{inj-on } f \text{ } (\text{set } xs \text{ } Un \text{ } \text{set } ys)]]$
 $\implies xs = ys$

$\langle proof \rangle$

lemma *inj-on-map-eq-map*:

$\text{inj-on } f \text{ } (\text{set } xs \text{ } Un \text{ } \text{set } ys) \implies (\text{map } f \text{ } xs = \text{map } f \text{ } ys) = (xs = ys)$

$\langle proof \rangle$

lemma *map-injective*:

$\text{map } f \text{ } xs = \text{map } f \text{ } ys \implies \text{inj } f \implies xs = ys$

$\langle proof \rangle$

lemma *inj-map-eq-map[simp]*: $\text{inj } f \implies (\text{map } f \text{ } xs = \text{map } f \text{ } ys) = (xs = ys)$

$\langle proof \rangle$

lemma *inj-mapI*: $\text{inj } f \implies \text{inj } (\text{map } f)$

$\langle proof \rangle$

lemma *inj-mapD*: $\text{inj } (\text{map } f) \implies \text{inj } f$

$\langle proof \rangle$

lemma *inj-map[iff]*: $\text{inj } (\text{map } f) = \text{inj } f$

$\langle proof \rangle$

lemma *inj-on-mapI*: $\text{inj-on } f \text{ } (\bigcup (\text{set } 'A)) \implies \text{inj-on } (\text{map } f) \text{ } A$

$\langle proof \rangle$

lemma *map-idI*: $(\bigwedge x. x \in \text{set } xs \implies f \text{ } x = x) \implies \text{map } f \text{ } xs = xs$

$\langle proof \rangle$

lemma *map-fun-upd [simp]*: $y \notin \text{set } xs \implies \text{map } (f(y:=v)) \text{ } xs = \text{map } f \text{ } xs$

$\langle proof \rangle$

lemma *map-fst-zip[simp]*:

$\text{length } xs = \text{length } ys \implies \text{map } \text{fst } (\text{zip } xs \text{ } ys) = xs$

$\langle proof \rangle$

lemma *map-snd-zip[simp]*:

$\text{length } xs = \text{length } ys \implies \text{map } \text{snd } (\text{zip } xs \text{ } ys) = ys$

$\langle proof \rangle$

39.1.6 rev

lemma *rev-append [simp]*: $\text{rev } (xs @ ys) = \text{rev } ys @ \text{rev } xs$

$\langle proof \rangle$

lemma *rev-rev-ident* [simp]: $\text{rev} (\text{rev } xs) = xs$
 ⟨proof⟩

lemma *rev-swap*: $(\text{rev } xs = ys) = (xs = \text{rev } ys)$
 ⟨proof⟩

lemma *rev-is-Nil-conv* [iff]: $(\text{rev } xs = []) = (xs = [])$
 ⟨proof⟩

lemma *Nil-is-rev-conv* [iff]: $([] = \text{rev } xs) = (xs = [])$
 ⟨proof⟩

lemma *rev-singleton-conv* [simp]: $(\text{rev } xs = [x]) = (xs = [x])$
 ⟨proof⟩

lemma *singleton-rev-conv* [simp]: $([x] = \text{rev } xs) = (xs = [x])$
 ⟨proof⟩

lemma *rev-is-rev-conv* [iff]: $(\text{rev } xs = \text{rev } ys) = (xs = ys)$
 ⟨proof⟩

lemma *inj-on-rev*[iff]: *inj-on* *rev* *A*
 ⟨proof⟩

lemma *rev-induct* [case-names *Nil snoc*]:
 $[\![P \]\!]; !!x \text{ } xs. P \text{ } xs \implies P \text{ } (xs @ [x]) \implies P \text{ } xs$
 ⟨proof⟩

lemma *rev-exhaust* [case-names *Nil snoc*]:
 $(xs = [] \implies P) \implies (!!ys \text{ } y. xs = ys @ [y] \implies P) \implies P$
 ⟨proof⟩

lemmas *rev-cases* = *rev-exhaust*

lemma *rev-eq-Cons-iff*[iff]: $(\text{rev } xs = y \# ys) = (xs = \text{rev } ys @ [y])$
 ⟨proof⟩

39.1.7 set

lemma *finite-set* [iff]: *finite* (*set* *xs*)
 ⟨proof⟩

lemma *set-append* [simp]: $\text{set } (xs @ ys) = (\text{set } xs \cup \text{set } ys)$
 ⟨proof⟩

lemma *hd-in-set*[simp]: $xs \neq [] \implies \text{hd } xs : \text{set } xs$
 ⟨proof⟩

lemma *set-subset-Cons*: $\text{set } xs \subseteq \text{set } (x \# xs)$

$\langle proof \rangle$

lemma *set-ConsD*: $y \in \text{set } (x \# xs) \implies y=x \vee y \in \text{set } xs$
 $\langle proof \rangle$

lemma *set-empty [iff]*: $(\text{set } xs = \{\}) = (xs = [])$
 $\langle proof \rangle$

lemma *set-empty2[iff]*: $(\{\} = \text{set } xs) = (xs = [])$
 $\langle proof \rangle$

lemma *set-rev [simp]*: $\text{set } (\text{rev } xs) = \text{set } xs$
 $\langle proof \rangle$

lemma *set-map [simp]*: $\text{set } (\text{map } f \ xs) = f^*(\text{set } xs)$
 $\langle proof \rangle$

lemma *set-filter [simp]*: $\text{set } (\text{filter } P \ xs) = \{x. x : \text{set } xs \wedge P \ x\}$
 $\langle proof \rangle$

lemma *set-upt [simp]*: $\text{set}[i..<j] = \{k. i \leq k \wedge k < j\}$
 $\langle proof \rangle$

lemma *split-list*: $x : \text{set } xs \implies \exists \ ys \ zs. xs = ys @ x \# zs$
 $\langle proof \rangle$

lemma *in-set-conv-decomp*: $x \in \text{set } xs \longleftrightarrow (\exists \ ys \ zs. xs = ys @ x \# zs)$
 $\langle proof \rangle$

lemma *split-list-first*: $x : \text{set } xs \implies \exists \ ys \ zs. xs = ys @ x \# zs \wedge x \notin \text{set } ys$
 $\langle proof \rangle$

lemma *in-set-conv-decomp-first*:
 $(x : \text{set } xs) = (\exists \ ys \ zs. xs = ys @ x \# zs \wedge x \notin \text{set } ys)$
 $\langle proof \rangle$

lemma *split-list-last*: $x : \text{set } xs \implies \exists \ ys \ zs. xs = ys @ x \# zs \wedge x \notin \text{set } zs$
 $\langle proof \rangle$

lemma *in-set-conv-decomp-last*:
 $(x : \text{set } xs) = (\exists \ ys \ zs. xs = ys @ x \# zs \wedge x \notin \text{set } zs)$
 $\langle proof \rangle$

lemma *split-list-prop*: $\exists x \in \text{set } xs. P \ x \implies \exists \ ys \ x \ zs. xs = ys @ x \# zs \ \& \ P \ x$
 $\langle proof \rangle$

lemma *split-list-propE*:
assumes $\exists x \in \text{set } xs. P \ x$

obtains $ys\ x\ zs$ **where** $xs = ys @ x \# zs$ **and** $P\ x$
 $\langle proof \rangle$

lemma *split-list-first-prop*:
 $\exists x \in set\ xs. P\ x \implies$
 $\exists ys\ x\ zs. xs = ys @ x \# zs \wedge P\ x \wedge (\forall y \in set\ ys. \neg P\ y)$
 $\langle proof \rangle$

lemma *split-list-first-propE*:
assumes $\exists x \in set\ xs. P\ x$
obtains $ys\ x\ zs$ **where** $xs = ys @ x \# zs$ **and** $P\ x$ **and** $\forall y \in set\ ys. \neg P\ y$
 $\langle proof \rangle$

lemma *split-list-first-prop-iff*:
 $(\exists x \in set\ xs. P\ x) \longleftrightarrow$
 $(\exists ys\ x\ zs. xs = ys @ x \# zs \wedge P\ x \wedge (\forall y \in set\ ys. \neg P\ y))$
 $\langle proof \rangle$

lemma *split-list-last-prop*:
 $\exists x \in set\ xs. P\ x \implies$
 $\exists ys\ x\ zs. xs = ys @ x \# zs \wedge P\ x \wedge (\forall z \in set\ zs. \neg P\ z)$
 $\langle proof \rangle$

lemma *split-list-last-propE*:
assumes $\exists x \in set\ xs. P\ x$
obtains $ys\ x\ zs$ **where** $xs = ys @ x \# zs$ **and** $P\ x$ **and** $\forall z \in set\ zs. \neg P\ z$
 $\langle proof \rangle$

lemma *split-list-last-prop-iff*:
 $(\exists x \in set\ xs. P\ x) \longleftrightarrow$
 $(\exists ys\ x\ zs. xs = ys @ x \# zs \wedge P\ x \wedge (\forall z \in set\ zs. \neg P\ z))$
 $\langle proof \rangle$

lemma *finite-list*: $finite\ A \implies EX\ xs. set\ xs = A$
 $\langle proof \rangle$

lemma *card-length*: $card\ (set\ xs) \leq length\ xs$
 $\langle proof \rangle$

lemma *set-minus-filter-out*:
 $set\ xs - \{y\} = set\ (filter\ (\lambda x. \neg (x = y))\ xs)$
 $\langle proof \rangle$

39.1.8 filter

lemma *filter-append* [simp]: $filter\ P\ (xs @ ys) = filter\ P\ xs @ filter\ P\ ys$
 $\langle proof \rangle$

lemma *rev-filter*: $rev\ (filter\ P\ xs) = filter\ P\ (rev\ xs)$

$\langle \text{proof} \rangle$

lemma *filter-filter* [simp]: $\text{filter } P (\text{filter } Q \text{ } xs) = \text{filter } (\lambda x. Q \text{ } x \wedge P \text{ } x) \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *length-filter-le* [simp]: $\text{length } (\text{filter } P \text{ } xs) \leq \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *sum-length-filter-compl*:
 $\text{length}(\text{filter } P \text{ } xs) + \text{length}(\text{filter } (\lambda x. \sim P \text{ } x) \text{ } xs) = \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *filter-True* [simp]: $\forall x \in \text{set } xs. P \text{ } x \implies \text{filter } P \text{ } xs = xs$
 $\langle \text{proof} \rangle$

lemma *filter-False* [simp]: $\forall x \in \text{set } xs. \neg P \text{ } x \implies \text{filter } P \text{ } xs = []$
 $\langle \text{proof} \rangle$

lemma *filter-empty-conv*: $(\text{filter } P \text{ } xs = []) = (\forall x \in \text{set } xs. \neg P \text{ } x)$
 $\langle \text{proof} \rangle$

lemma *filter-id-conv*: $(\text{filter } P \text{ } xs = xs) = (\forall x \in \text{set } xs. P \text{ } x)$
 $\langle \text{proof} \rangle$

lemma *filter-map*:
 $\text{filter } P (\text{map } f \text{ } xs) = \text{map } f (\text{filter } (P \circ f) \text{ } xs)$
 $\langle \text{proof} \rangle$

lemma *length-filter-map*[simp]:
 $\text{length } (\text{filter } P (\text{map } f \text{ } xs)) = \text{length}(\text{filter } (P \circ f) \text{ } xs)$
 $\langle \text{proof} \rangle$

lemma *filter-is-subset* [simp]: $\text{set } (\text{filter } P \text{ } xs) \leq \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *length-filter-less*:
 $\llbracket x : \text{set } xs; \sim P \text{ } x \rrbracket \implies \text{length}(\text{filter } P \text{ } xs) < \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *length-filter-conv-card*:
 $\text{length}(\text{filter } p \text{ } xs) = \text{card}\{i. i < \text{length } xs \ \& \ p(xs!i)\}$
 $\langle \text{proof} \rangle$

lemma *Cons-eq-filterD*:
 $x \# xs = \text{filter } P \text{ } ys \implies$
 $\exists us \text{ } vs. ys = us @ x \# vs \wedge (\forall u \in \text{set } us. \neg P \text{ } u) \wedge P \text{ } x \wedge xs = \text{filter } P \text{ } vs$
 $(\text{is } - \implies \exists us \text{ } vs. ?P \text{ } ys \text{ } us \text{ } vs)$
 $\langle \text{proof} \rangle$

lemma *filter-eq-ConsD*:

filter P ys = x # xs \implies
 $\exists us\ vs.\ ys = us @ x \# vs \wedge (\forall u \in \text{set } us.\ \neg P\ u) \wedge P\ x \wedge xs = \text{filter } P\ vs$
<proof>

lemma *filter-eq-Cons-iff*:

(filter P ys = x # xs) =
 $(\exists us\ vs.\ ys = us @ x \# vs \wedge (\forall u \in \text{set } us.\ \neg P\ u) \wedge P\ x \wedge xs = \text{filter } P\ vs)$
<proof>

lemma *Cons-eq-filter-iff*:

(x # xs = filter P ys) =
 $(\exists us\ vs.\ ys = us @ x \# vs \wedge (\forall u \in \text{set } us.\ \neg P\ u) \wedge P\ x \wedge xs = \text{filter } P\ vs)$
<proof>

lemma *filter-cong*[*fundef-cong*, *recdef-cong*]:

$xs = ys \implies (\bigwedge x.\ x \in \text{set } ys \implies P\ x = Q\ x) \implies \text{filter } P\ xs = \text{filter } Q\ ys$
<proof>

39.1.9 List partitioning

primrec *partition* :: $('a \Rightarrow \text{bool}) \Rightarrow 'a\ \text{list} \Rightarrow 'a\ \text{list} \times 'a\ \text{list}$ **where**

partition P [] = ([], [])
 $| \text{partition } P\ (x \# xs) =$
 $(\text{let } (yes, no) = \text{partition } P\ xs$
 $\text{in if } P\ x \text{ then } (x \# yes, no) \text{ else } (yes, x \# no))$

lemma *partition-filter1*:

fst (partition P xs) = filter P xs
<proof>

lemma *partition-filter2*:

snd (partition P xs) = filter (Not o P) xs
<proof>

lemma *partition-P*:

assumes *partition P xs = (yes, no)*
shows $(\forall p \in \text{set } yes.\ P\ p) \wedge (\forall p \in \text{set } no.\ \neg P\ p)$
<proof>

lemma *partition-set*:

assumes *partition P xs = (yes, no)*
shows $\text{set } yes \cup \text{set } no = \text{set } xs$
<proof>

39.1.10 concat

lemma *concat-append* [*simp*]: $\text{concat } (xs @ ys) = \text{concat } xs @ \text{concat } ys$

<proof>

lemma *concat-eq-Nil-conv* [simp]: $(\text{concat } xss = []) = (\forall xs \in \text{set } xss. xs = [])$
 ⟨proof⟩

lemma *Nil-eq-concat-conv* [simp]: $([] = \text{concat } xss) = (\forall xs \in \text{set } xss. xs = [])$
 ⟨proof⟩

lemma *set-concat* [simp]: $\text{set } (\text{concat } xs) = (\bigcup x:\text{set } xs. \text{set } x)$
 ⟨proof⟩

lemma *concat-map-singleton*[simp]: $\text{concat}(\text{map } (\%x. [f\ x])\ xs) = \text{map } f\ xs$
 ⟨proof⟩

lemma *map-concat*: $\text{map } f\ (\text{concat } xs) = \text{concat } (\text{map } (\text{map } f)\ xs)$
 ⟨proof⟩

lemma *filter-concat*: $\text{filter } p\ (\text{concat } xs) = \text{concat } (\text{map } (\text{filter } p)\ xs)$
 ⟨proof⟩

lemma *rev-concat*: $\text{rev } (\text{concat } xs) = \text{concat } (\text{map } \text{rev } (\text{rev } xs))$
 ⟨proof⟩

39.1.11 *nth*

lemma *nth-Cons-0* [simp, code]: $(x \# xs)!0 = x$
 ⟨proof⟩

lemma *nth-Cons-Suc* [simp, code]: $(x \# xs)!(\text{Suc } n) = xs!n$
 ⟨proof⟩

declare *nth.simps* [simp del]

lemma *nth-append*:
 $(xs @ ys)!n = (\text{if } n < \text{length } xs \text{ then } xs!n \text{ else } ys!(n - \text{length } xs))$
 ⟨proof⟩

lemma *nth-append-length* [simp]: $(xs @ x \# ys) ! \text{length } xs = x$
 ⟨proof⟩

lemma *nth-append-length-plus*[simp]: $(xs @ ys) ! (\text{length } xs + n) = ys ! n$
 ⟨proof⟩

lemma *nth-map* [simp]: $n < \text{length } xs \implies (\text{map } f\ xs)!n = f(xs!n)$
 ⟨proof⟩

lemma *hd-conv-nth*: $xs \neq [] \implies \text{hd } xs = xs!0$
 ⟨proof⟩

lemma *list-eq-iff-nth-eq*:

$(xs = ys) = (length\ xs = length\ ys \wedge (ALL\ i < length\ xs.\ xs!i = ys!i))$
 $\langle proof \rangle$

lemma *set-conv-nth*: $set\ xs = \{xs!i \mid i.\ i < length\ xs\}$
 $\langle proof \rangle$

lemma *in-set-conv-nth*: $(x \in set\ xs) = (\exists i < length\ xs.\ xs!i = x)$
 $\langle proof \rangle$

lemma *list-ball-nth*: $[| n < length\ xs; !x : set\ xs.\ P\ x|] ==> P(xs!n)$
 $\langle proof \rangle$

lemma *nth-mem* [simp]: $n < length\ xs ==> xs!n : set\ xs$
 $\langle proof \rangle$

lemma *all-nth-imp-all-set*:
 $[| !i < length\ xs.\ P(xs!i); x : set\ xs|] ==> P\ x$
 $\langle proof \rangle$

lemma *all-set-conv-all-nth*:
 $(\forall x \in set\ xs.\ P\ x) = (\forall i.\ i < length\ xs \longrightarrow P\ (xs\ !\ i))$
 $\langle proof \rangle$

lemma *rev-nth*:
 $n < size\ xs \implies rev\ xs\ !\ n = xs\ !\ (length\ xs - Suc\ n)$
 $\langle proof \rangle$

39.1.12 *list-update*

lemma *length-list-update* [simp]: $length(xs[i:=x]) = length\ xs$
 $\langle proof \rangle$

lemma *nth-list-update*:
 $i < length\ xs \implies (xs[i:=x])!j = (if\ i = j\ then\ x\ else\ xs!j)$
 $\langle proof \rangle$

lemma *nth-list-update-eq* [simp]: $i < length\ xs \implies (xs[i:=x])!i = x$
 $\langle proof \rangle$

lemma *nth-list-update-neq* [simp]: $i \neq j \implies xs[i:=x]!j = xs!j$
 $\langle proof \rangle$

lemma *list-update-id* [simp]: $xs[i := xs!i] = xs$
 $\langle proof \rangle$

lemma *list-update-beyond* [simp]: $length\ xs \leq i \implies xs[i:=x] = xs$
 $\langle proof \rangle$

lemma *list-update-same-conv*:

$i < \text{length } xs \implies (xs[i := x] = xs) = (xs!i = x)$
 $\langle \text{proof} \rangle$

lemma *list-update-append1*:
 $i < \text{size } xs \implies (xs @ ys)[i := x] = xs[i := x] @ ys$
 $\langle \text{proof} \rangle$

lemma *list-update-append*:
 $(xs @ ys)[n := x] =$
 $(\text{if } n < \text{length } xs \text{ then } xs[n := x] @ ys \text{ else } xs @ (ys[n - \text{length } xs := x]))$
 $\langle \text{proof} \rangle$

lemma *list-update-length [simp]*:
 $(xs @ x \# ys)[\text{length } xs := y] = (xs @ y \# ys)$
 $\langle \text{proof} \rangle$

lemma *update-zip*:
 $\text{length } xs = \text{length } ys \implies$
 $(\text{zip } xs \text{ } ys)[i := xy] = \text{zip } (xs[i := \text{fst } xy]) (ys[i := \text{snd } xy])$
 $\langle \text{proof} \rangle$

lemma *set-update-subset-insert*: $\text{set}(xs[i := x]) \leq \text{insert } x (\text{set } xs)$
 $\langle \text{proof} \rangle$

lemma *set-update-subsetI*: $[\text{set } xs \leq A; x:A] \implies \text{set}(xs[i := x]) \leq A$
 $\langle \text{proof} \rangle$

lemma *set-update-memI*: $n < \text{length } xs \implies x \in \text{set } (xs[n := x])$
 $\langle \text{proof} \rangle$

lemma *list-update-overwrite*:
 $xs[i := x, i := y] = xs[i := y]$
 $\langle \text{proof} \rangle$

lemma *list-update-swap*:
 $i \neq i' \implies xs[i := x, i' := x'] = xs[i' := x', i := x]$
 $\langle \text{proof} \rangle$

lemma *list-update-code [code]*:
 $[][i := y] = []$
 $(x \# xs)[0 := y] = y \# xs$
 $(x \# xs)[\text{Suc } i := y] = x \# xs[i := y]$
 $\langle \text{proof} \rangle$

39.1.13 last and butlast

lemma *last-snoc [simp]*: $\text{last } (xs @ [x]) = x$
 $\langle \text{proof} \rangle$

lemma *butlast-snoc* [simp]: $\text{butlast } (xs @ [x]) = xs$
 $\langle \text{proof} \rangle$

lemma *last-ConsL*: $xs = [] \implies \text{last}(x \# xs) = x$
 $\langle \text{proof} \rangle$

lemma *last-ConsR*: $xs \neq [] \implies \text{last}(x \# xs) = \text{last } xs$
 $\langle \text{proof} \rangle$

lemma *last-append*: $\text{last}(xs @ ys) = (\text{if } ys = [] \text{ then } \text{last } xs \text{ else } \text{last } ys)$
 $\langle \text{proof} \rangle$

lemma *last-appendL*[simp]: $ys = [] \implies \text{last}(xs @ ys) = \text{last } xs$
 $\langle \text{proof} \rangle$

lemma *last-appendR*[simp]: $ys \neq [] \implies \text{last}(xs @ ys) = \text{last } ys$
 $\langle \text{proof} \rangle$

lemma *hd-rev*: $xs \neq [] \implies \text{hd}(\text{rev } xs) = \text{last } xs$
 $\langle \text{proof} \rangle$

lemma *last-rev*: $xs \neq [] \implies \text{last}(\text{rev } xs) = \text{hd } xs$
 $\langle \text{proof} \rangle$

lemma *last-in-set*[simp]: $as \neq [] \implies \text{last } as \in \text{set } as$
 $\langle \text{proof} \rangle$

lemma *length-butlast* [simp]: $\text{length } (\text{butlast } xs) = \text{length } xs - 1$
 $\langle \text{proof} \rangle$

lemma *butlast-append*:
 $\text{butlast } (xs @ ys) = (\text{if } ys = [] \text{ then } \text{butlast } xs \text{ else } xs @ \text{butlast } ys)$
 $\langle \text{proof} \rangle$

lemma *append-butlast-last-id* [simp]:
 $xs \neq [] \implies \text{butlast } xs @ [\text{last } xs] = xs$
 $\langle \text{proof} \rangle$

lemma *in-set-butlastD*: $x : \text{set } (\text{butlast } xs) \implies x : \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *in-set-butlast-appendI*:
 $x : \text{set } (\text{butlast } xs) \mid x : \text{set } (\text{butlast } ys) \implies x : \text{set } (\text{butlast } (xs @ ys))$
 $\langle \text{proof} \rangle$

lemma *last-drop*[simp]: $n < \text{length } xs \implies \text{last } (\text{drop } n \text{ } xs) = \text{last } xs$
 $\langle \text{proof} \rangle$

lemma *last-conv-nth*: $xs \neq [] \implies \text{last } xs = xs!(\text{length } xs - 1)$

$\langle \text{proof} \rangle$

lemma *butlast-conv-take*: $\text{butlast } xs = \text{take } (\text{length } xs - 1) \ xs$
 $\langle \text{proof} \rangle$

39.1.14 *take and drop*

lemma *take-0* [simp]: $\text{take } 0 \ xs = []$
 $\langle \text{proof} \rangle$

lemma *drop-0* [simp]: $\text{drop } 0 \ xs = xs$
 $\langle \text{proof} \rangle$

lemma *take-Suc-Cons* [simp]: $\text{take } (\text{Suc } n) \ (x \# xs) = x \# \text{take } n \ xs$
 $\langle \text{proof} \rangle$

lemma *drop-Suc-Cons* [simp]: $\text{drop } (\text{Suc } n) \ (x \# xs) = \text{drop } n \ xs$
 $\langle \text{proof} \rangle$

declare *take-Cons* [simp del] **and** *drop-Cons* [simp del]

lemma *take-1-Cons* [simp]: $\text{take } 1 \ (x \# xs) = [x]$
 $\langle \text{proof} \rangle$

lemma *drop-1-Cons* [simp]: $\text{drop } 1 \ (x \# xs) = xs$
 $\langle \text{proof} \rangle$

lemma *take-Suc*: $xs \sim = [] \implies \text{take } (\text{Suc } n) \ xs = \text{hd } xs \# \text{take } n \ (\text{tl } xs)$
 $\langle \text{proof} \rangle$

lemma *drop-Suc*: $\text{drop } (\text{Suc } n) \ xs = \text{drop } n \ (\text{tl } xs)$
 $\langle \text{proof} \rangle$

lemma *take-tl*: $\text{take } n \ (\text{tl } xs) = \text{tl } (\text{take } (\text{Suc } n) \ xs)$
 $\langle \text{proof} \rangle$

lemma *drop-tl*: $\text{drop } n \ (\text{tl } xs) = \text{tl } (\text{drop } n \ xs)$
 $\langle \text{proof} \rangle$

lemma *tl-take*: $\text{tl } (\text{take } n \ xs) = \text{take } (n - 1) \ (\text{tl } xs)$
 $\langle \text{proof} \rangle$

lemma *tl-drop*: $\text{tl } (\text{drop } n \ xs) = \text{drop } n \ (\text{tl } xs)$
 $\langle \text{proof} \rangle$

lemma *nth-via-drop*: $\text{drop } n \ xs = y \# ys \implies xs!n = y$
 $\langle \text{proof} \rangle$

lemma *take-Suc-conv-app-nth*:

$i < \text{length } xs \implies \text{take } (\text{Suc } i) \text{ } xs = \text{take } i \text{ } xs @ [xs!i]$
 $\langle \text{proof} \rangle$

lemma *drop-Suc-conv-tl*:

$i < \text{length } xs \implies (xs!i) \# (\text{drop } (\text{Suc } i) \text{ } xs) = \text{drop } i \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *length-take [simp]*: $\text{length } (\text{take } n \text{ } xs) = \min (\text{length } xs) \ n$
 $\langle \text{proof} \rangle$

lemma *length-drop [simp]*: $\text{length } (\text{drop } n \text{ } xs) = (\text{length } xs - n)$
 $\langle \text{proof} \rangle$

lemma *take-all [simp]*: $\text{length } xs \leq n \implies \text{take } n \text{ } xs = xs$
 $\langle \text{proof} \rangle$

lemma *drop-all [simp]*: $\text{length } xs \leq n \implies \text{drop } n \text{ } xs = []$
 $\langle \text{proof} \rangle$

lemma *take-append [simp]*:

$\text{take } n \text{ } (xs @ ys) = (\text{take } n \text{ } xs @ \text{take } (n - \text{length } xs) \text{ } ys)$
 $\langle \text{proof} \rangle$

lemma *drop-append [simp]*:

$\text{drop } n \text{ } (xs @ ys) = \text{drop } n \text{ } xs @ \text{drop } (n - \text{length } xs) \text{ } ys$
 $\langle \text{proof} \rangle$

lemma *take-take [simp]*: $\text{take } n \text{ } (\text{take } m \text{ } xs) = \text{take } (\min n \ m) \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *drop-drop [simp]*: $\text{drop } n \text{ } (\text{drop } m \text{ } xs) = \text{drop } (n + m) \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *take-drop*: $\text{take } n \text{ } (\text{drop } m \text{ } xs) = \text{drop } m \text{ } (\text{take } (n + m) \text{ } xs)$
 $\langle \text{proof} \rangle$

lemma *drop-take*: $\text{drop } n \text{ } (\text{take } m \text{ } xs) = \text{take } (m - n) \text{ } (\text{drop } n \text{ } xs)$
 $\langle \text{proof} \rangle$

lemma *append-take-drop-id [simp]*: $\text{take } n \text{ } xs @ \text{drop } n \text{ } xs = xs$
 $\langle \text{proof} \rangle$

lemma *take-eq-Nil[simp]*: $(\text{take } n \text{ } xs = []) = (n = 0 \vee xs = [])$
 $\langle \text{proof} \rangle$

lemma *drop-eq-Nil[simp]*: $(\text{drop } n \text{ } xs = []) = (\text{length } xs \leq n)$
 $\langle \text{proof} \rangle$

lemma *take-map*: $\text{take } n \text{ } (\text{map } f \text{ } xs) = \text{map } f \text{ } (\text{take } n \text{ } xs)$

$\langle proof \rangle$

lemma *drop-map*: $drop\ n\ (map\ f\ xs) = map\ f\ (drop\ n\ xs)$
 $\langle proof \rangle$

lemma *rev-take*: $rev\ (take\ i\ xs) = drop\ (length\ xs - i)\ (rev\ xs)$
 $\langle proof \rangle$

lemma *rev-drop*: $rev\ (drop\ i\ xs) = take\ (length\ xs - i)\ (rev\ xs)$
 $\langle proof \rangle$

lemma *nth-take* [simp]: $i < n \implies (take\ n\ xs)!i = xs!i$
 $\langle proof \rangle$

lemma *nth-drop* [simp]:
 $n + i \leq length\ xs \implies (drop\ n\ xs)!i = xs!(n + i)$
 $\langle proof \rangle$

lemma *butlast-take*:
 $n \leq length\ xs \implies butlast\ (take\ n\ xs) = take\ (n - 1)\ xs$
 $\langle proof \rangle$

lemma *butlast-drop*: $butlast\ (drop\ n\ xs) = drop\ n\ (butlast\ xs)$
 $\langle proof \rangle$

lemma *take-butlast*: $n < length\ xs \implies take\ n\ (butlast\ xs) = take\ n\ xs$
 $\langle proof \rangle$

lemma *drop-butlast*: $drop\ n\ (butlast\ xs) = butlast\ (drop\ n\ xs)$
 $\langle proof \rangle$

lemma *hd-drop-conv-nth*: $\llbracket xs \neq []; n < length\ xs \rrbracket \implies hd(drop\ n\ xs) = xs!n$
 $\langle proof \rangle$

lemma *set-take-subset*: $set(take\ n\ xs) \subseteq set\ xs$
 $\langle proof \rangle$

lemma *set-drop-subset*: $set(drop\ n\ xs) \subseteq set\ xs$
 $\langle proof \rangle$

lemma *in-set-takeD*: $x : set(take\ n\ xs) \implies x : set\ xs$
 $\langle proof \rangle$

lemma *in-set-dropD*: $x : set(drop\ n\ xs) \implies x : set\ xs$
 $\langle proof \rangle$

lemma *append-eq-conv-conj*:
 $(xs @ ys = zs) = (xs = take\ (length\ xs)\ zs \wedge ys = drop\ (length\ xs)\ zs)$
 $\langle proof \rangle$

lemma *take-add*:

$i+j \leq \text{length}(xs) \implies \text{take } (i+j) \text{ } xs = \text{take } i \text{ } xs @ \text{take } j \text{ } (\text{drop } i \text{ } xs)$
 $\langle \text{proof} \rangle$

lemma *append-eq-append-conv-if*:

$(xs_1 @ xs_2 = ys_1 @ ys_2) =$
 $(\text{if } \text{size } xs_1 \leq \text{size } ys_1$
 $\text{ then } xs_1 = \text{take } (\text{size } xs_1) \text{ } ys_1 \wedge xs_2 = \text{drop } (\text{size } xs_1) \text{ } ys_1 @ ys_2$
 $\text{ else } \text{take } (\text{size } ys_1) \text{ } xs_1 = ys_1 \wedge \text{drop } (\text{size } ys_1) \text{ } xs_1 @ xs_2 = ys_2)$
 $\langle \text{proof} \rangle$

lemma *take-hd-drop*:

$n < \text{length } xs \implies \text{take } n \text{ } xs @ [\text{hd } (\text{drop } n \text{ } xs)] = \text{take } (\text{Suc } n) \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *id-take-nth-drop*:

$i < \text{length } xs \implies xs = \text{take } i \text{ } xs @ xs!i \# \text{drop } (\text{Suc } i) \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *upd-conv-take-nth-drop*:

$i < \text{length } xs \implies xs[i:=a] = \text{take } i \text{ } xs @ a \# \text{drop } (\text{Suc } i) \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *nth-drop'*:

$i < \text{length } xs \implies xs ! i \# \text{drop } (\text{Suc } i) \text{ } xs = \text{drop } i \text{ } xs$
 $\langle \text{proof} \rangle$

39.1.15 *takeWhile* and *dropWhile*

lemma *takeWhile-dropWhile-id* [simp]: $\text{takeWhile } P \text{ } xs @ \text{dropWhile } P \text{ } xs = xs$
 $\langle \text{proof} \rangle$

lemma *takeWhile-append1* [simp]:

$[| x : \text{set } xs; \sim P(x) |] \implies \text{takeWhile } P \text{ } (xs @ ys) = \text{takeWhile } P \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *takeWhile-append2* [simp]:

$(!!x. x : \text{set } xs \implies P \text{ } x) \implies \text{takeWhile } P \text{ } (xs @ ys) = xs @ \text{takeWhile } P \text{ } ys$
 $\langle \text{proof} \rangle$

lemma *takeWhile-tail*: $\neg P \text{ } x \implies \text{takeWhile } P \text{ } (xs @ (x \# l)) = \text{takeWhile } P \text{ } xs$
 $\langle \text{proof} \rangle$

lemma *dropWhile-append1* [simp]:

$[| x : \text{set } xs; \sim P(x) |] \implies \text{dropWhile } P \text{ } (xs @ ys) = (\text{dropWhile } P \text{ } xs) @ ys$
 $\langle \text{proof} \rangle$

lemma *dropWhile-append2* [simp]:

$(!!x. x : \text{set } xs \implies P(x)) \implies \text{dropWhile } P \ (xs @ ys) = \text{dropWhile } P \ ys$
 $\langle \text{proof} \rangle$

lemma *set-takeWhileD*: $x : \text{set } (\text{takeWhile } P \ xs) \implies x : \text{set } xs \wedge P \ x$
 $\langle \text{proof} \rangle$

lemma *takeWhile-eq-all-conv*[*simp*]:
 $(\text{takeWhile } P \ xs = xs) = (\forall x \in \text{set } xs. P \ x)$
 $\langle \text{proof} \rangle$

lemma *dropWhile-eq-Nil-conv*[*simp*]:
 $(\text{dropWhile } P \ xs = []) = (\forall x \in \text{set } xs. P \ x)$
 $\langle \text{proof} \rangle$

lemma *dropWhile-eq-Cons-conv*:
 $(\text{dropWhile } P \ xs = y \# ys) = (xs = \text{takeWhile } P \ xs @ y \# ys \ \& \ \neg P \ y)$
 $\langle \text{proof} \rangle$

The following two lemmas could be generalized to an arbitrary property.

lemma *takeWhile-neq-rev*: $\llbracket \text{distinct } xs; x \in \text{set } xs \rrbracket \implies$
 $\text{takeWhile } (\lambda y. y \neq x) \ (\text{rev } xs) = \text{rev } (\text{tl } (\text{dropWhile } (\lambda y. y \neq x) \ xs))$
 $\langle \text{proof} \rangle$

lemma *dropWhile-neq-rev*: $\llbracket \text{distinct } xs; x \in \text{set } xs \rrbracket \implies$
 $\text{dropWhile } (\lambda y. y \neq x) \ (\text{rev } xs) = x \# \text{rev } (\text{takeWhile } (\lambda y. y \neq x) \ xs)$
 $\langle \text{proof} \rangle$

lemma *takeWhile-not-last*:
 $\llbracket xs \neq []; \text{distinct } xs \rrbracket \implies \text{takeWhile } (\lambda y. y \neq \text{last } xs) \ xs = \text{butlast } xs$
 $\langle \text{proof} \rangle$

lemma *takeWhile-cong* [*fundef-cong*, *recdef-cong*]:
 $\llbracket l = k; !!x. x : \text{set } l \implies P \ x = Q \ x \rrbracket$
 $\implies \text{takeWhile } P \ l = \text{takeWhile } Q \ k$
 $\langle \text{proof} \rangle$

lemma *dropWhile-cong* [*fundef-cong*, *recdef-cong*]:
 $\llbracket l = k; !!x. x : \text{set } l \implies P \ x = Q \ x \rrbracket$
 $\implies \text{dropWhile } P \ l = \text{dropWhile } Q \ k$
 $\langle \text{proof} \rangle$

39.1.16 *zip*

lemma *zip-Nil* [*simp*]: $\text{zip } [] \ ys = []$
 $\langle \text{proof} \rangle$

lemma *zip-Cons-Cons* [*simp*]: $\text{zip } (x \# xs) \ (y \# ys) = (x, y) \# \text{zip } xs \ ys$
 $\langle \text{proof} \rangle$

declare *zip-Cons* [*simp del*]

lemma *zip-Cons1*:

$\text{zip } (x\#xs) \text{ } ys = (\text{case } ys \text{ of } [] \Rightarrow [] \mid y\#ys \Rightarrow (x,y)\#\text{zip } xs \text{ } ys)$
 $\langle \text{proof} \rangle$

lemma *length-zip* [*simp*]:

$\text{length } (\text{zip } xs \text{ } ys) = \min (\text{length } xs) (\text{length } ys)$
 $\langle \text{proof} \rangle$

lemma *zip-append1*:

$\text{zip } (xs \text{ @ } ys) \text{ } zs =$
 $\text{zip } xs \text{ } (\text{take } (\text{length } xs) \text{ } zs) \text{ @ } \text{zip } ys \text{ } (\text{drop } (\text{length } xs) \text{ } zs)$
 $\langle \text{proof} \rangle$

lemma *zip-append2*:

$\text{zip } xs \text{ } (ys \text{ @ } zs) =$
 $\text{zip } (\text{take } (\text{length } ys) \text{ } xs) \text{ } ys \text{ @ } \text{zip } (\text{drop } (\text{length } ys) \text{ } xs) \text{ } zs$
 $\langle \text{proof} \rangle$

lemma *zip-append* [*simp*]:

$[[] \text{ length } xs = \text{length } us; \text{ length } ys = \text{length } vs] ==>$
 $\text{zip } (xs@ys) \text{ } (us@vs) = \text{zip } xs \text{ } us \text{ @ } \text{zip } ys \text{ } vs$
 $\langle \text{proof} \rangle$

lemma *zip-rev*:

$\text{length } xs = \text{length } ys ==> \text{zip } (\text{rev } xs) \text{ } (\text{rev } ys) = \text{rev } (\text{zip } xs \text{ } ys)$
 $\langle \text{proof} \rangle$

lemma *map-zip-map*:

$\text{map } f \text{ } (\text{zip } (\text{map } g \text{ } xs) \text{ } ys) = \text{map } (\lambda(x,y). f(g \text{ } x, y)) \text{ } (\text{zip } xs \text{ } ys)$
 $\langle \text{proof} \rangle$

lemma *map-zip-map2*:

$\text{map } f \text{ } (\text{zip } xs \text{ } (\text{map } g \text{ } ys)) = \text{map } (\lambda(x,y). f(x, g \text{ } y)) \text{ } (\text{zip } xs \text{ } ys)$
 $\langle \text{proof} \rangle$

lemma *nth-zip* [*simp*]:

$[[] \text{ } i < \text{length } xs; i < \text{length } ys] ==> (\text{zip } xs \text{ } ys)!i = (xs!i, ys!i)$
 $\langle \text{proof} \rangle$

lemma *set-zip*:

$\text{set } (\text{zip } xs \text{ } ys) = \{(xs!i, ys!i) \mid i. i < \min (\text{length } xs) (\text{length } ys)\}$
 $\langle \text{proof} \rangle$

lemma *zip-update*:

$\text{length } xs = \text{length } ys ==> \text{zip } (xs[i:=x]) \text{ } (ys[i:=y]) = (\text{zip } xs \text{ } ys)[i:=(x,y)]$
 $\langle \text{proof} \rangle$

lemma *zip-replicate* [*simp*]:

$\text{zip } (\text{replicate } i \ x) \ (\text{replicate } j \ y) = \text{replicate } (\min i \ j) \ (x, y)$
 $\langle \text{proof} \rangle$

lemma *take-zip*:

$\text{take } n \ (\text{zip } xs \ ys) = \text{zip } (\text{take } n \ xs) \ (\text{take } n \ ys)$
 $\langle \text{proof} \rangle$

lemma *drop-zip*:

$\text{drop } n \ (\text{zip } xs \ ys) = \text{zip } (\text{drop } n \ xs) \ (\text{drop } n \ ys)$
 $\langle \text{proof} \rangle$

lemma *set-zip-leftD*:

$(x, y) \in \text{set } (\text{zip } xs \ ys) \implies x \in \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *set-zip-rightD*:

$(x, y) \in \text{set } (\text{zip } xs \ ys) \implies y \in \text{set } ys$
 $\langle \text{proof} \rangle$

lemma *in-set-zipE*:

$(x, y) : \text{set } (\text{zip } xs \ ys) \implies ([\ x : \text{set } xs; \ y : \text{set } ys \] \implies R) \implies R$
 $\langle \text{proof} \rangle$

lemma *zip-map-fst-snd*:

$\text{zip } (\text{map } \text{fst } zs) \ (\text{map } \text{snd } zs) = zs$
 $\langle \text{proof} \rangle$

lemma *zip-eq-conv*:

$\text{length } xs = \text{length } ys \implies \text{zip } xs \ ys = zs \longleftrightarrow \text{map } \text{fst } zs = xs \wedge \text{map } \text{snd } zs = ys$
 $\langle \text{proof} \rangle$

39.1.17 *list-all2*

lemma *list-all2-lengthD* [*intro?*]:

$\text{list-all2 } P \ xs \ ys \implies \text{length } xs = \text{length } ys$
 $\langle \text{proof} \rangle$

lemma *list-all2-Nil* [*iff*, *code*]: $\text{list-all2 } P \ [] \ ys = (ys = [])$

$\langle \text{proof} \rangle$

lemma *list-all2-Nil2* [*iff*, *code*]: $\text{list-all2 } P \ xs \ [] = (xs = [])$

$\langle \text{proof} \rangle$

lemma *list-all2-Cons* [*iff*, *code*]:

$\text{list-all2 } P \ (x \# xs) \ (y \# ys) = (P \ x \ y \wedge \text{list-all2 } P \ xs \ ys)$
 $\langle \text{proof} \rangle$

lemma *list-all2-Cons1*:

$list-all2\ P\ (x\ \# \ xs)\ ys = (\exists\ z\ zs.\ ys = z\ \# \ zs \wedge P\ x\ z \wedge list-all2\ P\ xs\ zs)$
 $\langle proof \rangle$

lemma *list-all2-Cons2*:

$list-all2\ P\ xs\ (y\ \# \ ys) = (\exists\ z\ zs.\ xs = z\ \# \ zs \wedge P\ z\ y \wedge list-all2\ P\ zs\ ys)$
 $\langle proof \rangle$

lemma *list-all2-rev [iff]*:

$list-all2\ P\ (rev\ xs)\ (rev\ ys) = list-all2\ P\ xs\ ys$
 $\langle proof \rangle$

lemma *list-all2-rev1*:

$list-all2\ P\ (rev\ xs)\ ys = list-all2\ P\ xs\ (rev\ ys)$
 $\langle proof \rangle$

lemma *list-all2-append1*:

$list-all2\ P\ (xs\ @\ ys)\ zs =$
 $(EX\ us\ vs.\ zs = us\ @\ vs \wedge length\ us = length\ xs \wedge length\ vs = length\ ys \wedge$
 $list-all2\ P\ xs\ us \wedge list-all2\ P\ ys\ vs)$
 $\langle proof \rangle$

lemma *list-all2-append2*:

$list-all2\ P\ xs\ (ys\ @\ zs) =$
 $(EX\ us\ vs.\ xs = us\ @\ vs \wedge length\ us = length\ ys \wedge length\ vs = length\ zs \wedge$
 $list-all2\ P\ us\ ys \wedge list-all2\ P\ vs\ zs)$
 $\langle proof \rangle$

lemma *list-all2-append*:

$length\ xs = length\ ys \implies$
 $list-all2\ P\ (xs@us)\ (ys@vs) = (list-all2\ P\ xs\ ys \wedge list-all2\ P\ us\ vs)$
 $\langle proof \rangle$

lemma *list-all2-appendI [intro?, trans]*:

$\llbracket list-all2\ P\ a\ b;\ list-all2\ P\ c\ d \rrbracket \implies list-all2\ P\ (a@c)\ (b@d)$
 $\langle proof \rangle$

lemma *list-all2-conv-all-nth*:

$list-all2\ P\ xs\ ys =$
 $(length\ xs = length\ ys \wedge (\forall\ i < length\ xs.\ P\ (xs!i)\ (ys!i)))$
 $\langle proof \rangle$

lemma *list-all2-trans*:

assumes *tr*: $!!a\ b\ c.\ P1\ a\ b \implies P2\ b\ c \implies P3\ a\ c$
shows $!!bs\ cs.\ list-all2\ P1\ as\ bs \implies list-all2\ P2\ bs\ cs \implies list-all2\ P3\ as\ cs$
 $(is\ !!bs\ cs.\ PROP\ ?Q\ as\ bs\ cs)$
 $\langle proof \rangle$

lemma *list-all2-all-nthI [intro?]*:

$length\ a = length\ b \implies (\bigwedge n.\ n < length\ a \implies P\ (a!n)\ (b!n)) \implies list-all2\ P\ a\ b$

$\langle \text{proof} \rangle$

lemma *list-all2I*:

$\forall x \in \text{set } (\text{zip } a \ b). \text{ split } P \ x \implies \text{length } a = \text{length } b \implies \text{list-all2 } P \ a \ b$
 $\langle \text{proof} \rangle$

lemma *list-all2-nthD*:

$\llbracket \text{list-all2 } P \ xs \ ys; p < \text{size } xs \rrbracket \implies P \ (xs!p) \ (ys!p)$
 $\langle \text{proof} \rangle$

lemma *list-all2-nthD2*:

$\llbracket \text{list-all2 } P \ xs \ ys; p < \text{size } ys \rrbracket \implies P \ (xs!p) \ (ys!p)$
 $\langle \text{proof} \rangle$

lemma *list-all2-map1*:

$\text{list-all2 } P \ (\text{map } f \ as) \ bs = \text{list-all2 } (\lambda x \ y. P \ (f \ x) \ y) \ as \ bs$
 $\langle \text{proof} \rangle$

lemma *list-all2-map2*:

$\text{list-all2 } P \ as \ (\text{map } f \ bs) = \text{list-all2 } (\lambda x \ y. P \ x \ (f \ y)) \ as \ bs$
 $\langle \text{proof} \rangle$

lemma *list-all2-refl* [intro?]:

$(\bigwedge x. P \ x \ x) \implies \text{list-all2 } P \ xs \ xs$
 $\langle \text{proof} \rangle$

lemma *list-all2-update-cong*:

$\llbracket i < \text{size } xs; \text{list-all2 } P \ xs \ ys; P \ x \ y \rrbracket \implies \text{list-all2 } P \ (xs[i:=x]) \ (ys[i:=y])$
 $\langle \text{proof} \rangle$

lemma *list-all2-update-cong2*:

$\llbracket \text{list-all2 } P \ xs \ ys; P \ x \ y; i < \text{length } ys \rrbracket \implies \text{list-all2 } P \ (xs[i:=x]) \ (ys[i:=y])$
 $\langle \text{proof} \rangle$

lemma *list-all2-takeI* [simp,intro?]:

$\text{list-all2 } P \ xs \ ys \implies \text{list-all2 } P \ (\text{take } n \ xs) \ (\text{take } n \ ys)$
 $\langle \text{proof} \rangle$

lemma *list-all2-dropI* [simp,intro?]:

$\text{list-all2 } P \ as \ bs \implies \text{list-all2 } P \ (\text{drop } n \ as) \ (\text{drop } n \ bs)$
 $\langle \text{proof} \rangle$

lemma *list-all2-mono* [intro?]:

$\text{list-all2 } P \ xs \ ys \implies (\bigwedge xs \ ys. P \ xs \ ys \implies Q \ xs \ ys) \implies \text{list-all2 } Q \ xs \ ys$
 $\langle \text{proof} \rangle$

lemma *list-all2-eq*:

$xs = ys \iff \text{list-all2 } (op =) \ xs \ ys$
 $\langle \text{proof} \rangle$

39.1.18 *foldl and foldr***lemma** *foldl-append* [*simp*]:
$$\text{foldl } f \ a \ (xs \ @ \ ys) = \text{foldl } f \ (\text{foldl } f \ a \ xs) \ ys$$

⟨*proof*⟩

lemma *foldr-append*[*simp*]: $\text{foldr } f \ (xs \ @ \ ys) \ a = \text{foldr } f \ xs \ (\text{foldr } f \ ys \ a)$ ⟨*proof*⟩**lemma** *foldr-map*: $\text{foldr } g \ (\text{map } f \ xs) \ a = \text{foldr } (g \ o \ f) \ xs \ a$ ⟨*proof*⟩

For efficient code generation: avoid intermediate list.

lemma *foldl-map*[*code unfold*]:
$$\text{foldl } g \ a \ (\text{map } f \ xs) = \text{foldl } (\%a \ x. \ g \ a \ (f \ x)) \ a \ xs$$

⟨*proof*⟩

lemma *foldl-cong* [*fundef-cong*, *recdef-cong*]:
$$\begin{aligned} & [| \ a = b; \ l = k; \ !a \ x. \ x : \text{set } l \implies f \ a \ x = g \ a \ x \ |] \\ & \implies \text{foldl } f \ a \ l = \text{foldl } g \ b \ k \end{aligned}$$

⟨*proof*⟩

lemma *foldr-cong* [*fundef-cong*, *recdef-cong*]:
$$\begin{aligned} & [| \ a = b; \ l = k; \ !a \ x. \ x : \text{set } l \implies f \ x \ a = g \ x \ a \ |] \\ & \implies \text{foldr } f \ l \ a = \text{foldr } g \ k \ b \end{aligned}$$

⟨*proof*⟩

lemma (in *semigroup-add*) *foldl-assoc*:
$$\text{shows } \text{foldl } op + \ (x+y) \ zs = x + (\text{foldl } op + \ y \ zs)$$

⟨*proof*⟩

lemma (in *monoid-add*) *foldl-absorb0*:
$$\text{shows } x + (\text{foldl } op + \ 0 \ zs) = \text{foldl } op + \ x \ zs$$

⟨*proof*⟩

The “First Duality Theorem” in Bird & Wadler:

lemma *foldl-foldr1-lemma*:
$$\text{foldl } op + \ a \ xs = a + \text{foldr } op + \ xs \ (0::'a::\text{monoid-add})$$

⟨*proof*⟩

corollary *foldl-foldr1*:
$$\text{foldl } op + \ 0 \ xs = \text{foldr } op + \ xs \ (0::'a::\text{monoid-add})$$

⟨*proof*⟩

The “Third Duality Theorem” in Bird & Wadler:

lemma *foldr-foldl*: $\text{foldr } f \ xs \ a = \text{foldl } (\%x \ y. \ f \ y \ x) \ a \ (\text{rev } xs)$ ⟨*proof*⟩**lemma** *foldl-foldr*: $\text{foldl } f \ a \ xs = \text{foldr } (\%x \ y. \ f \ y \ x) \ (\text{rev } xs) \ a$

<proof>

lemma (in *ab-semigroup-add*) *foldr-conv-foldl*: $\text{foldr } op + xs \ a = \text{foldl } op + a \ xs$
<proof>

Note: $n \leq \text{foldl } (op +) \ n \ ns$ looks simpler, but is more difficult to use because it requires an additional transitivity step.

lemma *start-le-sum*: $(m :: nat) \leq n \implies m \leq \text{foldl } (op +) \ 0 \ ns$
<proof>

lemma *elem-le-sum*: $(n :: nat) : \text{set } ns \implies n \leq \text{foldl } (op +) \ 0 \ ns$
<proof>

lemma *sum-eq-0-conv* [iff]:
 $(\text{foldl } (op +) \ (m :: nat) \ ns = 0) = (m = 0 \wedge (\forall n \in \text{set } ns. n = 0))$
<proof>

lemma *foldr-invariant*:
 $\llbracket Q \ x ; \forall x \in \text{set } xs. P \ x ; \forall x \ y. P \ x \wedge Q \ y \longrightarrow Q \ (f \ x \ y) \rrbracket \implies Q \ (\text{foldr } f \ xs \ x)$
<proof>

lemma *foldl-invariant*:
 $\llbracket Q \ x ; \forall x \in \text{set } xs. P \ x ; \forall x \ y. P \ x \wedge Q \ y \longrightarrow Q \ (f \ y \ x) \rrbracket \implies Q \ (\text{foldl } f \ x \ xs)$
<proof>

foldl and *concat*

lemma *foldl-conv-concat*:
 $\text{foldl } (op \ @) \ xs \ xss = xs \ @ \ \text{concat } xss$
<proof>

lemma *concat-conv-foldl*: $\text{concat } xss = \text{foldl } (op \ @) \ [] \ xss$
<proof>

39.1.19 List summation: *listsum* and \sum

lemma *listsum-append* [simp]: $\text{listsum } (xs \ @ \ ys) = \text{listsum } xs + \text{listsum } ys$
<proof>

lemma *listsum-rev* [simp]:
fixes $xs :: 'a :: \text{comm-monoid-add} \ \text{list}$
shows $\text{listsum } (\text{rev } xs) = \text{listsum } xs$
<proof>

lemma *listsum-foldr*: $\text{listsum } xs = \text{foldr } (op +) \ xs \ 0$
<proof>

lemma *length-concat*: $\text{length } (\text{concat } xss) = \text{listsum } (\text{map } \text{length } xss)$
<proof>

For efficient code generation — *listsum* is not tail recursive but *foldl* is.

lemma *listsum*[code unfold]: *listsum xs = foldl (op +) 0 xs*
 ⟨proof⟩

Some syntactic sugar for summing a function over a list:

syntax

-listsum :: pttrn => 'a list => 'b => 'b ((3SUM -<--. -) [0, 51, 10] 10)

syntax (*xsymbols*)

-listsum :: pttrn => 'a list => 'b => 'b ((3Σ -<--. -) [0, 51, 10] 10)

syntax (*HTML output*)

-listsum :: pttrn => 'a list => 'b => 'b ((3Σ -<--. -) [0, 51, 10] 10)

translations — Beware of argument permutation!

SUM x<-xs. b == CONST listsum (map (%x. b) xs)

Σ x<-xs. b == CONST listsum (map (%x. b) xs)

lemma *listsum-triv*: $(\sum x \leftarrow xs. r) = \text{of-nat } (\text{length } xs) * r$
 ⟨proof⟩

lemma *listsum-0* [simp]: $(\sum x \leftarrow xs. 0) = 0$
 ⟨proof⟩

For non-Abelian groups *xs* needs to be reversed on one side:

lemma *uminus-listsum-map*:

fixes *f :: 'a ⇒ 'b::ab-group-add*

shows — *listsum (map f xs) = (listsum (map (uminus o f) xs))*

⟨proof⟩

39.1.20 *upt*

lemma *upt-rec*[code]: $[i..<j] = (\text{if } i < j \text{ then } i \# [Suc\ i..<j] \text{ else } [])$
 — *simp* does not terminate!
 ⟨proof⟩

lemma *upt-conv-Nil* [simp]: $j \leq i \implies [i..<j] = []$
 ⟨proof⟩

lemma *upt-eq-Nil-conv*[simp]: $([i..<j] = []) = (j = 0 \vee j \leq i)$
 ⟨proof⟩

lemma *upt-eq-Cons-conv*:

$([i..<j] = x \# xs) = (i < j \ \& \ i = x \ \& \ [i+1..<j] = xs)$

⟨proof⟩

lemma *upt-Suc-append*: $i \leq j \implies [i..<(Suc\ j)] = [i..<j] @ [j]$
 — Only needed if *upt-Suc* is deleted from the simpset.
 ⟨proof⟩

lemma *upt-conv-Cons*: $i < j \implies [i..<j] = i \# [Suc\ i..<j]$
 ⟨proof⟩

lemma *upt-add-eq-append*: $i \leq j \implies [i..<j+k] = [i..<j] @ [j..<j+k]$
 — LOOPS as a simplrule, since $j \leq j$.
 $\langle proof \rangle$

lemma *length-upt [simp]*: $length [i..<j] = j - i$
 $\langle proof \rangle$

lemma *nth-upt [simp]*: $i + k < j \implies [i..<j] ! k = i + k$
 $\langle proof \rangle$

lemma *hd-upt [simp]*: $i < j \implies hd [i..<j] = i$
 $\langle proof \rangle$

lemma *last-upt [simp]*: $i < j \implies last [i..<j] = j - 1$
 $\langle proof \rangle$

lemma *take-upt [simp]*: $i+m \leq n \implies take\ m\ [i..<n] = [i..<i+m]$
 $\langle proof \rangle$

lemma *drop-upt [simp]*: $drop\ m\ [i..<j] = [i+m..<j]$
 $\langle proof \rangle$

lemma *map-Suc-upt*: $map\ Suc\ [m..<n] = [Suc\ m..<Suc\ n]$
 $\langle proof \rangle$

lemma *nth-map-upt*: $i < n-m \implies (map\ f\ [m..<n]) ! i = f(m+i)$
 $\langle proof \rangle$

lemma *nth-take-lemma*:
 $k \leq length\ xs \implies k \leq length\ ys \implies$
 $(!i. i < k \longrightarrow xs!i = ys!i) \implies take\ k\ xs = take\ k\ ys$
 $\langle proof \rangle$

lemma *nth-equalityI*:
 $[| length\ xs = length\ ys; ALL\ i < length\ xs. xs!i = ys!i |] \implies xs = ys$
 $\langle proof \rangle$

lemma *map-nth*:
 $map\ (\lambda i. xs ! i)\ [0..<length\ xs] = xs$
 $\langle proof \rangle$

lemma *list-all2-antisym*:
 $[| (\bigwedge x\ y. [P\ x\ y; Q\ y\ x] \implies x = y); list-all2\ P\ xs\ ys; list-all2\ Q\ ys\ xs |]$
 $\implies xs = ys$
 $\langle proof \rangle$

lemma *take-equalityI*: $(\forall i. \text{take } i \text{ } xs = \text{take } i \text{ } ys) \implies xs = ys$

— The famous take-lemma.

$\langle \text{proof} \rangle$

lemma *take-Cons'*:

$\text{take } n \text{ } (x \# xs) = (\text{if } n = 0 \text{ then } [] \text{ else } x \# \text{take } (n - 1) \text{ } xs)$

$\langle \text{proof} \rangle$

lemma *drop-Cons'*:

$\text{drop } n \text{ } (x \# xs) = (\text{if } n = 0 \text{ then } x \# xs \text{ else } \text{drop } (n - 1) \text{ } xs)$

$\langle \text{proof} \rangle$

lemma *nth-Cons'*: $(x \# xs)!n = (\text{if } n = 0 \text{ then } x \text{ else } xs!(n - 1))$

$\langle \text{proof} \rangle$

lemmas *take-Cons-number-of* = *take-Cons'*[*of number-of v, standard*]

lemmas *drop-Cons-number-of* = *drop-Cons'*[*of number-of v, standard*]

lemmas *nth-Cons-number-of* = *nth-Cons'*[*of - - number-of v, standard*]

declare *take-Cons-number-of* [*simp*]

drop-Cons-number-of [*simp*]

nth-Cons-number-of [*simp*]

39.1.21 *distinct and remdups*

lemma *distinct-append* [*simp*]:

$\text{distinct } (xs @ ys) = (\text{distinct } xs \wedge \text{distinct } ys \wedge \text{set } xs \cap \text{set } ys = \{\})$

$\langle \text{proof} \rangle$

lemma *distinct-rev*[*simp*]: $\text{distinct}(\text{rev } xs) = \text{distinct } xs$

$\langle \text{proof} \rangle$

lemma *set-remdups* [*simp*]: $\text{set } (\text{remdups } xs) = \text{set } xs$

$\langle \text{proof} \rangle$

lemma *distinct-remdups* [*iff*]: $\text{distinct } (\text{remdups } xs)$

$\langle \text{proof} \rangle$

lemma *distinct-remdups-id*: $\text{distinct } xs \implies \text{remdups } xs = xs$

$\langle \text{proof} \rangle$

lemma *remdups-id-iff-distinct* [*simp*]: $\text{remdups } xs = xs \longleftrightarrow \text{distinct } xs$

$\langle \text{proof} \rangle$

lemma *finite-distinct-list*: $\text{finite } A \implies \exists X \text{ } xs. \text{set } xs = A \ \& \ \text{distinct } xs$

$\langle \text{proof} \rangle$

lemma *remdups-eq-nil-iff* [*simp*]: $(\text{remdups } x = []) = (x = [])$

$\langle \text{proof} \rangle$

lemma *remdups-eq-nil-right-iff* [simp]: $([] = \text{remdups } x) = (x = [])$
 $\langle \text{proof} \rangle$

lemma *length-remdups-leq*[iff]: $\text{length}(\text{remdups } xs) \leq \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *length-remdups-eq*[iff]:
 $(\text{length } (\text{remdups } xs) = \text{length } xs) = (\text{remdups } xs = xs)$
 $\langle \text{proof} \rangle$

lemma *distinct-map*:
 $\text{distinct}(\text{map } f \ xs) = (\text{distinct } xs \ \& \ \text{inj-on } f \ (\text{set } xs))$
 $\langle \text{proof} \rangle$

lemma *distinct-filter* [simp]: $\text{distinct } xs \implies \text{distinct } (\text{filter } P \ xs)$
 $\langle \text{proof} \rangle$

lemma *distinct-upt*[simp]: $\text{distinct}[i..<j]$
 $\langle \text{proof} \rangle$

lemma *distinct-take*[simp]: $\text{distinct } xs \implies \text{distinct } (\text{take } i \ xs)$
 $\langle \text{proof} \rangle$

lemma *distinct-drop*[simp]: $\text{distinct } xs \implies \text{distinct } (\text{drop } i \ xs)$
 $\langle \text{proof} \rangle$

lemma *distinct-list-update*:
assumes d : $\text{distinct } xs$ **and** a : $a \notin \text{set } xs - \{xs[i]\}$
shows $\text{distinct } (xs[i:=a])$
 $\langle \text{proof} \rangle$

It is best to avoid this indexed version of *distinct*, but sometimes it is useful.

lemma *distinct-conv-nth*:
 $\text{distinct } xs = (\forall i < \text{size } xs. \forall j < \text{size } xs. i \neq j \longrightarrow xs[i] \neq xs[j])$
 $\langle \text{proof} \rangle$

lemma *nth-eq-iff-index-eq*:
 $\llbracket \text{distinct } xs; i < \text{length } xs; j < \text{length } xs \rrbracket \implies (xs[i] = xs[j]) = (i = j)$
 $\langle \text{proof} \rangle$

lemma *distinct-card*: $\text{distinct } xs \implies \text{card } (\text{set } xs) = \text{size } xs$
 $\langle \text{proof} \rangle$

lemma *card-distinct*: $\text{card } (\text{set } xs) = \text{size } xs \implies \text{distinct } xs$
 $\langle \text{proof} \rangle$

lemma *not-distinct-decomp*: $\sim \text{distinct } ws \implies \exists x s y s z s y. ws = xs @ [y] @ ys @ [y] @ zs$
 <proof>

lemma *length-remdups-concat*:
 $\text{length}(\text{remdups}(\text{concat } xss)) = \text{card}(\bigcup xs \in \text{set } xss. \text{set } xs)$
 <proof>

39.1.22 *remove1*

lemma *remove1-append*:
 $\text{remove1 } x (xs @ ys) =$
 $(\text{if } x \in \text{set } xs \text{ then } \text{remove1 } x xs @ ys \text{ else } xs @ \text{remove1 } x ys)$
 <proof>

lemma *in-set-remove1[simp]*:
 $a \neq b \implies a : \text{set}(\text{remove1 } b xs) = (a : \text{set } xs)$
 <proof>

lemma *set-remove1-subset*: $\text{set}(\text{remove1 } x xs) \leq \text{set } xs$
 <proof>

lemma *set-remove1-eq [simp]*: $\text{distinct } xs \implies \text{set}(\text{remove1 } x xs) = \text{set } xs - \{x\}$
 <proof>

lemma *length-remove1*:
 $\text{length}(\text{remove1 } x xs) = (\text{if } x : \text{set } xs \text{ then } \text{length } xs - 1 \text{ else } \text{length } xs)$
 <proof>

lemma *remove1-filter-not[simp]*:
 $\neg P x \implies \text{remove1 } x (\text{filter } P xs) = \text{filter } P xs$
 <proof>

lemma *notin-set-remove1[simp]*: $x \sim : \text{set } xs \implies x \sim : \text{set}(\text{remove1 } y xs)$
 <proof>

lemma *distinct-remove1[simp]*: $\text{distinct } xs \implies \text{distinct}(\text{remove1 } x xs)$
 <proof>

39.1.23 *removeAll*

lemma *removeAll-append[simp]*:
 $\text{removeAll } x (xs @ ys) = \text{removeAll } x xs @ \text{removeAll } x ys$
 <proof>

lemma *set-removeAll[simp]*: $\text{set}(\text{removeAll } x xs) = \text{set } xs - \{x\}$
 <proof>

lemma *removeAll-id[simp]*: $x \notin \text{set } xs \implies \text{removeAll } x xs = xs$
 <proof>

lemma *removeAll-filter-not*[simp]:
 $\neg P\ x \implies \text{removeAll}\ x\ (\text{filter}\ P\ xs) = \text{filter}\ P\ xs$
 <proof>

lemma *distinct-remove1-removeAll*:
 $\text{distinct}\ xs \implies \text{remove1}\ x\ xs = \text{removeAll}\ x\ xs$
 <proof>

lemma *map-removeAll-inj-on*: $\text{inj-on}\ f\ (\text{insert}\ x\ (\text{set}\ xs)) \implies$
 $\text{map}\ f\ (\text{removeAll}\ x\ xs) = \text{removeAll}\ (f\ x)\ (\text{map}\ f\ xs)$
 <proof>

lemma *map-removeAll-inj*: $\text{inj}\ f \implies$
 $\text{map}\ f\ (\text{removeAll}\ x\ xs) = \text{removeAll}\ (f\ x)\ (\text{map}\ f\ xs)$
 <proof>

39.1.24 replicate

lemma *length-replicate* [simp]: $\text{length}\ (\text{replicate}\ n\ x) = n$
 <proof>

lemma *map-replicate* [simp]: $\text{map}\ f\ (\text{replicate}\ n\ x) = \text{replicate}\ n\ (f\ x)$
 <proof>

lemma *replicate-app-Cons-same*:
 $(\text{replicate}\ n\ x) @ (x \# xs) = x \# \text{replicate}\ n\ x @ xs$
 <proof>

lemma *rev-replicate* [simp]: $\text{rev}\ (\text{replicate}\ n\ x) = \text{replicate}\ n\ x$
 <proof>

lemma *replicate-add*: $\text{replicate}\ (n + m)\ x = \text{replicate}\ n\ x @ \text{replicate}\ m\ x$
 <proof>

Courtesy of Matthias Daum:

lemma *append-replicate-commute*:
 $\text{replicate}\ n\ x @ \text{replicate}\ k\ x = \text{replicate}\ k\ x @ \text{replicate}\ n\ x$
 <proof>

lemma *hd-replicate* [simp]: $n \neq 0 \implies \text{hd}\ (\text{replicate}\ n\ x) = x$
 <proof>

lemma *tl-replicate* [simp]: $n \neq 0 \implies \text{tl}\ (\text{replicate}\ n\ x) = \text{replicate}\ (n - 1)\ x$
 <proof>

lemma *last-replicate* [simp]: $n \neq 0 \implies \text{last } (\text{replicate } n \ x) = x$
 ⟨proof⟩

lemma *nth-replicate* [simp]: $i < n \implies (\text{replicate } n \ x)!i = x$
 ⟨proof⟩

Courtesy of Matthias Daum (2 lemmas):

lemma *take-replicate* [simp]: $\text{take } i \ (\text{replicate } k \ x) = \text{replicate } (\min i \ k) \ x$
 ⟨proof⟩

lemma *drop-replicate* [simp]: $\text{drop } i \ (\text{replicate } k \ x) = \text{replicate } (k-i) \ x$
 ⟨proof⟩

lemma *set-replicate-Suc*: $\text{set } (\text{replicate } (\text{Suc } n) \ x) = \{x\}$
 ⟨proof⟩

lemma *set-replicate* [simp]: $n \neq 0 \implies \text{set } (\text{replicate } n \ x) = \{x\}$
 ⟨proof⟩

lemma *set-replicate-conv-if*: $\text{set } (\text{replicate } n \ x) = (\text{if } n = 0 \text{ then } \{\} \text{ else } \{x\})$
 ⟨proof⟩

lemma *in-set-replicateD*: $x : \text{set } (\text{replicate } n \ y) \implies x = y$
 ⟨proof⟩

lemma *replicate-append-same*:
 $\text{replicate } i \ x \ @ \ [x] = x \ \# \ \text{replicate } i \ x$
 ⟨proof⟩

lemma *map-replicate-trivial*:
 $\text{map } (\lambda i. \ x) \ [0..i] = \text{replicate } i \ x$
 ⟨proof⟩

lemma *replicate-empty* [simp]: $(\text{replicate } n \ x = []) \longleftrightarrow n=0$
 ⟨proof⟩

lemma *empty-replicate* [simp]: $([] = \text{replicate } n \ x) \longleftrightarrow n=0$
 ⟨proof⟩

lemma *replicate-eq-replicate* [simp]:
 $(\text{replicate } m \ x = \text{replicate } n \ y) \longleftrightarrow (m=n \ \& \ (m \neq 0 \longrightarrow x=y))$
 ⟨proof⟩

39.1.25 rotate1 and rotate

lemma *rotate-simps* [simp]: $\text{rotate1 } [] = [] \wedge \text{rotate1 } (x \# xs) = xs \ @ \ [x]$
 ⟨proof⟩

lemma *rotate0[simp]*: $\text{rotate } 0 = \text{id}$

$\langle \text{proof} \rangle$

lemma *rotate-Suc[simp]*: $\text{rotate } (\text{Suc } n) \text{ } xs = \text{rotate1 } (\text{rotate } n \text{ } xs)$

$\langle \text{proof} \rangle$

lemma *rotate-add*:

$\text{rotate } (m+n) = \text{rotate } m \circ \text{rotate } n$

$\langle \text{proof} \rangle$

lemma *rotate-rotate*: $\text{rotate } m \text{ } (\text{rotate } n \text{ } xs) = \text{rotate } (m+n) \text{ } xs$

$\langle \text{proof} \rangle$

lemma *rotate1-rotate-swap*: $\text{rotate1 } (\text{rotate } n \text{ } xs) = \text{rotate } n \text{ } (\text{rotate1 } xs)$

$\langle \text{proof} \rangle$

lemma *rotate1-length01[simp]*: $\text{length } xs \leq 1 \implies \text{rotate1 } xs = xs$

$\langle \text{proof} \rangle$

lemma *rotate-length01[simp]*: $\text{length } xs \leq 1 \implies \text{rotate } n \text{ } xs = xs$

$\langle \text{proof} \rangle$

lemma *rotate1-hd-tl*: $xs \neq [] \implies \text{rotate1 } xs = \text{tl } xs @ [\text{hd } xs]$

$\langle \text{proof} \rangle$

lemma *rotate-drop-take*:

$\text{rotate } n \text{ } xs = \text{drop } (n \bmod \text{length } xs) \text{ } xs @ \text{take } (n \bmod \text{length } xs) \text{ } xs$

$\langle \text{proof} \rangle$

lemma *rotate-conv-mod*: $\text{rotate } n \text{ } xs = \text{rotate } (n \bmod \text{length } xs) \text{ } xs$

$\langle \text{proof} \rangle$

lemma *rotate-id[simp]*: $n \bmod \text{length } xs = 0 \implies \text{rotate } n \text{ } xs = xs$

$\langle \text{proof} \rangle$

lemma *length-rotate1[simp]*: $\text{length}(\text{rotate1 } xs) = \text{length } xs$

$\langle \text{proof} \rangle$

lemma *length-rotate[simp]*: $\text{length}(\text{rotate } n \text{ } xs) = \text{length } xs$

$\langle \text{proof} \rangle$

lemma *distinct1-rotate[simp]*: $\text{distinct}(\text{rotate1 } xs) = \text{distinct } xs$

$\langle \text{proof} \rangle$

lemma *distinct-rotate[simp]*: $\text{distinct}(\text{rotate } n \text{ } xs) = \text{distinct } xs$

$\langle \text{proof} \rangle$

lemma *rotate-map*: $\text{rotate } n \text{ } (\text{map } f \text{ } xs) = \text{map } f \text{ } (\text{rotate } n \text{ } xs)$

$\langle \text{proof} \rangle$

lemma *set-rotate1*[simp]: $\text{set}(\text{rotate1 } xs) = \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *set-rotate*[simp]: $\text{set}(\text{rotate } n \ xs) = \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *rotate1-is-Nil-conv*[simp]: $(\text{rotate1 } xs = []) = (xs = [])$
 $\langle \text{proof} \rangle$

lemma *rotate-is-Nil-conv*[simp]: $(\text{rotate } n \ xs = []) = (xs = [])$
 $\langle \text{proof} \rangle$

lemma *rotate-rev*:
 $\text{rotate } n \ (\text{rev } xs) = \text{rev}(\text{rotate } (\text{length } xs - (n \bmod \text{length } xs)) \ xs)$
 $\langle \text{proof} \rangle$

lemma *hd-rotate-conv-nth*: $xs \neq [] \implies \text{hd}(\text{rotate } n \ xs) = xs!(n \bmod \text{length } xs)$
 $\langle \text{proof} \rangle$

39.1.26 *sublist* — a generalization of *nth* to sets

lemma *sublist-empty* [simp]: $\text{sublist } xs \ \{\} = []$
 $\langle \text{proof} \rangle$

lemma *sublist-nil* [simp]: $\text{sublist } [] \ A = []$
 $\langle \text{proof} \rangle$

lemma *length-sublist*:
 $\text{length}(\text{sublist } xs \ I) = \text{card}\{i. i < \text{length } xs \wedge i : I\}$
 $\langle \text{proof} \rangle$

lemma *sublist-shift-lemma-Suc*:
 $\text{map fst } (\text{filter } (\%p. P(\text{Suc}(\text{snd } p))) \ (\text{zip } xs \ is)) =$
 $\text{map fst } (\text{filter } (\%p. P(\text{snd } p)) \ (\text{zip } xs \ (\text{map } \text{Suc } is)))$
 $\langle \text{proof} \rangle$

lemma *sublist-shift-lemma*:
 $\text{map fst } [p < - \text{zip } xs \ [i..<i + \text{length } xs] . \text{snd } p : A] =$
 $\text{map fst } [p < - \text{zip } xs \ [0..<\text{length } xs] . \text{snd } p + i : A]$
 $\langle \text{proof} \rangle$

lemma *sublist-append*:
 $\text{sublist } (l @ l') \ A = \text{sublist } l \ A @ \text{sublist } l' \ \{j. j + \text{length } l : A\}$
 $\langle \text{proof} \rangle$

lemma *sublist-Cons*:
 $\text{sublist } (x \# l) \ A = (\text{if } 0:A \text{ then } [x] \text{ else } []) @ \text{sublist } l \ \{j. \text{Suc } j : A\}$

<proof>

lemma *set-sublist*: $\text{set}(\text{sublist } xs \ I) = \{xs!i \mid i.<\text{size } xs \wedge i \in I\}$
<proof>

lemma *set-sublist-subset*: $\text{set}(\text{sublist } xs \ I) \subseteq \text{set } xs$
<proof>

lemma *notin-set-sublistI*[simp]: $x \notin \text{set } xs \implies x \notin \text{set}(\text{sublist } xs \ I)$
<proof>

lemma *in-set-sublistD*: $x \in \text{set}(\text{sublist } xs \ I) \implies x \in \text{set } xs$
<proof>

lemma *sublist-singleton* [simp]: $\text{sublist } [x] \ A = (\text{if } 0 : A \text{ then } [x] \text{ else } [])$
<proof>

lemma *distinct-sublistI*[simp]: $\text{distinct } xs \implies \text{distinct}(\text{sublist } xs \ I)$
<proof>

lemma *sublist-upt-eq-take* [simp]: $\text{sublist } l \ \{.. $n\} = \text{take } n \ l$$

lemma *filter-in-sublist*:
 $\text{distinct } xs \implies \text{filter } (\%x. x \in \text{set}(\text{sublist } xs \ s)) \ xs = \text{sublist } xs \ s$
<proof>

39.1.27 splice

lemma *splice-Nil2* [simp, code]:
 $\text{splice } xs \ [] = xs$
<proof>

lemma *splice-Cons-Cons* [simp, code]:
 $\text{splice } (x \# xs) \ (y \# ys) = x \# y \# \text{splice } xs \ ys$
<proof>

declare *splice.simps*(2) [simp del, code del]

lemma *length-splice*[simp]: $\text{length}(\text{splice } xs \ ys) = \text{length } xs + \text{length } ys$
<proof>

39.1.28 Infiniteness

lemma *finite-maxlen*:
 $\text{finite } (M::'a \text{ list set}) \implies \exists n. \forall s:M. \text{size } s < n$
<proof>

lemma *infinite-UNIV-listI*: $\sim \text{finite}(\text{UNIV}::'a \text{ list set})$
 $\langle \text{proof} \rangle$

39.2 Sorting

Currently it is not shown that *sort* returns a permutation of its input because the nicest proof is via multisets, which are not yet available. Alternatively one could define a function that counts the number of occurrences of an element in a list and use that instead of multisets to state the correctness property.

context *linorder*
begin

lemma *sorted-Cons*: $\text{sorted } (x \# xs) = (\text{sorted } xs \ \& \ (\text{ALL } y:\text{set } xs. x \leq y))$
 $\langle \text{proof} \rangle$

lemma *sorted-append*:
 $\text{sorted } (xs @ ys) = (\text{sorted } xs \ \& \ \text{sorted } ys \ \& \ (\forall x \in \text{set } xs. \forall y \in \text{set } ys. x \leq y))$
 $\langle \text{proof} \rangle$

lemma *set-insort*: $\text{set}(\text{insort } x \ xs) = \text{insert } x \ (\text{set } xs)$
 $\langle \text{proof} \rangle$

lemma *set-sort[simp]*: $\text{set}(\text{sort } xs) = \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *distinct-insort*: $\text{distinct } (\text{insort } x \ xs) = (x \notin \text{set } xs \ \wedge \ \text{distinct } xs)$
 $\langle \text{proof} \rangle$

lemma *distinct-sort[simp]*: $\text{distinct } (\text{sort } xs) = \text{distinct } xs$
 $\langle \text{proof} \rangle$

lemma *sorted-insort*: $\text{sorted } (\text{insort } x \ xs) = \text{sorted } xs$
 $\langle \text{proof} \rangle$

theorem *sorted-sort[simp]*: $\text{sorted } (\text{sort } xs)$
 $\langle \text{proof} \rangle$

lemma *insort-is-Cons*: $\forall x \in \text{set } xs. a \leq x \implies \text{insort } a \ xs = a \ \# \ xs$
 $\langle \text{proof} \rangle$

lemma *sorted-remove1*: $\text{sorted } xs \implies \text{sorted } (\text{remove1 } a \ xs)$
 $\langle \text{proof} \rangle$

lemma *insort-remove1*: $\llbracket a \in \text{set } xs; \text{sorted } xs \rrbracket \implies \text{insort } a \ (\text{remove1 } a \ xs) = xs$
 $\langle \text{proof} \rangle$

lemma *sorted-remdups[simp]*:

sorted l \implies *sorted (remdups l)*
 $\langle \text{proof} \rangle$

lemma *sorted-distinct-set-unique*:
assumes *sorted xs distinct xs sorted ys distinct ys set xs = set ys*
shows *xs = ys*
 $\langle \text{proof} \rangle$

lemma *finite-sorted-distinct-unique*:
shows *finite A* \implies *EX! xs. set xs = A & sorted xs & distinct xs*
 $\langle \text{proof} \rangle$

lemma *sorted-take*:
sorted xs \implies *sorted (take n xs)*
 $\langle \text{proof} \rangle$

lemma *sorted-drop*:
sorted xs \implies *sorted (drop n xs)*
 $\langle \text{proof} \rangle$

end

lemma *sorted-upt[simp]: sorted[i..<j]*
 $\langle \text{proof} \rangle$

39.2.1 *sorted-list-of-set*

This function maps (finite) linearly ordered sets to sorted lists. Warning: in most cases it is not a good idea to convert from sets to lists but one should convert in the other direction (via *set*).

context *linorder*
begin

definition
sorted-list-of-set :: 'a set \Rightarrow 'a list **where**
 $\text{[code del]: sorted-list-of-set } A == \text{THE } xs. \text{ set } xs = A \ \& \ \text{sorted } xs \ \& \ \text{distinct } xs$

lemma *sorted-list-of-set[simp]: finite A* \implies
set(sorted-list-of-set A) = A &
sorted(sorted-list-of-set A) & distinct(sorted-list-of-set A)
 $\langle \text{proof} \rangle$

lemma *sorted-list-of-empty[simp]: sorted-list-of-set {} = []*
 $\langle \text{proof} \rangle$

end

39.2.2 upto: the generic interval-list

class *finite-intvl-succ* = *linorder* +
fixes *successor* :: 'a \Rightarrow 'a
assumes *finite-intvl*: *finite*{*a..b*}
and *successor-incr*: *a* < *successor a*
and *ord-discrete*: $\neg(\exists x. a < x \ \& \ x < \text{successor } a)$

context *finite-intvl-succ*
begin

definition

upto :: 'a \Rightarrow 'a \Rightarrow 'a list ((1[-./-])) **where**
upto *i j* == *sorted-list-of-set* {*i..j*}

lemma *upto[simp]*: *set*[*a..b*] = {*a..b*} & *sorted*[*a..b*] & *distinct*[*a..b*]
 <proof>

lemma *insert-intvl*: *i* \leq *j* \implies *insert* *i* {*successor i..j*} = {*i..j*}
 <proof>

lemma *sorted-list-of-set-rec*: *i* \leq *j* \implies
sorted-list-of-set {*i..j*} = *i* # *sorted-list-of-set* {*successor i..j*}
 <proof>

lemma *sorted-list-of-set-rec2*: *i* \leq *j* \implies
sorted-list-of-set {*i..successor j*} =
sorted-list-of-set {*i..j*} @ [*successor j*]
 <proof>

lemma *upto-rec[code]*: [*i..j*] = (if *i* \leq *j* then *i* # [*successor i..j*] else [])
 <proof>

lemma *upto-empty[simp]*: *j* < *i* \implies [*i..j*] = []
 <proof>

lemma *upto-rec2*: *i* \leq *j* \implies [*i..successor j*] = [*i..j*] @ [*successor j*]
 <proof>

end

The integers are an instance of the above class:

instantiation *int*:: *finite-intvl-succ*
begin

definition

successor-int-def: *successor* = (%*i*::int. *i*+1)

instance

<proof>

end

Now $[i..j]$ is defined for integers.

hide (open) *const successor*

lemma *upto-rec2-int*: $(i::int) \leq j \implies [i..j+1] = [i..j] @ [j+1]$
 $\langle proof \rangle$

39.2.3 *lists*: the list-forming operator over sets

inductive-set

lists :: 'a set ==> 'a list set

for *A* :: 'a set

where

Nil [*intro!*]: []: *lists A*

| *Cons* [*intro!*,*noatp*]: [| *a*: *A*; *l*: *lists A*] ==> *a*#*l* : *lists A*

inductive-cases *listsE* [*elim!*,*noatp*]: *x*#*l* : *lists A*

inductive-cases *listspE* [*elim!*,*noatp*]: *listsp A* (*x* # *l*)

lemma *listsp-mono* [*mono*]: $A \leq B \implies listsp A \leq listsp B$
 $\langle proof \rangle$

lemmas *lists-mono* = *listsp-mono* [*to-set pred-subset-eq*]

lemma *listsp-infI*:

assumes *l*: *listsp A l* **shows** *listsp B l* ==> *listsp (inf A B) l* $\langle proof \rangle$

lemmas *lists-IntI* = *listsp-infI* [*to-set*]

lemma *listsp-inf-eq* [*simp*]: $listsp (inf A B) = inf (listsp A) (listsp B)$
 $\langle proof \rangle$

lemmas *listsp-conj-eq* [*simp*] = *listsp-inf-eq* [*simplified inf-fun-eq inf-bool-eq*]

lemmas *lists-Int-eq* [*simp*] = *listsp-inf-eq* [*to-set pred-equals-eq*]

lemma *append-in-listsp-conv* [*iff*]:

$(listsp A (xs @ ys)) = (listsp A xs \wedge listsp A ys)$

$\langle proof \rangle$

lemmas *append-in-lists-conv* [*iff*] = *append-in-listsp-conv* [*to-set*]

lemma *in-listsp-conv-set*: $(listsp A xs) = (\forall x \in set xs. A x)$

— eliminate *listsp* in favour of *set*

$\langle proof \rangle$

lemmas *in-lists-conv-set* = *in-listsp-conv-set* [*to-set*]

lemma *in-listspD* [*dest!*,*noatp*]: *listsp A xs ==> $\forall x \in \text{set } xs. A x$*
<proof>

lemmas *in-listsD* [*dest!*,*noatp*] = *in-listspD* [*to-set*]

lemma *in-listspI* [*intro!*,*noatp*]: *$\forall x \in \text{set } xs. A x ==> \text{listsp } A xs$*
<proof>

lemmas *in-listsI* [*intro!*,*noatp*] = *in-listspI* [*to-set*]

lemma *lists-UNIV* [*simp*]: *lists UNIV = UNIV*
<proof>

39.2.4 Inductive definition for membership

inductive *ListMem* :: '*a* \Rightarrow '*a* list \Rightarrow bool

where

elem: *ListMem* *x* (*x* # *xs*)
| *insert*: *ListMem* *x* *xs* \Longrightarrow *ListMem* *x* (*y* # *xs*)

lemma *ListMem-iff*: (*ListMem* *x* *xs*) = (*x* \in *set* *xs*)
<proof>

39.2.5 Lists as Cartesian products

set-Cons *A* *Xs*: the set of lists with head drawn from *A* and tail drawn from *Xs*.

constdefs

set-Cons :: '*a* set \Rightarrow '*a* list set \Rightarrow '*a* list set
set-Cons *A* *XS* == {*z*. $\exists x xs. z = x \# xs$ & *x* \in *A* & *xs* \in *XS*}

declare *set-Cons-def* [*code del*]

lemma *set-Cons-sing-Nil* [*simp*]: *set-Cons* *A* {[]} = (%*x*. [*x*]) '*A*
<proof>

Yields the set of lists, all of the same length as the argument and with elements drawn from the corresponding element of the argument.

consts *listset* :: '*a* set list \Rightarrow '*a* list set

primrec

listset [] = {[]}
listset (*A* # *As*) = *set-Cons* *A* (*listset* *As*)

39.3 Relations on Lists

39.3.1 Length Lexicographic Ordering

These orderings preserve well-foundedness: shorter lists precede longer lists. These ordering are not used in dictionaries.

consts $lexn :: ('a * 'a) set \Rightarrow nat \Rightarrow ('a list * 'a list) set$
 — The lexicographic ordering for lists of the specified length

primrec
 $lexn\ r\ 0 = \{\}$
 $lexn\ r\ (Suc\ n) =$
 $(prod_fun\ (\%(x, xs). x \# xs)\ (\%(x, xs). x \# xs)\ ' (r < *lex* > lexn\ r\ n))\ Int$
 $\{(xs, ys). length\ xs = Suc\ n \wedge length\ ys = Suc\ n\}$

constdefs
 $lex :: ('a \times 'a) set \Rightarrow ('a list \times 'a list) set$
 $lex\ r == \bigcup n. lexn\ r\ n$
 — Holds only between lists of the same length

$lenlex :: ('a \times 'a) set \Rightarrow ('a list \times 'a list) set$
 $lenlex\ r == inv_image\ (less_than\ < *lex* > lex\ r)\ (\%xs. (length\ xs, xs))$
 — Compares lists by their length and then lexicographically

declare $lex_def\ [code\ del]$

lemma wf_lexn : $wf\ r \Rightarrow wf\ (lexn\ r\ n)$
 $\langle proof \rangle$

lemma $lexn_length$:
 $(xs, ys) : lexn\ r\ n \Rightarrow length\ xs = n \wedge length\ ys = n$
 $\langle proof \rangle$

lemma $wf_lex\ [intro!]$: $wf\ r \Rightarrow wf\ (lex\ r)$
 $\langle proof \rangle$

lemma $lexn_conv$:
 $lexn\ r\ n =$
 $\{(xs, ys). length\ xs = n \wedge length\ ys = n \wedge$
 $(\exists\ xys\ x\ y\ xs'\ ys'. xs = xys\ @\ x \# xs' \wedge ys = xys\ @\ y \# ys' \wedge (x, y):r)\}$
 $\langle proof \rangle$

lemma lex_conv :
 $lex\ r =$
 $\{(xs, ys). length\ xs = length\ ys \wedge$
 $(\exists\ xys\ x\ y\ xs'\ ys'. xs = xys\ @\ x \# xs' \wedge ys = xys\ @\ y \# ys' \wedge (x, y):r)\}$
 $\langle proof \rangle$

lemma $wf_lenlex\ [intro!]$: $wf\ r \Rightarrow wf\ (lenlex\ r)$
 $\langle proof \rangle$

lemma $lenlex_conv$:
 $lenlex\ r = \{(xs, ys). length\ xs < length\ ys \mid$
 $length\ xs = length\ ys \wedge (xs, ys) : lex\ r\}$
 $\langle proof \rangle$

lemma *Nil-notin-lex* [iff]: $([], ys) \notin lex\ r$
 $\langle proof \rangle$

lemma *Nil2-notin-lex* [iff]: $(xs, []) \notin lex\ r$
 $\langle proof \rangle$

lemma *Cons-in-lex* [simp]:
 $((x \# xs, y \# ys) : lex\ r) =$
 $((x, y) : r \wedge length\ xs = length\ ys \mid x = y \wedge (xs, ys) : lex\ r)$
 $\langle proof \rangle$

39.3.2 Lexicographic Ordering

Classical lexicographic ordering on lists, ie. ”a” j ”ab” j ”b”. This ordering does *not* preserve well-foundedness. Author: N. Voelker, March 2005.

constdefs
 $lexord :: ('a * 'a) set \Rightarrow ('a\ list * 'a\ list)\ set$
 $lexord\ r == \{(x, y). \exists a\ v. y = x @ a \# v \vee$
 $(\exists u\ a\ b\ v\ w. (a, b) \in r \wedge x = u @ (a \# v) \wedge y = u @ (b \# w))\}$
declare *lexord-def* [code del]

lemma *lexord-Nil-left*[simp]: $([], y) \in lexord\ r = (\exists a\ x. y = a \# x)$
 $\langle proof \rangle$

lemma *lexord-Nil-right*[simp]: $(x, []) \notin lexord\ r$
 $\langle proof \rangle$

lemma *lexord-cons-cons*[simp]:
 $((a \# x, b \# y) \in lexord\ r) = ((a, b) \in r \mid (a = b \ \& \ (x, y) \in lexord\ r))$
 $\langle proof \rangle$

lemmas *lexord-simps* = *lexord-Nil-left lexord-Nil-right lexord-cons-cons*

lemma *lexord-append-rightI*: $\exists b\ z. y = b \# z \Longrightarrow (x, x @ y) \in lexord\ r$
 $\langle proof \rangle$

lemma *lexord-append-left-rightI*:
 $(a, b) \in r \Longrightarrow (u @ a \# x, u @ b \# y) \in lexord\ r$
 $\langle proof \rangle$

lemma *lexord-append-leftI*: $(u, v) \in lexord\ r \Longrightarrow (x @ u, x @ v) \in lexord\ r$
 $\langle proof \rangle$

lemma *lexord-append-leftD*:
 $\llbracket (x @ u, x @ v) \in lexord\ r; (! a. (a, a) \notin r) \rrbracket \Longrightarrow (u, v) \in lexord\ r$
 $\langle proof \rangle$

lemma *lexord-take-index-conv*:

$((x,y) : \text{lexord } r) =$
 $((\text{length } x < \text{length } y \wedge \text{take } (\text{length } x) \ y = x) \vee$
 $(\exists i. i < \min(\text{length } x)(\text{length } y) \ \& \ \text{take } i \ x = \text{take } i \ y \ \& \ (x!i,y!i) \in r))$
 $\langle \text{proof} \rangle$
lemma *lexord-lex*: $(x,y) \in \text{lex } r = ((x,y) \in \text{lexord } r \wedge \text{length } x = \text{length } y)$
 $\langle \text{proof} \rangle$

lemma *lexord-irreflexive*: $(! x. (x,x) \notin r) \implies (y,y) \notin \text{lexord } r$
 $\langle \text{proof} \rangle$

lemma *lexord-trans*:
 $\llbracket (x, y) \in \text{lexord } r; (y, z) \in \text{lexord } r; \text{trans } r \rrbracket \implies (x, z) \in \text{lexord } r$
 $\langle \text{proof} \rangle$

lemma *lexord-transI*: $\text{trans } r \implies \text{trans } (\text{lexord } r)$
 $\langle \text{proof} \rangle$

lemma *lexord-linear*: $(! a \ b. (a,b) \in r \mid a = b \mid (b,a) \in r) \implies (x,y) : \text{lexord } r \mid x = y \mid (y,x) : \text{lexord } r$
 $\langle \text{proof} \rangle$

39.4 Lexicographic combination of measure functions

These are useful for termination proofs

definition

$\text{measures } fs = \text{inv-image } (\text{lex less-than}) \ (\%a. \text{map } (\%f. f \ a) \ fs)$

lemma *wf-measures[recdef-wf, simp]*: $\text{wf } (\text{measures } fs)$
 $\langle \text{proof} \rangle$

lemma *in-measures[simp]*:
 $(x, y) \in \text{measures } [] = \text{False}$
 $(x, y) \in \text{measures } (f \# fs)$
 $= (f \ x < f \ y \vee (f \ x = f \ y \wedge (x, y) \in \text{measures } fs))$
 $\langle \text{proof} \rangle$

lemma *measures-less*: $f \ x < f \ y \implies (x, y) \in \text{measures } (f \# fs)$
 $\langle \text{proof} \rangle$

lemma *measures-lesseq*: $f \ x \leq f \ y \implies (x, y) \in \text{measures } fs \implies (x, y) \in \text{measures } (f \# fs)$
 $\langle \text{proof} \rangle$

39.4.1 Lifting a Relation on List Elements to the Lists

inductive-set

$\text{listrel} :: ('a * 'a) \text{set} \implies ('a \text{ list} * 'a \text{ list}) \text{set}$
 $\text{for } r :: ('a * 'a) \text{set}$
where

Nil: $([], []) \in \text{listrel } r$
 | *Cons*: $[(x, y) \in r; (xs, ys) \in \text{listrel } r] \implies (x \# xs, y \# ys) \in \text{listrel } r$

inductive-cases *listrel-Nil1* [elim!]: $([], xs) \in \text{listrel } r$
inductive-cases *listrel-Nil2* [elim!]: $(xs, []) \in \text{listrel } r$
inductive-cases *listrel-Cons1* [elim!]: $(y \# ys, xs) \in \text{listrel } r$
inductive-cases *listrel-Cons2* [elim!]: $(xs, y \# ys) \in \text{listrel } r$

lemma *listrel-mono*: $r \subseteq s \implies \text{listrel } r \subseteq \text{listrel } s$
 <proof>

lemma *listrel-subset*: $r \subseteq A \times A \implies \text{listrel } r \subseteq \text{lists } A \times \text{lists } A$
 <proof>

lemma *listrel-refl-on*: $\text{refl-on } A \ r \implies \text{refl-on } (\text{lists } A) \ (\text{listrel } r)$
 <proof>

lemma *listrel-sym*: $\text{sym } r \implies \text{sym } (\text{listrel } r)$
 <proof>

lemma *listrel-trans*: $\text{trans } r \implies \text{trans } (\text{listrel } r)$
 <proof>

theorem *equiv-listrel*: $\text{equiv } A \ r \implies \text{equiv } (\text{lists } A) \ (\text{listrel } r)$
 <proof>

lemma *listrel-Nil* [simp]: $\text{listrel } r \text{ “ } \{[]\} = \{[]\}$
 <proof>

lemma *listrel-Cons*:
 $\text{listrel } r \text{ “ } \{x \# xs\} = \text{set-Cons } (r \text{ “ } \{x\}) \ (\text{listrel } r \text{ “ } \{xs\})$
 <proof>

39.5 Miscellany

39.5.1 Characters and strings

datatype *nibble* =
Nibble0 | *Nibble1* | *Nibble2* | *Nibble3* | *Nibble4* | *Nibble5* | *Nibble6* | *Nibble7*
 | *Nibble8* | *Nibble9* | *NibbleA* | *NibbleB* | *NibbleC* | *NibbleD* | *NibbleE* | *NibbleF*

lemma *UNIV-nibble*:
 $\text{UNIV} = \{\text{Nibble0}, \text{Nibble1}, \text{Nibble2}, \text{Nibble3}, \text{Nibble4}, \text{Nibble5}, \text{Nibble6}, \text{Nibble7},$
 $\text{Nibble8}, \text{Nibble9}, \text{NibbleA}, \text{NibbleB}, \text{NibbleC}, \text{NibbleD}, \text{NibbleE}, \text{NibbleF}\}$ (is -
 $= ?A)$
 <proof>

instance *nibble* :: *finite*
 <proof>

datatype *char* = *Char nibble nibble*

— Note: canonical order of character encoding coincides with standard term ordering

lemma *UNIV-char*:

$UNIV = \text{image } (\text{split } Char) (UNIV \times UNIV)$
 $\langle \text{proof} \rangle$

instance *char* :: *finite*

$\langle \text{proof} \rangle$

lemma *size-char* [*code*, *simp*]:

$\text{size } (c :: \text{char}) = 0$ $\langle \text{proof} \rangle$

lemma *char-size* [*code*, *simp*]:

$\text{char-size } (c :: \text{char}) = 0$ $\langle \text{proof} \rangle$

primrec *nibble-pair-of-char* :: *char* \Rightarrow *nibble* \times *nibble* **where**

$\text{nibble-pair-of-char } (Char\ n\ m) = (n, m)$

declare *nibble-pair-of-char.simps* [*code del*]

$\langle ML \rangle$

lemma *char-case-nibble-pair* [*code*, *code inline*]:

$\text{char-case } f = \text{split } f \circ \text{nibble-pair-of-char}$
 $\langle \text{proof} \rangle$

lemma *char-rec-nibble-pair* [*code*, *code inline*]:

$\text{char-rec } f = \text{split } f \circ \text{nibble-pair-of-char}$
 $\langle \text{proof} \rangle$

types *string* = *char list*

syntax

-*Char* :: *xstr* \Rightarrow *char* (*CHR* -)
 -*String* :: *xstr* \Rightarrow *string* (-)

$\langle ML \rangle$

39.6 Size function

lemma [*measure-function*]: *is-measure* *f* \Longrightarrow *is-measure* (*list-size* *f*)

$\langle \text{proof} \rangle$

lemma [*measure-function*]: *is-measure* *f* \Longrightarrow *is-measure* (*option-size* *f*)

$\langle \text{proof} \rangle$

lemma *list-size-estimation*[*termination-simp*]:
 $x \in \text{set } xs \implies y < f\ x \implies y < \text{list-size } f\ xs$
 <proof>

lemma *list-size-estimation'*[*termination-simp*]:
 $x \in \text{set } xs \implies y \leq f\ x \implies y \leq \text{list-size } f\ xs$
 <proof>

lemma *list-size-map*[*simp*]: $\text{list-size } f\ (\text{map } g\ xs) = \text{list-size } (f\ o\ g)\ xs$
 <proof>

lemma *list-size-pointwise*[*termination-simp*]:
 $(\bigwedge x. x \in \text{set } xs \implies f\ x < g\ x) \implies \text{list-size } f\ xs \leq \text{list-size } g\ xs$
 <proof>

39.7 Code generator

39.7.1 Setup

types-code

list (- *list*)

attach (*term-of*) <<
fun term-of-list *f* *T* = *HOLogic.mk-list* *T* *o* *map* *f*;
 >>

attach (*test*) <<
fun gen-list' *aG* *aT* *i* *j* = *frequency*
 [(*i*, *fn* () =>
 let
 val (*x*, *t*) = *aG* *j*;
 val (*xs*, *ts*) = *gen-list'* *aG* *aT* (*i*-1) *j*
 in (*x* :: *xs*, *fn* () => *HOLogic.cons-const* *aT* \$ *t* () \$ *ts* ()) *end*),
 (*1*, *fn* () => ([], *fn* () => *HOLogic.nil-const* *aT*))] ()
and gen-list *aG* *aT* *i* = *gen-list'* *aG* *aT* *i* *i*;
 >>

char (*string*)

attach (*term-of*) <<
val term-of-char = *HOLogic.mk-char* *o* *ord*;
 >>

attach (*test*) <<
fun gen-char *i* =
 let *val* *j* = *random-range* (*ord* *a*) (*Int.min* (*ord* *a* + *i*, *ord* *z*))
 in (*chr* *j*, *fn* () => *HOLogic.mk-char* *j*) *end*;
 >>

consts-code *Cons* ((- ::/ -))

code-type *list*

(*SML* - *list*)

(*OCaml* - *list*)

(*Haskell* ![-])

code-reserved *SML*

list

code-reserved *OCaml*

list

code-const *Nil*

(*SML* [])

(*OCaml* [])

(*Haskell* [])

⟨*ML*⟩

code-instance *list* :: *eq*

(*Haskell* −)

code-const *eq-class.eq* :: '*a*::*eq list* ⇒ '*a list* ⇒ *bool*

(*Haskell* **infixl** 4 ==)

⟨*ML*⟩

39.7.2 Generation of efficient code

primrec

member :: '*a* ⇒ '*a list* ⇒ *bool* (**infixl** *mem* 55)

where

x mem [] ⇔ *False*

| *x mem* (*y* # *ys*) ⇔ *x* = *y* ∨ *x mem ys*

primrec

null :: '*a list* ⇒ *bool*

where

null [] = *True*

| *null* (*x* # *xs*) = *False*

primrec

list-inter :: '*a list* ⇒ '*a list* ⇒ '*a list*

where

list-inter [] *bs* = []

| *list-inter* (*a* # *as*) *bs* =

(if *a* ∈ *set bs* then *a* # *list-inter as bs* else *list-inter as bs*)

primrec

list-all :: ('*a* ⇒ *bool*) ⇒ ('*a list* ⇒ *bool*)

where

list-all *P* [] = *True*

| *list-all* *P* (*x* # *xs*) = (*P x* ∧ *list-all P xs*)

primrec

$$\text{list-ex} :: ('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ list} \Rightarrow \text{bool}$$
where

$$\text{list-ex } P \ [] = \text{False}$$

$$| \text{list-ex } P \ (x \# xs) = (P \ x \vee \text{list-ex } P \ xs)$$
primrec

$$\text{filtermap} :: ('a \Rightarrow 'b \text{ option}) \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list}$$
where

$$\text{filtermap } f \ [] = []$$

$$| \text{filtermap } f \ (x \# xs) =$$

$$(\text{case } f \ x \text{ of } \text{None} \Rightarrow \text{filtermap } f \ xs$$

$$| \text{Some } y \Rightarrow y \# \text{filtermap } f \ xs)$$
primrec

$$\text{map-filter} :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list}$$
where

$$\text{map-filter } f \ P \ [] = []$$

$$| \text{map-filter } f \ P \ (x \# xs) =$$

$$(\text{if } P \ x \text{ then } f \ x \# \text{map-filter } f \ P \ xs \text{ else } \text{map-filter } f \ P \ xs)$$
primrec

$$\text{length-unique} :: 'a \text{ list} \Rightarrow \text{nat}$$
where

$$\text{length-unique} \ [] = 0$$

$$| \text{length-unique} \ (x \# xs) =$$

$$(\text{if } x \in \text{set } xs \text{ then } \text{length-unique } xs \text{ else } \text{Suc } (\text{length-unique } xs))$$

Only use *mem* for generating executable code. Otherwise use $x \in \text{set } xs$ instead — it is much easier to reason about. The same is true for *list-all* and *list-ex*: write $\forall x \in \text{set } xs$ and $\exists x \in \text{set } xs$ instead because the HOL quantifiers are already known to the automatic provers. In fact, the declarations in the code subsection make sure that \in , $\forall x \in \text{set } xs$ and $\exists x \in \text{set } xs$ are implemented efficiently.

Efficient emptiness check is implemented by *null*.

The functions *filtermap* and *map-filter* are just there to generate efficient code. Do not use them for modelling and proving.

lemma *rev-foldl-cons* [code]:

$$\text{rev } xs = \text{foldl } (\lambda xs \ x. x \# xs) \ [] \ xs$$

<proof>

lemma *mem-iff* [code post]:

$$x \text{ mem } xs \longleftrightarrow x \in \text{set } xs$$

<proof>

lemmas *in-set-code* [code unfold] = *mem-iff* [symmetric]

lemma *empty-null* [code inline]:

$xs = [] \longleftrightarrow null\ xs$
 $\langle proof \rangle$

lemmas *null-empty* [code post] =
empty-null [symmetric]

lemma *list-inter-conv*:
 $set\ (list\ inter\ xs\ ys) = set\ xs \cap set\ ys$
 $\langle proof \rangle$

lemma *list-all-iff* [code post]:
 $list\ all\ P\ xs \longleftrightarrow (\forall x \in set\ xs. P\ x)$
 $\langle proof \rangle$

lemmas *list-ball-code* [code unfold] = *list-all-iff* [symmetric]

lemma *list-all-append* [simp]:
 $list\ all\ P\ (xs\ @\ ys) \longleftrightarrow (list\ all\ P\ xs \wedge list\ all\ P\ ys)$
 $\langle proof \rangle$

lemma *list-all-rev* [simp]:
 $list\ all\ P\ (rev\ xs) \longleftrightarrow list\ all\ P\ xs$
 $\langle proof \rangle$

lemma *list-all-length*:
 $list\ all\ P\ xs \longleftrightarrow (\forall n < length\ xs. P\ (xs\ !\ n))$
 $\langle proof \rangle$

lemma *list-ex-iff* [code post]:
 $list\ ex\ P\ xs \longleftrightarrow (\exists x \in set\ xs. P\ x)$
 $\langle proof \rangle$

lemmas *list-bex-code* [code unfold] =
list-ex-iff [symmetric]

lemma *list-ex-length*:
 $list\ ex\ P\ xs \longleftrightarrow (\exists n < length\ xs. P\ (xs\ !\ n))$
 $\langle proof \rangle$

lemma *filtermap-conv*:
 $filtermap\ f\ xs = map\ (\lambda x. the\ (f\ x))\ (filter\ (\lambda x. f\ x \neq None)\ xs)$
 $\langle proof \rangle$

lemma *map-filter-conv* [simp]:
 $map\ filter\ f\ P\ xs = map\ f\ (filter\ P\ xs)$
 $\langle proof \rangle$

lemma *length-remdups-length-unique* [code inline]:
 $length\ (remdups\ xs) = length\ unique\ xs$

$\langle \text{proof} \rangle$

hide (open) *const length-unique*

Code for bounded quantification and summation over nats.

lemma *atMost-upto* [code unfold]:

$$\{..n\} = \text{set } [0..< \text{Suc } n]$$

$\langle \text{proof} \rangle$

lemma *atLeast-upt* [code unfold]:

$$\{..<n\} = \text{set } [0..<n]$$

$\langle \text{proof} \rangle$

lemma *greaterThanLessThan-upt* [code unfold]:

$$\{n<..$$

$\langle \text{proof} \rangle$

lemma *atLeastLessThan-upt* [code unfold]:

$$\{n..$$

$\langle \text{proof} \rangle$

lemma *greaterThanAtMost-upt* [code unfold]:

$$\{n<..$$

$\langle \text{proof} \rangle$

lemma *atLeastAtMost-upt* [code unfold]:

$$\{n..$$

$\langle \text{proof} \rangle$

lemma *all-nat-less-eq* [code unfold]:

$$(\forall m<n::\text{nat}. P\ m) \longleftrightarrow (\forall m \in \{0..$$

$\langle \text{proof} \rangle$

lemma *ex-nat-less-eq* [code unfold]:

$$(\exists m<n::\text{nat}. P\ m) \longleftrightarrow (\exists m \in \{0..$$

$\langle \text{proof} \rangle$

lemma *all-nat-less* [code unfold]:

$$(\forall m\leq n::\text{nat}. P\ m) \longleftrightarrow (\forall m \in \{0..n\}. P\ m)$$

$\langle \text{proof} \rangle$

lemma *ex-nat-less* [code unfold]:

$$(\exists m\leq n::\text{nat}. P\ m) \longleftrightarrow (\exists m \in \{0..n\}. P\ m)$$

$\langle \text{proof} \rangle$

lemma *setsum-set-distinct-conv-listsum*:

$$\text{distinct } xs \implies \text{setsum } f\ (\text{set } xs) = \text{listsum } (\text{map } f\ xs)$$

$\langle \text{proof} \rangle$

lemma *setsum-set-upt-conv-listsum* [code unfold]:
 $\text{setsum } f (\text{set } [m..<n]) = \text{listsum } (\text{map } f [m..<n])$
 ⟨proof⟩

Code for summation over ints.

lemma *greaterThanLessThan-upto* [code unfold]:
 $\{i < .. < j :: \text{int}\} = \text{set } [i+1..j - 1]$
 ⟨proof⟩

lemma *atLeastLessThan-upto* [code unfold]:
 $\{i..<j :: \text{int}\} = \text{set } [i..j - 1]$
 ⟨proof⟩

lemma *greaterThanAtMost-upto* [code unfold]:
 $\{i < .. j :: \text{int}\} = \text{set } [i+1..j]$
 ⟨proof⟩

lemma *atLeastAtMost-upto* [code unfold]:
 $\{i..j :: \text{int}\} = \text{set } [i..j]$
 ⟨proof⟩

lemma *setsum-set-upto-conv-listsum* [code unfold]:
 $\text{setsum } f (\text{set } [i..j :: \text{int}]) = \text{listsum } (\text{map } f [i..j])$
 ⟨proof⟩

end

40 Code-Message: Monolithic strings (message strings) for code generation

theory *Code-Message*
imports *Plain* $\sim\sim$ /src/HOL/List
begin

40.1 Datatype of messages

datatype *message-string* = STR *string*

lemmas [code del] = *message-string.recs message-string.cases*

lemma [code]: *size* (*s*::*message-string*) = 0
 ⟨proof⟩

lemma [code]: *message-string-size* (*s*::*message-string*) = 0
 ⟨proof⟩

40.2 ML interface

⟨ML⟩

40.3 Code serialization

code-type *message-string*

(*SML string*)

(*OCaml string*)

(*Haskell String*)

⟨ML⟩

code-reserved *SML string*

code-reserved *OCaml string*

code-instance *message-string* :: *eq*

(*Haskell -*)

code-const *eq-class.eq* :: *message-string* ⇒ *message-string* ⇒ *bool*

(*SML* !((- : *string*) = -))

(*OCaml* !((- : *string*) = -))

(*Haskell infixl 4 ==*)

end

41 Typerep: Reflecting Pure types into HOL

theory *Typerep*

imports *Plain List Code-Message*

begin

datatype *typerep* = *Typerep message-string typerep list*

class *typerep* =

fixes *typerep* :: 'a::{} *itself* ⇒ *typerep*

begin

definition *typerep-of* :: 'a ⇒ *typerep* **where**

[*simp*]: *typerep-of* *x* = *typerep TYPE('a)*

end

⟨ML⟩

lemma [*code*]:

eq-class.eq (*Typerep tyco1 tys1*) (*Typerep tyco2 tys2*) ⟷ *eq-class.eq tyco1 tyco2*
 ∧ *list-all2 eq-class.eq tys1 tys2*

```

  <proof>

code-type typerep
  (SML Term.typ)

code-const Typerep
  (SML Term.Type / (-, -))

code-reserved SML Term

hide (open) const typerep Typerep

end

```

42 Code-Eval: Term evaluation using the generic code generator

```

theory Code-Eval
imports Plain Typerep
begin

```

42.1 Term representation

42.1.1 Terms and class *term-of*

```

datatype term = dummy-term

```

```

definition Const :: message-string  $\Rightarrow$  typerep  $\Rightarrow$  term where
  Const - - = dummy-term

```

```

definition App :: term  $\Rightarrow$  term  $\Rightarrow$  term where
  App - - = dummy-term

```

```

code-datatype Const App

```

```

class term-of = typerep +
  fixes term-of :: 'a::{}  $\Rightarrow$  term

```

```

lemma term-of-anything: term-of x  $\equiv$  t
  <proof>

```

<ML>

42.1.2 *term-of* instances

<ML>

42.1.3 Code generator setup

lemmas [code del] = term.recs term.cases term.size

lemma [code, code del]: eq-class.eq (t1::term) t2 \longleftrightarrow eq-class.eq t1 t2 \langle proof \rangle

lemma [code, code del]: (term-of :: typerep \Rightarrow term) = term-of \langle proof \rangle

lemma [code, code del]: (term-of :: term \Rightarrow term) = term-of \langle proof \rangle

lemma [code, code del]: (term-of :: message-string \Rightarrow term) = term-of \langle proof \rangle

lemma [code, code del]:

(Code-Eval.term-of :: 'a::{type, term-of} Predicate.pred \Rightarrow Code-Eval.term) = Code-Eval.term-of \langle proof \rangle

lemma [code, code del]:

(Code-Eval.term-of :: 'a::{type, term-of} Predicate.seq \Rightarrow Code-Eval.term) = Code-Eval.term-of \langle proof \rangle

lemma term-of-char [unfolded typerep-fun-def typerep-char-def typerep-nibble-def, code]: Code-Eval.term-of c =

(let (n, m) = nibble-pair-of-char c
in Code-Eval.App (Code-Eval.App (Code-Eval.Const (STR "Pair") (TYPEREP(nibble
 \Rightarrow nibble \Rightarrow char))))
(Code-Eval.term-of n)) (Code-Eval.term-of m))
 \langle proof \rangle)

code-type term

(SML Term.term)

code-const Const and App

(SML Term.Const/ (-, -) and Term.\$/ (-, -))

code-const term-of :: message-string \Rightarrow term

(SML Message'-String.mk)

42.2 Evaluation setup

\langle ML \rangle

42.2.1 Syntax

\langle ML \rangle

hide const dummy-term

hide (open) const Const App

hide (open) const term-of

end

43 Map: Maps

theory Map

imports *List*
begin

types (*'a, 'b*) $\sim=> = 'a => 'b$ *option* (**infixr** 0)
translations (*type*) $a \sim=> b <= (type) a => b$ *option*

syntax (*xsymbols*)
 $\sim=> :: [type, type] => type$ (**infixr** \rightarrow 0)

abbreviation
 $empty :: 'a \sim=> 'b$ **where**
 $empty == \%x. None$

definition
 $map-comp :: ('b \sim=> 'c) => ('a \sim=> 'b) => ('a \sim=> 'c)$ (**infixl** *o'-m* 55)
where
 $f\ o\text{-}m\ g = (\lambda k. \text{case } g\ k \text{ of } None \Rightarrow None \mid Some\ v \Rightarrow f\ v)$

notation (*xsymbols*)
 $map-comp$ (**infixl** \circ_m 55)

definition
 $map-add :: ('a \sim=> 'b) => ('a \sim=> 'b) => ('a \sim=> 'b)$ (**infixl** $++$ 100)
where
 $m1\ ++\ m2 = (\lambda x. \text{case } m2\ x \text{ of } None \Rightarrow m1\ x \mid Some\ y \Rightarrow Some\ y)$

definition
 $restrict-map :: ('a \sim=> 'b) => 'a\ set => ('a \sim=> 'b)$ (**infixl** $|'$ 110) **where**
 $m|'A = (\lambda x. \text{if } x : A \text{ then } m\ x \text{ else } None)$

notation (*latex output*)
 $restrict-map$ ($-|'$ $[111,110]$ 110)

definition
 $dom :: ('a \sim=> 'b) => 'a\ set$ **where**
 $dom\ m = \{a. m\ a \neq None\}$

definition
 $ran :: ('a \sim=> 'b) => 'b\ set$ **where**
 $ran\ m = \{b. \exists a. m\ a = Some\ b\}$

definition
 $map-le :: ('a \sim=> 'b) => ('a \sim=> 'b) => bool$ (**infix** \subseteq_m 50) **where**
 $(m_1 \subseteq_m m_2) = (\forall a \in dom\ m_1. m_1\ a = m_2\ a)$

consts
 $map-of :: ('a * 'b) list => 'a \sim=> 'b$
 $map-upds :: ('a \sim=> 'b) => 'a\ list => 'b\ list => ('a \sim=> 'b)$

nonterminals*maplets maplet***syntax**

$-maplet :: ['a, 'a] \Rightarrow maplet \quad (- \text{ /| } \rightarrow \text{ / } -)$
 $-maplets :: ['a, 'a] \Rightarrow maplet \quad (- \text{ /| } \rightarrow \text{ / } -)$
 $\quad \quad \quad :: maplet \Rightarrow maplets \quad (-)$
 $-Maplets :: [maplet, maplets] \Rightarrow maplets \quad (-, \text{ / } -)$
 $-MapUpd :: ['a \rightsquigarrow 'b, maplets] \Rightarrow 'a \rightsquigarrow 'b \quad (-/'(-) [900, 0] 900)$
 $-Map \quad \quad :: maplets \Rightarrow 'a \rightsquigarrow 'b \quad ((1[-]))$

syntax (*xsymbols*)

$-maplet :: ['a, 'a] \Rightarrow maplet \quad (- \text{ / } \mapsto \text{ / } -)$
 $-maplets :: ['a, 'a] \Rightarrow maplet \quad (- \text{ / } \mapsto \text{ / } -)$

translations

$-MapUpd \ m \ (-Maplets \ xy \ ms) == -MapUpd \ (-MapUpd \ m \ xy) \ ms$
 $-MapUpd \ m \ (-maplet \ x \ y) == m(x := Some \ y)$
 $-MapUpd \ m \ (-maplets \ x \ y) == map-upds \ m \ x \ y$
 $-Map \ ms == -MapUpd \ (CONST \ empty) \ ms$
 $-Map \ (-Maplets \ ms1 \ ms2) \leq -MapUpd \ (-Map \ ms1) \ ms2$
 $-Maplets \ ms1 \ (-Maplets \ ms2 \ ms3) \leq -Maplets \ (-Maplets \ ms1 \ ms2) \ ms3$

primrec

$map\text{-}of \ [] = empty$
 $map\text{-}of \ (p \# ps) = (map\text{-}of \ ps)(fst \ p \mid \rightarrow \ snd \ p)$

declare *map-of.simps* [code del]**lemma** *map-of-Cons-code* [code]:

$map\text{-}of \ [] \ k = None$
 $map\text{-}of \ ((l, v) \# ps) \ k = (if \ l = k \ then \ Some \ v \ else \ map\text{-}of \ ps \ k)$
 $\langle proof \rangle$

defs

map-upds-def [code]: $m(xs \mid \rightarrow \ ys) == m \ ++ \ map\text{-}of \ (rev(zip \ xs \ ys))$

43.1 *empty*

lemma *empty-upd-none* [simp]: $empty(x := None) = empty$
 $\langle proof \rangle$

43.2 *map-upd*

lemma *map-upd-triv*: $t \ k = Some \ x \implies t(k \mid \rightarrow \ x) = t$
 $\langle proof \rangle$

lemma *map-upd-nonempty* [simp]: $t(k \mid \rightarrow \ x) \rightsquigarrow empty$
 $\langle proof \rangle$

lemma *map-upd-eqD1*:
assumes $m(a \mapsto x) = n(a \mapsto y)$
shows $x = y$
 $\langle \text{proof} \rangle$

lemma *map-upd-Some-unfold*:
 $((m(a | \rightarrow b))\ x = \text{Some } y) = (x = a \wedge b = y \vee x \neq a \wedge m\ x = \text{Some } y)$
 $\langle \text{proof} \rangle$

lemma *image-map-upd [simp]*: $x \notin A \implies m(x \mapsto y) \text{ ` } A = m \text{ ` } A$
 $\langle \text{proof} \rangle$

lemma *finite-range-updI*: $\text{finite } (\text{range } f) \implies \text{finite } (\text{range } (f(a | \rightarrow b)))$
 $\langle \text{proof} \rangle$

43.3 map-of

lemma *map-of-eq-None-iff*:
 $(\text{map-of } xys\ x = \text{None}) = (x \notin \text{fst ` } (\text{set } xys))$
 $\langle \text{proof} \rangle$

lemma *map-of-is-SomeD*: $\text{map-of } xys\ x = \text{Some } y \implies (x, y) \in \text{set } xys$
 $\langle \text{proof} \rangle$

lemma *map-of-eq-Some-iff [simp]*:
 $\text{distinct}(\text{map fst } xys) \implies (\text{map-of } xys\ x = \text{Some } y) = ((x, y) \in \text{set } xys)$
 $\langle \text{proof} \rangle$

lemma *Some-eq-map-of-iff [simp]*:
 $\text{distinct}(\text{map fst } xys) \implies (\text{Some } y = \text{map-of } xys\ x) = ((x, y) \in \text{set } xys)$
 $\langle \text{proof} \rangle$

lemma *map-of-is-SomeI [simp]*: $\llbracket \text{distinct}(\text{map fst } xys); (x, y) \in \text{set } xys \rrbracket$
 $\implies \text{map-of } xys\ x = \text{Some } y$
 $\langle \text{proof} \rangle$

lemma *map-of-zip-is-None [simp]*:
 $\text{length } xs = \text{length } ys \implies (\text{map-of } (\text{zip } xs\ ys)\ x = \text{None}) = (x \notin \text{set } xs)$
 $\langle \text{proof} \rangle$

lemma *map-of-zip-is-Some*:
assumes $\text{length } xs = \text{length } ys$
shows $x \in \text{set } xs \longleftrightarrow (\exists y. \text{map-of } (\text{zip } xs\ ys)\ x = \text{Some } y)$
 $\langle \text{proof} \rangle$

lemma *map-of-zip-upd*:
fixes $x :: 'a$ **and** $xs :: 'a \text{ list}$ **and** $ys\ zs :: 'b \text{ list}$
assumes $\text{length } ys = \text{length } xs$
and $\text{length } zs = \text{length } xs$

and $x \notin \text{set } xs$
and $\text{map-of } (\text{zip } xs \text{ } ys)(x \mapsto y) = \text{map-of } (\text{zip } xs \text{ } zs)(x \mapsto z)$
shows $\text{map-of } (\text{zip } xs \text{ } ys) = \text{map-of } (\text{zip } xs \text{ } zs)$
 $\langle \text{proof} \rangle$

lemma *map-of-zip-inject*:
assumes $\text{length } ys = \text{length } xs$
and $\text{length } zs = \text{length } xs$
and $\text{dist: distinct } xs$
and $\text{map-of: map-of } (\text{zip } xs \text{ } ys) = \text{map-of } (\text{zip } xs \text{ } zs)$
shows $ys = zs$
 $\langle \text{proof} \rangle$

lemma *finite-range-map-of*: $\text{finite } (\text{range } (\text{map-of } xys))$
 $\langle \text{proof} \rangle$

lemma *map-of-SomeD*: $\text{map-of } xs \text{ } k = \text{Some } y \implies (k, y) \in \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *map-of-mapk-SomeI*:
 $\text{inj } f \implies \text{map-of } t \text{ } k = \text{Some } x \implies$
 $\text{map-of } (\text{map } (\text{split } (\%k. \text{Pair } (f \text{ } k)))) \text{ } t \text{ } (f \text{ } k) = \text{Some } x$
 $\langle \text{proof} \rangle$

lemma *weak-map-of-SomeI*: $(k, x) : \text{set } l \implies \exists x. \text{map-of } l \text{ } k = \text{Some } x$
 $\langle \text{proof} \rangle$

lemma *map-of-filter-in*:
 $\text{map-of } xs \text{ } k = \text{Some } z \implies P \text{ } k \text{ } z \implies \text{map-of } (\text{filter } (\text{split } P) \text{ } xs) \text{ } k = \text{Some } z$
 $\langle \text{proof} \rangle$

lemma *map-of-map*: $\text{map-of } (\text{map } (\%(a,b). (a,f \text{ } b)) \text{ } xs) \text{ } x = \text{Option.map } f \text{ } (\text{map-of } xs \text{ } x)$
 $\langle \text{proof} \rangle$

43.4 Option.map related

lemma *option-map-o-empty* [simp]: $\text{Option.map } f \text{ } o \text{ empty} = \text{empty}$
 $\langle \text{proof} \rangle$

lemma *option-map-o-map-upd* [simp]:
 $\text{Option.map } f \text{ } o \text{ } m(a|->b) = (\text{Option.map } f \text{ } o \text{ } m)(a|->f \text{ } b)$
 $\langle \text{proof} \rangle$

43.5 map-comp related

lemma *map-comp-empty* [simp]:
 $m \circ_m \text{empty} = \text{empty}$
 $\text{empty} \circ_m m = \text{empty}$
 $\langle \text{proof} \rangle$

lemma *map-comp-simps* [*simp*]:

$$m2\ k = \text{None} \implies (m1\ \circ_m\ m2)\ k = \text{None}$$

$$m2\ k = \text{Some } k' \implies (m1\ \circ_m\ m2)\ k = m1\ k'$$

<proof>

lemma *map-comp-Some-iff*:

$$((m1\ \circ_m\ m2)\ k = \text{Some } v) = (\exists k'.\ m2\ k = \text{Some } k' \wedge m1\ k' = \text{Some } v)$$

<proof>

lemma *map-comp-None-iff*:

$$((m1\ \circ_m\ m2)\ k = \text{None}) = (m2\ k = \text{None} \vee (\exists k'.\ m2\ k = \text{Some } k' \wedge m1\ k' = \text{None}))$$

<proof>

43.6 ++

lemma *map-add-empty*[*simp*]: $m\ ++\ \text{empty} = m$

<proof>

lemma *empty-map-add*[*simp*]: $\text{empty}\ ++\ m = m$

<proof>

lemma *map-add-assoc*[*simp*]: $m1\ ++\ (m2\ ++\ m3) = (m1\ ++\ m2)\ ++\ m3$

<proof>

lemma *map-add-Some-iff*:

$$((m\ ++\ n)\ k = \text{Some } x) = (n\ k = \text{Some } x \mid n\ k = \text{None} \ \&\ m\ k = \text{Some } x)$$

<proof>

lemma *map-add-SomeD* [*dest!*]:

$$(m\ ++\ n)\ k = \text{Some } x \implies n\ k = \text{Some } x \vee n\ k = \text{None} \wedge m\ k = \text{Some } x$$

<proof>

lemma *map-add-find-right* [*simp*]: $!!xx.\ n\ k = \text{Some } xx \implies (m\ ++\ n)\ k = \text{Some } xx$

<proof>

lemma *map-add-None* [*iff*]: $((m\ ++\ n)\ k = \text{None}) = (n\ k = \text{None} \ \&\ m\ k = \text{None})$

<proof>

lemma *map-add-upd*[*simp*]: $f\ ++\ g(x|->y) = (f\ ++\ g)(x|->y)$

<proof>

lemma *map-add-upds*[*simp*]: $m1\ ++\ (m2(xs[\mapsto]ys)) = (m1\ ++\ m2)(xs[\mapsto]ys)$

<proof>

lemma *map-of-append*[*simp*]: $\text{map-of } (xs\ @\ ys) = \text{map-of } ys\ ++\ \text{map-of } xs$

$\langle \text{proof} \rangle$

lemma *finite-range-map-of-map-add*:

$\text{finite } (\text{range } f) \implies \text{finite } (\text{range } (f \text{ ++ map-of } l))$
 $\langle \text{proof} \rangle$

lemma *inj-on-map-add-dom [iff]*:

$\text{inj-on } (m \text{ ++ } m') (\text{dom } m') = \text{inj-on } m' (\text{dom } m')$
 $\langle \text{proof} \rangle$

43.7 restrict-map

lemma *restrict-map-to-empty [simp]*: $m|'\{\} = \text{empty}$
 $\langle \text{proof} \rangle$

lemma *restrict-map-empty [simp]*: $\text{empty}|'D = \text{empty}$
 $\langle \text{proof} \rangle$

lemma *restrict-in [simp]*: $x \in A \implies (m|'A) x = m x$
 $\langle \text{proof} \rangle$

lemma *restrict-out [simp]*: $x \notin A \implies (m|'A) x = \text{None}$
 $\langle \text{proof} \rangle$

lemma *ran-restrictD*: $y \in \text{ran } (m|'A) \implies \exists x \in A. m x = \text{Some } y$
 $\langle \text{proof} \rangle$

lemma *dom-restrict [simp]*: $\text{dom } (m|'A) = \text{dom } m \cap A$
 $\langle \text{proof} \rangle$

lemma *restrict-upd-same [simp]*: $m(x \mapsto y)|'(-\{x\}) = m|'(-\{x\})$
 $\langle \text{proof} \rangle$

lemma *restrict-restrict [simp]*: $m|'A|'B = m|'(A \cap B)$
 $\langle \text{proof} \rangle$

lemma *restrict-fun-upd [simp]*:

$m(x := y)|'D = (\text{if } x \in D \text{ then } (m|'(D - \{x\}))(x := y) \text{ else } m|'D)$
 $\langle \text{proof} \rangle$

lemma *fun-upd-None-restrict [simp]*:

$(m|'D)(x := \text{None}) = (\text{if } x:D \text{ then } m|'(D - \{x\}) \text{ else } m|'D)$
 $\langle \text{proof} \rangle$

lemma *fun-upd-restrict*: $(m|'D)(x := y) = (m|'(D - \{x\}))(x := y)$
 $\langle \text{proof} \rangle$

lemma *fun-upd-restrict-conv [simp]*:

$x \in D \implies (m|'D)(x := y) = (m|'(D - \{x\}))(x := y)$

$\langle proof \rangle$

43.8 map-upds

lemma *map-upds-Nil1* [simp]: $m([], [] \multimap bs) = m$
 $\langle proof \rangle$

lemma *map-upds-Nil2* [simp]: $m(as [] \multimap []) = m$
 $\langle proof \rangle$

lemma *map-upds-Cons* [simp]: $m(a \# as [] \multimap b \# bs) = (m(a | \multimap b))(as [] \multimap bs)$
 $\langle proof \rangle$

lemma *map-upds-append1* [simp]: $\bigwedge ys m. size\ xs < size\ ys \implies$
 $m(xs @ [x] [] \multimap ys) = m(xs [] \multimap ys)(x \mapsto ys!size\ xs)$
 $\langle proof \rangle$

lemma *map-upds-list-update2-drop* [simp]:
 $\llbracket size\ xs \leq i; i < size\ ys \rrbracket$
 $\implies m(xs [] \multimap ys[i := y]) = m(xs [] \multimap ys)$
 $\langle proof \rangle$

lemma *map-upd-upds-conv-if*:
 $(f(x | \multimap y))(xs [] \multimap ys) =$
 $(if\ x : set\ (take\ (length\ ys)\ xs)\ then\ f(xs [] \multimap ys)$
 $\quad\quad\quad else\ (f(xs [] \multimap ys))(x | \multimap y))$
 $\langle proof \rangle$

lemma *map-upds-twist* [simp]:
 $a \sim : set\ as \implies m(a | \multimap b)(as [] \multimap bs) = m(as [] \multimap bs)(a | \multimap b)$
 $\langle proof \rangle$

lemma *map-upds-apply-nontin* [simp]:
 $x \sim : set\ xs \implies (f(xs [] \multimap ys))\ x = f\ x$
 $\langle proof \rangle$

lemma *fun-upds-append-drop* [simp]:
 $size\ xs = size\ ys \implies m(xs @ zs [] \multimap ys) = m(xs [] \multimap ys)$
 $\langle proof \rangle$

lemma *fun-upds-append2-drop* [simp]:
 $size\ xs = size\ ys \implies m(xs [] \multimap ys @ zs) = m(xs [] \multimap ys)$
 $\langle proof \rangle$

lemma *restrict-map-upds* [simp]:
 $\llbracket length\ xs = length\ ys; set\ xs \subseteq D \rrbracket$
 $\implies m(xs [] \multimap ys) | 'D = (m | ' (D - set\ xs))(xs [] \multimap ys)$
 $\langle proof \rangle$

43.9 *dom*

lemma *domI*: $m\ a = \text{Some } b \implies a : \text{dom } m$
 $\langle \text{proof} \rangle$

lemma *domD*: $a : \text{dom } m \implies \exists b. m\ a = \text{Some } b$
 $\langle \text{proof} \rangle$

lemma *domIff* [*iff*, *simp del*]: $(a : \text{dom } m) = (m\ a \sim = \text{None})$
 $\langle \text{proof} \rangle$

lemma *dom-empty* [*simp*]: $\text{dom empty} = \{\}$
 $\langle \text{proof} \rangle$

lemma *dom-fun-upd* [*simp*]:
 $\text{dom}(f(x := y)) = (\text{if } y = \text{None} \text{ then } \text{dom } f - \{x\} \text{ else } \text{insert } x (\text{dom } f))$
 $\langle \text{proof} \rangle$

lemma *dom-map-of*: $\text{dom}(\text{map-of } xys) = \{x. \exists y. (x, y) : \text{set } xys\}$
 $\langle \text{proof} \rangle$

lemma *dom-map-of-conv-image-fst*:
 $\text{dom}(\text{map-of } xys) = \text{fst} \text{ ` } (\text{set } xys)$
 $\langle \text{proof} \rangle$

lemma *dom-map-of-zip* [*simp*]: $[\text{length } xs = \text{length } ys; \text{distinct } xs] \implies$
 $\text{dom}(\text{map-of}(\text{zip } xs\ ys)) = \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *finite-dom-map-of*: $\text{finite } (\text{dom } (\text{map-of } l))$
 $\langle \text{proof} \rangle$

lemma *dom-map-upds* [*simp*]:
 $\text{dom}(m(xs[->]ys)) = \text{set}(\text{take } (\text{length } ys) \ xs) \ \text{Un } \text{dom } m$
 $\langle \text{proof} \rangle$

lemma *dom-map-add* [*simp*]: $\text{dom}(m++n) = \text{dom } n \ \text{Un } \text{dom } m$
 $\langle \text{proof} \rangle$

lemma *dom-override-on* [*simp*]:
 $\text{dom}(\text{override-on } f\ g\ A) =$
 $(\text{dom } f - \{a. a : A - \text{dom } g\}) \ \text{Un } \{a. a : A \ \text{Int } \text{dom } g\}$
 $\langle \text{proof} \rangle$

lemma *map-add-comm*: $\text{dom } m1 \cap \text{dom } m2 = \{\} \implies m1++m2 = m2++m1$
 $\langle \text{proof} \rangle$

lemma *dom-const* [*simp*]:
 $\text{dom } (\lambda x. \text{Some } y) = \text{UNIV}$

$\langle \text{proof} \rangle$

lemma *dom-if*:

$$\text{dom } (\lambda x. \text{ if } P \ x \text{ then } f \ x \text{ else } g \ x) = \text{dom } f \cap \{x. P \ x\} \cup \text{dom } g \cap \{x. \neg P \ x\}$$

$\langle \text{proof} \rangle$

lemma *finite-map-freshness*:

$$\text{finite } (\text{dom } (f :: 'a \rightarrow 'b)) \implies \neg \text{finite } (\text{UNIV} :: 'a \text{ set}) \implies \\ \exists x. f \ x = \text{None}$$

$\langle \text{proof} \rangle$

lemma *dom-minus*:

$$f \ x = \text{None} \implies \text{dom } f - \text{insert } x \ A = \text{dom } f - A$$

$\langle \text{proof} \rangle$

lemma *insert-dom*:

$$f \ x = \text{Some } y \implies \text{insert } x \ (\text{dom } f) = \text{dom } f$$

$\langle \text{proof} \rangle$

43.10 *ran*

lemma *ranI*: $m \ a = \text{Some } b \implies b : \text{ran } m$

$\langle \text{proof} \rangle$

lemma *ran-empty* [simp]: $\text{ran } \text{empty} = \{\}$

$\langle \text{proof} \rangle$

lemma *ran-map-upd* [simp]: $m \ a = \text{None} \implies \text{ran}(m(a|->b)) = \text{insert } b \ (\text{ran } m)$

$\langle \text{proof} \rangle$

43.11 *map-le*

lemma *map-le-empty* [simp]: $\text{empty} \subseteq_m g$

$\langle \text{proof} \rangle$

lemma *upd-None-map-le* [simp]: $f(x := \text{None}) \subseteq_m f$

$\langle \text{proof} \rangle$

lemma *map-le-upd*[simp]: $f \subseteq_m g \implies f(a := b) \subseteq_m g(a := b)$

$\langle \text{proof} \rangle$

lemma *map-le-imp-upd-le* [simp]: $m1 \subseteq_m m2 \implies m1(x := \text{None}) \subseteq_m m2(x \mapsto y)$

$\langle \text{proof} \rangle$

lemma *map-le-upds* [simp]:

$f \subseteq_m g \implies f(as \ [|->] \ bs) \subseteq_m g(as \ [|->] \ bs)$
 $\langle proof \rangle$

lemma *map-le-implies-dom-le*: $(f \subseteq_m g) \implies (dom\ f \subseteq dom\ g)$
 $\langle proof \rangle$

lemma *map-le-refl* [*simp*]: $f \subseteq_m f$
 $\langle proof \rangle$

lemma *map-le-trans*[*trans*]: $\llbracket m1 \subseteq_m m2; m2 \subseteq_m m3 \rrbracket \implies m1 \subseteq_m m3$
 $\langle proof \rangle$

lemma *map-le-antisym*: $\llbracket f \subseteq_m g; g \subseteq_m f \rrbracket \implies f = g$
 $\langle proof \rangle$

lemma *map-le-map-add* [*simp*]: $f \subseteq_m (g ++ f)$
 $\langle proof \rangle$

lemma *map-le-iff-map-add-commute*: $(f \subseteq_m f ++ g) = (f ++ g = g ++ f)$
 $\langle proof \rangle$

lemma *map-add-le-mapE*: $f ++ g \subseteq_m h \implies g \subseteq_m h$
 $\langle proof \rangle$

lemma *map-add-le-mapI*: $\llbracket f \subseteq_m h; g \subseteq_m h; f \subseteq_m f ++ g \rrbracket \implies f ++ g \subseteq_m h$
 $\langle proof \rangle$

end

44 Refute: Refute

theory *Refute*

imports *Hilbert-Choice List Record*

uses

Tools/prop-logic.ML

Tools/sat-solver.ML

Tools/refute.ML

Tools/refute-isar.ML

begin

$\langle ML \rangle$

```
(* ----- *)
(* REFUTE *)
(*
(* We use a SAT solver to search for a (finite) model that refutes a given
(* HOL formula. *)
```

```

(* ----- *)

(* ----- *)
(* NOTE *)
(* ----- *)
(* I strongly recommend that you install a stand-alone SAT solver if you *)
(* want to use 'refute'. For details see 'HOL/Tools/sat_solver.ML'. If you *)
(* have installed (a supported version of) zChaff, simply set 'ZCHAFF_HOME' *)
(* in 'etc/settings'. *)
(* ----- *)

(* ----- *)
(* USAGE *)
(* ----- *)
(* See the file 'HOL/ex/Refute_Examples.thy' for examples. The supported *)
(* parameters are explained below. *)
(* ----- *)

(* ----- *)
(* CURRENT LIMITATIONS *)
(* ----- *)
(* 'refute' currently accepts formulas of higher-order predicate logic (with *)
(* equality), including free/bound/schematic variables, lambda abstractions, *)
(* sets and set membership, "arbitrary", "The", "Eps", records and *)
(* inductively defined sets. Constants are unfolded automatically, and sort *)
(* axioms are added as well. Other, user-asserted axioms however are *)
(* ignored. Inductive datatypes and recursive functions are supported, but *)
(* may lead to spurious countermodels. *)
(* ----- *)
(* The (space) complexity of the algorithm is non-elementary. *)
(* ----- *)
(* Schematic type variables are not supported. *)
(* ----- *)

(* ----- *)
(* PARAMETERS *)
(* ----- *)
(* The following global parameters are currently supported (and required): *)
(* ----- *)
(* Name          Type      Description *)
(* ----- *)
(* "minsize"      int       Only search for models with size at least *)
(*                  'minsize'. *)
(* "maxsize"      int       If >0, only search for models with size at most *)
(*                  'maxsize'. *)
(* "maxvars"      int       If >0, use at most 'maxvars' boolean variables *)
(*                  when transforming the term into a propositional *)
(*                  formula. *)
(* "maxtime"      int       If >0, terminate after at most 'maxtime' seconds. *)

```



```

(*)          This value is ignored under some ML compilers.      *)
(*) "satsolver"  string  Name of the SAT solver to be used.      *)
(*)                                                    *)
(*) See 'HOL/SAT.thy' for default values.                      *)
(*)                                                    *)
(*) The size of particular types can be specified in the form type=size *)
(*) (where 'type' is a string, and 'size' is an int).  Examples: *)
(*) "'a'=1                                             *)
(*) "List.list"=2                                       *)
(*) ----- *)

(*) ----- *)
(*) FILES                                             *)
(*)                                                    *)
(*) HOL/Tools/prop_logic.ML      Propositional logic      *)
(*) HOL/Tools/sat_solver.ML      SAT solvers              *)
(*) HOL/Tools/refute.ML          Translation HOL -> propositional logic and *)
(*)                               Boolean assignment -> HOL model *)
(*) HOL/Tools/refute_isar.ML      Adds 'refute'/'refute_params' to Isabelle's *)
(*)                               syntax *)
(*) HOL/Refute.thy                This file: loads the ML files, basic setup, *)
(*)                               documentation *)
(*) HOL/SAT.thy                   Sets default parameters *)
(*) HOL/ex/RefuteExamples.thy     Examples *)
(*) ----- *)

end

```

45 SAT: Reconstructing external resolution proofs for propositional logic

```

theory SAT
imports Refute
uses
  Tools/cnf-funcs.ML
  Tools/sat-funcs.ML
begin

```

Late package setup: default values for refute, see also theory *Refute*.

```

refute-params
[itself=1,
 minsize=1,
 maxsize=8,
 maxvars=10000,
 maxtime=60,
 satsolver=auto]

```

$\langle ML \rangle$

end

46 Main: Main HOL

theory *Main*
imports *Plain Code-Eval Map Recdef SAT*
begin

Classical Higher-order Logic – only “Main”, excluding real and complex numbers etc.

See further [1]

end

47 Lubs: Definitions of Upper Bounds and Least Upper Bounds

theory *Lubs*
imports *Main*
begin

Thanks to suggestions by James Margetson

definition
 $settle :: ['a\ set, 'a::ord] \Rightarrow bool$ (**infixl** $*\leq$ 70) **where**
 $S * \leq x = (ALL\ y: S. y \leq x)$

definition
 $setge :: ['a::ord, 'a\ set] \Rightarrow bool$ (**infixl** $\leq*$ 70) **where**
 $x \leq* S = (ALL\ y: S. x \leq y)$

definition
 $leastP :: ['a \Rightarrow bool, 'a::ord] \Rightarrow bool$ **where**
 $leastP\ P\ x = (P\ x \ \&\ x \leq* Collect\ P)$

definition
 $isUb :: ['a\ set, 'a\ set, 'a::ord] \Rightarrow bool$ **where**
 $isUb\ R\ S\ x = (S * \leq x \ \&\ x: R)$

definition
 $isLub :: ['a\ set, 'a\ set, 'a::ord] \Rightarrow bool$ **where**
 $isLub\ R\ S\ x = leastP\ (isUb\ R\ S)\ x$

definition

$ubs :: ['a\ set, 'a::ord\ set] \Rightarrow 'a\ set$ **where**
 $ubs\ R\ S = Collect\ (isUb\ R\ S)$

47.1 Rules for the Relations $*\leq$ and $\leq*$

lemma *settleI*: $ALL\ y: S. y \leq x \Rightarrow S * \leq x$
 $\langle proof \rangle$

lemma *settleD*: $[| S * \leq x; y: S |] \Rightarrow y \leq x$
 $\langle proof \rangle$

lemma *setgeI*: $ALL\ y: S. x \leq y \Rightarrow x \leq* S$
 $\langle proof \rangle$

lemma *setgeD*: $[| x \leq* S; y: S |] \Rightarrow x \leq y$
 $\langle proof \rangle$

47.2 Rules about the Operators *leastP*, *ub* and *lub*

lemma *leastPD1*: $leastP\ P\ x \Rightarrow P\ x$
 $\langle proof \rangle$

lemma *leastPD2*: $leastP\ P\ x \Rightarrow x \leq* Collect\ P$
 $\langle proof \rangle$

lemma *leastPD3*: $[| leastP\ P\ x; y: Collect\ P |] \Rightarrow x \leq y$
 $\langle proof \rangle$

lemma *isLubD1*: $isLub\ R\ S\ x \Rightarrow S * \leq x$
 $\langle proof \rangle$

lemma *isLubD1a*: $isLub\ R\ S\ x \Rightarrow x: R$
 $\langle proof \rangle$

lemma *isLub-isUb*: $isLub\ R\ S\ x \Rightarrow isUb\ R\ S\ x$
 $\langle proof \rangle$

lemma *isLubD2*: $[| isLub\ R\ S\ x; y: S |] \Rightarrow y \leq x$
 $\langle proof \rangle$

lemma *isLubD3*: $isLub\ R\ S\ x \Rightarrow leastP(isUb\ R\ S)\ x$
 $\langle proof \rangle$

lemma *isLubI1*: $leastP(isUb\ R\ S)\ x \Rightarrow isLub\ R\ S\ x$
 $\langle proof \rangle$

lemma *isLubI2*: $[| isUb\ R\ S\ x; x \leq* Collect\ (isUb\ R\ S) |] \Rightarrow isLub\ R\ S\ x$
 $\langle proof \rangle$

lemma *isUbd*: $[[\text{isUb } R \ S \ x; y : S] \implies y \leq x$
 $\langle \text{proof} \rangle$

lemma *isUbd2*: $\text{isUb } R \ S \ x \implies S \ * \leq x$
 $\langle \text{proof} \rangle$

lemma *isUbd2a*: $\text{isUb } R \ S \ x \implies x : R$
 $\langle \text{proof} \rangle$

lemma *isUbi*: $[[S \ * \leq x; x : R] \implies \text{isUb } R \ S \ x$
 $\langle \text{proof} \rangle$

lemma *isLub-le-isUb*: $[[\text{isLub } R \ S \ x; \text{isUb } R \ S \ y] \implies x \leq y$
 $\langle \text{proof} \rangle$

lemma *isLub-ubs*: $\text{isLub } R \ S \ x \implies x \leq * \text{ ubs } R \ S$
 $\langle \text{proof} \rangle$

end

48 GCD: The Greatest Common Divisor

theory *GCD*
imports *Main*
begin

See [?].

48.1 Specification of GCD on nats

definition

is-gcd :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where** — *gcd* as a relation
 $[\text{code del}]: \text{is-gcd } m \ n \ p \longleftrightarrow p \ \text{dvd } m \wedge p \ \text{dvd } n \wedge$
 $(\forall d. d \ \text{dvd } m \longrightarrow d \ \text{dvd } n \longrightarrow d \ \text{dvd } p)$

Uniqueness

lemma *is-gcd-unique*: $\text{is-gcd } a \ b \ m \implies \text{is-gcd } a \ b \ n \implies m = n$
 $\langle \text{proof} \rangle$

Connection to divides relation

lemma *is-gcd-dvd*: $\text{is-gcd } a \ b \ m \implies k \ \text{dvd } a \implies k \ \text{dvd } b \implies k \ \text{dvd } m$
 $\langle \text{proof} \rangle$

Commutativity

lemma *is-gcd-commute*: $\text{is-gcd } m \ n \ k = \text{is-gcd } n \ m \ k$
 $\langle \text{proof} \rangle$

48.2 GCD on nat by Euclid’s algorithm

```

fun
  gcd :: nat => nat => nat
where
  gcd m n = (if n = 0 then m else gcd n (m mod n))
lemma gcd-induct [case-names 0 rec]:
  fixes m n :: nat
  assumes  $\bigwedge m. P\ m\ 0$ 
  and  $\bigwedge m\ n. 0 < n \implies P\ n\ (m \bmod n) \implies P\ m\ n$ 
  shows  $P\ m\ n$ 
  <proof>

lemma gcd-0 [simp, algebra]: gcd m 0 = m
  <proof>

lemma gcd-0-left [simp, algebra]: gcd 0 m = m
  <proof>

lemma gcd-non-0:  $n > 0 \implies \text{gcd } m\ n = \text{gcd } n\ (m \bmod n)$ 
  <proof>

lemma gcd-1 [simp, algebra]: gcd m (Suc 0) = Suc 0
  <proof>

lemma nat-gcd-1-right [simp, algebra]: gcd m 1 = 1
  <proof>

declare gcd.simps [simp del]

```

$\text{gcd } m\ n$ divides m and n . The conjunctions don’t seem provable separately.

```

lemma gcd-dvd1 [iff, algebra]: gcd m n dvd m
  and gcd-dvd2 [iff, algebra]: gcd m n dvd n
  <proof>

```

Maximality: for all m, n, k naturals, if k divides m and k divides n then k divides $\text{gcd } m\ n$.

```

lemma gcd-greatest:  $k\ \text{dvd}\ m \implies k\ \text{dvd}\ n \implies k\ \text{dvd}\ \text{gcd } m\ n$ 
  <proof>

```

Function gcd yields the Greatest Common Divisor.

```

lemma is-gcd: is-gcd m n (gcd m n)
  <proof>

```

48.3 Derived laws for GCD

```

lemma gcd-greatest-iff [iff, algebra]:  $k\ \text{dvd}\ \text{gcd } m\ n \iff k\ \text{dvd}\ m \wedge k\ \text{dvd}\ n$ 
  <proof>

```

lemma *gcd-zero*[*algebra*]: $\text{gcd } m \ n = 0 \longleftrightarrow m = 0 \wedge n = 0$
 $\langle \text{proof} \rangle$

lemma *gcd-commute*: $\text{gcd } m \ n = \text{gcd } n \ m$
 $\langle \text{proof} \rangle$

lemma *gcd-assoc*: $\text{gcd } (\text{gcd } k \ m) \ n = \text{gcd } k \ (\text{gcd } m \ n)$
 $\langle \text{proof} \rangle$

lemma *gcd-1-left* [*simp*, *algebra*]: $\text{gcd } (\text{Suc } 0) \ m = \text{Suc } 0$
 $\langle \text{proof} \rangle$

lemma *nat-gcd-1-left* [*simp*, *algebra*]: $\text{gcd } 1 \ m = 1$
 $\langle \text{proof} \rangle$

Multiplication laws

lemma *gcd-mult-distrib2*: $k * \text{gcd } m \ n = \text{gcd } (k * m) \ (k * n)$
 — [?, page 27]
 $\langle \text{proof} \rangle$

lemma *gcd-mult* [*simp*, *algebra*]: $\text{gcd } k \ (k * n) = k$
 $\langle \text{proof} \rangle$

lemma *gcd-self* [*simp*, *algebra*]: $\text{gcd } k \ k = k$
 $\langle \text{proof} \rangle$

lemma *relprime-dvd-mult*: $\text{gcd } k \ n = 1 \implies k \ \text{dvd} \ m * n \implies k \ \text{dvd} \ m$
 $\langle \text{proof} \rangle$

lemma *relprime-dvd-mult-iff*: $\text{gcd } k \ n = 1 \implies (k \ \text{dvd} \ m * n) = (k \ \text{dvd} \ m)$
 $\langle \text{proof} \rangle$

lemma *gcd-mult-cancel*: $\text{gcd } k \ n = 1 \implies \text{gcd } (k * m) \ n = \text{gcd } m \ n$
 $\langle \text{proof} \rangle$

Addition laws

lemma *gcd-add1* [*simp*, *algebra*]: $\text{gcd } (m + n) \ n = \text{gcd } m \ n$
 $\langle \text{proof} \rangle$

lemma *gcd-add2* [*simp*, *algebra*]: $\text{gcd } m \ (m + n) = \text{gcd } m \ n$
 $\langle \text{proof} \rangle$

lemma *gcd-add2'* [*simp*, *algebra*]: $\text{gcd } m \ (n + m) = \text{gcd } m \ n$
 $\langle \text{proof} \rangle$

lemma *gcd-add-mult*[*algebra*]: $\text{gcd } m \ (k * m + n) = \text{gcd } m \ n$
 $\langle \text{proof} \rangle$

lemma *gcd-dvd-prod*: $\text{gcd } m \ n \ \text{dvd } m * n$
 $\langle \text{proof} \rangle$

Division by gcd yields relatively primes.

lemma *div-gcd-relprime*:
assumes $\text{nz}: a \neq 0 \vee b \neq 0$
shows $\text{gcd } (a \ \text{div } \text{gcd } a \ b) \ (b \ \text{div } \text{gcd } a \ b) = 1$
 $\langle \text{proof} \rangle$

lemma *gcd-unique*: $d \ \text{dvd } a \wedge d \ \text{dvd } b \wedge (\forall e. e \ \text{dvd } a \wedge e \ \text{dvd } b \longrightarrow e \ \text{dvd } d) \longleftrightarrow$
 $d = \text{gcd } a \ b$
 $\langle \text{proof} \rangle$

lemma *gcd-eq*: **assumes** $H: \forall d. d \ \text{dvd } x \wedge d \ \text{dvd } y \longleftrightarrow d \ \text{dvd } u \wedge d \ \text{dvd } v$
shows $\text{gcd } x \ y = \text{gcd } u \ v$
 $\langle \text{proof} \rangle$

lemma *ind-euclid*:
assumes $c: \forall a \ b. P \ (a::\text{nat}) \ b \longleftrightarrow P \ b \ a$ **and** $z: \forall a. P \ a \ 0$
and $\text{add}: \forall a \ b. P \ a \ b \longrightarrow P \ a \ (a + b)$
shows $P \ a \ b$
 $\langle \text{proof} \rangle$

lemma *bezout-lemma*:
assumes $\text{ex}: \exists (d::\text{nat}) \ x \ y. d \ \text{dvd } a \wedge d \ \text{dvd } b \wedge (a * x = b * y + d \vee b * x =$
 $a * y + d)$
shows $\exists d \ x \ y. d \ \text{dvd } a \wedge d \ \text{dvd } a + b \wedge (a * x = (a + b) * y + d \vee (a + b) *$
 $x = a * y + d)$
 $\langle \text{proof} \rangle$

lemma *bezout-add*: $\exists (d::\text{nat}) \ x \ y. d \ \text{dvd } a \wedge d \ \text{dvd } b \wedge (a * x = b * y + d \vee b * x =$
 $a * y + d)$
 $\langle \text{proof} \rangle$

lemma *bezout*: $\exists (d::\text{nat}) \ x \ y. d \ \text{dvd } a \wedge d \ \text{dvd } b \wedge (a * x - b * y = d \vee b * x -$
 $a * y = d)$
 $\langle \text{proof} \rangle$

We can get a stronger version with a nonzeroness assumption.

lemma *divides-le*: $m \ \text{dvd } n \implies m \leq n \vee n = (0::\text{nat})$ $\langle \text{proof} \rangle$

lemma *bezout-add-strong*: **assumes** $\text{nz}: a \neq (0::\text{nat})$
shows $\exists d \ x \ y. d \ \text{dvd } a \wedge d \ \text{dvd } b \wedge a * x = b * y + d$
 $\langle \text{proof} \rangle$

lemma *bezout-gcd*: $\exists x \ y. a * x - b * y = \text{gcd } a \ b \vee b * x - a * y = \text{gcd } a \ b$

$\langle proof \rangle$

lemma *bezout-gcd-strong*: **assumes** $a: a \neq 0$
shows $\exists x y. a * x = b * y + \gcd a b$
 $\langle proof \rangle$

lemma *gcd-mult-distrib*: $\gcd(a * c) (b * c) = c * \gcd a b$
 $\langle proof \rangle$

lemma *gcd-bezout*: $(\exists x y. a * x - b * y = d \vee b * x - a * y = d) \longleftrightarrow \gcd a b \mid d$
 $(\text{is } ?lhs \longleftrightarrow ?rhs)$
 $\langle proof \rangle$

lemma *gcd-bezout-sum*: **assumes** $H: a * x + b * y = d$ **shows** $\gcd a b \mid d$
 $\langle proof \rangle$

lemma *gcd-mult'*: $\gcd b (a * b) = b$
 $\langle proof \rangle$

lemma *gcd-add*: $\gcd(a + b) b = \gcd a b$
 $\gcd(b + a) b = \gcd a b \gcd a (a + b) = \gcd a b \gcd a (b + a) = \gcd a b$
 $\langle proof \rangle$

lemma *gcd-sub*: $b \leq a \implies \gcd(a - b) b = \gcd a b \wedge a \leq b \implies \gcd a (b - a) = \gcd a b$
 $\langle proof \rangle$

48.4 LCM defined by GCD

definition

$lcm :: nat \Rightarrow nat \Rightarrow nat$

where

lcm-def: $lcm\ m\ n = m * n \div \gcd m\ n$

lemma *prod-gcd-lcm*:
 $m * n = \gcd m\ n * lcm\ m\ n$
 $\langle proof \rangle$

lemma *lcm-0* [simp]: $lcm\ m\ 0 = 0$
 $\langle proof \rangle$

lemma *lcm-1* [simp]: $lcm\ m\ 1 = m$
 $\langle proof \rangle$

lemma *lcm-0-left* [simp]: $lcm\ 0\ n = 0$
 $\langle proof \rangle$

lemma *lcm-1-left* [simp]: $lcm\ 1\ m = m$

$\langle \text{proof} \rangle$

lemma *dvd-pos*:
 fixes $n\ m :: \text{nat}$
 assumes $n > 0$ and $m\ \text{dvd}\ n$
 shows $m > 0$
 $\langle \text{proof} \rangle$

lemma *lcm-least*:
 assumes $m\ \text{dvd}\ k$ and $n\ \text{dvd}\ k$
 shows $\text{lcm}\ m\ n\ \text{dvd}\ k$
 $\langle \text{proof} \rangle$

lemma *lcm-dvd1* [iff]:
 $m\ \text{dvd}\ \text{lcm}\ m\ n$
 $\langle \text{proof} \rangle$

lemma *lcm-dvd2* [iff]:
 $n\ \text{dvd}\ \text{lcm}\ m\ n$
 $\langle \text{proof} \rangle$

lemma *gcd-add1-eq*: $\text{gcd}\ (m + k)\ k = \text{gcd}\ (m + k)\ m$
 $\langle \text{proof} \rangle$

lemma *gcd-diff2*: $m \leq n \implies \text{gcd}\ n\ (n - m) = \text{gcd}\ n\ m$
 $\langle \text{proof} \rangle$

48.5 GCD and LCM on integers

definition
 $\text{zgcd} :: \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$ **where**
 $\text{zgcd}\ i\ j = \text{int}\ (\text{gcd}\ (\text{nat}\ (\text{abs}\ i))\ (\text{nat}\ (\text{abs}\ j)))$

lemma *zgcd-zdvd1* [iff, simp, algebra]: $\text{zgcd}\ i\ j\ \text{dvd}\ i$
 $\langle \text{proof} \rangle$

lemma *zgcd-zdvd2* [iff, simp, algebra]: $\text{zgcd}\ i\ j\ \text{dvd}\ j$
 $\langle \text{proof} \rangle$

lemma *zgcd-pos*: $\text{zgcd}\ i\ j \geq 0$
 $\langle \text{proof} \rangle$

lemma *zgcd0* [simp, algebra]: $(\text{zgcd}\ i\ j = 0) = (i = 0 \wedge j = 0)$
 $\langle \text{proof} \rangle$

lemma *zgcd-commute*: $\text{zgcd}\ i\ j = \text{zgcd}\ j\ i$
 $\langle \text{proof} \rangle$

lemma *zgcd-zminus* [simp, algebra]: $\text{zgcd}\ (-i)\ j = \text{zgcd}\ i\ j$

$\langle proof \rangle$

lemma *zgcd-zminus2* [*simp*, *algebra*]: $zgcd\ i\ (-j) = zgcd\ i\ j$
 $\langle proof \rangle$

lemma *zrelprime-dvd-mult*: $zgcd\ i\ j = 1 \implies i\ dvd\ k * j \implies i\ dvd\ k$
 $\langle proof \rangle$

lemma *int-nat-abs*: $int\ (nat\ (abs\ x)) = abs\ x$ $\langle proof \rangle$

lemma *zgcd-greatest*:
 assumes $k\ dvd\ m$ and $k\ dvd\ n$
 shows $k\ dvd\ zgcd\ m\ n$
 $\langle proof \rangle$

lemma *div-zgcd-relprime*:
 assumes $nz: a \neq 0 \vee b \neq 0$
 shows $zgcd\ (a\ div\ (zgcd\ a\ b))\ (b\ div\ (zgcd\ a\ b)) = 1$
 $\langle proof \rangle$

lemma *zgcd-0* [*simp*, *algebra*]: $zgcd\ m\ 0 = abs\ m$
 $\langle proof \rangle$

lemma *zgcd-0-left* [*simp*, *algebra*]: $zgcd\ 0\ m = abs\ m$
 $\langle proof \rangle$

lemma *zgcd-non-0*: $0 < n \implies zgcd\ m\ n = zgcd\ n\ (m\ mod\ n)$
 $\langle proof \rangle$

lemma *zgcd-eq*: $zgcd\ m\ n = zgcd\ n\ (m\ mod\ n)$
 $\langle proof \rangle$

lemma *zgcd-1* [*simp*, *algebra*]: $zgcd\ m\ 1 = 1$
 $\langle proof \rangle$

lemma *zgcd-0-1-iff* [*simp*, *algebra*]: $zgcd\ 0\ m = 1 \longleftrightarrow |m| = 1$
 $\langle proof \rangle$

lemma *zgcd-greatest-iff* [*algebra*]: $k\ dvd\ zgcd\ m\ n = (k\ dvd\ m \wedge k\ dvd\ n)$
 $\langle proof \rangle$

lemma *zgcd-1-left* [*simp*, *algebra*]: $zgcd\ 1\ m = 1$
 $\langle proof \rangle$

lemma *zgcd-assoc*: $zgcd\ (zgcd\ k\ m)\ n = zgcd\ k\ (zgcd\ m\ n)$
 $\langle proof \rangle$

lemma *zgcd-left-commute*: $zgcd\ k\ (zgcd\ m\ n) = zgcd\ m\ (zgcd\ k\ n)$

$\langle \text{proof} \rangle$

lemmas $\text{zgcd-ac} = \text{zgcd-assoc} \text{zgcd-commute} \text{zgcd-left-commute}$
 — addition is an AC-operator

lemma $\text{zgcd-zmult-distrib2}$: $0 \leq k \implies k * \text{zgcd } m \ n = \text{zgcd } (k * m) \ (k * n)$
 $\langle \text{proof} \rangle$

lemma $\text{zgcd-zmult-distrib2-abs}$: $\text{zgcd } (k * m) \ (k * n) = \text{abs } k * \text{zgcd } m \ n$
 $\langle \text{proof} \rangle$

lemma zgcd-self [simp]: $0 \leq m \implies \text{zgcd } m \ m = m$
 $\langle \text{proof} \rangle$

lemma $\text{zgcd-zmult-eq-self}$ [simp]: $0 \leq k \implies \text{zgcd } k \ (k * n) = k$
 $\langle \text{proof} \rangle$

lemma $\text{zgcd-zmult-eq-self2}$ [simp]: $0 \leq k \implies \text{zgcd } (k * n) \ k = k$
 $\langle \text{proof} \rangle$

definition $\text{z lcm } i \ j = \text{int } (\text{lcm}(\text{nat}(\text{abs } i)) \ (\text{nat}(\text{abs } j)))$

lemma dvd-zlcm-self1 [simp, algebra]: $i \ \text{dvd} \ \text{z lcm } i \ j$
 $\langle \text{proof} \rangle$

lemma dvd-zlcm-self2 [simp, algebra]: $j \ \text{dvd} \ \text{z lcm } i \ j$
 $\langle \text{proof} \rangle$

lemma dvd-imp-dvd-zlcm1 :
assumes $k \ \text{dvd} \ i$ **shows** $k \ \text{dvd} \ (\text{z lcm } i \ j)$
 $\langle \text{proof} \rangle$

lemma dvd-imp-dvd-zlcm2 :
assumes $k \ \text{dvd} \ j$ **shows** $k \ \text{dvd} \ (\text{z lcm } i \ j)$
 $\langle \text{proof} \rangle$

lemma zdvd-self-abs1 : $(d::\text{int}) \ \text{dvd} \ (\text{abs } d)$
 $\langle \text{proof} \rangle$

lemma zdvd-self-abs2 : $(\text{abs } (d::\text{int})) \ \text{dvd} \ d$
 $\langle \text{proof} \rangle$

lemma lcm-pos :
assumes mpos : $m > 0$

```

and npos:  $n > 0$ 
shows  $\text{lcm } m \ n > 0$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma z lcm-pos:
  assumes anz:  $a \neq 0$ 
  and bnz:  $b \neq 0$ 
  shows  $0 < \text{z lcm } a \ b$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma zgcd-code [code]:
   $\text{zgcd } k \ l = \text{if } l = 0 \text{ then } k \text{ else } \text{zgcd } l \ (|k| \bmod |l|)$ 
 $\langle \text{proof} \rangle$ 

```

```

end

```

49 Archimedean-Field: Archimedean Fields, Floor and Ceiling Functions

```

theory Archimedean-Field
imports Main
begin

```

49.1 Class of Archimedean fields

Archimedean fields have no infinite elements.

```

class archimedean-field = ordered-field + number-ring +
  assumes ex-le-of-int:  $\exists z. x \leq \text{of-int } z$ 

```

```

lemma ex-less-of-int:
  fixes  $x :: 'a :: \text{archimedean-field}$  shows  $\exists z. x < \text{of-int } z$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma ex-of-int-less:
  fixes  $x :: 'a :: \text{archimedean-field}$  shows  $\exists z. \text{of-int } z < x$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma ex-less-of-nat:
  fixes  $x :: 'a :: \text{archimedean-field}$  shows  $\exists n. x < \text{of-nat } n$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma ex-le-of-nat:
  fixes  $x :: 'a :: \text{archimedean-field}$  shows  $\exists n. x \leq \text{of-nat } n$ 
 $\langle \text{proof} \rangle$ 

```

Archimedean fields have no infinitesimal elements.

```

lemma ex-inverse-of-nat-Suc-less:

```

fixes $x :: 'a::\text{archimedean-field}$
assumes $0 < x$ **shows** $\exists n. \text{inverse } (\text{of-nat } (\text{Suc } n)) < x$
 $\langle \text{proof} \rangle$

lemma *ex-inverse-of-nat-less*:
fixes $x :: 'a::\text{archimedean-field}$
assumes $0 < x$ **shows** $\exists n > 0. \text{inverse } (\text{of-nat } n) < x$
 $\langle \text{proof} \rangle$

lemma *ex-less-of-nat-mult*:
fixes $x :: 'a::\text{archimedean-field}$
assumes $0 < x$ **shows** $\exists n. y < \text{of-nat } n * x$
 $\langle \text{proof} \rangle$

49.2 Existence and uniqueness of floor function

lemma *exists-least-lemma*:
assumes $\neg P\ 0$ **and** $\exists n. P\ n$
shows $\exists n. \neg P\ n \wedge P\ (\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma *floor-exists*:
fixes $x :: 'a::\text{archimedean-field}$
shows $\exists z. \text{of-int } z \leq x \wedge x < \text{of-int } (z + 1)$
 $\langle \text{proof} \rangle$

lemma *floor-exists1*:
fixes $x :: 'a::\text{archimedean-field}$
shows $\exists! z. \text{of-int } z \leq x \wedge x < \text{of-int } (z + 1)$
 $\langle \text{proof} \rangle$

49.3 Floor function

definition
 $\text{floor} :: 'a::\text{archimedean-field} \Rightarrow \text{int}$ **where**
 $[\text{code del}]: \text{floor } x = (\text{THE } z. \text{of-int } z \leq x \wedge x < \text{of-int } (z + 1))$

notation (*xsymbols*)
 $\text{floor } (\lfloor \cdot \rfloor)$

notation (*HTML output*)
 $\text{floor } (\lfloor \cdot \rfloor)$

lemma *floor-correct*: $\text{of-int } (\text{floor } x) \leq x \wedge x < \text{of-int } (\text{floor } x + 1)$
 $\langle \text{proof} \rangle$

lemma *floor-unique*: $\llbracket \text{of-int } z \leq x; x < \text{of-int } z + 1 \rrbracket \Longrightarrow \text{floor } x = z$
 $\langle \text{proof} \rangle$

lemma *of-int-floor-le*: $\text{of-int } (\text{floor } x) \leq x$

$\langle proof \rangle$

lemma *le-floor-iff*: $z \leq \text{floor } x \longleftrightarrow \text{of-int } z \leq x$
 $\langle proof \rangle$

lemma *floor-less-iff*: $\text{floor } x < z \longleftrightarrow x < \text{of-int } z$
 $\langle proof \rangle$

lemma *less-floor-iff*: $z < \text{floor } x \longleftrightarrow \text{of-int } z + 1 \leq x$
 $\langle proof \rangle$

lemma *floor-le-iff*: $\text{floor } x \leq z \longleftrightarrow x < \text{of-int } z + 1$
 $\langle proof \rangle$

lemma *floor-mono*: **assumes** $x \leq y$ **shows** $\text{floor } x \leq \text{floor } y$
 $\langle proof \rangle$

lemma *floor-less-cancel*: $\text{floor } x < \text{floor } y \implies x < y$
 $\langle proof \rangle$

lemma *floor-of-int [simp]*: $\text{floor } (\text{of-int } z) = z$
 $\langle proof \rangle$

lemma *floor-of-nat [simp]*: $\text{floor } (\text{of-nat } n) = \text{int } n$
 $\langle proof \rangle$

Floor with numerals

lemma *floor-zero [simp]*: $\text{floor } 0 = 0$
 $\langle proof \rangle$

lemma *floor-one [simp]*: $\text{floor } 1 = 1$
 $\langle proof \rangle$

lemma *floor-number-of [simp]*: $\text{floor } (\text{number-of } v) = \text{number-of } v$
 $\langle proof \rangle$

lemma *zero-le-floor [simp]*: $0 \leq \text{floor } x \longleftrightarrow 0 \leq x$
 $\langle proof \rangle$

lemma *one-le-floor [simp]*: $1 \leq \text{floor } x \longleftrightarrow 1 \leq x$
 $\langle proof \rangle$

lemma *number-of-le-floor [simp]*: $\text{number-of } v \leq \text{floor } x \longleftrightarrow \text{number-of } v \leq x$
 $\langle proof \rangle$

lemma *zero-less-floor [simp]*: $0 < \text{floor } x \longleftrightarrow 1 \leq x$
 $\langle proof \rangle$

lemma *one-less-floor [simp]*: $1 < \text{floor } x \longleftrightarrow 2 \leq x$

$\langle \text{proof} \rangle$

lemma *number-of-less-floor* [simp]:

$\text{number-of } v < \text{floor } x \longleftrightarrow \text{number-of } v + 1 \leq x$

$\langle \text{proof} \rangle$

lemma *floor-le-zero* [simp]: $\text{floor } x \leq 0 \longleftrightarrow x < 1$

$\langle \text{proof} \rangle$

lemma *floor-le-one* [simp]: $\text{floor } x \leq 1 \longleftrightarrow x < 2$

$\langle \text{proof} \rangle$

lemma *floor-le-number-of* [simp]:

$\text{floor } x \leq \text{number-of } v \longleftrightarrow x < \text{number-of } v + 1$

$\langle \text{proof} \rangle$

lemma *floor-less-zero* [simp]: $\text{floor } x < 0 \longleftrightarrow x < 0$

$\langle \text{proof} \rangle$

lemma *floor-less-one* [simp]: $\text{floor } x < 1 \longleftrightarrow x < 1$

$\langle \text{proof} \rangle$

lemma *floor-less-number-of* [simp]:

$\text{floor } x < \text{number-of } v \longleftrightarrow x < \text{number-of } v$

$\langle \text{proof} \rangle$

Addition and subtraction of integers

lemma *floor-add-of-int* [simp]: $\text{floor } (x + \text{of-int } z) = \text{floor } x + z$

$\langle \text{proof} \rangle$

lemma *floor-add-number-of* [simp]:

$\text{floor } (x + \text{number-of } v) = \text{floor } x + \text{number-of } v$

$\langle \text{proof} \rangle$

lemma *floor-add-one* [simp]: $\text{floor } (x + 1) = \text{floor } x + 1$

$\langle \text{proof} \rangle$

lemma *floor-diff-of-int* [simp]: $\text{floor } (x - \text{of-int } z) = \text{floor } x - z$

$\langle \text{proof} \rangle$

lemma *floor-diff-number-of* [simp]:

$\text{floor } (x - \text{number-of } v) = \text{floor } x - \text{number-of } v$

$\langle \text{proof} \rangle$

lemma *floor-diff-one* [simp]: $\text{floor } (x - 1) = \text{floor } x - 1$

$\langle \text{proof} \rangle$

49.4 Ceiling function

definition

$\text{ceiling} :: 'a :: \text{archimedean-field} \Rightarrow \text{int}$ **where**
 $[\text{code del}]: \text{ceiling } x = - \text{floor } (-x)$

notation (*xsymbols*)

$\text{ceiling} \ (\lceil - \rceil)$

notation (*HTML output*)

$\text{ceiling} \ (\lceil - \rceil)$

lemma *ceiling-correct*: $\text{of-int } (\text{ceiling } x) - 1 < x \wedge x \leq \text{of-int } (\text{ceiling } x)$
 $\langle \text{proof} \rangle$

lemma *ceiling-unique*: $\llbracket \text{of-int } z - 1 < x; x \leq \text{of-int } z \rrbracket \Longrightarrow \text{ceiling } x = z$
 $\langle \text{proof} \rangle$

lemma *le-of-int-ceiling*: $x \leq \text{of-int } (\text{ceiling } x)$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-iff*: $\text{ceiling } x \leq z \longleftrightarrow x \leq \text{of-int } z$
 $\langle \text{proof} \rangle$

lemma *less-ceiling-iff*: $z < \text{ceiling } x \longleftrightarrow \text{of-int } z < x$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-iff*: $\text{ceiling } x < z \longleftrightarrow x \leq \text{of-int } z - 1$
 $\langle \text{proof} \rangle$

lemma *le-ceiling-iff*: $z \leq \text{ceiling } x \longleftrightarrow \text{of-int } z - 1 < x$
 $\langle \text{proof} \rangle$

lemma *ceiling-mono*: $x \geq y \Longrightarrow \text{ceiling } x \geq \text{ceiling } y$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-cancel*: $\text{ceiling } x < \text{ceiling } y \Longrightarrow x < y$
 $\langle \text{proof} \rangle$

lemma *ceiling-of-int [simp]*: $\text{ceiling } (\text{of-int } z) = z$
 $\langle \text{proof} \rangle$

lemma *ceiling-of-nat [simp]*: $\text{ceiling } (\text{of-nat } n) = \text{int } n$
 $\langle \text{proof} \rangle$

Ceiling with numerals

lemma *ceiling-zero [simp]*: $\text{ceiling } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *ceiling-one [simp]*: $\text{ceiling } 1 = 1$

$\langle \text{proof} \rangle$

lemma *ceiling-number-of* [simp]: $\text{ceiling} (\text{number-of } v) = \text{number-of } v$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-zero* [simp]: $\text{ceiling } x \leq 0 \longleftrightarrow x \leq 0$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-one* [simp]: $\text{ceiling } x \leq 1 \longleftrightarrow x \leq 1$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-number-of* [simp]:
 $\text{ceiling } x \leq \text{number-of } v \longleftrightarrow x \leq \text{number-of } v$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-zero* [simp]: $\text{ceiling } x < 0 \longleftrightarrow x \leq -1$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-one* [simp]: $\text{ceiling } x < 1 \longleftrightarrow x \leq 0$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-number-of* [simp]:
 $\text{ceiling } x < \text{number-of } v \longleftrightarrow x \leq \text{number-of } v - 1$
 $\langle \text{proof} \rangle$

lemma *zero-le-ceiling* [simp]: $0 \leq \text{ceiling } x \longleftrightarrow -1 < x$
 $\langle \text{proof} \rangle$

lemma *one-le-ceiling* [simp]: $1 \leq \text{ceiling } x \longleftrightarrow 0 < x$
 $\langle \text{proof} \rangle$

lemma *number-of-le-ceiling* [simp]:
 $\text{number-of } v \leq \text{ceiling } x \longleftrightarrow \text{number-of } v - 1 < x$
 $\langle \text{proof} \rangle$

lemma *zero-less-ceiling* [simp]: $0 < \text{ceiling } x \longleftrightarrow 0 < x$
 $\langle \text{proof} \rangle$

lemma *one-less-ceiling* [simp]: $1 < \text{ceiling } x \longleftrightarrow 1 < x$
 $\langle \text{proof} \rangle$

lemma *number-of-less-ceiling* [simp]:
 $\text{number-of } v < \text{ceiling } x \longleftrightarrow \text{number-of } v < x$
 $\langle \text{proof} \rangle$

Addition and subtraction of integers

lemma *ceiling-add-of-int* [simp]: $\text{ceiling } (x + \text{of-int } z) = \text{ceiling } x + z$
 $\langle \text{proof} \rangle$

lemma *ceiling-add-number-of* [simp]:

$$\text{ceiling } (x + \text{number-of } v) = \text{ceiling } x + \text{number-of } v$$

<proof>

lemma *ceiling-add-one* [simp]: $\text{ceiling } (x + 1) = \text{ceiling } x + 1$

<proof>

lemma *ceiling-diff-of-int* [simp]: $\text{ceiling } (x - \text{of-int } z) = \text{ceiling } x - z$

<proof>

lemma *ceiling-diff-number-of* [simp]:

$$\text{ceiling } (x - \text{number-of } v) = \text{ceiling } x - \text{number-of } v$$

<proof>

lemma *ceiling-diff-one* [simp]: $\text{ceiling } (x - 1) = \text{ceiling } x - 1$

<proof>

49.5 Negation

lemma *floor-minus*: $\text{floor } (-x) = -\text{ceiling } x$

<proof>

lemma *ceiling-minus*: $\text{ceiling } (-x) = -\text{floor } x$

<proof>

end

50 Rational: Rational numbers

theory *Rational*

imports *GCD Archimedean-Field*

uses (*Tools/rat-arith.ML*)

begin

50.1 Rational numbers as quotient

50.1.1 Construction of the type of rational numbers

definition

$\text{ratrel} :: ((\text{int} \times \text{int}) \times (\text{int} \times \text{int})) \text{ set}$ **where**

$$\text{ratrel} = \{(x, y). \text{snd } x \neq 0 \wedge \text{snd } y \neq 0 \wedge \text{fst } x * \text{snd } y = \text{fst } y * \text{snd } x\}$$

lemma *ratrel-iff* [simp]:

$$(x, y) \in \text{ratrel} \longleftrightarrow \text{snd } x \neq 0 \wedge \text{snd } y \neq 0 \wedge \text{fst } x * \text{snd } y = \text{fst } y * \text{snd } x$$

<proof>

lemma *refl-on-ratrel*: $\text{refl-on } \{x. \text{snd } x \neq 0\} \text{ ratrel}$

<proof>

lemma *sym-ratrel*: *sym ratrel*
 ⟨*proof*⟩

lemma *trans-ratrel*: *trans ratrel*
 ⟨*proof*⟩

lemma *equiv-ratrel*: *equiv {x. snd x ≠ 0} ratrel*
 ⟨*proof*⟩

lemmas *UN-ratrel* = *UN-equiv-class* [*OF equiv-ratrel*]
lemmas *UN-ratrel2* = *UN-equiv-class2* [*OF equiv-ratrel equiv-ratrel*]

lemma *equiv-ratrel-iff* [*iff*]:
 assumes *snd x ≠ 0* and *snd y ≠ 0*
 shows *ratrel “ {x} = ratrel “ {y} ⟷ (x, y) ∈ ratrel*
 ⟨*proof*⟩

typedef (*Rat*) *rat* = {*x. snd x ≠ 0*} // *ratrel*
 ⟨*proof*⟩

lemma *ratrel-in-Rat* [*simp*]: *snd x ≠ 0 ⟹ ratrel “ {x} ∈ Rat*
 ⟨*proof*⟩

declare *Abs-Rat-inject* [*simp*] *Abs-Rat-inverse* [*simp*]

50.1.2 Representation and basic operations

definition

Fract :: *int* ⇒ *int* ⇒ *rat* **where**
 [*code del*]: *Fract a b* = *Abs-Rat* (*ratrel “ {if b = 0 then (0, 1) else (a, b)}*)

code-datatype *Fract*

lemma *Rat-cases* [*case-names Fract, cases type: rat*]:
 assumes $\bigwedge a b. q = \text{Fract } a b \implies b \neq 0 \implies C$
 shows *C*
 ⟨*proof*⟩

lemma *Rat-induct* [*case-names Fract, induct type: rat*]:
 assumes $\bigwedge a b. b \neq 0 \implies P (\text{Fract } a b)$
 shows *P q*
 ⟨*proof*⟩

lemma *eq-rat*:
 shows $\bigwedge a b c d. b \neq 0 \implies d \neq 0 \implies \text{Fract } a b = \text{Fract } c d \iff a * d = c * b$
 and $\bigwedge a. \text{Fract } a 0 = \text{Fract } 0 1$
 and $\bigwedge a c. \text{Fract } 0 a = \text{Fract } 0 c$
 ⟨*proof*⟩

instantiation *rat* :: {*comm-ring-1*, *recpower*}
begin

definition

Zero-rat-def [*code*, *code unfold*]: $0 = \text{Fract } 0 \ 1$

definition

One-rat-def [*code*, *code unfold*]: $1 = \text{Fract } 1 \ 1$

definition

add-rat-def [*code del*]:

$q + r = \text{Abs-Rat } (\bigcup x \in \text{Rep-Rat } q. \bigcup y \in \text{Rep-Rat } r.$

$\text{ratrel } \{ (fst\ x * snd\ y + fst\ y * snd\ x, snd\ x * snd\ y) \}$)

lemma *add-rat* [*simp*]:

assumes $b \neq 0$ **and** $d \neq 0$

shows $\text{Fract } a \ b + \text{Fract } c \ d = \text{Fract } (a * d + c * b) \ (b * d)$

<proof>

definition

minus-rat-def [*code del*]:

$- q = \text{Abs-Rat } (\bigcup x \in \text{Rep-Rat } q. \text{ratrel } \{ (-\text{fst } x, snd\ x) \})$

lemma *minus-rat* [*simp*, *code*]: $-\text{Fract } a \ b = \text{Fract } (-a) \ b$

<proof>

lemma *minus-rat-cancel* [*simp*]: $\text{Fract } (-a) \ (-b) = \text{Fract } a \ b$

<proof>

definition

diff-rat-def [*code del*]: $q - r = q + - (r::\text{rat})$

lemma *diff-rat* [*simp*]:

assumes $b \neq 0$ **and** $d \neq 0$

shows $\text{Fract } a \ b - \text{Fract } c \ d = \text{Fract } (a * d - c * b) \ (b * d)$

<proof>

definition

mult-rat-def [*code del*]:

$q * r = \text{Abs-Rat } (\bigcup x \in \text{Rep-Rat } q. \bigcup y \in \text{Rep-Rat } r.$

$\text{ratrel } \{ (fst\ x * fst\ y, snd\ x * snd\ y) \}$)

lemma *mult-rat* [*simp*]: $\text{Fract } a \ b * \text{Fract } c \ d = \text{Fract } (a * c) \ (b * d)$

<proof>

lemma *mult-rat-cancel*:

assumes $c \neq 0$

shows $\text{Fract } (c * a) \ (c * b) = \text{Fract } a \ b$

$\langle proof \rangle$

primrec *power-rat*

where

$q \wedge 0 = (1::rat)$
 $| q \wedge Suc\ n = (q::rat) * (q \wedge n)$

instance $\langle proof \rangle$

declare *power-rat.simps* [*simp del*]

end

lemma *of-nat-rat*: $of_nat\ k = Fract\ (of_nat\ k)\ 1$
 $\langle proof \rangle$

lemma *of-int-rat*: $of_int\ k = Fract\ k\ 1$
 $\langle proof \rangle$

lemma *Fract-of-nat-eq*: $Fract\ (of_nat\ k)\ 1 = of_nat\ k$
 $\langle proof \rangle$

lemma *Fract-of-int-eq*: $Fract\ k\ 1 = of_int\ k$
 $\langle proof \rangle$

instantiation *rat* :: *number-ring*
begin

definition

rat-number-of-def [*code del*]: $number_of\ w = Fract\ w\ 1$

instance $\langle proof \rangle$

end

lemma *rat-number-collapse* [*code post*]:

$Fract\ 0\ k = 0$
 $Fract\ 1\ 1 = 1$
 $Fract\ (number_of\ k)\ 1 = number_of\ k$
 $Fract\ k\ 0 = 0$
 $\langle proof \rangle$

lemma *rat-number-expand* [*code unfold*]:

$0 = Fract\ 0\ 1$
 $1 = Fract\ 1\ 1$
 $number_of\ k = Fract\ (number_of\ k)\ 1$
 $\langle proof \rangle$

lemma *iszero-rat* [*simp*]:

iszero (*number-of* *k* :: *rat*) \longleftrightarrow *iszero* (*number-of* *k* :: *int*)
 ⟨*proof*⟩

lemma *Rat-cases-nonzero* [*case-names* *Fract 0*]:
 assumes *Fract*: $\bigwedge a\ b. q = \text{Fract } a\ b \implies b \neq 0 \implies a \neq 0 \implies C$
 assumes *0*: $q = 0 \implies C$
 shows *C*
 ⟨*proof*⟩

50.1.3 The field of rational numbers

instantiation *rat* :: {*field*, *division-by-zero*}
begin

definition
inverse-rat-def [*code del*]:
inverse *q* = *Abs-Rat* ($\bigcup x \in \text{Rep-Rat } q.$
ratrel “ {if *fst* *x* = 0 then (0, 1) else (*snd* *x*, *fst* *x*)}”)

lemma *inverse-rat* [*simp*]: *inverse* (*Fract* *a* *b*) = *Fract* *b* *a*
 ⟨*proof*⟩

definition
divide-rat-def [*code del*]: *q* / *r* = *q* * *inverse* (*r*::*rat*)

lemma *divide-rat* [*simp*]: *Fract* *a* *b* / *Fract* *c* *d* = *Fract* (*a* * *d*) (*b* * *c*)
 ⟨*proof*⟩

instance ⟨*proof*⟩

end

50.1.4 Various

lemma *Fract-add-one*: $n \neq 0 \implies \text{Fract } (m + n)\ n = \text{Fract } m\ n + 1$
 ⟨*proof*⟩

lemma *Fract-of-int-quotient*: *Fract* *k* *l* = *of-int* *k* / *of-int* *l*
 ⟨*proof*⟩

lemma *Fract-number-of-quotient* [*code post*]:
Fract (*number-of* *k*) (*number-of* *l*) = *number-of* *k* / *number-of* *l*
 ⟨*proof*⟩

lemma *Fract-1-number-of* [*code post*]:
Fract 1 (*number-of* *k*) = 1 / *number-of* *k*
 ⟨*proof*⟩

50.1.5 The ordered field of rational numbers**instantiation** *rat* :: *linorder***begin****definition***le-rat-def* [*code del*]:
$$q \leq r \longleftrightarrow \text{contents } (\bigcup x \in \text{Rep-Rat } q. \bigcup y \in \text{Rep-Rat } r. \\ \{(fst\ x * snd\ y) * (snd\ x * snd\ y) \leq (fst\ y * snd\ x) * (snd\ x * snd\ y)\})$$
lemma *le-rat* [*simp*]:**assumes** $b \neq 0$ **and** $d \neq 0$ **shows** $\text{Fract } a\ b \leq \text{Fract } c\ d \longleftrightarrow (a * d) * (b * d) \leq (c * b) * (b * d)$ *<proof>***definition***less-rat-def* [*code del*]: $z < (w::rat) \longleftrightarrow z \leq w \wedge z \neq w$ **lemma** *less-rat* [*simp*]:**assumes** $b \neq 0$ **and** $d \neq 0$ **shows** $\text{Fract } a\ b < \text{Fract } c\ d \longleftrightarrow (a * d) * (b * d) < (c * b) * (b * d)$ *<proof>***instance** *<proof>***end****instantiation** *rat* :: {*distrib-lattice*, *abs-if*, *sgn-if*}**begin****definition***abs-rat-def* [*code del*]: $|q| = (\text{if } q < 0 \text{ then } -q \text{ else } (q::rat))$ **lemma** *abs-rat* [*simp*, *code*]: $|\text{Fract } a\ b| = \text{Fract } |a|\ |b|$ *<proof>***definition***sgn-rat-def* [*code del*]: $\text{sgn } (q::rat) = (\text{if } q = 0 \text{ then } 0 \text{ else if } 0 < q \text{ then } 1 \text{ else } -1)$ **lemma** *sgn-rat* [*simp*, *code*]: $\text{sgn } (\text{Fract } a\ b) = \text{of-int } (\text{sgn } a * \text{sgn } b)$ *<proof>***definition** $(inf :: rat \Rightarrow rat \Rightarrow rat) = min$ **definition** $(sup :: rat \Rightarrow rat \Rightarrow rat) = max$ **instance** *<proof>*

end

instance *rat* :: *ordered-field*
 ⟨*proof*⟩

lemma *Rat-induct-pos* [*case-names Fract, induct type: rat*]:
assumes *step*: $\bigwedge a\ b. 0 < b \implies P\ (\text{Fract } a\ b)$
shows $P\ q$
 ⟨*proof*⟩

lemma *zero-less-Fract-iff*:
 $0 < b \implies 0 < \text{Fract } a\ b \longleftrightarrow 0 < a$
 ⟨*proof*⟩

lemma *Fract-less-zero-iff*:
 $0 < b \implies \text{Fract } a\ b < 0 \longleftrightarrow a < 0$
 ⟨*proof*⟩

lemma *zero-le-Fract-iff*:
 $0 < b \implies 0 \leq \text{Fract } a\ b \longleftrightarrow 0 \leq a$
 ⟨*proof*⟩

lemma *Fract-le-zero-iff*:
 $0 < b \implies \text{Fract } a\ b \leq 0 \longleftrightarrow a \leq 0$
 ⟨*proof*⟩

lemma *one-less-Fract-iff*:
 $0 < b \implies 1 < \text{Fract } a\ b \longleftrightarrow b < a$
 ⟨*proof*⟩

lemma *Fract-less-one-iff*:
 $0 < b \implies \text{Fract } a\ b < 1 \longleftrightarrow a < b$
 ⟨*proof*⟩

lemma *one-le-Fract-iff*:
 $0 < b \implies 1 \leq \text{Fract } a\ b \longleftrightarrow b \leq a$
 ⟨*proof*⟩

lemma *Fract-le-one-iff*:
 $0 < b \implies \text{Fract } a\ b \leq 1 \longleftrightarrow a \leq b$
 ⟨*proof*⟩

50.1.6 Rationals are an Archimedean field

lemma *rat-floor-lemma*:
assumes $0 < b$
shows $\text{of-int } (a \text{ div } b) \leq \text{Fract } a\ b \wedge \text{Fract } a\ b < \text{of-int } (a \text{ div } b + 1)$
 ⟨*proof*⟩

instance *rat* :: *archimedean-field*

<proof>

lemma *floor-Fract*:

assumes $0 < b$ **shows** $\text{floor } (\text{Fract } a \ b) = a \ \text{div } b$

<proof>

50.2 Arithmetic setup

<ML>

50.3 Embedding from Rationals to other Fields

class *field-char-0* = *field* + *ring-char-0*

subclass (in *ordered-field*) *field-char-0* *<proof>*

context *field-char-0*

begin

definition *of-rat* :: *rat* \Rightarrow 'a **where**

[code del]: *of-rat* *q* = *contents* ($\bigcup (a,b) \in \text{Rep-Rat } q. \{ \text{of-int } a \ / \ \text{of-int } b \}$)

end

lemma *of-rat-congruent*:

$(\lambda(a, b). \{ \text{of-int } a \ / \ \text{of-int } b :: 'a::\text{field-char-0} \})$ respects *ratrel*

<proof>

lemma *of-rat-rat*: $b \neq 0 \implies \text{of-rat } (\text{Fract } a \ b) = \text{of-int } a \ / \ \text{of-int } b$

<proof>

lemma *of-rat-0* [*simp*]: *of-rat* 0 = 0

<proof>

lemma *of-rat-1* [*simp*]: *of-rat* 1 = 1

<proof>

lemma *of-rat-add*: *of-rat* ($a + b$) = *of-rat* *a* + *of-rat* *b*

<proof>

lemma *of-rat-minus*: *of-rat* ($- a$) = - *of-rat* *a*

<proof>

lemma *of-rat-diff*: *of-rat* ($a - b$) = *of-rat* *a* - *of-rat* *b*

<proof>

lemma *of-rat-mult*: *of-rat* ($a * b$) = *of-rat* *a* * *of-rat* *b*

<proof>

lemma *nonzero-of-rat-inverse*:

$a \neq 0 \implies \text{of-rat } (\text{inverse } a) = \text{inverse } (\text{of-rat } a)$
 $\langle \text{proof} \rangle$

lemma *of-rat-inverse*:

$(\text{of-rat } (\text{inverse } a) :: 'a :: \{\text{field-char-0}, \text{division-by-zero}\}) =$
 $\text{inverse } (\text{of-rat } a)$
 $\langle \text{proof} \rangle$

lemma *nonzero-of-rat-divide*:

$b \neq 0 \implies \text{of-rat } (a / b) = \text{of-rat } a / \text{of-rat } b$
 $\langle \text{proof} \rangle$

lemma *of-rat-divide*:

$(\text{of-rat } (a / b) :: 'a :: \{\text{field-char-0}, \text{division-by-zero}\})$
 $= \text{of-rat } a / \text{of-rat } b$
 $\langle \text{proof} \rangle$

lemma *of-rat-power*:

$(\text{of-rat } (a ^ n) :: 'a :: \{\text{field-char-0}, \text{recpower}\}) = \text{of-rat } a ^ n$
 $\langle \text{proof} \rangle$

lemma *of-rat-eq-iff [simp]*: $(\text{of-rat } a = \text{of-rat } b) = (a = b)$
 $\langle \text{proof} \rangle$

lemma *of-rat-less*:

$(\text{of-rat } r :: 'a :: \text{ordered-field}) < \text{of-rat } s \longleftrightarrow r < s$
 $\langle \text{proof} \rangle$

lemma *of-rat-less-eq*:

$(\text{of-rat } r :: 'a :: \text{ordered-field}) \leq \text{of-rat } s \longleftrightarrow r \leq s$
 $\langle \text{proof} \rangle$

lemmas *of-rat-eq-0-iff [simp]* = *of-rat-eq-iff [of - 0, simplified]*

lemma *of-rat-eq-id [simp]*: $\text{of-rat } = \text{id}$
 $\langle \text{proof} \rangle$

Collapse nested embeddings

lemma *of-rat-of-nat-eq [simp]*: $\text{of-rat } (\text{of-nat } n) = \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *of-rat-of-int-eq [simp]*: $\text{of-rat } (\text{of-int } z) = \text{of-int } z$
 $\langle \text{proof} \rangle$

lemma *of-rat-number-of-eq [simp]*:

$\text{of-rat } (\text{number-of } w) = (\text{number-of } w :: 'a :: \{\text{number-ring}, \text{field-char-0}\})$
 $\langle \text{proof} \rangle$

lemmas *zero-rat* = *Zero-rat-def*

lemmas *one-rat* = *One-rat-def*

abbreviation

rat-of-nat :: *nat* \Rightarrow *rat*

where

rat-of-nat \equiv *of-nat*

abbreviation

rat-of-int :: *int* \Rightarrow *rat*

where

rat-of-int \equiv *of-int*

50.4 The Set of Rational Numbers

context *field-char-0*

begin

definition

Rats :: 'a set **where**

[code del]: *Rats* = *range of-rat*

notation (*xsymbols*)

Rats (\mathbb{Q})

end

lemma *Rats-of-rat* [*simp*]: *of-rat* *r* \in *Rats*

\langle *proof* \rangle

lemma *Rats-of-int* [*simp*]: *of-int* *z* \in *Rats*

\langle *proof* \rangle

lemma *Rats-of-nat* [*simp*]: *of-nat* *n* \in *Rats*

\langle *proof* \rangle

lemma *Rats-number-of* [*simp*]:

(*number-of* *w*::'a::{*number-ring*,*field-char-0*}) \in *Rats*

\langle *proof* \rangle

lemma *Rats-0* [*simp*]: 0 \in *Rats*

\langle *proof* \rangle

lemma *Rats-1* [*simp*]: 1 \in *Rats*

\langle *proof* \rangle

lemma *Rats-add* [*simp*]: $\llbracket a \in \textit{Rats}; b \in \textit{Rats} \rrbracket \Longrightarrow a + b \in \textit{Rats}$

\langle *proof* \rangle

lemma *Rats-minus* [simp]: $a \in \text{Rats} \implies -a \in \text{Rats}$
 <proof>

lemma *Rats-diff* [simp]: $\llbracket a \in \text{Rats}; b \in \text{Rats} \rrbracket \implies a - b \in \text{Rats}$
 <proof>

lemma *Rats-mult* [simp]: $\llbracket a \in \text{Rats}; b \in \text{Rats} \rrbracket \implies a * b \in \text{Rats}$
 <proof>

lemma *nonzero-Rats-inverse*:
 fixes $a :: 'a::\text{field-char-0}$
 shows $\llbracket a \in \text{Rats}; a \neq 0 \rrbracket \implies \text{inverse } a \in \text{Rats}$
 <proof>

lemma *Rats-inverse* [simp]:
 fixes $a :: 'a::\{\text{field-char-0}, \text{division-by-zero}\}$
 shows $a \in \text{Rats} \implies \text{inverse } a \in \text{Rats}$
 <proof>

lemma *nonzero-Rats-divide*:
 fixes $a b :: 'a::\text{field-char-0}$
 shows $\llbracket a \in \text{Rats}; b \in \text{Rats}; b \neq 0 \rrbracket \implies a / b \in \text{Rats}$
 <proof>

lemma *Rats-divide* [simp]:
 fixes $a b :: 'a::\{\text{field-char-0}, \text{division-by-zero}\}$
 shows $\llbracket a \in \text{Rats}; b \in \text{Rats} \rrbracket \implies a / b \in \text{Rats}$
 <proof>

lemma *Rats-power* [simp]:
 fixes $a :: 'a::\{\text{field-char-0}, \text{recpower}\}$
 shows $a \in \text{Rats} \implies a^n \in \text{Rats}$
 <proof>

lemma *Rats-cases* [cases set: *Rats*]:
 assumes $q \in \mathbb{Q}$
 obtains $(\text{of-rat } r) \text{ where } q = \text{of-rat } r$
 <proof>

lemma *Rats-induct* [case-names *of-rat*, induct set: *Rats*]:
 $q \in \mathbb{Q} \implies (\bigwedge r. P (\text{of-rat } r)) \implies P q$
 <proof>

50.5 Implementation of rational numbers as pairs of integers

lemma *Fract-norm*: $\text{Fract } (a \text{ div } \text{zgcd } a \ b) (b \text{ div } \text{zgcd } a \ b) = \text{Fract } a \ b$
 <proof>

definition *Fract-norm* :: *int* \Rightarrow *int* \Rightarrow *rat* **where**
 [simp, code del]: *Fract-norm* *a b* = *Fract a b*

lemma *Fract-norm-code* [code]: *Fract-norm a b* = (if *a* = 0 \vee *b* = 0 then 0 else
 let *c* = *zgcd a b* in
 if *b* > 0 then *Fract (a div c) (b div c)* else *Fract (-(a div c)) (-(b div c))*)
 <proof>

lemma [code]:
of-rat (Fract a b) = (if *b* \neq 0 then *of-int a / of-int b* else 0)
 <proof>

instantiation *rat* :: *eq*
begin

definition [code del]: *eq-class.eq (a::rat) b* \longleftrightarrow *a - b* = 0

instance <proof>

lemma *rat-eq-code* [code]:
eq-class.eq (Fract a b) (Fract c d) \longleftrightarrow (if *b* = 0
 then *c* = 0 \vee *d* = 0
 else if *d* = 0
 then *a* = 0 \vee *b* = 0
 else *a * d* = *b * c*)
 <proof>

lemma *rat-eq-refl* [code nbe]:
eq-class.eq (r::rat) r \longleftrightarrow *True*
 <proof>

end

lemma *le-rat'*:
 assumes *b* \neq 0
 and *d* \neq 0
 shows *Fract a b* \leq *Fract c d* \longleftrightarrow *a * |d| * sgn b* \leq *c * |b| * sgn d*
 <proof>

lemma *less-rat'*:
 assumes *b* \neq 0
 and *d* \neq 0
 shows *Fract a b* < *Fract c d* \longleftrightarrow *a * |d| * sgn b* < *c * |b| * sgn d*
 <proof>

lemma (in *ordered-idom*) *sgn-greater* [simp]:
 0 < *sgn a* \longleftrightarrow 0 < *a*
 <proof>

lemma (in ordered-idom) sgn-less [simp]:
 $\text{sgn } a < 0 \iff a < 0$
 ⟨proof⟩

lemma rat-le-eq-code [code]:
 $\text{Fract } a \ b < \text{Fract } c \ d \iff (\text{if } b = 0$
 $\text{then } \text{sgn } c * \text{sgn } d > 0$
 $\text{else if } d = 0$
 $\text{then } \text{sgn } a * \text{sgn } b < 0$
 $\text{else } a * |d| * \text{sgn } b < c * |b| * \text{sgn } d)$
 ⟨proof⟩

lemma rat-less-eq-code [code]:
 $\text{Fract } a \ b \leq \text{Fract } c \ d \iff (\text{if } b = 0$
 $\text{then } \text{sgn } c * \text{sgn } d \geq 0$
 $\text{else if } d = 0$
 $\text{then } \text{sgn } a * \text{sgn } b \leq 0$
 $\text{else } a * |d| * \text{sgn } b \leq c * |b| * \text{sgn } d)$
 ⟨proof⟩

lemma rat-plus-code [code]:
 $\text{Fract } a \ b + \text{Fract } c \ d = (\text{if } b = 0$
 $\text{then } \text{Fract } c \ d$
 $\text{else if } d = 0$
 $\text{then } \text{Fract } a \ b$
 $\text{else } \text{Fract-norm } (a * d + c * b) (b * d))$
 ⟨proof⟩

lemma rat-times-code [code]:
 $\text{Fract } a \ b * \text{Fract } c \ d = \text{Fract-norm } (a * c) (b * d)$
 ⟨proof⟩

lemma rat-minus-code [code]:
 $\text{Fract } a \ b - \text{Fract } c \ d = (\text{if } b = 0$
 $\text{then } \text{Fract } (- c) \ d$
 $\text{else if } d = 0$
 $\text{then } \text{Fract } a \ b$
 $\text{else } \text{Fract-norm } (a * d - c * b) (b * d))$
 ⟨proof⟩

lemma rat-inverse-code [code]:
 $\text{inverse } (\text{Fract } a \ b) = (\text{if } b = 0 \text{ then } \text{Fract } 1 \ 0$
 $\text{else if } a < 0 \text{ then } \text{Fract } (- b) \ (- a)$
 $\text{else } \text{Fract } b \ a)$
 ⟨proof⟩

lemma rat-divide-code [code]:
 $\text{Fract } a \ b / \text{Fract } c \ d = \text{Fract-norm } (a * d) (b * c)$

```

    <proof>

hide (open) const Fract-norm

Setup for SML code generator

types-code
  rat ((int */ int))
attach (term-of) <<
fun term-of-rat (p, q) =
  let
    val rT = Type (Rational.rat, [])
  in
    if q = 1 orelse p = 0 then HOLogic.mk-number rT p
    else @{term op / :: rat ⇒ rat ⇒ rat} $
      HOLogic.mk-number rT p $ HOLogic.mk-number rT q
    end;
  >>
attach (test) <<
fun gen-rat i =
  let
    val p = random-range 0 i;
    val q = random-range 1 (i + 1);
    val g = Integer.gcd p q;
    val p' = p div g;
    val q' = q div g;
    val r = (if one-of [true, false] then p' else ~ p',
      if p' = 0 then 0 else q')
  in
    (r, fn () => term-of-rat r)
  end;
>>

consts-code
  Fract ((-, / -))

consts-code
  of-int :: int ⇒ rat (<module>rat'-of'-int)
attach <<
fun rat-of-int 0 = (0, 0)
  | rat-of-int i = (i, 1);
>>

end

```

51 PReal: Positive real numbers

```

theory PReal
imports Rational

```

begin

Could be generalized and moved to *Ring-and-Field*

lemma *add-eq-exists*: $\exists x. a+x = (b::rat)$
 $\langle proof \rangle$

definition

cut :: *rat set* => *bool* **where**
 $[code\ del]: cut\ A = (\{\} \subset A \ \& \ A < \{r. 0 < r\} \ \& \ (\forall y \in A. (\forall z. 0 < z \ \& \ z < y \longrightarrow z \in A) \ \& \ (\exists u \in A. y < u))))$

lemma *interval-empty-iff*:

$\{y. (x::'a::dense-linear-order) < y \wedge y < z\} = \{\} \longleftrightarrow \neg x < z$
 $\langle proof \rangle$

lemma *cut-of-rat*:

assumes $q: 0 < q$ **shows** $cut\ \{r::rat. 0 < r \ \& \ r < q\}$ (**is** $cut\ ?A$)
 $\langle proof \rangle$

typedef *preal* = $\{A. cut\ A\}$
 $\langle proof \rangle$

definition

preal-of-rat :: *rat* => *preal* **where**
 $preal-of-rat\ q = Abs-preal\ \{x::rat. 0 < x \ \& \ x < q\}$

definition

psup :: *preal set* => *preal* **where**
 $psup\ P = Abs-preal\ (\bigcup X \in P. Rep-preal\ X)$

definition

add-set :: $[rat\ set, rat\ set] \Rightarrow rat\ set$ **where**
 $add-set\ A\ B = \{w. \exists x \in A. \exists y \in B. w = x + y\}$

definition

diff-set :: $[rat\ set, rat\ set] \Rightarrow rat\ set$ **where**
 $[code\ del]: diff-set\ A\ B = \{w. \exists x. 0 < w \ \& \ 0 < x \ \& \ x \notin B \ \& \ x + w \in A\}$

definition

mult-set :: $[rat\ set, rat\ set] \Rightarrow rat\ set$ **where**
 $mult-set\ A\ B = \{w. \exists x \in A. \exists y \in B. w = x * y\}$

definition

inverse-set :: *rat set* => *rat set* **where**
 $[code\ del]: inverse-set\ A = \{x. \exists y. 0 < x \ \& \ x < y \ \& \ inverse\ y \notin A\}$

instantiation *preal* :: {ord, plus, minus, times, inverse, one}
begin

definition

preal-less-def [code del]:
 $R < S == \text{Rep-preal } R < \text{Rep-preal } S$

definition

preal-le-def [code del]:
 $R \leq S == \text{Rep-preal } R \subseteq \text{Rep-preal } S$

definition

preal-add-def:
 $R + S == \text{Abs-preal } (\text{add-set } (\text{Rep-preal } R) (\text{Rep-preal } S))$

definition

preal-diff-def:
 $R - S == \text{Abs-preal } (\text{diff-set } (\text{Rep-preal } R) (\text{Rep-preal } S))$

definition

preal-mult-def:
 $R * S == \text{Abs-preal } (\text{mult-set } (\text{Rep-preal } R) (\text{Rep-preal } S))$

definition

preal-inverse-def:
 $\text{inverse } R == \text{Abs-preal } (\text{inverse-set } (\text{Rep-preal } R))$

definition $R / S = R * \text{inverse } (S::\text{preal})$

definition

preal-one-def:
 $1 == \text{preal-of-rat } 1$

instance $\langle \text{proof} \rangle$

end

Reduces equality on abstractions to equality on representatives

declare *Abs-preal-inject* [simp]

declare *Abs-preal-inverse* [simp]

lemma *rat-mem-preal*: $0 < q ==> \{r::\text{rat}. 0 < r \ \& \ r < q\} \in \text{preal}$
 $\langle \text{proof} \rangle$

lemma *preal-nonempty*: $A \in \text{preal} ==> \exists x \in A. 0 < x$
 $\langle \text{proof} \rangle$

lemma *preal-Ex-mem*: $A \in \text{preal} ==> \exists x. x \in A$
 $\langle \text{proof} \rangle$

lemma *preal-imp-psubset-positives*: $A \in \text{preal} \implies A < \{r. 0 < r\}$
 $\langle \text{proof} \rangle$

lemma *preal-exists-bound*: $A \in \text{preal} \implies \exists x. 0 < x \ \& \ x \notin A$
 $\langle \text{proof} \rangle$

lemma *preal-exists-greater*: $[\![A \in \text{preal}; y \in A]\!] \implies \exists u \in A. y < u$
 $\langle \text{proof} \rangle$

lemma *preal-downwards-closed*: $[\![A \in \text{preal}; y \in A; 0 < z; z < y]\!] \implies z \in A$
 $\langle \text{proof} \rangle$

Relaxing the final premise

lemma *preal-downwards-closed'*:
 $[\![A \in \text{preal}; y \in A; 0 < z; z \leq y]\!] \implies z \in A$
 $\langle \text{proof} \rangle$

A positive fraction not in a positive real is an upper bound. Gleason p. 122
 - Remark (1)

lemma *not-in-preal-ub*:
assumes $A: A \in \text{preal}$
and $\text{not}x: x \notin A$
and $y: y \in A$
and $\text{pos}: 0 < x$
shows $y < x$
 $\langle \text{proof} \rangle$

preal lemmas instantiated to *Rep-preal X*

lemma *mem-Rep-preal-Ex*: $\exists x. x \in \text{Rep-preal } X$
 $\langle \text{proof} \rangle$

lemma *Rep-preal-exists-bound*: $\exists x > 0. x \notin \text{Rep-preal } X$
 $\langle \text{proof} \rangle$

lemmas *not-in-Rep-preal-ub* = *not-in-preal-ub* [OF *Rep-preal*]

51.1 *preal-of-prat*: the Injection from *prat* to *preal*

lemma *rat-less-set-mem-preal*: $0 < y \implies \{u::\text{rat}. 0 < u \ \& \ u < y\} \in \text{preal}$
 $\langle \text{proof} \rangle$

lemma *rat-subset-imp-le*:
 $[\![\{u::\text{rat}. 0 < u \ \& \ u < x\} \subseteq \{u. 0 < u \ \& \ u < y\}; 0 < x]\!] \implies x \leq y$
 $\langle \text{proof} \rangle$

lemma *rat-set-eq-imp-eq*:
 $[\![\{u::\text{rat}. 0 < u \ \& \ u < x\} = \{u. 0 < u \ \& \ u < y\}; 0 < x; 0 < y]\!] \implies x = y$

$\langle proof \rangle$

51.2 Properties of Ordering

instance *preal* :: *order*

$\langle proof \rangle$

lemma *preal-imp-pos*: $[|A \in preal; r \in A|] ==> 0 < r$

$\langle proof \rangle$

instance *preal* :: *linorder*

$\langle proof \rangle$

instantiation *preal* :: *distrib-lattice*

begin

definition

$(inf :: preal \Rightarrow preal \Rightarrow preal) = min$

definition

$(sup :: preal \Rightarrow preal \Rightarrow preal) = max$

instance

$\langle proof \rangle$

end

51.3 Properties of Addition

lemma *preal-add-commute*: $(x::preal) + y = y + x$

$\langle proof \rangle$

Lemmas for proving that addition of two positive reals gives a positive real

lemma *empty-psubset-nonempty*: $a \in A ==> \{\} \subset A$

$\langle proof \rangle$

Part 1 of Dedekind sections definition

lemma *add-set-not-empty*:

$[|A \in preal; B \in preal|] ==> \{\} \subset add-set\ A\ B$

$\langle proof \rangle$

Part 2 of Dedekind sections definition. A structured version of this proof is *preal-not-mem-mult-set-Ex* below.

lemma *preal-not-mem-add-set-Ex*:

$[|A \in preal; B \in preal|] ==> \exists q > 0. q \notin add-set\ A\ B$

$\langle proof \rangle$

lemma *add-set-not-rat-set*:

assumes *A*: $A \in preal$

and $B: B \in \text{preal}$
shows $\text{add-set } A \ B < \{r. \ 0 < r\}$
 $\langle \text{proof} \rangle$

Part 3 of Dedekind sections definition

lemma *add-set-lemma3*:
 $[|A \in \text{preal}; B \in \text{preal}; u \in \text{add-set } A \ B; 0 < z; z < u|]$
 $\implies z \in \text{add-set } A \ B$
 $\langle \text{proof} \rangle$

Part 4 of Dedekind sections definition

lemma *add-set-lemma4*:
 $[|A \in \text{preal}; B \in \text{preal}; y \in \text{add-set } A \ B|] \implies \exists u \in \text{add-set } A \ B. \ y < u$
 $\langle \text{proof} \rangle$

lemma *mem-add-set*:
 $[|A \in \text{preal}; B \in \text{preal}|] \implies \text{add-set } A \ B \in \text{preal}$
 $\langle \text{proof} \rangle$

lemma *preal-add-assoc*: $((x::\text{preal}) + y) + z = x + (y + z)$
 $\langle \text{proof} \rangle$

instance *preal :: ab-semigroup-add*
 $\langle \text{proof} \rangle$

lemma *preal-add-left-commute*: $x + (y + z) = y + ((x + z)::\text{preal})$
 $\langle \text{proof} \rangle$

Positive Real addition is an AC operator

lemmas *preal-add-ac = preal-add-assoc preal-add-commute preal-add-left-commute*

51.4 Properties of Multiplication

Proofs essentially same as for addition

lemma *preal-mult-commute*: $(x::\text{preal}) * y = y * x$
 $\langle \text{proof} \rangle$

Multiplication of two positive reals gives a positive real.

Lemmas for proving positive reals multiplication set in *preal*

Part 1 of Dedekind sections definition

lemma *mult-set-not-empty*:
 $[|A \in \text{preal}; B \in \text{preal}|] \implies \{\} \subset \text{mult-set } A \ B$
 $\langle \text{proof} \rangle$

Part 2 of Dedekind sections definition

lemma *preal-not-mem-mult-set-Ex*:

assumes $A: A \in \text{preal}$
and $B: B \in \text{preal}$
shows $\exists q. 0 < q \ \& \ q \notin \text{mult-set } A \ B$
 $\langle \text{proof} \rangle$

lemma *mult-set-not-rat-set*:
assumes $A: A \in \text{preal}$
and $B: B \in \text{preal}$
shows $\text{mult-set } A \ B < \{r. 0 < r\}$
 $\langle \text{proof} \rangle$

Part 3 of Dedekind sections definition

lemma *mult-set-lemma3*:
 $[[A \in \text{preal}; B \in \text{preal}; u \in \text{mult-set } A \ B; 0 < z; z < u]]$
 $\implies z \in \text{mult-set } A \ B$
 $\langle \text{proof} \rangle$

Part 4 of Dedekind sections definition

lemma *mult-set-lemma4*:
 $[[A \in \text{preal}; B \in \text{preal}; y \in \text{mult-set } A \ B]] \implies \exists u \in \text{mult-set } A \ B. y < u$
 $\langle \text{proof} \rangle$

lemma *mem-mult-set*:
 $[[A \in \text{preal}; B \in \text{preal}]] \implies \text{mult-set } A \ B \in \text{preal}$
 $\langle \text{proof} \rangle$

lemma *preal-mult-assoc*: $((x::\text{preal}) * y) * z = x * (y * z)$
 $\langle \text{proof} \rangle$

instance *preal :: ab-semigroup-mult*
 $\langle \text{proof} \rangle$

lemma *preal-mult-left-commute*: $x * (y * z) = y * ((x * z)::\text{preal})$
 $\langle \text{proof} \rangle$

Positive Real multiplication is an AC operator

lemmas *preal-mult-ac* =
 $\text{preal-mult-assoc } \text{preal-mult-commute } \text{preal-mult-left-commute}$

Positive real 1 is the multiplicative identity element

lemma *preal-mult-1*: $(1::\text{preal}) * z = z$
 $\langle \text{proof} \rangle$

instance *preal :: comm-monoid-mult*
 $\langle \text{proof} \rangle$

lemma *preal-mult-1-right*: $z * (1::\text{preal}) = z$
 $\langle \text{proof} \rangle$

51.5 Distribution of Multiplication across Addition

lemma *mem-Rep-preal-add-iff*:

$(z \in \text{Rep-preal}(R+S)) = (\exists x \in \text{Rep-preal } R. \exists y \in \text{Rep-preal } S. z = x + y)$
 $\langle \text{proof} \rangle$

lemma *mem-Rep-preal-mult-iff*:

$(z \in \text{Rep-preal}(R*S)) = (\exists x \in \text{Rep-preal } R. \exists y \in \text{Rep-preal } S. z = x * y)$
 $\langle \text{proof} \rangle$

lemma *distrib-subset1*:

$\text{Rep-preal } (w * (x + y)) \subseteq \text{Rep-preal } (w * x + w * y)$
 $\langle \text{proof} \rangle$

lemma *preal-add-mult-distrib-mean*:

assumes $a: a \in \text{Rep-preal } w$
and $b: b \in \text{Rep-preal } w$
and $d: d \in \text{Rep-preal } x$
and $e: e \in \text{Rep-preal } y$
shows $\exists c \in \text{Rep-preal } w. a * d + b * e = c * (d + e)$
 $\langle \text{proof} \rangle$

lemma *distrib-subset2*:

$\text{Rep-preal } (w * x + w * y) \subseteq \text{Rep-preal } (w * (x + y))$
 $\langle \text{proof} \rangle$

lemma *preal-add-mult-distrib2*: $(w * ((x::\text{preal}) + y)) = (w * x) + (w * y)$
 $\langle \text{proof} \rangle$

lemma *preal-add-mult-distrib*: $((x::\text{preal}) + y) * w = (x * w) + (y * w)$
 $\langle \text{proof} \rangle$

instance *preal :: comm-semiring*

$\langle \text{proof} \rangle$

51.6 Existence of Inverse, a Positive Real

lemma *mem-inv-set-ex*:

assumes $A: A \in \text{preal}$ **shows** $\exists x y. 0 < x \ \& \ x < y \ \& \ \text{inverse } y \notin A$
 $\langle \text{proof} \rangle$

Part 1 of Dedekind sections definition

lemma *inverse-set-not-empty*:

$A \in \text{preal} ==> \{\} \subset \text{inverse-set } A$
 $\langle \text{proof} \rangle$

Part 2 of Dedekind sections definition

lemma *preal-not-mem-inverse-set-Ex*:

assumes $A: A \in \text{preal}$ **shows** $\exists q. 0 < q \ \& \ q \notin \text{inverse-set } A$
 $\langle \text{proof} \rangle$

lemma *inverse-set-not-rat-set*:

assumes $A: A \in \text{preal}$ **shows** $\text{inverse-set } A < \{r. 0 < r\}$
 $\langle \text{proof} \rangle$

Part 3 of Dedekind sections definition

lemma *inverse-set-lemma3*:

$[|A \in \text{preal}; u \in \text{inverse-set } A; 0 < z; z < u|]$
 $\implies z \in \text{inverse-set } A$
 $\langle \text{proof} \rangle$

Part 4 of Dedekind sections definition

lemma *inverse-set-lemma4*:

$[|A \in \text{preal}; y \in \text{inverse-set } A|] \implies \exists u \in \text{inverse-set } A. y < u$
 $\langle \text{proof} \rangle$

lemma *mem-inverse-set*:

$A \in \text{preal} \implies \text{inverse-set } A \in \text{preal}$
 $\langle \text{proof} \rangle$

51.7 Gleason’s Lemma 9-3.4, page 122

lemma *Gleason9-34-exists*:

assumes $A: A \in \text{preal}$
and $\forall x \in A. x + u \in A$
and $0 \leq z$
shows $\exists b \in A. b + (\text{of-int } z) * u \in A$
 $\langle \text{proof} \rangle$

lemma *Gleason9-34-contr*:

assumes $A: A \in \text{preal}$
shows $[|\forall x \in A. x + u \in A; 0 < u; 0 < y; y \notin A|] \implies \text{False}$
 $\langle \text{proof} \rangle$

lemma *Gleason9-34*:

assumes $A: A \in \text{preal}$
and $\text{upos}: 0 < u$
shows $\exists r \in A. r + u \notin A$
 $\langle \text{proof} \rangle$

51.8 Gleason’s Lemma 9-3.6

lemma *lemma-gleason9-36*:

assumes $A: A \in \text{preal}$
and $x: 1 < x$
shows $\exists r \in A. r * x \notin A$
 $\langle \text{proof} \rangle$

51.9 Existence of Inverse: Part 2

lemma *mem-Rep-preal-inverse-iff*:

$(z \in \text{Rep-preal}(\text{inverse } R)) =$
 $(0 < z \wedge (\exists y. z < y \wedge \text{inverse } y \notin \text{Rep-preal } R))$
 ⟨proof⟩

lemma *Rep-preal-of-rat*:

$0 < q ==> \text{Rep-preal}(\text{preal-of-rat } q) = \{x. 0 < x \wedge x < q\}$
 ⟨proof⟩

lemma *subset-inverse-mult-lemma*:

assumes *xpos*: $0 < x$ **and** *xless*: $x < 1$
shows $\exists r u y. 0 < r \ \& \ r < y \ \& \ \text{inverse } y \notin \text{Rep-preal } R \ \& \$
 $u \in \text{Rep-preal } R \ \& \ x = r * u$
 ⟨proof⟩

lemma *subset-inverse-mult*:

$\text{Rep-preal}(\text{preal-of-rat } 1) \subseteq \text{Rep-preal}(\text{inverse } R * R)$
 ⟨proof⟩

lemma *inverse-mult-subset-lemma*:

assumes *rpos*: $0 < r$
and *rless*: $r < y$
and *notin*: $\text{inverse } y \notin \text{Rep-preal } R$
and *q*: $q \in \text{Rep-preal } R$
shows $r * q < 1$
 ⟨proof⟩

lemma *inverse-mult-subset*:

$\text{Rep-preal}(\text{inverse } R * R) \subseteq \text{Rep-preal}(\text{preal-of-rat } 1)$
 ⟨proof⟩

lemma *preal-mult-inverse*: $\text{inverse } R * R = (1::\text{preal})$

⟨proof⟩

lemma *preal-mult-inverse-right*: $R * \text{inverse } R = (1::\text{preal})$

⟨proof⟩

Theorems needing *Gleason9-34*

lemma *Rep-preal-self-subset*: $\text{Rep-preal } (R) \subseteq \text{Rep-preal}(R + S)$

⟨proof⟩

lemma *Rep-preal-sum-not-subset*: $\sim \text{Rep-preal } (R + S) \subseteq \text{Rep-preal}(R)$

⟨proof⟩

lemma *Rep-preal-sum-not-eq*: $\text{Rep-preal } (R + S) \neq \text{Rep-preal}(R)$

⟨proof⟩

at last, Gleason prop. 9-3.5(iii) page 123

lemma *preal-self-less-add-left*: $(R::preal) < R + S$
 $\langle proof \rangle$

lemma *preal-self-less-add-right*: $(R::preal) < S + R$
 $\langle proof \rangle$

lemma *preal-not-eq-self*: $x \neq x + (y::preal)$
 $\langle proof \rangle$

51.10 Subtraction for Positive Reals

Gleason prop. 9-3.5(iv), page 123: proving $A < B \implies \exists D. A + D = B$.
 We define the claimed D and show that it is a positive real

Part 1 of Dedekind sections definition

lemma *diff-set-not-empty*:
 $R < S \implies \{\} \subset \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R)$
 $\langle proof \rangle$

Part 2 of Dedekind sections definition

lemma *diff-set-nonempty*:
 $\exists q. 0 < q \ \& \ q \notin \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R)$
 $\langle proof \rangle$

lemma *diff-set-not-rat-set*:
 $\text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R) < \{r. 0 < r\} \text{ (is ?lhs < ?rhs)}$
 $\langle proof \rangle$

Part 3 of Dedekind sections definition

lemma *diff-set-lemma3*:
 $[|R < S; u \in \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R); 0 < z; z < u|]$
 $\implies z \in \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R)$
 $\langle proof \rangle$

Part 4 of Dedekind sections definition

lemma *diff-set-lemma4*:
 $[|R < S; y \in \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R)|]$
 $\implies \exists u \in \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R). y < u$
 $\langle proof \rangle$

lemma *mem-diff-set*:
 $R < S \implies \text{diff-set } (Rep\text{-preal } S) (Rep\text{-preal } R) \in preal$
 $\langle proof \rangle$

lemma *mem-Rep-preal-diff-iff*:
 $R < S \implies$
 $(z \in Rep\text{-preal}(S - R)) =$
 $(\exists x. 0 < x \ \& \ 0 < z \ \& \ x \notin Rep\text{-preal } R \ \& \ x + z \in Rep\text{-preal } S)$

$\langle proof \rangle$

proving that $R + D \leq S$

lemma *less-add-left-lemma*:

assumes *Rless*: $R < S$
and *a*: $a \in \text{Rep-preal } R$
and *cb*: $c + b \in \text{Rep-preal } S$
and $c \notin \text{Rep-preal } R$
and $0 < b$
and $0 < c$

shows $a + b \in \text{Rep-preal } S$

$\langle proof \rangle$

lemma *less-add-left-le1*:

$R < (S::\text{preal}) \implies R + (S - R) \leq S$

$\langle proof \rangle$

51.11 proving that $S \leq R + D$ — trickier

lemma *lemma-sum-mem-Rep-preal-ex*:

$x \in \text{Rep-preal } S \implies \exists e. 0 < e \ \& \ x + e \in \text{Rep-preal } S$

$\langle proof \rangle$

lemma *less-add-left-lemma2*:

assumes *Rless*: $R < S$
and *x*: $x \in \text{Rep-preal } S$
and *xnot*: $x \notin \text{Rep-preal } R$
shows $\exists u \ v \ z. 0 < v \ \& \ 0 < z \ \& \ u \in \text{Rep-preal } R \ \& \ z \notin \text{Rep-preal } R \ \& \ z + v \in \text{Rep-preal } S \ \& \ x = u + v$

$\langle proof \rangle$

lemma *less-add-left-le2*: $R < (S::\text{preal}) \implies S \leq R + (S - R)$

$\langle proof \rangle$

lemma *less-add-left*: $R < (S::\text{preal}) \implies R + (S - R) = S$

$\langle proof \rangle$

lemma *less-add-left-Ex*: $R < (S::\text{preal}) \implies \exists D. R + D = S$

$\langle proof \rangle$

lemma *preal-add-less2-mono1*: $R < (S::\text{preal}) \implies R + T < S + T$

$\langle proof \rangle$

lemma *preal-add-less2-mono2*: $R < (S::\text{preal}) \implies T + R < T + S$

$\langle proof \rangle$

lemma *preal-add-right-less-cancel*: $R + T < S + T \implies R < (S::\text{preal})$

$\langle proof \rangle$

lemma *preal-add-left-less-cancel*: $T + R < T + S \implies R < (S::preal)$
 $\langle proof \rangle$

lemma *preal-add-less-cancel-right*: $((R::preal) + T < S + T) = (R < S)$
 $\langle proof \rangle$

lemma *preal-add-less-cancel-left*: $(T + (R::preal) < T + S) = (R < S)$
 $\langle proof \rangle$

lemma *preal-add-le-cancel-right*: $((R::preal) + T \leq S + T) = (R \leq S)$
 $\langle proof \rangle$

lemma *preal-add-le-cancel-left*: $(T + (R::preal) \leq T + S) = (R \leq S)$
 $\langle proof \rangle$

lemma *preal-add-less-mono*:
 $\llbracket x1 < y1; x2 < y2 \rrbracket \implies x1 + x2 < y1 + (y2::preal)$
 $\langle proof \rangle$

lemma *preal-add-right-cancel*: $(R::preal) + T = S + T \implies R = S$
 $\langle proof \rangle$

lemma *preal-add-left-cancel*: $C + A = C + B \implies A = (B::preal)$
 $\langle proof \rangle$

lemma *preal-add-left-cancel-iff*: $(C + A = C + B) = ((A::preal) = B)$
 $\langle proof \rangle$

lemma *preal-add-right-cancel-iff*: $(A + C = B + C) = ((A::preal) = B)$
 $\langle proof \rangle$

lemmas *preal-cancels* =
preal-add-less-cancel-right preal-add-less-cancel-left
preal-add-le-cancel-right preal-add-le-cancel-left
preal-add-left-cancel-iff preal-add-right-cancel-iff

instance *preal* :: *ordered-cancel-ab-semigroup-add*
 $\langle proof \rangle$

51.12 Completeness of type *preal*

Prove that supremum is a cut

Part 1 of Dedekind sections definition

lemma *preal-sup-set-not-empty*:
 $P \neq \{\} \implies \{\} \subset (\bigcup X \in P. \text{Rep-}preal(X))$
 $\langle proof \rangle$

Part 2 of Dedekind sections definition

lemma *preal-sup-not-exists*:

$\forall X \in P. X \leq Y \implies \exists q. 0 < q \ \& \ q \notin (\bigcup X \in P. \text{Rep-preal}(X))$
 $\langle \text{proof} \rangle$

lemma *preal-sup-set-not-rat-set*:

$\forall X \in P. X \leq Y \implies (\bigcup X \in P. \text{Rep-preal}(X)) < \{r. 0 < r\}$
 $\langle \text{proof} \rangle$

Part 3 of Dedekind sections definition

lemma *preal-sup-set-lemma3*:

$[[P \neq \{\}; \forall X \in P. X \leq Y; u \in (\bigcup X \in P. \text{Rep-preal}(X)); 0 < z; z < u]]$
 $\implies z \in (\bigcup X \in P. \text{Rep-preal}(X))$
 $\langle \text{proof} \rangle$

Part 4 of Dedekind sections definition

lemma *preal-sup-set-lemma4*:

$[[P \neq \{\}; \forall X \in P. X \leq Y; y \in (\bigcup X \in P. \text{Rep-preal}(X)) \]]$
 $\implies \exists u \in (\bigcup X \in P. \text{Rep-preal}(X)). y < u$
 $\langle \text{proof} \rangle$

lemma *preal-sup*:

$[[P \neq \{\}; \forall X \in P. X \leq Y]] \implies (\bigcup X \in P. \text{Rep-preal}(X)) \in \text{preal}$
 $\langle \text{proof} \rangle$

lemma *preal-psup-le*:

$[[\forall X \in P. X \leq Y; x \in P \]] \implies x \leq \text{psup } P$
 $\langle \text{proof} \rangle$

lemma *psup-le-ub*: $[[P \neq \{\}; \forall X \in P. X \leq Y \]] \implies \text{psup } P \leq Y$

$\langle \text{proof} \rangle$

Supremum property

lemma *preal-complete*:

$[[P \neq \{\}; \forall X \in P. X \leq Y \]] \implies (\exists X \in P. Z < X) = (Z < \text{psup } P)$
 $\langle \text{proof} \rangle$

51.13 The Embedding from *rat* into *preal*

lemma *preal-of-rat-add-lemma1*:

$[[x < y + z; 0 < x; 0 < y]] \implies x * y * \text{inverse } (y + z) < (y::\text{rat})$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-add-lemma2*:

assumes $u < x + y$
and $0 < x$
and $0 < y$
and $0 < u$
shows $\exists v w::\text{rat}. w < y \ \& \ 0 < v \ \& \ v < x \ \& \ 0 < w \ \& \ u = v + w$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-add*:

$[[\ 0 < x; \ 0 < y]]$
 $\implies \text{preal-of-rat } ((x::\text{rat}) + y) = \text{preal-of-rat } x + \text{preal-of-rat } y$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-mult-lemma1*:

$[[x < y; \ 0 < x; \ 0 < z]] \implies x * z * \text{inverse } y < (z::\text{rat})$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-mult-lemma2*:

assumes *xless*: $x < y * z$
and *xpos*: $0 < x$
and *ypos*: $0 < y$
shows $x * z * \text{inverse } y * \text{inverse } z < (z::\text{rat})$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-mult-lemma3*:

assumes *uless*: $u < x * y$
and $0 < x$
and $0 < y$
and $0 < u$
shows $\exists v \ w::\text{rat}. \ v < x \ \& \ w < y \ \& \ 0 < v \ \& \ 0 < w \ \& \ u = v * w$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-mult*:

$[[\ 0 < x; \ 0 < y]]$
 $\implies \text{preal-of-rat } ((x::\text{rat}) * y) = \text{preal-of-rat } x * \text{preal-of-rat } y$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-less-iff*:

$[[\ 0 < x; \ 0 < y]] \implies (\text{preal-of-rat } x < \text{preal-of-rat } y) = (x < y)$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-le-iff*:

$[[\ 0 < x; \ 0 < y]] \implies (\text{preal-of-rat } x \leq \text{preal-of-rat } y) = (x \leq y)$
 $\langle \text{proof} \rangle$

lemma *preal-of-rat-eq-iff*:

$[[\ 0 < x; \ 0 < y]] \implies (\text{preal-of-rat } x = \text{preal-of-rat } y) = (x = y)$
 $\langle \text{proof} \rangle$

end

52 RealDef: Defining the Reals from the Positive Reals

```

theory RealDef
imports PReal
uses (Tools/real-arith.ML)
begin

```

definition

```

  realrel  :: ((preal * preal) * (preal * preal)) set where
  [code del]: realrel = {p.  $\exists x1\ y1\ x2\ y2. p = ((x1,y1),(x2,y2)) \ \& \ x1+y2 = x2+y1$ }

```

```

typedef (Real)  real = UNIV//realrel
  <proof>

```

definition

```

  real-of-preal :: preal => real where
  [code del]: real-of-preal m = Abs-Real (realrel “ {(m + 1, 1)}”)

```

```

instantiation real :: {zero, one, plus, minus, uminus, times, inverse, ord, abs,
sgn}
begin

```

definition

```

  real-zero-def [code del]: 0 = Abs-Real(realrel“{(1, 1)}”)

```

definition

```

  real-one-def [code del]: 1 = Abs-Real(realrel“{(1 + 1, 1)}”)

```

definition

```

  real-add-def [code del]: z + w =
    contents ( $\bigcup (x,y) \in \text{Rep-Real}(z). \bigcup (u,v) \in \text{Rep-Real}(w). \{ \text{Abs-Real}(\text{realrel“}\{(x+u, y+v)\}) \}$ )

```

definition

```

  real-minus-def [code del]: - r = contents ( $\bigcup (x,y) \in \text{Rep-Real}(r). \{ \text{Abs-Real}(\text{realrel“}\{(y,x)\}) \}$ )

```

definition

```

  real-diff-def [code del]: r - (s::real) = r + - s

```

definition

```

  real-mult-def [code del]:
    z * w =
    contents ( $\bigcup (x,y) \in \text{Rep-Real}(z). \bigcup (u,v) \in \text{Rep-Real}(w). \{ \text{Abs-Real}(\text{realrel“}\{(x*u + y*v, x*v + y*u)\}) \}$ )

```

definition

real-inverse-def [code del]: $\text{inverse } (R::\text{real}) = (\text{THE } S. (R = 0 \ \& \ S = 0) \mid S * R = 1)$

definition

real-divide-def [code del]: $R / (S::\text{real}) = R * \text{inverse } S$

definition

real-le-def [code del]: $z \leq (w::\text{real}) \longleftrightarrow (\exists x \ y \ u \ v. x+v \leq u+y \ \& \ (x,y) \in \text{Rep-Real } z \ \& \ (u,v) \in \text{Rep-Real } w)$

definition

real-less-def [code del]: $x < (y::\text{real}) \longleftrightarrow x \leq y \wedge x \neq y$

definition

real-abs-def: $\text{abs } (r::\text{real}) = (\text{if } r < 0 \text{ then } - r \text{ else } r)$

definition

real-sgn-def: $\text{sgn } (x::\text{real}) = (\text{if } x=0 \text{ then } 0 \text{ else if } 0<x \text{ then } 1 \text{ else } -1)$

instance $\langle \text{proof} \rangle$

end

52.1 Equivalence relation over positive reals

lemma *preal-trans-lemma*:

assumes $x + y1 = x1 + y$

and $x + y2 = x2 + y$

shows $x1 + y2 = x2 + (y1::\text{preal})$

$\langle \text{proof} \rangle$

lemma *realrel-iff* [simp]: $((x1,y1),(x2,y2)) \in \text{realrel} = (x1 + y2 = x2 + y1)$

$\langle \text{proof} \rangle$

lemma *equiv-realrel*: *equiv UNIV realrel*

$\langle \text{proof} \rangle$

Reduces equality of equivalence classes to the *realrel* relation: $(\text{realrel} \text{ “ } \{x\} = \text{realrel} \text{ “ } \{y\}) = ((x, y) \in \text{realrel})$

lemmas *equiv-realrel-iff* =

eq-equiv-class-iff [OF *equiv-realrel UNIV-I UNIV-I*]

declare *equiv-realrel-iff* [simp]

lemma *realrel-in-real* [simp]: $\text{realrel} \text{ “ } \{(x,y)\}: \text{Real}$

$\langle \text{proof} \rangle$

declare *Abs-Real-inject* [simp]
declare *Abs-Real-inverse* [simp]

Case analysis on the representation of a real number as an equivalence class of pairs of positive reals.

lemma *eq-Abs-Real* [case-names *Abs-Real*, cases type: *real*]:

$$(!x\ y.\ z = \text{Abs-Real}(\text{realrel}\{\{(x,y)\}\}) \implies P) \implies P$$

 <proof>

52.2 Addition and Subtraction

lemma *real-add-congruent2-lemma*:

$$[| a + ba = aa + b; ab + bc = ac + bb |]$$

$$\implies a + ab + (ba + bc) = aa + ac + (b + (bb::preal))$$

 <proof>

lemma *real-add*:

$$\text{Abs-Real}(\text{realrel}\{\{(x,y)\}\}) + \text{Abs-Real}(\text{realrel}\{\{(u,v)\}\}) =$$

$$\text{Abs-Real}(\text{realrel}\{\{(x+u, y+v)\}\})$$

 <proof>

lemma *real-minus*: $-\text{Abs-Real}(\text{realrel}\{\{(x,y)\}\}) = \text{Abs-Real}(\text{realrel}\{\{(y,x)\}\})$
 <proof>

instance *real* :: *ab-group-add*
 <proof>

52.3 Multiplication

lemma *real-mult-congruent2-lemma*:

$$!!(x1::preal). [| x1 + y2 = x2 + y1 |] \implies$$

$$x * x1 + y * y1 + (x * y2 + y * x2) =$$

$$x * x2 + y * y2 + (x * y1 + y * x1)$$

 <proof>

lemma *real-mult-congruent2*:

$$(\%p1\ p2.$$

$$(\%(x1,y1). (\%(x2,y2).$$

$$\{ \text{Abs-Real}(\text{realrel}\{\{(x1*x2 + y1*y2, x1*y2+y1*x2)\}\}) \})\ p2)\ p1)$$

$$\text{respects2 realrel}$$

 <proof>

lemma *real-mult*:

$$\text{Abs-Real}((\text{realrel}\{\{(x1,y1)\}\}) * \text{Abs-Real}((\text{realrel}\{\{(x2,y2)\}\})) =$$

$$\text{Abs-Real}(\text{realrel}\{\{(x1*x2+y1*y2, x1*y2+y1*x2)\}\})$$

 <proof>

lemma *real-mult-commute*: $(z::real) * w = w * z$
 <proof>

lemma *real-mult-assoc*: $((z1::real) * z2) * z3 = z1 * (z2 * z3)$
 $\langle proof \rangle$

lemma *real-mult-1*: $(1::real) * z = z$
 $\langle proof \rangle$

lemma *real-add-mult-distrib*: $((z1::real) + z2) * w = (z1 * w) + (z2 * w)$
 $\langle proof \rangle$

one and zero are distinct

lemma *real-zero-not-eq-one*: $0 \neq (1::real)$
 $\langle proof \rangle$

instance *real* :: *comm-ring-1*
 $\langle proof \rangle$

52.4 Inverse and Division

lemma *real-zero-iff*: $Abs\text{-}Real\ (realrel\ \{\{(x, x)\}\}) = 0$
 $\langle proof \rangle$

Instead of using an existential quantifier and constructing the inverse within the proof, we could define the inverse explicitly.

lemma *real-mult-inverse-left-ex*: $x \neq 0 \implies \exists y. y * x = (1::real)$
 $\langle proof \rangle$

lemma *real-mult-inverse-left*: $x \neq 0 \implies inverse(x) * x = (1::real)$
 $\langle proof \rangle$

52.5 The Real Numbers form a Field

instance *real* :: *field*
 $\langle proof \rangle$

Inverse of zero! Useful to simplify certain equations

lemma *INVERSE-ZERO*: $inverse\ 0 = (0::real)$
 $\langle proof \rangle$

instance *real* :: *division-by-zero*
 $\langle proof \rangle$

52.6 The \leq Ordering

lemma *real-le-reft*: $w \leq (w::real)$
 $\langle proof \rangle$

The arithmetic decision procedure is not set up for type *preal*. This lemma is currently unused, but it could simplify the proofs of the following two lemmas.

lemma *preal-eq-le-imp-le*:

assumes *eq*: $a+b = c+d$ **and** *le*: $c \leq a$

shows $b \leq (d::preal)$

$\langle proof \rangle$

lemma *real-le-lemma*:

assumes *l*: $u1 + v2 \leq u2 + v1$

and $x1 + v1 = u1 + y1$

and $x2 + v2 = u2 + y2$

shows $x1 + y2 \leq x2 + (y1::preal)$

$\langle proof \rangle$

lemma *real-le*:

$(Abs-Real(realrel``\{(x1,y1)\}) \leq Abs-Real(realrel``\{(x2,y2)\})) =$
 $(x1 + y2 \leq x2 + y1)$

$\langle proof \rangle$

lemma *real-le-anti-sym*: $[| z \leq w; w \leq z |] ==> z = (w::real)$

$\langle proof \rangle$

lemma *real-trans-lemma*:

assumes $x + v \leq u + y$

and $u + v' \leq u' + v$

and $x2 + v2 = u2 + y2$

shows $x + v' \leq u' + (y::preal)$

$\langle proof \rangle$

lemma *real-le-trans*: $[| i \leq j; j \leq k |] ==> i \leq (k::real)$

$\langle proof \rangle$

instance *real* :: *order*

$\langle proof \rangle$

lemma *real-le-linear*: $(z::real) \leq w \mid w \leq z$

$\langle proof \rangle$

instance *real* :: *linorder*

$\langle proof \rangle$

lemma *real-le-eq-diff*: $(x \leq y) = (x-y \leq (0::real))$

$\langle proof \rangle$

lemma *real-add-left-mono*:

assumes *le*: $x \leq y$ **shows** $z + x \leq z + (y::real)$

$\langle proof \rangle$

lemma *real-sum-gt-zero-less*: $(0 < S + (-W::real)) ==> (W < S)$

$\langle proof \rangle$

lemma *real-less-sum-gt-zero*: $(W < S) ==> (0 < S + (-W::real))$
 $\langle proof \rangle$

lemma *real-mult-order*: $[| 0 < x; 0 < y |] ==> (0::real) < x * y$
 $\langle proof \rangle$

lemma *real-mult-less-mono2*: $[| (0::real) < z; x < y |] ==> z * x < z * y$
 $\langle proof \rangle$

instantiation *real* :: *distrib-lattice*
begin

definition
 $(inf :: real \Rightarrow real \Rightarrow real) = min$

definition
 $(sup :: real \Rightarrow real \Rightarrow real) = max$

instance
 $\langle proof \rangle$

end

52.7 The Reals Form an Ordered Field

instance *real* :: *ordered-field*
 $\langle proof \rangle$

instance *real* :: *lordered-ab-group-add* $\langle proof \rangle$

The function *real-of-preal* requires many proofs, but it seems to be essential for proving completeness of the reals from that of the positive reals.

lemma *real-of-preal-add*:
 $real-of-preal ((x::preal) + y) = real-of-preal x + real-of-preal y$
 $\langle proof \rangle$

lemma *real-of-preal-mult*:
 $real-of-preal ((x::preal) * y) = real-of-preal x * real-of-preal y$
 $\langle proof \rangle$

Gleason prop 9-4.4 p 127

lemma *real-of-preal-trichotomy*:
 $\exists m. (x::real) = real-of-preal m \mid x = 0 \mid x = -(real-of-preal m)$
 $\langle proof \rangle$

lemma *real-of-preal-leD*:
 $real-of-preal m1 \leq real-of-preal m2 ==> m1 \leq m2$

$\langle \text{proof} \rangle$

lemma *real-of-preal-lessI*: $m1 < m2 \implies \text{real-of-preal } m1 < \text{real-of-preal } m2$
 $\langle \text{proof} \rangle$

lemma *real-of-preal-lessD*:
 $\text{real-of-preal } m1 < \text{real-of-preal } m2 \implies m1 < m2$
 $\langle \text{proof} \rangle$

lemma *real-of-preal-less-iff [simp]*:
 $(\text{real-of-preal } m1 < \text{real-of-preal } m2) = (m1 < m2)$
 $\langle \text{proof} \rangle$

lemma *real-of-preal-le-iff*:
 $(\text{real-of-preal } m1 \leq \text{real-of-preal } m2) = (m1 \leq m2)$
 $\langle \text{proof} \rangle$

lemma *real-of-preal-zero-less*: $0 < \text{real-of-preal } m$
 $\langle \text{proof} \rangle$

lemma *real-of-preal-minus-less-zero*: $-\text{real-of-preal } m < 0$
 $\langle \text{proof} \rangle$

lemma *real-of-preal-not-minus-gt-zero*: $\sim 0 < -\text{real-of-preal } m$
 $\langle \text{proof} \rangle$

52.8 Theorems About the Ordering

lemma *real-gt-zero-preal-Ex*: $(0 < x) = (\exists y. x = \text{real-of-preal } y)$
 $\langle \text{proof} \rangle$

lemma *real-gt-preal-preal-Ex*:
 $\text{real-of-preal } z < x \implies \exists y. x = \text{real-of-preal } y$
 $\langle \text{proof} \rangle$

lemma *real-ge-preal-preal-Ex*:
 $\text{real-of-preal } z \leq x \implies \exists y. x = \text{real-of-preal } y$
 $\langle \text{proof} \rangle$

lemma *real-less-all-preal*: $y \leq 0 \implies \forall x. y < \text{real-of-preal } x$
 $\langle \text{proof} \rangle$

lemma *real-less-all-real2*: $\sim 0 < y \implies \forall x. y < \text{real-of-preal } x$
 $\langle \text{proof} \rangle$

52.9 More Lemmas

lemma *real-mult-left-cancel*: $(c::\text{real}) \neq 0 \implies (c*a=c*b) = (a=b)$
 $\langle \text{proof} \rangle$

lemma *real-mult-right-cancel*: $(c::\text{real}) \neq 0 \implies (a*c=b*c) = (a=b)$
 $\langle \text{proof} \rangle$

lemma *real-mult-less-iff1* [simp]: $(0::\text{real}) < z \implies (x*z < y*z) = (x < y)$
 $\langle \text{proof} \rangle$

lemma *real-mult-le-cancel-iff1* [simp]: $(0::\text{real}) < z \implies (x*z \leq y*z) = (x \leq y)$
 $\langle \text{proof} \rangle$

lemma *real-mult-le-cancel-iff2* [simp]: $(0::\text{real}) < z \implies (z*x \leq z*y) = (x \leq y)$
 $\langle \text{proof} \rangle$

lemma *real-inverse-gt-one*: $[(0::\text{real}) < x; x < 1] \implies 1 < \text{inverse } x$
 $\langle \text{proof} \rangle$

52.10 Embedding numbers into the Reals

abbreviation

real-of-nat :: $\text{nat} \Rightarrow \text{real}$

where

real-of-nat \equiv *of-nat*

abbreviation

real-of-int :: $\text{int} \Rightarrow \text{real}$

where

real-of-int \equiv *of-int*

abbreviation

real-of-rat :: $\text{rat} \Rightarrow \text{real}$

where

real-of-rat \equiv *of-rat*

consts

real :: $'a \Rightarrow \text{real}$

defs (overloaded)

real-of-nat-def [code unfold]: $\text{real} == \text{real-of-nat}$

real-of-int-def [code unfold]: $\text{real} == \text{real-of-int}$

lemma *real-eq-of-nat*: $\text{real} = \text{of-nat}$
 $\langle \text{proof} \rangle$

lemma *real-eq-of-int*: $\text{real} = \text{of-int}$
 $\langle \text{proof} \rangle$

lemma *real-of-int-zero* [simp]: $\text{real } (0::\text{int}) = 0$
 $\langle \text{proof} \rangle$

lemma *real-of-one* [simp]: $\text{real } (1::\text{int}) = (1::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-of-int-add* [simp]: $\text{real}(x + y) = \text{real } (x::\text{int}) + \text{real } y$
 $\langle \text{proof} \rangle$

lemma *real-of-int-minus* [simp]: $\text{real}(-x) = -\text{real } (x::\text{int})$
 $\langle \text{proof} \rangle$

lemma *real-of-int-diff* [simp]: $\text{real}(x - y) = \text{real } (x::\text{int}) - \text{real } y$
 $\langle \text{proof} \rangle$

lemma *real-of-int-mult* [simp]: $\text{real}(x * y) = \text{real } (x::\text{int}) * \text{real } y$
 $\langle \text{proof} \rangle$

lemma *real-of-int-setsum* [simp]: $\text{real } ((\text{SUM } x:A. f x)::\text{int}) = (\text{SUM } x:A. \text{real}(f x))$
 $\langle \text{proof} \rangle$

lemma *real-of-int-setprod* [simp]: $\text{real } ((\text{PROD } x:A. f x)::\text{int}) = (\text{PROD } x:A. \text{real}(f x))$
 $\langle \text{proof} \rangle$

lemma *real-of-int-zero-cancel* [simp, algebra, presburger]: $(\text{real } x = 0) = (x = (0::\text{int}))$
 $\langle \text{proof} \rangle$

lemma *real-of-int-inject* [iff, algebra, presburger]: $(\text{real } (x::\text{int}) = \text{real } y) = (x = y)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-less-iff* [iff, presburger]: $(\text{real } (x::\text{int}) < \text{real } y) = (x < y)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-le-iff* [simp, presburger]: $(\text{real } (x::\text{int}) \leq \text{real } y) = (x \leq y)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-gt-zero-cancel-iff* [simp, presburger]: $(0 < \text{real } (n::\text{int})) = (0 < n)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-ge-zero-cancel-iff* [simp, presburger]: $(0 \leq \text{real } (n::\text{int})) = (0 \leq n)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-lt-zero-cancel-iff* [simp, presburger]: $(\text{real } (n::\text{int}) < 0) = (n < 0)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-le-zero-cancel-iff* [simp, presburger]: $(\text{real } (n::\text{int}) \leq 0) = (n \leq 0)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-abs* [simp]: $\text{real } (\text{abs } x) = \text{abs}(\text{real } (x::\text{int}))$
 $\langle \text{proof} \rangle$

lemma *int-less-real-le*: $((n::\text{int}) < m) = (\text{real } n + 1 \leq \text{real } m)$
 $\langle \text{proof} \rangle$

lemma *int-le-real-less*: $((n::\text{int}) \leq m) = (\text{real } n < \text{real } m + 1)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-div-aux*: $d \sim 0 \implies (\text{real } (x::\text{int})) / (\text{real } d) =$
 $\text{real } (x \text{ div } d) + (\text{real } (x \bmod d)) / (\text{real } d)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-div*: $(d::\text{int}) \sim 0 \implies d \text{ dvd } n \implies$
 $\text{real}(n \text{ div } d) = \text{real } n / \text{real } d$
 $\langle \text{proof} \rangle$

lemma *real-of-int-div2*:
 $0 \leq \text{real } (n::\text{int}) / \text{real } (x) - \text{real } (n \text{ div } x)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-div3*:
 $\text{real } (n::\text{int}) / \text{real } (x) - \text{real } (n \text{ div } x) \leq 1$
 $\langle \text{proof} \rangle$

lemma *real-of-int-div4*: $\text{real } (n \text{ div } x) \leq \text{real } (n::\text{int}) / \text{real } x$
 $\langle \text{proof} \rangle$

52.11 Embedding the Naturals into the Reals

lemma *real-of-nat-zero* [simp]: $\text{real } (0::\text{nat}) = 0$
 $\langle \text{proof} \rangle$

lemma *real-of-nat-1* [simp]: $\text{real } (1::\text{nat}) = 1$
 $\langle \text{proof} \rangle$

lemma *real-of-nat-one* [simp]: $\text{real } (\text{Suc } 0) = (1::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-of-nat-add* [simp]: $\text{real } (m + n) = \text{real } (m::\text{nat}) + \text{real } n$
 $\langle \text{proof} \rangle$

lemma *real-of-nat-Suc*: $\text{real } (\text{Suc } n) = \text{real } n + (1::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-of-nat-less-iff* [iff]:

$(\text{real } (n::\text{nat}) < \text{real } m) = (n < m)$
 $\langle \text{proof} \rangle$

lemma *real-of-nat-le-iff* [iff]: $(\text{real } (n::\text{nat}) \leq \text{real } m) = (n \leq m)$

$\langle \text{proof} \rangle$

lemma *real-of-nat-ge-zero* [iff]: $0 \leq \text{real } (n::\text{nat})$

$\langle \text{proof} \rangle$

lemma *real-of-nat-Suc-gt-zero*: $0 < \text{real } (\text{Suc } n)$

$\langle \text{proof} \rangle$

lemma *real-of-nat-mult* [simp]: $\text{real } (m * n) = \text{real } (m::\text{nat}) * \text{real } n$

$\langle \text{proof} \rangle$

lemma *real-of-nat-setsum* [simp]: $\text{real } ((\text{SUM } x:A. f x)::\text{nat}) =$

$(\text{SUM } x:A. \text{real}(f x))$

$\langle \text{proof} \rangle$

lemma *real-of-nat-setprod* [simp]: $\text{real } ((\text{PROD } x:A. f x)::\text{nat}) =$

$(\text{PROD } x:A. \text{real}(f x))$

$\langle \text{proof} \rangle$

lemma *real-of-card*: $\text{real } (\text{card } A) = \text{setsum } (\%x.1) A$

$\langle \text{proof} \rangle$

lemma *real-of-nat-inject* [iff]: $(\text{real } (n::\text{nat}) = \text{real } m) = (n = m)$

$\langle \text{proof} \rangle$

lemma *real-of-nat-zero-iff* [iff]: $(\text{real } (n::\text{nat}) = 0) = (n = 0)$

$\langle \text{proof} \rangle$

lemma *real-of-nat-diff*: $n \leq m \implies \text{real } (m - n) = \text{real } (m::\text{nat}) - \text{real } n$

$\langle \text{proof} \rangle$

lemma *real-of-nat-gt-zero-cancel-iff* [simp]: $(0 < \text{real } (n::\text{nat})) = (0 < n)$

$\langle \text{proof} \rangle$

lemma *real-of-nat-le-zero-cancel-iff* [simp]: $(\text{real } (n::\text{nat}) \leq 0) = (n = 0)$

$\langle \text{proof} \rangle$

lemma *not-real-of-nat-less-zero* [simp]: $\sim \text{real } (n::\text{nat}) < 0$

$\langle \text{proof} \rangle$

lemma *real-of-nat-ge-zero-cancel-iff* [simp]: $(0 \leq \text{real } (n::\text{nat}))$

$\langle \text{proof} \rangle$

lemma *nat-less-real-le*: $((n::nat) < m) = (real\ n + 1 \leq real\ m)$
 $\langle proof \rangle$

lemma *nat-le-real-less*: $((n::nat) \leq m) = (real\ n < real\ m + 1)$
 $\langle proof \rangle$

lemma *real-of-nat-div-aux*: $0 < d \implies (real\ (x::nat)) / (real\ d) =$
 $real\ (x\ div\ d) + (real\ (x\ mod\ d)) / (real\ d)$
 $\langle proof \rangle$

lemma *real-of-nat-div*: $0 < (d::nat) \implies d\ dvd\ n \implies$
 $real\ (n\ div\ d) = real\ n / real\ d$
 $\langle proof \rangle$

lemma *real-of-nat-div2*:
 $0 \leq real\ (n::nat) / real\ (x) - real\ (n\ div\ x)$
 $\langle proof \rangle$

lemma *real-of-nat-div3*:
 $real\ (n::nat) / real\ (x) - real\ (n\ div\ x) \leq 1$
 $\langle proof \rangle$

lemma *real-of-nat-div4*: $real\ (n\ div\ x) \leq real\ (n::nat) / real\ x$
 $\langle proof \rangle$

lemma *real-of-int-real-of-nat*: $real\ (int\ n) = real\ n$
 $\langle proof \rangle$

lemma *real-of-int-of-nat-eq [simp]*: $real\ (of_nat\ n :: int) = real\ n$
 $\langle proof \rangle$

lemma *real-nat-eq-real [simp]*: $0 \leq x \implies real\ (nat\ x) = real\ x$
 $\langle proof \rangle$

52.12 Rationals

lemma *Rats-real-nat [simp]*: $real\ (n::nat) \in \mathbb{Q}$
 $\langle proof \rangle$

lemma *Rats-eq-int-div-int*:
 $\mathbb{Q} = \{ real\ (i::int) / real\ (j::int) \mid i\ j. j \neq 0 \}$ (**is** - = ?S)
 $\langle proof \rangle$

lemma *Rats-eq-int-div-nat*:
 $\mathbb{Q} = \{ real\ (i::int) / real\ (n::nat) \mid i\ n. n \neq 0 \}$
 $\langle proof \rangle$

lemma *Rats-abs-nat-div-natE*:

assumes $x \in \mathbb{Q}$
obtains $m\ n$ **where** $n \neq 0$ **and** $|x| = \text{real } m / \text{real } n$ **and** $\text{gcd } m\ n = 1$
 $\langle \text{proof} \rangle$

52.13 Numerals and Arithmetic

instantiation $\text{real} :: \text{number-ring}$
begin

definition
 $\text{real-number-of-def } [\text{code del}]: \text{number-of } w = \text{real-of-int } w$

instance
 $\langle \text{proof} \rangle$

end

lemma $[\text{code unfold, symmetric, code post}]:$
 $\text{number-of } k = \text{real-of-int } (\text{number-of } k)$
 $\langle \text{proof} \rangle$

Collapse applications of real to number-of

lemma $\text{real-number-of } [\text{simp}]: \text{real } (\text{number-of } v :: \text{int}) = \text{number-of } v$
 $\langle \text{proof} \rangle$

lemma $\text{real-of-nat-number-of } [\text{simp}]:$
 $\text{real } (\text{number-of } v :: \text{nat}) =$
 $(\text{if neg } (\text{number-of } v :: \text{int}) \text{ then } 0$
 $\text{else } (\text{number-of } v :: \text{real}))$
 $\langle \text{proof} \rangle$

$\langle \text{ML} \rangle$

52.14 Simprules combining $x+y$ and 0 : ARE THEY NEEDED?

Needed in this non-standard form by Hyperreal/Transcendental

lemma $\text{real-0-le-divide-iff}:$
 $((0::\text{real}) \leq x/y) = ((x \leq 0 \mid 0 \leq y) \ \& \ (0 \leq x \mid y \leq 0))$
 $\langle \text{proof} \rangle$

lemma $\text{real-add-minus-iff } [\text{simp}]: (x + - a = (0::\text{real})) = (x=a)$
 $\langle \text{proof} \rangle$

lemma $\text{real-add-eq-0-iff}: (x+y = (0::\text{real})) = (y = -x)$
 $\langle \text{proof} \rangle$

lemma $\text{real-add-less-0-iff}: (x+y < (0::\text{real})) = (y < -x)$
 $\langle \text{proof} \rangle$

lemma *real-0-less-add-iff*: $((0::real) < x+y) = (-x < y)$
 $\langle proof \rangle$

lemma *real-add-le-0-iff*: $(x+y \leq (0::real)) = (y \leq -x)$
 $\langle proof \rangle$

lemma *real-0-le-add-iff*: $((0::real) \leq x+y) = (-x \leq y)$
 $\langle proof \rangle$

52.14.1 Density of the Reals

lemma *real-lbound-gt-zero*:
 $[(0::real) < d1; 0 < d2] ==> \exists e. 0 < e \ \& \ e < d1 \ \& \ e < d2$
 $\langle proof \rangle$

Similar results are proved in *Ring-and-Field*

lemma *real-less-half-sum*: $x < y ==> x < (x+y) / (2::real)$
 $\langle proof \rangle$

lemma *real-gt-half-sum*: $x < y ==> (x+y)/(2::real) < y$
 $\langle proof \rangle$

52.15 Absolute Value Function for the Reals

lemma *abs-minus-add-cancel*: $abs(x + (-y)) = abs(y + -(x::real))$
 $\langle proof \rangle$

lemma *abs-le-interval-iff*: $(abs\ x \leq r) = (-r \leq x \ \& \ x \leq (r::real))$
 $\langle proof \rangle$

lemma *abs-add-one-gt-zero* [simp]: $(0::real) < 1 + abs(x)$
 $\langle proof \rangle$

lemma *abs-real-of-nat-cancel* [simp]: $abs\ (real\ x) = real\ (x::nat)$
 $\langle proof \rangle$

lemma *abs-add-one-not-less-self* [simp]: $\sim abs(x) + (1::real) < x$
 $\langle proof \rangle$

lemma *abs-sum-triangle-ineq*: $abs\ ((x::real) + y + (-l + -m)) \leq abs(x + -l) + abs(y + -m)$
 $\langle proof \rangle$

instance *real :: lordered-ring*
 $\langle proof \rangle$

52.16 Implementation of rational real numbers

definition *Ratreal* :: *rat* \Rightarrow *real* **where**

[*simp*]: *Ratreal* = *of-rat*

code-datatype *Ratreal*

lemma *Ratreal-number-collapse* [*code post*]:

Ratreal 0 = 0

Ratreal 1 = 1

Ratreal (*number-of* *k*) = *number-of* *k*

\langle *proof* \rangle

lemma *zero-real-code* [*code*, *code unfold*]:

0 = *Ratreal* 0

\langle *proof* \rangle

lemma *one-real-code* [*code*, *code unfold*]:

1 = *Ratreal* 1

\langle *proof* \rangle

lemma *number-of-real-code* [*code unfold*]:

number-of *k* = *Ratreal* (*number-of* *k*)

\langle *proof* \rangle

lemma *Ratreal-number-of-quotient* [*code post*]:

Ratreal (*number-of* *r*) / *Ratreal* (*number-of* *s*) = *number-of* *r* / *number-of* *s*

\langle *proof* \rangle

lemma *Ratreal-number-of-quotient2* [*code post*]:

Ratreal (*number-of* *r* / *number-of* *s*) = *number-of* *r* / *number-of* *s*

\langle *proof* \rangle

instantiation *real* :: *eq*

begin

definition *eq-class.eq* (*x::real*) *y* \longleftrightarrow *x* - *y* = 0

instance \langle *proof* \rangle

lemma *real-eq-code* [*code*]: *eq-class.eq* (*Ratreal* *x*) (*Ratreal* *y*) \longleftrightarrow *eq-class.eq* *x* *y*

\langle *proof* \rangle

lemma *real-eq-refl* [*code nbe*]:

eq-class.eq (*x::real*) *x* \longleftrightarrow *True*

\langle *proof* \rangle

end

lemma *real-less-eq-code* [*code*]: *Ratreal* *x* \leq *Ratreal* *y* \longleftrightarrow *x* \leq *y*

$\langle \text{proof} \rangle$

lemma *real-less-code* [code]: $\text{Ratreal } x < \text{Ratreal } y \longleftrightarrow x < y$
 $\langle \text{proof} \rangle$

lemma *real-plus-code* [code]: $\text{Ratreal } x + \text{Ratreal } y = \text{Ratreal } (x + y)$
 $\langle \text{proof} \rangle$

lemma *real-times-code* [code]: $\text{Ratreal } x * \text{Ratreal } y = \text{Ratreal } (x * y)$
 $\langle \text{proof} \rangle$

lemma *real-uminus-code* [code]: $-\text{Ratreal } x = \text{Ratreal } (-x)$
 $\langle \text{proof} \rangle$

lemma *real-minus-code* [code]: $\text{Ratreal } x - \text{Ratreal } y = \text{Ratreal } (x - y)$
 $\langle \text{proof} \rangle$

lemma *real-inverse-code* [code]: $\text{inverse } (\text{Ratreal } x) = \text{Ratreal } (\text{inverse } x)$
 $\langle \text{proof} \rangle$

lemma *real-divide-code* [code]: $\text{Ratreal } x / \text{Ratreal } y = \text{Ratreal } (x / y)$
 $\langle \text{proof} \rangle$

Setup for SML code generator

types-code

real ((*int* */ *int*))

attach (*term-of*) $\langle\langle$

fun *term-of-real* (*p*, *q*) =

let

val *rT* = *HOLogic.realT*

in

if *q* = 1 *orelse* *p* = 0 *then* *HOLogic.mk-number rT p*

else @{*term op / :: real \Rightarrow real \Rightarrow real*} \$

HOLogic.mk-number rT p \$ *HOLogic.mk-number rT q*

end;

$\rangle\rangle$

attach (*test*) $\langle\langle$

fun *gen-real* *i* =

let

val *p* = *random-range* 0 *i*;

val *q* = *random-range* 1 (*i* + 1);

val *g* = *Integer.gcd* *p* *q*;

val *p'* = *p div g*;

val *q'* = *q div g*;

val *r* = (*if one-of* [*true*, *false*] *then* *p'* *else* \sim *p'*,

if *p'* = 0 *then* 0 *else* *q'*)

in

(*r*, *fn* () => *term-of-real* *r*)

end;

```

>>

consts-code
  Ratreal ((-))

consts-code
  of-int :: int ⇒ real (⟨module⟩real'-of'-int)
attach ⟨⟨
  fun real-of-int 0 = (0, 0)
    | real-of-int i = (i, 1);
  ⟩⟩

end

```

53 RComplete: Completeness of the Reals; Floor and Ceiling Functions

```

theory RComplete
imports Lubs RealDef
begin

```

```

lemma real-sum-of-halves:  $x/2 + x/2 = (x::real)$ 
  ⟨proof⟩

```

53.1 Completeness of Positive Reals

Supremum property for the set of positive reals

Let P be a non-empty set of positive reals, with an upper bound y . Then P has a least upper bound (written S).

FIXME: Can the premise be weakened to $\forall x \in P. x \leq y$?

```

lemma posreal-complete:
  assumes positive-P:  $\forall x \in P. (0::real) < x$ 
  and not-empty-P:  $\exists x. x \in P$ 
  and upper-bound-Ex:  $\exists y. \forall x \in P. x < y$ 
  shows  $\exists S. \forall y. (\exists x \in P. y < x) = (y < S)$ 
  ⟨proof⟩

```

Completeness properties using *isUb*, *isLub* etc.

```

lemma real-isLub-unique:  $[[ \text{isLub } R \ S \ x; \text{isLub } R \ S \ y ]] \implies x = (y::real)$ 
  ⟨proof⟩

```

Completeness theorem for the positive reals (again).

```

lemma posreals-complete:
  assumes positive-S:  $\forall x \in S. 0 < x$ 

```

and *not-empty-S*: $\exists x. x \in S$
and *upper-bound-Ex*: $\exists u. \text{isUb } (UNIV::\text{real set}) S u$
shows $\exists t. \text{isLub } (UNIV::\text{real set}) S t$
 $\langle \text{proof} \rangle$

reals Completeness (again!)

lemma *reals-complete*:
assumes *notempty-S*: $\exists X. X \in S$
and *exists-Ub*: $\exists Y. \text{isUb } (UNIV::\text{real set}) S Y$
shows $\exists t. \text{isLub } (UNIV::\text{real set}) S t$
 $\langle \text{proof} \rangle$

53.2 The Archimedean Property of the Reals

theorem *reals-Archimedean*:
assumes *x-pos*: $0 < x$
shows $\exists n. \text{inverse } (\text{real } (\text{Suc } n)) < x$
 $\langle \text{proof} \rangle$

There must be other proofs, e.g. *Suc* of the largest integer in the cut representing x .

lemma *reals-Archimedean2*: $\exists n. (x::\text{real}) < \text{real } (n::\text{nat})$
 $\langle \text{proof} \rangle$

instance *real :: archimedean-field*
 $\langle \text{proof} \rangle$

lemma *reals-Archimedean3*:
assumes *x-greater-zero*: $0 < x$
shows $\forall (y::\text{real}). \exists (n::\text{nat}). y < \text{real } n * x$
 $\langle \text{proof} \rangle$

lemma *reals-Archimedean6*:
 $0 \leq r \implies \exists (n::\text{nat}). \text{real } (n - 1) \leq r \ \& \ r < \text{real } (n)$
 $\langle \text{proof} \rangle$

lemma *reals-Archimedean6a*: $0 \leq r \implies \exists n. \text{real } (n) \leq r \ \& \ r < \text{real } (\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma *reals-Archimedean-6b-int*:
 $0 \leq r \implies \exists n::\text{int}. \text{real } n \leq r \ \& \ r < \text{real } (n+1)$
 $\langle \text{proof} \rangle$

lemma *reals-Archimedean-6c-int*:
 $r < 0 \implies \exists n::\text{int}. \text{real } n \leq r \ \& \ r < \text{real } (n+1)$
 $\langle \text{proof} \rangle$

53.3 Density of the Rational Reals in the Reals

This density proof is due to Stefan Richter and was ported by TN. The original source is *Real Analysis* by H.L. Royden. It employs the Archimedean property of the reals.

lemma *Rats-dense-in-nn-real*: **fixes** $x::\text{real}$
assumes $0 \leq x$ **and** $x < y$ **shows** $\exists r \in \mathbb{Q}. x < r \wedge r < y$
 $\langle \text{proof} \rangle$

theorem *Rats-dense-in-real*: **fixes** $x y :: \text{real}$
assumes $x < y$ **shows** $\exists r \in \mathbb{Q}. x < r \wedge r < y$
 $\langle \text{proof} \rangle$

53.4 Floor and Ceiling Functions from the Reals to the Integers

lemma *number-of-less-real-of-int-iff* [simp]:
 $((\text{number-of } n) < \text{real } (m::\text{int})) = (\text{number-of } n < m)$
 $\langle \text{proof} \rangle$

lemma *number-of-less-real-of-int-iff2* [simp]:
 $(\text{real } (m::\text{int}) < (\text{number-of } n)) = (m < \text{number-of } n)$
 $\langle \text{proof} \rangle$

lemma *number-of-le-real-of-int-iff* [simp]:
 $((\text{number-of } n) \leq \text{real } (m::\text{int})) = (\text{number-of } n \leq m)$
 $\langle \text{proof} \rangle$

lemma *number-of-le-real-of-int-iff2* [simp]:
 $(\text{real } (m::\text{int}) \leq (\text{number-of } n)) = (m \leq \text{number-of } n)$
 $\langle \text{proof} \rangle$

lemma *floor-real-of-nat-zero*: $\text{floor } (\text{real } (0::\text{nat})) = 0$
 $\langle \text{proof} \rangle$

lemma *floor-real-of-nat* [simp]: $\text{floor } (\text{real } (n::\text{nat})) = \text{int } n$
 $\langle \text{proof} \rangle$

lemma *floor-minus-real-of-nat* [simp]: $\text{floor } (- \text{real } (n::\text{nat})) = - \text{int } n$
 $\langle \text{proof} \rangle$

lemma *floor-real-of-int* [simp]: $\text{floor } (\text{real } (n::\text{int})) = n$
 $\langle \text{proof} \rangle$

lemma *floor-minus-real-of-int* [simp]: $\text{floor } (- \text{real } (n::\text{int})) = - n$
 $\langle \text{proof} \rangle$

lemma *real-lb-ub-int*: $\exists n::\text{int}. \text{real } n \leq r \ \& \ r < \text{real } (n+1)$
 $\langle \text{proof} \rangle$

lemma *lemma-floor*:

assumes $a1$: $\text{real } m \leq r$ **and** $a2$: $r < \text{real } n + 1$

shows $m \leq (n::\text{int})$

$\langle \text{proof} \rangle$

lemma *real-of-int-floor-le* [simp]: $\text{real } (\text{floor } r) \leq r$

$\langle \text{proof} \rangle$

lemma *lemma-floor2*: $\text{real } n < \text{real } (x::\text{int}) + 1 \implies n \leq x$

$\langle \text{proof} \rangle$

lemma *real-of-int-floor-cancel* [simp]:

$(\text{real } (\text{floor } x) = x) = (\exists n::\text{int}. x = \text{real } n)$

$\langle \text{proof} \rangle$

lemma *floor-eq*: $[\mid \text{real } n < x; x < \text{real } n + 1 \mid] \implies \text{floor } x = n$

$\langle \text{proof} \rangle$

lemma *floor-eq2*: $[\mid \text{real } n \leq x; x < \text{real } n + 1 \mid] \implies \text{floor } x = n$

$\langle \text{proof} \rangle$

lemma *floor-eq3*: $[\mid \text{real } n < x; x < \text{real } (\text{Suc } n) \mid] \implies \text{nat}(\text{floor } x) = n$

$\langle \text{proof} \rangle$

lemma *floor-eq4*: $[\mid \text{real } n \leq x; x < \text{real } (\text{Suc } n) \mid] \implies \text{nat}(\text{floor } x) = n$

$\langle \text{proof} \rangle$

lemma *floor-number-of-eq*:

$\text{floor}(\text{number-of } n :: \text{real}) = (\text{number-of } n :: \text{int})$

$\langle \text{proof} \rangle$

lemma *real-of-int-floor-ge-diff-one* [simp]: $r - 1 \leq \text{real}(\text{floor } r)$

$\langle \text{proof} \rangle$

lemma *real-of-int-floor-gt-diff-one* [simp]: $r - 1 < \text{real}(\text{floor } r)$

$\langle \text{proof} \rangle$

lemma *real-of-int-floor-add-one-ge* [simp]: $r \leq \text{real}(\text{floor } r) + 1$

$\langle \text{proof} \rangle$

lemma *real-of-int-floor-add-one-gt* [simp]: $r < \text{real}(\text{floor } r) + 1$

$\langle \text{proof} \rangle$

lemma *le-floor*: $\text{real } a \leq x \implies a \leq \text{floor } x$

$\langle \text{proof} \rangle$

lemma *real-le-floor*: $a \leq \text{floor } x \implies \text{real } a \leq x$

$\langle \text{proof} \rangle$

lemma *le-floor-eq*: $(a \leq \text{floor } x) = (\text{real } a \leq x)$
 $\langle \text{proof} \rangle$

lemma *le-floor-eq-number-of*:
 $(\text{number-of } n \leq \text{floor } x) = (\text{number-of } n \leq x)$
 $\langle \text{proof} \rangle$

lemma *le-floor-eq-zero*: $(0 \leq \text{floor } x) = (0 \leq x)$
 $\langle \text{proof} \rangle$

lemma *le-floor-eq-one*: $(1 \leq \text{floor } x) = (1 \leq x)$
 $\langle \text{proof} \rangle$

lemma *floor-less-eq*: $(\text{floor } x < a) = (x < \text{real } a)$
 $\langle \text{proof} \rangle$

lemma *floor-less-eq-number-of*:
 $(\text{floor } x < \text{number-of } n) = (x < \text{number-of } n)$
 $\langle \text{proof} \rangle$

lemma *floor-less-eq-zero*: $(\text{floor } x < 0) = (x < 0)$
 $\langle \text{proof} \rangle$

lemma *floor-less-eq-one*: $(\text{floor } x < 1) = (x < 1)$
 $\langle \text{proof} \rangle$

lemma *less-floor-eq*: $(a < \text{floor } x) = (\text{real } a + 1 \leq x)$
 $\langle \text{proof} \rangle$

lemma *less-floor-eq-number-of*:
 $(\text{number-of } n < \text{floor } x) = (\text{number-of } n + 1 \leq x)$
 $\langle \text{proof} \rangle$

lemma *less-floor-eq-zero*: $(0 < \text{floor } x) = (1 \leq x)$
 $\langle \text{proof} \rangle$

lemma *less-floor-eq-one*: $(1 < \text{floor } x) = (2 \leq x)$
 $\langle \text{proof} \rangle$

lemma *floor-le-eq*: $(\text{floor } x \leq a) = (x < \text{real } a + 1)$
 $\langle \text{proof} \rangle$

lemma *floor-le-eq-number-of*:
 $(\text{floor } x \leq \text{number-of } n) = (x < \text{number-of } n + 1)$
 $\langle \text{proof} \rangle$

lemma *floor-le-eq-zero*: $(\text{floor } x \leq 0) = (x < 1)$
 $\langle \text{proof} \rangle$

lemma *floor-le-eq-one*: $(\text{floor } x \leq 1) = (x < 2)$
 $\langle \text{proof} \rangle$

lemma *floor-add* [simp]: $\text{floor } (x + \text{real } a) = \text{floor } x + a$
 $\langle \text{proof} \rangle$

lemma *floor-subtract* [simp]: $\text{floor } (x - \text{real } a) = \text{floor } x - a$
 $\langle \text{proof} \rangle$

lemma *floor-subtract-number-of*: $\text{floor } (x - \text{number-of } n) =$
 $\text{floor } x - \text{number-of } n$
 $\langle \text{proof} \rangle$

lemma *floor-subtract-one*: $\text{floor } (x - 1) = \text{floor } x - 1$
 $\langle \text{proof} \rangle$

lemma *ceiling-real-of-nat* [simp]: $\text{ceiling } (\text{real } (n::\text{nat})) = \text{int } n$
 $\langle \text{proof} \rangle$

lemma *ceiling-real-of-nat-zero*: $\text{ceiling } (\text{real } (0::\text{nat})) = 0$
 $\langle \text{proof} \rangle$

lemma *ceiling-floor* [simp]: $\text{ceiling } (\text{real } (\text{floor } r)) = \text{floor } r$
 $\langle \text{proof} \rangle$

lemma *floor-ceiling* [simp]: $\text{floor } (\text{real } (\text{ceiling } r)) = \text{ceiling } r$
 $\langle \text{proof} \rangle$

lemma *real-of-int-ceiling-ge* [simp]: $r \leq \text{real } (\text{ceiling } r)$
 $\langle \text{proof} \rangle$

lemma *ceiling-real-of-int* [simp]: $\text{ceiling } (\text{real } (n::\text{int})) = n$
 $\langle \text{proof} \rangle$

lemma *real-of-int-ceiling-cancel* [simp]:
 $(\text{real } (\text{ceiling } x) = x) = (\exists n::\text{int}. x = \text{real } n)$
 $\langle \text{proof} \rangle$

lemma *ceiling-eq*: $[\text{real } n < x; x < \text{real } n + 1] \implies \text{ceiling } x = n + 1$
 $\langle \text{proof} \rangle$

lemma *ceiling-eq2*: $[\text{real } n < x; x \leq \text{real } n + 1] \implies \text{ceiling } x = n + 1$
 $\langle \text{proof} \rangle$

lemma *ceiling-eq3*: $[\text{real } n - 1 < x; x \leq \text{real } n] \implies \text{ceiling } x = n$
 $\langle \text{proof} \rangle$

lemma *ceiling-number-of-eq*:

$\text{ceiling } (\text{number-of } n :: \text{real}) = (\text{number-of } n)$
 $\langle \text{proof} \rangle$

lemma *real-of-int-ceiling-diff-one-le* [simp]: $\text{real } (\text{ceiling } r) - 1 \leq r$
 $\langle \text{proof} \rangle$

lemma *real-of-int-ceiling-le-add-one* [simp]: $\text{real } (\text{ceiling } r) \leq r + 1$
 $\langle \text{proof} \rangle$

lemma *ceiling-le*: $x \leq \text{real } a \implies \text{ceiling } x \leq a$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-real*: $\text{ceiling } x \leq a \implies x \leq \text{real } a$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-eq*: $(\text{ceiling } x \leq a) = (x \leq \text{real } a)$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-eq-number-of*:
 $(\text{ceiling } x \leq \text{number-of } n) = (x \leq \text{number-of } n)$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-zero-eq*: $(\text{ceiling } x \leq 0) = (x \leq 0)$
 $\langle \text{proof} \rangle$

lemma *ceiling-le-eq-one*: $(\text{ceiling } x \leq 1) = (x \leq 1)$
 $\langle \text{proof} \rangle$

lemma *less-ceiling-eq*: $(a < \text{ceiling } x) = (\text{real } a < x)$
 $\langle \text{proof} \rangle$

lemma *less-ceiling-eq-number-of*:
 $(\text{number-of } n < \text{ceiling } x) = (\text{number-of } n < x)$
 $\langle \text{proof} \rangle$

lemma *less-ceiling-eq-zero*: $(0 < \text{ceiling } x) = (0 < x)$
 $\langle \text{proof} \rangle$

lemma *less-ceiling-eq-one*: $(1 < \text{ceiling } x) = (1 < x)$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-eq*: $(\text{ceiling } x < a) = (x \leq \text{real } a - 1)$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-eq-number-of*:
 $(\text{ceiling } x < \text{number-of } n) = (x \leq \text{number-of } n - 1)$
 $\langle \text{proof} \rangle$

lemma *ceiling-less-eq-zero*: $(\text{ceiling } x < 0) = (x \leq -1)$

$\langle \text{proof} \rangle$

lemma *ceiling-less-eq-one*: $(\text{ceiling } x < 1) = (x \leq 0)$
 $\langle \text{proof} \rangle$

lemma *le-ceiling-eq*: $(a \leq \text{ceiling } x) = (\text{real } a - 1 < x)$
 $\langle \text{proof} \rangle$

lemma *le-ceiling-eq-number-of*:
 $(\text{number-of } n \leq \text{ceiling } x) = (\text{number-of } n - 1 < x)$
 $\langle \text{proof} \rangle$

lemma *le-ceiling-eq-zero*: $(0 \leq \text{ceiling } x) = (-1 < x)$
 $\langle \text{proof} \rangle$

lemma *le-ceiling-eq-one*: $(1 \leq \text{ceiling } x) = (0 < x)$
 $\langle \text{proof} \rangle$

lemma *ceiling-add* [simp]: $\text{ceiling } (x + \text{real } a) = \text{ceiling } x + a$
 $\langle \text{proof} \rangle$

lemma *ceiling-subtract* [simp]: $\text{ceiling } (x - \text{real } a) = \text{ceiling } x - a$
 $\langle \text{proof} \rangle$

lemma *ceiling-subtract-number-of*: $\text{ceiling } (x - \text{number-of } n) =$
 $\text{ceiling } x - \text{number-of } n$
 $\langle \text{proof} \rangle$

lemma *ceiling-subtract-one*: $\text{ceiling } (x - 1) = \text{ceiling } x - 1$
 $\langle \text{proof} \rangle$

53.5 Versions for the natural numbers

definition

$\text{natfloor} :: \text{real} \Rightarrow \text{nat}$ **where**
 $\text{natfloor } x = \text{nat}(\text{floor } x)$

definition

$\text{natceiling} :: \text{real} \Rightarrow \text{nat}$ **where**
 $\text{natceiling } x = \text{nat}(\text{ceiling } x)$

lemma *natfloor-zero* [simp]: $\text{natfloor } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *natfloor-one* [simp]: $\text{natfloor } 1 = 1$
 $\langle \text{proof} \rangle$

lemma *zero-le-natfloor* [simp]: $0 \leq \text{natfloor } x$
 $\langle \text{proof} \rangle$

lemma *natfloor-number-of-eq* [simp]: $\text{natfloor } (\text{number-of } n) = \text{number-of } n$
 ⟨proof⟩

lemma *natfloor-real-of-nat* [simp]: $\text{natfloor}(\text{real } n) = n$
 ⟨proof⟩

lemma *real-natfloor-le*: $0 \leq x \implies \text{real}(\text{natfloor } x) \leq x$
 ⟨proof⟩

lemma *natfloor-neg*: $x \leq 0 \implies \text{natfloor } x = 0$
 ⟨proof⟩

lemma *natfloor-mono*: $x \leq y \implies \text{natfloor } x \leq \text{natfloor } y$
 ⟨proof⟩

lemma *le-natfloor*: $\text{real } x \leq a \implies x \leq \text{natfloor } a$
 ⟨proof⟩

lemma *le-natfloor-eq*: $0 \leq x \implies (a \leq \text{natfloor } x) = (\text{real } a \leq x)$
 ⟨proof⟩

lemma *le-natfloor-eq-number-of* [simp]:
 $\sim \text{neg}((\text{number-of } n)::\text{int}) \implies 0 \leq x \implies$
 $(\text{number-of } n \leq \text{natfloor } x) = (\text{number-of } n \leq x)$
 ⟨proof⟩

lemma *le-natfloor-eq-one* [simp]: $(1 \leq \text{natfloor } x) = (1 \leq x)$
 ⟨proof⟩

lemma *natfloor-eq*: $\text{real } n \leq x \implies x < \text{real } n + 1 \implies \text{natfloor } x = n$
 ⟨proof⟩

lemma *real-natfloor-add-one-gt*: $x < \text{real}(\text{natfloor } x) + 1$
 ⟨proof⟩

lemma *real-natfloor-gt-diff-one*: $x - 1 < \text{real}(\text{natfloor } x)$
 ⟨proof⟩

lemma *ge-natfloor-plus-one-imp-gt*: $\text{natfloor } z + 1 \leq n \implies z < \text{real } n$
 ⟨proof⟩

lemma *natfloor-add* [simp]: $0 \leq x \implies \text{natfloor } (x + \text{real } a) = \text{natfloor } x + a$
 ⟨proof⟩

lemma *natfloor-add-number-of* [simp]:
 $\sim \text{neg}((\text{number-of } n)::\text{int}) \implies 0 \leq x \implies$
 $\text{natfloor } (x + \text{number-of } n) = \text{natfloor } x + \text{number-of } n$
 ⟨proof⟩

lemma *natfloor-add-one*: $0 \leq x \implies \text{natfloor}(x + 1) = \text{natfloor } x + 1$
 ⟨proof⟩

lemma *natfloor-subtract* [simp]: $\text{real } a \leq x \implies$
 $\text{natfloor}(x - \text{real } a) = \text{natfloor } x - a$
 ⟨proof⟩

lemma *natceiling-zero* [simp]: $\text{natceiling } 0 = 0$
 ⟨proof⟩

lemma *natceiling-one* [simp]: $\text{natceiling } 1 = 1$
 ⟨proof⟩

lemma *zero-le-natceiling* [simp]: $0 \leq \text{natceiling } x$
 ⟨proof⟩

lemma *natceiling-number-of-eq* [simp]: $\text{natceiling } (\text{number-of } n) = \text{number-of } n$
 ⟨proof⟩

lemma *natceiling-real-of-nat* [simp]: $\text{natceiling}(\text{real } n) = n$
 ⟨proof⟩

lemma *real-natceiling-ge*: $x \leq \text{real}(\text{natceiling } x)$
 ⟨proof⟩

lemma *natceiling-neg*: $x \leq 0 \implies \text{natceiling } x = 0$
 ⟨proof⟩

lemma *natceiling-mono*: $x \leq y \implies \text{natceiling } x \leq \text{natceiling } y$
 ⟨proof⟩

lemma *natceiling-le*: $x \leq \text{real } a \implies \text{natceiling } x \leq a$
 ⟨proof⟩

lemma *natceiling-le-eq*: $0 \leq x \implies (\text{natceiling } x \leq a) = (x \leq \text{real } a)$
 ⟨proof⟩

lemma *natceiling-le-eq-number-of* [simp]:
 $\sim \text{neg}((\text{number-of } n)::\text{int}) \implies 0 \leq x \implies$
 $(\text{natceiling } x \leq \text{number-of } n) = (x \leq \text{number-of } n)$
 ⟨proof⟩

lemma *natceiling-le-eq-one*: $(\text{natceiling } x \leq 1) = (x \leq 1)$
 ⟨proof⟩

lemma *natceiling-eq*: $\text{real } n < x \implies x \leq \text{real } n + 1 \implies \text{natceiling } x = n + 1$
 ⟨proof⟩

lemma *natceiling-add* [*simp*]: $0 \leq x \implies$
 $\text{natceiling } (x + \text{real } a) = \text{natceiling } x + a$
 ⟨*proof*⟩

lemma *natceiling-add-number-of* [*simp*]:
 $\sim \text{neg } ((\text{number-of } n)::\text{int}) \implies 0 \leq x \implies$
 $\text{natceiling } (x + \text{number-of } n) = \text{natceiling } x + \text{number-of } n$
 ⟨*proof*⟩

lemma *natceiling-add-one*: $0 \leq x \implies \text{natceiling}(x + 1) = \text{natceiling } x + 1$
 ⟨*proof*⟩

lemma *natceiling-subtract* [*simp*]: $\text{real } a \leq x \implies$
 $\text{natceiling}(x - \text{real } a) = \text{natceiling } x - a$
 ⟨*proof*⟩

lemma *natfloor-div-nat*: $1 \leq x \implies y > 0 \implies$
 $\text{natfloor } (x / \text{real } y) = \text{natfloor } x \text{ div } y$
 ⟨*proof*⟩

end

54 RealPow: Natural powers theory

theory *RealPow*
imports *RealDef*
uses (*Tools/float-syntax.ML*)
begin

declare *abs-mult-self* [*simp*]

instantiation *real* :: *recpower*
begin

primrec *power-real* **where**
 $r \wedge 0 = (1::\text{real})$
 $| r \wedge \text{Suc } n = (r::\text{real}) * r \wedge n$

instance ⟨*proof*⟩

declare *power-real.simps* [*simp del*]

end

lemma *two-realpow-ge-one* [*simp*]: $(1::\text{real}) \leq 2 \wedge n$
 ⟨*proof*⟩

lemma *two-realpow-gt* [*simp*]: $\text{real } (n::\text{nat}) < 2 \wedge n$
 $\langle \text{proof} \rangle$

lemma *realpow-Suc-le-self*: $[[0 \leq r; r \leq (1::\text{real})]] \implies r \wedge \text{Suc } n \leq r$
 $\langle \text{proof} \rangle$

lemma *realpow-minus-mult* [*rule-format*]:
 $0 < n \longleftrightarrow (x::\text{real}) \wedge (n - 1) * x = x \wedge n$
 $\langle \text{proof} \rangle$

lemma *realpow-two-mult-inverse* [*simp*]:
 $r \neq 0 \implies r * \text{inverse } r \wedge \text{Suc } (\text{Suc } 0) = \text{inverse } (r::\text{real})$
 $\langle \text{proof} \rangle$

lemma *realpow-two-minus* [*simp*]: $(-x) \wedge \text{Suc } (\text{Suc } 0) = (x::\text{real}) \wedge \text{Suc } (\text{Suc } 0)$
 $\langle \text{proof} \rangle$

lemma *realpow-two-diff*:
 $(x::\text{real}) \wedge \text{Suc } (\text{Suc } 0) - y \wedge \text{Suc } (\text{Suc } 0) = (x - y) * (x + y)$
 $\langle \text{proof} \rangle$

lemma *realpow-two-disj*:
 $((x::\text{real}) \wedge \text{Suc } (\text{Suc } 0) = y \wedge \text{Suc } (\text{Suc } 0)) = (x = y \mid x = -y)$
 $\langle \text{proof} \rangle$

lemma *realpow-real-of-nat*: $\text{real } (m::\text{nat}) \wedge n = \text{real } (m \wedge n)$
 $\langle \text{proof} \rangle$

lemma *realpow-real-of-nat-two-pos* [*simp*]: $0 < \text{real } (\text{Suc } (\text{Suc } 0) \wedge n)$
 $\langle \text{proof} \rangle$

lemma *realpow-increasing*:
 $[[(0::\text{real}) \leq x; 0 \leq y; x \wedge \text{Suc } n \leq y \wedge \text{Suc } n]] \implies x \leq y$
 $\langle \text{proof} \rangle$

54.1 Literal Arithmetic Involving Powers, Type *real*

lemma *real-of-int-power*: $\text{real } (x::\text{int}) \wedge n = \text{real } (x \wedge n)$
 $\langle \text{proof} \rangle$

declare *real-of-int-power* [*symmetric, simp*]

lemma *power-real-number-of*:
 $(\text{number-of } v :: \text{real}) \wedge n = \text{real } ((\text{number-of } v :: \text{int}) \wedge n)$
 $\langle \text{proof} \rangle$

declare *power-real-number-of* [*of - number-of w, standard, simp*]

54.2 Properties of Squares

lemma *sum-squares-ge-zero*:

fixes $x\ y :: 'a::\text{ordered-ring-strict}$

shows $0 \leq x * x + y * y$

<proof>

lemma *not-sum-squares-lt-zero*:

fixes $x\ y :: 'a::\text{ordered-ring-strict}$

shows $\neg x * x + y * y < 0$

<proof>

lemma *sum-nonneg-eq-zero-iff*:

fixes $x\ y :: 'a::\text{pordered-ab-group-add}$

assumes $x: 0 \leq x$ **and** $y: 0 \leq y$

shows $(x + y = 0) = (x = 0 \wedge y = 0)$

<proof>

lemma *sum-squares-eq-zero-iff*:

fixes $x\ y :: 'a::\text{ordered-ring-strict}$

shows $(x * x + y * y = 0) = (x = 0 \wedge y = 0)$

<proof>

lemma *sum-squares-le-zero-iff*:

fixes $x\ y :: 'a::\text{ordered-ring-strict}$

shows $(x * x + y * y \leq 0) = (x = 0 \wedge y = 0)$

<proof>

lemma *sum-squares-gt-zero-iff*:

fixes $x\ y :: 'a::\text{ordered-ring-strict}$

shows $(0 < x * x + y * y) = (x \neq 0 \vee y \neq 0)$

<proof>

lemma *sum-power2-ge-zero*:

fixes $x\ y :: 'a::\{\text{ordered-idom}, \text{recpower}\}$

shows $0 \leq x^2 + y^2$

<proof>

lemma *not-sum-power2-lt-zero*:

fixes $x\ y :: 'a::\{\text{ordered-idom}, \text{recpower}\}$

shows $\neg x^2 + y^2 < 0$

<proof>

lemma *sum-power2-eq-zero-iff*:

fixes $x\ y :: 'a::\{\text{ordered-idom}, \text{recpower}\}$

shows $(x^2 + y^2 = 0) = (x = 0 \wedge y = 0)$

<proof>

lemma *sum-power2-le-zero-iff*:

fixes $x\ y :: 'a::\{\text{ordered-idom}, \text{recpower}\}$

shows $(x^2 + y^2 \leq 0) = (x = 0 \wedge y = 0)$
 $\langle \text{proof} \rangle$

lemma *sum-power2-gt-zero-iff*:
fixes $x\ y :: 'a::\{\text{ordered-idom}, \text{recpower}\}$
shows $(0 < x^2 + y^2) = (x \neq 0 \vee y \neq 0)$
 $\langle \text{proof} \rangle$

54.3 Squares of Reals

lemma *real-two-squares-add-zero-iff* [simp]:
 $(x * x + y * y = 0) = ((x::\text{real}) = 0 \wedge y = 0)$
 $\langle \text{proof} \rangle$

lemma *real-sum-squares-cancel*: $x * x + y * y = 0 ==> x = (0::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-sum-squares-cancel2*: $x * x + y * y = 0 ==> y = (0::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-mult-self-sum-ge-zero*: $(0::\text{real}) \leq x*x + y*y$
 $\langle \text{proof} \rangle$

lemma *real-sum-squares-cancel-a*: $x * x = -(y * y) ==> x = (0::\text{real}) \ \& \ y=0$
 $\langle \text{proof} \rangle$

lemma *real-squared-diff-one-factored*: $x*x - (1::\text{real}) = (x + 1)*(x - 1)$
 $\langle \text{proof} \rangle$

lemma *real-mult-is-one* [simp]: $(x*x = (1::\text{real})) = (x = 1 \mid x = -1)$
 $\langle \text{proof} \rangle$

lemma *real-sum-squares-not-zero*: $x \sim 0 ==> x * x + y * y \sim (0::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-sum-squares-not-zero2*: $y \sim 0 ==> x * x + y * y \sim (0::\text{real})$
 $\langle \text{proof} \rangle$

lemma *realpow-two-sum-zero-iff* [simp]:
 $(x^2 + y^2 = (0::\text{real})) = (x = 0 \ \& \ y = 0)$
 $\langle \text{proof} \rangle$

lemma *realpow-two-le-add-order* [simp]: $(0::\text{real}) \leq u^2 + v^2$
 $\langle \text{proof} \rangle$

lemma *realpow-two-le-add-order2* [simp]: $(0::\text{real}) \leq u^2 + v^2 + w^2$
 $\langle \text{proof} \rangle$

lemma *real-sum-square-gt-zero*: $x \sim 0 ==> (0::\text{real}) < x * x + y * y$

$\langle \text{proof} \rangle$

lemma *real-sum-square-gt-zero2*: $y \sim= 0 \implies (0::\text{real}) < x * x + y * y$
 $\langle \text{proof} \rangle$

lemma *real-minus-mult-self-le* [simp]: $-(u * u) \leq (x * (x::\text{real}))$
 $\langle \text{proof} \rangle$

lemma *realpow-square-minus-le* [simp]: $-(u ^ 2) \leq (x::\text{real}) ^ 2$
 $\langle \text{proof} \rangle$

lemma *real-sq-order*:
 fixes $x::\text{real}$
 assumes $xgt0$: $0 \leq x$ and $ygt0$: $0 \leq y$ and sq : $x^2 \leq y^2$
 shows $x \leq y$
 $\langle \text{proof} \rangle$

54.4 Various Other Theorems

lemma *real-le-add-half-cancel*: $(x + y/2 \leq (y::\text{real})) = (x \leq y/2)$
 $\langle \text{proof} \rangle$

lemma *real-minus-half-eq* [simp]: $(x::\text{real}) - x/2 = x/2$
 $\langle \text{proof} \rangle$

lemma *real-mult-inverse-cancel*:
 $[(0::\text{real}) < x; 0 < x1; x1 * y < x * u] \implies \text{inverse } x * y < \text{inverse } x1 * u$
 $\langle \text{proof} \rangle$

lemma *real-mult-inverse-cancel2*:
 $[(0::\text{real}) < x; 0 < x1; x1 * y < x * u] \implies y * \text{inverse } x < u * \text{inverse } x1$
 $\langle \text{proof} \rangle$

lemma *inverse-real-of-nat-gt-zero* [simp]: $0 < \text{inverse } (\text{real } (\text{Suc } n))$
 $\langle \text{proof} \rangle$

lemma *inverse-real-of-nat-ge-zero* [simp]: $0 \leq \text{inverse } (\text{real } (\text{Suc } n))$
 $\langle \text{proof} \rangle$

lemma *realpow-num-eq-if*: $(m::\text{real}) ^ n = (\text{if } n=0 \text{ then } 1 \text{ else } m * m ^ (n - 1))$
 $\langle \text{proof} \rangle$

54.5 Float syntax

syntax *-Float* :: *float-const* \Rightarrow 'a $\quad (-)$

$\langle \text{ML} \rangle$

Test:

lemma $123.456 = -111.111 + 200 + 30 + 4 + 5/10 + 6/100 + (7/1000::real)$
 $\langle proof \rangle$

end

55 RealVector: Vector Spaces and Algebras over the Reals

theory *RealVector*
imports *RealPow*
begin

55.1 Locale for additive functions

locale *additive* =
fixes $f :: 'a::ab-group-add \Rightarrow 'b::ab-group-add$
assumes $add: f (x + y) = f x + f y$
begin

lemma *zero*: $f 0 = 0$
 $\langle proof \rangle$

lemma *minus*: $f (- x) = - f x$
 $\langle proof \rangle$

lemma *diff*: $f (x - y) = f x - f y$
 $\langle proof \rangle$

lemma *setsum*: $f (setsum g A) = (\sum x \in A. f (g x))$
 $\langle proof \rangle$

end

55.2 Vector spaces

locale *vector-space* =
fixes $scale :: 'a::field \Rightarrow 'b::ab-group-add \Rightarrow 'b$
assumes *scale-right-distrib* [*algebra-simps*]:
 $scale a (x + y) = scale a x + scale a y$
and *scale-left-distrib* [*algebra-simps*]:
 $scale (a + b) x = scale a x + scale b x$
and *scale-scale* [*simp*]: $scale a (scale b x) = scale (a * b) x$
and *scale-one* [*simp*]: $scale 1 x = x$
begin

lemma *scale-left-commute*:

$scale\ a\ (scale\ b\ x) = scale\ b\ (scale\ a\ x)$
 $\langle proof \rangle$

lemma *scale-zero-left* [simp]: $scale\ 0\ x = 0$
and *scale-minus-left* [simp]: $scale\ (-\ a)\ x = -\ (scale\ a\ x)$
and *scale-left-diff-distrib* [algebra-simps]:
 $scale\ (a - b)\ x = scale\ a\ x - scale\ b\ x$
 $\langle proof \rangle$

lemma *scale-zero-right* [simp]: $scale\ a\ 0 = 0$
and *scale-minus-right* [simp]: $scale\ a\ (-\ x) = -\ (scale\ a\ x)$
and *scale-right-diff-distrib* [algebra-simps]:
 $scale\ a\ (x - y) = scale\ a\ x - scale\ a\ y$
 $\langle proof \rangle$

lemma *scale-eq-0-iff* [simp]:
 $scale\ a\ x = 0 \longleftrightarrow a = 0 \vee x = 0$
 $\langle proof \rangle$

lemma *scale-left-imp-eq*:
 $\llbracket a \neq 0; scale\ a\ x = scale\ a\ y \rrbracket \Longrightarrow x = y$
 $\langle proof \rangle$

lemma *scale-right-imp-eq*:
 $\llbracket x \neq 0; scale\ a\ x = scale\ b\ x \rrbracket \Longrightarrow a = b$
 $\langle proof \rangle$

lemma *scale-cancel-left*:
 $scale\ a\ x = scale\ a\ y \longleftrightarrow x = y \vee a = 0$
 $\langle proof \rangle$

lemma *scale-cancel-right*:
 $scale\ a\ x = scale\ b\ x \longleftrightarrow a = b \vee x = 0$
 $\langle proof \rangle$

end

55.3 Real vector spaces

class *scaleR* =
fixes *scaleR* :: $real \Rightarrow 'a \Rightarrow 'a$ (**infixr** $*_R$ 75)
begin

abbreviation
 $divideR :: 'a \Rightarrow real \Rightarrow 'a$ (**infixl** $'/_R$ 70)
where
 $x\ /_R\ r == scaleR\ (inverse\ r)\ x$

end

```

class real-vector = scaleR + ab-group-add +
  assumes scaleR-right-distrib: scaleR a (x + y) = scaleR a x + scaleR a y
  and scaleR-left-distrib: scaleR (a + b) x = scaleR a x + scaleR b x
  and scaleR-scaleR: scaleR a (scaleR b x) = scaleR (a * b) x
  and scaleR-one: scaleR 1 x = x

```

```

interpretation real-vector:
  vector-space scaleR :: real  $\Rightarrow$  'a  $\Rightarrow$  'a::real-vector
  <proof>

```

Recover original theorem names

```

lemmas scaleR-left-commute = real-vector.scale-left-commute
lemmas scaleR-zero-left = real-vector.scale-zero-left
lemmas scaleR-minus-left = real-vector.scale-minus-left
lemmas scaleR-left-diff-distrib = real-vector.scale-left-diff-distrib
lemmas scaleR-zero-right = real-vector.scale-zero-right
lemmas scaleR-minus-right = real-vector.scale-minus-right
lemmas scaleR-right-diff-distrib = real-vector.scale-right-diff-distrib
lemmas scaleR-eq-0-iff = real-vector.scale-eq-0-iff
lemmas scaleR-left-imp-eq = real-vector.scale-left-imp-eq
lemmas scaleR-right-imp-eq = real-vector.scale-right-imp-eq
lemmas scaleR-cancel-left = real-vector.scale-cancel-left
lemmas scaleR-cancel-right = real-vector.scale-cancel-right

```

```

class real-algebra = real-vector + ring +
  assumes mult-scaleR-left [simp]: scaleR a x * y = scaleR a (x * y)
  and mult-scaleR-right [simp]: x * scaleR a y = scaleR a (x * y)

```

```

class real-algebra-1 = real-algebra + ring-1

```

```

class real-div-algebra = real-algebra-1 + division-ring

```

```

class real-field = real-div-algebra + field

```

```

instantiation real :: real-field
begin

```

```

definition
  real-scaleR-def [simp]: scaleR a x = a * x

```

```

instance <proof>

```

```

end

```

```

interpretation scaleR-left: additive ( $\lambda a. \text{scaleR } a \text{ } x :: 'a :: \text{real-vector}$ )
  <proof>

```

```

interpretation scaleR-right: additive ( $\lambda x. \text{scaleR } a \text{ } x :: 'a :: \text{real-vector}$ )

```

⟨proof⟩

lemma *nonzero-inverse-scaleR-distrib*:

fixes $x :: 'a::\text{real-div-algebra}$ **shows**

$\llbracket a \neq 0; x \neq 0 \rrbracket \implies \text{inverse} (\text{scaleR } a \ x) = \text{scaleR } (\text{inverse } a) (\text{inverse } x)$

⟨proof⟩

lemma *inverse-scaleR-distrib*:

fixes $x :: 'a::\{\text{real-div-algebra}, \text{division-by-zero}\}$

shows $\text{inverse} (\text{scaleR } a \ x) = \text{scaleR } (\text{inverse } a) (\text{inverse } x)$

⟨proof⟩

55.4 Embedding of the Reals into any *real-algebra-1*: *of-real*

definition

of-real $:: \text{real} \Rightarrow 'a::\text{real-algebra-1}$ **where**

of-real $r = \text{scaleR } r \ 1$

lemma *scaleR-conv-of-real*: $\text{scaleR } r \ x = \text{of-real } r * x$

⟨proof⟩

lemma *of-real-0 [simp]*: $\text{of-real } 0 = 0$

⟨proof⟩

lemma *of-real-1 [simp]*: $\text{of-real } 1 = 1$

⟨proof⟩

lemma *of-real-add [simp]*: $\text{of-real } (x + y) = \text{of-real } x + \text{of-real } y$

⟨proof⟩

lemma *of-real-minus [simp]*: $\text{of-real } (-x) = - \text{of-real } x$

⟨proof⟩

lemma *of-real-diff [simp]*: $\text{of-real } (x - y) = \text{of-real } x - \text{of-real } y$

⟨proof⟩

lemma *of-real-mult [simp]*: $\text{of-real } (x * y) = \text{of-real } x * \text{of-real } y$

⟨proof⟩

lemma *nonzero-of-real-inverse*:

$x \neq 0 \implies \text{of-real } (\text{inverse } x) =$

$\text{inverse } (\text{of-real } x :: 'a::\text{real-div-algebra})$

⟨proof⟩

lemma *of-real-inverse [simp]*:

$\text{of-real } (\text{inverse } x) =$

$\text{inverse } (\text{of-real } x :: 'a::\{\text{real-div-algebra}, \text{division-by-zero}\})$

⟨proof⟩

lemma *nonzero-of-real-divide*:

$y \neq 0 \implies \text{of-real } (x / y) =$
 $(\text{of-real } x / \text{of-real } y :: 'a::\text{real-field})$
 $\langle \text{proof} \rangle$

lemma *of-real-divide [simp]*:

$\text{of-real } (x / y) =$
 $(\text{of-real } x / \text{of-real } y :: 'a::\{\text{real-field}, \text{division-by-zero}\})$
 $\langle \text{proof} \rangle$

lemma *of-real-power [simp]*:

$\text{of-real } (x ^ n) = (\text{of-real } x :: 'a::\{\text{real-algebra-1}, \text{recpower}\}) ^ n$
 $\langle \text{proof} \rangle$

lemma *of-real-eq-iff [simp]*: $(\text{of-real } x = \text{of-real } y) = (x = y)$
 $\langle \text{proof} \rangle$

lemmas *of-real-eq-0-iff [simp]* = *of-real-eq-iff [of - 0, simplified]*

lemma *of-real-eq-id [simp]*: $\text{of-real} = (\text{id} :: \text{real} \Rightarrow \text{real})$
 $\langle \text{proof} \rangle$

Collapse nested embeddings

lemma *of-real-of-nat-eq [simp]*: $\text{of-real } (\text{of-nat } n) = \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *of-real-of-int-eq [simp]*: $\text{of-real } (\text{of-int } z) = \text{of-int } z$
 $\langle \text{proof} \rangle$

lemma *of-real-number-of-eq*:

$\text{of-real } (\text{number-of } w) = (\text{number-of } w :: 'a::\{\text{number-ring}, \text{real-algebra-1}\})$
 $\langle \text{proof} \rangle$

Every real algebra has characteristic zero

instance *real-algebra-1 < ring-char-0*
 $\langle \text{proof} \rangle$

instance *real-field < field-char-0* $\langle \text{proof} \rangle$

55.5 The Set of Real Numbers

definition

Reals :: *'a::real-algebra-1 set* **where**
 $[\text{code del}]: \text{Reals} = \text{range of-real}$

notation (*xsymbols*)

Reals (\mathbb{R})

lemma *Reals-of-real [simp]*: $\text{of-real } r \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-of-int* [simp]: *of-int* $z \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-of-nat* [simp]: *of-nat* $n \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-number-of* [simp]:

(*number-of* $w :: 'a :: \{\text{number-ring}, \text{real-algebra-1}\}$) $\in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-0* [simp]: $0 \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-1* [simp]: $1 \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-add* [simp]: $\llbracket a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies a + b \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-minus* [simp]: $a \in \text{Reals} \implies -a \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-diff* [simp]: $\llbracket a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies a - b \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-mult* [simp]: $\llbracket a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies a * b \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *nonzero-Reals-inverse*:

fixes $a :: 'a :: \text{real-div-algebra}$

shows $\llbracket a \in \text{Reals}; a \neq 0 \rrbracket \implies \text{inverse } a \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-inverse* [simp]:

fixes $a :: 'a :: \{\text{real-div-algebra}, \text{division-by-zero}\}$

shows $a \in \text{Reals} \implies \text{inverse } a \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *nonzero-Reals-divide*:

fixes $a b :: 'a :: \text{real-field}$

shows $\llbracket a \in \text{Reals}; b \in \text{Reals}; b \neq 0 \rrbracket \implies a / b \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-divide* [simp]:

fixes $a b :: 'a :: \{\text{real-field}, \text{division-by-zero}\}$

shows $\llbracket a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies a / b \in \text{Reals}$

$\langle \text{proof} \rangle$

lemma *Reals-power* [*simp*]:
fixes $a :: 'a :: \{\text{real-algebra-1}, \text{recpower}\}$
shows $a \in \text{Reals} \implies a^n \in \text{Reals}$
 ⟨*proof*⟩

lemma *Reals-cases* [*cases set: Reals*]:
assumes $q \in \mathbb{R}$
obtains (*of-real*) r **where** $q = \text{of-real } r$
 ⟨*proof*⟩

lemma *Reals-induct* [*case-names of-real, induct set: Reals*]:
 $q \in \mathbb{R} \implies (\bigwedge r. P (\text{of-real } r)) \implies P q$
 ⟨*proof*⟩

55.6 Real normed vector spaces

class *norm* =
fixes $\text{norm} :: 'a \Rightarrow \text{real}$

class *sgn-div-norm* = *scaleR* + *norm* + *sgn* +
assumes *sgn-div-norm*: $\text{sgn } x = x /_R \text{ norm } x$

class *real-normed-vector* = *real-vector* + *sgn-div-norm* +
assumes *norm-ge-zero* [*simp*]: $0 \leq \text{norm } x$
and *norm-eq-zero* [*simp*]: $\text{norm } x = 0 \iff x = 0$
and *norm-triangle-ineq*: $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$
and *norm-scaleR*: $\text{norm } (\text{scaleR } a \ x) = |a| * \text{norm } x$

class *real-normed-algebra* = *real-algebra* + *real-normed-vector* +
assumes *norm-mult-ineq*: $\text{norm } (x * y) \leq \text{norm } x * \text{norm } y$

class *real-normed-algebra-1* = *real-algebra-1* + *real-normed-algebra* +
assumes *norm-one* [*simp*]: $\text{norm } 1 = 1$

class *real-normed-div-algebra* = *real-div-algebra* + *real-normed-vector* +
assumes *norm-mult*: $\text{norm } (x * y) = \text{norm } x * \text{norm } y$

class *real-normed-field* = *real-field* + *real-normed-div-algebra*

instance *real-normed-div-algebra* < *real-normed-algebra-1*
 ⟨*proof*⟩

instantiation *real* :: *real-normed-field*
begin

definition
real-norm-def [*simp*]: $\text{norm } r = |r|$

instance

$\langle proof \rangle$

end

lemma *norm-zero* [simp]: $norm\ (0 :: 'a :: real-normed-vector) = 0$

$\langle proof \rangle$

lemma *zero-less-norm-iff* [simp]:

fixes $x :: 'a :: real-normed-vector$

shows $(0 < norm\ x) = (x \neq 0)$

$\langle proof \rangle$

lemma *norm-not-less-zero* [simp]:

fixes $x :: 'a :: real-normed-vector$

shows $\neg norm\ x < 0$

$\langle proof \rangle$

lemma *norm-le-zero-iff* [simp]:

fixes $x :: 'a :: real-normed-vector$

shows $(norm\ x \leq 0) = (x = 0)$

$\langle proof \rangle$

lemma *norm-minus-cancel* [simp]:

fixes $x :: 'a :: real-normed-vector$

shows $norm\ (-\ x) = norm\ x$

$\langle proof \rangle$

lemma *norm-minus-commute*:

fixes $a\ b :: 'a :: real-normed-vector$

shows $norm\ (a - b) = norm\ (b - a)$

$\langle proof \rangle$

lemma *norm-triangle-ineq2*:

fixes $a\ b :: 'a :: real-normed-vector$

shows $norm\ a - norm\ b \leq norm\ (a - b)$

$\langle proof \rangle$

lemma *norm-triangle-ineq3*:

fixes $a\ b :: 'a :: real-normed-vector$

shows $|norm\ a - norm\ b| \leq norm\ (a - b)$

$\langle proof \rangle$

lemma *norm-triangle-ineq4*:

fixes $a\ b :: 'a :: real-normed-vector$

shows $norm\ (a - b) \leq norm\ a + norm\ b$

$\langle proof \rangle$

lemma *norm-diff-ineq*:

fixes $a\ b :: 'a::\text{real-normed-vector}$
shows $\text{norm } a - \text{norm } b \leq \text{norm } (a + b)$
 $\langle \text{proof} \rangle$

lemma *norm-diff-triangle-ineq*:
fixes $a\ b\ c\ d :: 'a::\text{real-normed-vector}$
shows $\text{norm } ((a + b) - (c + d)) \leq \text{norm } (a - c) + \text{norm } (b - d)$
 $\langle \text{proof} \rangle$

lemma *abs-norm-cancel [simp]*:
fixes $a :: 'a::\text{real-normed-vector}$
shows $|\text{norm } a| = \text{norm } a$
 $\langle \text{proof} \rangle$

lemma *norm-add-less*:
fixes $x\ y :: 'a::\text{real-normed-vector}$
shows $\llbracket \text{norm } x < r; \text{norm } y < s \rrbracket \implies \text{norm } (x + y) < r + s$
 $\langle \text{proof} \rangle$

lemma *norm-mult-less*:
fixes $x\ y :: 'a::\text{real-normed-algebra}$
shows $\llbracket \text{norm } x < r; \text{norm } y < s \rrbracket \implies \text{norm } (x * y) < r * s$
 $\langle \text{proof} \rangle$

lemma *norm-of-real [simp]*:
 $\text{norm } (\text{of-real } r :: 'a::\text{real-normed-algebra-1}) = |r|$
 $\langle \text{proof} \rangle$

lemma *norm-number-of [simp]*:
 $\text{norm } (\text{number-of } w :: 'a::\{\text{number-ring}, \text{real-normed-algebra-1}\})$
 $= |\text{number-of } w|$
 $\langle \text{proof} \rangle$

lemma *norm-of-int [simp]*:
 $\text{norm } (\text{of-int } z :: 'a::\text{real-normed-algebra-1}) = |\text{of-int } z|$
 $\langle \text{proof} \rangle$

lemma *norm-of-nat [simp]*:
 $\text{norm } (\text{of-nat } n :: 'a::\text{real-normed-algebra-1}) = \text{of-nat } n$
 $\langle \text{proof} \rangle$

lemma *nonzero-norm-inverse*:
fixes $a :: 'a::\text{real-normed-div-algebra}$
shows $a \neq 0 \implies \text{norm } (\text{inverse } a) = \text{inverse } (\text{norm } a)$
 $\langle \text{proof} \rangle$

lemma *norm-inverse*:
fixes $a :: 'a::\{\text{real-normed-div-algebra}, \text{division-by-zero}\}$
shows $\text{norm } (\text{inverse } a) = \text{inverse } (\text{norm } a)$

⟨proof⟩

lemma *nonzero-norm-divide*:

fixes $a\ b :: 'a::\text{real-normed-field}$

shows $b \neq 0 \implies \text{norm } (a / b) = \text{norm } a / \text{norm } b$

⟨proof⟩

lemma *norm-divide*:

fixes $a\ b :: 'a::\{\text{real-normed-field}, \text{division-by-zero}\}$

shows $\text{norm } (a / b) = \text{norm } a / \text{norm } b$

⟨proof⟩

lemma *norm-power-ineq*:

fixes $x :: 'a::\{\text{real-normed-algebra-1}, \text{recpower}\}$

shows $\text{norm } (x ^ n) \leq \text{norm } x ^ n$

⟨proof⟩

lemma *norm-power*:

fixes $x :: 'a::\{\text{real-normed-div-algebra}, \text{recpower}\}$

shows $\text{norm } (x ^ n) = \text{norm } x ^ n$

⟨proof⟩

55.7 Sign function

lemma *norm-sgn*:

$\text{norm } (\text{sgn}(x::'a::\text{real-normed-vector})) = (\text{if } x = 0 \text{ then } 0 \text{ else } 1)$

⟨proof⟩

lemma *sgn-zero [simp]*: $\text{sgn}(0::'a::\text{real-normed-vector}) = 0$

⟨proof⟩

lemma *sgn-zero-iff*: $(\text{sgn}(x::'a::\text{real-normed-vector}) = 0) = (x = 0)$

⟨proof⟩

lemma *sgn-minus*: $\text{sgn } (- x) = - \text{sgn}(x::'a::\text{real-normed-vector})$

⟨proof⟩

lemma *sgn-scaleR*:

$\text{sgn } (\text{scaleR } r\ x) = \text{scaleR } (\text{sgn } r) (\text{sgn}(x::'a::\text{real-normed-vector}))$

⟨proof⟩

lemma *sgn-one [simp]*: $\text{sgn } (1::'a::\text{real-normed-algebra-1}) = 1$

⟨proof⟩

lemma *sgn-of-real*:

$\text{sgn } (\text{of-real } r::'a::\text{real-normed-algebra-1}) = \text{of-real } (\text{sgn } r)$

⟨proof⟩

lemma *sgn-mult*:

```

fixes  $x\ y :: 'a::\text{real-normed-div-algebra}$ 
shows  $\text{sgn } (x * y) = \text{sgn } x * \text{sgn } y$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma real-sgn-eq:  $\text{sgn } (x::\text{real}) = x / |x|$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma real-sgn-pos:  $0 < (x::\text{real}) \implies \text{sgn } x = 1$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma real-sgn-neg:  $(x::\text{real}) < 0 \implies \text{sgn } x = -1$ 
 $\langle \text{proof} \rangle$ 

```

55.8 Bounded Linear and Bilinear Operators

```

locale bounded-linear = additive +
  constrains  $f :: 'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}$ 
  assumes scaleR:  $f (\text{scaleR } r\ x) = \text{scaleR } r\ (f\ x)$ 
  assumes bounded:  $\exists K. \forall x. \text{norm } (f\ x) \leq \text{norm } x * K$ 
begin

```

```

lemma pos-bounded:
   $\exists K > 0. \forall x. \text{norm } (f\ x) \leq \text{norm } x * K$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma nonneg-bounded:
   $\exists K \geq 0. \forall x. \text{norm } (f\ x) \leq \text{norm } x * K$ 
 $\langle \text{proof} \rangle$ 

```

```

end

```

```

locale bounded-bilinear =
  fixes  $\text{prod} :: ['a::\text{real-normed-vector}, 'b::\text{real-normed-vector}]$ 
     $\Rightarrow 'c::\text{real-normed-vector}$ 
  (infixl ** 70)
  assumes add-left:  $\text{prod } (a + a')\ b = \text{prod } a\ b + \text{prod } a'\ b$ 
  assumes add-right:  $\text{prod } a\ (b + b') = \text{prod } a\ b + \text{prod } a\ b'$ 
  assumes scaleR-left:  $\text{prod } (\text{scaleR } r\ a)\ b = \text{scaleR } r\ (\text{prod } a\ b)$ 
  assumes scaleR-right:  $\text{prod } a\ (\text{scaleR } r\ b) = \text{scaleR } r\ (\text{prod } a\ b)$ 
  assumes bounded:  $\exists K. \forall a\ b. \text{norm } (\text{prod } a\ b) \leq \text{norm } a * \text{norm } b * K$ 
begin

```

```

lemma pos-bounded:
   $\exists K > 0. \forall a\ b. \text{norm } (a ** b) \leq \text{norm } a * \text{norm } b * K$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma nonneg-bounded:
   $\exists K \geq 0. \forall a\ b. \text{norm } (a ** b) \leq \text{norm } a * \text{norm } b * K$ 
 $\langle \text{proof} \rangle$ 

```

lemma *additive-right: additive* ($\lambda b. \text{prod } a \ b$)
 $\langle \text{proof} \rangle$

lemma *additive-left: additive* ($\lambda a. \text{prod } a \ b$)
 $\langle \text{proof} \rangle$

lemma *zero-left: prod* $0 \ b = 0$
 $\langle \text{proof} \rangle$

lemma *zero-right: prod* $a \ 0 = 0$
 $\langle \text{proof} \rangle$

lemma *minus-left: prod* $(- \ a) \ b = - \ \text{prod } a \ b$
 $\langle \text{proof} \rangle$

lemma *minus-right: prod* $a \ (- \ b) = - \ \text{prod } a \ b$
 $\langle \text{proof} \rangle$

lemma *diff-left:*
 $\text{prod } (a - a') \ b = \text{prod } a \ b - \text{prod } a' \ b$
 $\langle \text{proof} \rangle$

lemma *diff-right:*
 $\text{prod } a \ (b - b') = \text{prod } a \ b - \text{prod } a \ b'$
 $\langle \text{proof} \rangle$

lemma *bounded-linear-left:*
 $\text{bounded-linear } (\lambda a. \ a \ ** \ b)$
 $\langle \text{proof} \rangle$

lemma *bounded-linear-right:*
 $\text{bounded-linear } (\lambda b. \ a \ ** \ b)$
 $\langle \text{proof} \rangle$

lemma *prod-diff-prod:*
 $(x \ ** \ y - a \ ** \ b) = (x - a) \ ** \ (y - b) + (x - a) \ ** \ b + a \ ** \ (y - b)$
 $\langle \text{proof} \rangle$

end

interpretation *mult:*
 $\text{bounded-bilinear } \text{op } * :: 'a \Rightarrow 'a \Rightarrow 'a :: \text{real-normed-algebra}$
 $\langle \text{proof} \rangle$

interpretation *mult-left:*
 $\text{bounded-linear } (\lambda x :: 'a :: \text{real-normed-algebra}. \ x \ * \ y)$
 $\langle \text{proof} \rangle$


```

interpretation mult-right:
  bounded-linear ( $\lambda y::'a::\text{real-normed-algebra. } x * y$ )
   $\langle \text{proof} \rangle$ 

interpretation divide:
  bounded-linear ( $\lambda x::'a::\text{real-normed-field. } x / y$ )
   $\langle \text{proof} \rangle$ 

interpretation scaleR: bounded-bilinear scaleR
   $\langle \text{proof} \rangle$ 

interpretation scaleR-left: bounded-linear  $\lambda r. \text{scaleR } r \ x$ 
   $\langle \text{proof} \rangle$ 

interpretation scaleR-right: bounded-linear  $\lambda x. \text{scaleR } r \ x$ 
   $\langle \text{proof} \rangle$ 

interpretation of-real: bounded-linear  $\lambda r. \text{of-real } r$ 
   $\langle \text{proof} \rangle$ 

end

theory Real
imports RComplete RealVector
begin

end

```

56 Fact: Factorial Function

```

theory Fact
imports Main
begin

consts fact :: nat => nat
primrec
  fact-0: fact 0 = 1
  fact-Suc: fact (Suc n) = (Suc n) * fact n

lemma fact-gt-zero [simp]: 0 < fact n
   $\langle \text{proof} \rangle$ 

lemma fact-not-eq-zero [simp]: fact n  $\neq$  0
   $\langle \text{proof} \rangle$ 

lemma of-nat-fact-not-zero [simp]: of-nat (fact n)  $\neq$  (0::a::semiring-char-0)
   $\langle \text{proof} \rangle$ 

```

class *ordered-semiring-1* = *ordered-semiring* + *semiring-1*

class *ordered-semiring-1-strict* = *ordered-semiring-strict* + *semiring-1*

lemma *of-nat-fact-gt-zero* [*simp*]: $(0::'a::\{\text{ordered-semidom}\}) < \text{of-nat}(\text{fact } n)$ $\langle \text{proof} \rangle$

lemma *of-nat-fact-ge-zero* [*simp*]: $(0::'a::\text{ordered-semidom}) \leq \text{of-nat}(\text{fact } n)$
 $\langle \text{proof} \rangle$

lemma *fact-ge-one* [*simp*]: $1 \leq \text{fact } n$
 $\langle \text{proof} \rangle$

lemma *fact-mono*: $m \leq n \implies \text{fact } m \leq \text{fact } n$
 $\langle \text{proof} \rangle$

Note that *fact 0* = *fact 1*

lemma *fact-less-mono*: $[| 0 < m; m < n |] \implies \text{fact } m < \text{fact } n$
 $\langle \text{proof} \rangle$

lemma *inv-of-nat-fact-gt-zero* [*simp*]: $(0::'a::\text{ordered-field}) < \text{inverse}(\text{of-nat}(\text{fact } n))$
 $\langle \text{proof} \rangle$

lemma *inv-of-nat-fact-ge-zero* [*simp*]: $(0::'a::\text{ordered-field}) \leq \text{inverse}(\text{of-nat}(\text{fact } n))$
 $\langle \text{proof} \rangle$

lemma *fact-diff-Suc* [*rule-format*]:
 $n < \text{Suc } m \implies \text{fact } (\text{Suc } m - n) = (\text{Suc } m - n) * \text{fact } (m - n)$
 $\langle \text{proof} \rangle$

lemma *fact-num0*: *fact 0* = 1
 $\langle \text{proof} \rangle$

lemma *fact-num-eq-if*: *fact m* = (if *m*=0 then 1 else *m* * *fact (m - 1)*)
 $\langle \text{proof} \rangle$

lemma *fact-add-num-eq-if*:
 $\text{fact } (m + n) = (\text{if } m + n = 0 \text{ then } 1 \text{ else } (m + n) * \text{fact } (m + n - 1))$
 $\langle \text{proof} \rangle$

lemma *fact-add-num-eq-if2*:
 $\text{fact } (m + n) = (\text{if } m = 0 \text{ then } \text{fact } n \text{ else } (m + n) * \text{fact } ((m - 1) + n))$
 $\langle \text{proof} \rangle$

end

57 SEQ: Sequences and Convergence

```
theory SEQ
imports RealVector RComplete
begin
```

definition

```
Zseq :: [nat => 'a::real-normed-vector] => bool where
  — Standard definition of sequence converging to zero
[code del]: Zseq X = (∀ r>0. ∃ no. ∀ n≥no. norm (X n) < r)
```

definition

```
LIMSEQ :: [nat => 'a::real-normed-vector, 'a] => bool
  (((-)/ ----> (-)) [60, 60] 60) where
  — Standard definition of convergence of sequence
[code del]: X ----> L = (∀ r. 0 < r --> (∃ no. ∀ n. no ≤ n --> norm (X
n - L) < r))
```

definition

```
lim :: (nat => 'a::real-normed-vector) => 'a where
  — Standard definition of limit using choice operator
lim X = (THE L. X ----> L)
```

definition

```
convergent :: (nat => 'a::real-normed-vector) => bool where
  — Standard definition of convergence
convergent X = (∃ L. X ----> L)
```

definition

```
Bseq :: (nat => 'a::real-normed-vector) => bool where
  — Standard definition for bounded sequence
[code del]: Bseq X = (∃ K>0. ∀ n. norm (X n) ≤ K)
```

definition

```
monoseq :: (nat=>real)=>bool where
  — Definition of monotonicity. The use of disjunction here complicates proofs
  considerably. One alternative is to add a Boolean argument to indicate the direc-
  tion. Another is to develop the notions of increasing and decreasing first.
[code del]: monoseq X = ((∀ m. ∀ n≥m. X m ≤ X n) | (∀ m. ∀ n≥m. X n ≤ X
m))
```

definition

```
incseq :: (nat=>real)=>bool where
  — Increasing sequence
[code del]: incseq X = (∀ m. ∀ n≥m. X m ≤ X n)
```

definition

```
decseq :: (nat=>real)=>bool where
  — Decreasing sequence
```

[code del]: $\text{decseq } X = (\forall m. \forall n \geq m. X\ n \leq X\ m)$

definition

$\text{subseq} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}$ **where**

— Definition of subsequence

[code del]: $\text{subseq } f = (\forall m. \forall n > m. (f\ m) < (f\ n))$

definition

$\text{Cauchy} :: (\text{nat} \Rightarrow 'a::\text{real-normed-vector}) \Rightarrow \text{bool}$ **where**

— Standard definition of the Cauchy condition

[code del]: $\text{Cauchy } X = (\forall e > 0. \exists M. \forall m \geq M. \forall n \geq M. \text{norm } (X\ m - X\ n) < e)$

57.1 Bounded Sequences

lemma BseqI' : **assumes** $K: \bigwedge n. \text{norm } (X\ n) \leq K$ **shows** $\text{Bseq } X$
 ⟨proof⟩

lemma BseqE : $\llbracket \text{Bseq } X; \bigwedge K. \llbracket 0 < K; \forall n. \text{norm } (X\ n) \leq K \rrbracket \Longrightarrow Q \rrbracket \Longrightarrow Q$
 ⟨proof⟩

lemma $\text{BseqI2}'$: **assumes** $K: \forall n \geq N. \text{norm } (X\ n) \leq K$ **shows** $\text{Bseq } X$
 ⟨proof⟩

lemma $\text{Bseq-ignore-initial-segment}$: $\text{Bseq } X \Longrightarrow \text{Bseq } (\lambda n. X\ (n + k))$
 ⟨proof⟩

lemma Bseq-offset : $\text{Bseq } (\lambda n. X\ (n + k)) \Longrightarrow \text{Bseq } X$
 ⟨proof⟩

57.2 Sequences That Converge to Zero

lemma ZseqI :
 $(\bigwedge r. 0 < r \Longrightarrow \exists no. \forall n \geq no. \text{norm } (X\ n) < r) \Longrightarrow \text{Zseq } X$
 ⟨proof⟩

lemma ZseqD :
 $\llbracket \text{Zseq } X; 0 < r \rrbracket \Longrightarrow \exists no. \forall n \geq no. \text{norm } (X\ n) < r$
 ⟨proof⟩

lemma Zseq-zero : $\text{Zseq } (\lambda n. 0)$
 ⟨proof⟩

lemma Zseq-const-iff : $\text{Zseq } (\lambda n. k) = (k = 0)$
 ⟨proof⟩

lemma Zseq-norm-iff : $\text{Zseq } (\lambda n. \text{norm } (X\ n)) = \text{Zseq } (\lambda n. X\ n)$
 ⟨proof⟩

lemma Zseq-imp-Zseq :

assumes X : $Zseq\ X$
assumes Y : $\bigwedge n. norm\ (Y\ n) \leq norm\ (X\ n) * K$
shows $Zseq\ (\lambda n. Y\ n)$
 $\langle proof \rangle$

lemma $Zseq-le$: $\llbracket Zseq\ Y; \forall n. norm\ (X\ n) \leq norm\ (Y\ n) \rrbracket \Longrightarrow Zseq\ X$
 $\langle proof \rangle$

lemma $Zseq-add$:
assumes X : $Zseq\ X$
assumes Y : $Zseq\ Y$
shows $Zseq\ (\lambda n. X\ n + Y\ n)$
 $\langle proof \rangle$

lemma $Zseq-minus$: $Zseq\ X \Longrightarrow Zseq\ (\lambda n. -\ X\ n)$
 $\langle proof \rangle$

lemma $Zseq-diff$: $\llbracket Zseq\ X; Zseq\ Y \rrbracket \Longrightarrow Zseq\ (\lambda n. X\ n - Y\ n)$
 $\langle proof \rangle$

lemma (**in** $bounded-linear$) $Zseq$:
assumes X : $Zseq\ X$
shows $Zseq\ (\lambda n. f\ (X\ n))$
 $\langle proof \rangle$

lemma (**in** $bounded-bilinear$) $Zseq$:
assumes X : $Zseq\ X$
assumes Y : $Zseq\ Y$
shows $Zseq\ (\lambda n. X\ n ** Y\ n)$
 $\langle proof \rangle$

lemma (**in** $bounded-bilinear$) $Zseq-prod-Bseq$:
assumes X : $Zseq\ X$
assumes Y : $Bseq\ Y$
shows $Zseq\ (\lambda n. X\ n ** Y\ n)$
 $\langle proof \rangle$

lemma (**in** $bounded-bilinear$) $Bseq-prod-Zseq$:
assumes X : $Bseq\ X$
assumes Y : $Zseq\ Y$
shows $Zseq\ (\lambda n. X\ n ** Y\ n)$
 $\langle proof \rangle$

lemma (**in** $bounded-bilinear$) $Zseq-left$:
 $Zseq\ X \Longrightarrow Zseq\ (\lambda n. X\ n ** a)$
 $\langle proof \rangle$

lemma (**in** $bounded-bilinear$) $Zseq-right$:
 $Zseq\ X \Longrightarrow Zseq\ (\lambda n. a ** X\ n)$

$\langle proof \rangle$

lemmas $Zseq-mult = mult.Zseq$

lemmas $Zseq-mult-right = mult.Zseq-right$

lemmas $Zseq-mult-left = mult.Zseq-left$

57.3 Limits of Sequences

lemma $LIMSEQ-iff$:

$$(X \text{ ----> } L) = (\forall r > 0. \exists no. \forall n \geq no. norm (X\ n - L) < r)$$

$\langle proof \rangle$

lemma $LIMSEQ-Zseq-iff$: $((\lambda n. X\ n) \text{ ----> } L) = Zseq (\lambda n. X\ n - L)$

$\langle proof \rangle$

lemma $LIMSEQ-I$:

$$(\bigwedge r. 0 < r \implies \exists no. \forall n \geq no. norm (X\ n - L) < r) \implies X \text{ ----> } L$$

$\langle proof \rangle$

lemma $LIMSEQ-D$:

$$\llbracket X \text{ ----> } L; 0 < r \rrbracket \implies \exists no. \forall n \geq no. norm (X\ n - L) < r$$

$\langle proof \rangle$

lemma $LIMSEQ-const$: $(\lambda n. k) \text{ ----> } k$

$\langle proof \rangle$

lemma $LIMSEQ-const-iff$: $(\lambda n. k) \text{ ----> } l = (k = l)$

$\langle proof \rangle$

lemma $LIMSEQ-norm$: $X \text{ ----> } a \implies (\lambda n. norm (X\ n)) \text{ ----> } norm\ a$

$\langle proof \rangle$

lemma $LIMSEQ-ignore-initial-segment$:

$$f \text{ ----> } a \implies (\lambda n. f\ (n + k)) \text{ ----> } a$$

$\langle proof \rangle$

lemma $LIMSEQ-offset$:

$$(\lambda n. f\ (n + k)) \text{ ----> } a \implies f \text{ ----> } a$$

$\langle proof \rangle$

lemma $LIMSEQ-Suc$: $f \text{ ----> } l \implies (\lambda n. f\ (Suc\ n)) \text{ ----> } l$

$\langle proof \rangle$

lemma $LIMSEQ-imp-Suc$: $(\lambda n. f\ (Suc\ n)) \text{ ----> } l \implies f \text{ ----> } l$

$\langle proof \rangle$

lemma $LIMSEQ-Suc-iff$: $(\lambda n. f\ (Suc\ n)) \text{ ----> } l = f \text{ ----> } l$

$\langle proof \rangle$

lemma *LIMSEQ-linear*: $\llbracket X \text{ ----> } x ; l > 0 \rrbracket \implies (\lambda n. X (n * l)) \text{ ----> } x$
 $\langle \text{proof} \rangle$

lemma *add-diff-add*:
fixes $a\ b\ c\ d :: 'a::\text{ab-group-add}$
shows $(a + c) - (b + d) = (a - b) + (c - d)$
 $\langle \text{proof} \rangle$

lemma *minus-diff-minus*:
fixes $a\ b :: 'a::\text{ab-group-add}$
shows $(- a) - (- b) = - (a - b)$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-add*: $\llbracket X \text{ ----> } a ; Y \text{ ----> } b \rrbracket \implies (\lambda n. X\ n + Y\ n) \text{ ----> } a + b$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-minus*: $X \text{ ----> } a \implies (\lambda n. - X\ n) \text{ ----> } - a$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-minus-cancel*: $(\lambda n. - X\ n) \text{ ----> } - a \implies X \text{ ----> } a$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-diff*: $\llbracket X \text{ ----> } a ; Y \text{ ----> } b \rrbracket \implies (\lambda n. X\ n - Y\ n) \text{ ----> } a - b$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-unique*: $\llbracket X \text{ ----> } a ; X \text{ ----> } b \rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

lemma (*in bounded-linear*) *LIMSEQ*:
 $X \text{ ----> } a \implies (\lambda n. f (X\ n)) \text{ ----> } f a$
 $\langle \text{proof} \rangle$

lemma (*in bounded-bilinear*) *LIMSEQ*:
 $\llbracket X \text{ ----> } a ; Y \text{ ----> } b \rrbracket \implies (\lambda n. X\ n ** Y\ n) \text{ ----> } a ** b$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-mult*:
fixes $a\ b :: 'a::\text{real-normed-algebra}$
shows $\llbracket X \text{ ----> } a ; Y \text{ ----> } b \rrbracket \implies (\%n. X\ n * Y\ n) \text{ ----> } a * b$
 $\langle \text{proof} \rangle$

lemma *inverse-diff-inverse*:
 $\llbracket (a::'a::\text{division-ring}) \neq 0 ; b \neq 0 \rrbracket$
 $\implies \text{inverse } a - \text{inverse } b = - (\text{inverse } a * (a - b) * \text{inverse } b)$
 $\langle \text{proof} \rangle$

lemma *Bseq-inverse-lemma*:

fixes $x :: 'a::\text{real-normed-div-algebra}$
shows $\llbracket r \leq \text{norm } x; 0 < r \rrbracket \implies \text{norm } (\text{inverse } x) \leq \text{inverse } r$
 $\langle \text{proof} \rangle$

lemma *Bseq-inverse*:

fixes $a :: 'a::\text{real-normed-div-algebra}$
assumes $X: X \text{ ----} > a$
assumes $a: a \neq 0$
shows $Bseq (\lambda n. \text{inverse } (X \ n))$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-inverse-lemma*:

fixes $a :: 'a::\text{real-normed-div-algebra}$
shows $\llbracket X \text{ ----} > a; a \neq 0; \forall n. X \ n \neq 0 \rrbracket$
 $\implies (\lambda n. \text{inverse } (X \ n)) \text{ ----} > \text{inverse } a$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-inverse*:

fixes $a :: 'a::\text{real-normed-div-algebra}$
assumes $X: X \text{ ----} > a$
assumes $a: a \neq 0$
shows $(\lambda n. \text{inverse } (X \ n)) \text{ ----} > \text{inverse } a$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-divide*:

fixes $a \ b :: 'a::\text{real-normed-field}$
shows $\llbracket X \text{ ----} > a; Y \text{ ----} > b; b \neq 0 \rrbracket \implies (\lambda n. X \ n / Y \ n) \text{ ----} > a / b$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-pow*:

fixes $a :: 'a::\{\text{real-normed-algebra}, \text{recpower}\}$
shows $X \text{ ----} > a \implies (\lambda n. (X \ n) ^ m) \text{ ----} > a ^ m$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-setsum*:

assumes $n: \bigwedge n. n \in S \implies X \ n \text{ ----} > L \ n$
shows $(\lambda m. \sum_{n \in S} X \ n \ m) \text{ ----} > (\sum_{n \in S} L \ n)$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-setprod*:

fixes $L :: 'a \Rightarrow 'b::\{\text{real-normed-algebra}, \text{comm-ring-1}\}$
assumes $n: \bigwedge n. n \in S \implies X \ n \text{ ----} > L \ n$
shows $(\lambda m. \prod_{n \in S} X \ n \ m) \text{ ----} > (\prod_{n \in S} L \ n)$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-add-const*: $f \text{ ----} > a \implies (\%n.(f \ n + b)) \text{ ----} > a + b$

$\langle proof \rangle$

lemma *LIMSEQ-add-minus*:

$[[X \text{ ----} > a; Y \text{ ----} > b]] \implies (\%n. X\ n + -Y\ n) \text{ ----} > a + -b$
 $\langle proof \rangle$

lemma *LIMSEQ-diff-const*: $f \text{ ----} > a \implies (\%n. (f\ n - b)) \text{ ----} > a - b$
 $\langle proof \rangle$

lemma *LIMSEQ-diff-approach-zero*:

$g \text{ ----} > L \implies (\%x. f\ x - g\ x) \text{ ----} > 0 \implies$
 $f \text{ ----} > L$
 $\langle proof \rangle$

lemma *LIMSEQ-diff-approach-zero2*:

$f \text{ ----} > L \implies (\%x. f\ x - g\ x) \text{ ----} > 0 \implies$
 $g \text{ ----} > L$
 $\langle proof \rangle$

A sequence tends to zero iff its abs does

lemma *LIMSEQ-norm-zero*: $((\lambda n. \text{norm } (X\ n)) \text{ ----} > 0) = (X \text{ ----} > 0)$
 $\langle proof \rangle$

lemma *LIMSEQ-rabs-zero*: $((\%n. |f\ n|) \text{ ----} > 0) = (f \text{ ----} > (0::real))$
 $\langle proof \rangle$

lemma *LIMSEQ-imp-rabs*: $f \text{ ----} > (l::real) \implies (\%n. |f\ n|) \text{ ----} > |l|$
 $\langle proof \rangle$

An unbounded sequence’s inverse tends to 0

lemma *LIMSEQ-inverse-zero*:

$\forall r::real. \exists N. \forall n \geq N. r < X\ n \implies (\lambda n. \text{inverse } (X\ n)) \text{ ----} > 0$
 $\langle proof \rangle$

The sequence $(1::'a) / n$ tends to 0 as n tends to infinity

lemma *LIMSEQ-inverse-real-of-nat*: $(\%n. \text{inverse}(\text{real}(\text{Suc } n))) \text{ ----} > 0$
 $\langle proof \rangle$

The sequence $r + (1::'a) / n$ tends to r as n tends to infinity is now easily proved

lemma *LIMSEQ-inverse-real-of-nat-add*:

$(\%n. r + \text{inverse}(\text{real}(\text{Suc } n))) \text{ ----} > r$
 $\langle proof \rangle$

lemma *LIMSEQ-inverse-real-of-nat-add-minus*:

$(\%n. r + -\text{inverse}(\text{real}(\text{Suc } n))) \text{ ----} > r$
 $\langle proof \rangle$

lemma *LIMSEQ-inverse-real-of-nat-add-minus-mult*:

$(\%n. r * (1 + -inverse(real(Suc n)))) \text{---->} r$
 $\langle proof \rangle$

lemma *LIMSEQ-le-const*:

$\llbracket X \text{---->} (x::real); \exists N. \forall n \geq N. a \leq X n \rrbracket \implies a \leq x$
 $\langle proof \rangle$

lemma *LIMSEQ-le-const2*:

$\llbracket X \text{---->} (x::real); \exists N. \forall n \geq N. X n \leq a \rrbracket \implies x \leq a$
 $\langle proof \rangle$

lemma *LIMSEQ-le*:

$\llbracket X \text{---->} x; Y \text{---->} y; \exists N. \forall n \geq N. X n \leq Y n \rrbracket \implies x \leq (y::real)$
 $\langle proof \rangle$

57.4 Convergence

lemma *limI*: $X \text{---->} L \implies \lim X = L$

$\langle proof \rangle$

lemma *convergentD*: $\text{convergent } X \implies \exists L. (X \text{---->} L)$

$\langle proof \rangle$

lemma *convergentI*: $(X \text{---->} L) \implies \text{convergent } X$

$\langle proof \rangle$

lemma *convergent-LIMSEQ-iff*: $\text{convergent } X = (X \text{---->} \lim X)$

$\langle proof \rangle$

lemma *convergent-minus-iff*: $(\text{convergent } X) = (\text{convergent } (\%n. -(X n)))$

$\langle proof \rangle$

Given a binary function $f:: \text{nat} \Rightarrow 'a \Rightarrow 'a$, its values are uniquely determined by a function g

lemma *nat-function-unique*: $EX! g. g \ 0 = e \wedge (\forall n. g \ (Suc \ n) = f \ n \ (g \ n))$

$\langle proof \rangle$

Subsequence (alternative definition, (e.g. Hoskins))

lemma *subseq-Suc-iff*: $\text{subseq } f = (\forall n. (f \ n) < (f \ (Suc \ n)))$

$\langle proof \rangle$

lemma *monoseq-Suc*:

$\text{monoseq } X = ((\forall n. X \ n \leq X \ (Suc \ n))$
 $\quad | (\forall n. X \ (Suc \ n) \leq X \ n))$

$\langle proof \rangle$

lemma *monoI1*: $\forall m. \forall n \geq m. X \ m \leq X \ n \implies \text{monoseq } X$

$\langle \text{proof} \rangle$

lemma *monoI2*: $\forall m. \forall n \geq m. X\ n \leq X\ m \implies \text{monoseq}\ X$
 $\langle \text{proof} \rangle$

lemma *mono-SucI1*: $\forall n. X\ n \leq X\ (\text{Suc}\ n) \implies \text{monoseq}\ X$
 $\langle \text{proof} \rangle$

lemma *mono-SucI2*: $\forall n. X\ (\text{Suc}\ n) \leq X\ n \implies \text{monoseq}\ X$
 $\langle \text{proof} \rangle$

lemma *monoseq-minus*: **assumes** *monoseq a*
shows *monoseq* $(\lambda n. -\ a\ n)$
 $\langle \text{proof} \rangle$

lemma *monoseq-le*: **assumes** *monoseq a* **and** $a\ \text{----}\>\ x$
shows $((\forall n. a\ n \leq x) \wedge (\forall m. \forall n \geq m. a\ m \leq a\ n)) \vee$
 $((\forall n. x \leq a\ n) \wedge (\forall m. \forall n \geq m. a\ n \leq a\ m))$
 $\langle \text{proof} \rangle$

for any sequence, there is a monotonic subsequence

lemma *seq-monosub*: $\exists f. \text{subseq}\ f \wedge \text{monoseq}\ (\lambda n. (s\ (f\ n)))$
 $\langle \text{proof} \rangle$

lemma *seq-suble*: **assumes** *sf: subseq f* **shows** $n \leq f\ n$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-subseq-LIMSEQ*:
 $\llbracket X\ \text{----}\>\ L; \text{subseq}\ f \rrbracket \implies (X\ o\ f)\ \text{----}\>\ L$
 $\langle \text{proof} \rangle$

57.5 Bounded Monotonic Sequences

Bounded Sequence

lemma *BseqD*: *Bseq X* $\implies \exists K. 0 < K \ \& \ (\forall n. \text{norm}\ (X\ n) \leq K)$
 $\langle \text{proof} \rangle$

lemma *BseqI*: $\llbracket 0 < K; \forall n. \text{norm}\ (X\ n) \leq K \rrbracket \implies \text{Bseq}\ X$
 $\langle \text{proof} \rangle$

lemma *lemma-NBseq-def*:
 $(\exists K > 0. \forall n. \text{norm}\ (X\ n) \leq K) =$
 $(\exists N. \forall n. \text{norm}\ (X\ n) \leq \text{real}(\text{Suc}\ N))$
 $\langle \text{proof} \rangle$

alternative definition for Bseq

lemma *Bseq-iff*: *Bseq X* $= (\exists N. \forall n. \text{norm}\ (X\ n) \leq \text{real}(\text{Suc}\ N))$
 $\langle \text{proof} \rangle$

lemma *lemma-NBseq-def2*:

$(\exists K > 0. \forall n. \text{norm } (X\ n) \leq K) = (\exists N. \forall n. \text{norm } (X\ n) < \text{real}(Suc\ N))$
 $\langle \text{proof} \rangle$

lemma *Bseq-iff1a*: $Bseq\ X = (\exists N. \forall n. \text{norm } (X\ n) < \text{real}(Suc\ N))$

$\langle \text{proof} \rangle$

57.5.1 Upper Bounds and Lubs of Bounded Sequences

lemma *Bseq-isUb*:

$!!(X::\text{nat} \Rightarrow \text{real}). Bseq\ X \Rightarrow \exists U. \text{isUb } (UNIV::\text{real set}) \{x. \exists n. X\ n = x\}\ U$
 $\langle \text{proof} \rangle$

Use completeness of reals (supremum property) to show that any bounded sequence has a least upper bound

lemma *Bseq-isLub*:

$!!(X::\text{nat} \Rightarrow \text{real}). Bseq\ X \Rightarrow$
 $\exists U. \text{isLub } (UNIV::\text{real set}) \{x. \exists n. X\ n = x\}\ U$
 $\langle \text{proof} \rangle$

57.5.2 A Bounded and Monotonic Sequence Converges

lemma *lemma-converg1*:

$!!(X::\text{nat} \Rightarrow \text{real}). [\forall m. \forall n \geq m. X\ m \leq X\ n;$
 $\text{isLub } (UNIV::\text{real set}) \{x. \exists n. X\ n = x\} (X\ ma)$
 $]\Rightarrow \forall n \geq ma. X\ n = X\ ma$
 $\langle \text{proof} \rangle$

The best of both worlds: Easier to prove this result as a standard theorem and then use equivalence to ”transfer” it into the equivalent nonstandard form if needed!

lemma *Bmonoseq-LIMSEQ*: $\forall n. m \leq n \longrightarrow X\ n = X\ m \Rightarrow \exists L. (X \dashrightarrow L)$

$\langle \text{proof} \rangle$

lemma *lemma-converg2*:

$!!(X::\text{nat} \Rightarrow \text{real}).$
 $[\forall m. X\ m \sim U; \text{isLub } UNIV \{x. \exists n. X\ n = x\}\ U] \Rightarrow \forall m. X\ m < U$
 $\langle \text{proof} \rangle$

lemma *lemma-converg3*: $!!(X::\text{nat} \Rightarrow \text{real}). \forall m. X\ m \leq U \Rightarrow \text{isUb } UNIV \{x. \exists n. X\ n = x\}\ U$

$\langle \text{proof} \rangle$

FIXME: $U - T < U$ is redundant

lemma *lemma-converg4*: $!!(X::\text{nat} \Rightarrow \text{real}).$

$[\forall m. X\ m \sim U;$

$$\begin{aligned}
& \text{isLub UNIV } \{x. \exists n. X\ n = x\} \ U; \\
& 0 < T; \\
& U + -\ T < U \\
& [] ==> \exists m. U + -T < X\ m \ \& \ X\ m < U
\end{aligned}$$

⟨proof⟩

A standard proof of the theorem for monotone increasing sequence

lemma *Bseq-mono-convergent*:

$$[] \text{ Bseq } X; \forall m. \forall n \geq m. X\ m \leq X\ n \implies \text{convergent } (X::\text{nat} \Rightarrow \text{real})$$

⟨proof⟩

lemma *Bseq-minus-iff*: $\text{Bseq } (\%n. -(X\ n)) = \text{Bseq } X$

⟨proof⟩

Main monotonicity theorem

lemma *Bseq-monoseq-convergent*: $[] \text{ Bseq } X; \text{monoseq } X \implies \text{convergent } X$

⟨proof⟩

57.5.3 Increasing and Decreasing Series

lemma *incseq-imp-monoseq*: $\text{incseq } X \implies \text{monoseq } X$

⟨proof⟩

lemma *incseq-le*: **assumes** *inc*: $\text{incseq } X$ **and** *lim*: $X \dashrightarrow L$ **shows** $X\ n \leq L$

⟨proof⟩

lemma *decseq-imp-monoseq*: $\text{decseq } X \implies \text{monoseq } X$

⟨proof⟩

lemma *decseq-eq-incseq*: $\text{decseq } X = \text{incseq } (\lambda n. - X\ n)$

⟨proof⟩

lemma *decseq-le*: **assumes** *dec*: $\text{decseq } X$ **and** *lim*: $X \dashrightarrow L$ **shows** $L \leq X\ n$

⟨proof⟩

57.5.4 A Few More Equivalence Theorems for Boundedness

alternative formulation for boundedness

lemma *Bseq-iff2*: $\text{Bseq } X = (\exists k > 0. \exists x. \forall n. \text{norm } (X(n) + -x) \leq k)$

⟨proof⟩

alternative formulation for boundedness

lemma *Bseq-iff3*: $\text{Bseq } X = (\exists k > 0. \exists N. \forall n. \text{norm}(X(n) + -X(N)) \leq k)$

⟨proof⟩

lemma *BseqI2*: $(\forall n. k \leq f\ n \ \& \ f\ n \leq (K::real)) \implies Bseq\ f$
 $\langle proof \rangle$

57.6 Cauchy Sequences

lemma *CauchyI*:

$(\bigwedge e. 0 < e \implies \exists M. \forall m \geq M. \forall n \geq M. norm\ (X\ m - X\ n) < e) \implies Cauchy\ X$
 $\langle proof \rangle$

lemma *CauchyD*:

$\llbracket Cauchy\ X; 0 < e \rrbracket \implies \exists M. \forall m \geq M. \forall n \geq M. norm\ (X\ m - X\ n) < e$
 $\langle proof \rangle$

lemma *Cauchy-subseq-Cauchy*:

$\llbracket Cauchy\ X; subseq\ f \rrbracket \implies Cauchy\ (X\ o\ f)$
 $\langle proof \rangle$

57.6.1 Cauchy Sequences are Bounded

A Cauchy sequence is bounded – this is the standard proof mechanization rather than the nonstandard proof

lemma *lemmaCauchy*: $\forall n \geq M. norm\ (X\ M - X\ n) < (1::real)$
 $\implies \forall n \geq M. norm\ (X\ n :: 'a::real-normed-vector) < 1 + norm\ (X\ M)$
 $\langle proof \rangle$

lemma *Cauchy-Bseq*: $Cauchy\ X \implies Bseq\ X$
 $\langle proof \rangle$

57.6.2 Cauchy Sequences are Convergent

axclass *banach* \subseteq *real-normed-vector*

Cauchy-convergent: $Cauchy\ X \implies convergent\ X$

theorem *LIMSEQ-imp-Cauchy*:

assumes $X: X \dashrightarrow a$ **shows** $Cauchy\ X$
 $\langle proof \rangle$

lemma *convergent-Cauchy*: $convergent\ X \implies Cauchy\ X$
 $\langle proof \rangle$

Proof that Cauchy sequences converge based on the one from <http://pirate.shu.edu/~wachsmut/ira/nu>

If sequence X is Cauchy, then its limit is the lub of $\{r. \exists N. \forall n \geq N. r < X\ n\}$

lemma *isUb-UNIV-I*: $(\bigwedge y. y \in S \implies y \leq u) \implies isUb\ UNIV\ S\ u$
 $\langle proof \rangle$

lemma *real-abs-diff-less-iff*:

$(|x - a| < (r::real)) = (a - r < x \wedge x < a + r)$

<proof>

locale *real-Cauchy* =
fixes $X :: \text{nat} \Rightarrow \text{real}$
assumes $X: \text{Cauchy } X$
fixes $S :: \text{real set}$
defines $S\text{-def}: S \equiv \{x::\text{real}. \exists N. \forall n \geq N. x < X \ n\}$

lemma *real-CauchyI*:
assumes $\text{Cauchy } X$
shows $\text{real-Cauchy } X$
<proof>

lemma (**in** *real-Cauchy*) *mem-S*: $\forall n \geq N. x < X \ n \implies x \in S$
<proof>

lemma (**in** *real-Cauchy*) *bound-isUb*:
assumes $N: \forall n \geq N. X \ n < x$
shows $\text{isUb UNIV } S \ x$
<proof>

lemma (**in** *real-Cauchy*) *isLub-ex*: $\exists u. \text{isLub UNIV } S \ u$
<proof>

lemma (**in** *real-Cauchy*) *isLub-imp-LIMSEQ*:
assumes $x: \text{isLub UNIV } S \ x$
shows $X \text{ ----} > x$
<proof>

lemma (**in** *real-Cauchy*) *LIMSEQ-ex*: $\exists x. X \text{ ----} > x$
<proof>

lemma *real-Cauchy-convergent*:
fixes $X :: \text{nat} \Rightarrow \text{real}$
shows $\text{Cauchy } X \implies \text{convergent } X$
<proof>

instance *real* :: *banach*
<proof>

lemma *Cauchy-convergent-iff*:
fixes $X :: \text{nat} \Rightarrow 'a::\text{banach}$
shows $\text{Cauchy } X = \text{convergent } X$
<proof>

lemma *convergent-subseq-convergent*:
fixes $X :: \text{nat} \Rightarrow 'a::\text{banach}$
shows $\llbracket \text{convergent } X; \text{subseq } f \rrbracket \implies \text{convergent } (X \circ f)$
<proof>

57.7 Power Sequences

The sequence x^n tends to 0 if $(0::'a) \leq x$ and $x < (1::'a)$. Proof will use (NS) Cauchy equivalence for convergence and also fact that bounded and monotonic sequence converges.

lemma *Bseq-realpow*: $\llbracket 0 \leq (x::real); x \leq 1 \rrbracket \implies Bseq (\%n. x^n)$
 $\langle proof \rangle$

lemma *monoseq-realpow*: $\llbracket 0 \leq x; x \leq 1 \rrbracket \implies monoseq (\%n. x^n)$
 $\langle proof \rangle$

lemma *convergent-realpow*:
 $\llbracket 0 \leq (x::real); x \leq 1 \rrbracket \implies convergent (\%n. x^n)$
 $\langle proof \rangle$

lemma *LIMSEQ-inverse-realpow-zero-lemma*:
fixes $x :: real$
assumes $x: 0 \leq x$
shows $real\ n * x + 1 \leq (x + 1)^n$
 $\langle proof \rangle$

lemma *LIMSEQ-inverse-realpow-zero*:
 $1 < (x::real) \implies (\lambda n. inverse\ (x^n)) \dashrightarrow 0$
 $\langle proof \rangle$

lemma *LIMSEQ-realpow-zero*:
 $\llbracket 0 \leq (x::real); x < 1 \rrbracket \implies (\lambda n. x^n) \dashrightarrow 0$
 $\langle proof \rangle$

lemma *LIMSEQ-power-zero*:
fixes $x :: 'a::\{real-normed-algebra-1,recpower\}$
shows $norm\ x < 1 \implies (\lambda n. x^n) \dashrightarrow 0$
 $\langle proof \rangle$

lemma *LIMSEQ-divide-realpow-zero*:
 $1 < (x::real) \implies (\%n. a / (x^n)) \dashrightarrow 0$
 $\langle proof \rangle$

Limit of c^n for $|c| < (1::'a)$

lemma *LIMSEQ-rabs-realpow-zero*: $|c| < (1::real) \implies (\%n. |c|^n) \dashrightarrow 0$
 $\langle proof \rangle$

lemma *LIMSEQ-rabs-realpow-zero2*: $|c| < (1::real) \implies (\%n. c^n) \dashrightarrow 0$
 $\langle proof \rangle$

end

58 Series: Finite Summation and Infinite Series

```
theory Series
imports SEQ
begin
```

definition

```
sums :: (nat ⇒ 'a::real-normed-vector) ⇒ 'a ⇒ bool
  (infixr sums 80) where
  f sums s = (%n. setsum f {0..

```

definition

```
summable :: (nat ⇒ 'a::real-normed-vector) ⇒ bool where
  summable f = (∃ s. f sums s)
```

definition

```
suminf :: (nat ⇒ 'a::real-normed-vector) ⇒ 'a where
  suminf f = (THE s. f sums s)
```

syntax

```
-suminf :: idt ⇒ 'a ⇒ 'a (∑ -. - [0, 10] 10)
```

translations

```
∑ i. b == CONST suminf (%i. b)
```

lemma *sumr-diff-mult-const*:

```
setsum f {0..

```

lemma *real-setsum-nat-ivl-bounded*:

```
(!p. p < n ⇒ f(p) ≤ K)
⇒ setsum f {0..

```

lemma *sumr-minus-one-realpow-zero* [simp]:

```
(∑ i=0..<2*n. (-1) ^ Suc i) = (0::real)
⟨proof⟩
```

lemma *sumr-one-lb-realpow-zero* [simp]:

```
(∑ n=Suc 0..

```

lemma *sumr-group*:

```
(∑ m=0..

```

lemma *sumr-offset3*:

$\text{setsum } f \{0::\text{nat}..<n+k\} = (\sum m=0..<n. f(m+k)) + \text{setsum } f \{0..<k\}$
 $\langle \text{proof} \rangle$

lemma *sumr-offset*:

fixes $f :: \text{nat} \Rightarrow 'a::\text{ab-group-add}$

shows $(\sum m=0..<n. f(m+k)) = \text{setsum } f \{0..<n+k\} - \text{setsum } f \{0..<k\}$
 $\langle \text{proof} \rangle$

lemma *sumr-offset2*:

$\forall f. (\sum m=0..<n::\text{nat}. f(m+k)::\text{real}) = \text{setsum } f \{0..<n+k\} - \text{setsum } f \{0..<k\}$
 $\langle \text{proof} \rangle$

lemma *sumr-offset4*:

$\forall n f. \text{setsum } f \{0::\text{nat}..<n+k\} = (\sum m=0..<n. f(m+k)::\text{real}) + \text{setsum } f \{0..<k\}$
 $\langle \text{proof} \rangle$

58.1 Infinite Sums, by the Properties of Limits

lemma *sums-summable*: $f \text{ sums } l \implies \text{summable } f$

$\langle \text{proof} \rangle$

lemma *summable-sums*: $\text{summable } f \implies f \text{ sums } (\text{suminf } f)$

$\langle \text{proof} \rangle$

lemma *summable-sumr-LIMSEQ-suminf*:

$\text{summable } f \implies (\%n. \text{setsum } f \{0..<n\}) \dashrightarrow (\text{suminf } f)$
 $\langle \text{proof} \rangle$

lemma *sums-unique*: $f \text{ sums } s \implies (s = \text{suminf } f)$

$\langle \text{proof} \rangle$

lemma *sums-split-initial-segment*: $f \text{ sums } s \implies$

$(\%n. f(n+k)) \text{ sums } (s - (\text{SUM } i = 0..<k. f i))$

$\langle \text{proof} \rangle$

lemma *summable-ignore-initial-segment*: $\text{summable } f \implies$

$\text{summable } (\%n. f(n+k))$

$\langle \text{proof} \rangle$

lemma *suminf-minus-initial-segment*: $\text{summable } f \implies$

$\text{suminf } f = s \implies \text{suminf } (\%n. f(n+k)) = s - (\text{SUM } i = 0..<k. f i)$

$\langle \text{proof} \rangle$

lemma *suminf-split-initial-segment*: $\text{summable } f \implies$

$\text{suminf } f = (\text{SUM } i = 0..<k. f i) + \text{suminf } (\%n. f(n+k))$

$\langle \text{proof} \rangle$

lemma *suminf-exist-split*: **fixes** $r :: \text{real}$ **assumes** $0 < r$ **and** *summable* a
shows $\exists N. \forall n \geq N. |\sum i. a (i + n)| < r$
 $\langle \text{proof} \rangle$

lemma *sums-Suc*: **assumes** *sumSuc*: $(\lambda n. f (Suc\ n)) \text{ sums } l$ **shows** $f \text{ sums } (l + f\ 0)$
 $\langle \text{proof} \rangle$

lemma *series-zero*:
 $(\forall m. n \leq m \longrightarrow f(m) = 0) \implies f \text{ sums } (\text{setsum } f \ \{0..<n\})$
 $\langle \text{proof} \rangle$

lemma *sums-zero*: $(\lambda n. 0) \text{ sums } 0$
 $\langle \text{proof} \rangle$

lemma *summable-zero*: *summable* $(\lambda n. 0)$
 $\langle \text{proof} \rangle$

lemma *suminf-zero*: *suminf* $(\lambda n. 0) = 0$
 $\langle \text{proof} \rangle$

lemma (**in** *bounded-linear*) *sums*:
 $(\lambda n. X\ n) \text{ sums } a \implies (\lambda n. f\ (X\ n)) \text{ sums } (f\ a)$
 $\langle \text{proof} \rangle$

lemma (**in** *bounded-linear*) *summable*:
 $\text{summable } (\lambda n. X\ n) \implies \text{summable } (\lambda n. f\ (X\ n))$
 $\langle \text{proof} \rangle$

lemma (**in** *bounded-linear*) *suminf*:
 $\text{summable } (\lambda n. X\ n) \implies f\ (\sum n. X\ n) = (\sum n. f\ (X\ n))$
 $\langle \text{proof} \rangle$

lemma *sums-mult*:
fixes $c :: 'a::\text{real-normed-algebra}$
shows $f \text{ sums } a \implies (\lambda n. c * f\ n) \text{ sums } (c * a)$
 $\langle \text{proof} \rangle$

lemma *summable-mult*:
fixes $c :: 'a::\text{real-normed-algebra}$
shows $\text{summable } f \implies \text{summable } (\%n. c * f\ n)$
 $\langle \text{proof} \rangle$

lemma *suminf-mult*:
fixes $c :: 'a::\text{real-normed-algebra}$
shows $\text{summable } f \implies \text{suminf } (\lambda n. c * f\ n) = c * \text{suminf } f$
 $\langle \text{proof} \rangle$

lemma *sums-mult2*:

fixes $c :: 'a::\text{real-normed-algebra}$
shows $f \text{ sums } a \implies (\lambda n. f\ n * c) \text{ sums } (a * c)$
 $\langle \text{proof} \rangle$

lemma *summable-mult2*:
fixes $c :: 'a::\text{real-normed-algebra}$
shows $\text{summable } f \implies \text{summable } (\lambda n. f\ n * c)$
 $\langle \text{proof} \rangle$

lemma *suminf-mult2*:
fixes $c :: 'a::\text{real-normed-algebra}$
shows $\text{summable } f \implies \text{suminf } f * c = (\sum n. f\ n * c)$
 $\langle \text{proof} \rangle$

lemma *sums-divide*:
fixes $c :: 'a::\text{real-normed-field}$
shows $f \text{ sums } a \implies (\lambda n. f\ n / c) \text{ sums } (a / c)$
 $\langle \text{proof} \rangle$

lemma *summable-divide*:
fixes $c :: 'a::\text{real-normed-field}$
shows $\text{summable } f \implies \text{summable } (\lambda n. f\ n / c)$
 $\langle \text{proof} \rangle$

lemma *suminf-divide*:
fixes $c :: 'a::\text{real-normed-field}$
shows $\text{summable } f \implies \text{suminf } (\lambda n. f\ n / c) = \text{suminf } f / c$
 $\langle \text{proof} \rangle$

lemma *sums-add*: $\llbracket X \text{ sums } a; Y \text{ sums } b \rrbracket \implies (\lambda n. X\ n + Y\ n) \text{ sums } (a + b)$
 $\langle \text{proof} \rangle$

lemma *summable-add*: $\llbracket \text{summable } X; \text{summable } Y \rrbracket \implies \text{summable } (\lambda n. X\ n + Y\ n)$
 $\langle \text{proof} \rangle$

lemma *suminf-add*:
 $\llbracket \text{summable } X; \text{summable } Y \rrbracket \implies \text{suminf } X + \text{suminf } Y = (\sum n. X\ n + Y\ n)$
 $\langle \text{proof} \rangle$

lemma *sums-diff*: $\llbracket X \text{ sums } a; Y \text{ sums } b \rrbracket \implies (\lambda n. X\ n - Y\ n) \text{ sums } (a - b)$
 $\langle \text{proof} \rangle$

lemma *summable-diff*: $\llbracket \text{summable } X; \text{summable } Y \rrbracket \implies \text{summable } (\lambda n. X\ n - Y\ n)$
 $\langle \text{proof} \rangle$

lemma *suminf-diff*:
 $\llbracket \text{summable } X; \text{summable } Y \rrbracket \implies \text{suminf } X - \text{suminf } Y = (\sum n. X\ n - Y\ n)$

<proof>

lemma *sums-minus*: $X \text{ sums } a \implies (\lambda n. - X n) \text{ sums } (- a)$
<proof>

lemma *summable-minus*: $\text{summable } X \implies \text{summable } (\lambda n. - X n)$
<proof>

lemma *suminf-minus*: $\text{summable } X \implies (\sum n. - X n) = - (\sum n. X n)$
<proof>

lemma *sums-group*:
 $[\text{summable } f; 0 < k] \implies (\%n. \text{setsum } f \{n*k..<n*k+k\}) \text{ sums } (\text{suminf } f)$
<proof>

A summable series of positive terms has limit that is at least as great as any partial sum.

lemma *series-pos-le*:
fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $[\text{summable } f; \forall m \geq n. 0 \leq f m] \implies \text{setsum } f \{0..<n\} \leq \text{suminf } f$
<proof>

lemma *series-pos-less*:
fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $[\text{summable } f; \forall m \geq n. 0 < f m] \implies \text{setsum } f \{0..<n\} < \text{suminf } f$
<proof>

lemma *suminf-gt-zero*:
fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $[\text{summable } f; \forall n. 0 < f n] \implies 0 < \text{suminf } f$
<proof>

lemma *suminf-ge-zero*:
fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $[\text{summable } f; \forall n. 0 \leq f n] \implies 0 \leq \text{suminf } f$
<proof>

lemma *sumr-pos-lt-pair*:
fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $[\text{summable } f;$
 $\forall d. 0 < f (k + (\text{Suc}(\text{Suc } 0) * d)) + f (k + ((\text{Suc}(\text{Suc } 0) * d) + 1))]$
 $\implies \text{setsum } f \{0..<k\} < \text{suminf } f$
<proof>

Sum of a geometric progression.

lemmas *sumr-geometric* = *geometric-sum* [**where** $'a = \text{real}$]

lemma *geometric-sums*:
fixes $x :: 'a :: \{\text{real-normed-field}, \text{recpower}\}$

shows $\text{norm } x < 1 \implies (\lambda n. x \wedge n) \text{ sums } (1 / (1 - x))$
 $\langle \text{proof} \rangle$

lemma *summable-geometric*:
fixes $x :: 'a :: \{\text{real-normed-field}, \text{recpower}\}$
shows $\text{norm } x < 1 \implies \text{summable } (\lambda n. x \wedge n)$
 $\langle \text{proof} \rangle$

Cauchy-type criterion for convergence of series (c.f. Harrison)

lemma *summable-convergent-sumr-iff*:
 $\text{summable } f = \text{convergent } (\%n. \text{setsum } f \{0..<n\})$
 $\langle \text{proof} \rangle$

lemma *summable-LIMSEQ-zero*: $\text{summable } f \implies f \text{ ----} > 0$
 $\langle \text{proof} \rangle$

lemma *summable-Cauchy*:
 $\text{summable } (f :: \text{nat} \Rightarrow 'a :: \text{banach}) =$
 $(\forall e > 0. \exists N. \forall m \geq N. \forall n. \text{norm } (\text{setsum } f \{m..<n\}) < e)$
 $\langle \text{proof} \rangle$

Comparison test

lemma *norm-setsum*:
fixes $f :: 'a \Rightarrow 'b :: \text{real-normed-vector}$
shows $\text{norm } (\text{setsum } f A) \leq (\sum_{i \in A} \text{norm } (f i))$
 $\langle \text{proof} \rangle$

lemma *summable-comparison-test*:
fixes $f :: \text{nat} \Rightarrow 'a :: \text{banach}$
shows $[\exists N. \forall n \geq N. \text{norm } (f n) \leq g n; \text{summable } g] \implies \text{summable } f$
 $\langle \text{proof} \rangle$

lemma *summable-norm-comparison-test*:
fixes $f :: \text{nat} \Rightarrow 'a :: \text{banach}$
shows $[\exists N. \forall n \geq N. \text{norm } (f n) \leq g n; \text{summable } g]$
 $\implies \text{summable } (\lambda n. \text{norm } (f n))$
 $\langle \text{proof} \rangle$

lemma *summable-rabs-comparison-test*:
fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $[\exists N. \forall n \geq N. |f n| \leq g n; \text{summable } g] \implies \text{summable } (\lambda n. |f n|)$
 $\langle \text{proof} \rangle$

Summability of geometric series for real algebras

lemma *complete-algebra-summable-geometric*:
fixes $x :: 'a :: \{\text{real-normed-algebra-1}, \text{banach}, \text{recpower}\}$
shows $\text{norm } x < 1 \implies \text{summable } (\lambda n. x \wedge n)$
 $\langle \text{proof} \rangle$

Limit comparison property for series (c.f. jrh)

lemma *summable-le*:

fixes $f\ g :: \text{nat} \Rightarrow \text{real}$
shows $\llbracket \forall n. f\ n \leq g\ n; \text{summable } f; \text{summable } g \rrbracket \Longrightarrow \text{suminf } f \leq \text{suminf } g$
 $\langle \text{proof} \rangle$

lemma *summable-le2*:

fixes $f\ g :: \text{nat} \Rightarrow \text{real}$
shows $\llbracket \forall n. |f\ n| \leq g\ n; \text{summable } g \rrbracket \Longrightarrow \text{summable } f \wedge \text{suminf } f \leq \text{suminf } g$
 $\langle \text{proof} \rangle$

lemma *suminf-0-le*:

fixes $f :: \text{nat} \Rightarrow \text{real}$
assumes $gt0: \forall n. 0 \leq f\ n$ **and** $sm: \text{summable } f$
shows $0 \leq \text{suminf } f$
 $\langle \text{proof} \rangle$

Absolute convergence implies normal convergence

lemma *summable-norm-cancel*:

fixes $f :: \text{nat} \Rightarrow 'a::\text{banach}$
shows $\text{summable } (\lambda n. \text{norm } (f\ n)) \Longrightarrow \text{summable } f$
 $\langle \text{proof} \rangle$

lemma *summable-rabs-cancel*:

fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $\text{summable } (\lambda n. |f\ n|) \Longrightarrow \text{summable } f$
 $\langle \text{proof} \rangle$

Absolute convergence of series

lemma *summable-norm*:

fixes $f :: \text{nat} \Rightarrow 'a::\text{banach}$
shows $\text{summable } (\lambda n. \text{norm } (f\ n)) \Longrightarrow \text{norm } (\text{suminf } f) \leq (\sum n. \text{norm } (f\ n))$
 $\langle \text{proof} \rangle$

lemma *summable-rabs*:

fixes $f :: \text{nat} \Rightarrow \text{real}$
shows $\text{summable } (\lambda n. |f\ n|) \Longrightarrow |\text{suminf } f| \leq (\sum n. |f\ n|)$
 $\langle \text{proof} \rangle$

58.2 The Ratio Test

lemma *norm-ratiotest-lemma*:

fixes $x\ y :: 'a::\text{real-normed-vector}$
shows $\llbracket c \leq 0; \text{norm } x \leq c * \text{norm } y \rrbracket \Longrightarrow x = 0$
 $\langle \text{proof} \rangle$

lemma *rabs-ratiotest-lemma*: $\llbracket c \leq 0; \text{abs } x \leq c * \text{abs } y \rrbracket \Longrightarrow x = (0::\text{real})$

$\langle \text{proof} \rangle$

lemma *le-Suc-ex*: $(k::nat) \leq l \implies (\exists n. l = k + n)$
 $\langle proof \rangle$

lemma *le-Suc-ex-iff*: $((k::nat) \leq l) = (\exists n. l = k + n)$
 $\langle proof \rangle$

lemma *ratio-test-lemma2*:
fixes $f :: nat \Rightarrow 'a::banach$
shows $\llbracket \forall n \geq N. norm (f (Suc n)) \leq c * norm (f n) \rrbracket \implies 0 < c \vee summable f$
 $\langle proof \rangle$

lemma *ratio-test*:
fixes $f :: nat \Rightarrow 'a::banach$
shows $\llbracket c < 1; \forall n \geq N. norm (f (Suc n)) \leq c * norm (f n) \rrbracket \implies summable f$
 $\langle proof \rangle$

58.3 Cauchy Product Formula

lemma *setsum-triangle-reindex*:
fixes $n :: nat$
shows $(\sum_{(i,j) \in \{(i,j). i+j < n\}} f i j) = (\sum_{k=0..<n. \sum_{i=0..k} f i (k-i))$
 $\langle proof \rangle$

lemma *Cauchy-product-sums*:
fixes $a b :: nat \Rightarrow 'a::\{real-normed-algebra, banach\}$
assumes $a: summable (\lambda k. norm (a k))$
assumes $b: summable (\lambda k. norm (b k))$
shows $(\lambda k. \sum_{i=0..k} a i * b (k-i)) sums ((\sum k. a k) * (\sum k. b k))$
 $\langle proof \rangle$

lemma *Cauchy-product*:
fixes $a b :: nat \Rightarrow 'a::\{real-normed-algebra, banach\}$
assumes $a: summable (\lambda k. norm (a k))$
assumes $b: summable (\lambda k. norm (b k))$
shows $(\sum k. a k) * (\sum k. b k) = (\sum k. \sum_{i=0..k} a i * b (k-i))$
 $\langle proof \rangle$

end

59 Lim: Limits and Continuity

theory *Lim*
imports *SEQ*
begin

Standard Definitions

definition

$LIM :: [a::real-normed-vector \Rightarrow b::real-normed-vector, 'a, 'b] \Rightarrow bool$
 $((-)/ \dashv (-)/ \dashv (-)) [60, 0, 60] 60$ **where**
 $[code\ del]: f \dashv a \dashv L =$
 $(\forall r > 0. \exists s > 0. \forall x. x \neq a \ \& \ norm\ (x - a) < s$
 $\dashv norm\ (f\ x - L) < r)$

definition

$isCont :: [a::real-normed-vector \Rightarrow b::real-normed-vector, 'a] \Rightarrow bool$ **where**
 $isCont\ f\ a = (f \dashv a \dashv (f\ a))$

definition

$isUCont :: [a::real-normed-vector \Rightarrow b::real-normed-vector] \Rightarrow bool$ **where**
 $[code\ del]: isUCont\ f = (\forall r > 0. \exists s > 0. \forall x\ y. norm\ (x - y) < s \longrightarrow norm\ (f\ x - f\ y) < r)$

59.1 Limits of Functions**59.1.1 Purely standard proofs****lemma LIM-eq:**

$f \dashv a \dashv L =$
 $(\forall r > 0. \exists s > 0. \forall x. x \neq a \ \& \ norm\ (x - a) < s \dashv norm\ (f\ x - L) < r)$
 $\langle proof \rangle$

lemma LIM-I:

$(!!r. 0 < r \implies \exists s > 0. \forall x. x \neq a \ \& \ norm\ (x - a) < s \dashv norm\ (f\ x - L) < r)$
 $\implies f \dashv a \dashv L$
 $\langle proof \rangle$

lemma LIM-D:

$[| f \dashv a \dashv L; 0 < r |]$
 $\implies \exists s > 0. \forall x. x \neq a \ \& \ norm\ (x - a) < s \dashv norm\ (f\ x - L) < r$
 $\langle proof \rangle$

lemma LIM-offset: $f \dashv a \dashv L \implies (\lambda x. f\ (x + k)) \dashv a - k \dashv L$
 $\langle proof \rangle$

lemma LIM-offset-zero: $f \dashv a \dashv L \implies (\lambda h. f\ (a + h)) \dashv 0 \dashv L$
 $\langle proof \rangle$

lemma LIM-offset-zero-cancel: $(\lambda h. f\ (a + h)) \dashv 0 \dashv L \implies f \dashv a \dashv L$
 $\langle proof \rangle$

lemma LIM-const [simp]: $(\%x. k) \dashv x \dashv k$
 $\langle proof \rangle$

lemma LIM-add:

fixes $f\ g :: 'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}$
assumes $f: f \dashrightarrow a \dashrightarrow L$ **and** $g: g \dashrightarrow a \dashrightarrow M$
shows $(\%x. f\ x + g(x)) \dashrightarrow a \dashrightarrow (L + M)$
 $\langle\text{proof}\rangle$

lemma *LIM-add-zero*:
 $\llbracket f \dashrightarrow a \dashrightarrow 0; g \dashrightarrow a \dashrightarrow 0 \rrbracket \Longrightarrow (\lambda x. f\ x + g\ x) \dashrightarrow a \dashrightarrow 0$
 $\langle\text{proof}\rangle$

lemma *minus-diff-minus*:
fixes $a\ b :: 'a::\text{ab-group-add}$
shows $(- a) - (- b) = - (a - b)$
 $\langle\text{proof}\rangle$

lemma *LIM-minus*: $f \dashrightarrow a \dashrightarrow L \Longrightarrow (\%x. -f(x)) \dashrightarrow a \dashrightarrow -L$
 $\langle\text{proof}\rangle$

lemma *LIM-add-minus*:
 $\llbracket f \dashrightarrow x \dashrightarrow l; g \dashrightarrow x \dashrightarrow m \rrbracket \Longrightarrow (\%x. f(x) + -g(x)) \dashrightarrow x \dashrightarrow (l + -m)$
 $\langle\text{proof}\rangle$

lemma *LIM-diff*:
 $\llbracket f \dashrightarrow x \dashrightarrow l; g \dashrightarrow x \dashrightarrow m \rrbracket \Longrightarrow (\%x. f(x) - g(x)) \dashrightarrow x \dashrightarrow l - m$
 $\langle\text{proof}\rangle$

lemma *LIM-zero*: $f \dashrightarrow a \dashrightarrow l \Longrightarrow (\lambda x. f\ x - l) \dashrightarrow a \dashrightarrow 0$
 $\langle\text{proof}\rangle$

lemma *LIM-zero-cancel*: $(\lambda x. f\ x - l) \dashrightarrow a \dashrightarrow 0 \Longrightarrow f \dashrightarrow a \dashrightarrow l$
 $\langle\text{proof}\rangle$

lemma *LIM-zero-iff*: $(\lambda x. f\ x - l) \dashrightarrow a \dashrightarrow 0 = f \dashrightarrow a \dashrightarrow l$
 $\langle\text{proof}\rangle$

lemma *LIM-imp-LIM*:
assumes $f: f \dashrightarrow a \dashrightarrow l$
assumes $le: \bigwedge x. x \neq a \Longrightarrow \text{norm } (g\ x - m) \leq \text{norm } (f\ x - l)$
shows $g \dashrightarrow a \dashrightarrow m$
 $\langle\text{proof}\rangle$

lemma *LIM-norm*: $f \dashrightarrow a \dashrightarrow l \Longrightarrow (\lambda x. \text{norm } (f\ x)) \dashrightarrow a \dashrightarrow \text{norm } l$
 $\langle\text{proof}\rangle$

lemma *LIM-norm-zero*: $f \dashrightarrow a \dashrightarrow 0 \Longrightarrow (\lambda x. \text{norm } (f\ x)) \dashrightarrow a \dashrightarrow 0$
 $\langle\text{proof}\rangle$

lemma *LIM-norm-zero-cancel*: $(\lambda x. \text{norm } (f\ x)) \dashrightarrow a \dashrightarrow 0 \Longrightarrow f \dashrightarrow a \dashrightarrow 0$
 $\langle\text{proof}\rangle$

$\langle \text{proof} \rangle$

lemma *LIM-norm-zero-iff*: $(\lambda x. \text{norm } (f\ x)) \dashrightarrow a \dashrightarrow 0 = f \dashrightarrow a \dashrightarrow 0$
 $\langle \text{proof} \rangle$

lemma *LIM-rabs*: $f \dashrightarrow a \dashrightarrow (l::\text{real}) \implies (\lambda x. |f\ x|) \dashrightarrow a \dashrightarrow |l|$
 $\langle \text{proof} \rangle$

lemma *LIM-rabs-zero*: $f \dashrightarrow a \dashrightarrow (0::\text{real}) \implies (\lambda x. |f\ x|) \dashrightarrow a \dashrightarrow 0$
 $\langle \text{proof} \rangle$

lemma *LIM-rabs-zero-cancel*: $(\lambda x. |f\ x|) \dashrightarrow a \dashrightarrow (0::\text{real}) \implies f \dashrightarrow a \dashrightarrow 0$
 $\langle \text{proof} \rangle$

lemma *LIM-rabs-zero-iff*: $(\lambda x. |f\ x|) \dashrightarrow a \dashrightarrow (0::\text{real}) = f \dashrightarrow a \dashrightarrow 0$
 $\langle \text{proof} \rangle$

lemma *LIM-const-not-eq*:
fixes $a :: 'a::\text{real-normed-algebra-1}$
shows $k \neq L \implies \neg (\lambda x. k) \dashrightarrow a \dashrightarrow L$
 $\langle \text{proof} \rangle$

lemmas *LIM-not-zero* = *LIM-const-not-eq* [where $L = 0$]

lemma *LIM-const-eq*:
fixes $a :: 'a::\text{real-normed-algebra-1}$
shows $(\lambda x. k) \dashrightarrow a \dashrightarrow L \implies k = L$
 $\langle \text{proof} \rangle$

lemma *LIM-unique*:
fixes $a :: 'a::\text{real-normed-algebra-1}$
shows $\llbracket f \dashrightarrow a \dashrightarrow L; f \dashrightarrow a \dashrightarrow M \rrbracket \implies L = M$
 $\langle \text{proof} \rangle$

lemma *LIM-ident* [simp]: $(\lambda x. x) \dashrightarrow a \dashrightarrow a$
 $\langle \text{proof} \rangle$

Limits are equal for functions equal except at limit point

lemma *LIM-equal*:
 $\llbracket \forall x. x \neq a \dashrightarrow (f\ x = g\ x) \rrbracket \implies (f \dashrightarrow a \dashrightarrow l) = (g \dashrightarrow a \dashrightarrow l)$
 $\langle \text{proof} \rangle$

lemma *LIM-cong*:
 $\llbracket a = b; \bigwedge x. x \neq b \implies f\ x = g\ x; l = m \rrbracket$
 $\implies ((\lambda x. f\ x) \dashrightarrow a \dashrightarrow l) = ((\lambda x. g\ x) \dashrightarrow b \dashrightarrow m)$
 $\langle \text{proof} \rangle$

lemma *LIM-equal2*:

assumes 1: $0 < R$
assumes 2: $\bigwedge x. \llbracket x \neq a; \text{norm } (x - a) < R \rrbracket \implies f\ x = g\ x$
shows $g \dashrightarrow a \dashrightarrow l \implies f \dashrightarrow a \dashrightarrow l$
 <proof>

Two uses in Hyperreal/Transcendental.ML

lemma *LIM-trans*:

$\llbracket (\%x. f(x) + -g(x)) \dashrightarrow a \dashrightarrow 0; g \dashrightarrow a \dashrightarrow l \rrbracket \implies f \dashrightarrow a \dashrightarrow l$
 <proof>

lemma *LIM-compose*:

assumes $g: g \dashrightarrow l \dashrightarrow g\ l$
assumes $f: f \dashrightarrow a \dashrightarrow l$
shows $(\lambda x. g\ (f\ x)) \dashrightarrow a \dashrightarrow g\ l$
 <proof>

lemma *LIM-compose2*:

assumes $f: f \dashrightarrow a \dashrightarrow b$
assumes $g: g \dashrightarrow b \dashrightarrow c$
assumes *inj*: $\exists d > 0. \forall x. x \neq a \wedge \text{norm } (x - a) < d \longrightarrow f\ x \neq b$
shows $(\lambda x. g\ (f\ x)) \dashrightarrow a \dashrightarrow c$
 <proof>

lemma *LIM-o*: $\llbracket g \dashrightarrow l \dashrightarrow g\ l; f \dashrightarrow a \dashrightarrow l \rrbracket \implies (g \circ f) \dashrightarrow a \dashrightarrow g\ l$
 <proof>

lemma *real-LIM-sandwich-zero*:

fixes $f\ g :: 'a::\text{real-normed-vector} \Rightarrow \text{real}$
assumes $f: f \dashrightarrow a \dashrightarrow 0$
assumes 1: $\bigwedge x. x \neq a \implies 0 \leq g\ x$
assumes 2: $\bigwedge x. x \neq a \implies g\ x \leq f\ x$
shows $g \dashrightarrow a \dashrightarrow 0$
 <proof>

Bounded Linear Operators

lemma (in *bounded-linear*) *cont*: $f \dashrightarrow a \dashrightarrow f\ a$
 <proof>

lemma (in *bounded-linear*) *LIM*:

$g \dashrightarrow a \dashrightarrow l \implies (\lambda x. f\ (g\ x)) \dashrightarrow a \dashrightarrow f\ l$
 <proof>

lemma (in *bounded-linear*) *LIM-zero*:

$g \dashrightarrow a \dashrightarrow 0 \implies (\lambda x. f\ (g\ x)) \dashrightarrow a \dashrightarrow 0$
 <proof>

Bounded Bilinear Operators

lemma (in *bounded-bilinear*) *LIM-prod-zero*:

assumes $f: f \dashrightarrow a \dashrightarrow 0$

assumes $g: g \dashv\dashv a \dashv\dashv 0$
shows $(\lambda x. f\ x \ **\ g\ x) \dashv\dashv a \dashv\dashv 0$
 $\langle proof \rangle$

lemma (**in** *bounded-bilinear*) *LIM-left-zero*:
 $f \dashv\dashv a \dashv\dashv 0 \implies (\lambda x. f\ x \ **\ c) \dashv\dashv a \dashv\dashv 0$
 $\langle proof \rangle$

lemma (**in** *bounded-bilinear*) *LIM-right-zero*:
 $f \dashv\dashv a \dashv\dashv 0 \implies (\lambda x. c \ **\ f\ x) \dashv\dashv a \dashv\dashv 0$
 $\langle proof \rangle$

lemma (**in** *bounded-bilinear*) *LIM*:
 $\llbracket f \dashv\dashv a \dashv\dashv L; g \dashv\dashv a \dashv\dashv M \rrbracket \implies (\lambda x. f\ x \ **\ g\ x) \dashv\dashv a \dashv\dashv L \ **\ M$
 $\langle proof \rangle$

lemmas *LIM-mult* = *mult.LIM*

lemmas *LIM-mult-zero* = *mult.LIM-prod-zero*

lemmas *LIM-mult-left-zero* = *mult.LIM-left-zero*

lemmas *LIM-mult-right-zero* = *mult.LIM-right-zero*

lemmas *LIM-scaleR* = *scaleR.LIM*

lemmas *LIM-of-real* = *of-real.LIM*

lemma *LIM-power*:
fixes $f :: 'a::real-normed-vector \Rightarrow 'b::\{recpower,real-normed-algebra\}$
assumes $f: f \dashv\dashv a \dashv\dashv l$
shows $(\lambda x. f\ x \ ^\wedge\ n) \dashv\dashv a \dashv\dashv l \ ^\wedge\ n$
 $\langle proof \rangle$

59.1.2 Derived theorems about *LIM*

lemma *LIM-inverse-lemma*:
fixes $x :: 'a::real-normed-div-algebra$
assumes $r: 0 < r$
assumes $x: norm\ (x - 1) < \min\ (1/2)\ (r/2)$
shows $norm\ (inverse\ x - 1) < r$
 $\langle proof \rangle$

lemma *LIM-inverse-fun*:
assumes $a: a \neq (0::'a::real-normed-div-algebra)$
shows $inverse \dashv\dashv a \dashv\dashv inverse\ a$
 $\langle proof \rangle$

lemma *LIM-inverse*:

fixes $L :: 'a::\text{real-normed-div-algebra}$
shows $\llbracket f \dashv\dashv a \dashv\dashv L; L \neq 0 \rrbracket \implies (\lambda x. \text{inverse } (f x)) \dashv\dashv a \dashv\dashv \text{inverse } L$
 $\langle \text{proof} \rangle$

lemma *LIM-sgn*:
 $\llbracket f \dashv\dashv a \dashv\dashv l; l \neq 0 \rrbracket \implies (\lambda x. \text{sgn } (f x)) \dashv\dashv a \dashv\dashv \text{sgn } l$
 $\langle \text{proof} \rangle$

59.2 Continuity

59.2.1 Purely standard proofs

lemma *LIM-isCont-iff*: $(f \dashv\dashv a \dashv\dashv f a) = ((\lambda h. f (a + h)) \dashv\dashv 0 \dashv\dashv f a)$
 $\langle \text{proof} \rangle$

lemma *isCont-iff*: $\text{isCont } f x = (\lambda h. f (x + h)) \dashv\dashv 0 \dashv\dashv f x$
 $\langle \text{proof} \rangle$

lemma *isCont-ident* [simp]: $\text{isCont } (\lambda x. x) a$
 $\langle \text{proof} \rangle$

lemma *isCont-const* [simp]: $\text{isCont } (\lambda x. k) a$
 $\langle \text{proof} \rangle$

lemma *isCont-norm*: $\text{isCont } f a \implies \text{isCont } (\lambda x. \text{norm } (f x)) a$
 $\langle \text{proof} \rangle$

lemma *isCont-rabs*: $\text{isCont } f a \implies \text{isCont } (\lambda x. |f x|) a$
 $\langle \text{proof} \rangle$

lemma *isCont-add*: $\llbracket \text{isCont } f a; \text{isCont } g a \rrbracket \implies \text{isCont } (\lambda x. f x + g x) a$
 $\langle \text{proof} \rangle$

lemma *isCont-minus*: $\text{isCont } f a \implies \text{isCont } (\lambda x. - f x) a$
 $\langle \text{proof} \rangle$

lemma *isCont-diff*: $\llbracket \text{isCont } f a; \text{isCont } g a \rrbracket \implies \text{isCont } (\lambda x. f x - g x) a$
 $\langle \text{proof} \rangle$

lemma *isCont-mult*:
fixes $f g :: 'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-algebra}$
shows $\llbracket \text{isCont } f a; \text{isCont } g a \rrbracket \implies \text{isCont } (\lambda x. f x * g x) a$
 $\langle \text{proof} \rangle$

lemma *isCont-inverse*:
fixes $f :: 'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-div-algebra}$
shows $\llbracket \text{isCont } f a; f a \neq 0 \rrbracket \implies \text{isCont } (\lambda x. \text{inverse } (f x)) a$
 $\langle \text{proof} \rangle$

lemma *isCont-LIM-compose*:

$\llbracket \text{isCont } g \text{ } l; f \text{ } \text{---} \text{ } a \text{ } \text{---} > l \rrbracket \Longrightarrow (\lambda x. g \text{ } (f \text{ } x)) \text{ } \text{---} \text{ } a \text{ } \text{---} > g \text{ } l$
 $\langle \text{proof} \rangle$

lemma *isCont-LIM-compose2*:

assumes $f \text{ } [\text{unfolded } \text{isCont-def}]: \text{isCont } f \text{ } a$

assumes $g: g \text{ } \text{---} \text{ } f \text{ } a \text{ } \text{---} > l$

assumes *inj*: $\exists d > 0. \forall x. x \neq a \wedge \text{norm } (x - a) < d \longrightarrow f \text{ } x \neq f \text{ } a$

shows $(\lambda x. g \text{ } (f \text{ } x)) \text{ } \text{---} \text{ } a \text{ } \text{---} > l$

$\langle \text{proof} \rangle$

lemma *isCont-o2*: $\llbracket \text{isCont } f \text{ } a; \text{isCont } g \text{ } (f \text{ } a) \rrbracket \Longrightarrow \text{isCont } (\lambda x. g \text{ } (f \text{ } x)) \text{ } a$

$\langle \text{proof} \rangle$

lemma *isCont-o*: $\llbracket \text{isCont } f \text{ } a; \text{isCont } g \text{ } (f \text{ } a) \rrbracket \Longrightarrow \text{isCont } (g \text{ } o \text{ } f) \text{ } a$

$\langle \text{proof} \rangle$

lemma (*in bounded-linear*) *isCont*: $\text{isCont } f \text{ } a$

$\langle \text{proof} \rangle$

lemma (*in bounded-bilinear*) *isCont*:

$\llbracket \text{isCont } f \text{ } a; \text{isCont } g \text{ } a \rrbracket \Longrightarrow \text{isCont } (\lambda x. f \text{ } x \text{ } ** \text{ } g \text{ } x) \text{ } a$

$\langle \text{proof} \rangle$

lemmas *isCont-scaleR* = *scaleR.isCont*

lemma *isCont-of-real*:

$\text{isCont } f \text{ } a \Longrightarrow \text{isCont } (\lambda x. \text{of-real } (f \text{ } x)) \text{ } a$

$\langle \text{proof} \rangle$

lemma *isCont-power*:

fixes $f :: 'a :: \text{real-normed-vector} \Rightarrow 'b :: \{\text{recpower, real-normed-algebra}\}$

shows $\text{isCont } f \text{ } a \Longrightarrow \text{isCont } (\lambda x. f \text{ } x \text{ } ^ \text{ } n) \text{ } a$

$\langle \text{proof} \rangle$

lemma *isCont-sgn*:

$\llbracket \text{isCont } f \text{ } a; f \text{ } a \neq 0 \rrbracket \Longrightarrow \text{isCont } (\lambda x. \text{sgn } (f \text{ } x)) \text{ } a$

$\langle \text{proof} \rangle$

lemma *isCont-abs* [*simp*]: $\text{isCont } \text{abs } (a :: \text{real})$

$\langle \text{proof} \rangle$

lemma *isCont-setsum*: **fixes** $A :: \text{nat set}$ **assumes** *finite A*

shows $\forall i \in A. \text{isCont } (f \text{ } i) \text{ } x \Longrightarrow \text{isCont } (\lambda x. \sum i \in A. f \text{ } i \text{ } x) \text{ } x$

$\langle \text{proof} \rangle$

lemma *LIM-less-bound*: **fixes** $f :: \text{real} \Rightarrow \text{real}$ **assumes** $b < x$

and *all-le*: $\forall x' \in \{ b <..< x \}. 0 \leq f \text{ } x'$ **and** *isCont*: $\text{isCont } f \text{ } x$

shows $0 \leq f \text{ } x$

$\langle \text{proof} \rangle$

59.3 Uniform Continuity

lemma *isUCont-isCont*: $\text{isUCont } f \implies \text{isCont } f$
 $\langle \text{proof} \rangle$

lemma *isUCont-Cauchy*:
 $\llbracket \text{isUCont } f; \text{Cauchy } X \rrbracket \implies \text{Cauchy } (\lambda n. f (X\ n))$
 $\langle \text{proof} \rangle$

lemma (in *bounded-linear*) *isUCont*: $\text{isUCont } f$
 $\langle \text{proof} \rangle$

lemma (in *bounded-linear*) *Cauchy*: $\text{Cauchy } X \implies \text{Cauchy } (\lambda n. f (X\ n))$
 $\langle \text{proof} \rangle$

59.4 Relation of LIM and LIMSEQ

lemma *LIMSEQ-SEQ-conv1*:
fixes $a :: 'a::\text{real-normed-vector}$
assumes $X: X \dashrightarrow a \dashrightarrow L$
shows $\forall S. (\forall n. S\ n \neq a) \wedge S \dashrightarrow a \longrightarrow (\lambda n. X\ (S\ n)) \dashrightarrow L$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-SEQ-conv2*:
fixes $a :: \text{real}$
assumes $\forall S. (\forall n. S\ n \neq a) \wedge S \dashrightarrow a \longrightarrow (\lambda n. X\ (S\ n)) \dashrightarrow L$
shows $X \dashrightarrow a \dashrightarrow L$
 $\langle \text{proof} \rangle$

lemma *LIMSEQ-SEQ-conv*:
 $(\forall S. (\forall n. S\ n \neq a) \wedge S \dashrightarrow a \longrightarrow (a::\text{real}) \longrightarrow (\lambda n. X\ (S\ n)) \dashrightarrow L) =$
 $(X \dashrightarrow a \dashrightarrow L)$
 $\langle \text{proof} \rangle$

end

60 Deriv: Differentiation

theory *Deriv*
imports *Lim*
begin

Standard Definitions

definition
 $\text{deriv} :: ['a::\text{real-normed-field} \Rightarrow 'a, 'a, 'a] \Rightarrow \text{bool}$
 — Differentiation: D is derivative of function f at x
 $((\text{DERIV } (-) / (-) / :> (-)) [1000, 1000, 60] 60)$ **where**
 $\text{DERIV } f\ x :> D = ((\%h. (f(x + h) - f\ x) / h) \dashrightarrow 0 \dashrightarrow D)$

consts

Bolzano-bisect :: [real*real=>bool, real, real, nat] => (real*real)

primrec

Bolzano-bisect *P a b 0* = (*a,b*)

Bolzano-bisect *P a b (Suc n)* =

(let (*x,y*) = *Bolzano-bisect P a b n*
in if *P(x, (x+y)/2)* then ((*x+y*)/2, *y*)
else (*x, (x+y)/2*))

60.1 Derivatives

lemma *DERIV-iff*: (*DERIV f x* :> *D*) = ((%*h*. (*f(x + h) - f(x)*)/*h*) -- 0 --> *D*)

⟨*proof*⟩

lemma *DERIV-D*: *DERIV f x* :> *D* ==> (%*h*. (*f(x + h) - f(x)*)/*h*) -- 0 --> *D*

⟨*proof*⟩

lemma *DERIV-const* [*simp*]: *DERIV* ($\lambda x. k$) *x* :> 0

⟨*proof*⟩

lemma *DERIV-ident* [*simp*]: *DERIV* ($\lambda x. x$) *x* :> 1

⟨*proof*⟩

lemma *add-diff-add*:

fixes *a b c d* :: 'a::ab-group-add

shows (*a + c*) - (*b + d*) = (*a - b*) + (*c - d*)

⟨*proof*⟩

lemma *DERIV-add*:

[[*DERIV f x* :> *D*; *DERIV g x* :> *E*]] ==> *DERIV* ($\lambda x. f x + g x$) *x* :> *D + E*

⟨*proof*⟩

lemma *DERIV-minus*:

DERIV f x :> *D* ==> *DERIV* ($\lambda x. - f x$) *x* :> - *D*

⟨*proof*⟩

lemma *DERIV-diff*:

[[*DERIV f x* :> *D*; *DERIV g x* :> *E*]] ==> *DERIV* ($\lambda x. f x - g x$) *x* :> *D - E*

⟨*proof*⟩

lemma *DERIV-add-minus*:

[[*DERIV f x* :> *D*; *DERIV g x* :> *E*]] ==> *DERIV* ($\lambda x. f x + - g x$) *x* :> *D + - E*

⟨*proof*⟩

lemma *DERIV-isCont*: *DERIV f x* :> *D* ==> *isCont f x*

$\langle proof \rangle$

lemma *DERIV-mult-lemma*:

fixes $a\ b\ c\ d :: 'a::real-field$

shows $(a * b - c * d) / h = a * ((b - d) / h) + ((a - c) / h) * d$

$\langle proof \rangle$

lemma *DERIV-mult'*:

assumes $f: DERIV\ f\ x :> D$

assumes $g: DERIV\ g\ x :> E$

shows $DERIV\ (\lambda x. f\ x * g\ x)\ x :> f\ x * E + D * g\ x$

$\langle proof \rangle$

lemma *DERIV-mult*:

$[| DERIV\ f\ x :> Da; DERIV\ g\ x :> Db |]$

$==> DERIV\ (\%x. f\ x * g\ x)\ x :> (Da * g(x)) + (Db * f(x))$

$\langle proof \rangle$

lemma *DERIV-unique*:

$[| DERIV\ f\ x :> D; DERIV\ f\ x :> E |] ==> D = E$

$\langle proof \rangle$

Differentiation of finite sum

lemma *DERIV-sumr* [*rule-format* (*no-asm*)]:

$(\forall r. m \leq r \ \& \ r < (m + n) \longrightarrow DERIV\ (\%x. f\ r\ x)\ x :> (f'\ r\ x))$

$\longrightarrow DERIV\ (\%x. \sum_{n=m..<n::nat. f\ n\ x :: real})\ x :> (\sum_{r=m..<n. f'\ r\ x})$

$\langle proof \rangle$

Alternative definition for differentiability

lemma *DERIV-LIM-iff*:

$((\%h. (f(a + h) - f(a)) / h) \longrightarrow 0 \longrightarrow D) =$

$((\%x. (f(x) - f(a)) / (x - a)) \longrightarrow a \longrightarrow D)$

$\langle proof \rangle$

lemma *DERIV-iff2*: $(DERIV\ f\ x :> D) = ((\%z. (f(z) - f(x)) / (z - x)) \longrightarrow x \longrightarrow D)$

$\langle proof \rangle$

lemma *inverse-diff-inverse*:

$[| (a::'a::division-ring) \neq 0; b \neq 0 |]$

$\implies inverse\ a - inverse\ b = - (inverse\ a * (a - b) * inverse\ b)$

$\langle proof \rangle$

lemma *DERIV-inverse-lemma*:

$[| a \neq 0; b \neq (0::'a::real-normed-field) |]$

$\implies (inverse\ a - inverse\ b) / h$

$= - (inverse\ a * ((a - b) / h) * inverse\ b)$

$\langle proof \rangle$

lemma *DERIV-inverse'*:

assumes *der*: *DERIV* *f* *x* :> *D*

assumes *neg*: *f* *x* ≠ 0

shows *DERIV* ($\lambda x. \text{inverse } (f\ x)$) *x* :> $-(\text{inverse } (f\ x) * D * \text{inverse } (f\ x))$
(is *DERIV* - - :> ?*E*)

⟨*proof*⟩

lemma *DERIV-divide*:

$\llbracket \text{DERIV } f\ x :> D; \text{DERIV } g\ x :> E; g\ x \neq 0 \rrbracket$

$\implies \text{DERIV } (\lambda x. f\ x / g\ x)\ x :> (D * g\ x - f\ x * E) / (g\ x * g\ x)$

⟨*proof*⟩

lemma *DERIV-power-Suc*:

fixes *f* :: 'a ⇒ 'a::{*real-normed-field*,*recpower*}

assumes *f*: *DERIV* *f* *x* :> *D*

shows *DERIV* ($\lambda x. f\ x ^ \text{Suc } n$) *x* :> $(1 + \text{of-nat } n) * (D * f\ x ^ n)$

⟨*proof*⟩

lemma *DERIV-power*:

fixes *f* :: 'a ⇒ 'a::{*real-normed-field*,*recpower*}

assumes *f*: *DERIV* *f* *x* :> *D*

shows *DERIV* ($\lambda x. f\ x ^ n$) *x* :> $\text{of-nat } n * (D * f\ x ^ (n - \text{Suc } 0))$

⟨*proof*⟩

Caratheodory formulation of derivative at a point

lemma *CARAT-DERIV*:

$(\text{DERIV } f\ x :> l) =$

$(\exists g. (\forall z. f\ z - f\ x = g\ z * (z - x)) \ \& \ \text{isCont } g\ x \ \& \ g\ x = l)$

(is ?*lhs* = ?*rhs*)

⟨*proof*⟩

lemma *DERIV-chain'*:

assumes *f*: *DERIV* *f* *x* :> *D*

assumes *g*: *DERIV* *g* (*f* *x*) :> *E*

shows *DERIV* ($\lambda x. g\ (f\ x)$) *x* :> *E* * *D*

⟨*proof*⟩

lemma *DERIV-cmult*:

$\text{DERIV } f\ x :> D \implies \text{DERIV } (\%x. c * f\ x)\ x :> c * D$

⟨*proof*⟩

lemma *DERIV-chain*: $\llbracket \text{DERIV } f\ (g\ x) :> Da; \text{DERIV } g\ x :> Db \rrbracket \implies \text{DERIV } (f \circ g)\ x :> Da * Db$

⟨*proof*⟩

lemma *DERIV-chain2*: $[[\text{DERIV } f \ (g \ x) :> Da; \text{DERIV } g \ x :> Db]]$ \implies
 $\text{DERIV } (\%x. f \ (g \ x)) \ x :> Da * Db$
 $\langle \text{proof} \rangle$

lemma *DERIV-cmult-Id* [*simp*]: $\text{DERIV } (op * c) \ x :> c$
 $\langle \text{proof} \rangle$

lemma *DERIV-pow*: $\text{DERIV } (\%x. x \wedge n) \ x :> \text{real } n * (x \wedge (n - \text{Suc } 0))$
 $\langle \text{proof} \rangle$

Power of -1

lemma *DERIV-inverse*:
fixes $x :: 'a :: \{\text{real-normed-field}, \text{recpower}\}$
shows $x \neq 0 \implies \text{DERIV } (\%x. \text{inverse}(x)) \ x :> -(\text{inverse } x \wedge \text{Suc } (\text{Suc } 0))$
 $\langle \text{proof} \rangle$

Derivative of inverse

lemma *DERIV-inverse-fun*:
fixes $x :: 'a :: \{\text{real-normed-field}, \text{recpower}\}$
shows $[[\text{DERIV } f \ x :> d; f(x) \neq 0]]$
 $\implies \text{DERIV } (\%x. \text{inverse}(f \ x)) \ x :> -(d * \text{inverse}(f(x) \wedge \text{Suc } (\text{Suc } 0)))$
 $\langle \text{proof} \rangle$

Derivative of quotient

lemma *DERIV-quotient*:
fixes $x :: 'a :: \{\text{real-normed-field}, \text{recpower}\}$
shows $[[\text{DERIV } f \ x :> d; \text{DERIV } g \ x :> e; g(x) \neq 0]]$
 $\implies \text{DERIV } (\%y. f(y) / (g \ y)) \ x :> (d * g(x) - (e * f(x))) / (g(x) \wedge \text{Suc } (\text{Suc } 0))$
 $\langle \text{proof} \rangle$

lemma *lemma-DERIV-subst*: $[[\text{DERIV } f \ x :> D; D = E]]$ $\implies \text{DERIV } f \ x :> E$
 $\langle \text{proof} \rangle$

60.2 Differentiability predicate

definition

$\text{differentiable} :: ['a :: \text{real-normed-field} \Rightarrow 'a, 'a] \Rightarrow \text{bool}$
 $(\text{infixl } \text{differentiable } 60) \text{ where}$
 $f \text{ differentiable } x = (\exists D. \text{DERIV } f \ x :> D)$

lemma *differentiableE* [*elim?*]:
assumes $f \text{ differentiable } x$
obtains df **where** $\text{DERIV } f \ x :> df$
 $\langle \text{proof} \rangle$

lemma *differentiableD*: f *differentiable* $x \implies \exists D. \text{DERIV } f \ x :> D$
 ⟨*proof*⟩

lemma *differentiableI*: $\text{DERIV } f \ x :> D \implies f$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-ident* [*simp*]: $(\lambda x. x)$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-const* [*simp*]: $(\lambda z. a)$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-compose*:
 assumes $f: f$ *differentiable* $(g \ x)$
 assumes $g: g$ *differentiable* x
 shows $(\lambda x. f \ (g \ x))$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-sum* [*simp*]:
 assumes f *differentiable* x
 and g *differentiable* x
 shows $(\lambda x. f \ x + g \ x)$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-minus* [*simp*]:
 assumes f *differentiable* x
 shows $(\lambda x. - f \ x)$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-diff* [*simp*]:
 assumes f *differentiable* x
 assumes g *differentiable* x
 shows $(\lambda x. f \ x - g \ x)$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-mult* [*simp*]:
 assumes f *differentiable* x
 assumes g *differentiable* x
 shows $(\lambda x. f \ x * g \ x)$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-inverse* [*simp*]:
 assumes f *differentiable* x and $f \ x \neq 0$
 shows $(\lambda x. \text{inverse } (f \ x))$ *differentiable* x
 ⟨*proof*⟩

lemma *differentiable-divide* [*simp*]:
 assumes f *differentiable* x
 assumes g *differentiable* x and $g \ x \neq 0$

shows $(\lambda x. f\ x / g\ x)$ differentiable x
 $\langle proof \rangle$

lemma *differentiable-power* [simp]:
fixes $f :: 'a :: \{recpower, real-normed-field\} \Rightarrow 'a$
assumes f differentiable x
shows $(\lambda x. f\ x \wedge n)$ differentiable x
 $\langle proof \rangle$

60.3 Nested Intervals and Bisection

Lemmas about nested intervals and proof by bisection (cf. Harrison). All considerably tidied by lcp.

lemma *lemma-f-mono-add* [rule-format (no-asm)]: $(\forall n. (f :: nat \Rightarrow real) \ n \leq f\ (Suc\ n)) \longrightarrow f\ m \leq f\ (m + n)$
 $\langle proof \rangle$

lemma *f-inc-g-dec-Beq-f*: $[\![\forall n. f(n) \leq f(Suc\ n);$
 $\forall n. g(Suc\ n) \leq g(n);$
 $\forall n. f(n) \leq g(n) \!]\!] \Longrightarrow Bseq\ (f :: nat \Rightarrow real)$
 $\langle proof \rangle$

lemma *f-inc-g-dec-Beq-g*: $[\![\forall n. f(n) \leq f(Suc\ n);$
 $\forall n. g(Suc\ n) \leq g(n);$
 $\forall n. f(n) \leq g(n) \!]\!] \Longrightarrow Bseq\ (g :: nat \Rightarrow real)$
 $\langle proof \rangle$

lemma *f-inc-imp-le-lim*:
fixes $f :: nat \Rightarrow real$
shows $[\![\forall n. f\ n \leq f\ (Suc\ n); \text{convergent } f]\!] \Longrightarrow f\ n \leq \lim\ f$
 $\langle proof \rangle$

lemma *lim-uminus*: $\text{convergent } g \Longrightarrow \lim\ (\%x. -\ g\ x) = -\ (\lim\ g)$
 $\langle proof \rangle$

lemma *g-dec-imp-lim-le*:
fixes $g :: nat \Rightarrow real$
shows $[\![\forall n. g\ (Suc\ n) \leq g(n); \text{convergent } g]\!] \Longrightarrow \lim\ g \leq g\ n$
 $\langle proof \rangle$

lemma *lemma-nest*: $[\![\forall n. f(n) \leq f(Suc\ n);$
 $\forall n. g(Suc\ n) \leq g(n);$
 $\forall n. f(n) \leq g(n) \!]\!] \Longrightarrow \exists l\ m :: real. l \leq m \ \& \ ((\forall n. f(n) \leq l) \ \& \ f \dashrightarrow l) \ \& \ ((\forall n. m \leq g(n)) \ \& \ g \dashrightarrow m)$
 $\langle proof \rangle$

lemma *lemma-nest-unique*: $[[\forall n. f(n) \leq f(\text{Suc } n);$
 $\forall n. g(\text{Suc } n) \leq g(n);$
 $\forall n. f(n) \leq g(n);$
 $(\%n. f(n) - g(n)) \text{ ----} > 0 \]]$
 $\implies \exists l::\text{real}. ((\forall n. f(n) \leq l) \ \& \ f \text{ ----} > l) \ \&$
 $(\forall n. l \leq g(n)) \ \& \ g \text{ ----} > l)$
 $\langle \text{proof} \rangle$

The universal quantifiers below are required for the declaration of *Bolzano-nest-unique* below.

lemma *Bolzano-bisect-le*:
 $a \leq b \implies \forall n. \text{fst}(\text{Bolzano-bisect } P \ a \ b \ n) \leq \text{snd}(\text{Bolzano-bisect } P \ a \ b \ n)$
 $\langle \text{proof} \rangle$

lemma *Bolzano-bisect-fst-le-Suc*: $a \leq b \implies$
 $\forall n. \text{fst}(\text{Bolzano-bisect } P \ a \ b \ n) \leq \text{fst}(\text{Bolzano-bisect } P \ a \ b \ (\text{Suc } n))$
 $\langle \text{proof} \rangle$

lemma *Bolzano-bisect-Suc-le-snd*: $a \leq b \implies$
 $\forall n. \text{snd}(\text{Bolzano-bisect } P \ a \ b \ (\text{Suc } n)) \leq \text{snd}(\text{Bolzano-bisect } P \ a \ b \ n)$
 $\langle \text{proof} \rangle$

lemma *eq-divide-2-times-iff*: $((x::\text{real}) = y / (2 * z)) = (2 * x = y/z)$
 $\langle \text{proof} \rangle$

lemma *Bolzano-bisect-diff*:
 $a \leq b \implies$
 $\text{snd}(\text{Bolzano-bisect } P \ a \ b \ n) - \text{fst}(\text{Bolzano-bisect } P \ a \ b \ n) =$
 $(b-a) / (2 ^ n)$
 $\langle \text{proof} \rangle$

lemmas *Bolzano-nest-unique* =
lemma-nest-unique
 $[OF \ \text{Bolzano-bisect-fst-le-Suc} \ \text{Bolzano-bisect-Suc-le-snd} \ \text{Bolzano-bisect-le}]$

lemma *not-P-Bolzano-bisect*:
assumes $P: \quad !!a \ b \ c. \ [\ P(a,b); P(b,c); a \leq b; b \leq c] \implies P(a,c)$
and $\text{not}P: \sim P(a,b)$
and $\text{le}: \quad a \leq b$
shows $\sim P(\text{fst}(\text{Bolzano-bisect } P \ a \ b \ n), \text{snd}(\text{Bolzano-bisect } P \ a \ b \ n))$
 $\langle \text{proof} \rangle$

lemma *not-P-Bolzano-bisect'*:
 $[\ \forall a \ b \ c. \ P(a,b) \ \& \ P(b,c) \ \& \ a \leq b \ \& \ b \leq c \text{ --} > P(a,c);$
 $\sim P(a,b); \ a \leq b \] \implies$
 $\forall n. \sim P(\text{fst}(\text{Bolzano-bisect } P \ a \ b \ n), \text{snd}(\text{Bolzano-bisect } P \ a \ b \ n))$
 $\langle \text{proof} \rangle$

lemma *lemma-BOLZANO*:

$$\begin{aligned} & [\forall a \ b \ c. P(a,b) \ \& \ P(b,c) \ \& \ a \leq b \ \& \ b \leq c \ \longrightarrow P(a,c); \\ & \quad \forall x. \exists d::real. 0 < d \ \& \\ & \quad \quad (\forall a \ b. a \leq x \ \& \ x \leq b \ \& \ (b-a) < d \ \longrightarrow P(a,b)); \\ & \quad a \leq b \] \\ & \implies P(a,b) \\ & \langle proof \rangle \end{aligned}$$

lemma *lemma-BOLZANO2*: $((\forall a \ b \ c. (a \leq b \ \& \ b \leq c \ \& \ P(a,b) \ \& \ P(b,c)) \longrightarrow P(a,c)) \ \&$

$$\begin{aligned} & (\forall x. \exists d::real. 0 < d \ \& \\ & \quad (\forall a \ b. a \leq x \ \& \ x \leq b \ \& \ (b-a) < d \ \longrightarrow P(a,b)))) \\ & \longrightarrow (\forall a \ b. a \leq b \ \longrightarrow P(a,b)) \\ & \langle proof \rangle \end{aligned}$$

60.4 Intermediate Value Theorem

Prove Contrapositive by Bisection

lemma *IVT*: $[\ f(a::real) \leq (y::real); y \leq f(b);$
 $a \leq b;$
 $(\forall x. a \leq x \ \& \ x \leq b \ \longrightarrow isCont \ f \ x) \]$
 $\implies \exists x. a \leq x \ \& \ x \leq b \ \& \ f(x) = y$
 $\langle proof \rangle$

lemma *IVT2*: $[\ f(b::real) \leq (y::real); y \leq f(a);$
 $a \leq b;$
 $(\forall x. a \leq x \ \& \ x \leq b \ \longrightarrow isCont \ f \ x)$
 $] \implies \exists x. a \leq x \ \& \ x \leq b \ \& \ f(x) = y$
 $\langle proof \rangle$

lemma *IVT-objl*: $(f(a::real) \leq (y::real) \ \& \ y \leq f(b) \ \& \ a \leq b \ \&$
 $(\forall x. a \leq x \ \& \ x \leq b \ \longrightarrow isCont \ f \ x))$
 $\longrightarrow (\exists x. a \leq x \ \& \ x \leq b \ \& \ f(x) = y)$
 $\langle proof \rangle$

lemma *IVT2-objl*: $(f(b::real) \leq (y::real) \ \& \ y \leq f(a) \ \& \ a \leq b \ \&$
 $(\forall x. a \leq x \ \& \ x \leq b \ \longrightarrow isCont \ f \ x))$
 $\longrightarrow (\exists x. a \leq x \ \& \ x \leq b \ \& \ f(x) = y)$
 $\langle proof \rangle$

60.5 Boundedness of continuous functions

By bisection, function continuous on closed interval is bounded above

lemma *isCont-bounded*:

$$\begin{aligned} & \llbracket a \leq b; \forall x. a \leq x \ \& \ x \leq b \dashrightarrow \text{isCont } f \ x \rrbracket \\ & \implies \exists M :: \text{real}. \forall x :: \text{real}. a \leq x \ \& \ x \leq b \dashrightarrow f(x) \leq M \end{aligned}$$

 $\langle \text{proof} \rangle$

Refine the above to existence of least upper bound

lemma *lemma-reals-complete*: $((\exists x. x \in S) \ \& \ (\exists y. \text{isUb } \text{UNIV } S \ (y :: \text{real}))) \dashrightarrow$
 $(\exists t. \text{isLub } \text{UNIV } S \ t)$
 $\langle \text{proof} \rangle$

lemma *isCont-has-Ub*: $\llbracket a \leq b; \forall x. a \leq x \ \& \ x \leq b \dashrightarrow \text{isCont } f \ x \rrbracket$
 $\implies \exists M :: \text{real}. (\forall x :: \text{real}. a \leq x \ \& \ x \leq b \dashrightarrow f(x) \leq M) \ \&$
 $(\forall N. N < M \dashrightarrow (\exists x. a \leq x \ \& \ x \leq b \ \& \ N < f(x)))$
 $\langle \text{proof} \rangle$

Now show that it attains its upper bound

lemma *isCont-eq-Ub*:
assumes *le*: $a \leq b$
and con: $\forall x :: \text{real}. a \leq x \ \& \ x \leq b \dashrightarrow \text{isCont } f \ x$
shows $\exists M :: \text{real}. (\forall x. a \leq x \ \& \ x \leq b \dashrightarrow f(x) \leq M) \ \&$
 $(\exists x. a \leq x \ \& \ x \leq b \ \& \ f(x) = M)$
 $\langle \text{proof} \rangle$

Same theorem for lower bound

lemma *isCont-eq-Lb*: $\llbracket a \leq b; \forall x. a \leq x \ \& \ x \leq b \dashrightarrow \text{isCont } f \ x \rrbracket$
 $\implies \exists M :: \text{real}. (\forall x :: \text{real}. a \leq x \ \& \ x \leq b \dashrightarrow M \leq f(x)) \ \&$
 $(\exists x. a \leq x \ \& \ x \leq b \ \& \ f(x) = M)$
 $\langle \text{proof} \rangle$

Another version.

lemma *isCont-Lb-Ub*: $\llbracket a \leq b; \forall x. a \leq x \ \& \ x \leq b \dashrightarrow \text{isCont } f \ x \rrbracket$
 $\implies \exists L \ M :: \text{real}. (\forall x :: \text{real}. a \leq x \ \& \ x \leq b \dashrightarrow L \leq f(x) \ \& \ f(x) \leq M) \ \&$
 $(\forall y. L \leq y \ \& \ y \leq M \dashrightarrow (\exists x. a \leq x \ \& \ x \leq b \ \& \ (f(x) = y)))$
 $\langle \text{proof} \rangle$

60.6 Local extrema

If $(0 :: 'a) < f' \ x$ then x is Locally Strictly Increasing At The Right

lemma *DERIV-left-inc*:

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes *der*: $\text{DERIV } f \ x \ :> l$
and *l*: $0 < l$
shows $\exists d > 0. \forall h > 0. h < d \dashrightarrow f(x) < f(x + h)$
 $\langle \text{proof} \rangle$

lemma *DERIV-left-dec*:

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes *der*: $\text{DERIV } f \ x \ :> l$

and $l: l < 0$
shows $\exists d > 0. \forall h > 0. h < d \longrightarrow f(x) < f(x-h)$
 $\langle \text{proof} \rangle$

lemma *DERIV-local-max*:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $\text{der}: \text{DERIV } f \, x :> l$
and $d: 0 < d$
and $le: \forall y. |x-y| < d \longrightarrow f(y) \leq f(x)$
shows $l = 0$
 $\langle \text{proof} \rangle$

Similar theorem for a local minimum

lemma *DERIV-local-min*:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $[[\text{DERIV } f \, x :> l; 0 < d; \forall y. |x-y| < d \longrightarrow f(x) \leq f(y)]] \Longrightarrow l = 0$
 $\langle \text{proof} \rangle$

In particular, if a function is locally flat

lemma *DERIV-local-const*:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $[[\text{DERIV } f \, x :> l; 0 < d; \forall y. |x-y| < d \longrightarrow f(x) = f(y)]] \Longrightarrow l = 0$
 $\langle \text{proof} \rangle$

60.7 Rolle’s Theorem

Lemma about introducing open ball in open interval

lemma *lemma-interval-lt*:
 $[[a < x; x < b]]$
 $\Longrightarrow \exists d :: \text{real}. 0 < d \ \& \ (\forall y. |x-y| < d \longrightarrow a < y \ \& \ y < b)$
 $\langle \text{proof} \rangle$

lemma *lemma-interval*: $[[a < x; x < b]] \Longrightarrow$
 $\exists d :: \text{real}. 0 < d \ \& \ (\forall y. |x-y| < d \longrightarrow a \leq y \ \& \ y \leq b)$
 $\langle \text{proof} \rangle$

Rolle’s Theorem. If f is defined and continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b) , and $f \, a = f \, b$, then there exists $x_0 \in (a, b)$ such that $f' \, x_0 = (0 :: 'a)$

theorem *Rolle*:
assumes $lt: a < b$
and $eq: f(a) = f(b)$
and $con: \forall x. a \leq x \ \& \ x \leq b \longrightarrow \text{isCont } f \, x$
and $dif \text{ [rule-format]: } \forall x. a < x \ \& \ x < b \longrightarrow f \text{ differentiable } x$
shows $\exists z :: \text{real}. a < z \ \& \ z < b \ \& \ \text{DERIV } f \, z :> 0$
 $\langle \text{proof} \rangle$

60.8 Mean Value Theorem

lemma *lemma-MVT*:

$f\ a - (f\ b - f\ a)/(b-a) * a = f\ b - (f\ b - f\ a)/(b-a) * (b::real)$
 $\langle proof \rangle$

theorem *MVT*:

assumes *lt*: $a < b$
and *con*: $\forall x. a \leq x \ \& \ x \leq b \longrightarrow isCont\ f\ x$
and *dif* [rule-format]: $\forall x. a < x \ \& \ x < b \longrightarrow f\ differentiable\ x$
shows $\exists l\ z::real. a < z \ \& \ z < b \ \& \ DERIV\ f\ z :> l \ \& \ (f(b) - f(a) = (b-a) * l)$
 $\langle proof \rangle$

lemma *MVT2*:

$[| a < b; \forall x. a \leq x \ \& \ x \leq b \longrightarrow DERIV\ f\ x :> f'(x) |]$
 $\implies \exists z::real. a < z \ \& \ z < b \ \& \ (f\ b - f\ a = (b - a) * f'(z))$
 $\langle proof \rangle$

A function is constant if its derivative is 0 over an interval.

lemma *DERIV-isconst-end*:

fixes $f :: real \Rightarrow real$
shows $[| a < b;$
 $\forall x. a \leq x \ \& \ x \leq b \longrightarrow isCont\ f\ x;$
 $\forall x. a < x \ \& \ x < b \longrightarrow DERIV\ f\ x :> 0 \ |]$
 $\implies f\ b = f\ a$
 $\langle proof \rangle$

lemma *DERIV-isconst1*:

fixes $f :: real \Rightarrow real$
shows $[| a < b;$
 $\forall x. a \leq x \ \& \ x \leq b \longrightarrow isCont\ f\ x;$
 $\forall x. a < x \ \& \ x < b \longrightarrow DERIV\ f\ x :> 0 \ |]$
 $\implies \forall x. a \leq x \ \& \ x \leq b \longrightarrow f\ x = f\ a$
 $\langle proof \rangle$

lemma *DERIV-isconst2*:

fixes $f :: real \Rightarrow real$
shows $[| a < b;$
 $\forall x. a \leq x \ \& \ x \leq b \longrightarrow isCont\ f\ x;$
 $\forall x. a < x \ \& \ x < b \longrightarrow DERIV\ f\ x :> 0;$
 $a \leq x; x \leq b \ |]$
 $\implies f\ x = f\ a$
 $\langle proof \rangle$

lemma *DERIV-isconst3*: **fixes** $a\ b\ x\ y :: real$

assumes $a < b$ **and** $x \in \{a <..< b\}$ **and** $y \in \{a <..< b\}$
assumes *derivable*: $\bigwedge x. x \in \{a <..< b\} \implies DERIV\ f\ x :> 0$
shows $f\ x = f\ y$
 $\langle proof \rangle$

lemma *DERIV-isconst-all*:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $\forall x. \text{DERIV } f \ x :> 0 \implies f(x) = f(y)$
 $\langle \text{proof} \rangle$

lemma *DERIV-const-ratio-const*:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $[|a \neq b; \forall x. \text{DERIV } f \ x :> k|] \implies (f(b) - f(a)) = (b-a) * k$
 $\langle \text{proof} \rangle$

lemma *DERIV-const-ratio-const2*:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $[|a \neq b; \forall x. \text{DERIV } f \ x :> k|] \implies (f(b) - f(a))/(b-a) = k$
 $\langle \text{proof} \rangle$

lemma *real-average-minus-first [simp]*: $((a + b) / 2 - a) = (b-a)/(2::\text{real})$
 $\langle \text{proof} \rangle$

lemma *real-average-minus-second [simp]*: $((b + a) / 2 - a) = (b-a)/(2::\text{real})$
 $\langle \text{proof} \rangle$

Gallileo’s ”trick”: average velocity = av. of end velocities

lemma *DERIV-const-average*:
fixes $v :: \text{real} \Rightarrow \text{real}$
assumes $\text{neg: } a \neq (b::\text{real})$
and $\text{der: } \forall x. \text{DERIV } v \ x :> k$
shows $v ((a + b)/2) = (v \ a + v \ b)/2$
 $\langle \text{proof} \rangle$

60.9 Continuous injective functions

Dull lemma: an continuous injection on an interval must have a strict maximum at an end point, not in the middle.

lemma *lemma-isCont-inj*:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $d: 0 < d$
and $\text{inj [rule-format]: } \forall z. |z-x| \leq d \longrightarrow g(f \ z) = z$
and $\text{cont: } \forall z. |z-x| \leq d \longrightarrow \text{isCont } f \ z$
shows $\exists z. |z-x| \leq d \ \& \ f \ x < f \ z$
 $\langle \text{proof} \rangle$

Similar version for lower bound.

lemma *lemma-isCont-inj2*:
fixes $f \ g :: \text{real} \Rightarrow \text{real}$
shows $[|0 < d; \forall z. |z-x| \leq d \longrightarrow g(f \ z) = z;$
 $\forall z. |z-x| \leq d \longrightarrow \text{isCont } f \ z|]$
 $\implies \exists z. |z-x| \leq d \ \& \ f \ z < f \ x$

$\langle proof \rangle$

Show there’s an interval surrounding $f x$ in $f[[x - d, x + d]]$.

lemma *isCont-inj-range*:

fixes $f :: real \Rightarrow real$

assumes $d: 0 < d$

and *inj*: $\forall z. |z-x| \leq d \longrightarrow g(f z) = z$

and *cont*: $\forall z. |z-x| \leq d \longrightarrow isCont f z$

shows $\exists e > 0. \forall y. |y - f x| \leq e \longrightarrow (\exists z. |z-x| \leq d \ \& \ f z = y)$

$\langle proof \rangle$

Continuity of inverse function

lemma *isCont-inverse-function*:

fixes $f g :: real \Rightarrow real$

assumes $d: 0 < d$

and *inj*: $\forall z. |z-x| \leq d \longrightarrow g(f z) = z$

and *cont*: $\forall z. |z-x| \leq d \longrightarrow isCont f z$

shows $isCont g (f x)$

$\langle proof \rangle$

Derivative of inverse function

lemma *DERIV-inverse-function*:

fixes $f g :: real \Rightarrow real$

assumes *der*: $DERIV f (g x) :> D$

assumes *neq*: $D \neq 0$

assumes *a*: $a < x$ **and** *b*: $x < b$

assumes *inj*: $\forall y. a < y \wedge y < b \longrightarrow f (g y) = y$

assumes *cont*: $isCont g x$

shows $DERIV g x :> inverse D$

$\langle proof \rangle$

60.10 Generalized Mean Value Theorem

theorem *GMVT*:

fixes $a b :: real$

assumes *alb*: $a < b$

and *fc*: $\forall x. a \leq x \wedge x \leq b \longrightarrow isCont f x$

and *fd*: $\forall x. a < x \wedge x < b \longrightarrow f \text{ differentiable } x$

and *gc*: $\forall x. a \leq x \wedge x \leq b \longrightarrow isCont g x$

and *gd*: $\forall x. a < x \wedge x < b \longrightarrow g \text{ differentiable } x$

shows $\exists g'c f'c c. DERIV g c :> g'c \wedge DERIV f c :> f'c \wedge a < c \wedge c < b \wedge ((f b - f a) * g'c) = ((g b - g a) * f'c)$

$\langle proof \rangle$

60.11 Theorems about Limits

lemma *isCont-inv-fun*:

fixes $f g :: real \Rightarrow real$

shows $[| 0 < d; \forall z. |z - x| \leq d \longrightarrow g(f(z)) = z;$

$$\begin{aligned} & \forall z. |z - x| \leq d \dashv\dashv isCont\ f\ z \] \\ \implies & isCont\ g\ (f\ x) \\ \langle proof \rangle & \end{aligned}$$

lemma *isCont-inv-fun-inv*:

fixes $f\ g :: real \Rightarrow real$

shows $[| 0 < d;$

$$\begin{aligned} & \forall z. |z - x| \leq d \dashv\dashv g(f(z)) = z; \\ & \forall z. |z - x| \leq d \dashv\dashv isCont\ f\ z \] \\ \implies & \exists e. 0 < e \ \& \\ & (\forall y. 0 < |y - f(x)| \ \& |y - f(x)| < e \dashv\dashv f(g(y)) = y) \end{aligned}$$

$\langle proof \rangle$

Bartle/Sherbert: Introduction to Real Analysis, Theorem 4.2.9, p. 110

lemma *LIM-fun-gt-zero*:

$$\begin{aligned} & [f \dashv\dashv c \dashv\dashv (l::real); 0 < l \] \\ \implies & \exists r. 0 < r \ \& (\forall x::real. x \neq c \ \& |c - x| < r \dashv\dashv 0 < f\ x) \end{aligned}$$

$\langle proof \rangle$

lemma *LIM-fun-less-zero*:

$$\begin{aligned} & [f \dashv\dashv c \dashv\dashv (l::real); l < 0 \] \\ \implies & \exists r. 0 < r \ \& (\forall x::real. x \neq c \ \& |c - x| < r \dashv\dashv f\ x < 0) \end{aligned}$$

$\langle proof \rangle$

lemma *LIM-fun-not-zero*:

$$\begin{aligned} & [f \dashv\dashv c \dashv\dashv (l::real); l \neq 0 \] \\ \implies & \exists r. 0 < r \ \& (\forall x::real. x \neq c \ \& |c - x| < r \dashv\dashv f\ x \neq 0) \end{aligned}$$

$\langle proof \rangle$

end

61 Parity: Even and Odd for int and nat

theory *Parity*

imports *Main*

begin

class *even-odd* =

fixes $even :: 'a \Rightarrow bool$

abbreviation

$odd :: 'a::even-odd \Rightarrow bool$ **where**

$odd\ x \equiv \neg\ even\ x$

instantiation *nat* **and** *int* $:: even-odd$

begin

definition

even-def [*presburger*]: $\text{even } x \longleftrightarrow (x::\text{int}) \bmod 2 = 0$

definition

even-nat-def [*presburger*]: $\text{even } x \longleftrightarrow \text{even } (\text{int } x)$

instance $\langle \text{proof} \rangle$

end

61.1 Even and odd are mutually exclusive

lemma *int-pos-lt-two-imp-zero-or-one*:

$0 \leq x \implies (x::\text{int}) < 2 \implies x = 0 \mid x = 1$
 $\langle \text{proof} \rangle$

lemma *neq-one-mod-two* [*simp*, *presburger*]:

$((x::\text{int}) \bmod 2 \sim 0) = (x \bmod 2 = 1) \langle \text{proof} \rangle$

61.2 Behavior under integer arithmetic operations

declare *dvd-def* [*algebra*]

lemma *nat-even-iff-2-dvd* [*algebra*]: $\text{even } (x::\text{nat}) \longleftrightarrow 2 \text{ dvd } x$
 $\langle \text{proof} \rangle$

lemma *int-even-iff-2-dvd* [*algebra*]: $\text{even } (x::\text{int}) \longleftrightarrow 2 \text{ dvd } x$
 $\langle \text{proof} \rangle$

lemma *even-times-anything*: $\text{even } (x::\text{int}) \implies \text{even } (x * y)$
 $\langle \text{proof} \rangle$

lemma *anything-times-even*: $\text{even } (y::\text{int}) \implies \text{even } (x * y) \langle \text{proof} \rangle$

lemma *odd-times-odd*: $\text{odd } (x::\text{int}) \implies \text{odd } y \implies \text{odd } (x * y)$
 $\langle \text{proof} \rangle$

lemma *even-product* [*presburger*]: $\text{even } ((x::\text{int}) * y) = (\text{even } x \mid \text{even } y)$
 $\langle \text{proof} \rangle$

lemma *even-plus-even*: $\text{even } (x::\text{int}) \implies \text{even } y \implies \text{even } (x + y)$
 $\langle \text{proof} \rangle$

lemma *even-plus-odd*: $\text{even } (x::\text{int}) \implies \text{odd } y \implies \text{odd } (x + y)$
 $\langle \text{proof} \rangle$

lemma *odd-plus-even*: $\text{odd } (x::\text{int}) \implies \text{even } y \implies \text{odd } (x + y)$
 $\langle \text{proof} \rangle$

lemma *odd-plus-odd*: $\text{odd } (x::\text{int}) \implies \text{odd } y \implies \text{even } (x + y) \langle \text{proof} \rangle$

lemma *even-sum*[presburger]: $\text{even } ((x::\text{int}) + y) = ((\text{even } x \ \& \ \text{even } y) \mid (\text{odd } x \ \& \ \text{odd } y))$
 $\langle \text{proof} \rangle$

lemma *even-neg*[presburger, algebra]: $\text{even } (-(x::\text{int})) = \text{even } x$ $\langle \text{proof} \rangle$

lemma *even-difference*:
 $\text{even } ((x::\text{int}) - y) = ((\text{even } x \ \& \ \text{even } y) \mid (\text{odd } x \ \& \ \text{odd } y))$ $\langle \text{proof} \rangle$

lemma *even-pow-gt-zero*:
 $\text{even } (x::\text{int}) \implies 0 < n \implies \text{even } (x^n)$
 $\langle \text{proof} \rangle$

lemma *odd-pow-iff*[presburger, algebra]:
 $\text{odd } ((x::\text{int})^n) \iff (n = 0 \vee \text{odd } x)$
 $\langle \text{proof} \rangle$

lemma *odd-pow*: $\text{odd } x \implies \text{odd } ((x::\text{int})^n)$ $\langle \text{proof} \rangle$

lemma *even-power*[presburger]: $\text{even } ((x::\text{int})^n) = (\text{even } x \ \& \ 0 < n)$
 $\langle \text{proof} \rangle$

lemma *even-zero*[presburger]: $\text{even } (0::\text{int})$ $\langle \text{proof} \rangle$

lemma *odd-one*[presburger]: $\text{odd } (1::\text{int})$ $\langle \text{proof} \rangle$

lemmas *even-odd-simps* [simp] = *even-def*[of number-of v, standard] *even-zero*
odd-one *even-product* *even-sum* *even-neg* *even-difference* *even-power*

61.3 Equivalent definitions

lemma *two-times-even-div-two*: $\text{even } (x::\text{int}) \implies 2 * (x \text{ div } 2) = x$
 $\langle \text{proof} \rangle$

lemma *two-times-odd-div-two-plus-one*: $\text{odd } (x::\text{int}) \implies$
 $2 * (x \text{ div } 2) + 1 = x$ $\langle \text{proof} \rangle$

lemma *even-equiv-def*: $\text{even } (x::\text{int}) = (EX \ y. x = 2 * y)$ $\langle \text{proof} \rangle$

lemma *odd-equiv-def*: $\text{odd } (x::\text{int}) = (EX \ y. x = 2 * y + 1)$ $\langle \text{proof} \rangle$

61.4 even and odd for nats

lemma *pos-int-even-equiv-nat-even*: $0 \leq x \implies \text{even } x = \text{even } (\text{nat } x)$
 $\langle \text{proof} \rangle$

lemma *even-nat-product*[presburger, algebra]: $\text{even } ((x::\text{nat}) * y) = (\text{even } x \mid \text{even } y)$
 $\langle \text{proof} \rangle$

lemma *even-nat-sum*[presburger, algebra]: *even* $((x::nat) + y) =$
 $((even\ x \ \&\ even\ y) \mid (odd\ x \ \&\ odd\ y)) \langle proof \rangle$

lemma *even-nat-difference*[presburger, algebra]:
 $even\ ((x::nat) - y) = (x < y \mid (even\ x \ \&\ even\ y) \mid (odd\ x \ \&\ odd\ y))$
 $\langle proof \rangle$

lemma *even-nat-Suc*[presburger, algebra]: *even* $(Suc\ x) = odd\ x \langle proof \rangle$

lemma *even-nat-power*[presburger, algebra]: *even* $((x::nat) ^ y) = (even\ x \ \&\ 0 < y)$
 $\langle proof \rangle$

lemma *even-nat-zero*[presburger]: *even* $(0::nat) \langle proof \rangle$

lemmas *even-odd-nat-simps* [simp] = *even-nat-def*[of number-of v, standard]
even-nat-zero even-nat-Suc even-nat-product even-nat-sum even-nat-power

61.5 Equivalent definitions

lemma *nat-lt-two-imp-zero-or-one*: $(x::nat) < Suc\ (Suc\ 0) ==>$
 $x = 0 \mid x = Suc\ 0 \langle proof \rangle$

lemma *even-nat-mod-two-eq-zero*: *even* $(x::nat) ==> x\ mod\ (Suc\ (Suc\ 0)) = 0$
 $\langle proof \rangle$

lemma *odd-nat-mod-two-eq-one*: *odd* $(x::nat) ==> x\ mod\ (Suc\ (Suc\ 0)) = Suc\ 0$
 $\langle proof \rangle$

lemma *even-nat-equiv-def*: *even* $(x::nat) = (x\ mod\ Suc\ (Suc\ 0) = 0)$
 $\langle proof \rangle$

lemma *odd-nat-equiv-def*: *odd* $(x::nat) = (x\ mod\ Suc\ (Suc\ 0) = Suc\ 0)$
 $\langle proof \rangle$

lemma *even-nat-div-two-times-two*: *even* $(x::nat) ==>$
 $Suc\ (Suc\ 0) * (x\ div\ Suc\ (Suc\ 0)) = x \langle proof \rangle$

lemma *odd-nat-div-two-times-two-plus-one*: *odd* $(x::nat) ==>$
 $Suc\ (Suc\ (Suc\ 0) * (x\ div\ Suc\ (Suc\ 0))) = x \langle proof \rangle$

lemma *even-nat-equiv-def2*: *even* $(x::nat) = (EX\ y. x = Suc\ (Suc\ 0) * y)$
 $\langle proof \rangle$

lemma *odd-nat-equiv-def2*: *odd* $(x::nat) = (EX\ y. x = Suc\ (Suc\ (Suc\ 0) * y))$
 $\langle proof \rangle$

61.6 Parity and powers

lemma *minus-one-even-odd-power*:

$(\text{even } x \rightarrow (-1 :: 'a :: \{\text{comm-ring-1}, \text{recpower}\})^x = 1) \ \&$
 $(\text{odd } x \rightarrow (-1 :: 'a)^x = -1)$
 $\langle \text{proof} \rangle$

lemma *minus-one-even-power* [simp]:
 $\text{even } x \Rightarrow (-1 :: 'a :: \{\text{comm-ring-1}, \text{recpower}\})^x = 1$
 $\langle \text{proof} \rangle$

lemma *minus-one-odd-power* [simp]:
 $\text{odd } x \Rightarrow (-1 :: 'a :: \{\text{comm-ring-1}, \text{recpower}\})^x = -1$
 $\langle \text{proof} \rangle$

lemma *neg-one-even-odd-power*:
 $(\text{even } x \rightarrow (-1 :: 'a :: \{\text{number-ring}, \text{recpower}\})^x = 1) \ \&$
 $(\text{odd } x \rightarrow (-1 :: 'a)^x = -1)$
 $\langle \text{proof} \rangle$

lemma *neg-one-even-power* [simp]:
 $\text{even } x \Rightarrow (-1 :: 'a :: \{\text{number-ring}, \text{recpower}\})^x = 1$
 $\langle \text{proof} \rangle$

lemma *neg-one-odd-power* [simp]:
 $\text{odd } x \Rightarrow (-1 :: 'a :: \{\text{number-ring}, \text{recpower}\})^x = -1$
 $\langle \text{proof} \rangle$

lemma *neg-power-if*:
 $(-x :: 'a :: \{\text{comm-ring-1}, \text{recpower}\})^n =$
 $(\text{if even } n \text{ then } (x^{\wedge} n) \text{ else } -(x^{\wedge} n))$
 $\langle \text{proof} \rangle$

lemma *zero-le-even-power*: $\text{even } n \Rightarrow$
 $0 \leq (x :: 'a :: \{\text{recpower}, \text{ordered-ring-strict}\})^n$
 $\langle \text{proof} \rangle$

lemma *zero-le-odd-power*: $\text{odd } n \Rightarrow$
 $(0 \leq (x :: 'a :: \{\text{recpower}, \text{ordered-idom}\})^n) = (0 \leq x)$
 $\langle \text{proof} \rangle$

lemma *zero-le-power-eq*[presburger]: $(0 \leq (x :: 'a :: \{\text{recpower}, \text{ordered-idom}\})^n)$
 $=$
 $(\text{even } n \mid (\text{odd } n \ \& \ 0 \leq x))$
 $\langle \text{proof} \rangle$

lemma *zero-less-power-eq*[presburger]: $(0 < (x :: 'a :: \{\text{recpower}, \text{ordered-idom}\})^n)$
 $=$
 $(n = 0 \mid (\text{even } n \ \& \ x \sim 0) \mid (\text{odd } n \ \& \ 0 < x))$
 $\langle \text{proof} \rangle$

lemma *power-less-zero-eq*[presburger]: $((x::'a::\{\text{recpower}, \text{ordered-idom}\})^{\wedge} n < 0)$
 $=$
 $(\text{odd } n \ \& \ x < 0)$
 $\langle \text{proof} \rangle$

lemma *power-le-zero-eq*[presburger]: $((x::'a::\{\text{recpower}, \text{ordered-idom}\})^{\wedge} n \leq 0)$
 $=$
 $(n \sim 0 \ \& \ ((\text{odd } n \ \& \ x \leq 0) \mid (\text{even } n \ \& \ x = 0)))$
 $\langle \text{proof} \rangle$

lemma *power-even-abs*: $\text{even } n \implies$
 $(\text{abs } (x::'a::\{\text{recpower}, \text{ordered-idom}\}))^{\wedge} n = x^{\wedge} n$
 $\langle \text{proof} \rangle$

lemma *zero-less-power-nat-eq*[presburger]: $(0 < (x::\text{nat})^{\wedge} n) = (n = 0 \mid 0 < x)$
 $\langle \text{proof} \rangle$

lemma *power-minus-even* [simp]: $\text{even } n \implies$
 $(- \ x)^{\wedge} n = (x^{\wedge} n::'a::\{\text{recpower}, \text{comm-ring-1}\})$
 $\langle \text{proof} \rangle$

lemma *power-minus-odd* [simp]: $\text{odd } n \implies$
 $(- \ x)^{\wedge} n = - \ (x^{\wedge} n::'a::\{\text{recpower}, \text{comm-ring-1}\})$
 $\langle \text{proof} \rangle$

lemma *power-mono-even*: **fixes** $x \ y :: 'a :: \{\text{recpower}, \text{ordered-idom}\}$
assumes $\text{even } n$ **and** $|x| \leq |y|$
shows $x^{\wedge} n \leq y^{\wedge} n$
 $\langle \text{proof} \rangle$

lemma *odd-pos*: $\text{odd } (n::\text{nat}) \implies 0 < n$ $\langle \text{proof} \rangle$

lemma *power-mono-odd*: **fixes** $x \ y :: 'a :: \{\text{recpower}, \text{ordered-idom}\}$
assumes $\text{odd } n$ **and** $x \leq y$
shows $x^{\wedge} n \leq y^{\wedge} n$
 $\langle \text{proof} \rangle$

61.7 General Lemmas About Division

lemma *Suc-times-mod-eq*: $1 < k \implies \text{Suc } (k * m) \bmod k = 1$
 $\langle \text{proof} \rangle$

declare *Suc-times-mod-eq* [of number-of w , standard, simp]

lemma [simp]: $n \bmod k \leq (\text{Suc } n) \bmod k$
 $\langle \text{proof} \rangle$

lemma *Suc-n-div-2-gt-zero* [simp]: $(0::\text{nat}) < n \implies 0 < (n + 1) \bmod 2$
 $\langle \text{proof} \rangle$

lemma *div-2-gt-zero* [simp]: $(1::nat) < n \implies 0 < n \text{ div } 2$
 ⟨proof⟩

lemma *mod-mult-self3* [simp]: $(k*n + m) \text{ mod } n = m \text{ mod } (n::nat)$
 ⟨proof⟩

lemma *mod-mult-self4* [simp]: $\text{Suc } (k*n + m) \text{ mod } n = \text{Suc } m \text{ mod } n$
 ⟨proof⟩

lemma *mod-Suc-eq-Suc-mod*: $\text{Suc } m \text{ mod } n = \text{Suc } (m \text{ mod } n) \text{ mod } n$
 ⟨proof⟩

61.8 More Even/Odd Results

lemma *even-mult-two-ex*: $\text{even}(n) = (\exists m::nat. n = 2*m)$ ⟨proof⟩

lemma *odd-Suc-mult-two-ex*: $\text{odd}(n) = (\exists m. n = \text{Suc } (2*m))$ ⟨proof⟩

lemma *even-add* [simp]: $\text{even}(m + n::nat) = (\text{even } m = \text{even } n)$ ⟨proof⟩

lemma *odd-add* [simp]: $\text{odd}(m + n::nat) = (\text{odd } m \neq \text{odd } n)$ ⟨proof⟩

lemma *div-Suc*: $\text{Suc } a \text{ div } c = a \text{ div } c + \text{Suc } 0 \text{ div } c +$
 $(a \text{ mod } c + \text{Suc } 0 \text{ mod } c) \text{ div } c$
 ⟨proof⟩

lemma *lemma-even-div2* [simp]: $\text{even } (n::nat) \implies (n + 1) \text{ div } 2 = n \text{ div } 2$
 ⟨proof⟩

lemma *lemma-not-even-div2* [simp]: $\sim \text{even } n \implies (n + 1) \text{ div } 2 = \text{Suc } (n \text{ div } 2)$
 ⟨proof⟩

lemma *even-num-iff*: $0 < n \implies \text{even } n = (\sim \text{even}(n - 1 :: nat))$ ⟨proof⟩

lemma *even-even-mod-4-iff*: $\text{even } (n::nat) = \text{even } (n \text{ mod } 4)$ ⟨proof⟩

lemma *lemma-odd-mod-4-div-2*: $n \text{ mod } 4 = (3::nat) \implies \text{odd}((n - 1) \text{ div } 2)$
 ⟨proof⟩

lemma *lemma-even-mod-4-div-2*: $n \text{ mod } 4 = (1::nat) \implies \text{even}((n - 1) \text{ div } 2)$
 ⟨proof⟩

Simplify, when the exponent is a numeral

lemmas *power-0-left-number-of* = *power-0-left* [of number-of w, standard]

declare *power-0-left-number-of* [simp]

lemmas *zero-le-power-eq-number-of* [simp] =
zero-le-power-eq [of - number-of w, standard]

lemmas *zero-less-power-eq-number-of* [simp] =
zero-less-power-eq [of - number-of *w*, standard]

lemmas *power-le-zero-eq-number-of* [simp] =
power-le-zero-eq [of - number-of *w*, standard]

lemmas *power-less-zero-eq-number-of* [simp] =
power-less-zero-eq [of - number-of *w*, standard]

lemmas *zero-less-power-nat-eq-number-of* [simp] =
zero-less-power-nat-eq [of - number-of *w*, standard]

lemmas *power-eq-0-iff-number-of* [simp] = *power-eq-0-iff* [of - number-of *w*, standard]

lemmas *power-even-abs-number-of* [simp] = *power-even-abs* [of number-of *w* -, standard]

61.9 An Equivalence for $0 \leq a^n$

lemma *even-power-le-0-imp-0*:
 $a^n (2*k) \leq (0::'a::\{\text{ordered-idom}, \text{recpower}\}) \implies a=0$
 ⟨proof⟩

lemma *zero-le-power-iff* [presburger]:
 $(0 \leq a^n) = (0 \leq (a::'a::\{\text{ordered-idom}, \text{recpower}\}) \mid \text{even } n)$
 ⟨proof⟩

61.10 Miscellaneous

lemma [presburger]: $(x + 1) \text{ div } 2 = x \text{ div } 2 \iff \text{even } (x::\text{int})$ ⟨proof⟩

lemma [presburger]: $(x + 1) \text{ div } 2 = x \text{ div } 2 + 1 \iff \text{odd } (x::\text{int})$ ⟨proof⟩

lemma *even-plus-one-div-two*: $\text{even } (x::\text{int}) \implies (x + 1) \text{ div } 2 = x \text{ div } 2$ ⟨proof⟩

lemma *odd-plus-one-div-two*: $\text{odd } (x::\text{int}) \implies (x + 1) \text{ div } 2 = x \text{ div } 2 + 1$
 ⟨proof⟩

lemma [presburger]: $(\text{Suc } x) \text{ div } \text{Suc } (\text{Suc } 0) = x \text{ div } \text{Suc } (\text{Suc } 0) \iff \text{even } x$
 ⟨proof⟩

lemma [presburger]: $(\text{Suc } x) \text{ div } \text{Suc } (\text{Suc } 0) = x \text{ div } \text{Suc } (\text{Suc } 0) \iff \text{even } x$
 ⟨proof⟩

lemma *even-nat-plus-one-div-two*: $\text{even } (x::\text{nat}) \implies$
 $(\text{Suc } x) \text{ div } \text{Suc } (\text{Suc } 0) = x \text{ div } \text{Suc } (\text{Suc } 0)$ ⟨proof⟩

lemma *odd-nat-plus-one-div-two*: $\text{odd } (x::\text{nat}) \implies$
 $(\text{Suc } x) \text{ div } \text{Suc } (\text{Suc } 0) = \text{Suc } (x \text{ div } \text{Suc } (\text{Suc } 0))$ ⟨proof⟩

end

62 NthRoot: Nth Roots of Real Numbers

```
theory NthRoot
imports Parity Deriv
begin
```

62.1 Existence of Nth Root

Existence follows from the Intermediate Value Theorem

```
lemma realpow-pos-nth:
  assumes n: 0 < n
  assumes a: 0 < a
  shows  $\exists r > 0. r^n = (a::real)$ 
<proof>
```

```
lemma realpow-pos-nth2:  $(0::real) < a \implies \exists r > 0. r^{Suc\ n} = a$ 
<proof>
```

Uniqueness of nth positive root

```
lemma realpow-pos-nth-unique:
   $\llbracket 0 < n; 0 < a \rrbracket \implies \exists! r. 0 < r \wedge r^n = (a::real)$ 
<proof>
```

62.2 Nth Root

We define roots of negative reals such that $root\ n\ (-x) = -\ root\ n\ x$. This allows us to omit side conditions from many theorems.

definition

```
root :: [nat, real]  $\Rightarrow$  real where
  root n x = (if 0 < x then (THE u. 0 < u  $\wedge$  u^n = x) else
    if x < 0 then - (THE u. 0 < u  $\wedge$  u^n = - x) else 0)
```

```
lemma real-root-zero [simp]: root n 0 = 0
<proof>
```

```
lemma real-root-minus: 0 < n  $\implies$  root n (- x) = - root n x
<proof>
```

```
lemma real-root-gt-zero:  $\llbracket 0 < n; 0 < x \rrbracket \implies 0 < root\ n\ x$ 
<proof>
```

```
lemma real-root-pow-pos:
   $\llbracket 0 < n; 0 < x \rrbracket \implies root\ n\ x^n = x$ 
<proof>
```

```
lemma real-root-pow-pos2 [simp]:
   $\llbracket 0 < n; 0 \leq x \rrbracket \implies root\ n\ x^n = x$ 
```

<proof>

lemma *odd-real-root-pow*: $\text{odd } n \implies \text{root } n \ x \ ^n = x$
<proof>

lemma *real-root-ge-zero*: $\llbracket 0 < n; 0 \leq x \rrbracket \implies 0 \leq \text{root } n \ x$
<proof>

lemma *real-root-power-cancel*: $\llbracket 0 < n; 0 \leq x \rrbracket \implies \text{root } n \ (x \ ^n) = x$
<proof>

lemma *odd-real-root-power-cancel*: $\text{odd } n \implies \text{root } n \ (x \ ^n) = x$
<proof>

lemma *real-root-pos-unique*:
 $\llbracket 0 < n; 0 \leq y; y \ ^n = x \rrbracket \implies \text{root } n \ x = y$
<proof>

lemma *odd-real-root-unique*:
 $\llbracket \text{odd } n; y \ ^n = x \rrbracket \implies \text{root } n \ x = y$
<proof>

lemma *real-root-one [simp]*: $0 < n \implies \text{root } n \ 1 = 1$
<proof>

Root function is strictly monotonic, hence injective

lemma *real-root-less-mono-lemma*:
 $\llbracket 0 < n; 0 \leq x; x < y \rrbracket \implies \text{root } n \ x < \text{root } n \ y$
<proof>

lemma *real-root-less-mono*: $\llbracket 0 < n; x < y \rrbracket \implies \text{root } n \ x < \text{root } n \ y$
<proof>

lemma *real-root-le-mono*: $\llbracket 0 < n; x \leq y \rrbracket \implies \text{root } n \ x \leq \text{root } n \ y$
<proof>

lemma *real-root-less-iff [simp]*:
 $0 < n \implies (\text{root } n \ x < \text{root } n \ y) = (x < y)$
<proof>

lemma *real-root-le-iff [simp]*:
 $0 < n \implies (\text{root } n \ x \leq \text{root } n \ y) = (x \leq y)$
<proof>

lemma *real-root-eq-iff [simp]*:
 $0 < n \implies (\text{root } n \ x = \text{root } n \ y) = (x = y)$
<proof>

lemmas *real-root-gt-0-iff [simp] = real-root-less-iff [where x=0, simplified]*

lemmas *real-root-lt-0-iff* [simp] = *real-root-less-iff* [where $y=0$, simplified]

lemmas *real-root-ge-0-iff* [simp] = *real-root-le-iff* [where $x=0$, simplified]

lemmas *real-root-le-0-iff* [simp] = *real-root-le-iff* [where $y=0$, simplified]

lemmas *real-root-eq-0-iff* [simp] = *real-root-eq-iff* [where $y=0$, simplified]

lemma *real-root-gt-1-iff* [simp]: $0 < n \implies (1 < \text{root } n \ y) = (1 < y)$
 <proof>

lemma *real-root-lt-1-iff* [simp]: $0 < n \implies (\text{root } n \ x < 1) = (x < 1)$
 <proof>

lemma *real-root-ge-1-iff* [simp]: $0 < n \implies (1 \leq \text{root } n \ y) = (1 \leq y)$
 <proof>

lemma *real-root-le-1-iff* [simp]: $0 < n \implies (\text{root } n \ x \leq 1) = (x \leq 1)$
 <proof>

lemma *real-root-eq-1-iff* [simp]: $0 < n \implies (\text{root } n \ x = 1) = (x = 1)$
 <proof>

Roots of roots

lemma *real-root-Suc-0* [simp]: $\text{root } (\text{Suc } 0) \ x = x$
 <proof>

lemma *real-root-pos-mult-exp*:
 $\llbracket 0 < m; 0 < n; 0 < x \rrbracket \implies \text{root } (m * n) \ x = \text{root } m \ (\text{root } n \ x)$
 <proof>

lemma *real-root-mult-exp*:
 $\llbracket 0 < m; 0 < n \rrbracket \implies \text{root } (m * n) \ x = \text{root } m \ (\text{root } n \ x)$
 <proof>

lemma *real-root-commute*:
 $\llbracket 0 < m; 0 < n \rrbracket \implies \text{root } m \ (\text{root } n \ x) = \text{root } n \ (\text{root } m \ x)$
 <proof>

Monotonicity in first argument

lemma *real-root-strict-decreasing*:
 $\llbracket 0 < n; n < N; 1 < x \rrbracket \implies \text{root } N \ x < \text{root } n \ x$
 <proof>

lemma *real-root-strict-increasing*:
 $\llbracket 0 < n; n < N; 0 < x; x < 1 \rrbracket \implies \text{root } n \ x < \text{root } N \ x$
 <proof>

lemma *real-root-decreasing*:
 $\llbracket 0 < n; n < N; 1 \leq x \rrbracket \implies \text{root } N \ x \leq \text{root } n \ x$
 <proof>

lemma *real-root-increasing*:

$\llbracket 0 < n; n < N; 0 \leq x; x \leq 1 \rrbracket \implies \text{root } n \ x \leq \text{root } N \ x$
 $\langle \text{proof} \rangle$

Roots of multiplication and division

lemma *real-root-mult-lemma*:

$\llbracket 0 < n; 0 \leq x; 0 \leq y \rrbracket \implies \text{root } n \ (x * y) = \text{root } n \ x * \text{root } n \ y$
 $\langle \text{proof} \rangle$

lemma *real-root-inverse-lemma*:

$\llbracket 0 < n; 0 \leq x \rrbracket \implies \text{root } n \ (\text{inverse } x) = \text{inverse } (\text{root } n \ x)$
 $\langle \text{proof} \rangle$

lemma *real-root-mult*:

assumes $n: 0 < n$
shows $\text{root } n \ (x * y) = \text{root } n \ x * \text{root } n \ y$
 $\langle \text{proof} \rangle$

lemma *real-root-inverse*:

assumes $n: 0 < n$
shows $\text{root } n \ (\text{inverse } x) = \text{inverse } (\text{root } n \ x)$
 $\langle \text{proof} \rangle$

lemma *real-root-divide*:

$0 < n \implies \text{root } n \ (x / y) = \text{root } n \ x / \text{root } n \ y$
 $\langle \text{proof} \rangle$

lemma *real-root-power*:

$0 < n \implies \text{root } n \ (x ^ k) = \text{root } n \ x ^ k$
 $\langle \text{proof} \rangle$

lemma *real-root-abs*: $0 < n \implies \text{root } n \ |x| = |\text{root } n \ x|$
 $\langle \text{proof} \rangle$

Continuity and derivatives

lemma *isCont-root-pos*:

assumes $n: 0 < n$
assumes $x: 0 < x$
shows $\text{isCont } (\text{root } n) \ x$
 $\langle \text{proof} \rangle$

lemma *isCont-root-neg*:

$\llbracket 0 < n; x < 0 \rrbracket \implies \text{isCont } (\text{root } n) \ x$
 $\langle \text{proof} \rangle$

lemma *isCont-root-zero*:

$0 < n \implies \text{isCont } (\text{root } n) \ 0$
 $\langle \text{proof} \rangle$

lemma *isCont-real-root*: $0 < n \implies \text{isCont } (\text{root } n) \ x$
 $\langle \text{proof} \rangle$

lemma *DERIV-real-root*:
 assumes $n: 0 < n$
 assumes $x: 0 < x$
 shows $\text{DERIV } (\text{root } n) \ x :> \text{inverse } (\text{real } n * \text{root } n \ x \ ^{\wedge} (n - \text{Suc } 0))$
 $\langle \text{proof} \rangle$

lemma *DERIV-odd-real-root*:
 assumes $n: \text{odd } n$
 assumes $x: x \neq 0$
 shows $\text{DERIV } (\text{root } n) \ x :> \text{inverse } (\text{real } n * \text{root } n \ x \ ^{\wedge} (n - \text{Suc } 0))$
 $\langle \text{proof} \rangle$

62.3 Square Root

definition
 $\text{sqrt} :: \text{real} \Rightarrow \text{real}$ **where**
 $\text{sqrt} = \text{root } 2$

lemma *pos2*: $0 < (2::\text{nat}) \ \langle \text{proof} \rangle$

lemma *real-sqrt-unique*: $\llbracket y^2 = x; 0 \leq y \rrbracket \implies \text{sqrt } x = y$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-abs* [simp]: $\text{sqrt } (x^2) = |x|$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-pow2* [simp]: $0 \leq x \implies (\text{sqrt } x)^2 = x$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-pow2-iff* [simp]: $((\text{sqrt } x)^2 = x) = (0 \leq x)$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-zero* [simp]: $\text{sqrt } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-one* [simp]: $\text{sqrt } 1 = 1$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-minus*: $\text{sqrt } (-x) = - \text{sqrt } x$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-mult*: $\text{sqrt } (x * y) = \text{sqrt } x * \text{sqrt } y$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-inverse*: $\text{sqrt } (\text{inverse } x) = \text{inverse } (\text{sqrt } x)$
 $\langle \text{proof} \rangle$

lemma *real-sqrt-divide*: $\text{sqrt } (x / y) = \text{sqrt } x / \text{sqrt } y$
 ⟨proof⟩

lemma *real-sqrt-power*: $\text{sqrt } (x ^ k) = \text{sqrt } x ^ k$
 ⟨proof⟩

lemma *real-sqrt-gt-zero*: $0 < x \implies 0 < \text{sqrt } x$
 ⟨proof⟩

lemma *real-sqrt-ge-zero*: $0 \leq x \implies 0 \leq \text{sqrt } x$
 ⟨proof⟩

lemma *real-sqrt-less-mono*: $x < y \implies \text{sqrt } x < \text{sqrt } y$
 ⟨proof⟩

lemma *real-sqrt-le-mono*: $x \leq y \implies \text{sqrt } x \leq \text{sqrt } y$
 ⟨proof⟩

lemma *real-sqrt-less-iff* [simp]: $(\text{sqrt } x < \text{sqrt } y) = (x < y)$
 ⟨proof⟩

lemma *real-sqrt-le-iff* [simp]: $(\text{sqrt } x \leq \text{sqrt } y) = (x \leq y)$
 ⟨proof⟩

lemma *real-sqrt-eq-iff* [simp]: $(\text{sqrt } x = \text{sqrt } y) = (x = y)$
 ⟨proof⟩

lemmas *real-sqrt-gt-0-iff* [simp] = *real-sqrt-less-iff* [where $x=0$, simplified]
lemmas *real-sqrt-lt-0-iff* [simp] = *real-sqrt-less-iff* [where $y=0$, simplified]
lemmas *real-sqrt-ge-0-iff* [simp] = *real-sqrt-le-iff* [where $x=0$, simplified]
lemmas *real-sqrt-le-0-iff* [simp] = *real-sqrt-le-iff* [where $y=0$, simplified]
lemmas *real-sqrt-eq-0-iff* [simp] = *real-sqrt-eq-iff* [where $y=0$, simplified]

lemmas *real-sqrt-gt-1-iff* [simp] = *real-sqrt-less-iff* [where $x=1$, simplified]
lemmas *real-sqrt-lt-1-iff* [simp] = *real-sqrt-less-iff* [where $y=1$, simplified]
lemmas *real-sqrt-ge-1-iff* [simp] = *real-sqrt-le-iff* [where $x=1$, simplified]
lemmas *real-sqrt-le-1-iff* [simp] = *real-sqrt-le-iff* [where $y=1$, simplified]
lemmas *real-sqrt-eq-1-iff* [simp] = *real-sqrt-eq-iff* [where $y=1$, simplified]

lemma *isCont-real-sqrt*: *isCont* $\text{sqrt } x$
 ⟨proof⟩

lemma *DERIV-real-sqrt*:
 $0 < x \implies \text{DERIV } \text{sqrt } x \text{ :> inverse } (\text{sqrt } x) / 2$
 ⟨proof⟩

lemma *not-real-square-gt-zero* [simp]: $(\sim (0::\text{real}) < x*x) = (x = 0)$
 ⟨proof⟩

lemma *real-sqrt-abs2* [simp]: $\text{sqrt}(x*x) = |x|$
 ⟨proof⟩

lemma *real-sqrt-pow2-gt-zero*: $0 < x \implies 0 < (\text{sqrt } x)^2$
 ⟨proof⟩

lemma *real-sqrt-not-eq-zero*: $0 < x \implies \text{sqrt } x \neq 0$
 ⟨proof⟩

lemma *real-inv-sqrt-pow2*: $0 < x \implies \text{inverse } (\text{sqrt}(x)) ^ 2 = \text{inverse } x$
 ⟨proof⟩

lemma *real-sqrt-eq-zero-cancel*: $[| 0 \leq x; \text{sqrt}(x) = 0 |] \implies x = 0$
 ⟨proof⟩

lemma *real-sqrt-ge-one*: $1 \leq x \implies 1 \leq \text{sqrt } x$
 ⟨proof⟩

lemma *real-sqrt-two-gt-zero* [simp]: $0 < \text{sqrt } 2$
 ⟨proof⟩

lemma *real-sqrt-two-ge-zero* [simp]: $0 \leq \text{sqrt } 2$
 ⟨proof⟩

lemma *real-sqrt-two-gt-one* [simp]: $1 < \text{sqrt } 2$
 ⟨proof⟩

lemma *sqrt-divide-self-eq*:
 assumes *nneg*: $0 \leq x$
 shows $\text{sqrt } x / x = \text{inverse } (\text{sqrt } x)$
 ⟨proof⟩

lemma *real-divide-square-eq* [simp]: $((r::\text{real}) * a) / (r * r) = a / r$
 ⟨proof⟩

lemma *lemma-real-divide-sqrt-less*: $0 < u \implies u / \text{sqrt } 2 < u$
 ⟨proof⟩

lemma *four-x-squared*:
 fixes *x*::*real*
 shows $4 * x^2 = (2 * x)^2$
 ⟨proof⟩

62.4 Square Root of Sum of Squares

lemma *real-sqrt-mult-self-sum-ge-zero* [simp]: $0 \leq \text{sqrt}(x*x + y*y)$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-ge-zero* [simp]: $0 \leq \text{sqrt } (x^2 + y^2)$
 ⟨proof⟩

declare *real-sqrt-sum-squares-ge-zero* [THEN abs-of-nonneg, simp]

lemma *real-sqrt-sum-squares-mult-ge-zero* [simp]:
 $0 \leq \text{sqrt } ((x^2 + y^2) * (xa^2 + ya^2))$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-mult-squared-eq* [simp]:
 $\text{sqrt } ((x^2 + y^2) * (xa^2 + ya^2)) ^ 2 = (x^2 + y^2) * (xa^2 + ya^2)$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-eq-cancel*: $\text{sqrt } (x^2 + y^2) = x \implies y = 0$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-eq-cancel2*: $\text{sqrt } (x^2 + y^2) = y \implies x = 0$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-ge1* [simp]: $x \leq \text{sqrt } (x^2 + y^2)$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-ge2* [simp]: $y \leq \text{sqrt } (x^2 + y^2)$
 ⟨proof⟩

lemma *real-sqrt-ge-abs1* [simp]: $|x| \leq \text{sqrt } (x^2 + y^2)$
 ⟨proof⟩

lemma *real-sqrt-ge-abs2* [simp]: $|y| \leq \text{sqrt } (x^2 + y^2)$
 ⟨proof⟩

lemma *le-real-sqrt-sumsq* [simp]: $x \leq \text{sqrt } (x * x + y * y)$
 ⟨proof⟩

lemma *power2-sum*:
 fixes $x y :: 'a :: \{\text{number-ring}, \text{recpower}\}$
 shows $(x + y)^2 = x^2 + y^2 + 2 * x * y$
 ⟨proof⟩

lemma *power2-diff*:
 fixes $x y :: 'a :: \{\text{number-ring}, \text{recpower}\}$
 shows $(x - y)^2 = x^2 + y^2 - 2 * x * y$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-triangle-ineq*:
 $\text{sqrt } ((a + c)^2 + (b + d)^2) \leq \text{sqrt } (a^2 + b^2) + \text{sqrt } (c^2 + d^2)$
 ⟨proof⟩

lemma *real-sqrt-sum-squares-less*:

$\llbracket |x| < u / \text{sqrt } 2; |y| < u / \text{sqrt } 2 \rrbracket \implies \text{sqrt } (x^2 + y^2) < u$
 $\langle \text{proof} \rangle$

Needed for the infinitely close relation over the nonstandard complex numbers

lemma *lemma-sqrt-hcomplex-capprox*:

$\llbracket 0 < u; x < u/2; y < u/2; 0 \leq x; 0 \leq y \rrbracket \implies \text{sqrt } (x^2 + y^2) < u$
 $\langle \text{proof} \rangle$

Legacy theorem names:

lemmas *real-root-pos2 = real-root-power-cancel*

lemmas *real-root-pos-pos = real-root-gt-zero [THEN order-less-imp-le]*

lemmas *real-root-pos-pos-le = real-root-ge-zero*

lemmas *real-sqrt-mult-distrib = real-sqrt-mult*

lemmas *real-sqrt-mult-distrib2 = real-sqrt-mult*

lemmas *real-sqrt-eq-zero-cancel-iff = real-sqrt-eq-0-iff*

lemma *real-root-pos*: $0 < x \implies \text{root } (\text{Suc } n) (x \wedge (\text{Suc } n)) = x$
 $\langle \text{proof} \rangle$

end

63 Transcendental: Power Series, Transcendental Functions etc.

theory *Transcendental*

imports *Fact Series Deriv NthRoot*

begin

63.1 Properties of Power Series

lemma *lemma-realpow-diff*:

fixes $y :: 'a::\text{recpower}$

shows $p \leq n \implies y \wedge (\text{Suc } n - p) = (y \wedge (n - p)) * y$

$\langle \text{proof} \rangle$

lemma *lemma-realpow-diff-sumr*:

fixes $y :: 'a::\{\text{recpower}, \text{comm-semiring-0}\}$ **shows**

$(\sum p=0..<\text{Suc } n. (x \wedge p) * y \wedge (\text{Suc } n - p)) =$
 $y * (\sum p=0..<\text{Suc } n. (x \wedge p) * y \wedge (n - p))$

$\langle \text{proof} \rangle$

lemma *lemma-realpow-diff-sumr2*:

fixes $y :: 'a::\{\text{recpower}, \text{comm-ring}\}$ **shows**

$x \wedge (\text{Suc } n) - y \wedge (\text{Suc } n) =$
 $(x - y) * (\sum p=0..<\text{Suc } n. (x \wedge p) * y \wedge (n - p))$

⟨proof⟩

lemma *lemma-realpow-rev-sumr*:

$$\left(\sum_{p=0..< \text{Suc } n}. (x \wedge p) * (y \wedge (n - p))\right) = \left(\sum_{p=0..< \text{Suc } n}. (x \wedge (n - p)) * (y \wedge p)\right)$$

⟨proof⟩

Power series has a ‘circle’ of convergence, i.e. if it sums for x , then it sums absolutely for z with $|z| < |x|$.

lemma *powser-insidea*:

fixes $x z :: 'a :: \{\text{real-normed-field}, \text{banach}, \text{recpower}\}$

assumes 1: *summable* $(\lambda n. f\ n * x \wedge n)$

assumes 2: *norm* $z < \text{norm } x$

shows *summable* $(\lambda n. \text{norm } (f\ n * z \wedge n))$

⟨proof⟩

lemma *powser-inside*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{real-normed-field}, \text{banach}, \text{recpower}\}$ **shows**

$[[\text{summable } (\%n. f(n) * (x \wedge n)); \text{norm } z < \text{norm } x]]$

$\implies \text{summable } (\%n. f(n) * (z \wedge n))$

⟨proof⟩

lemma *sum-split-even-odd*: **fixes** $f :: \text{nat} \Rightarrow \text{real}$ **shows**

$(\sum i = 0 ..< 2 * n. \text{if even } i \text{ then } f\ i \text{ else } g\ i) =$

$(\sum i = 0 ..< n. f\ (2 * i)) + (\sum i = 0 ..< n. g\ (2 * i + 1))$

⟨proof⟩

lemma *sums-if'*: **fixes** $g :: \text{nat} \Rightarrow \text{real}$ **assumes** $g \text{ sums } x$

shows $(\lambda n. \text{if even } n \text{ then } 0 \text{ else } g\ ((n - 1) \text{ div } 2)) \text{ sums } x$

⟨proof⟩

lemma *sums-if*: **fixes** $g :: \text{nat} \Rightarrow \text{real}$ **assumes** $g \text{ sums } x$ **and** $f \text{ sums } y$

shows $(\lambda n. \text{if even } n \text{ then } f\ (n \text{ div } 2) \text{ else } g\ ((n - 1) \text{ div } 2)) \text{ sums } (x + y)$

⟨proof⟩

63.2 Alternating series test / Leibniz formula

lemma *sums-alternating-upper-lower*:

fixes $a :: \text{nat} \Rightarrow \text{real}$

assumes *mono*: $\bigwedge n. a\ (\text{Suc } n) \leq a\ n$ **and** *a-pos*: $\bigwedge n. 0 \leq a\ n$ **and** $a \text{ ----> } 0$

shows $\exists l. ((\forall n. (\sum_{i=0..< 2*n}. -1^{i*a\ i}) \leq l) \wedge (\lambda n. \sum_{i=0..< 2*n}. -1^{i*a\ i}) \text{ ----> } l) \wedge$

$((\forall n. l \leq (\sum_{i=0..< 2*n+1}. -1^{i*a\ i})) \wedge (\lambda n. \sum_{i=0..< 2*n+1}. -1^{i*a\ i}) \text{ ----> } l)$

$(\text{is } \exists l. ((\forall n. ?f\ n \leq l) \wedge -) \wedge ((\forall n. l \leq ?g\ n) \wedge -))$

⟨proof⟩

lemma *summable-Leibniz'*: **fixes** $a :: \text{nat} \Rightarrow \text{real}$

assumes *a-zero*: $a \dashrightarrow 0$ **and** *a-pos*: $\bigwedge n. 0 \leq a\ n$
and *a-monotone*: $\bigwedge n. a\ (Suc\ n) \leq a\ n$
shows *summable*: $summable\ (\lambda n. (-1)^n * a\ n)$
and $\bigwedge n. (\sum_{i=0..<2*n} (-1)^i * a\ i) \leq (\sum i. (-1)^i * a\ i)$
and $(\lambda n. \sum_{i=0..<2*n} (-1)^i * a\ i) \dashrightarrow (\sum i. (-1)^i * a\ i)$
and $\bigwedge n. (\sum i. (-1)^i * a\ i) \leq (\sum_{i=0..<2*n+1} (-1)^i * a\ i)$
and $(\lambda n. \sum_{i=0..<2*n+1} (-1)^i * a\ i) \dashrightarrow (\sum i. (-1)^i * a\ i)$
 <proof>

theorem *summable-Leibniz*: **fixes** $a :: nat \Rightarrow real$
assumes *a-zero*: $a \dashrightarrow 0$ **and** *monoseq* a
shows *summable* $(\lambda n. (-1)^n * a\ n)$ **(is ?summable)**
and $0 < a\ 0 \longrightarrow (\forall n. (\sum i. -1^i * a\ i) \in \{ \sum_{i=0..<2*n} -1^i * a\ i .. \sum_{i=0..<2*n+1} -1^i * a\ i \})$ **(is ?pos)**
and $a\ 0 < 0 \longrightarrow (\forall n. (\sum i. -1^i * a\ i) \in \{ \sum_{i=0..<2*n+1} -1^i * a\ i .. \sum_{i=0..<2*n} -1^i * a\ i \})$ **(is ?neg)**
and $(\lambda n. \sum_{i=0..<2*n} -1^i * a\ i) \dashrightarrow (\sum i. -1^i * a\ i)$ **(is ?f)**
and $(\lambda n. \sum_{i=0..<2*n+1} -1^i * a\ i) \dashrightarrow (\sum i. -1^i * a\ i)$ **(is ?g)**
 <proof>

63.3 Term-by-Term Differentiability of Power Series

definition

diffs :: $(nat \Rightarrow 'a::ring-1) \Rightarrow nat \Rightarrow 'a$ **where**
diffs $c = (\%n. of_nat\ (Suc\ n) * c(Suc\ n))$

Lemma about distributing negation over it

lemma *diffs-minus*: $diffs\ (\%n. -\ c\ n) = (\%n. -\ diffs\ c\ n)$
 <proof>

lemma *sums-Suc-imp*:

assumes $f: f\ 0 = 0$
shows $(\lambda n. f\ (Suc\ n))\ sums\ s \implies (\lambda n. f\ n)\ sums\ s$
 <proof>

lemma *diffs-equiv*:

$summable\ (\%n. (diffs\ c)(n) * (x^{\wedge} n)) \implies$
 $(\%n. of_nat\ n * c(n) * (x^{\wedge} (n - Suc\ 0)))\ sums$
 $(\sum n. (diffs\ c)(n) * (x^{\wedge} n))$
 <proof>

lemma *lemma-termdiff1*:

fixes $z :: 'a :: \{recpower, comm-ring\}$ **shows**
 $(\sum_{p=0..<m} (((z + h)^{\wedge} (m - p)) * (z^{\wedge} p)) - (z^{\wedge} m)) =$
 $(\sum_{p=0..<m} (z^{\wedge} p) * (((z + h)^{\wedge} (m - p)) - (z^{\wedge} (m - p))))$
 <proof>

lemma *sumr-diff-mult-const2*:

$setsum\ f\ \{0..<n\} - of_nat\ n * (r::'a::ring-1) = (\sum i = 0..<n. f\ i - r)$

⟨proof⟩

lemma *lemma-termdiff2*:

fixes $h :: 'a :: \{\text{recpower}, \text{field}\}$

assumes $h: h \neq 0$ **shows**

$((z + h) ^ n - z ^ n) / h - \text{of-nat } n * z ^ (n - \text{Suc } 0) =$
 $h * (\sum p=0..< n - \text{Suc } 0. \sum q=0..< n - \text{Suc } 0 - p.$
 $(z + h) ^ q * z ^ (n - 2 - q))$ **(is ?lhs = ?rhs)**

⟨proof⟩

lemma *real-setsum-nat-ivl-bounded2*:

fixes $K :: 'a::\text{ordered-semidom}$

assumes $f: \bigwedge p::\text{nat}. p < n \implies f\ p \leq K$

assumes $K: 0 \leq K$

shows $\text{setsum } f \ \{0..<n-k\} \leq \text{of-nat } n * K$

⟨proof⟩

lemma *lemma-termdiff3*:

fixes $h\ z :: 'a::\{\text{real-normed-field}, \text{recpower}\}$

assumes $1: h \neq 0$

assumes $2: \text{norm } z \leq K$

assumes $3: \text{norm } (z + h) \leq K$

shows $\text{norm } (((z + h) ^ n - z ^ n) / h - \text{of-nat } n * z ^ (n - \text{Suc } 0))$
 $\leq \text{of-nat } n * \text{of-nat } (n - \text{Suc } 0) * K ^ (n - 2) * \text{norm } h$

⟨proof⟩

lemma *lemma-termdiff4*:

fixes $f :: 'a::\{\text{real-normed-field}, \text{recpower}\} \Rightarrow$

$'b::\text{real-normed-vector}$

assumes $k: 0 < (k::\text{real})$

assumes $le: \bigwedge h. \llbracket h \neq 0; \text{norm } h < k \rrbracket \implies \text{norm } (f\ h) \leq K * \text{norm } h$

shows $f \dashrightarrow 0 \dashrightarrow 0$

⟨proof⟩

lemma *lemma-termdiff5*:

fixes $g :: 'a::\{\text{recpower}, \text{real-normed-field}\} \Rightarrow$

$\text{nat} \Rightarrow 'b::\text{banach}$

assumes $k: 0 < (k::\text{real})$

assumes $f: \text{summable } f$

assumes $le: \bigwedge h\ n. \llbracket h \neq 0; \text{norm } h < k \rrbracket \implies \text{norm } (g\ h\ n) \leq f\ n * \text{norm } h$

shows $(\lambda h. \text{suminf } (g\ h)) \dashrightarrow 0 \dashrightarrow 0$

⟨proof⟩

FIXME: Long proofs

lemma *termdiffs-aux*:

fixes $x :: 'a::\{\text{recpower}, \text{real-normed-field}, \text{banach}\}$

assumes $1: \text{summable } (\lambda n. \text{diffs } (\text{diffs } c)\ n * K ^ n)$

assumes $2: \text{norm } x < \text{norm } K$

shows $(\lambda h. \sum n. c\ n * (((x + h) ^ n - x ^ n) / h$

— of-nat $n * x \wedge (n - \text{Suc } 0))) \text{---} 0 \text{---} > 0$
 <proof>

lemma *termdiffs*:

fixes $K\ x :: 'a :: \{\text{recpower}, \text{real-normed-field}, \text{banach}\}$
assumes 1: *summable* $(\lambda n. c\ n * K \wedge n)$
assumes 2: *summable* $(\lambda n. (\text{diffs } c)\ n * K \wedge n)$
assumes 3: *summable* $(\lambda n. (\text{diffs } (\text{diffs } c))\ n * K \wedge n)$
assumes 4: *norm* $x < \text{norm } K$
shows *DERIV* $(\lambda x. \sum n. c\ n * x \wedge n)\ x :> (\sum n. (\text{diffs } c)\ n * x \wedge n)$
 <proof>

63.4 Some properties of factorials

lemma *real-of-nat-fact-not-zero* [simp]: *real* (fact n) $\neq 0$
 <proof>

lemma *real-of-nat-fact-gt-zero* [simp]: $0 < \text{real}(\text{fact } n)$
 <proof>

lemma *real-of-nat-fact-ge-zero* [simp]: $0 \leq \text{real}(\text{fact } n)$
 <proof>

lemma *inv-real-of-nat-fact-gt-zero* [simp]: $0 < \text{inverse } (\text{real } (\text{fact } n))$
 <proof>

lemma *inv-real-of-nat-fact-ge-zero* [simp]: $0 \leq \text{inverse } (\text{real } (\text{fact } n))$
 <proof>

63.5 Derivability of power series

lemma *DERIV-series'*: **fixes** $f :: \text{real} \Rightarrow \text{nat} \Rightarrow \text{real}$
assumes *DERIV-f*: $\bigwedge n. \text{DERIV } (\lambda x. f\ x\ n)\ x0 :> (f'\ x0\ n)$
and *allf-summable*: $\bigwedge x. x \in \{a <..< b\} \implies \text{summable } (f\ x)$ **and** *x0-in-I*: $x0 \in \{a <..< b\}$
and *summable* $(f'\ x0)$
and *summable L* **and** *L-def*: $\bigwedge n\ x\ y. \llbracket x \in \{a <..< b\} ; y \in \{a <..< b\} \rrbracket$
 $\implies |f\ x\ n - f\ y\ n| \leq L\ n * |x - y|$
shows *DERIV* $(\lambda x. \text{suminf } (f\ x))\ x0 :> (\text{suminf } (f'\ x0))$
 <proof>

lemma *DERIV-power-series'*: **fixes** $f :: \text{nat} \Rightarrow \text{real}$
assumes *converges*: $\bigwedge x. x \in \{-R <..< R\} \implies \text{summable } (\lambda n. f\ n * \text{real } (\text{Suc } n) * x \wedge n)$
and *x0-in-I*: $x0 \in \{-R <..< R\}$ **and** $0 < R$
shows *DERIV* $(\lambda x. (\sum n. f\ n * x \wedge (\text{Suc } n)))\ x0 :> (\sum n. f\ n * \text{real } (\text{Suc } n) * x0 \wedge n)$
 (is *DERIV* $(\lambda x. (\text{suminf } (?f\ x)))\ x0 :> (\text{suminf } (?f'\ x0))$)
 <proof>

63.6 Exponential Function

definition

$exp :: 'a \Rightarrow 'a::\{recpower, real-normed-field, banach\}$ **where**
 $exp\ x = (\sum n. x ^ n /_R real\ (fact\ n))$

lemma *summable-exp-generic*:

fixes $x :: 'a::\{real-normed-algebra-1, recpower, banach\}$
defines $S\text{-def}$: $S \equiv \lambda n. x ^ n /_R real\ (fact\ n)$

shows *summable* S

<proof>

lemma *summable-norm-exp*:

fixes $x :: 'a::\{real-normed-algebra-1, recpower, banach\}$
shows *summable* $(\lambda n. norm\ (x ^ n /_R real\ (fact\ n)))$

<proof>

lemma *summable-exp*: *summable* $(\%n. inverse\ (real\ (fact\ n)) * x ^ n)$

<proof>

lemma *exp-converges*: $(\lambda n. x ^ n /_R real\ (fact\ n))\ sums\ exp\ x$

<proof>

lemma *exp-diffs*:

$diffs\ (\%n. inverse(real\ (fact\ n))) = (\%n. inverse(real\ (fact\ n)))$

<proof>

lemma *diffs-of-real*: $diffs\ (\lambda n. of\text{-}real\ (f\ n)) = (\lambda n. of\text{-}real\ (difs\ f\ n))$

<proof>

lemma *lemma-exp-ext*: $exp = (\lambda x. \sum n. x ^ n /_R real\ (fact\ n))$

<proof>

lemma *DERIV-exp [simp]*: $DERIV\ exp\ x :> exp(x)$

<proof>

lemma *isCont-exp [simp]*: *isCont* $exp\ x$

<proof>

63.6.1 Properties of the Exponential Function

lemma *power-zero*:

fixes $f :: nat \Rightarrow 'a::\{real-normed-algebra-1, recpower\}$
shows $(\sum n. f\ n * 0 ^ n) = f\ 0$

<proof>

lemma *exp-zero [simp]*: $exp\ 0 = 1$

<proof>

lemma *setsum-cl-ivl-Suc2*:

$(\sum_{i=m..Suc\ n}. f\ i) = (if\ Suc\ n < m\ then\ 0\ else\ f\ m + (\sum_{i=m..n}. f\ (Suc\ i)))$
 $\langle proof \rangle$

lemma *exp-series-add*:

fixes $x\ y :: 'a::\{real-field,recpower\}$
defines $S\text{-def}$: $S \equiv \lambda x\ n. x \wedge n /_R\ real\ (fact\ n)$
shows $S\ (x + y)\ n = (\sum_{i=0..n}. S\ x\ i * S\ y\ (n - i))$
 $\langle proof \rangle$

lemma *exp-add*: $exp\ (x + y) = exp\ x * exp\ y$
 $\langle proof \rangle$

lemma *mult-exp-exp*: $exp\ x * exp\ y = exp\ (x + y)$
 $\langle proof \rangle$

lemma *exp-of-real*: $exp\ (of\ real\ x) = of\ real\ (exp\ x)$
 $\langle proof \rangle$

lemma *exp-not-eq-zero* [simp]: $exp\ x \neq 0$
 $\langle proof \rangle$

lemma *exp-minus*: $exp\ (-\ x) = inverse\ (exp\ x)$
 $\langle proof \rangle$

lemma *exp-diff*: $exp\ (x - y) = exp\ x / exp\ y$
 $\langle proof \rangle$

63.6.2 Properties of the Exponential Function on Reals

Comparisons of $exp\ x$ with zero.

Proof: because every exponential can be seen as a square.

lemma *exp-ge-zero* [simp]: $0 \leq exp\ (x::real)$
 $\langle proof \rangle$

lemma *exp-gt-zero* [simp]: $0 < exp\ (x::real)$
 $\langle proof \rangle$

lemma *not-exp-less-zero* [simp]: $\neg exp\ (x::real) < 0$
 $\langle proof \rangle$

lemma *not-exp-le-zero* [simp]: $\neg exp\ (x::real) \leq 0$
 $\langle proof \rangle$

lemma *abs-exp-cancel* [simp]: $|exp\ x::real| = exp\ x$
 $\langle proof \rangle$

lemma *exp-real-of-nat-mult*: $exp(real\ n * x) = exp(x) \wedge n$

$\langle proof \rangle$

Strict monotonicity of exponential.

lemma *exp-ge-add-one-self-aux*: $0 \leq (x::real) \implies (1 + x) \leq \exp(x)$
 $\langle proof \rangle$

lemma *exp-gt-one*: $0 < (x::real) \implies 1 < \exp x$
 $\langle proof \rangle$

lemma *exp-less-mono*:
fixes $x\ y :: real$
assumes $x < y$ **shows** $\exp x < \exp y$
 $\langle proof \rangle$

lemma *exp-less-cancel*: $\exp (x::real) < \exp y \implies x < y$
 $\langle proof \rangle$

lemma *exp-less-cancel-iff* [iff]: $\exp (x::real) < \exp y \longleftrightarrow x < y$
 $\langle proof \rangle$

lemma *exp-le-cancel-iff* [iff]: $\exp (x::real) \leq \exp y \longleftrightarrow x \leq y$
 $\langle proof \rangle$

lemma *exp-inj-iff* [iff]: $\exp (x::real) = \exp y \longleftrightarrow x = y$
 $\langle proof \rangle$

Comparisons of $\exp x$ with one.

lemma *one-less-exp-iff* [simp]: $1 < \exp (x::real) \longleftrightarrow 0 < x$
 $\langle proof \rangle$

lemma *exp-less-one-iff* [simp]: $\exp (x::real) < 1 \longleftrightarrow x < 0$
 $\langle proof \rangle$

lemma *one-le-exp-iff* [simp]: $1 \leq \exp (x::real) \longleftrightarrow 0 \leq x$
 $\langle proof \rangle$

lemma *exp-le-one-iff* [simp]: $\exp (x::real) \leq 1 \longleftrightarrow x \leq 0$
 $\langle proof \rangle$

lemma *exp-eq-one-iff* [simp]: $\exp (x::real) = 1 \longleftrightarrow x = 0$
 $\langle proof \rangle$

lemma *lemma-exp-total*: $1 \leq y \implies \exists x. 0 \leq x \ \& \ x \leq y - 1 \ \& \ \exp(x::real) = y$
 $\langle proof \rangle$

lemma *exp-total*: $0 < (y::real) \implies \exists x. \exp x = y$
 $\langle proof \rangle$

63.7 Natural Logarithm

definition

$\ln :: \text{real} \Rightarrow \text{real}$ **where**
 $\ln x = (\text{THE } u. \exp u = x)$

lemma *ln-exp* [simp]: $\ln (\exp x) = x$
 $\langle \text{proof} \rangle$

lemma *exp-ln* [simp]: $0 < x \Longrightarrow \exp (\ln x) = x$
 $\langle \text{proof} \rangle$

lemma *exp-ln-iff* [simp]: $\exp (\ln x) = x \longleftrightarrow 0 < x$
 $\langle \text{proof} \rangle$

lemma *ln-unique*: $\exp y = x \Longrightarrow \ln x = y$
 $\langle \text{proof} \rangle$

lemma *ln-one* [simp]: $\ln 1 = 0$
 $\langle \text{proof} \rangle$

lemma *ln-mult*: $\llbracket 0 < x; 0 < y \rrbracket \Longrightarrow \ln (x * y) = \ln x + \ln y$
 $\langle \text{proof} \rangle$

lemma *ln-inverse*: $0 < x \Longrightarrow \ln (\text{inverse } x) = - \ln x$
 $\langle \text{proof} \rangle$

lemma *ln-div*: $\llbracket 0 < x; 0 < y \rrbracket \Longrightarrow \ln (x / y) = \ln x - \ln y$
 $\langle \text{proof} \rangle$

lemma *ln-realpow*: $0 < x \Longrightarrow \ln (x ^ n) = \text{real } n * \ln x$
 $\langle \text{proof} \rangle$

lemma *ln-less-cancel-iff* [simp]: $\llbracket 0 < x; 0 < y \rrbracket \Longrightarrow \ln x < \ln y \longleftrightarrow x < y$
 $\langle \text{proof} \rangle$

lemma *ln-le-cancel-iff* [simp]: $\llbracket 0 < x; 0 < y \rrbracket \Longrightarrow \ln x \leq \ln y \longleftrightarrow x \leq y$
 $\langle \text{proof} \rangle$

lemma *ln-inj-iff* [simp]: $\llbracket 0 < x; 0 < y \rrbracket \Longrightarrow \ln x = \ln y \longleftrightarrow x = y$
 $\langle \text{proof} \rangle$

lemma *ln-add-one-self-le-self* [simp]: $0 \leq x \Longrightarrow \ln (1 + x) \leq x$
 $\langle \text{proof} \rangle$

lemma *ln-less-self* [simp]: $0 < x \Longrightarrow \ln x < x$
 $\langle \text{proof} \rangle$

lemma *ln-ge-zero* [simp]:
assumes $x: 1 \leq x$ **shows** $0 \leq \ln x$

$\langle \text{proof} \rangle$

lemma *ln-ge-zero-imp-ge-one*:

assumes *ln*: $0 \leq \ln x$

and *x*: $0 < x$

shows $1 \leq x$

$\langle \text{proof} \rangle$

lemma *ln-ge-zero-iff [simp]*: $0 < x \implies (0 \leq \ln x) = (1 \leq x)$

$\langle \text{proof} \rangle$

lemma *ln-less-zero-iff [simp]*: $0 < x \implies (\ln x < 0) = (x < 1)$

$\langle \text{proof} \rangle$

lemma *ln-gt-zero*:

assumes *x*: $1 < x$ **shows** $0 < \ln x$

$\langle \text{proof} \rangle$

lemma *ln-gt-zero-imp-gt-one*:

assumes *ln*: $0 < \ln x$

and *x*: $0 < x$

shows $1 < x$

$\langle \text{proof} \rangle$

lemma *ln-gt-zero-iff [simp]*: $0 < x \implies (0 < \ln x) = (1 < x)$

$\langle \text{proof} \rangle$

lemma *ln-eq-zero-iff [simp]*: $0 < x \implies (\ln x = 0) = (x = 1)$

$\langle \text{proof} \rangle$

lemma *ln-less-zero*: $[[0 < x; x < 1]] \implies \ln x < 0$

$\langle \text{proof} \rangle$

lemma *exp-ln-eq*: $\exp u = x \implies \ln x = u$

$\langle \text{proof} \rangle$

lemma *isCont-ln*: $0 < x \implies \text{isCont } \ln x$

$\langle \text{proof} \rangle$

lemma *DERIV-ln*: $0 < x \implies \text{DERIV } \ln x :> \text{inverse } x$

$\langle \text{proof} \rangle$

lemma *ln-series*: **assumes** $0 < x$ **and** $x < 2$

shows $\ln x = \left(\sum n. (-1)^n * (1 / \text{real } (n + 1)) * (x - 1)^{(Suc\ n)} \right)$ (**is** $\ln x$
 $= \text{suminf } (?f\ (x - 1)))$

$\langle \text{proof} \rangle$

63.8 Sine and Cosine

definition

$\sin :: \text{real} \Rightarrow \text{real}$ **where**
 $\sin x = (\sum n. (\text{if even}(n) \text{ then } 0 \text{ else } (-1 \wedge ((n - \text{Suc } 0) \text{ div } 2)) / (\text{real } (\text{fact } n)))) * x \wedge n)$

definition

$\cos :: \text{real} \Rightarrow \text{real}$ **where**
 $\cos x = (\sum n. (\text{if even}(n) \text{ then } (-1 \wedge (n \text{ div } 2)) / (\text{real } (\text{fact } n)) \text{ else } 0) * x \wedge n)$

lemma *summable-sin*:

$\text{summable } (\%n. (\text{if even } n \text{ then } 0 \text{ else } -1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n)))) * x \wedge n)$

$\langle \text{proof} \rangle$

lemma *summable-cos*:

$\text{summable } (\%n. (\text{if even } n \text{ then } -1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n)) \text{ else } 0) * x \wedge n)$

$\langle \text{proof} \rangle$

lemma *lemma-STAR-sin*:

$(\text{if even } n \text{ then } 0 \text{ else } -1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))) * 0 \wedge n = 0$

$\langle \text{proof} \rangle$

lemma *lemma-STAR-cos*:

$0 < n \longrightarrow -1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n)) * 0 \wedge n = 0$

$\langle \text{proof} \rangle$

lemma *lemma-STAR-cos1*:

$0 < n \longrightarrow (-1) \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n)) * 0 \wedge n = 0$

$\langle \text{proof} \rangle$

lemma *lemma-STAR-cos2*:

$(\sum n=1..<n. \text{if even } n \text{ then } -1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n)) * 0 \wedge n \text{ else } 0) = 0$

$\langle \text{proof} \rangle$

lemma *sin-converges*:

$(\%n. (\text{if even } n \text{ then } 0 \text{ else } -1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n)))) * x \wedge n \text{ sums } \sin(x)$

$\langle \text{proof} \rangle$

lemma *cos-converges*:

(%n. (if even n then
 $-1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n))$
 else 0) * x ^ n) sums cos(x)
 <proof>

lemma *sin-fdiffs*:

diffs(%n. if even n then 0
 else $-1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))$)
 = (%n. if even n then
 $-1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n))$
 else 0)
 <proof>

lemma *sin-fdiffs2*:

diffs(%n. if even n then 0
 else $-1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))$) n
 = (if even n then
 $-1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n))$
 else 0)
 <proof>

lemma *cos-fdiffs*:

diffs(%n. if even n then
 $-1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n))$ else 0)
 = (%n. - (if even n then 0
 else $-1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))$))
 <proof>

lemma *cos-fdiffs2*:

diffs(%n. if even n then
 $-1 \wedge (n \text{ div } 2) / (\text{real } (\text{fact } n))$ else 0) n
 = - (if even n then 0
 else $-1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))$)
 <proof>

Now at last we can get the derivatives of exp, sin and cos

lemma *lemma-sin-minus*:

- sin x = ($\sum n.$ - ((if even n then 0
 else $-1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))$)) * x ^ n)
 <proof>

lemma *lemma-sin-ext*:

sin = (%x. $\sum n.$
 (if even n then 0
 else $-1 \wedge ((n - \text{Suc } 0) \text{ div } 2) / (\text{real } (\text{fact } n))$) *
 x ^ n)

$\langle proof \rangle$

lemma *lemma-cos-ext*:

$$\cos = (\%x. \sum n. \\ \text{(if even } n \text{ then } -1 \wedge (n \text{ div } 2) / (\text{real (fact } n)) \text{ else } 0) * \\ x \wedge n)$$

$\langle proof \rangle$

lemma *DERIV-sin [simp]*: $DERIV \sin x :> \cos(x)$

$\langle proof \rangle$

lemma *DERIV-cos [simp]*: $DERIV \cos x :> -\sin(x)$

$\langle proof \rangle$

lemma *isCont-sin [simp]*: $isCont \sin x$

$\langle proof \rangle$

lemma *isCont-cos [simp]*: $isCont \cos x$

$\langle proof \rangle$

63.9 Properties of Sine and Cosine

lemma *sin-zero [simp]*: $\sin 0 = 0$

$\langle proof \rangle$

lemma *cos-zero [simp]*: $\cos 0 = 1$

$\langle proof \rangle$

lemma *DERIV-sin-sin-mult [simp]*:

$$DERIV (\%x. \sin(x) * \sin(x)) x :> \cos(x) * \sin(x) + \cos(x) * \sin(x)$$

$\langle proof \rangle$

lemma *DERIV-sin-sin-mult2 [simp]*:

$$DERIV (\%x. \sin(x) * \sin(x)) x :> 2 * \cos(x) * \sin(x)$$

$\langle proof \rangle$

lemma *DERIV-sin-realpow2 [simp]*:

$$DERIV (\%x. (\sin x)^2) x :> \cos(x) * \sin(x) + \cos(x) * \sin(x)$$

$\langle proof \rangle$

lemma *DERIV-sin-realpow2a [simp]*:

$$DERIV (\%x. (\sin x)^2) x :> 2 * \cos(x) * \sin(x)$$

$\langle proof \rangle$

lemma *DERIV-cos-cos-mult [simp]*:

$$DERIV (\%x. \cos(x) * \cos(x)) x :> -\sin(x) * \cos(x) + -\sin(x) * \cos(x)$$

$\langle proof \rangle$

lemma *DERIV-cos-cos-mult2 [simp]*:

$DERIV (\%x. \cos(x)*\cos(x)) \ x :> -2 * \cos(x) * \sin(x)$
 $\langle proof \rangle$

lemma *DERIV-cos-realpow2* [simp]:
 $DERIV (\%x. (\cos x)^2) \ x :> -\sin(x) * \cos(x) + -\sin(x) * \cos(x)$
 $\langle proof \rangle$

lemma *DERIV-cos-realpow2a* [simp]:
 $DERIV (\%x. (\cos x)^2) \ x :> -2 * \cos(x) * \sin(x)$
 $\langle proof \rangle$

lemma *lemma-DERIV-subst*: $[| \ DERIV \ f \ x :> D; \ D = E \ |] ==> \ DERIV \ f \ x :> E$
 $\langle proof \rangle$

lemma *DERIV-cos-realpow2b*: $DERIV (\%x. (\cos x)^2) \ x :> -(2 * \cos(x) * \sin(x))$
 $\langle proof \rangle$

lemma *DERIV-cos-cos-mult3* [simp]:
 $DERIV (\%x. \cos(x)*\cos(x)) \ x :> -(2 * \cos(x) * \sin(x))$
 $\langle proof \rangle$

lemma *DERIV-sin-circle-all*:
 $\forall x. \ DERIV (\%x. (\sin x)^2 + (\cos x)^2) \ x :>$
 $(2*\cos(x)*\sin(x) - 2*\cos(x)*\sin(x))$
 $\langle proof \rangle$

lemma *DERIV-sin-circle-all-zero* [simp]:
 $\forall x. \ DERIV (\%x. (\sin x)^2 + (\cos x)^2) \ x :> 0$
 $\langle proof \rangle$

lemma *sin-cos-squared-add* [simp]: $((\sin x)^2) + ((\cos x)^2) = 1$
 $\langle proof \rangle$

lemma *sin-cos-squared-add2* [simp]: $((\cos x)^2) + ((\sin x)^2) = 1$
 $\langle proof \rangle$

lemma *sin-cos-squared-add3* [simp]: $\cos x * \cos x + \sin x * \sin x = 1$
 $\langle proof \rangle$

lemma *sin-squared-eq*: $(\sin x)^2 = 1 - (\cos x)^2$
 $\langle proof \rangle$

lemma *cos-squared-eq*: $(\cos x)^2 = 1 - (\sin x)^2$
 $\langle proof \rangle$

lemma *abs-sin-le-one* [simp]: $|\sin x| \leq 1$
 $\langle proof \rangle$

lemma *sin-ge-minus-one* [simp]: $-1 \leq \sin x$
 ⟨proof⟩

lemma *sin-le-one* [simp]: $\sin x \leq 1$
 ⟨proof⟩

lemma *abs-cos-le-one* [simp]: $|\cos x| \leq 1$
 ⟨proof⟩

lemma *cos-ge-minus-one* [simp]: $-1 \leq \cos x$
 ⟨proof⟩

lemma *cos-le-one* [simp]: $\cos x \leq 1$
 ⟨proof⟩

lemma *DERIV-fun-pow*: $DERIV\ g\ x\ :\>\ m\ ==>$
 $DERIV\ (\%x.\ (g\ x)\ ^\ n)\ x\ :\>\ real\ n\ * (g\ x)\ ^\ (n - 1) * m$
 ⟨proof⟩

lemma *DERIV-fun-exp*:
 $DERIV\ g\ x\ :\>\ m\ ==>\ DERIV\ (\%x.\ exp(g\ x))\ x\ :\>\ exp(g\ x) * m$
 ⟨proof⟩

lemma *DERIV-fun-sin*:
 $DERIV\ g\ x\ :\>\ m\ ==>\ DERIV\ (\%x.\ sin(g\ x))\ x\ :\>\ cos(g\ x) * m$
 ⟨proof⟩

lemma *DERIV-fun-cos*:
 $DERIV\ g\ x\ :\>\ m\ ==>\ DERIV\ (\%x.\ cos(g\ x))\ x\ :\>\ -sin(g\ x) * m$
 ⟨proof⟩

lemmas *DERIV-intros* = *DERIV-ident* *DERIV-const* *DERIV-cos* *DERIV-cmult*
DERIV-sin *DERIV-exp* *DERIV-inverse* *DERIV-pow*
DERIV-add *DERIV-diff* *DERIV-mult* *DERIV-minus*
DERIV-inverse-fun *DERIV-quotient* *DERIV-fun-pow*
DERIV-fun-exp *DERIV-fun-sin* *DERIV-fun-cos*

lemma *lemma-DERIV-sin-cos-add*:
 $\forall x.$
 $DERIV\ (\%x.\ (sin\ (x + y) - (sin\ x * cos\ y + cos\ x * sin\ y))\ ^\ 2 +$
 $(cos\ (x + y) - (cos\ x * cos\ y - sin\ x * sin\ y))\ ^\ 2)\ x\ :\>\ 0$
 ⟨proof⟩

lemma *sin-cos-add* [simp]:
 $(sin\ (x + y) - (sin\ x * cos\ y + cos\ x * sin\ y))\ ^\ 2 +$
 $(cos\ (x + y) - (cos\ x * cos\ y - sin\ x * sin\ y))\ ^\ 2 = 0$
 ⟨proof⟩

lemma *sin-add*: $\sin (x + y) = \sin x * \cos y + \cos x * \sin y$
 $\langle \text{proof} \rangle$

lemma *cos-add*: $\cos (x + y) = \cos x * \cos y - \sin x * \sin y$
 $\langle \text{proof} \rangle$

lemma *lemma-DERIV-sin-cos-minus*:
 $\forall x. \text{DERIV } (\%x. (\sin(-x) + (\sin x)) ^ 2 + (\cos(-x) - (\cos x)) ^ 2) x :> 0$
 $\langle \text{proof} \rangle$

lemma *sin-cos-minus*:
 $(\sin(-x) + (\sin x)) ^ 2 + (\cos(-x) - (\cos x)) ^ 2 = 0$
 $\langle \text{proof} \rangle$

lemma *sin-minus* [*simp*]: $\sin (-x) = -\sin(x)$
 $\langle \text{proof} \rangle$

lemma *cos-minus* [*simp*]: $\cos (-x) = \cos(x)$
 $\langle \text{proof} \rangle$

lemma *sin-diff*: $\sin (x - y) = \sin x * \cos y - \cos x * \sin y$
 $\langle \text{proof} \rangle$

lemma *sin-diff2*: $\sin (x - y) = \cos y * \sin x - \sin y * \cos x$
 $\langle \text{proof} \rangle$

lemma *cos-diff*: $\cos (x - y) = \cos x * \cos y + \sin x * \sin y$
 $\langle \text{proof} \rangle$

lemma *cos-diff2*: $\cos (x - y) = \cos y * \cos x + \sin y * \sin x$
 $\langle \text{proof} \rangle$

lemma *sin-double* [*simp*]: $\sin(2 * x) = 2 * \sin x * \cos x$
 $\langle \text{proof} \rangle$

lemma *cos-double*: $\cos(2 * x) = ((\cos x)^2) - ((\sin x)^2)$
 $\langle \text{proof} \rangle$

63.10 The Constant Pi

definition

$\pi :: \text{real}$ **where**
 $\pi = 2 * (\text{THE } x. 0 \leq (x::\text{real}) \ \& \ x \leq 2 \ \& \ \cos x = 0)$

Show that there's a least positive x with $\cos x = 0$; hence define π .

lemma *sin-paired*:
 $(\%n. -1 ^ n / (\text{real } (\text{fact } (2 * n + 1)))) * x ^ (2 * n + 1))$
 $\text{sums } \sin x$

<proof>

FIXME: This is a long, ugly proof!

lemma *sin-gt-zero*: $[|0 < x; x < 2|] ==> 0 < \sin x$
<proof>

lemma *sin-gt-zero1*: $[|0 < x; x < 2|] ==> 0 < \sin x$
<proof>

lemma *cos-double-less-one*: $[|0 < x; x < 2|] ==> \cos (2 * x) < 1$
<proof>

lemma *cos-paired*:
 $(\%n. -1 ^ n / (\text{real } (\text{fact } (2 * n))) * x ^ (2 * n)) \text{ sums } \cos x$
<proof>

lemma *fact-lemma*: $\text{real } (n::\text{nat}) * 4 = \text{real } (4 * n)$
<proof>

lemma *cos-two-less-zero* [simp]: $\cos (2) < 0$
<proof>

lemmas *cos-two-neq-zero* [simp] = *cos-two-less-zero* [THEN *less-imp-neq*]
lemmas *cos-two-le-zero* [simp] = *cos-two-less-zero* [THEN *order-less-imp-le*]

lemma *cos-is-zero*: *EX!* $x. 0 \leq x \ \& \ x \leq 2 \ \& \ \cos x = 0$
<proof>

lemma *pi-half*: $\pi/2 = (\text{THE } x. 0 \leq x \ \& \ x \leq 2 \ \& \ \cos x = 0)$
<proof>

lemma *cos-pi-half* [simp]: $\cos (\pi / 2) = 0$
<proof>

lemma *pi-half-gt-zero* [simp]: $0 < \pi / 2$
<proof>

lemmas *pi-half-neq-zero* [simp] = *pi-half-gt-zero* [THEN *less-imp-neq*, *symmetric*]
lemmas *pi-half-ge-zero* [simp] = *pi-half-gt-zero* [THEN *order-less-imp-le*]

lemma *pi-half-less-two* [simp]: $\pi / 2 < 2$
<proof>

lemmas *pi-half-neq-two* [simp] = *pi-half-less-two* [THEN *less-imp-neq*]
lemmas *pi-half-le-two* [simp] = *pi-half-less-two* [THEN *order-less-imp-le*]

lemma *pi-gt-zero* [simp]: $0 < \pi$
<proof>

lemma *pi-ge-zero* [*simp*]: $0 \leq \pi$
 $\langle \text{proof} \rangle$

lemma *pi-neq-zero* [*simp*]: $\pi \neq 0$
 $\langle \text{proof} \rangle$

lemma *pi-not-less-zero* [*simp*]: $\neg \pi < 0$
 $\langle \text{proof} \rangle$

lemma *minus-pi-half-less-zero*: $-(\pi/2) < 0$
 $\langle \text{proof} \rangle$

lemma *m2pi-less-pi*: $-(2 * \pi) < \pi$
 $\langle \text{proof} \rangle$

lemma *sin-pi-half* [*simp*]: $\sin(\pi/2) = 1$
 $\langle \text{proof} \rangle$

lemma *cos-pi* [*simp*]: $\cos \pi = -1$
 $\langle \text{proof} \rangle$

lemma *sin-pi* [*simp*]: $\sin \pi = 0$
 $\langle \text{proof} \rangle$

lemma *sin-cos-eq*: $\sin x = \cos (\pi/2 - x)$
 $\langle \text{proof} \rangle$

declare *sin-cos-eq* [*symmetric, simp*]

lemma *minus-sin-cos-eq*: $-\sin x = \cos (x + \pi/2)$
 $\langle \text{proof} \rangle$

declare *minus-sin-cos-eq* [*symmetric, simp*]

lemma *cos-sin-eq*: $\cos x = \sin (\pi/2 - x)$
 $\langle \text{proof} \rangle$

declare *cos-sin-eq* [*symmetric, simp*]

lemma *sin-periodic-pi* [*simp*]: $\sin (x + \pi) = -\sin x$
 $\langle \text{proof} \rangle$

lemma *sin-periodic-pi2* [*simp*]: $\sin (\pi + x) = -\sin x$
 $\langle \text{proof} \rangle$

lemma *cos-periodic-pi* [*simp*]: $\cos (x + \pi) = -\cos x$
 $\langle \text{proof} \rangle$

lemma *sin-periodic* [*simp*]: $\sin (x + 2*\pi) = \sin x$
 $\langle \text{proof} \rangle$

lemma *cos-periodic* [*simp*]: $\cos (x + 2*\pi) = \cos x$

$\langle \text{proof} \rangle$

lemma *cos-npi* [simp]: $\cos (\text{real } n * \pi) = -1 ^ n$
 $\langle \text{proof} \rangle$

lemma *cos-npi2* [simp]: $\cos (\pi * \text{real } n) = -1 ^ n$
 $\langle \text{proof} \rangle$

lemma *sin-npi* [simp]: $\sin (\text{real } (n::\text{nat}) * \pi) = 0$
 $\langle \text{proof} \rangle$

lemma *sin-npi2* [simp]: $\sin (\pi * \text{real } (n::\text{nat})) = 0$
 $\langle \text{proof} \rangle$

lemma *cos-two-pi* [simp]: $\cos (2 * \pi) = 1$
 $\langle \text{proof} \rangle$

lemma *sin-two-pi* [simp]: $\sin (2 * \pi) = 0$
 $\langle \text{proof} \rangle$

lemma *sin-gt-zero2*: $[| 0 < x; x < \pi/2 |] \implies 0 < \sin x$
 $\langle \text{proof} \rangle$

lemma *sin-less-zero*:
 assumes *lb*: $-\pi/2 < x$ and $x < 0$ shows $\sin x < 0$
 $\langle \text{proof} \rangle$

lemma *pi-less-4*: $\pi < 4$
 $\langle \text{proof} \rangle$

lemma *cos-gt-zero*: $[| 0 < x; x < \pi/2 |] \implies 0 < \cos x$
 $\langle \text{proof} \rangle$

lemma *cos-gt-zero-pi*: $[| -(\pi/2) < x; x < \pi/2 |] \implies 0 < \cos x$
 $\langle \text{proof} \rangle$

lemma *cos-ge-zero*: $[| -(\pi/2) \leq x; x \leq \pi/2 |] \implies 0 \leq \cos x$
 $\langle \text{proof} \rangle$

lemma *sin-gt-zero-pi*: $[| 0 < x; x < \pi |] \implies 0 < \sin x$
 $\langle \text{proof} \rangle$

lemma *pi-ge-two*: $2 \leq \pi$
 $\langle \text{proof} \rangle$

lemma *sin-ge-zero*: $[| 0 \leq x; x \leq \pi |] \implies 0 \leq \sin x$
 $\langle \text{proof} \rangle$

lemma *cos-total*: $[\![-1 \leq y; y \leq 1 \!]\!] \implies \text{EX! } x. 0 \leq x \ \& \ x \leq \text{pi} \ \& \ (\cos x = y)$
 $\langle \text{proof} \rangle$

lemma *sin-total*:
 $[\![-1 \leq y; y \leq 1 \!]\!] \implies \text{EX! } x. -(pi/2) \leq x \ \& \ x \leq pi/2 \ \& \ (\sin x = y)$
 $\langle \text{proof} \rangle$

lemma *reals-Archimedean4*:
 $[\![0 < y; 0 \leq x \!]\!] \implies \exists n. \text{real } n * y \leq x \ \& \ x < \text{real } (\text{Suc } n) * y$
 $\langle \text{proof} \rangle$

lemma *cos-zero-lemma*:
 $[\![0 \leq x; \cos x = 0 \!]\!] \implies$
 $\exists n::\text{nat}. \sim \text{even } n \ \& \ x = \text{real } n * (pi/2)$
 $\langle \text{proof} \rangle$

lemma *sin-zero-lemma*:
 $[\![0 \leq x; \sin x = 0 \!]\!] \implies$
 $\exists n::\text{nat}. \text{even } n \ \& \ x = \text{real } n * (pi/2)$
 $\langle \text{proof} \rangle$

lemma *cos-zero-iff*:
 $(\cos x = 0) =$
 $((\exists n::\text{nat}. \sim \text{even } n \ \& \ (x = \text{real } n * (pi/2))) \mid$
 $(\exists n::\text{nat}. \sim \text{even } n \ \& \ (x = -(\text{real } n * (pi/2))))$
 $\langle \text{proof} \rangle$

lemma *sin-zero-iff*:
 $(\sin x = 0) =$
 $((\exists n::\text{nat}. \text{even } n \ \& \ (x = \text{real } n * (pi/2))) \mid$
 $(\exists n::\text{nat}. \text{even } n \ \& \ (x = -(\text{real } n * (pi/2))))$
 $\langle \text{proof} \rangle$

lemma *cos-monotone-0-pi*: **assumes** $0 \leq y$ **and** $y < x$ **and** $x \leq pi$
shows $\cos x < \cos y$
 $\langle \text{proof} \rangle$

lemma *cos-monotone-0-pi'*: **assumes** $0 \leq y$ **and** $y \leq x$ **and** $x \leq pi$ **shows** $\cos x$
 $\leq \cos y$
 $\langle \text{proof} \rangle$

lemma *cos-monotone-minus-pi-0*: **assumes** $-pi \leq y$ **and** $y < x$ **and** $x \leq 0$
shows $\cos y < \cos x$
 $\langle \text{proof} \rangle$

lemma *cos-monotone-minus-pi-0'*: **assumes** $-pi \leq y$ **and** $y \leq x$ **and** $x \leq 0$
shows $\cos y \leq \cos x$
 $\langle proof \rangle$

lemma *sin-monotone-2pi'*: **assumes** $-(pi / 2) \leq y$ **and** $y \leq x$ **and** $x \leq pi / 2$
shows $\sin y \leq \sin x$
 $\langle proof \rangle$

63.11 Tangent

definition

$tan :: real \Rightarrow real$ **where**
 $tan\ x = (\sin\ x) / (\cos\ x)$

lemma *tan-zero* [simp]: $tan\ 0 = 0$
 $\langle proof \rangle$

lemma *tan-pi* [simp]: $tan\ pi = 0$
 $\langle proof \rangle$

lemma *tan-npi* [simp]: $tan\ (real\ (n::nat) * pi) = 0$
 $\langle proof \rangle$

lemma *tan-minus* [simp]: $tan\ (-x) = -\ tan\ x$
 $\langle proof \rangle$

lemma *tan-periodic* [simp]: $tan\ (x + 2*pi) = tan\ x$
 $\langle proof \rangle$

lemma *lemma-tan-add1*:
 $[| \cos\ x \neq 0; \cos\ y \neq 0 |]$
 $\implies 1 - tan(x)*tan(y) = \cos\ (x + y) / (\cos\ x * \cos\ y)$
 $\langle proof \rangle$

lemma *add-tan-eq*:
 $[| \cos\ x \neq 0; \cos\ y \neq 0 |]$
 $\implies tan\ x + tan\ y = \sin(x + y) / (\cos\ x * \cos\ y)$
 $\langle proof \rangle$

lemma *tan-add*:
 $[| \cos\ x \neq 0; \cos\ y \neq 0; \cos\ (x + y) \neq 0 |]$
 $\implies tan(x + y) = (tan(x) + tan(y)) / (1 - tan(x) * tan(y))$
 $\langle proof \rangle$

lemma *tan-double*:
 $[| \cos\ x \neq 0; \cos\ (2 * x) \neq 0 |]$
 $\implies tan\ (2 * x) = (2 * tan\ x) / (1 - (tan(x) ^ 2))$
 $\langle proof \rangle$

lemma *tan-gt-zero*: $[| 0 < x; x < \pi/2 |] \implies 0 < \tan x$
 $\langle \text{proof} \rangle$

lemma *tan-less-zero*:
assumes $lb: -\pi/2 < x$ **and** $x < 0$ **shows** $\tan x < 0$
 $\langle \text{proof} \rangle$

lemma *tan-half*: **fixes** $x :: \text{real}$ **assumes** $-(\pi/2) < x$ **and** $x < \pi/2$
shows $\tan x = \sin(2 * x) / (\cos(2 * x) + 1)$
 $\langle \text{proof} \rangle$

lemma *lemma-DERIV-tan*:
 $\cos x \neq 0 \implies \text{DERIV } (\%x. \sin(x)/\cos(x)) \ x :> \text{inverse}((\cos x)^2)$
 $\langle \text{proof} \rangle$

lemma *DERIV-tan [simp]*: $\cos x \neq 0 \implies \text{DERIV } \tan x :> \text{inverse}((\cos x)^2)$
 $\langle \text{proof} \rangle$

lemma *isCont-tan [simp]*: $\cos x \neq 0 \implies \text{isCont } \tan x$
 $\langle \text{proof} \rangle$

lemma *LIM-cos-div-sin [simp]*: $(\%x. \cos(x)/\sin(x)) \ -- \ \pi/2 \ --> 0$
 $\langle \text{proof} \rangle$

lemma *lemma-tan-total*: $0 < y \implies \exists x. 0 < x \ \& \ x < \pi/2 \ \& \ y < \tan x$
 $\langle \text{proof} \rangle$

lemma *tan-total-pos*: $0 \leq y \implies \exists x. 0 \leq x \ \& \ x < \pi/2 \ \& \ \tan x = y$
 $\langle \text{proof} \rangle$

lemma *lemma-tan-total1*: $\exists x. -(\pi/2) < x \ \& \ x < (\pi/2) \ \& \ \tan x = y$
 $\langle \text{proof} \rangle$

lemma *tan-total*: $\text{EX! } x. -(\pi/2) < x \ \& \ x < (\pi/2) \ \& \ \tan x = y$
 $\langle \text{proof} \rangle$

lemma *tan-monotone*: **assumes** $-(\pi/2) < y$ **and** $y < x$ **and** $x < \pi/2$
shows $\tan y < \tan x$
 $\langle \text{proof} \rangle$

lemma *tan-monotone'*: **assumes** $-(\pi/2) < y$ **and** $y < \pi/2$ **and** $-(\pi/2) < x$ **and** $x < \pi/2$
shows $(y < x) = (\tan y < \tan x)$
 $\langle \text{proof} \rangle$

lemma *tan-inverse*: $1 / (\tan y) = \tan(\pi/2 - y)$ $\langle \text{proof} \rangle$

lemma *tan-periodic-pi[simp]*: $\tan(x + \pi) = \tan x$
 $\langle \text{proof} \rangle$

lemma *tan-periodic-nat*[simp]: **fixes** $n :: \text{nat}$ **shows** $\tan (x + \text{real } n * \pi) = \tan x$
 $\langle \text{proof} \rangle$

lemma *tan-periodic-int*[simp]: **fixes** $i :: \text{int}$ **shows** $\tan (x + \text{real } i * \pi) = \tan x$
 $\langle \text{proof} \rangle$

lemma *tan-periodic-n*[simp]: $\tan (x + \text{number-of } n * \pi) = \tan x$
 $\langle \text{proof} \rangle$

63.12 Inverse Trigonometric Functions

definition

$\arcsin :: \text{real} \Rightarrow \text{real}$ **where**
 $\arcsin y = (\text{THE } x. -(pi/2) \leq x \ \& \ x \leq pi/2 \ \& \ \sin x = y)$

definition

$\arccos :: \text{real} \Rightarrow \text{real}$ **where**
 $\arccos y = (\text{THE } x. 0 \leq x \ \& \ x \leq pi \ \& \ \cos x = y)$

definition

$\arctan :: \text{real} \Rightarrow \text{real}$ **where**
 $\arctan y = (\text{THE } x. -(pi/2) < x \ \& \ x < pi/2 \ \& \ \tan x = y)$

lemma

arcsin:
 $[-1 \leq y; y \leq 1] \implies -(pi/2) \leq \arcsin y \ \& \ \arcsin y \leq pi/2 \ \& \ \sin(\arcsin y) = y$
 $\langle \text{proof} \rangle$

lemma

arcsin-pi:
 $[-1 \leq y; y \leq 1] \implies -(pi/2) \leq \arcsin y \ \& \ \arcsin y \leq pi \ \& \ \sin(\arcsin y) = y$
 $\langle \text{proof} \rangle$

lemma *sin-arcsin* [simp]: $[-1 \leq y; y \leq 1] \implies \sin(\arcsin y) = y$
 $\langle \text{proof} \rangle$

lemma

arcsin-bounded:
 $[-1 \leq y; y \leq 1] \implies -(pi/2) \leq \arcsin y \ \& \ \arcsin y \leq pi/2$
 $\langle \text{proof} \rangle$

lemma *arcsin-lbound*: $[-1 \leq y; y \leq 1] \implies -(pi/2) \leq \arcsin y$
 $\langle \text{proof} \rangle$

lemma *arcsin-ubound*: $[-1 \leq y; y \leq 1] \implies \arcsin y \leq pi/2$
 $\langle \text{proof} \rangle$

lemma *arcsin-lt-bounded*:

$\llbracket -1 < y; y < 1 \rrbracket \implies -(pi/2) < \arcsin y \ \& \ \arcsin y < pi/2$
 $\langle proof \rangle$

lemma *arcsin-sin*: $\llbracket -(pi/2) \leq x; x \leq pi/2 \rrbracket \implies \arcsin(\sin x) = x$
 $\langle proof \rangle$

lemma *arccos*:

$\llbracket -1 \leq y; y \leq 1 \rrbracket$
 $\implies 0 \leq \arccos y \ \& \ \arccos y \leq pi \ \& \ \cos(\arccos y) = y$
 $\langle proof \rangle$

lemma *cos-arccos [simp]*: $\llbracket -1 \leq y; y \leq 1 \rrbracket \implies \cos(\arccos y) = y$
 $\langle proof \rangle$

lemma *arccos-bounded*: $\llbracket -1 \leq y; y \leq 1 \rrbracket \implies 0 \leq \arccos y \ \& \ \arccos y \leq pi$
 $\langle proof \rangle$

lemma *arccos-lbound*: $\llbracket -1 \leq y; y \leq 1 \rrbracket \implies 0 \leq \arccos y$
 $\langle proof \rangle$

lemma *arccos-ubound*: $\llbracket -1 \leq y; y \leq 1 \rrbracket \implies \arccos y \leq pi$
 $\langle proof \rangle$

lemma *arccos-lt-bounded*:

$\llbracket -1 < y; y < 1 \rrbracket$
 $\implies 0 < \arccos y \ \& \ \arccos y < pi$
 $\langle proof \rangle$

lemma *arccos-cos*: $\llbracket 0 \leq x; x \leq pi \rrbracket \implies \arccos(\cos x) = x$
 $\langle proof \rangle$

lemma *arccos-cos2*: $\llbracket x \leq 0; -pi \leq x \rrbracket \implies \arccos(\cos x) = -x$
 $\langle proof \rangle$

lemma *cos-arcsin*: $\llbracket -1 \leq x; x \leq 1 \rrbracket \implies \cos(\arcsin x) = \text{sqrt}(1 - x^2)$
 $\langle proof \rangle$

lemma *sin-arccos*: $\llbracket -1 \leq x; x \leq 1 \rrbracket \implies \sin(\arccos x) = \text{sqrt}(1 - x^2)$
 $\langle proof \rangle$

lemma *arctan [simp]*:

$-(pi/2) < \arctan y \ \& \ \arctan y < pi/2 \ \& \ \tan(\arctan y) = y$
 $\langle proof \rangle$

lemma *tan-arctan*: $\tan(\arctan y) = y$
 $\langle proof \rangle$

lemma *arctan-bounded*: $-(pi/2) < \arctan y \ \& \ \arctan y < pi/2$

$\langle proof \rangle$

lemma *arctan-lbound*: $-(\pi/2) < \arctan y$
 $\langle proof \rangle$

lemma *arctan-ubound*: $\arctan y < \pi/2$
 $\langle proof \rangle$

lemma *arctan-tan*:
 $\llbracket -(\pi/2) < x; x < \pi/2 \rrbracket \implies \arctan(\tan x) = x$
 $\langle proof \rangle$

lemma *arctan-zero-zero* [*simp*]: $\arctan 0 = 0$
 $\langle proof \rangle$

lemma *cos-arctan-not-zero* [*simp*]: $\cos(\arctan x) \neq 0$
 $\langle proof \rangle$

lemma *tan-sec*: $\cos x \neq 0 \implies 1 + \tan(x)^2 = \text{inverse}(\cos x)^2$
 $\langle proof \rangle$

lemma *isCont-inverse-function2*:
fixes $f g :: \text{real} \Rightarrow \text{real}$ **shows**
 $\llbracket a < x; x < b; \forall z. a \leq z \wedge z \leq b \longrightarrow g(f z) = z; \forall z. a \leq z \wedge z \leq b \longrightarrow \text{isCont } f z \rrbracket$
 $\implies \text{isCont } g(f x)$
 $\langle proof \rangle$

lemma *isCont-arcsin*: $\llbracket -1 < x; x < 1 \rrbracket \implies \text{isCont } \arcsin x$
 $\langle proof \rangle$

lemma *isCont-arccos*: $\llbracket -1 < x; x < 1 \rrbracket \implies \text{isCont } \arccos x$
 $\langle proof \rangle$

lemma *isCont-arctan*: $\text{isCont } \arctan x$
 $\langle proof \rangle$

lemma *DERIV-arcsin*:
 $\llbracket -1 < x; x < 1 \rrbracket \implies \text{DERIV } \arcsin x :> \text{inverse}(\text{sqrt}(1 - x^2))$
 $\langle proof \rangle$

lemma *DERIV-arccos*:
 $\llbracket -1 < x; x < 1 \rrbracket \implies \text{DERIV } \arccos x :> \text{inverse}(-\text{sqrt}(1 - x^2))$
 $\langle proof \rangle$

lemma *DERIV-arctan*: $\text{DERIV } \arctan x :> \text{inverse}(1 + x^2)$
 $\langle proof \rangle$

63.13 More Theorems about Sin and Cos

lemma *cos-45*: $\cos (\pi / 4) = \text{sqrt } 2 / 2$
 $\langle \text{proof} \rangle$

lemma *cos-30*: $\cos (\pi / 6) = \text{sqrt } 3 / 2$
 $\langle \text{proof} \rangle$

lemma *sin-45*: $\sin (\pi / 4) = \text{sqrt } 2 / 2$
 $\langle \text{proof} \rangle$

lemma *sin-60*: $\sin (\pi / 3) = \text{sqrt } 3 / 2$
 $\langle \text{proof} \rangle$

lemma *cos-60*: $\cos (\pi / 3) = 1 / 2$
 $\langle \text{proof} \rangle$

lemma *sin-30*: $\sin (\pi / 6) = 1 / 2$
 $\langle \text{proof} \rangle$

lemma *tan-30*: $\tan (\pi / 6) = 1 / \text{sqrt } 3$
 $\langle \text{proof} \rangle$

lemma *tan-45*: $\tan (\pi / 4) = 1$
 $\langle \text{proof} \rangle$

lemma *tan-60*: $\tan (\pi / 3) = \text{sqrt } 3$
 $\langle \text{proof} \rangle$

NEEDED??

lemma [*simp*]:
 $\sin (x + 1 / 2 * \text{real } (\text{Suc } m) * \pi) =$
 $\cos (x + 1 / 2 * \text{real } (m) * \pi)$
 $\langle \text{proof} \rangle$

NEEDED??

lemma [*simp*]:
 $\sin (x + \text{real } (\text{Suc } m) * \pi / 2) =$
 $\cos (x + \text{real } (m) * \pi / 2)$
 $\langle \text{proof} \rangle$

lemma *DERIV-sin-add* [*simp*]: $\text{DERIV } (\%x. \sin (x + k)) \text{ } xa :> \cos (xa + k)$
 $\langle \text{proof} \rangle$

lemma *sin-cos-npi* [*simp*]: $\sin (\text{real } (\text{Suc } (2 * n)) * \pi / 2) = (-1) ^ n$
 $\langle \text{proof} \rangle$

lemma *cos-2npi* [*simp*]: $\cos (2 * \text{real } (n::\text{nat}) * \pi) = 1$
 $\langle \text{proof} \rangle$

lemma *cos-3over2-pi* [simp]: $\cos (3 / 2 * \pi) = 0$
 <proof>

lemma *sin-2npi* [simp]: $\sin (2 * \text{real } (n::\text{nat}) * \pi) = 0$
 <proof>

lemma *sin-3over2-pi* [simp]: $\sin (3 / 2 * \pi) = -1$
 <proof>

lemma [simp]:
 $\cos(x + 1 / 2 * \text{real}(Suc\ m) * \pi) = -\sin (x + 1 / 2 * \text{real } m * \pi)$
 <proof>

lemma [simp]: $\cos (x + \text{real}(Suc\ m) * \pi / 2) = -\sin (x + \text{real } m * \pi / 2)$
 <proof>

lemma *cos-pi-eq-zero* [simp]: $\cos (\pi * \text{real } (Suc\ (2 * m)) / 2) = 0$
 <proof>

lemma *DERIV-cos-add* [simp]: $DERIV (\%x. \cos (x + k))\ xa :> -\sin (xa + k)$
 <proof>

lemma *sin-zero-abs-cos-one*: $\sin x = 0 ==> |\cos x| = 1$
 <proof>

lemma *cos-one-sin-zero*: $\cos x = 1 ==> \sin x = 0$
 <proof>

63.14 Machins formula

lemma *tan-total-pi4*: **assumes** $|x| < 1$
shows $\exists z. -(\pi / 4) < z \wedge z < \pi / 4 \wedge \tan z = x$
 <proof>

lemma *arctan-add*: **assumes** $|x| \leq 1$ **and** $|y| < 1$
shows $\arctan x + \arctan y = \arctan ((x + y) / (1 - x * y))$
 <proof>

lemma *arctan1-eq-pi4*: $\arctan 1 = \pi / 4$ <proof>

theorem *machin*: $\pi / 4 = 4 * \arctan (1/5) - \arctan (1 / 239)$
 <proof>

63.15 Introducing the arcus tangens power series

lemma *monoseq-arctan-series*: **fixes** $x :: \text{real}$
assumes $|x| \leq 1$ **shows** $\text{monoseq } (\lambda n. 1 / \text{real } (n*2+1) * x^{(n*2+1)})$ **(is**
 $\text{monoseq } ?a)$

<proof>

lemma *zeroseq-arctan-series*: **fixes** $x :: \text{real}$

assumes $|x| \leq 1$ **shows** $(\lambda n. 1 / \text{real } (n*2+1) * x^{(n*2+1)}) \text{ ----> } 0$ (**is** $?a \text{ ----> } 0$)

<proof>

lemma *summable-arctan-series*: **fixes** $x :: \text{real}$ **and** $n :: \text{nat}$

assumes $|x| \leq 1$ **shows** $\text{summable } (\lambda k. (-1)^k * (1 / \text{real } (k*2+1) * x^{(k*2+1)}))$ (**is** $\text{summable } (?c x)$)

<proof>

lemma *less-one-imp-sqr-less-one*: **fixes** $x :: \text{real}$ **assumes** $|x| < 1$ **shows** $x^2 < 1$

<proof>

lemma *DERIV-arctan-series*: **assumes** $|x| < 1$

shows $\text{DERIV } (\lambda x'. \sum k. (-1)^k * (1 / \text{real } (k*2+1) * x'^{(k*2+1)})) x :> (\sum k. (-1)^k * x^{(k*2)})$ (**is** $\text{DERIV } ?\text{arctan} - :> ?\text{Int}$)

<proof>

lemma *arctan-series*: **assumes** $|x| \leq 1$

shows $\text{arctan } x = (\sum k. (-1)^k * (1 / \text{real } (k*2+1) * x^{(k*2+1)}))$ (**is** $= \text{suminf } (\lambda n. ?c x n)$)

<proof>

lemma *arctan-half*: **fixes** $x :: \text{real}$

shows $\text{arctan } x = 2 * \text{arctan } (x / (1 + \text{sqrt}(1 + x^2)))$

<proof>

lemma *arctan-monotone*: **assumes** $x < y$

shows $\text{arctan } x < \text{arctan } y$

<proof>

lemma *arctan-monotone'*: **assumes** $x \leq y$ **shows** $\text{arctan } x \leq \text{arctan } y$

<proof>

lemma *arctan-minus*: $\text{arctan } (-x) = - \text{arctan } x$

<proof>

lemma *arctan-inverse*: **assumes** $x \neq 0$ **shows** $\text{arctan } (1 / x) = \text{sgn } x * \pi / 2 - \text{arctan } x$

<proof>

theorem *pi-series*: $\pi / 4 = (\sum k. (-1)^k * 1 / \text{real } (k*2+1))$ (**is** $= ?\text{SUM}$)

<proof>

63.16 Existence of Polar Coordinates

lemma *cos-x-y-le-one*: $|x / \text{sqrt } (x^2 + y^2)| \leq 1$
 $\langle \text{proof} \rangle$

lemma *cos-arccos-abs*: $|y| \leq 1 \implies \cos (\arccos y) = y$
 $\langle \text{proof} \rangle$

lemma *sin-arccos-abs*: $|y| \leq 1 \implies \sin (\arccos y) = \text{sqrt } (1 - y^2)$
 $\langle \text{proof} \rangle$

lemmas *cos-arccos-lemma1* = *cos-arccos-abs* [*OF cos-x-y-le-one*]

lemmas *sin-arccos-lemma1* = *sin-arccos-abs* [*OF cos-x-y-le-one*]

lemma *polar-ex1*:
 $0 < y \implies \exists r a. x = r * \cos a \ \& \ y = r * \sin a$
 $\langle \text{proof} \rangle$

lemma *polar-ex2*:
 $y < 0 \implies \exists r a. x = r * \cos a \ \& \ y = r * \sin a$
 $\langle \text{proof} \rangle$

lemma *polar-Ex*: $\exists r a. x = r * \cos a \ \& \ y = r * \sin a$
 $\langle \text{proof} \rangle$

end

64 Complex: Complex Numbers: Rectangular and Polar Representations

theory *Complex*
imports *Transcendental*
begin

datatype *complex* = *Complex real real*

primrec
 $Re :: \text{complex} \Rightarrow \text{real}$
where
 $Re: Re (\text{Complex } x \ y) = x$

primrec
 $Im :: \text{complex} \Rightarrow \text{real}$
where
 $Im: Im (\text{Complex } x \ y) = y$

lemma *complex-surj* [*simp*]: $\text{Complex } (Re \ z) \ (Im \ z) = z$

$\langle \text{proof} \rangle$

lemma *complex-equality* [intro?]: $\llbracket \text{Re } x = \text{Re } y; \text{Im } x = \text{Im } y \rrbracket \implies x = y$
 $\langle \text{proof} \rangle$

lemma *expand-complex-eq*: $x = y \iff \text{Re } x = \text{Re } y \wedge \text{Im } x = \text{Im } y$
 $\langle \text{proof} \rangle$

lemmas *complex-Re-Im-cancel-iff* = *expand-complex-eq*

64.1 Addition and Subtraction

instantiation *complex* :: *ab-group-add*
begin

definition
complex-zero-def: $0 = \text{Complex } 0 \ 0$

definition
complex-add-def: $x + y = \text{Complex } (\text{Re } x + \text{Re } y) (\text{Im } x + \text{Im } y)$

definition
complex-minus-def: $- x = \text{Complex } (- \text{Re } x) (- \text{Im } x)$

definition
complex-diff-def: $x - (y :: \text{complex}) = x + - y$

lemma *Complex-eq-0* [simp]: $\text{Complex } a \ b = 0 \iff a = 0 \wedge b = 0$
 $\langle \text{proof} \rangle$

lemma *complex-Re-zero* [simp]: $\text{Re } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *complex-Im-zero* [simp]: $\text{Im } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *complex-add* [simp]:
 $\text{Complex } a \ b + \text{Complex } c \ d = \text{Complex } (a + c) (b + d)$
 $\langle \text{proof} \rangle$

lemma *complex-Re-add* [simp]: $\text{Re } (x + y) = \text{Re } x + \text{Re } y$
 $\langle \text{proof} \rangle$

lemma *complex-Im-add* [simp]: $\text{Im } (x + y) = \text{Im } x + \text{Im } y$
 $\langle \text{proof} \rangle$

lemma *complex-minus* [simp]:
 $- (\text{Complex } a \ b) = \text{Complex } (- a) (- b)$
 $\langle \text{proof} \rangle$

lemma *complex-Re-minus* [simp]: $\text{Re } (-x) = -\text{Re } x$
 $\langle \text{proof} \rangle$

lemma *complex-Im-minus* [simp]: $\text{Im } (-x) = -\text{Im } x$
 $\langle \text{proof} \rangle$

lemma *complex-diff* [simp]:
 $\text{Complex } a \ b - \text{Complex } c \ d = \text{Complex } (a - c) \ (b - d)$
 $\langle \text{proof} \rangle$

lemma *complex-Re-diff* [simp]: $\text{Re } (x - y) = \text{Re } x - \text{Re } y$
 $\langle \text{proof} \rangle$

lemma *complex-Im-diff* [simp]: $\text{Im } (x - y) = \text{Im } x - \text{Im } y$
 $\langle \text{proof} \rangle$

instance
 $\langle \text{proof} \rangle$

end

64.2 Multiplication and Division

instantiation *complex* :: {field, division-by-zero}
begin

definition
complex-one-def: $1 = \text{Complex } 1 \ 0$

definition
complex-mult-def: $x * y =$
 $\text{Complex } (\text{Re } x * \text{Re } y - \text{Im } x * \text{Im } y) \ (\text{Re } x * \text{Im } y + \text{Im } x * \text{Re } y)$

definition
complex-inverse-def: $\text{inverse } x =$
 $\text{Complex } (\text{Re } x / ((\text{Re } x)^2 + (\text{Im } x)^2)) \ (-\text{Im } x / ((\text{Re } x)^2 + (\text{Im } x)^2))$

definition
complex-divide-def: $x / (y::\text{complex}) = x * \text{inverse } y$

lemma *Complex-eq-1* [simp]: $(\text{Complex } a \ b = 1) = (a = 1 \ \wedge \ b = 0)$
 $\langle \text{proof} \rangle$

lemma *complex-Re-one* [simp]: $\text{Re } 1 = 1$
 $\langle \text{proof} \rangle$

lemma *complex-Im-one* [simp]: $\text{Im } 1 = 0$
 $\langle \text{proof} \rangle$

lemma *complex-mult* [simp]:

*Complex a b * Complex c d = Complex (a * c - b * d) (a * d + b * c)*
 ⟨proof⟩

lemma *complex-Re-mult* [simp]: *Re (x * y) = Re x * Re y - Im x * Im y*
 ⟨proof⟩

lemma *complex-Im-mult* [simp]: *Im (x * y) = Re x * Im y + Im x * Re y*
 ⟨proof⟩

lemma *complex-inverse* [simp]:

inverse (Complex a b) = Complex (a / (a² + b²)) (- b / (a² + b²))
 ⟨proof⟩

lemma *complex-Re-inverse*:

Re (inverse x) = Re x / ((Re x)² + (Im x)²)
 ⟨proof⟩

lemma *complex-Im-inverse*:

Im (inverse x) = - Im x / ((Re x)² + (Im x)²)
 ⟨proof⟩

instance

⟨proof⟩

end

64.3 Exponentiation

instantiation *complex* :: *recpower*

begin

primrec *power-complex* **where**

z ^ 0 = (1::complex)
 | *z* ^ Suc *n* = (*z*::complex) * *z* ^ *n*

instance ⟨proof⟩

declare *power-complex.simps* [simp del]

end

64.4 Numerals and Arithmetic

instantiation *complex* :: *number-ring*

begin

definition *number-of-complex* **where**

complex-number-of-def: *number-of w = (of-int w :: complex)*

instance

$\langle proof \rangle$

end

lemma *complex-Re-of-nat* [simp]: $Re\ (of\text{-}nat\ n) = of\text{-}nat\ n$

$\langle proof \rangle$

lemma *complex-Im-of-nat* [simp]: $Im\ (of\text{-}nat\ n) = 0$

$\langle proof \rangle$

lemma *complex-Re-of-int* [simp]: $Re\ (of\text{-}int\ z) = of\text{-}int\ z$

$\langle proof \rangle$

lemma *complex-Im-of-int* [simp]: $Im\ (of\text{-}int\ z) = 0$

$\langle proof \rangle$

lemma *complex-Re-number-of* [simp]: $Re\ (number\text{-}of\ v) = number\text{-}of\ v$

$\langle proof \rangle$

lemma *complex-Im-number-of* [simp]: $Im\ (number\text{-}of\ v) = 0$

$\langle proof \rangle$

lemma *Complex-eq-number-of* [simp]:

$(Complex\ a\ b = number\text{-}of\ w) = (a = number\text{-}of\ w \wedge b = 0)$

$\langle proof \rangle$

64.5 Scalar Multiplication

instantiation *complex* :: *real-field*

begin

definition

complex-scaleR-def: $scaleR\ r\ x = Complex\ (r * Re\ x)\ (r * Im\ x)$

lemma *complex-scaleR* [simp]:

$scaleR\ r\ (Complex\ a\ b) = Complex\ (r * a)\ (r * b)$

$\langle proof \rangle$

lemma *complex-Re-scaleR* [simp]: $Re\ (scaleR\ r\ x) = r * Re\ x$

$\langle proof \rangle$

lemma *complex-Im-scaleR* [simp]: $Im\ (scaleR\ r\ x) = r * Im\ x$

$\langle proof \rangle$

instance

$\langle proof \rangle$

end

64.6 Properties of Embedding from Reals

abbreviation

$\text{complex-of-real} :: \text{real} \Rightarrow \text{complex}$ **where**
 $\text{complex-of-real} \equiv \text{of-real}$

lemma $\text{complex-of-real-def}$: $\text{complex-of-real } r = \text{Complex } r \ 0$
 $\langle \text{proof} \rangle$

lemma $\text{Re-complex-of-real}$ [simp]: $\text{Re } (\text{complex-of-real } z) = z$
 $\langle \text{proof} \rangle$

lemma $\text{Im-complex-of-real}$ [simp]: $\text{Im } (\text{complex-of-real } z) = 0$
 $\langle \text{proof} \rangle$

lemma $\text{Complex-add-complex-of-real}$ [simp]:
 $\text{Complex } x \ y + \text{complex-of-real } r = \text{Complex } (x+r) \ y$
 $\langle \text{proof} \rangle$

lemma $\text{complex-of-real-add-Complex}$ [simp]:
 $\text{complex-of-real } r + \text{Complex } x \ y = \text{Complex } (r+x) \ y$
 $\langle \text{proof} \rangle$

lemma $\text{Complex-mult-complex-of-real}$:
 $\text{Complex } x \ y * \text{complex-of-real } r = \text{Complex } (x*r) \ (y*r)$
 $\langle \text{proof} \rangle$

lemma $\text{complex-of-real-mult-Complex}$:
 $\text{complex-of-real } r * \text{Complex } x \ y = \text{Complex } (r*x) \ (r*y)$
 $\langle \text{proof} \rangle$

64.7 Vector Norm

instantiation $\text{complex} :: \text{real-normed-field}$
begin

definition

complex-norm-def : $\text{norm } z = \text{sqrt } ((\text{Re } z)^2 + (\text{Im } z)^2)$

abbreviation

$\text{cmod} :: \text{complex} \Rightarrow \text{real}$ **where**
 $\text{cmod} \equiv \text{norm}$

definition

complex-sgn-def : $\text{sgn } x = x /_R \text{cmod } x$

lemmas $\text{cmod-def} = \text{complex-norm-def}$

lemma *complex-norm* [simp]: $cmod \ (Complex \ x \ y) = sqrt \ (x^2 + y^2)$
 ⟨proof⟩

instance
 ⟨proof⟩

end

lemma *cmod-unit-one* [simp]: $cmod \ (Complex \ (cos \ a) \ (sin \ a)) = 1$
 ⟨proof⟩

lemma *cmod-complex-polar* [simp]:
 $cmod \ (complex-of-real \ r * Complex \ (cos \ a) \ (sin \ a)) = abs \ r$
 ⟨proof⟩

lemma *complex-Re-le-cmod*: $Re \ x \leq cmod \ x$
 ⟨proof⟩

lemma *complex-mod-minus-le-complex-mod* [simp]: $- \ cmod \ x \leq cmod \ x$
 ⟨proof⟩

lemma *complex-mod-triangle-ineq2* [simp]: $cmod(b + a) - cmod \ b \leq cmod \ a$
 ⟨proof⟩

lemmas *real-sum-squared-expand* = *power2-sum* [where 'a=real]

lemma *abs-Re-le-cmod*: $|Re \ x| \leq cmod \ x$
 ⟨proof⟩

lemma *abs-Im-le-cmod*: $|Im \ x| \leq cmod \ x$
 ⟨proof⟩

64.8 Completeness of the Complexes

interpretation *Re*: *bounded-linear Re*
 ⟨proof⟩

interpretation *Im*: *bounded-linear Im*
 ⟨proof⟩

lemma *LIMSEQ-Complex*:

$$\llbracket X \text{ ----> } a; Y \text{ ----> } b \rrbracket \implies (\lambda n. Complex \ (X \ n) \ (Y \ n)) \text{ ----> } Complex$$

$$a \ b$$

 ⟨proof⟩

instance *complex :: banach*
 ⟨proof⟩

64.9 The Complex Number i

definition

$ii :: \text{complex } (i)$ **where**

$i\text{-def}: ii \equiv \text{Complex } 0\ 1$

lemma $\text{complex-Re-}i$ [simp]: $\text{Re } ii = 0$

$\langle \text{proof} \rangle$

lemma $\text{complex-Im-}i$ [simp]: $\text{Im } ii = 1$

$\langle \text{proof} \rangle$

lemma $\text{Complex-eq-}i$ [simp]: $(\text{Complex } x\ y = ii) = (x = 0 \wedge y = 1)$

$\langle \text{proof} \rangle$

lemma $\text{complex-}i\text{-not-zero}$ [simp]: $ii \neq 0$

$\langle \text{proof} \rangle$

lemma $\text{complex-}i\text{-not-one}$ [simp]: $ii \neq 1$

$\langle \text{proof} \rangle$

lemma $\text{complex-}i\text{-not-number-of}$ [simp]: $ii \neq \text{number-of } w$

$\langle \text{proof} \rangle$

lemma $i\text{-mult-Complex}$ [simp]: $ii * \text{Complex } a\ b = \text{Complex } (-\ b)\ a$

$\langle \text{proof} \rangle$

lemma $\text{Complex-mult-}i$ [simp]: $\text{Complex } a\ b * ii = \text{Complex } (-\ b)\ a$

$\langle \text{proof} \rangle$

lemma $i\text{-complex-of-real}$ [simp]: $ii * \text{complex-of-real } r = \text{Complex } 0\ r$

$\langle \text{proof} \rangle$

lemma $\text{complex-of-real-}i$ [simp]: $\text{complex-of-real } r * ii = \text{Complex } 0\ r$

$\langle \text{proof} \rangle$

lemma $i\text{-squared}$ [simp]: $ii * ii = -1$

$\langle \text{proof} \rangle$

lemma $\text{power2-}i$ [simp]: $ii^2 = -1$

$\langle \text{proof} \rangle$

lemma $\text{inverse-}i$ [simp]: $\text{inverse } ii = -\ ii$

$\langle \text{proof} \rangle$

64.10 Complex Conjugation

definition

$cnj :: \text{complex} \Rightarrow \text{complex}$ **where**

$cnj\ z = \text{Complex } (\text{Re } z)\ (-\ \text{Im } z)$

lemma *complex-cnj [simp]: $\text{cnj } (\text{Complex } a \ b) = \text{Complex } a \ (- \ b)$*
 $\langle \text{proof} \rangle$

lemma *complex-Re-cnj [simp]: $\text{Re } (\text{cnj } x) = \text{Re } x$*
 $\langle \text{proof} \rangle$

lemma *complex-Im-cnj [simp]: $\text{Im } (\text{cnj } x) = - \text{Im } x$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-cancel-iff [simp]: $(\text{cnj } x = \text{cnj } y) = (x = y)$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-cnj [simp]: $\text{cnj } (\text{cnj } z) = z$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-zero [simp]: $\text{cnj } 0 = 0$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-zero-iff [iff]: $(\text{cnj } z = 0) = (z = 0)$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-add: $\text{cnj } (x + y) = \text{cnj } x + \text{cnj } y$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-diff: $\text{cnj } (x - y) = \text{cnj } x - \text{cnj } y$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-minus: $\text{cnj } (- \ x) = - \ \text{cnj } x$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-one [simp]: $\text{cnj } 1 = 1$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-mult: $\text{cnj } (x * y) = \text{cnj } x * \text{cnj } y$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-inverse: $\text{cnj } (\text{inverse } x) = \text{inverse } (\text{cnj } x)$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-divide: $\text{cnj } (x / y) = \text{cnj } x / \text{cnj } y$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-power: $\text{cnj } (x ^ n) = \text{cnj } x ^ n$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-of-nat [simp]: $\text{cnj } (\text{of-nat } n) = \text{of-nat } n$*
 $\langle \text{proof} \rangle$

lemma *complex-cnj-of-int* [simp]: $\text{cnj } (\text{of-int } z) = \text{of-int } z$
 $\langle \text{proof} \rangle$

lemma *complex-cnj-number-of* [simp]: $\text{cnj } (\text{number-of } w) = \text{number-of } w$
 $\langle \text{proof} \rangle$

lemma *complex-cnj-scaleR*: $\text{cnj } (\text{scaleR } r \ x) = \text{scaleR } r \ (\text{cnj } x)$
 $\langle \text{proof} \rangle$

lemma *complex-mod-cnj* [simp]: $\text{cmod } (\text{cnj } z) = \text{cmod } z$
 $\langle \text{proof} \rangle$

lemma *complex-cnj-complex-of-real* [simp]: $\text{cnj } (\text{of-real } x) = \text{of-real } x$
 $\langle \text{proof} \rangle$

lemma *complex-cnj-i* [simp]: $\text{cnj } ii = - \ ii$
 $\langle \text{proof} \rangle$

lemma *complex-add-cnj*: $z + \text{cnj } z = \text{complex-of-real } (2 * \text{Re } z)$
 $\langle \text{proof} \rangle$

lemma *complex-diff-cnj*: $z - \text{cnj } z = \text{complex-of-real } (2 * \text{Im } z) * ii$
 $\langle \text{proof} \rangle$

lemma *complex-mult-cnj*: $z * \text{cnj } z = \text{complex-of-real } ((\text{Re } z)^2 + (\text{Im } z)^2)$
 $\langle \text{proof} \rangle$

lemma *complex-mod-mult-cnj*: $\text{cmod } (z * \text{cnj } z) = (\text{cmod } z)^2$
 $\langle \text{proof} \rangle$

interpretation *cnj*: *bounded-linear cnj*
 $\langle \text{proof} \rangle$

64.11 The Functions *sgn* and *arg*

————— Argand —————

definition

$\text{arg} :: \text{complex} \Rightarrow \text{real}$ **where**
 $\text{arg } z = (\text{SOME } a. \text{Re}(\text{sgn } z) = \cos a \ \& \ \text{Im}(\text{sgn } z) = \sin a \ \& \ -\pi < a \ \& \ a \leq \pi)$

lemma *sgn-eq*: $\text{sgn } z = z / \text{complex-of-real } (\text{cmod } z)$
 $\langle \text{proof} \rangle$

lemma *i-mult-eq*: $ii * ii = \text{complex-of-real } (-1)$
 $\langle \text{proof} \rangle$

lemma *i-mult-eq2* [simp]: $ii * ii = -(1::\text{complex})$
 $\langle \text{proof} \rangle$

lemma *complex-eq-cancel-iff2* [simp]:
 $(\text{Complex } x \ y = \text{complex-of-real } xa) = (x = xa \ \& \ y = 0)$
 <proof>

lemma *Re-sgn* [simp]: $\text{Re}(\text{sgn } z) = \text{Re}(z) / \text{cmod } z$
 <proof>

lemma *Im-sgn* [simp]: $\text{Im}(\text{sgn } z) = \text{Im}(z) / \text{cmod } z$
 <proof>

lemma *complex-inverse-complex-split*:
 $\text{inverse}(\text{complex-of-real } x + ii * \text{complex-of-real } y) =$
 $\text{complex-of-real}(x / (x^2 + y^2)) -$
 $ii * \text{complex-of-real}(y / (x^2 + y^2))$
 <proof>

lemma *cos-arg-i-mult-zero-pos*:
 $0 < y ==> \cos(\arg(\text{Complex } 0 \ y)) = 0$
 <proof>

lemma *cos-arg-i-mult-zero-neg*:
 $y < 0 ==> \cos(\arg(\text{Complex } 0 \ y)) = 0$
 <proof>

lemma *cos-arg-i-mult-zero* [simp]:
 $y \neq 0 ==> \cos(\arg(\text{Complex } 0 \ y)) = 0$
 <proof>

64.12 Finally! Polar Form for Complex Numbers

definition

$\text{cis} :: \text{real} \Rightarrow \text{complex}$ **where**
 $\text{cis } a = \text{Complex } (\cos a) (\sin a)$

definition

$\text{rcis} :: [\text{real}, \text{real}] \Rightarrow \text{complex}$ **where**
 $\text{rcis } r \ a = \text{complex-of-real } r * \text{cis } a$

definition

expi :: *complex* => *complex* **where**
expi *z* = *complex-of-real*(*exp* (*Re* *z*)) * *cis* (*Im* *z*)

lemma *complex-split-polar*:
 $\exists r\ a.\ z = \text{complex-of-real } r * (\text{Complex } (\cos a) (\sin a))$
 <proof>

lemma *rcis-Ex*: $\exists r\ a.\ z = \text{rcis } r\ a$
 <proof>

lemma *Re-rcis [simp]*: $\text{Re}(\text{rcis } r\ a) = r * \cos a$
 <proof>

lemma *Im-rcis [simp]*: $\text{Im}(\text{rcis } r\ a) = r * \sin a$
 <proof>

lemma *sin-cos-squared-add2-mult*: $(r * \cos a)^2 + (r * \sin a)^2 = r^2$
 <proof>

lemma *complex-mod-rcis [simp]*: $\text{cmod}(\text{rcis } r\ a) = \text{abs } r$
 <proof>

lemma *complex-mod-sqrt-Re-mult-cnj*: $\text{cmod } z = \sqrt{\text{Re } (z * \text{cnj } z)}$
 <proof>

lemma *complex-In-mult-cnj-zero [simp]*: $\text{Im } (z * \text{cnj } z) = 0$
 <proof>

lemma *cis-rcis-eq*: $\text{cis } a = \text{rcis } 1\ a$
 <proof>

lemma *rcis-mult*: $\text{rcis } r1\ a * \text{rcis } r2\ b = \text{rcis } (r1 * r2) (a + b)$
 <proof>

lemma *cis-mult*: $\text{cis } a * \text{cis } b = \text{cis } (a + b)$
 <proof>

lemma *cis-zero [simp]*: $\text{cis } 0 = 1$
 <proof>

lemma *rcis-zero-mod [simp]*: $\text{rcis } 0\ a = 0$
 <proof>

lemma *rcis-zero-arg [simp]*: $\text{rcis } r\ 0 = \text{complex-of-real } r$

$\langle \text{proof} \rangle$

lemma *complex-of-real-minus-one*:

$$\text{complex-of-real } (-(1::\text{real})) = -(1::\text{complex})$$

$\langle \text{proof} \rangle$

lemma *complex-i-mult-minus* [simp]: $ii * (ii * x) = - x$

$\langle \text{proof} \rangle$

lemma *cis-real-of-nat-Suc-mult*:

$$\text{cis } (\text{real } (\text{Suc } n) * a) = \text{cis } a * \text{cis } (\text{real } n * a)$$

$\langle \text{proof} \rangle$

lemma *DeMoivre*: $(\text{cis } a) ^ n = \text{cis } (\text{real } n * a)$

$\langle \text{proof} \rangle$

lemma *DeMoivre2*: $(\text{rcis } r a) ^ n = \text{rcis } (r ^ n) (\text{real } n * a)$

$\langle \text{proof} \rangle$

lemma *cis-inverse* [simp]: $\text{inverse}(\text{cis } a) = \text{cis } (-a)$

$\langle \text{proof} \rangle$

lemma *rcis-inverse*: $\text{inverse}(\text{rcis } r a) = \text{rcis } (1/r) (-a)$

$\langle \text{proof} \rangle$

lemma *cis-divide*: $\text{cis } a / \text{cis } b = \text{cis } (a - b)$

$\langle \text{proof} \rangle$

lemma *rcis-divide*: $\text{rcis } r1 a / \text{rcis } r2 b = \text{rcis } (r1/r2) (a - b)$

$\langle \text{proof} \rangle$

lemma *Re-cis* [simp]: $\text{Re}(\text{cis } a) = \cos a$

$\langle \text{proof} \rangle$

lemma *Im-cis* [simp]: $\text{Im}(\text{cis } a) = \sin a$

$\langle \text{proof} \rangle$

lemma *cos-n-Re-cis-pow-n*: $\cos (\text{real } n * a) = \text{Re}(\text{cis } a ^ n)$

$\langle \text{proof} \rangle$

lemma *sin-n-Im-cis-pow-n*: $\sin (\text{real } n * a) = \text{Im}(\text{cis } a ^ n)$

$\langle \text{proof} \rangle$

lemma *expi-add*: $\text{expi}(a + b) = \text{expi}(a) * \text{expi}(b)$

$\langle \text{proof} \rangle$

lemma *expi-zero* [simp]: $\text{expi } (0::\text{complex}) = 1$

$\langle \text{proof} \rangle$

lemma *complex-expi-Ex*: $\exists a\ r. z = \text{complex-of-real } r * \text{expi } a$
 $\langle \text{proof} \rangle$

lemma *expi-two-pi-i [simp]*: $\text{expi}((2::\text{complex}) * \text{complex-of-real } \pi * i) = 1$
 $\langle \text{proof} \rangle$

end

65 Log: Logarithms: Standard Version

theory *Log*
imports *Transcendental*
begin

definition
 $\text{powr} :: [\text{real}, \text{real}] \Rightarrow \text{real} \quad (\text{infixr } \text{powr } 80) \text{ where}$
 — exponentiation with real exponent
 $x \text{ powr } a = \exp(a * \ln x)$

definition
 $\text{log} :: [\text{real}, \text{real}] \Rightarrow \text{real} \text{ where}$
 — logarithm of x to base a
 $\text{log } a\ x = \ln x / \ln a$

lemma *powr-one-eq-one [simp]*: $1 \text{ powr } a = 1$
 $\langle \text{proof} \rangle$

lemma *powr-zero-eq-one [simp]*: $x \text{ powr } 0 = 1$
 $\langle \text{proof} \rangle$

lemma *powr-one-gt-zero-iff [simp]*: $(x \text{ powr } 1 = x) = (0 < x)$
 $\langle \text{proof} \rangle$

declare *powr-one-gt-zero-iff [THEN iffD2, simp]*

lemma *powr-mult*:
 $[| 0 < x; 0 < y |] \Rightarrow (x * y) \text{ powr } a = (x \text{ powr } a) * (y \text{ powr } a)$
 $\langle \text{proof} \rangle$

lemma *powr-gt-zero [simp]*: $0 < x \text{ powr } a$
 $\langle \text{proof} \rangle$

lemma *powr-ge-pzero [simp]*: $0 \leq x \text{ powr } y$
 $\langle \text{proof} \rangle$

lemma *powr-not-zero [simp]*: $x \text{ powr } a \neq 0$

$\langle proof \rangle$

lemma *powr-divide*:

$\llbracket 0 < x; 0 < y \rrbracket \implies (x / y) \text{ powr } a = (x \text{ powr } a) / (y \text{ powr } a)$
 $\langle proof \rangle$

lemma *powr-divide2*: $x \text{ powr } a / x \text{ powr } b = x \text{ powr } (a - b)$
 $\langle proof \rangle$

lemma *powr-add*: $x \text{ powr } (a + b) = (x \text{ powr } a) * (x \text{ powr } b)$
 $\langle proof \rangle$

lemma *powr-powr*: $(x \text{ powr } a) \text{ powr } b = x \text{ powr } (a * b)$
 $\langle proof \rangle$

lemma *powr-powr-swap*: $(x \text{ powr } a) \text{ powr } b = (x \text{ powr } b) \text{ powr } a$
 $\langle proof \rangle$

lemma *powr-minus*: $x \text{ powr } (-a) = \text{inverse } (x \text{ powr } a)$
 $\langle proof \rangle$

lemma *powr-minus-divide*: $x \text{ powr } (-a) = 1 / (x \text{ powr } a)$
 $\langle proof \rangle$

lemma *powr-less-mono*: $\llbracket a < b; 1 < x \rrbracket \implies x \text{ powr } a < x \text{ powr } b$
 $\langle proof \rangle$

lemma *powr-less-cancel*: $\llbracket x \text{ powr } a < x \text{ powr } b; 1 < x \rrbracket \implies a < b$
 $\langle proof \rangle$

lemma *powr-less-cancel-iff* [simp]: $1 < x \implies (x \text{ powr } a < x \text{ powr } b) = (a < b)$
 $\langle proof \rangle$

lemma *powr-le-cancel-iff* [simp]: $1 < x \implies (x \text{ powr } a \leq x \text{ powr } b) = (a \leq b)$
 $\langle proof \rangle$

lemma *log-ln*: $\ln x = \log (\exp(1)) x$
 $\langle proof \rangle$

lemma *powr-log-cancel* [simp]:

$\llbracket 0 < a; a \neq 1; 0 < x \rrbracket \implies a \text{ powr } (\log a x) = x$
 $\langle proof \rangle$

lemma *log-powr-cancel* [simp]: $\llbracket 0 < a; a \neq 1 \rrbracket \implies \log a (a \text{ powr } y) = y$
 $\langle proof \rangle$

lemma *log-mult*:

$\llbracket 0 < a; a \neq 1; 0 < x; 0 < y \rrbracket$
 $\implies \log a (x * y) = \log a x + \log a y$

$\langle proof \rangle$

lemma *log-eq-div-ln-mult-log*:

$$[[0 < a; a \neq 1; 0 < b; b \neq 1; 0 < x]] \\ \implies \log a x = (\ln b / \ln a) * \log b x$$

$\langle proof \rangle$

Base 10 logarithms

lemma *log-base-10-eq1*: $0 < x \implies \log 10 x = (\ln (\exp 1) / \ln 10) * \ln x$

$\langle proof \rangle$

lemma *log-base-10-eq2*: $0 < x \implies \log 10 x = (\log 10 (\exp 1)) * \ln x$

$\langle proof \rangle$

lemma *log-one* [simp]: $\log a 1 = 0$

$\langle proof \rangle$

lemma *log-eq-one* [simp]: $[[0 < a; a \neq 1]] \implies \log a a = 1$

$\langle proof \rangle$

lemma *log-inverse*:

$$[[0 < a; a \neq 1; 0 < x]] \implies \log a (\text{inverse } x) = - \log a x$$

$\langle proof \rangle$

lemma *log-divide*:

$$[[0 < a; a \neq 1; 0 < x; 0 < y]] \implies \log a (x/y) = \log a x - \log a y$$

$\langle proof \rangle$

lemma *log-less-cancel-iff* [simp]:

$$[[1 < a; 0 < x; 0 < y]] \implies (\log a x < \log a y) = (x < y)$$

$\langle proof \rangle$

lemma *log-le-cancel-iff* [simp]:

$$[[1 < a; 0 < x; 0 < y]] \implies (\log a x \leq \log a y) = (x \leq y)$$

$\langle proof \rangle$

lemma *powr-realpow*: $0 < x \implies x \text{ powr } (\text{real } n) = x^{\text{real } n}$

$\langle proof \rangle$

lemma *powr-realpow2*: $0 \leq x \implies 0 < n \implies x^{\text{real } n} = (\text{if } (x = 0) \text{ then } 0 \text{ else } x \text{ powr } (\text{real } n))$

$\langle proof \rangle$

lemma *ln-pwr*: $0 < x \implies 0 < y \implies \ln(x \text{ powr } y) = y * \ln x$

$\langle proof \rangle$

lemma *ln-bound*: $1 \leq x \implies \ln x \leq x$

$\langle proof \rangle$

lemma *powr-mono*: $a \leq b \implies 1 \leq x \implies x \text{ powr } a \leq x \text{ powr } b$
 ⟨proof⟩

lemma *ge-one-powr-ge-zero*: $1 \leq x \implies 0 \leq a \implies 1 \leq x \text{ powr } a$
 ⟨proof⟩

lemma *powr-less-mono2*: $0 < a \implies 0 < x \implies x < y \implies x \text{ powr } a < y \text{ powr } a$
 ⟨proof⟩

lemma *powr-less-mono2-neg*: $a < 0 \implies 0 < x \implies x < y \implies y \text{ powr } a < x \text{ powr } a$
 ⟨proof⟩

lemma *powr-mono2*: $0 \leq a \implies 0 < x \implies x \leq y \implies x \text{ powr } a \leq y \text{ powr } a$
 ⟨proof⟩

lemma *ln-powr-bound*: $1 \leq x \implies 0 < a \implies \ln x \leq (x \text{ powr } a) / a$
 ⟨proof⟩

lemma *ln-powr-bound2*: $1 < x \implies 0 < a \implies (\ln x) \text{ powr } a \leq (a \text{ powr } a) * x$
 ⟨proof⟩

lemma *LIMSEQ-neg-powr*: $0 < s \implies (\%x. (\text{real } x) \text{ powr } - s) \text{ ----} > 0$
 ⟨proof⟩

end

66 Ln: Properties of ln

theory *Ln*

imports *Transcendental*

begin

lemma *exp-first-two-terms*: $\exp x = 1 + x + \text{suminf } (\%n. \text{inverse}(\text{real } (\text{fact } (n+2)))) * (x ^ (n+2)))$
 ⟨proof⟩

lemma *exp-tail-after-first-two-terms-summable*:
 $\text{summable } (\%n. \text{inverse}(\text{real } (\text{fact } (n+2)))) * (x ^ (n+2)))$
 ⟨proof⟩

lemma *aux1*: **assumes** $a: 0 \leq x$ **and** $b: x \leq 1$
shows $\text{inverse } (\text{real } (\text{fact } (n + 2))) * x ^ (n + 2) \leq (x^2/2) * ((1/2) ^ n)$
 ⟨proof⟩

lemma *aux2*: (%n. (x::real) ^ 2 / 2 * (1 / 2) ^ n) sums x^2
 <proof>

lemma *exp-bound*: 0 <= (x::real) ==> x <= 1 ==> exp x <= 1 + x + x^2
 <proof>

lemma *aux4*: 0 <= (x::real) ==> x <= 1 ==> exp (x - x^2) <= 1 + x
 <proof>

lemma *ln-one-plus-pos-lower-bound*: 0 <= x ==> x <= 1 ==>
 x - x^2 <= ln (1 + x)
 <proof>

lemma *ln-one-minus-pos-upper-bound*: 0 <= x ==> x < 1 ==> ln (1 - x) <= - x
 <proof>

lemma *aux5*: x < 1 ==> ln(1 - x) = - ln(1 + x / (1 - x))
 <proof>

lemma *ln-one-minus-pos-lower-bound*: 0 <= x ==> x <= (1 / 2) ==>
 - x - 2 * x^2 <= ln (1 - x)
 <proof>

lemma *exp-ge-add-one-self* [simp]: 1 + (x::real) <= exp x
 <proof>

lemma *ln-add-one-self-le-self2*: -1 < x ==> ln(1 + x) <= x
 <proof>

lemma *abs-ln-one-plus-x-minus-x-bound-nonneg*:
 0 <= x ==> x <= 1 ==> abs(ln (1 + x) - x) <= x^2
 <proof>

lemma *abs-ln-one-plus-x-minus-x-bound-nonpos*:
 -(1 / 2) <= x ==> x <= 0 ==> abs(ln (1 + x) - x) <= 2 * x^2
 <proof>

lemma *abs-ln-one-plus-x-minus-x-bound*:
 abs x <= 1 / 2 ==> abs(ln (1 + x) - x) <= 2 * x^2
 <proof>

lemma *DERIV-ln*: 0 < x ==> DERIV ln x :> 1 / x
 <proof>

lemma *ln-x-over-x-mono*: exp 1 <= x ==> x <= y ==> (ln y / y) <= (ln x / x)
 <proof>

end

67 MacLaurin: MacLaurin Series

theory *MacLaurin*
imports *Transcendental*
begin

67.1 Maclaurin’s Theorem with Lagrange Form of Remainder

This is a very long, messy proof even now that it’s been broken down into lemmas.

lemma *Maclaurin-lemma*:

$$0 < h ==> \\ \exists B. f\ h = (\sum_{m=0..<n.} (j\ m\ /\ real\ (fact\ m)) * (h^m)) + \\ (B * ((h^n) /\ real(fact\ n)))$$

<proof>

lemma *eq-diff-eq'*: $(x = y - z) = (y = x + (z::real))$

<proof>

A crude tactic to differentiate by proof.

lemmas *deriv-rulesI* =

DERIV-ident DERIV-const DERIV-cos DERIV-cmult
DERIV-sin DERIV-exp DERIV-inverse DERIV-pow
DERIV-add DERIV-diff DERIV-mult DERIV-minus
DERIV-inverse-fun DERIV-quotient DERIV-fun-pow
DERIV-fun-exp DERIV-fun-sin DERIV-fun-cos
DERIV-ident DERIV-const DERIV-cos

<ML>

lemma *Maclaurin-lemma2*:

assumes *diff*: $\forall m\ t. m < n \wedge 0 \leq t \wedge t \leq h \longrightarrow DERIV\ (diff\ m)\ t :\> diff\ (Suc\ m)\ t$

assumes *n*: $n = Suc\ k$

assumes *difg*: *difg* =

$$(\lambda m\ t. diff\ m\ t - \\ ((\sum_{p=0..<n-m.} diff\ (m+p)\ 0 /\ real\ (fact\ p) * t^p) + \\ B * (t^n /\ real\ (fact\ (n-m))))))$$

shows

$\forall m\ t. m < n \ \& \ 0 \leq t \ \& \ t \leq h \longrightarrow DERIV\ (difg\ m)\ t :\> difg\ (Suc\ m)\ t$

<proof>

lemma *Maclaurin*:

assumes h : $0 < h$

assumes n : $0 < n$

assumes *diff-0*: $\text{diff } 0 = f$

assumes *diff-Suc*:

$\forall m\ t. m < n \ \& \ 0 \leq t \ \& \ t \leq h \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t$

shows

$\exists t. 0 < t \ \& \ t < h \ \&$

$f\ h =$

$\text{setsum } (\%m. (\text{diff } m\ 0 / \text{real } (\text{fact } m)) * h ^ m) \ \{0..<n\} +$
 $(\text{diff } n\ t / \text{real } (\text{fact } n)) * h ^ n$

$\langle \text{proof} \rangle$

lemma *Maclaurin-objl*:

$0 < h \ \& \ n > 0 \ \& \ \text{diff } 0 = f \ \&$

$(\forall m\ t. m < n \ \& \ 0 \leq t \ \& \ t \leq h \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t)$

$\longrightarrow (\exists t. 0 < t \ \& \ t < h \ \&$

$f\ h = (\sum m=0..<n. \text{diff } m\ 0 / \text{real } (\text{fact } m) * h ^ m) +$
 $\text{diff } n\ t / \text{real } (\text{fact } n) * h ^ n)$

$\langle \text{proof} \rangle$

lemma *Maclaurin2*:

$[[\ 0 < h; \text{diff } 0 = f;$

$\forall m\ t.$

$m < n \ \& \ 0 \leq t \ \& \ t \leq h \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t \]]$

$\implies \exists t. 0 < t \ \&$

$t \leq h \ \&$

$f\ h =$

$(\sum m=0..<n. \text{diff } m\ 0 / \text{real } (\text{fact } m) * h ^ m) +$
 $\text{diff } n\ t / \text{real } (\text{fact } n) * h ^ n$

$\langle \text{proof} \rangle$

lemma *Maclaurin2-objl*:

$0 < h \ \& \ \text{diff } 0 = f \ \&$

$(\forall m\ t.$

$m < n \ \& \ 0 \leq t \ \& \ t \leq h \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t)$

$\longrightarrow (\exists t. 0 < t \ \&$

$t \leq h \ \&$

$f\ h =$

$(\sum m=0..<n. \text{diff } m\ 0 / \text{real } (\text{fact } m) * h ^ m) +$
 $\text{diff } n\ t / \text{real } (\text{fact } n) * h ^ n)$

$\langle \text{proof} \rangle$

lemma *Maclaurin-minus*:

$[[\ h < 0; n > 0; \text{diff } 0 = f;$

$\forall m\ t. m < n \ \& \ h \leq t \ \& \ t \leq 0 \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t \]]$

$\implies \exists t. h < t \ \&$

$t < 0 \ \&$

$$f h =$$

$$(\sum_{m=0..<n.} \text{diff } m \ 0 / \text{real } (\text{fact } m) * h \wedge m) +$$

$$\text{diff } n \ t / \text{real } (\text{fact } n) * h \wedge n$$

⟨proof⟩

lemma *Maclaurin-minus-objl*:

$$(h < 0 \ \& \ n > 0 \ \& \ \text{diff } 0 = f \ \&$$

$$(\forall m \ t.$$

$$m < n \ \& \ h \leq t \ \& \ t \leq 0 \ \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t))$$

$$\longrightarrow (\exists t. h < t \ \&$$

$$t < 0 \ \&$$

$$f h =$$

$$(\sum_{m=0..<n.} \text{diff } m \ 0 / \text{real } (\text{fact } m) * h \wedge m) +$$

$$\text{diff } n \ t / \text{real } (\text{fact } n) * h \wedge n)$$

⟨proof⟩

67.2 More Convenient ”Bidirectional” Version.

lemma *Maclaurin-bi-le-lemma* [rule-format]:

$$n > 0 \longrightarrow$$

$$\text{diff } 0 \ 0 =$$

$$(\sum_{m=0..<n.} \text{diff } m \ 0 * 0 \wedge m / \text{real } (\text{fact } m)) +$$

$$\text{diff } n \ 0 * 0 \wedge n / \text{real } (\text{fact } n)$$

⟨proof⟩

lemma *Maclaurin-bi-le*:

$$[\text{diff } 0 = f ;$$

$$\forall m \ t. m < n \ \& \ \text{abs } t \leq \text{abs } x \longrightarrow \text{DERIV } (\text{diff } m) \ t :> \text{diff } (\text{Suc } m) \ t]$$

$$\implies \exists t. \text{abs } t \leq \text{abs } x \ \&$$

$$f x =$$

$$(\sum_{m=0..<n.} \text{diff } m \ 0 / \text{real } (\text{fact } m) * x \wedge m) +$$

$$\text{diff } n \ t / \text{real } (\text{fact } n) * x \wedge n$$

⟨proof⟩

lemma *Maclaurin-all-lt*:

$$[\text{diff } 0 = f ;$$

$$\forall m \ x. \text{DERIV } (\text{diff } m) \ x :> \text{diff } (\text{Suc } m) \ x;$$

$$x \sim 0; n > 0$$

$$] \implies \exists t. 0 < \text{abs } t \ \& \ \text{abs } t < \text{abs } x \ \&$$

$$f x = (\sum_{m=0..<n.} (\text{diff } m \ 0 / \text{real } (\text{fact } m)) * x \wedge m) +$$

$$(\text{diff } n \ t / \text{real } (\text{fact } n)) * x \wedge n$$

⟨proof⟩

lemma *Maclaurin-all-lt-objl*:

$$\text{diff } 0 = f \ \&$$

$$(\forall m \ x. \text{DERIV } (\text{diff } m) \ x :> \text{diff } (\text{Suc } m) \ x) \ \&$$

$$x \sim 0 \ \& \ n > 0$$

$$\longrightarrow (\exists t. 0 < \text{abs } t \ \& \ \text{abs } t < \text{abs } x \ \&$$

$$f x = (\sum_{m=0..<n.} (\text{diff } m \ 0 / \text{real } (\text{fact } m)) * x \wedge m) +$$

$(\text{diff } n \ t \ / \ \text{real } (\text{fact } n)) * x \wedge n$

$\langle \text{proof} \rangle$

lemma *Maclaurin-zero* [rule-format]:

$x = (0::\text{real})$
 $\implies n \neq 0 \implies$
 $(\sum_{m=0..<n.} (\text{diff } m \ (0::\text{real}) \ / \ \text{real } (\text{fact } m)) * x \wedge m) =$
 $\text{diff } 0 \ 0$

$\langle \text{proof} \rangle$

lemma *Maclaurin-all-le*: $[\mid \text{diff } 0 = f;$

$\forall m \ x. \text{DERIV } (\text{diff } m) \ x :> \text{diff } (\text{Suc } m) \ x$
 $\mid \implies \exists t. \text{abs } t \leq \text{abs } x \ \&$
 $f \ x = (\sum_{m=0..<n.} (\text{diff } m \ 0 \ / \ \text{real } (\text{fact } m)) * x \wedge m) +$
 $(\text{diff } n \ t \ / \ \text{real } (\text{fact } n)) * x \wedge n$

$\langle \text{proof} \rangle$

lemma *Maclaurin-all-le-objl*: $\text{diff } 0 = f \ \&$

$(\forall m \ x. \text{DERIV } (\text{diff } m) \ x :> \text{diff } (\text{Suc } m) \ x)$
 $\implies (\exists t. \text{abs } t \leq \text{abs } x \ \&$
 $f \ x = (\sum_{m=0..<n.} (\text{diff } m \ 0 \ / \ \text{real } (\text{fact } m)) * x \wedge m) +$
 $(\text{diff } n \ t \ / \ \text{real } (\text{fact } n)) * x \wedge n)$

$\langle \text{proof} \rangle$

67.3 Version for Exponential Function

lemma *Maclaurin-exp-lt*: $[\mid x \sim 0; n > 0 \mid]$

$\implies (\exists t. 0 < \text{abs } t \ \&$
 $\text{abs } t < \text{abs } x \ \&$
 $\text{exp } x = (\sum_{m=0..<n.} (x \wedge m) \ / \ \text{real } (\text{fact } m)) +$
 $(\text{exp } t \ / \ \text{real } (\text{fact } n)) * x \wedge n$

$\langle \text{proof} \rangle$

lemma *Maclaurin-exp-le*:

$\exists t. \text{abs } t \leq \text{abs } x \ \&$
 $\text{exp } x = (\sum_{m=0..<n.} (x \wedge m) \ / \ \text{real } (\text{fact } m)) +$
 $(\text{exp } t \ / \ \text{real } (\text{fact } n)) * x \wedge n$

$\langle \text{proof} \rangle$

67.4 Version for Sine Function

lemma *mod-exhaust-less-4*:

$m \bmod 4 = 0 \mid m \bmod 4 = 1 \mid m \bmod 4 = 2 \mid m \bmod 4 = (3::\text{nat})$

$\langle \text{proof} \rangle$

lemma *Suc-Suc-mult-two-diff-two* [rule-format, simp]:

$n \neq 0 \implies \text{Suc } (\text{Suc } (2 * n - 2)) = 2 * n$

$\langle \text{proof} \rangle$

lemma *lemma-Suc-Suc-4n-diff-2* [rule-format, simp]:

$$n \neq 0 \longrightarrow \text{Suc} (\text{Suc} (4 * n - 2)) = 4 * n$$

⟨proof⟩

lemma *Suc-mult-two-diff-one* [rule-format, simp]:

$$n \neq 0 \longrightarrow \text{Suc} (2 * n - 1) = 2 * n$$

⟨proof⟩

It is unclear why so many variant results are needed.

lemma *Maclaurin-sin-expansion2*:

$$\begin{aligned} & \exists t. \text{abs } t \leq \text{abs } x \ \& \\ & \sin x = \\ & \quad \left(\sum_{m=0}^{<n.} (\text{if even } m \text{ then } 0 \right. \\ & \quad \quad \left. \text{else } (-1)^{(m - \text{Suc } 0) \text{ div } 2}) / \text{real } (\text{fact } m)) * \right. \\ & \quad \quad \left. x^{\wedge} m \right) \\ & \quad + ((\sin(t + 1/2 * \text{real } (n) * \pi) / \text{real } (\text{fact } n)) * x^{\wedge} n) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *Maclaurin-sin-expansion*:

$$\begin{aligned} & \exists t. \sin x = \\ & \quad \left(\sum_{m=0}^{<n.} (\text{if even } m \text{ then } 0 \right. \\ & \quad \quad \left. \text{else } (-1)^{(m - \text{Suc } 0) \text{ div } 2}) / \text{real } (\text{fact } m)) * \right. \\ & \quad \quad \left. x^{\wedge} m \right) \\ & \quad + ((\sin(t + 1/2 * \text{real } (n) * \pi) / \text{real } (\text{fact } n)) * x^{\wedge} n) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *Maclaurin-sin-expansion3*:

$$\begin{aligned} & [| \ n > 0; \ 0 < x \ |] \implies \\ & \quad \exists t. \ 0 < t \ \& \ t < x \ \& \\ & \quad \sin x = \\ & \quad \left(\sum_{m=0}^{<n.} (\text{if even } m \text{ then } 0 \right. \\ & \quad \quad \left. \text{else } (-1)^{(m - \text{Suc } 0) \text{ div } 2}) / \text{real } (\text{fact } m)) * \right. \\ & \quad \quad \left. x^{\wedge} m \right) \\ & \quad + ((\sin(t + 1/2 * \text{real } (n) * \pi) / \text{real } (\text{fact } n)) * x^{\wedge} n) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *Maclaurin-sin-expansion4*:

$$\begin{aligned} & 0 < x \implies \\ & \quad \exists t. \ 0 < t \ \& \ t \leq x \ \& \\ & \quad \sin x = \\ & \quad \left(\sum_{m=0}^{<n.} (\text{if even } m \text{ then } 0 \right. \\ & \quad \quad \left. \text{else } (-1)^{(m - \text{Suc } 0) \text{ div } 2}) / \text{real } (\text{fact } m)) * \right. \\ & \quad \quad \left. x^{\wedge} m \right) \\ & \quad + ((\sin(t + 1/2 * \text{real } (n) * \pi) / \text{real } (\text{fact } n)) * x^{\wedge} n) \\ & \langle \text{proof} \rangle \end{aligned}$$

67.5 Maclaurin Expansion for Cosine Function

lemma *sumr-cos-zero-one* [simp]:

$(\sum m=0..<(Suc\ n). (if\ even\ m\ then\ -1\ ^\ (m\ div\ 2)/(real\ (fact\ m))\ else\ 0) * 0\ ^\ m) = 1$
 <proof>

lemma *Maclaurin-cos-expansion*:

$\exists t. abs\ t \leq abs\ x \ \&$
 $cos\ x =$
 $(\sum m=0..<n. (if\ even\ m$
 $then\ -1\ ^\ (m\ div\ 2)/(real\ (fact\ m))$
 $else\ 0) * x\ ^\ m)$
 $+ ((cos(t + 1/2 * real\ (n) * pi) / real\ (fact\ n)) * x\ ^\ n)$
 <proof>

lemma *Maclaurin-cos-expansion2*:

$[| 0 < x; n > 0 |] ==>$
 $\exists t. 0 < t \ \& \ t < x \ \&$
 $cos\ x =$
 $(\sum m=0..<n. (if\ even\ m$
 $then\ -1\ ^\ (m\ div\ 2)/(real\ (fact\ m))$
 $else\ 0) * x\ ^\ m)$
 $+ ((cos(t + 1/2 * real\ (n) * pi) / real\ (fact\ n)) * x\ ^\ n)$
 <proof>

lemma *Maclaurin-minus-cos-expansion*:

$[| x < 0; n > 0 |] ==>$
 $\exists t. x < t \ \& \ t < 0 \ \&$
 $cos\ x =$
 $(\sum m=0..<n. (if\ even\ m$
 $then\ -1\ ^\ (m\ div\ 2)/(real\ (fact\ m))$
 $else\ 0) * x\ ^\ m)$
 $+ ((cos(t + 1/2 * real\ (n) * pi) / real\ (fact\ n)) * x\ ^\ n)$
 <proof>

lemma *sin-bound-lemma*:

$[| x = y; abs\ u \leq (v::real) |] ==> |(x + u) - y| \leq v$
 <proof>

TODO: move to Parity.thy

lemma *nat-odd-1* [simp]: $odd\ (1::nat)$

<proof>

lemma *Maclaurin-sin-bound*:

$abs(sin\ x - (\sum_{m=0..<n}. (if\ even\ m\ then\ 0\ else\ (-1\ ^\ ((m - Suc\ 0)\ div\ 2)) /$
 $real\ (fact\ m)) *$
 $x\ ^\ m)) \leq inverse(real\ (fact\ n)) * |x|^\ n$
 $\langle proof \rangle$

end

68 Taylor: Taylor series

theory *Taylor*

imports *MacLaurin*

begin

We use MacLaurin and the translation of the expansion point c to 0 to prove Taylor’s theorem.

lemma *taylor-up*:

assumes *INIT*: $n > 0 \text{ diff } 0 = f$
and *DERIV*: $(\forall\ m\ t. m < n \ \&\ a \leq t \ \&\ t \leq b \longrightarrow \text{DERIV } (\text{diff } m)\ t :> (\text{diff } (Suc\ m)\ t))$
and *INTERV*: $a \leq c < b$
shows $\exists\ t. c < t \ \&\ t < b \ \&$
 $f\ b = \text{setsum } (\%m. (\text{diff } m\ c / \text{real } (fact\ m)) * (b - c)^\ m) \{0..<n\} +$
 $(\text{diff } n\ t / \text{real } (fact\ n)) * (b - c)^\ n$
 $\langle proof \rangle$

lemma *taylor-down*:

assumes *INIT*: $n > 0 \text{ diff } 0 = f$
and *DERIV*: $(\forall\ m\ t. m < n \ \&\ a \leq t \ \&\ t \leq b \longrightarrow \text{DERIV } (\text{diff } m)\ t :> (\text{diff } (Suc\ m)\ t))$
and *INTERV*: $a < c \leq b$
shows $\exists\ t. a < t \ \&\ t < c \ \&$
 $f\ a = \text{setsum } (\%m. (\text{diff } m\ c / \text{real } (fact\ m)) * (a - c)^\ m) \{0..<n\} +$
 $(\text{diff } n\ t / \text{real } (fact\ n)) * (a - c)^\ n$
 $\langle proof \rangle$

lemma *taylor*:

assumes *INIT*: $n > 0 \text{ diff } 0 = f$
and *DERIV*: $(\forall\ m\ t. m < n \ \&\ a \leq t \ \&\ t \leq b \longrightarrow \text{DERIV } (\text{diff } m)\ t :> (\text{diff } (Suc\ m)\ t))$
and *INTERV*: $a \leq c \leq b \ a \leq x \leq b \ x \neq c$
shows $\exists\ t. (if\ x < c\ then\ (x < t \ \&\ t < c) \ else\ (c < t \ \&\ t < x)) \ \&$
 $f\ x = \text{setsum } (\%m. (\text{diff } m\ c / \text{real } (fact\ m)) * (x - c)^\ m) \{0..<n\} +$
 $(\text{diff } n\ t / \text{real } (fact\ n)) * (x - c)^\ n$
 $\langle proof \rangle$

end

69 Integration: Theory of Integration

theory *Integration*
imports *Deriv ATP-Linkup*
begin

We follow John Harrison in formalizing the Gauge integral.

definition

— Partitions and tagged partitions etc.

partition :: [(*real***real*), *nat* => *real*] => *bool* **where**

$$[code\ del]: partition = (\%(a,b)\ D.\ D\ 0 = a \ \& \ (\exists N.\ (\forall n < N.\ D(n) < D(Suc\ n)) \ \& \ (\forall n \geq N.\ D(n) = b)))$$

definition

psize :: (*nat* => *real*) => *nat* **where**

$$[code\ del]: psize\ D = (SOME\ N.\ (\forall n < N.\ D(n) < D(Suc\ n)) \ \& \ (\forall n \geq N.\ D(n) = D(N)))$$

definition

tpart :: [(*real***real*), ((*nat* => *real*)*(*nat* => *real*))] => *bool* **where**

$$[code\ del]: tpart = (\%(a,b)\ (D,p).\ partition(a,b)\ D \ \& \ (\forall n.\ D(n) \leq p(n) \ \& \ p(n) \leq D(Suc\ n)))$$

— Gauges and gauge-fine divisions

definition

gauge :: [*real* => *bool*, *real* => *real*] => *bool* **where**

$$[code\ del]: gauge\ E\ g = (\forall x.\ E\ x \dashrightarrow 0 < g(x))$$

definition

fine :: [*real* => *real*, ((*nat* => *real*)*(*nat* => *real*))] => *bool* **where**

$$[code\ del]: fine = (\%g\ (D,p).\ \forall n.\ n < (psize\ D) \dashrightarrow D(Suc\ n) - D(n) < g(p\ n))$$

— Riemann sum

definition

rsum :: (((*nat* => *real*)*(*nat* => *real*), *real* => *real*) => *real*) **where**

$$rsum = (\%(D,p)\ f.\ \sum n=0..<psize(D).\ f(p\ n) * (D(Suc\ n) - D(n)))$$

— Gauge integrability (definite)

definition

Integral :: [(*real***real*), *real* => *real*, *real*] => *bool* **where**

$$[code\ del]: Integral = (\%(a,b)\ f\ k.\ \forall e > 0.$$

$$(\exists g. \text{gauge}(\%x. a \leq x \ \& \ x \leq b) \ g \ \& \\ (\forall D \ p. \text{tpart}(a,b) \ (D,p) \ \& \ \text{fine}(g)(D,p) \ \longrightarrow \\ |\text{rsum}(D,p) \ f - k| < e)))$$

lemma *psize-unique*:

assumes 1: $\forall n < N. D(n) < D(\text{Suc } n)$

assumes 2: $\forall n \geq N. D(n) = D(N)$

shows *psize* $D = N$

$\langle \text{proof} \rangle$

lemma *partition-zero* [*simp*]: $a = b \implies \text{psize } (\%n. \text{if } n = 0 \text{ then } a \text{ else } b) = 0$

$\langle \text{proof} \rangle$

lemma *partition-one* [*simp*]: $a < b \implies \text{psize } (\%n. \text{if } n = 0 \text{ then } a \text{ else } b) = 1$

$\langle \text{proof} \rangle$

lemma *partition-single* [*simp*]:

$a \leq b \implies \text{partition}(a,b) (\%n. \text{if } n = 0 \text{ then } a \text{ else } b)$

$\langle \text{proof} \rangle$

lemma *partition-lhs*: $\text{partition}(a,b) \ D \implies (D(0) = a)$

$\langle \text{proof} \rangle$

lemma *partition*:

$(\text{partition}(a,b) \ D) =$

$((D \ 0 = a) \ \&$

$(\forall n < \text{psize } D. D \ n < D(\text{Suc } n)) \ \&$

$(\forall n \geq \text{psize } D. D \ n = b))$

$\langle \text{proof} \rangle$

lemma *partition-rhs*: $\text{partition}(a,b) \ D \implies (D(\text{psize } D) = b)$

$\langle \text{proof} \rangle$

lemma *partition-rhs2*: $[[\text{partition}(a,b) \ D; \text{psize } D \leq n]] \implies (D \ n = b)$

$\langle \text{proof} \rangle$

lemma *lemma-partition-lt-gen* [*rule-format*]:

$\text{partition}(a,b) \ D \ \& \ m + \text{Suc } d \leq n \ \& \ n \leq (\text{psize } D) \longrightarrow D(m) < D(m + \text{Suc } d)$

$\langle \text{proof} \rangle$

lemma *less-eq-add-Suc*: $m < n \implies \exists d. n = m + \text{Suc } d$

$\langle \text{proof} \rangle$

lemma *partition-lt-gen*:

$[[\text{partition}(a,b) \ D; m < n; n \leq (\text{psize } D)]] \implies D(m) < D(n)$

$\langle \text{proof} \rangle$

lemma *partition-lt*: $\text{partition}(a,b) \ D \implies n < (\text{psize } D) \implies D(0) < D(\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma *partition-le*: $\text{partition}(a,b) \ D \implies a \leq b$
 $\langle \text{proof} \rangle$

lemma *partition-gt*: $[\text{partition}(a,b) \ D; n < (\text{psize } D)] \implies D(n) < D(\text{psize } D)$
 $\langle \text{proof} \rangle$

lemma *partition-eq*: $\text{partition}(a,b) \ D \implies ((a = b) = (\text{psize } D = 0))$
 $\langle \text{proof} \rangle$

lemma *partition-lb*: $\text{partition}(a,b) \ D \implies a \leq D(r)$
 $\langle \text{proof} \rangle$

lemma *partition-lb-lt*: $[\text{partition}(a,b) \ D; \text{psize } D \sim 0] \implies a < D(\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma *partition-ub*: $\text{partition}(a,b) \ D \implies D(r) \leq b$
 $\langle \text{proof} \rangle$

lemma *partition-ub-lt*: $[\text{partition}(a,b) \ D; n < \text{psize } D] \implies D(n) < b$
 $\langle \text{proof} \rangle$

lemma *lemma-partition-append1*:

$$[\text{partition } (a, b) \ D1; \text{partition } (b, c) \ D2] \implies (\forall n < \text{psize } D1 + \text{psize } D2.$$

$$\quad (\text{if } n < \text{psize } D1 \text{ then } D1 \ n \text{ else } D2 \ (n - \text{psize } D1))$$

$$\quad < (\text{if } \text{Suc } n < \text{psize } D1 \text{ then } D1 \ (\text{Suc } n)$$

$$\quad \text{else } D2 \ (\text{Suc } n - \text{psize } D1))) \ \&$$

$$(\forall n \geq \text{psize } D1 + \text{psize } D2.$$

$$\quad (\text{if } n < \text{psize } D1 \text{ then } D1 \ n \text{ else } D2 \ (n - \text{psize } D1)) =$$

$$\quad (\text{if } \text{psize } D1 + \text{psize } D2 < \text{psize } D1 \text{ then } D1 \ (\text{psize } D1 + \text{psize } D2)$$

$$\quad \text{else } D2 \ (\text{psize } D1 + \text{psize } D2 - \text{psize } D1)))$$
 $\langle \text{proof} \rangle$

lemma *lemma-psize1*:

$$[\text{partition } (a, b) \ D1; \text{partition } (b, c) \ D2; N < \text{psize } D1] \implies D1(N) < D2 \ (\text{psize } D2)$$
 $\langle \text{proof} \rangle$

lemma *lemma-partition-append2*:

$$[\text{partition } (a, b) \ D1; \text{partition } (b, c) \ D2] \implies \text{psize } (\%n. \text{if } n < \text{psize } D1 \text{ then } D1 \ n \text{ else } D2 \ (n - \text{psize } D1)) =$$

$$\text{psize } D1 + \text{psize } D2$$
 $\langle \text{proof} \rangle$

lemma *tpart-eq-lhs-rhs*: $[\text{psize } D = 0; \text{tpart}(a,b) \ (D,p)] \implies a = b$
 $\langle \text{proof} \rangle$

lemma *tpart-partition*: $tpart(a,b) (D,p) ==> partition(a,b) D$
 <proof>

lemma *partition-append*:
 $[| tpart(a,b) (D1,p1); fine(g) (D1,p1);$
 $tpart(b,c) (D2,p2); fine(g) (D2,p2) |]$
 $==> \exists D p. tpart(a,c) (D,p) \ \& \ fine(g) (D,p)$
 <proof>

We can always find a division that is fine wrt any gauge

lemma *partition-exists*:
 $[| a \leq b; gauge(\%x. a \leq x \ \& \ x \leq b) g |]$
 $==> \exists D p. tpart(a,b) (D,p) \ \& \ fine g (D,p)$
 <proof>

Lemmas about combining gauges

lemma *gauge-min*:
 $[| gauge(E) g1; gauge(E) g2 |]$
 $==> gauge(E) (\%x. if g1(x) < g2(x) then g1(x) else g2(x))$
 <proof>

lemma *fine-min*:
 $fine (\%x. if g1(x) < g2(x) then g1(x) else g2(x)) (D,p)$
 $==> fine(g1) (D,p) \ \& \ fine(g2) (D,p)$
 <proof>

The integral is unique if it exists

lemma *Integral-unique*:
 $[| a \leq b; Integral(a,b) f k1; Integral(a,b) f k2 |] ==> k1 = k2$
 <proof>

lemma *Integral-zero [simp]*: $Integral(a,a) f 0$
 <proof>

lemma *sumr-partition-eq-diff-bounds [simp]*:
 $(\sum n=0..<m. D (Suc n) - D n::real) = D(m) - D 0$
 <proof>

lemma *Integral-eq-diff-bounds*: $a \leq b ==> Integral(a,b) (\%x. 1) (b - a)$
 <proof>

lemma *Integral-mult-const*: $a \leq b ==> Integral(a,b) (\%x. c) (c*(b - a))$
 <proof>

lemma *Integral-mult*:
 $[| a \leq b; Integral(a,b) f k |] ==> Integral(a,b) (\%x. c * f x) (c * k)$
 <proof>

Fundamental theorem of calculus (Part I)

”Straddle Lemma” : Swartz and Thompson: AMM 95(7) 1988

lemma *choiceP*: $\forall x. P(x) \dashv\vdash (\exists y. Q\ x\ y) \implies \exists f. (\forall x. P(x) \dashv\vdash Q\ x\ (f\ x))$
 <proof>

lemma *strad1*:

$$\begin{aligned} & \llbracket \forall xa::real. xa \neq x \wedge |xa - x| < s \longrightarrow \\ & \quad |(f\ xa - f\ x) / (xa - x) - f'\ x| * 2 < e; \\ & \quad 0 < e; a \leq x; x \leq b; 0 < s \rrbracket \\ & \implies \forall z. |z - x| < s \dashv\vdash |f\ z - f\ x - f'\ x * (z - x)| * 2 \leq e * |z - x| \end{aligned}$$

 <proof>

lemma *lemma-straddle*:

$$\begin{aligned} & \llbracket \forall x. a \leq x \ \& \ x \leq b \dashv\vdash DERIV\ f\ x :> f'(x); 0 < e \rrbracket \\ & \implies \exists g. gauge(\%x. a \leq x \ \& \ x \leq b)\ g \ \& \\ & \quad (\forall x\ u\ v. a \leq u \ \& \ u \leq x \ \& \ x \leq v \ \& \ v \leq b \ \& \ (v - u) < g(x) \\ & \quad \dashv\vdash |(f(v) - f(u)) - (f'(x) * (v - u))| \leq e * (v - u)) \end{aligned}$$

 <proof>

lemma *FTC1*: $\llbracket a \leq b; \forall x. a \leq x \ \& \ x \leq b \dashv\vdash DERIV\ f\ x :> f'(x) \rrbracket$
 $\implies Integral(a,b)\ f' (f(b) - f(a))$
 <proof>

lemma *Integral-subst*: $\llbracket Integral(a,b)\ f\ k1; k2=k1 \rrbracket \implies Integral(a,b)\ f\ k2$
 <proof>

lemma *Integral-add*:

$$\begin{aligned} & \llbracket a \leq b; b \leq c; Integral(a,b)\ f'\ k1; Integral(b,c)\ f'\ k2; \\ & \quad \forall x. a \leq x \ \& \ x \leq c \dashv\vdash DERIV\ f\ x :> f'\ x \rrbracket \\ & \implies Integral(a,c)\ f' (k1 + k2) \end{aligned}$$

 <proof>

lemma *partition-psize-Least*:

$partition(a,b)\ D \implies psize\ D = (LEAST\ n. D(n) = b)$
 <proof>

lemma *lemma-partition-bounded*: $partition\ (a, c)\ D \implies \sim (\exists n. c < D(n))$
 <proof>

lemma *lemma-partition-eq*:

$partition\ (a, c)\ D \implies D = (\%n. if\ D\ n < c\ then\ D\ n\ else\ c)$
 <proof>

lemma *lemma-partition-eq2:*

$\text{partition } (a, c) \ D \implies D = (\%n. \text{ if } D \ n \leq c \text{ then } D \ n \text{ else } c)$
 $\langle \text{proof} \rangle$

lemma *partition-lt-Suc:*

$\llbracket \text{partition}(a, b) \ D; \ n < \text{psize } D \rrbracket \implies D \ n < D \ (\text{Suc } n)$
 $\langle \text{proof} \rangle$

lemma *tpart-tag-eq:* $\text{tpart}(a, c) \ (D, p) \implies p = (\%n. \text{ if } D \ n < c \text{ then } p \ n \text{ else } c)$

$\langle \text{proof} \rangle$

69.1 Lemmas for Additivity Theorem of Gauge Integral

lemma *lemma-additivity1:*

$\llbracket a \leq D \ n; \ D \ n < b; \ \text{partition}(a, b) \ D \rrbracket \implies n < \text{psize } D$
 $\langle \text{proof} \rangle$

lemma *lemma-additivity2:* $\llbracket a \leq D \ n; \ \text{partition}(a, D \ n) \ D \rrbracket \implies \text{psize } D \leq n$

$\langle \text{proof} \rangle$

lemma *partition-eq-bound:*

$\llbracket \text{partition}(a, b) \ D; \ \text{psize } D < m \rrbracket \implies D(m) = D(\text{psize } D)$
 $\langle \text{proof} \rangle$

lemma *partition-ub2:* $\llbracket \text{partition}(a, b) \ D; \ \text{psize } D < m \rrbracket \implies D(r) \leq D(m)$

$\langle \text{proof} \rangle$

lemma *tag-point-eq-partition-point:*

$\llbracket \text{tpart}(a, b) \ (D, p); \ \text{psize } D \leq m \rrbracket \implies p(m) = D(m)$
 $\langle \text{proof} \rangle$

lemma *partition-lt-cancel:* $\llbracket \text{partition}(a, b) \ D; \ D \ m < D \ n \rrbracket \implies m < n$

$\langle \text{proof} \rangle$

lemma *lemma-additivity4-psize-eq:*

$\llbracket a \leq D \ n; \ D \ n < b; \ \text{partition } (a, b) \ D \rrbracket$
 $\implies \text{psize } (\%x. \text{ if } D \ x < D \ n \text{ then } D(x) \text{ else } D \ n) = n$
 $\langle \text{proof} \rangle$

lemma *lemma-psize-left-less-psize:*

$\text{partition } (a, b) \ D$
 $\implies \text{psize } (\%x. \text{ if } D \ x < D \ n \text{ then } D(x) \text{ else } D \ n) \leq \text{psize } D$
 $\langle \text{proof} \rangle$

lemma *lemma-psize-left-less-psize2:*

$\llbracket \text{partition}(a, b) \ D; \ na < \text{psize } (\%x. \text{ if } D \ x < D \ n \text{ then } D(x) \text{ else } D \ n) \rrbracket$
 $\implies na < \text{psize } D$
 $\langle \text{proof} \rangle$

lemma *lemma-additivity3*:

[[*partition*(*a,b*) *D*; *D na* < *D n*; *D n* < *D (Suc na)*;
 n < *psize D*]]
 ==> *False*
 <proof>

lemma *psize-const [simp]*: *psize* (%*x. k*) = 0

<proof>

lemma *lemma-additivity3a*:

[[*partition*(*a,b*) *D*; *D na* < *D n*; *D n* < *D (Suc na)*;
 na < *psize D*]]
 ==> *False*
 <proof>

lemma *better-lemma-psize-right-eq1*:

[[*partition*(*a,b*) *D*; *D n* < *b*]] ==> *psize* (%*x. D (x + n)*) ≤ *psize D* - *n*
 <proof>

lemma *psize-le-n*: *partition (a, D n) D* ==> *psize D* ≤ *n*

<proof>

lemma *better-lemma-psize-right-eq1a*:

partition(a,D n) D ==> *psize* (%*x. D (x + n)*) ≤ *psize D* - *n*
 <proof>

lemma *better-lemma-psize-right-eq*:

partition(a,b) D ==> *psize* (%*x. D (x + n)*) ≤ *psize D* - *n*
 <proof>

lemma *lemma-psize-right-eq1*:

[[*partition*(*a,b*) *D*; *D n* < *b*]] ==> *psize* (%*x. D (x + n)*) ≤ *psize D*
 <proof>

lemma *lemma-psize-right-eq1a*:

partition(a,D n) D ==> *psize* (%*x. D (x + n)*) ≤ *psize D*
 <proof>

lemma *lemma-psize-right-eq*:

[[*partition*(*a,b*) *D*]] ==> *psize* (%*x. D (x + n)*) ≤ *psize D*
 <proof>

lemma *tpart-left1*:

[[*a* ≤ *D n*; *tpart (a, b) (D, p)*]]
 ==> *tpart(a, D n) (%x. if D x < D n then D(x) else D n,*

$\%x. \text{ if } D\ x < D\ n \text{ then } p(x) \text{ else } D\ n)$
 $\langle \text{proof} \rangle$

lemma *fine-left1*:

$[[\ a \leq D\ n; \text{tpart}(a, b)\ (D, p); \text{gauge } (\%x. a \leq x \ \& \ x \leq D\ n) \ g;$
 $\text{fine } (\%x. \text{ if } x < D\ n \text{ then } \min(g\ x) ((D\ n - x)/\ 2)$
 $\text{else if } x = D\ n \text{ then } \min(g\ (D\ n)) (ga\ (D\ n))$
 $\text{else } \min(ga\ x) ((x - D\ n)/\ 2))\ (D, p)\]]$
 $\implies \text{fine } g$
 $(\%x. \text{ if } D\ x < D\ n \text{ then } D(x) \text{ else } D\ n,$
 $\%x. \text{ if } D\ x < D\ n \text{ then } p(x) \text{ else } D\ n)$
 $\langle \text{proof} \rangle$

lemma *tpart-right1*:

$[[\ a \leq D\ n; \text{tpart}(a, b)\ (D, p)\]]$
 $\implies \text{tpart}(D\ n, b)\ (\%x. D(x + n), \%x. p(x + n))$
 $\langle \text{proof} \rangle$

lemma *fine-right1*:

$[[\ a \leq D\ n; \text{tpart}(a, b)\ (D, p); \text{gauge } (\%x. D\ n \leq x \ \& \ x \leq b) \ ga;$
 $\text{fine } (\%x. \text{ if } x < D\ n \text{ then } \min(g\ x) ((D\ n - x)/\ 2)$
 $\text{else if } x = D\ n \text{ then } \min(g\ (D\ n)) (ga\ (D\ n))$
 $\text{else } \min(ga\ x) ((x - D\ n)/\ 2))\ (D, p)\]]$
 $\implies \text{fine } ga\ (\%x. D(x + n), \%x. p(x + n))$
 $\langle \text{proof} \rangle$

lemma *rsum-add*: $rsum\ (D, p)\ (\%x. f\ x + g\ x) = rsum\ (D, p)\ f + rsum(D, p)\ g$
 $\langle \text{proof} \rangle$

Bartle/Sherbert: Theorem 10.1.5 p. 278

lemma *Integral-add-fun*:

$[[\ a \leq b; \text{Integral}(a, b)\ f\ k1; \text{Integral}(a, b)\ g\ k2\]]$
 $\implies \text{Integral}(a, b)\ (\%x. f\ x + g\ x)\ (k1 + k2)$
 $\langle \text{proof} \rangle$

lemma *partition-lt-gen2*:

$[[\ \text{partition}(a, b)\ D; r < psize\ D\]] \implies 0 < D\ (Suc\ r) - D\ r$
 $\langle \text{proof} \rangle$

lemma *lemma-Integral-le*:

$[[\ \forall x. a \leq x \ \& \ x \leq b \implies f\ x \leq g\ x;$
 $\text{tpart}(a, b)\ (D, p)$
 $]] \implies \forall n \leq psize\ D. f\ (p\ n) \leq g\ (p\ n)$
 $\langle \text{proof} \rangle$

lemma *lemma-Integral-rsum-le*:

$[[\ \forall x. a \leq x \ \& \ x \leq b \implies f\ x \leq g\ x;$
 $\text{tpart}(a, b)\ (D, p)$

$|| \implies rsum(D,p) f \leq rsum(D,p) g$
 $\langle proof \rangle$

lemma *Integral-le*:

$|| a \leq b;$
 $\forall x. a \leq x \ \& \ x \leq b \implies f(x) \leq g(x);$
 $Integral(a,b) f \ k1; Integral(a,b) g \ k2$
 $|| \implies k1 \leq k2$
 $\langle proof \rangle$

lemma *Integral-imp-Cauchy*:

$(\exists k. Integral(a,b) f \ k) \implies$
 $(\forall e > 0. \exists g. gauge (\%x. a \leq x \ \& \ x \leq b) g \ \&$
 $(\forall D1 \ D2 \ p1 \ p2.$
 $tpart(a,b) (D1, p1) \ \& \ fine \ g \ (D1,p1) \ \&$
 $tpart(a,b) (D2, p2) \ \& \ fine \ g \ (D2,p2) \implies$
 $|rsum(D1,p1) f - rsum(D2,p2) f| < e))$
 $\langle proof \rangle$

lemma *Cauchy-iff2*:

$Cauchy \ X =$
 $(\forall j. (\exists M. \forall m \geq M. \forall n \geq M. |X \ m - X \ n| < inverse(real (Suc j))))$
 $\langle proof \rangle$

lemma *partition-exists2*:

$|| a \leq b; \forall n. gauge (\%x. a \leq x \ \& \ x \leq b) (fa \ n) ||$
 $\implies \forall n. \exists D \ p. tpart(a, b) (D, p) \ \& \ fine (fa \ n) (D, p)$
 $\langle proof \rangle$

lemma *monotonic-anti-derivative*:

fixes $f \ g :: real \implies real$ **shows**
 $|| a \leq b; \forall c. a \leq c \ \& \ c \leq b \implies f' \ c \leq g' \ c;$
 $\forall x. DERIV f \ x :> f' \ x; \forall x. DERIV g \ x :> g' \ x ||$
 $\implies f \ b - f \ a \leq g \ b - g \ a$
 $\langle proof \rangle$

end

70 Complex-Main: Comprehensive Complex Theory

theory *Complex-Main*

imports

Main

Real

Complex

Log

Ln
Taylor
Integration
begin

end

References

- [1] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.