

Measure and Probability Theory

October 8, 2017

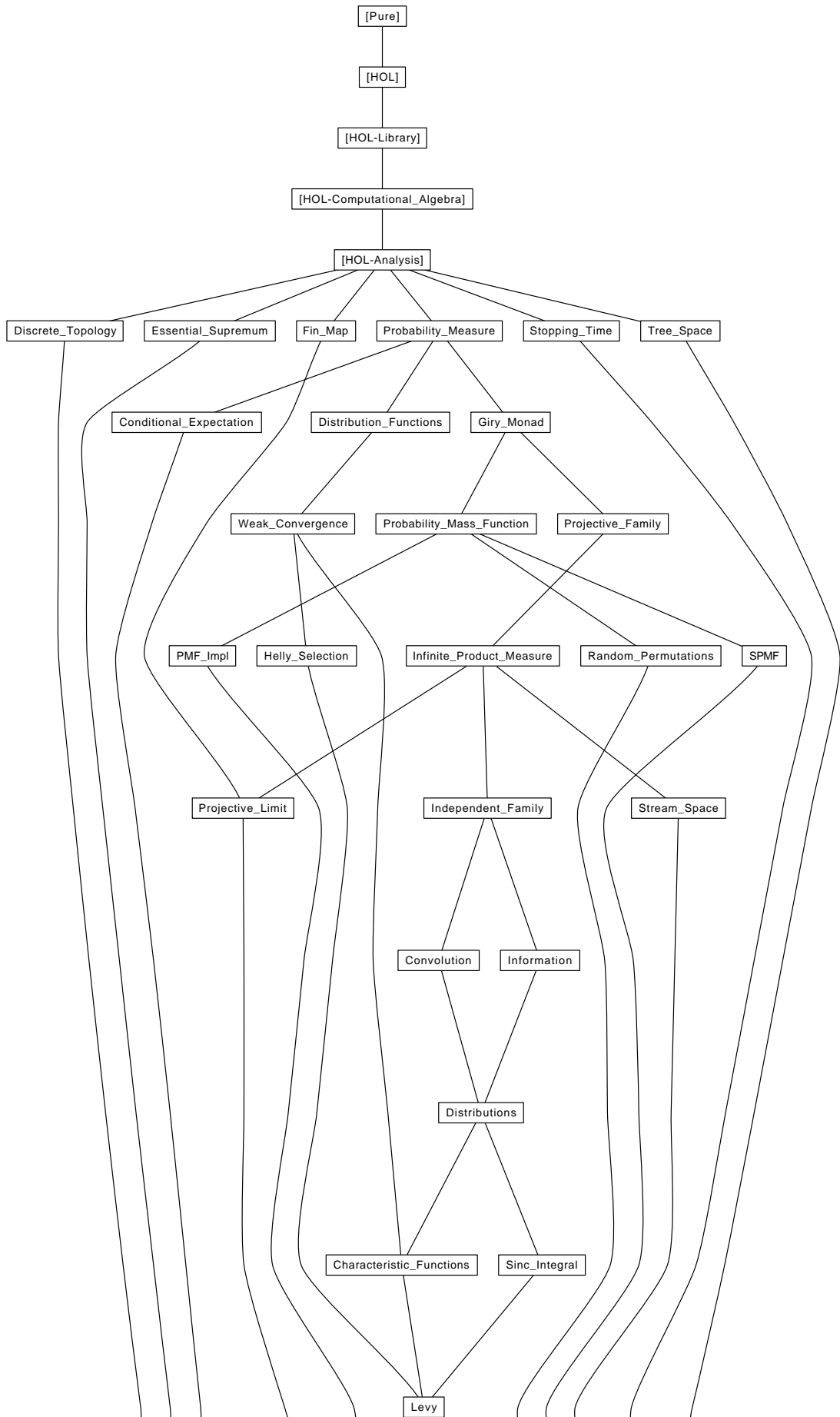
Contents

1	Probability measure	6
1.1	Introduce binder for probability	11
1.2	Distributions	16
2	Distribution Functions	33
2.1	Properties of cdf's	33
2.2	Uniqueness	36
3	Weak Convergence of Functions and Distributions	39
4	Weak Convergence of Functions	39
5	Weak Convergence of Distributions	39
6	Skorohod's theorem	40
7	Sub-probability spaces	48
8	Properties of return	60
9	Join	65
9.1	Giry monad on probability spaces	83
10	Projective Family	88
11	Infinite Product Measure	103
11.1	Sequence space	109
12	Independent families of events, event sets, and random variables	113
13	Convolution Measure	143

14 Information theory	148
14.1 Information theory	148
14.2 Kullback–Leibler divergence	149
14.3 Finite Entropy	155
14.4 Mutual Information	157
14.5 Entropy	165
14.6 Conditional Mutual Information	169
14.7 Conditional Entropy	182
14.8 Equalities	187
15 Properties of Various Distributions	192
15.1 Erlang	194
15.2 Exponential distribution	200
15.3 Uniform distribution	207
15.4 Normal distribution	211
16 Characteristic Functions	225
16.1 Application of the FTC: integrating e^{ix}	225
16.2 The Characteristic Function of a Real Measure.	226
16.3 Independence	227
16.4 Approximations to e^{ix}	227
16.5 Calculation of the Characteristic Function of the Standard Distribution	235
17 Helly’s selection theorem	237
18 Integral of sinc	244
18.1 Various preparatory integrals	244
19 The sinc function, and the sine integral (Si)	246
19.1 The final theorems: boundedness and scalability	251
20 The Levy inversion theorem, and the Levy continuity theo- rem.	253
20.1 The Levy inversion theorem	253
20.2 The Levy continuity theorem	259
21 The Central Limit Theorem	265
22 Probability mass function	269
22.1 PMF as measure	271
22.2 Monad Interpretation	276
22.3 PMFs as function	285
22.4 Conditional Probabilities	294
22.5 Relator	296

22.6	Distributions	306
22.6.1	Bernoulli Distribution	306
22.6.2	Geometric Distribution	307
22.6.3	Uniform Multiset Distribution	308
22.6.4	Uniform Distribution	308
22.6.5	Poisson Distribution	311
22.6.6	Binomial Distribution	312
22.7	PMFs from association lists	314
23	Code generation for PMFs	319
23.1	General code generation setup	319
23.2	Code abbreviations for integrals and probabilities	329
24	Finite Maps	331
24.1	Domain and Application	331
24.2	Constructor of Finite Maps	332
24.3	Product set of Finite Maps	333
24.3.1	Basic Properties of Pi'	333
24.4	Topological Space of Finite Maps	334
24.5	Metric Space of Finite Maps	336
24.6	Complete Space of Finite Maps	340
24.7	Second Countable Space of Finite Maps	342
24.8	Polish Space of Finite Maps	344
24.9	Product Measurable Space of Finite Maps	344
24.10	Isomorphism between Functions and Finite Maps	356
25	Projective Limit	360
25.1	Sequences of Finite Maps in Compact Sets	360
25.2	Daniell-Kolmogorov Theorem	362
26	Random Permutations	371
27	Discrete subprobability distribution	375
27.1	Auxiliary material	375
27.1.1	More about extended reals	375
27.1.2	More about <i>'a option</i>	376
27.1.3	A relator for sets that treats sets like predicates	379
27.1.4	Monotonicity rules	379
27.1.5	Bijections	380
27.2	Subprobability mass function	380
27.3	Support	383
27.4	Functorial structure	385
27.5	Monad operations	387
27.5.1	Return	387

27.5.2 Bind	387
27.6 Relator	391
27.7 From 'a pmf to 'a spmf	394
27.8 Weight of a subprobability	395
27.9 From density to spmfs	397
27.10 Ordering on spmfs	398
27.11 CCPO structure for the flat ccpo <i>ord-option op =</i>	404
27.11.1 Admissibility of <i>rel-spmf</i>	415
27.12 Restrictions on spmfs	418
27.13 Subprobability distributions of sets	420
27.14 Losslessness	424
27.15 Scaling	426
27.16 Conditional spmfs	431
27.17 Product spmf	432
27.18 Assertions	436
27.19 Try	437
27.20 Miscellaneous	439
28 Conditional Expectation	465
28.1 Restricting a measure to a sub-sigma-algebra	465
28.2 Nonnegative conditional expectation	469
28.3 Real conditional expectation	475
29 The essential supremum	497
30 Stopping times	500
30.1 Stopping Time	500
31 Filtration	501
31.1 σ -algebra of a Stopping Time	501



1 Probability measure

```
theory Probability-Measure
  imports HOL-Analysis.Analysis
begin
```

```
locale prob-space = finite-measure +
  assumes emeasure-space-1: emeasure M (space M) = 1
```

```
lemma prob-spaceI[Pure.intro!]:
  assumes *: emeasure M (space M) = 1
  shows prob-space M
proof -
  interpret finite-measure M
  proof
    show emeasure M (space M)  $\neq$   $\infty$  using * by simp
  qed
  show prob-space M by standard fact
qed
```

```
lemma prob-space-imp-sigma-finite: prob-space M  $\implies$  sigma-finite-measure M
  unfolding prob-space-def finite-measure-def by simp
```

```
abbreviation (in prob-space) events  $\equiv$  sets M
abbreviation (in prob-space) prob  $\equiv$  measure M
abbreviation (in prob-space) random-variable  $M' X \equiv X \in$  measurable  $M M'$ 
abbreviation (in prob-space) expectation  $\equiv$  integralL M
abbreviation (in prob-space) variance  $X \equiv$  integralL M ( $\lambda x. (X x -$  expectation  $X)^2$ )
```

```
lemma (in prob-space) finite-measure [simp]: finite-measure M
  by unfold-locales
```

```
lemma (in prob-space) prob-space-distr:
  assumes f:  $f \in$  measurable  $M M'$  shows prob-space (distr M M' f)
proof (rule prob-spaceI)
  have  $f - 'space M' \cap space M = space M$  using f by (auto dest: measurable-space)
  with f show emeasure (distr M M' f) (space (distr M M' f)) = 1
  by (auto simp: emeasure-distr emeasure-space-1)
qed
```

```
lemma prob-space-distrD:
  assumes f:  $f \in$  measurable  $M N$  and M: prob-space (distr M N f) shows
  prob-space M
proof
  interpret M: prob-space distr M N f by fact
  have  $f - 'space N \cap space M = space M$ 
  using f[THEN measurable-space] by auto
  then show emeasure M (space M) = 1
```

using $M.emeasure-space-1$ by (simp add: emeasure-distr[OF f])
qed

lemma (in prob-space) prob-space: prob (space M) = 1
using emeasure-space-1 unfolding measure-def by (simp add: one-ennreal.rep-eq)

lemma (in prob-space) prob-le-1[simp, intro]: prob A ≤ 1
using bounded-measure[of A] by (simp add: prob-space)

lemma (in prob-space) not-empty: space M ≠ {}
using prob-space by auto

lemma (in prob-space) emeasure-eq-1-AE:
 $S \in sets M \implies AE x \text{ in } M. x \in S \implies emeasure M S = 1$
by (subst emeasure-eq-AE[where B=space M]) (auto simp: emeasure-space-1)

lemma (in prob-space) emeasure-le-1: emeasure M S ≤ 1
unfolding ennreal-1[symmetric] emeasure-eq-measure by (subst ennreal-le-iff)
auto

lemma (in prob-space) emeasure-ge-1-iff: emeasure M A ≥ 1 \longleftrightarrow emeasure M A = 1
by (rule iffI, intro antisym emeasure-le-1) simp-all

lemma (in prob-space) AE-iff-emeasure-eq-1:
assumes [measurable]: Measurable.pred M P
shows (AE x in M. P x) \longleftrightarrow emeasure M {x ∈ space M. P x} = 1
proof –
have *: {x ∈ space M. ¬ P x} = space M - {x ∈ space M. P x}
by auto
show ?thesis
by (auto simp add: ennreal-minus-eq-0 * emeasure-compl emeasure-space-1
AE-iff-measurable[OF - refl]
intro: antisym emeasure-le-1)

qed

lemma (in prob-space) measure-le-1: emeasure M X ≤ 1
using emeasure-space[of M X] by (simp add: emeasure-space-1)

lemma (in prob-space) measure-ge-1-iff: measure M A ≥ 1 \longleftrightarrow measure M A = 1
by (auto intro!: antisym)

lemma (in prob-space) AE-I-eq-1:
assumes emeasure M {x ∈ space M. P x} = 1 {x ∈ space M. P x} ∈ sets M
shows AE x in M. P x
proof (rule AE-I)
show emeasure M (space M - {x ∈ space M. P x}) = 0
using assms emeasure-space-1 by (simp add: emeasure-compl)

qed (*insert assms, auto*)

lemma *prob-space-restrict-space*:

$S \in \text{sets } M \implies \text{emeasure } M S = 1 \implies \text{prob-space } (\text{restrict-space } M S)$

by (*intro prob-spaceI*)

(*simp add: emeasure-restrict-space space-restrict-space*)

lemma (*in prob-space*) *prob-compl*:

assumes $A: A \in \text{events}$

shows $\text{prob } (\text{space } M - A) = 1 - \text{prob } A$

using *finite-measure-compl[OF A]* **by** (*simp add: prob-space*)

lemma (*in prob-space*) *AE-in-set-eq-1*:

assumes $A[\text{measurable}]: A \in \text{events}$ **shows** $(\text{AE } x \text{ in } M. x \in A) \longleftrightarrow \text{prob } A = 1$

proof –

have $*$: $\{x \in \text{space } M. x \in A\} = A$

using $A[\text{THEN sets.sets-into-space}]$ **by** *auto*

show *?thesis*

by (*subst AE-iff-emeasure-eq-1*) (*auto simp: emeasure-eq-measure **)

qed

lemma (*in prob-space*) *AE-False*: $(\text{AE } x \text{ in } M. \text{False}) \longleftrightarrow \text{False}$

proof

assume $\text{AE } x \text{ in } M. \text{False}$

then have $\text{AE } x \text{ in } M. x \in \{\}$ **by** *simp*

then show *False*

by (*subst (asm) AE-in-set-eq-1*) *auto*

qed *simp*

lemma (*in prob-space*) *AE-prob-1*:

assumes $\text{prob } A = 1$ **shows** $\text{AE } x \text{ in } M. x \in A$

proof –

from $\langle \text{prob } A = 1 \rangle$ **have** $A \in \text{events}$

by (*metis measure-notin-sets zero-neq-one*)

with *AE-in-set-eq-1* *assms* **show** *?thesis* **by** *simp*

qed

lemma (*in prob-space*) *AE-const[*simp*]*: $(\text{AE } x \text{ in } M. P) \longleftrightarrow P$

by (*cases P*) (*auto simp: AE-False*)

lemma (*in prob-space*) *ae-filter-bot*: $\text{ae-filter } M \neq \text{bot}$

by (*simp add: trivial-limit-def*)

lemma (*in prob-space*) *AE-contr*:

assumes $\text{ae}: \text{AE } \omega \text{ in } M. P \ \omega \ \text{AE } \omega \text{ in } M. \neg P \ \omega$

shows *False*

proof –

from *ae* **have** $\text{AE } \omega \text{ in } M. \text{False}$ **by** *eventually-elim auto*

then show *False* **by** *auto*
qed

lemma (**in** *prob-space*) *integral-ge-const*:
fixes $c :: \text{real}$
shows $\text{integrable } M f \implies (\text{AE } x \text{ in } M. c \leq f x) \implies c \leq (\int x. f x \partial M)$
using *integral-mono-AE*[of $M \lambda x. c f$] *prob-space* **by** *simp*

lemma (**in** *prob-space*) *integral-le-const*:
fixes $c :: \text{real}$
shows $\text{integrable } M f \implies (\text{AE } x \text{ in } M. f x \leq c) \implies (\int x. f x \partial M) \leq c$
using *integral-mono-AE*[of $M f \lambda x. c$] *prob-space* **by** *simp*

lemma (**in** *prob-space*) *nn-integral-ge-const*:
 $(\text{AE } x \text{ in } M. c \leq f x) \implies c \leq (\int^+ x. f x \partial M)$
using *nn-integral-mono-AE*[of $\lambda x. c f M$] *emeasure-space-1*
by (*simp split: if-split-asm*)

lemma (**in** *prob-space*) *expectation-less*:
fixes $X :: - \Rightarrow \text{real}$
assumes [*simp*]: *integrable* $M X$
assumes *gt*: $\text{AE } x \text{ in } M. X x < b$
shows *expectation* $X < b$
proof –
have *expectation* $X < \text{expectation } (\lambda x. b)$
using *gt emeasure-space-1*
by (*intro integral-less-AE-space*) *auto*
then show *?thesis* **using** *prob-space* **by** *simp*
qed

lemma (**in** *prob-space*) *expectation-greater*:
fixes $X :: - \Rightarrow \text{real}$
assumes [*simp*]: *integrable* $M X$
assumes *gt*: $\text{AE } x \text{ in } M. a < X x$
shows $a < \text{expectation } X$
proof –
have *expectation* $(\lambda x. a) < \text{expectation } X$
using *gt emeasure-space-1*
by (*intro integral-less-AE-space*) *auto*
then show *?thesis* **using** *prob-space* **by** *simp*
qed

lemma (**in** *prob-space*) *jensens-inequality*:
fixes $q :: \text{real} \Rightarrow \text{real}$
assumes X : *integrable* $M X$ *AE* $x \text{ in } M. X x \in I$
assumes I : $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = \text{UNIV}$
assumes q : *integrable* $M (\lambda x. q (X x))$ *convex-on* $I q$
shows $q (\text{expectation } X) \leq \text{expectation } (\lambda x. q (X x))$
proof –

```

let ?F =  $\lambda x. \text{Inf } ((\lambda t. (q\ x - q\ t) / (x - t)) \text{ ' } (\{x < ..\} \cap I))$ 
from X(2) AE-False have I  $\neq \{\}$  by auto

from I have open I by auto

note I
moreover
{ assume I  $\subseteq \{a < ..\}$ 
  with X have a < expectation X
  by (intro expectation-greater) auto }
moreover
{ assume I  $\subseteq \{.. < b\}$ 
  with X have expectation X < b
  by (intro expectation-less) auto }
ultimately have expectation X  $\in I$ 
  by (elim disjE) (auto simp: subset-eq)
moreover
{ fix y assume y: y  $\in I$ 
  with q(2) <open I> have Sup (( $\lambda x. q\ x + ?F\ x * (y - x)$ ) ' I) = q y
  by (auto intro!: cSup-eq-maximum convex-le-Inf-differential image-eqI [OF -
y] simp: interior-open) }
ultimately have q (expectation X) = Sup (( $\lambda x. q\ x + ?F\ x * (\text{expectation X} - x)$ ) ' I)
  by simp
also have ...  $\leq$  expectation ( $\lambda w. q\ (X\ w)$ )
proof (rule cSup-least)
  show ( $\lambda x. q\ x + ?F\ x * (\text{expectation X} - x)$ ) ' I  $\neq \{\}$ 
    using <I  $\neq \{\}$ > by auto
next
  fix k assume k  $\in (\lambda x. q\ x + ?F\ x * (\text{expectation X} - x))$  ' I
  then guess x .. note x = this
  have q x + ?F x * (expectation X - x) = expectation ( $\lambda w. q\ x + ?F\ x * (X\ w - x)$ )
    using prob-space by (simp add: X)
  also have ...  $\leq$  expectation ( $\lambda w. q\ (X\ w)$ )
    using <x  $\in I$ > <open I> X(2)
  apply (intro integral-mono-AE Bochner-Integration.integrable-add Bochner-Integration.integrable-mult-right
Bochner-Integration.integrable-diff
  integrable-const X q)
  apply (elim eventually-mono)
  apply (intro convex-le-Inf-differential)
  apply (auto simp: interior-open q)
  done
  finally show k  $\leq$  expectation ( $\lambda w. q\ (X\ w)$ ) using x by auto
qed
finally show q (expectation X)  $\leq$  expectation ( $\lambda x. q\ (X\ x)$ ) .
qed

```

1.1 Introduce binder for probability

syntax

$-prob :: ptnrn \Rightarrow logic \Rightarrow logic \Rightarrow logic ((\mathcal{P}'((- in -./ -)))$

translations

$\mathcal{P}(x \text{ in } M. P) \Rightarrow CONST \text{ measure } M \{x \in CONST \text{ space } M. P\}$

print-translation <

```

let
  fun to-pattern (Const (@{const-syntax Pair}, -) $ l $ r) =
    Syntax.const @{const-syntax Pair} :: to-pattern l @ to-pattern r
  | to-pattern (t as (Const (@{syntax-const -bound}, -)) $ -) = [t]

  fun mk-pattern ((t, n) :: xs) = mk-patterns n xs |>> curry list-comb t
  and mk-patterns 0 xs = ([], xs)
  | mk-patterns n xs =
    let
      val (t, xs') = mk-pattern xs
      val (ts, xs'') = mk-patterns (n - 1) xs'
    in
      (t :: ts, xs'')
    end

  fun unnest-tuples
    (Const (@{syntax-const -pattern}, -) $
      t1 $
      (t as (Const (@{syntax-const -pattern}, -) $ - $ -)))
  = let
    val (- $ t2 $ t3) = unnest-tuples t
  in
    Syntax.const @{syntax-const -pattern} $
      unnest-tuples t1 $
      (Syntax.const @{syntax-const -patterns} $ t2 $ t3)
  end
  | unnest-tuples pat = pat

  fun tr' [sig-alg, Const (@{const-syntax Collect}, -) $ t] =
    let
      val bound-dummyT = Const (@{syntax-const -bound}, dummyT)
    in
      fun go pattern elem
        (Const (@{const-syntax conj}, -) $
          (Const (@{const-syntax Set.member}, -) $ elem' $ (Const (@{const-syntax
space}, -) $ sig-alg'))) $
          u)
        = let
          val - = if sig-alg aconv sig-alg' andalso to-pattern elem' = rev elem then
            () else raise Match;
          val (pat, rest) = mk-pattern (rev pattern);

```

```

      val - = case rest of [] => () | - => raise Match
    in
      Syntax.const @{\syntax-const -prob} $ unnest-tuples pat $ sig-alg $ u
    end
  | go pattern elem (Abs abs) =
    let
      val (x as (- $ tx), t) = Syntax-Trans.atomic-abs-tr' abs
    in
      go ((x, 0) :: pattern) (bound-dummyT $ tx :: elem) t
    end
  | go pattern elem (Const (@{\const-syntax case-prod}, -) $ t) =
    go
      ((Syntax.const @{\syntax-const -pattern}, 2) :: pattern)
      (Syntax.const @{\const-syntax Pair} :: elem)
      t
    in
      go [] [] t
    end
  in
    [(@{\const-syntax Sigma-Algebra.measure}, K tr')]
  end
)

```

definition

$$\text{cond-prob } M P Q = \mathcal{P}(\omega \text{ in } M. P \omega \wedge Q \omega) / \mathcal{P}(\omega \text{ in } M. Q \omega)$$
syntax

$$\text{-conditional-prob} :: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} ((\mathcal{P}'(- \text{ in } -. - / -')))$$
translations

$$\mathcal{P}(x \text{ in } M. P \mid Q) \Rightarrow \text{CONST cond-prob } M (\lambda x. P) (\lambda x. Q)$$
lemma (in prob-space) AE-E-prob:

assumes *ae*: AE *x* in *M*. *P x*

obtains *S* **where** $S \subseteq \{x \in \text{space } M. P x\}$ *S* ∈ events prob *S* = 1

proof –

from *ae*[THEN AE-E] **guess** *N* .

then show *thesis*

by (*intro that*[of space *M* – *N*])

(*auto simp: prob-compl prob-space emeasure-eq-measure measure-nonneg*)

qed

lemma (in prob-space) prob-neg: $\{x \in \text{space } M. P x\} \in \text{events} \implies \mathcal{P}(x \text{ in } M. \neg P x) = 1 - \mathcal{P}(x \text{ in } M. P x)$

by (*auto intro!: arg-cong[where f=prob] simp add: prob-compl[symmetric]*)

lemma (in prob-space) prob-eq-AE:

(AE *x* in *M*. $P x \longleftrightarrow Q x$) $\implies \{x \in \text{space } M. P x\} \in \text{events} \implies \{x \in \text{space } M. Q x\} \in \text{events} \implies \mathcal{P}(x \text{ in } M. P x) = \mathcal{P}(x \text{ in } M. Q x)$

by (rule finite-measure-eq-AE) auto

lemma (in prob-space) prob-eq-0-AE:

assumes not: AE x in M . $\neg P x$ shows $\mathcal{P}(x \text{ in } M. P x) = 0$

proof cases

assume $\{x \in \text{space } M. P x\} \in \text{events}$

with not have $\mathcal{P}(x \text{ in } M. P x) = \mathcal{P}(x \text{ in } M. \text{False})$

by (intro prob-eq-AE) auto

then show ?thesis by simp

qed (simp add: measure-notin-sets)

lemma (in prob-space) prob-Collect-eq-0:

$\{x \in \text{space } M. P x\} \in \text{sets } M \implies \mathcal{P}(x \text{ in } M. P x) = 0 \iff (AE x \text{ in } M. \neg P x)$

using AE-iff-measurable[OF - refl, of $M \lambda x. \neg P x$] by (simp add: emeasure-eq-measure measure-nonneg)

lemma (in prob-space) prob-Collect-eq-1:

$\{x \in \text{space } M. P x\} \in \text{sets } M \implies \mathcal{P}(x \text{ in } M. P x) = 1 \iff (AE x \text{ in } M. P x)$

using AE-in-set-eq-1[of $\{x \in \text{space } M. P x\}$] by simp

lemma (in prob-space) prob-eq-0:

$A \in \text{sets } M \implies \text{prob } A = 0 \iff (AE x \text{ in } M. x \notin A)$

using AE-iff-measurable[OF - refl, of $M \lambda x. x \notin A$]

by (auto simp add: emeasure-eq-measure Int-def[symmetric] measure-nonneg)

lemma (in prob-space) prob-eq-1:

$A \in \text{sets } M \implies \text{prob } A = 1 \iff (AE x \text{ in } M. x \in A)$

using AE-in-set-eq-1[of A] by simp

lemma (in prob-space) prob-sums:

assumes $P: \bigwedge n. \{x \in \text{space } M. P n x\} \in \text{events}$

assumes $Q: \{x \in \text{space } M. Q x\} \in \text{events}$

assumes ae: AE x in M . $(\forall n. P n x \longrightarrow Q x) \wedge (Q x \longrightarrow (\exists! n. P n x))$

shows $(\lambda n. \mathcal{P}(x \text{ in } M. P n x)) \text{ sums } \mathcal{P}(x \text{ in } M. Q x)$

proof –

from ae[THEN AE-E-prob] guess S . note $S = \text{this}$

then have disj: disjoint-family $(\lambda n. \{x \in \text{space } M. P n x\} \cap S)$

by (auto simp: disjoint-family-on-def)

from S have ae-S:

$AE x \text{ in } M. x \in \{x \in \text{space } M. Q x\} \iff x \in (\bigcup n. \{x \in \text{space } M. P n x\} \cap S)$

$\bigwedge n. AE x \text{ in } M. x \in \{x \in \text{space } M. P n x\} \iff x \in \{x \in \text{space } M. P n x\} \cap S$

using ae by (auto dest!: AE-prob-1)

from ae-S have *:

$\mathcal{P}(x \text{ in } M. Q x) = \text{prob } (\bigcup n. \{x \in \text{space } M. P n x\} \cap S)$

using $P Q S$ by (intro finite-measure-eq-AE) auto

from ae-S have **::

$\bigwedge n. \mathcal{P}(x \text{ in } M. P n x) = \text{prob } (\{x \in \text{space } M. P n x\} \cap S)$

using $P Q S$ by (intro finite-measure-eq-AE) auto

show ?thesis

unfolding * ** **using** $S P$ *disj*
by (*intro finite-measure-UNION*) *auto*
qed

lemma (*in prob-space*) *prob-sum*:

assumes [*simp, intro*]: *finite I*

assumes P : $\bigwedge n. n \in I \implies \{x \in \text{space } M. P \ n \ x\} \in \text{events}$

assumes Q : $\{x \in \text{space } M. Q \ x\} \in \text{events}$

assumes *ae*: $AE \ x \ \text{in } M. (\forall n \in I. P \ n \ x \longrightarrow Q \ x) \wedge (Q \ x \longrightarrow (\exists ! n \in I. P \ n \ x))$

shows $\mathcal{P}(x \ \text{in } M. Q \ x) = (\sum_{n \in I}. \mathcal{P}(x \ \text{in } M. P \ n \ x))$

proof –

from *ae* [*THEN AE-E-prob*] **guess** S . **note** $S = \text{this}$

then have *disj*: *disjoint-family-on* $(\lambda n. \{x \in \text{space } M. P \ n \ x\} \cap S) \ I$

by (*auto simp: disjoint-family-on-def*)

from S **have** *ae-S*:

$AE \ x \ \text{in } M. x \in \{x \in \text{space } M. Q \ x\} \longleftrightarrow x \in (\bigcup_{n \in I}. \{x \in \text{space } M. P \ n \ x\} \cap S)$

$\bigwedge n. n \in I \implies AE \ x \ \text{in } M. x \in \{x \in \text{space } M. P \ n \ x\} \longleftrightarrow x \in \{x \in \text{space } M. P \ n \ x\} \cap S$

using *ae* **by** (*auto dest!: AE-prob-1*)

from *ae-S* **have** *:

$\mathcal{P}(x \ \text{in } M. Q \ x) = \text{prob} (\bigcup_{n \in I}. \{x \in \text{space } M. P \ n \ x\} \cap S)$

using $P \ Q \ S$ **by** (*intro finite-measure-eq-AE*) (*auto intro!: sets.Int*)

from *ae-S* **have** **::

$\bigwedge n. n \in I \implies \mathcal{P}(x \ \text{in } M. P \ n \ x) = \text{prob} (\{x \in \text{space } M. P \ n \ x\} \cap S)$

using $P \ Q \ S$ **by** (*intro finite-measure-eq-AE*) *auto*

show *?thesis*

using $S \ P$ *disj*

by (*auto simp add: * ** simp del: UN-simps intro!: finite-measure-finite-Union*)

qed

lemma (*in prob-space*) *prob-EX-countable*:

assumes *sets*: $\bigwedge i. i \in I \implies \{x \in \text{space } M. P \ i \ x\} \in \text{sets } M$ **and** I : *countable I*

assumes *disj*: $AE \ x \ \text{in } M. \forall i \in I. \forall j \in I. P \ i \ x \longrightarrow P \ j \ x \longrightarrow i = j$

shows $\mathcal{P}(x \ \text{in } M. \exists i \in I. P \ i \ x) = (\int^{+i}. \mathcal{P}(x \ \text{in } M. P \ i \ x) \ \partial \text{count-space } I)$

proof –

let $?N = \lambda x. \exists ! i \in I. P \ i \ x$

have *ennreal* $(\mathcal{P}(x \ \text{in } M. \exists i \in I. P \ i \ x)) = \mathcal{P}(x \ \text{in } M. (\exists i \in I. P \ i \ x \wedge ?N \ x))$

unfolding *ennreal-inj* [*OF measure-nonneg measure-nonneg*]

proof (*rule prob-eq-AE*)

show $AE \ x \ \text{in } M. (\exists i \in I. P \ i \ x) = (\exists i \in I. P \ i \ x \wedge ?N \ x)$

using *disj* **by** *eventually-elim blast*

qed (*auto intro!: sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1' I sets*)+

also have $\mathcal{P}(x \ \text{in } M. (\exists i \in I. P \ i \ x \wedge ?N \ x)) = \text{emeasure } M (\bigcup_{i \in I}. \{x \in \text{space } M. P \ i \ x \wedge ?N \ x\})$

unfolding *emeasure-eq-measure* **by** (*auto intro!: arg-cong[where f=prob] simp: measure-nonneg*)

also have $\dots = (\int^{+i}. \text{emeasure } M \ \{x \in \text{space } M. P \ i \ x \wedge ?N \ x\} \ \partial \text{count-space}$

I)

by (*rule emeasure-UN-countable*)
(auto intro!: sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1'
I sets

simp: disjoint-family-on-def)

also have $\dots = (\int^+ i. \mathcal{P}(x \text{ in } M. P \ i \ x) \ \partial \text{count-space } I)$

unfolding *emeasure-eq-measure* **using** *disj*

by (*intro nn-integral-cong ennreal-inj[THEN iffD2] prob-eq-AE*)
(auto intro!: sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1'
I sets measure-nonneg)+

finally show *?thesis .*

qed

lemma (*in prob-space*) *cond-prob-eq-AE*:

assumes $P: AE \ x \ \text{in } M. Q \ x \longrightarrow P \ x \longleftrightarrow P' \ x \ \{x \in \text{space } M. P \ x\} \in \text{events}$
 $\{x \in \text{space } M. P' \ x\} \in \text{events}$

assumes $Q: AE \ x \ \text{in } M. Q \ x \longleftrightarrow Q' \ x \ \{x \in \text{space } M. Q \ x\} \in \text{events}$
 $\{x \in \text{space } M. Q' \ x\} \in \text{events}$

shows $\text{cond-prob } M \ P \ Q = \text{cond-prob } M \ P' \ Q'$

using $P \ Q$

by (*auto simp: cond-prob-def intro!: arg-cong2[where f=op /] prob-eq-AE sets.sets-Collect-conj*)

lemma (*in prob-space*) *joint-distribution-Times-le-fst*:

random-variable $MX \ X \Longrightarrow \text{random-variable } MY \ Y \Longrightarrow A \in \text{sets } MX \Longrightarrow B \in \text{sets } MY$

$\Longrightarrow \text{emeasure } (\text{distr } M \ (MX \ \otimes_M \ MY) \ (\lambda x. (X \ x, Y \ x))) \ (A \times B) \leq \text{emeasure}$
 $(\text{distr } M \ MX \ X) \ A$

by (*auto simp: emeasure-distr measurable-pair-iff comp-def intro!: emeasure-mono measurable-sets*)

lemma (*in prob-space*) *joint-distribution-Times-le-snd*:

random-variable $MX \ X \Longrightarrow \text{random-variable } MY \ Y \Longrightarrow A \in \text{sets } MX \Longrightarrow B \in \text{sets } MY$

$\Longrightarrow \text{emeasure } (\text{distr } M \ (MX \ \otimes_M \ MY) \ (\lambda x. (X \ x, Y \ x))) \ (A \times B) \leq \text{emeasure}$
 $(\text{distr } M \ MY \ Y) \ B$

by (*auto simp: emeasure-distr measurable-pair-iff comp-def intro!: emeasure-mono measurable-sets*)

lemma (*in prob-space*) *variance-eq*:

fixes $X :: 'a \Rightarrow \text{real}$

assumes [*simp*]: *integrable* $M \ X$

assumes [*simp*]: *integrable* $M \ (\lambda x. (X \ x)^2)$

shows $\text{variance } X = \text{expectation } (\lambda x. (X \ x)^2) - (\text{expectation } X)^2$

by (*simp add: field-simps prob-space power2-diff power2-eq-square[symmetric]*)

lemma (*in prob-space*) *variance-positive: 0 ≤ variance (X::'a ⇒ real)*

by (*intro integral-nonneg-AE*) (*auto intro!: integral-nonneg-AE*)

lemma (in *prob-space*) *variance-mean-zero*:

expectation $X = 0 \implies$ *variance* $X = \text{expectation } (\lambda x. (X x)^2)$

by *simp*

locale *pair-prob-space* = *pair-sigma-finite* $M1\ M2 + M1$: *prob-space* $M1 + M2$:
prob-space $M2$ **for** $M1\ M2$

sublocale *pair-prob-space* $\subseteq P?$: *prob-space* $M1 \otimes_M M2$

proof

show *emeasure* $(M1 \otimes_M M2)$ (*space* $(M1 \otimes_M M2)$) = 1

by (*simp add*: $M2.$ *emeasure-pair-measure-Times* $M1.$ *emeasure-space-1* $M2.$ *emeasure-space-1*
space-pair-measure)

qed

locale *product-prob-space* = *product-sigma-finite* M **for** $M :: 'i \Rightarrow 'a$ *measure* +
fixes $I :: 'i$ *set*

assumes *prob-space*: $\bigwedge i. \text{prob-space } (M\ i)$

sublocale *product-prob-space* $\subseteq M?$: *prob-space* $M\ i$ **for** i

by (*rule prob-space*)

locale *finite-product-prob-space* = *finite-product-sigma-finite* $M\ I +$ *product-prob-space*
 $M\ I$ **for** $M\ I$

sublocale *finite-product-prob-space* \subseteq *prob-space* $\prod_M i \in I. M\ i$

proof

show *emeasure* $(\prod_M i \in I. M\ i)$ (*space* $(\prod_M i \in I. M\ i)$) = 1

by (*simp add*: *measure-times* $M.$ *emeasure-space-1* *prod.neutral-const* *space-PiM*)

qed

lemma (in *finite-product-prob-space*) *prob-times*:

assumes $X: \bigwedge i. i \in I \implies X\ i \in \text{sets } (M\ i)$

shows *prob* $(\prod_E i \in I. X\ i) = (\prod i \in I. M.\text{prob } i\ (X\ i))$

proof –

have *ennreal* (*measure* $(\prod_M i \in I. M\ i)$ $(\prod_E i \in I. X\ i)$) = *emeasure* $(\prod_M i \in I. M\ i)$
 $(\prod_E i \in I. X\ i)$

using X **by** (*simp add*: *emeasure-eq-measure*)

also have $\dots = (\prod i \in I. \text{emeasure } (M\ i)\ (X\ i))$

using *measure-times* X **by** *simp*

also have $\dots = \text{ennreal } (\prod i \in I. \text{measure } (M\ i)\ (X\ i))$

using X **by** (*simp add*: $M.$ *emeasure-eq-measure* *prod-ennreal* *measure-nonneg*)

finally show *?thesis* **by** (*simp add*: *measure-nonneg* *prod-nonneg*)

qed

1.2 Distributions

definition *distributed* :: $'a$ *measure* \Rightarrow $'b$ *measure* \Rightarrow $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow \text{ennreal})$
 $\Rightarrow \text{bool}$

where

$distributed\ M\ N\ X\ f \iff$
 $distr\ M\ N\ X = density\ N\ f \wedge f \in borel\text{-}measurable\ N \wedge X \in measurable\ M\ N$

lemma

assumes $distributed\ M\ N\ X\ f$
shows $distributed\text{-}distr\text{-}eq\text{-}density: distr\ M\ N\ X = density\ N\ f$
and $distributed\text{-}measurable: X \in measurable\ M\ N$
and $distributed\text{-}borel\text{-}measurable: f \in borel\text{-}measurable\ N$
using $assms$ **by** ($simp\text{-}all\ add: distributed\text{-}def$)

lemma

assumes $D: distributed\ M\ N\ X\ f$
shows $distributed\text{-}measurable'[measurable\text{-}dest]:$
 $g \in measurable\ L\ M \implies (\lambda x. X\ (g\ x)) \in measurable\ L\ N$
and $distributed\text{-}borel\text{-}measurable'[measurable\text{-}dest]:$
 $h \in measurable\ L\ N \implies (\lambda x. f\ (h\ x)) \in borel\text{-}measurable\ L$
using $distributed\text{-}measurable[OF\ D]\ distributed\text{-}borel\text{-}measurable[OF\ D]$
by $simp\text{-}all$

lemma $distributed\text{-}real\text{-}measurable:$

$(\bigwedge x. x \in space\ N \implies 0 \leq f\ x) \implies distributed\ M\ N\ X\ (\lambda x. ennreal\ (f\ x)) \implies$
 $f \in borel\text{-}measurable\ N$
by ($simp\text{-}all\ add: distributed\text{-}def$)

lemma $distributed\text{-}real\text{-}measurable':$

$(\bigwedge x. x \in space\ N \implies 0 \leq f\ x) \implies distributed\ M\ N\ X\ (\lambda x. ennreal\ (f\ x)) \implies$
 $h \in measurable\ L\ N \implies (\lambda x. f\ (h\ x)) \in borel\text{-}measurable\ L$
using $distributed\text{-}real\text{-}measurable[measurable]$ **by** $simp$

lemma $joint\text{-}distributed\text{-}measurable1:$

$distributed\ M\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))\ f \implies h1 \in measurable\ N\ M \implies$
 $(\lambda x. X\ (h1\ x)) \in measurable\ N\ S$
by $simp$

lemma $joint\text{-}distributed\text{-}measurable2:$

$distributed\ M\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))\ f \implies h2 \in measurable\ N\ M \implies$
 $(\lambda x. Y\ (h2\ x)) \in measurable\ N\ T$
by $simp$

lemma $distributed\text{-}count\text{-}space:$

assumes $X: distributed\ M\ (count\text{-}space\ A)\ X\ P$ **and** $a: a \in A$ **and** $A: finite\ A$
shows $P\ a = emeasure\ M\ (X\ \text{-}\{a\} \cap space\ M)$

proof –

have $emeasure\ M\ (X\ \text{-}\{a\} \cap space\ M) = emeasure\ (distr\ M\ (count\text{-}space\ A)\ X)\ \{a\}$

using $X\ a\ A$ **by** ($simp\ add: emeasure\text{-}distr$)

also have $\dots = emeasure\ (density\ (count\text{-}space\ A)\ P)\ \{a\}$

using X **by** ($simp\ add: distributed\text{-}distr\text{-}eq\text{-}density$)

also have $\dots = (\int^+ x. P\ a * indicator\ \{a\}\ x\ \partial count\text{-}space\ A)$

using X **a by** (*auto simp add: emeasure-density distributed-def indicator-def intro!: nn-integral-cong*)
also have $\dots = P$ a
using X **a by** (*subst nn-integral-cmult-indicator*) (*auto simp: distributed-def one-ennreal-def[symmetric] AE-count-space*)
finally show *?thesis ..*
qed

lemma *distributed-cong-density:*

(AE x in N. f x = g x) \implies g \in borel-measurable N \implies f \in borel-measurable N
 \implies
distributed M N X f \longleftrightarrow distributed M N X g
by (*auto simp: distributed-def intro!: density-cong*)

lemma (*in prob-space*) *distributed-imp-emeasure-nonzero:*

assumes X : *distributed M MX X Px*
shows *emeasure MX {x \in space MX. Px x \neq 0} \neq 0*

proof

note $Px =$ *distributed-borel-measurable[OF X]*
interpret X : *prob-space distr M MX X*
using *distributed-measurable[OF X]* **by** (*rule prob-space-distr*)

assume *emeasure MX {x \in space MX. Px x \neq 0} = 0*

with Px **have** *AE x in MX. Px x = 0*

by (*intro AE-I[OF subset-refl]*) (*auto simp: borel-measurable-ennreal-iff*)

moreover

from X .*emeasure-space-1* **have** $(\int^+ x. Px x \partial MX) = 1$

unfolding *distributed-distr-eq-density[OF X]* **using** Px

by (*subst (asm) emeasure-density*)

(*auto simp: borel-measurable-ennreal-iff intro!: integral-cong cong: nn-integral-cong*)

ultimately show *False*

by (*simp add: nn-integral-cong-AE*)

qed

lemma *subdensity:*

assumes T : *T \in measurable P Q*

assumes f : *distributed M P X f*

assumes g : *distributed M Q Y g*

assumes Y : *Y = T \circ X*

shows *AE x in P. g (T x) = 0 \longrightarrow f x = 0*

proof –

have $\{x \in \text{space } Q. g x = 0\} \in$ *null-sets (distr M Q (T \circ X))*

using g Y **by** (*auto simp: null-sets-density-iff distributed-def*)

also have *distr M Q (T \circ X) = distr (distr M P X) Q T*

using T f [*THEN distributed-measurable*] **by** (*rule distr-distr[symmetric]*)

finally have T – ‘ $\{x \in \text{space } Q. g x = 0\} \cap \text{space } P \in$ *null-sets (distr M P X)*

using T **by** (*subst (asm) null-sets-distr-iff*) *auto*

also have T – ‘ $\{x \in \text{space } Q. g x = 0\} \cap \text{space } P = \{x \in \text{space } P. g (T x) = 0\}$

using T **by** (*auto dest: measurable-space*)

finally show *?thesis*
using $f g$ **by** (*auto simp add: null-sets-density-iff distributed-def*)
qed

lemma *subdensity-real*:

fixes $g :: 'a \Rightarrow \text{real}$ **and** $f :: 'b \Rightarrow \text{real}$
assumes $T: T \in \text{measurable } P Q$
assumes $f: \text{distributed } M P X f$
assumes $g: \text{distributed } M Q Y g$
assumes $Y: Y = T \circ X$
shows $(AE x \text{ in } P. 0 \leq g (T x)) \Longrightarrow (AE x \text{ in } P. 0 \leq f x) \Longrightarrow AE x \text{ in } P. g (T x) = 0 \longrightarrow f x = 0$
using *subdensity[OF T, of M X $\lambda x. \text{ennreal } (f x) Y \lambda x. \text{ennreal } (g x)]$* *assms*
by *auto*

lemma *distributed-emeasure*:

$\text{distributed } M N X f \Longrightarrow A \in \text{sets } N \Longrightarrow \text{emeasure } M (X -^{\cdot} A \cap \text{space } M) = (\int^{+x}. f x * \text{indicator } A x \partial N)$
by (*auto simp: distributed-distr-eq-density[symmetric] emeasure-density[symmetric] emeasure-distr*)

lemma *distributed-nn-integral*:

$\text{distributed } M N X f \Longrightarrow g \in \text{borel-measurable } N \Longrightarrow (\int^{+x}. f x * g x \partial N) = (\int^{+x}. g (X x) \partial M)$
by (*auto simp: distributed-distr-eq-density[symmetric] nn-integral-density[symmetric] nn-integral-distr*)

lemma *distributed-integral*:

$\text{distributed } M N X f \Longrightarrow g \in \text{borel-measurable } N \Longrightarrow (\bigwedge x. x \in \text{space } N \Longrightarrow 0 \leq f x) \Longrightarrow (\int x. f x * g x \partial N) = (\int x. g (X x) \partial M)$
supply *distributed-real-measurable[measurable]*
by (*auto simp: distributed-distr-eq-density[symmetric] integral-real-density[symmetric] integral-distr*)

lemma *distributed-transform-integral*:

assumes $Px: \text{distributed } M N X Px \bigwedge x. x \in \text{space } N \Longrightarrow 0 \leq Px x$
assumes $\text{distributed } M P Y Py \bigwedge x. x \in \text{space } P \Longrightarrow 0 \leq Py x$
assumes $Y: Y = T \circ X$ **and** $T: T \in \text{measurable } N P$ **and** $f: f \in \text{borel-measurable } P$
shows $(\int x. Py x * f x \partial P) = (\int x. Px x * f (T x) \partial N)$
proof –
have $(\int x. Py x * f x \partial P) = (\int x. f (Y x) \partial M)$
by (*rule distributed-integral*) *fact+*
also have $\dots = (\int x. f (T (X x)) \partial M)$
using Y **by** *simp*
also have $\dots = (\int x. Px x * f (T x) \partial N)$
using *measurable-comp[OF T f] Px* **by** (*intro distributed-integral[symmetric]*)
(auto simp: comp-def)

finally show *?thesis* .
qed

lemma (in *prob-space*) *distributed-unique*:

assumes *Px*: *distributed M S X Px*

assumes *Py*: *distributed M S X Py*

shows *AE x in S. Px x = Py x*

proof –

interpret *X*: *prob-space distr M S X*

using *Px* **by** (*intro prob-space-distr*) *simp*

have *sigma-finite-measure (distr M S X) ..*

with *sigma-finite-density-unique[of Px S Py] Px Py*

show *?thesis*

by (*auto simp: distributed-def*)

qed

lemma (in *prob-space*) *distributed-jointI*:

assumes *sigma-finite-measure S sigma-finite-measure T*

assumes *X[measurable]: X ∈ measurable M S* **and** *Y[measurable]: Y ∈ measurable M T*

assumes [*measurable*]: *f ∈ borel-measurable (S ⊗_M T)* **and** *f: AE x in S ⊗_M T. 0 ≤ f x*

assumes *eq: ∧A B. A ∈ sets S ⇒ B ∈ sets T ⇒*

*emeasure M {x ∈ space M. X x ∈ A ∧ Y x ∈ B} = (∫⁺x. (∫⁺y. f (x, y) * indicator B y ∂T) * indicator A x ∂S)*

shows *distributed M (S ⊗_M T) (λx. (X x, Y x)) f*

unfolding *distributed-def*

proof *safe*

interpret *S*: *sigma-finite-measure S* **by** *fact*

interpret *T*: *sigma-finite-measure T* **by** *fact*

interpret *ST*: *pair-sigma-finite S T ..*

from *ST.sigma-finite-up-in-pair-measure-generator* **guess** *F :: nat ⇒ ('b × 'c) set ..* **note** *F = this*

let *?E = {a × b | a b. a ∈ sets S ∧ b ∈ sets T}*

let *?P = S ⊗_M T*

show *distr M ?P (λx. (X x, Y x)) = density ?P f (is ?L = ?R)*

proof (*rule measure-eqI-generator-eq[OF Int-stable-pair-measure-generator[of S T]]*)

show *?E ⊆ Pow (space ?P)*

using *sets.space-closed[of S] sets.space-closed[of T]* **by** (*auto simp: space-pair-measure*)

show *sets ?L = sigma-sets (space ?P) ?E*

by (*simp add: sets-pair-measure space-pair-measure*)

then show *sets ?R = sigma-sets (space ?P) ?E*

by *simp*

next

interpret *L*: *prob-space ?L*

by (*rule prob-space-distr (auto intro!: measurable-Pair)*)

show *range F ⊆ ?E (∪ i. F i) = space ?P ∧ i. emeasure ?L (F i) ≠ ∞*

```

    using F by (auto simp: space-pair-measure)
  next
    fix E assume E ∈ ?E
    then obtain A B where E[simp]: E = A × B
      and A[measurable]: A ∈ sets S and B[measurable]: B ∈ sets T by auto
    have emeasure ?L E = emeasure M {x ∈ space M. X x ∈ A ∧ Y x ∈ B}
      by (auto intro!: arg-cong[where f=emeasure M] simp add: emeasure-distr
measurable-Pair)
    also have ... = (∫+x. (∫+y. (f (x, y) * indicator B y) * indicator A x ∂T)
∂S)
      using f by (auto simp add: eq nn-integral-multc intro!: nn-integral-cong)
    also have ... = emeasure ?R E
      by (auto simp add: emeasure-density T.nn-integral-fst[symmetric]
intro!: nn-integral-cong split: split-indicator)
    finally show emeasure ?L E = emeasure ?R E .
  qed
qed (auto simp: f)

```

lemma (in prob-space) *distributed-swap*:

```

  assumes sigma-finite-measure S sigma-finite-measure T
  assumes Pxy: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
  shows distributed M (T ⊗M S) (λx. (Y x, X x)) (λ(x, y). Pxy (y, x))
proof –
  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..
  interpret TS: pair-sigma-finite T S ..

```

```

  note Pxy[measurable]
  show ?thesis
    apply (subst TS.distr-pair-swap)
    unfolding distributed-def
  proof safe
    let ?D = distr (S ⊗M T) (T ⊗M S) (λ(x, y). (y, x))
    show 1: (λ(x, y). Pxy (y, x)) ∈ borel-measurable ?D
      by auto
    show 2: random-variable (distr (S ⊗M T) (T ⊗M S) (λ(x, y). (y, x))) (λx.
(Y x, X x))
      using Pxy by auto
    { fix A assume A: A ∈ sets (T ⊗M S)
      let ?B = (λ(x, y). (y, x)) -‘ A ∩ space (S ⊗M T)
      from sets.sets-into-space[OF A]
      have emeasure M ((λx. (Y x, X x)) -‘ A ∩ space M) =
emeasure M ((λx. (X x, Y x)) -‘ ?B ∩ space M)
        by (auto intro!: arg-cong2[where f=emeasure] simp: space-pair-measure)
      also have ... = (∫+x. Pxy x * indicator ?B x ∂(S ⊗M T))
        using Pxy A by (intro distributed-emeasure) auto
      finally have emeasure M ((λx. (Y x, X x)) -‘ A ∩ space M) =
(∫+x. Pxy x * indicator A (snd x, fst x) ∂(S ⊗M T))

```

```

    by (auto intro!: nn-integral-cong split: split-indicator) }
  note * = this
  show distr M ?D (λx. (Y x, X x)) = density ?D (λ(x, y). Pxy (y, x))
  apply (intro measure-eqI)
  apply (simp-all add: emeasure-distr[OF 2] emeasure-density[OF 1])
  apply (subst nn-integral-distr)
  apply (auto intro!: * simp: comp-def split-beta)
  done
qed
qed

lemma (in prob-space) distr-marginal1:
  assumes sigma-finite-measure S sigma-finite-measure T
  assumes Pxy: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
  defines Px ≡ λx. (∫+ z. Pxy (x, z) ∂T)
  shows distributed M S X Px
  unfolding distributed-def
proof safe
  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..

  note Pxy[measurable]
  show X: X ∈ measurable M S by simp

  show borel: Px ∈ borel-measurable S
    by (auto intro!: T.nn-integral-fst simp: Px-def)

  interpret Pxy: prob-space distr M (S ⊗M T) (λx. (X x, Y x))
    by (intro prob-space-distr) simp

  show distr M S X = density S Px
proof (rule measure-eqI)
  fix A assume A: A ∈ sets (distr M S X)
  with X measurable-space[of Y M T]
  have emeasure (distr M S X) A = emeasure (distr M (S ⊗M T) (λx. (X x,
  Y x))) (A × space T)
    by (auto simp add: emeasure-distr intro!: arg-cong[where f=emeasure M])
  also have ... = emeasure (density (S ⊗M T) Pxy) (A × space T)
    using Pxy by (simp add: distributed-def)
  also have ... = ∫+ x. ∫+ y. Pxy (x, y) * indicator (A × space T) (x, y) ∂T
  ∂S
    using A borel Pxy
    by (simp add: emeasure-density T.nn-integral-fst[symmetric])
  also have ... = ∫+ x. Px x * indicator A x ∂S
proof (rule nn-integral-cong)
  fix x assume x ∈ space S
  moreover have eq: ∧y. y ∈ space T ⇒ indicator (A × space T) (x, y) =
  indicator A x

```

by (*auto simp: indicator-def*)
ultimately have $(\int^+ y. Pxy (x, y) * \text{indicator } (A \times \text{space } T) (x, y) \partial T)$
 $= (\int^+ y. Pxy (x, y) \partial T) * \text{indicator } A x$
by (*simp add: eq nn-integral-multc cong: nn-integral-cong*)
also have $(\int^+ y. Pxy (x, y) \partial T) = Px x$
by (*simp add: Px-def*)
finally show $(\int^+ y. Pxy (x, y) * \text{indicator } (A \times \text{space } T) (x, y) \partial T) = Px$
 $x * \text{indicator } A x .$
qed
finally show $\text{emeasure } (\text{distr } M S X) A = \text{emeasure } (\text{density } S Px) A$
using $A \text{ borel } Pxy$ **by** (*simp add: emeasure-density*)
qed *simp*
qed

lemma (*in prob-space*) *distr-marginal2*:
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes Pxy : *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$
shows *distributed* $M T Y (\lambda y. (\int^+ x. Pxy (x, y) \partial S))$
using *distr-marginal1*[$OF T S \text{ distributed-swap}[OF S T]$] Pxy **by** *simp*

lemma (*in prob-space*) *distributed-marginal-eq-joint1*:
assumes T : *sigma-finite-measure* T
assumes S : *sigma-finite-measure* S
assumes Px : *distributed* $M S X Px$
assumes Pxy : *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$
shows $AE x \text{ in } S. Px x = (\int^+ y. Pxy (x, y) \partial T)$
using $Px \text{ distr-marginal1}[OF S T Pxy]$ **by** (*rule distributed-unique*)

lemma (*in prob-space*) *distributed-marginal-eq-joint2*:
assumes T : *sigma-finite-measure* T
assumes S : *sigma-finite-measure* S
assumes Py : *distributed* $M T Y Py$
assumes Pxy : *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$
shows $AE y \text{ in } T. Py y = (\int^+ x. Pxy (x, y) \partial S)$
using $Py \text{ distr-marginal2}[OF S T Pxy]$ **by** (*rule distributed-unique*)

lemma (*in prob-space*) *distributed-joint-indep'*:
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes X [*measurable*]: *distributed* $M S X Px$ **and** Y [*measurable*]: *distributed*
 $M T Y Py$
assumes *indep*: $\text{distr } M S X \otimes_M \text{distr } M T Y = \text{distr } M (S \otimes_M T) (\lambda x. (X$
 $x, Y x))$
shows *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) (\lambda(x, y). Px x * Py y)$
unfolding *distributed-def*
proof *safe*
interpret S : *sigma-finite-measure* S **by** *fact*
interpret T : *sigma-finite-measure* T **by** *fact*
interpret ST : *pair-sigma-finite* $S T ..$

```

interpret X: prob-space density S Px
  unfolding distributed-distr-eq-density[OF X, symmetric]
  by (rule prob-space-distr) simp
have sf-X: sigma-finite-measure (density S Px) ..

interpret Y: prob-space density T Py
  unfolding distributed-distr-eq-density[OF Y, symmetric]
  by (rule prob-space-distr) simp
have sf-Y: sigma-finite-measure (density T Py) ..

show distr M (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ ) = density (S  $\otimes_M$  T) ( $\lambda(x, y). Px$ 
x * Py y)
  unfolding indep[symmetric] distributed-distr-eq-density[OF X] distributed-distr-eq-density[OF
Y]
  using distributed-borel-measurable[OF X]
  using distributed-borel-measurable[OF Y]
  by (rule pair-measure-density[OF - - T sf-Y])

show random-variable (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ ) by auto

show Pxy: ( $\lambda(x, y). Px x * Py y$ )  $\in$  borel-measurable (S  $\otimes_M$  T) by auto
qed

lemma distributed-integrable:
  distributed M N X f  $\implies$  g  $\in$  borel-measurable N  $\implies$  ( $\bigwedge x. x \in$  space N  $\implies$  0
 $\leq$  f x)  $\implies$ 
  integrable N ( $\lambda x. f x * g x$ )  $\longleftrightarrow$  integrable M ( $\lambda x. g (X x)$ )
  supply distributed-real-measurable[measurable]
  by (auto simp: distributed-distr-eq-density[symmetric] integrable-real-density[symmetric]
integrable-distr-eq)

lemma distributed-transform-integrable:
  assumes Px: distributed M N X Px  $\bigwedge x. x \in$  space N  $\implies$  0  $\leq$  Px x
  assumes distP: distributed M P Y Py  $\bigwedge x. x \in$  space P  $\implies$  0  $\leq$  Py x
  assumes Y: Y = ( $\lambda x. T (X x)$ ) and T: T  $\in$  measurable N P and f: f  $\in$ 
borel-measurable P
  shows integrable P ( $\lambda x. Py x * f x$ )  $\longleftrightarrow$  integrable N ( $\lambda x. Px x * f (T x)$ )
proof –
  have integrable P ( $\lambda x. Py x * f x$ )  $\longleftrightarrow$  integrable M ( $\lambda x. f (Y x)$ )
  by (rule distributed-integrable) fact+
  also have ...  $\longleftrightarrow$  integrable M ( $\lambda x. f (T (X x))$ )
  using Y by simp
  also have ...  $\longleftrightarrow$  integrable N ( $\lambda x. Px x * f (T x)$ )
  using measurable-comp[OF T f] Px by (intro distributed-integrable[symmetric])
(auto simp: comp-def)
  finally show ?thesis .
qed

lemma distributed-integrable-var:

```


fixes $X :: 'a \Rightarrow \text{real}$
shows $\text{distributed } M \text{ lborel } X (\lambda x. \text{ennreal } (f x)) \Longrightarrow (\bigwedge x. 0 \leq f x) \Longrightarrow$
 $\text{integrable lborel } (\lambda x. f x * x) \Longrightarrow \text{integrable } M X$
using $\text{distributed-integrable}[of } M \text{ lborel } X f \lambda x. x]$ **by** simp

lemma (in prob-space) $\text{distributed-variance}$:

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes D : $\text{distributed } M \text{ lborel } X f$ **and** $[\text{simp}]$: $\bigwedge x. 0 \leq f x$
shows $\text{variance } X = (\int x. x^2 * f (x + \text{expectation } X) \partial \text{lborel})$
proof ($\text{subst distributed-integral}[OF } D, \text{symmetric}]$)
show $(\int x. f x * (x - \text{expectation } X)^2 \partial \text{lborel}) = (\int x. x^2 * f (x + \text{expectation } X) \partial \text{lborel})$
by ($\text{subst lborel-integral-real-affine}[\text{where } c=1 \text{ and } t=\text{expectation } X]$) ($\text{auto simp: ac-simps}$)
qed simp-all

lemma (in prob-space) variance-affine :

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $[\text{arith}]$: $b \neq 0$
assumes $D[\text{intro}]$: $\text{distributed } M \text{ lborel } X f$
assumes $[\text{simp}]$: $\text{prob-space } (\text{density lborel } f)$
assumes $I[\text{simp}]$: $\text{integrable } M X$
assumes $I2[\text{simp}]$: $\text{integrable } M (\lambda x. (X x)^2)$
shows $\text{variance } (\lambda x. a + b * X x) = b^2 * \text{variance } X$
by (subst variance-eq)
 $(\text{auto simp: power2-sum power-mult-distrib prob-space variance-eq right-diff-distrib})$

definition

$\text{simple-distributed } M X f \longleftrightarrow$
 $(\forall x. 0 \leq f x) \wedge$
 $\text{distributed } M (\text{count-space } (X' \text{space } M)) X (\lambda x. \text{ennreal } (f x)) \wedge$
 $\text{finite } (X' \text{space } M)$

lemma $\text{simple-distributed-nonneg}[\text{dest}]$: $\text{simple-distributed } M X f \Longrightarrow 0 \leq f x$
by ($\text{auto simp: simple-distributed-def}$)

lemma $\text{simple-distributed}$:

$\text{simple-distributed } M X P x \Longrightarrow \text{distributed } M (\text{count-space } (X' \text{space } M)) X P x$
unfolding $\text{simple-distributed-def}$ **by** auto

lemma $\text{simple-distributed-finite}[\text{dest}]$: $\text{simple-distributed } M X P \Longrightarrow \text{finite } (X' \text{space } M)$

by ($\text{simp add: simple-distributed-def}$)

lemma (in prob-space) $\text{distributed-simple-function-superset}$:

assumes X : $\text{simple-function } M X \bigwedge x. x \in X' \text{space } M \Longrightarrow P x = \text{measure } M$
 $(X - \{x\} \cap \text{space } M)$
assumes A : $X' \text{space } M \subseteq A$ $\text{finite } A$
defines $S \equiv \text{count-space } A$ **and** $P' \equiv (\lambda x. \text{if } x \in X' \text{space } M \text{ then } P x \text{ else } 0)$

shows *distributed* $M S X P'$
unfolding *distributed-def*
proof *safe*
show $(\lambda x. \text{ennreal } (P' x)) \in \text{borel-measurable } S$ **unfolding** *S-def* **by** *simp*
show $\text{distr } M S X = \text{density } S P'$
proof (*rule measure-eqI-finite*)
show $\text{sets } (\text{distr } M S X) = \text{Pow } A$ $\text{sets } (\text{density } S P') = \text{Pow } A$
using A **unfolding** *S-def* **by** *auto*
show *finite* A **by** *fact*
fix a **assume** $a: a \in A$
then **have** $a \notin X \text{'space } M \implies X - \{a\} \cap \text{space } M = \{\}$ **by** *auto*
with $A a X$ **have** $\text{emeasure } (\text{distr } M S X) \{a\} = P' a$
by (*subst emeasure-distr*)
(auto simp add: S-def P'-def simple-functionD emeasure-eq-measure measurable-count-space-eq2
intro!: arg-cong[where f=prob])
also **have** $\dots = (\int^+ x. \text{ennreal } (P' a) * \text{indicator } \{a\} x \partial S)$
using $A X a$
by (*subst nn-integral-cmult-indicator*)
(auto simp: S-def P'-def simple-distributed-def simple-functionD measure-nonneg)
also **have** $\dots = (\int^+ x. \text{ennreal } (P' x) * \text{indicator } \{a\} x \partial S)$
by (*auto simp: indicator-def intro!: nn-integral-cong*)
also **have** $\dots = \text{emeasure } (\text{density } S P') \{a\}$
using $a A$ **by** (*intro emeasure-density[symmetric]*) (*auto simp: S-def*)
finally **show** $\text{emeasure } (\text{distr } M S X) \{a\} = \text{emeasure } (\text{density } S P') \{a\}$.
qed
show *random-variable* $S X$
using $X(1) A$ **by** (*auto simp: measurable-def simple-functionD S-def*)
qed

lemma (*in prob-space*) *simple-distributedI*:
assumes $X: \text{simple-function } M X$
 $\bigwedge x. 0 \leq P x$
 $\bigwedge x. x \in X \text{'space } M \implies P x = \text{measure } M (X - \{x\} \cap \text{space } M)$
shows *simple-distributed* $M X P$
unfolding *simple-distributed-def*
proof (*safe intro!: X*)
have *distributed* $M (\text{count-space } (X \text{'space } M)) X (\lambda x. \text{ennreal } (\text{if } x \in X \text{'space } M \text{ then } P x \text{ else } 0))$
(is ?A)
using *simple-functionD[OF X(1)]* **by** (*intro distributed-simple-function-superset[OF X(1,3)]*) *auto*
also **have** $?A \longleftrightarrow \text{distributed } M (\text{count-space } (X \text{'space } M)) X (\lambda x. \text{ennreal } (P x))$
by (*rule distributed-cong-density*) *auto*
finally **show** \dots .
qed (*rule simple-functionD[OF X(1)]*)

lemma *simple-distributed-joint-finite*:
assumes $X: \text{simple-distributed } M (\lambda x. (X x, Y x)) P x$

shows *finite* (X 'space M) *finite* (Y 'space M)
proof –
have *finite* (($\lambda x.$ (X x, Y x)) 'space M)
using X **by** (*auto simp: simple-distributed-def simple-functionD*)
then have *finite* (*fst* ' ($\lambda x.$ (X x, Y x)) 'space M) *finite* (*snd* ' ($\lambda x.$ (X x, Y x)) 'space M)
by *auto*
then show *fn: finite* (X 'space M) *finite* (Y 'space M)
by (*auto simp: image-image*)
qed

lemma *simple-distributed-joint2-finite*:
assumes X: *simple-distributed* M ($\lambda x.$ (X x, Y x, Z x)) Px
shows *finite* (X 'space M) *finite* (Y 'space M) *finite* (Z 'space M)
proof –
have *finite* (($\lambda x.$ (X x, Y x, Z x)) 'space M)
using X **by** (*auto simp: simple-distributed-def simple-functionD*)
then have *finite* (*fst* ' ($\lambda x.$ (X x, Y x, Z x)) 'space M)
finite ((*fst* \circ *snd*) ' ($\lambda x.$ (X x, Y x, Z x)) 'space M)
finite ((*snd* \circ *snd*) ' ($\lambda x.$ (X x, Y x, Z x)) 'space M)
by *auto*
then show *fn: finite* (X 'space M) *finite* (Y 'space M) *finite* (Z 'space M)
by (*auto simp: image-image*)
qed

lemma *simple-distributed-simple-function*:
simple-distributed M X Px \implies *simple-function* M X
unfolding *simple-distributed-def distributed-def*
by (*auto simp: simple-function-def measurable-count-space-eq2*)

lemma *simple-distributed-measure*:
simple-distributed M X P \implies $a \in X$ 'space M \implies P a = *measure* M (X - ' {a}
 \cap space M)
using *distributed-count-space*[of M X'space M X P a, *symmetric*]
by (*auto simp: simple-distributed-def measure-def*)

lemma (**in** *prob-space*) *simple-distributed-joint*:
assumes X: *simple-distributed* M ($\lambda x.$ (X x, Y x)) Px
defines S \equiv *count-space* (X'space M) \otimes_M *count-space* (Y'space M)
defines P \equiv ($\lambda x.$ *if* x \in ($\lambda x.$ (X x, Y x))'space M *then* Px x *else* 0)
shows *distributed* M S ($\lambda x.$ (X x, Y x)) P
proof –
from *simple-distributed-joint-finite*[OF X, *simp*]
have S-eq: S = *count-space* (X'space M \times Y'space M)
by (*simp add: S-def pair-measure-count-space*)
show ?thesis
unfolding S-eq P-def
proof (*rule distributed-simple-function-superset*)
show *simple-function* M ($\lambda x.$ (X x, Y x))

```

    using X by (rule simple-distributed-simple-function)
    fix x assume x ∈ (λx. (X x, Y x)) ' space M
    from simple-distributed-measure[OF X this]
    show Px x = prob ((λx. (X x, Y x)) - ' {x} ∩ space M) .
  qed auto
qed

```

```

lemma (in prob-space) simple-distributed-joint2:
  assumes X: simple-distributed M (λx. (X x, Y x, Z x)) Px
  defines S ≡ count-space (X'space M) ⊗M count-space (Y'space M) ⊗M
  count-space (Z'space M)
  defines P ≡ (λx. if x ∈ (λx. (X x, Y x, Z x))'space M then Px x else 0)
  shows distributed M S (λx. (X x, Y x, Z x)) P
proof -
  from simple-distributed-joint2-finite[OF X, simp]
  have S-eq: S = count-space (X'space M × Y'space M × Z'space M)
    by (simp add: S-def pair-measure-count-space)
  show ?thesis
    unfolding S-eq P-def
  proof (rule distributed-simple-function-superset)
    show simple-function M (λx. (X x, Y x, Z x))
      using X by (rule simple-distributed-simple-function)
    fix x assume x ∈ (λx. (X x, Y x, Z x)) ' space M
    from simple-distributed-measure[OF X this]
    show Px x = prob ((λx. (X x, Y x, Z x)) - ' {x} ∩ space M) .
  qed auto
qed

```

```

lemma (in prob-space) simple-distributed-sum-space:
  assumes X: simple-distributed M X f
  shows sum f (X'space M) = 1
proof -
  from X have sum f (X'space M) = prob (⋃i∈X'space M. X - ' {i} ∩ space
  M)
    by (subst finite-measure-finite-Union)
    (auto simp add: disjoint-family-on-def simple-distributed-measure simple-distributed-simple-function
  simple-functionD
    intro!: sum.cong arg-cong[where f=prob])
  also have ... = prob (space M)
    by (auto intro!: arg-cong[where f=prob])
  finally show ?thesis
    using emeasure-space-1 by (simp add: emeasure-eq-measure)
qed

```

```

lemma (in prob-space) distributed-marginal-eq-joint-simple:
  assumes Px: simple-function M X
  assumes Py: simple-distributed M Y Py
  assumes Pxy: simple-distributed M (λx. (X x, Y x)) Pxy
  assumes y: y ∈ Y'space M

```

shows $P_y y = (\sum_{x \in X} \text{space } M. \text{if } (x, y) \in (\lambda x. (X x, Y x)) \text{ ' space } M \text{ then } P_{xy} (x, y) \text{ else } 0)$

proof –

note $P_x = \text{simple-distributedI}[OF P_x \text{ measure-nonneg refl}]$

have $AE y \text{ in count-space } (Y \text{ ' space } M). \text{ennreal } (P_y y) = \int^+ x. \text{ennreal } (\text{if } (x, y) \in (\lambda x. (X x, Y x)) \text{ ' space } M \text{ then } P_{xy} (x, y) \text{ else } 0) \partial \text{count-space } (X \text{ ' space } M)$

using $\text{sigma-finite-measure-count-space-finite sigma-finite-measure-count-space-finite simple-distributed}[OF P_y] \text{ simple-distributed-joint}[OF P_{xy}]$

by $(\text{rule distributed-marginal-eq-joint2})$
 $(\text{auto intro: } P_y P_x \text{ simple-distributed-finite})$

then have $\text{ennreal } (P_y y) = (\sum_{x \in X} \text{space } M. \text{ennreal } (\text{if } (x, y) \in (\lambda x. (X x, Y x)) \text{ ' space } M \text{ then } P_{xy} (x, y) \text{ else } 0))$

using $y P_x [THEN \text{ simple-distributed-finite}]$

by $(\text{auto simp: } AE\text{-count-space nn-integral-count-space-finite})$

also have $\dots = (\sum_{x \in X} \text{space } M. \text{if } (x, y) \in (\lambda x. (X x, Y x)) \text{ ' space } M \text{ then } P_{xy} (x, y) \text{ else } 0)$

using $P_{xy} \text{ by } (\text{intro sum-ennreal}) \text{ auto}$

finally show $?thesis$

using $\text{simple-distributed-nonneg}[OF P_y] \text{ simple-distributed-nonneg}[OF P_{xy}]$

by $(\text{subst } (asm) \text{ ennreal-inj}) (\text{auto intro!: sum-nonneg})$

qed

lemma *distributedI-real*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $gen: \text{sets } M1 = \text{sigma-sets } (\text{space } M1) E \text{ and } \text{Int-stable } E$

and $A: \text{range } A \subseteq E \ (\bigcup i::\text{nat. } A i) = \text{space } M1 \wedge i. \text{emeasure } (\text{distr } M M1 X) (A i) \neq \infty$

and $X: X \in \text{measurable } M M1$

and $f: f \in \text{borel-measurable } M1 \ AE x \text{ in } M1. 0 \leq f x$

and $eq: \bigwedge A. A \in E \implies \text{emeasure } M (X \text{ -' } A \cap \text{space } M) = (\int^+ x. f x * \text{indicator } A x \partial M1)$

shows $\text{distributed } M M1 X f$

unfolding *distributed-def*

proof (intro conjI)

show $\text{distr } M M1 X = \text{density } M1 f$

proof $(\text{rule measure-eqI-generator-eq}[\text{where } A=A])$

{ fix } A \text{ assume } A: A \in E

then have $A \in \text{sigma-sets } (\text{space } M1) E \text{ by auto}$

then have $A \in \text{sets } M1$

using $gen \text{ by simp}$

with $f A \text{ eq[of } A] X \text{ show } \text{emeasure } (\text{distr } M M1 X) A = \text{emeasure } (\text{density } M1 f) A$

by $(\text{auto simp add: emeasure-distr emeasure-density ennreal-indicator intro!: nn-integral-cong split: split-indicator}) \}$

note $eq-E = \text{this}$

show $\text{Int-stable } E \text{ by fact}$

{ fix } e \text{ assume } e \in E

```

then have  $e \in \text{sigma-sets } (\text{space } M1) E$  by auto
then have  $e \in \text{sets } M1$  unfolding gen .
then have  $e \subseteq \text{space } M1$  by (rule sets.sets-into-space) }
then show  $E \subseteq \text{Pow } (\text{space } M1)$  by auto
show sets  $(\text{distr } M M1 X) = \text{sigma-sets } (\text{space } M1) E$ 
sets  $(\text{density } M1 (\lambda x. \text{ennreal } (f x))) = \text{sigma-sets } (\text{space } M1) E$ 
unfolding gen[symmetric] by auto
qed fact+
qed (insert X f, auto)

```

lemma *distributedI-borel-atMost:*

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes [measurable]:  $X \in \text{borel-measurable } M$ 
and [measurable]:  $f \in \text{borel-measurable borel}$  and  $f[\text{simp}]: \forall x \text{ in } \text{lborel}. 0 \leq f x$ 
and g-eq:  $\bigwedge a. (\int^+ x. f x * \text{indicator } \{..a\} x \partial \text{lborel}) = \text{ennreal } (g a)$ 
and M-eq:  $\bigwedge a. \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal } (g a)$ 
shows distributed M lborel X f
proof (rule distributedI-real)
show sets  $(\text{lborel}::\text{real measure}) = \text{sigma-sets } (\text{space } \text{lborel}) (\text{range } \text{atMost})$ 
by (simp add: borel-eq-atMost)
show Int-stable (range atMost :: real set set)
by (auto simp: Int-stable-def)
have vimage-eq:  $\bigwedge a. (X -' \{..a\} \cap \text{space } M) = \{x \in \text{space } M. X x \leq a\}$  by auto
define A where  $A i = \{.. \text{real } i\}$  for  $i :: \text{nat}$ 
then show  $\text{range } A \subseteq \text{range } \text{atMost } (\bigcup i. A i) = \text{space } \text{lborel}$ 
 $\bigwedge i. \text{emeasure } (\text{distr } M \text{lborel } X) (A i) \neq \infty$ 
by (auto simp: real-arch-simple emeasure-distr vimage-eq M-eq)

fix  $A :: \text{real set}$  assume  $A \in \text{range } \text{atMost}$ 
then obtain a where  $A = \{..a\}$  by auto
show  $\text{emeasure } M (X -' A \cap \text{space } M) = (\int^+ x. f x * \text{indicator } A x \partial \text{lborel})$ 
unfolding vimage-eq A M-eq g-eq ..
qed auto

```

lemma (in *prob-space*) *uniform-distributed-params:*

```

assumes  $X: \text{distributed } M MX X (\lambda x. \text{indicator } A x / \text{measure } MX A)$ 
shows  $A \in \text{sets } MX \text{ measure } MX A \neq 0$ 
proof -
interpret  $X: \text{prob-space } \text{distr } M MX X$ 
using distributed-measurable[OF X] by (rule prob-space-distr)

show  $\text{measure } MX A \neq 0$ 
proof
assume  $\text{measure } MX A = 0$ 
with  $X.\text{emeasure-space-1 } X.\text{prob-space distributed-distr-eq-density[OF X]}$ 
show False
by (simp add: emeasure-density zero-ennreal-def[symmetric])
qed

```

with *measure-notin-sets*[of A MX] **show** $A \in \text{sets } MX$
by *blast*
qed

lemma *prob-space-uniform-measure*:

assumes A : *emeasure* M $A \neq 0$ *emeasure* M $A \neq \infty$
shows *prob-space* (*uniform-measure* M A)

proof

show *emeasure* (*uniform-measure* M A) (*space* (*uniform-measure* M A)) = 1
using *emeasure-uniform-measure*[*OF* *emeasure-neq-0-sets*[*OF* $A(1)$], of *space* M]
using *sets.sets-into-space*[*OF* *emeasure-neq-0-sets*[*OF* $A(1)$]] A
by (*simp add: Int-absorb2 less-top*)

qed

lemma *prob-space-uniform-count-measure*: $\text{finite } A \implies A \neq \{\} \implies \text{prob-space}$
(*uniform-count-measure* A)

by *standard* (*auto simp: emeasure-uniform-count-measure space-uniform-count-measure one-enreal-def*)

lemma (**in** *prob-space*) *measure-uniform-measure-eq-cond-prob*:

assumes [*measurable*]: *Measurable.pred* M P *Measurable.pred* M Q

shows $\mathcal{P}(x \text{ in } \text{uniform-measure } M \{x \in \text{space } M. Q\ x\}. P\ x) = \mathcal{P}(x \text{ in } M. P\ x \mid Q\ x)$

proof *cases*

assume Q : *measure* M $\{x \in \text{space } M. Q\ x\} = 0$

then have $*$: $\text{AE } x \text{ in } M. \neg Q\ x$

by (*simp add: prob-eq-0*)

then have *density* M ($\lambda x. \text{indicator } \{x \in \text{space } M. Q\ x\} x / \text{emeasure } M \{x \in \text{space } M. Q\ x\}$) = *density* M ($\lambda x. 0$)

by (*intro density-cong*) *auto*

with $*$ **show** *?thesis*

unfolding *uniform-measure-def*

by (*simp add: emeasure-density measure-def cond-prob-def emeasure-eq-0-AE*)

next

assume Q : *measure* M $\{x \in \text{space } M. Q\ x\} \neq 0$

then show $\mathcal{P}(x \text{ in } \text{uniform-measure } M \{x \in \text{space } M. Q\ x\}. P\ x) = \text{cond-prob}$
 $M\ P\ Q$

by (*subst measure-uniform-measure*)

(*auto simp: emeasure-eq-measure cond-prob-def measure-nonneg intro!: arg-cong[where*
 $f = \text{prob}$])

qed

lemma *prob-space-point-measure*:

$\text{finite } S \implies (\bigwedge s. s \in S \implies 0 \leq p\ s) \implies (\sum s \in S. p\ s) = 1 \implies \text{prob-space}$
(*point-measure* S p)

by (*rule prob-spaceI*) (*simp add: space-point-measure emeasure-point-measure-finite*)

lemma (**in** *prob-space*) *distr-pair-fst*: $\text{distr } (N \otimes_M M) N\ \text{fst} = N$

proof (*intro measure-eqI*)

fix A **assume** $A: A \in \text{sets } (\text{distr } (N \otimes_M M) N \text{ fst})$
from A **have** $\text{emeasure } (\text{distr } (N \otimes_M M) N \text{ fst}) A = \text{emeasure } (N \otimes_M M)$
 $(A \times \text{space } M)$
by (*auto simp add: emeasure-distr space-pair-measure dest: sets.sets-into-space*
intro!: arg-cong2[where f=emeasure])
with A **show** $\text{emeasure } (\text{distr } (N \otimes_M M) N \text{ fst}) A = \text{emeasure } N A$
by (*simp add: emeasure-pair-measure-Times emeasure-space-1*)
qed *simp*

lemma (*in product-prob-space*) *distr-reorder*:

assumes *inj-on* $t J t \in J \rightarrow K$ *finite* K
shows $\text{distr } (PiM K M) (PiM J (\lambda x. M (t x))) (\lambda \omega. \lambda n \in J. \omega (t n)) = PiM J$
 $(\lambda x. M (t x))$

proof (*rule product-sigma-finite.PiM-eqI*)

show *product-sigma-finite* $(\lambda x. M (t x))$..
have $t'J \subseteq K$ **using** *assms* **by** *auto*
then show [*simp*]: *finite* J
by (*rule finite-imageD[OF finite-subset]*) *fact+*
fix A **assume** $A: \bigwedge i. i \in J \implies A i \in \text{sets } (M (t i))$
moreover have $((\lambda \omega. \lambda n \in J. \omega (t n)) - ' PiE J A \cap \text{space } (PiM K M)) =$
 $(\Pi_E i \in K. \text{if } i \in t'J \text{ then } A (the-inv-into J t i) \text{ else } \text{space } (M i))$
using $A A$ [*THEN sets.sets-into-space*] $\langle t \in J \rightarrow K \rangle \langle inj-on t J \rangle$
by (*subst prod-emb-Pi[symmetric]*) (*auto simp: space-PiM PiE-iff the-inv-into-f-f*
prod-emb-def)
ultimately show $\text{distr } (PiM K M) (PiM J (\lambda x. M (t x))) (\lambda \omega. \lambda n \in J. \omega (t n))$
 $(PiE J A) = (\prod i \in J. M (t i) (A i))$
using *assms*
apply (*subst emeasure-distr*)
apply (*auto intro!: sets-PiM-I-finite simp: Pi-iff*)
apply (*subst emeasure-PiM*)
apply (*auto simp: the-inv-into-f-f <inj-on t J> prod.reindex[OF <inj-on t J>*
if-distrib[where f=emeasure (M -)] prod.If-cases emeasure-space-1 Int-absorb1
 $\langle t'J \subseteq K \rangle$)
done
qed *simp*

lemma (*in product-prob-space*) *distr-restrict*:

$J \subseteq K \implies \text{finite } K \implies (\Pi_M i \in J. M i) = \text{distr } (\Pi_M i \in K. M i) (\Pi_M i \in J. M$
 $i) (\lambda f. \text{restrict } f J)$
using *distr-reorder[of $\lambda x. x J K$]* **by** (*simp add: Pi-iff subset-eq*)

lemma (*in product-prob-space*) *emeasure-prod-emb[simp]*:

assumes $L: J \subseteq L$ *finite* L **and** $X: X \in \text{sets } (PiM J M)$
shows $\text{emeasure } (PiM L M) (\text{prod-emb } L M J X) = \text{emeasure } (PiM J M) X$
by (*subst distr-restrict[OF L]*)
(simp add: prod-emb-def space-PiM emeasure-distr measurable-restrict-subset
 $L X)$

lemma *emeasure-distr-restrict*:

assumes $I \subseteq K$ **and** $Q[\text{measurable-cong}]$: *sets* $Q = \text{sets } (PiM\ K\ M)$ **and**
 $A[\text{measurable}]$: $A \in \text{sets } (PiM\ I\ M)$
shows $\text{emeasure } (\text{distr } Q\ (PiM\ I\ M)\ (\lambda\omega. \text{restrict } \omega\ I))\ A = \text{emeasure } Q$
(prod-emb K M I A)
using $\langle I \subseteq K \rangle$ *sets-eq-imp-space-eq*[*OF Q*]
by *(subst emeasure-distr)*
(auto simp: measurable-cong-sets[*OF Q*] *prod-emb-def space-PiM*[*symmetric*]
intro!: *measurable-restrict*)

lemma *(in prob-space) prob-space-completion*: *prob-space (completion M)*
by *(rule prob-spaceI) (simp add: emeasure-space-1)*

end

2 Distribution Functions

Shows that the cumulative distribution function (cdf) of a distribution (a measure on the reals) is nondecreasing and right continuous, which tends to 0 and 1 in either direction.

Conversely, every such function is the cdf of a unique distribution. This direction defines the measure in the obvious way on half-open intervals, and then applies the Caratheodory extension theorem.

theory *Distribution-Functions*
imports *Probability-Measure*
begin

lemma *UN-Ioc-eq-UNIV*: $(\bigcup n. \{ -\text{real } n <.. \text{real } n \}) = \text{UNIV}$
by *auto*
(metis le-less-trans minus-minus neg-less-iff-less not-le real-arch-simple
of-nat-0-le-iff reals-Archimedean2)

2.1 Properties of cdf's

definition
 $\text{cdf} :: \text{real measure} \Rightarrow \text{real} \Rightarrow \text{real}$
where
 $\text{cdf } M \equiv \lambda x. \text{measure } M \{..x\}$

lemma *cdf-def2*: $\text{cdf } M\ x = \text{measure } M \{..x\}$
by *(simp add: cdf-def)*

locale *finite-borel-measure = finite-measure M for M :: real measure +*
assumes *M-is-borel*: *sets M = sets borel*
begin

lemma *sets-M*[*intro*]: $a \in \text{sets borel} \implies a \in \text{sets } M$

using *M-is-borel* by *auto*

lemma *cdf-diff-eq*:

assumes $x < y$

shows $\text{cdf } M \ y - \text{cdf } M \ x = \text{measure } M \ \{x < ..y\}$

proof –

from *assms* have $*$: $\{..x\} \cup \{x < ..y\} = \{..y\}$ by *auto*

have $\text{measure } M \ \{..y\} = \text{measure } M \ \{..x\} + \text{measure } M \ \{x < ..y\}$

by (*subst finite-measure-Union [symmetric], auto simp add: **)

thus *?thesis*

unfolding *cdf-def* by *auto*

qed

lemma *cdf-nondecreasing*: $x \leq y \implies \text{cdf } M \ x \leq \text{cdf } M \ y$

unfolding *cdf-def* by (*auto intro!: finite-measure-mono*)

lemma *borel-UNIV*: $\text{space } M = \text{UNIV}$

by (*metis in-mono sets.sets-into-space space-in-borel top-le M-is-borel*)

lemma *cdf-nonneg*: $\text{cdf } M \ x \geq 0$

unfolding *cdf-def* by (*rule measure-nonneg*)

lemma *cdf-bounded*: $\text{cdf } M \ x \leq \text{measure } M \ (\text{space } M)$

unfolding *cdf-def* by (*intro bounded-measure*)

lemma *cdf-lim-infnty*:

$((\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow \text{measure } M \ (\text{space } M))$

proof –

have $(\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow \text{measure } M \ (\bigcup i::\text{nat}. \{.. \text{real } i\})$

unfolding *cdf-def* by (*rule finite-Lim-measure-incseq*) (*auto simp: incseq-def*)

also have $(\bigcup i::\text{nat}. \{.. \text{real } i\}) = \text{space } M$

by (*auto simp: borel-UNIV intro: real-arch-simple*)

finally show *?thesis* .

qed

lemma *cdf-lim-at-top*: $(\text{cdf } M \longrightarrow \text{measure } M \ (\text{space } M)) \text{ at-top}$

by (*rule tendsto-at-topI-sequentially-real*)

(*simp-all add: mono-def cdf-nondecreasing cdf-lim-infnty*)

lemma *cdf-lim-neg-infnty*: $((\lambda i. \text{cdf } M \ (- \text{real } i)) \longrightarrow 0)$

proof –

have $(\lambda i. \text{cdf } M \ (- \text{real } i)) \longrightarrow \text{measure } M \ (\bigcap i::\text{nat}. \{.. - \text{real } i\})$

unfolding *cdf-def* by (*rule finite-Lim-measure-decseq*) (*auto simp: decseq-def*)

also have $(\bigcap i::\text{nat}. \{.. - \text{real } i\}) = \{\}$

by *auto* (*metis leD le-minus-iff reals-Archimedean2*)

finally show *?thesis*

by *simp*

qed

lemma *cdf-lim-at-bot*: $(cdf\ M \longrightarrow 0)$ *at-bot*
proof –
have *: $((\lambda x :: real. -\ cdf\ M\ (-\ x)) \longrightarrow 0)$ *at-top*
by (*intro tendsto-at-topI-sequentially-real monoI*)
 (*auto simp: cdf-nondecreasing cdf-lim-neg-infty tendsto-minus-cancel-left[symmetric]*)
from *filterlim-compose [OF *, OF filterlim-uminus-at-top-at-bot]*
show *?thesis*
unfolding *tendsto-minus-cancel-left[symmetric]* **by** *simp*
qed

lemma *cdf-is-right-cont*: *continuous (at-right a) (cdf M)*
unfolding *continuous-within*
proof (*rule tendsto-at-right-sequentially[where b=a + 1]*)
fix $f :: nat \Rightarrow real$ **and** x **assume** $f: decseq\ f\ f \longrightarrow a$
then have $(\lambda n. cdf\ M\ (f\ n)) \longrightarrow measure\ M\ (\bigcap i. \{.. f\ i\})$
using $\langle decseq\ f \rangle$ **unfolding** *cdf-def*
by (*intro finite-Lim-measure-decseq*) (*auto simp: decseq-def*)
also have $(\bigcap i. \{.. f\ i\}) = \{.. a\}$
using *decseq-le[OF f]* **by** (*auto intro: order-trans LIMSEQ-le-const[OF f(2)]*)
finally show $(\lambda n. cdf\ M\ (f\ n)) \longrightarrow cdf\ M\ a$
by (*simp add: cdf-def*)
qed *simp*

lemma *cdf-at-left*: $(cdf\ M \longrightarrow measure\ M\ \{..<a\})$ (*at-left a*)
proof (*rule tendsto-at-left-sequentially[of a - 1]*)
fix $f :: nat \Rightarrow real$ **and** x **assume** $f: incseq\ f\ f \longrightarrow a \wedge x. f\ x < a \wedge x. a - 1 < f\ x$
then have $(\lambda n. cdf\ M\ (f\ n)) \longrightarrow measure\ M\ (\bigcup i. \{.. f\ i\})$
using $\langle incseq\ f \rangle$ **unfolding** *cdf-def*
by (*intro finite-Lim-measure-incseq*) (*auto simp: incseq-def*)
also have $(\bigcup i. \{.. f\ i\}) = \{..<a\}$
by (*auto dest!: order-tendstoD(1)[OF f(2)] eventually-happens'[OF sequentially-bot]*)
intro: less-imp-le le-less-trans f(3)
finally show $(\lambda n. cdf\ M\ (f\ n)) \longrightarrow measure\ M\ \{..<a\}$
by (*simp add: cdf-def*)
qed *auto*

lemma *isCont-cdf*: $isCont\ (cdf\ M)\ x \longleftrightarrow measure\ M\ \{x\} = 0$
proof –
have $isCont\ (cdf\ M)\ x \longleftrightarrow cdf\ M\ x = measure\ M\ \{..<x\}$
by (*auto simp: continuous-at-split cdf-is-right-cont continuous-within[where s={..<-}]*)
cdf-at-left tendsto-unique[OF - cdf-at-left]
also have $cdf\ M\ x = measure\ M\ \{..<x\} \longleftrightarrow measure\ M\ \{x\} = 0$
unfolding *cdf-def ivl-disj-un(2)[symmetric]*
by (*subst finite-measure-Union*) *auto*
finally show *?thesis* .
qed

lemma *countable-atoms*: *countable* $\{x. \text{measure } M \{x\} > 0\}$
using *countable-support* **unfolding** *zero-less-measure-iff* .

end

locale *real-distribution* = *prob-space* M **for** $M :: \text{real measure} +$
assumes *events-eq-borel* [*simp*, *measurable-cong*]: *sets* $M = \text{sets borel}$
begin

lemma *finite-borel-measure-M*: *finite-borel-measure* M
by *standard auto*

sublocale *finite-borel-measure* M
by (*rule finite-borel-measure-M*)

lemma *space-eq-univ* [*simp*]: *space* $M = \text{UNIV}$
using *events-eq-borel* [*THEN sets-eq-imp-space-eq*] **by** *simp*

lemma *cdf-bounded-prob*: $\bigwedge x. \text{cdf } M \ x \leq 1$
by (*subst prob-space* [*symmetric*], *rule cdf-bounded*)

lemma *cdf-lim-inf-prob*: $(\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow 1$
by (*subst prob-space* [*symmetric*], *rule cdf-lim-inf*)

lemma *cdf-lim-at-top-prob*: $(\text{cdf } M \longrightarrow 1)$ *at-top*
by (*subst prob-space* [*symmetric*], *rule cdf-lim-at-top*)

lemma *measurable-finite-borel* [*simp*]:
 $f \in \text{borel-measurable borel} \implies f \in \text{borel-measurable } M$
by (*rule borel-measurable-subalgebra* [**where** $N = \text{borel}$]) *auto*

end

lemma (**in** *prob-space*) *real-distribution-distr* [*intro*, *simp*]:
random-variable borel $X \implies \text{real-distribution} \ (\text{distr } M \ \text{borel } X)$
unfolding *real-distribution-def* *real-distribution-axioms-def* **by** (*auto intro!*: *prob-space-distr*)

2.2 Uniqueness

lemma (**in** *finite-borel-measure*) *emeasure-Ioc*:
assumes $a \leq b$ **shows** *emeasure* $M \ \{a <.. b\} = \text{cdf } M \ b - \text{cdf } M \ a$

proof –

have $\{a <.. b\} = \{..b\} - \{..a\}$

by *auto*

moreover have $\{..x\} \in \text{sets } M$ **for** x

using *atMost-borel* [*of* x] *M-is-borel* **by** *auto*

moreover note $\langle a \leq b \rangle$

ultimately show *?thesis*

by (*simp add: emeasure-eq-measure finite-measure-Diff cdf-def*)

qed

lemma *cdf-unique'*:

fixes $M1\ M2$

assumes *finite-borel-measure* $M1$ **and** *finite-borel-measure* $M2$

assumes $\text{cdf } M1 = \text{cdf } M2$

shows $M1 = M2$

proof (*rule measure-eqI-generator-eq*[**where** $\Omega = UNIV$])

fix X **assume** $X \in \text{range } (\lambda(a, b). \{a <..b\}::\text{real})$

then obtain $a\ b$ **where** $X \in \{a <..b\}$ **by** *auto*

then show $\text{emeasure } M1\ X = \text{emeasure } M2\ X$

by (*cases* $a \leq b$)

(*simp-all add: assms(1,2)*[*THEN finite-borel-measure.emeasure-Ioc*] *assms(3)*)

next

show $(\bigcup i. \{- \text{real } (i::\text{nat}) <.. \text{real } i\}) = UNIV$

by (*rule UN-Ioc-eq-UNIV*)

qed (*auto simp: finite-borel-measure.emeasure-Ioc*[*OF assms(1)*])

assms(1,2)[*THEN finite-borel-measure.M-is-borel*] *borel-sigma-sets-Ioc*

Int-stable-def)

lemma *cdf-unique*:

real-distribution $M1 \implies \text{real-distribution } M2 \implies \text{cdf } M1 = \text{cdf } M2 \implies M1 = M2$

using *cdf-unique'*[*of* $M1\ M2$] **by** (*simp add: real-distribution.finite-borel-measure-M*)

lemma

fixes $F :: \text{real} \Rightarrow \text{real}$

assumes *nondecF* : $\bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$

and *right-cont-F* : $\bigwedge a. \text{continuous } (\text{at-right } a)\ F$

and *lim-F-at-bot* : $(F \longrightarrow 0)\ \text{at-bot}$

and *lim-F-at-top* : $(F \longrightarrow m)\ \text{at-top}$

and $m: 0 \leq m$

shows *interval-measure-UNIV*: $\text{emeasure } (\text{interval-measure } F)\ UNIV = m$

and *finite-borel-measure-interval-measure*: *finite-borel-measure* (*interval-measure* F)

proof –

let $?F = \text{interval-measure } F$

{ **have** $\text{ennreal } (m - 0) = (\text{SUP } i::\text{nat}. \text{ennreal } (F\ (\text{real } i) - F\ (- \text{real } i)))$

by (*intro LIMSEQ-unique*[*OF - LIMSEQ-SUP*] *tendsto-ennrealI* *tendsto-intros*

lim-F-at-bot[*THEN filterlim-compose*] *lim-F-at-top*[*THEN filterlim-compose*]

lim-F-at-bot[*THEN filterlim-compose*] *filterlim-real-sequentially*

filterlim-uminus-at-top[*THEN iffD1*])

(*auto simp: incseq-def nondecF intro!: diff-mono*)

also have $\dots = (\text{SUP } i::\text{nat}. \text{emeasure } ?F\ \{- \text{real } i <.. \text{real } i\})$

by (*subst emeasure-interval-measure-Ioc*) (*simp-all add: nondecF right-cont-F*)

also have $\dots = \text{emeasure } ?F\ (\bigcup i::\text{nat}. \{- \text{real } i <.. \text{real } i\})$

by (*rule SUP-emeasure-incseq*) (*auto simp: incseq-def*)

also have $(\bigcup i. \{- \text{real } (i::\text{nat}) <.. \text{real } i\}) = \text{space } ?F$

by (*simp add: UN-Ioc-eq-UNIV*)

```

finally have emeasure ?F (space ?F) = m
  by simp }
note * = this
then show emeasure (interval-measure F) UNIV = m
  by simp

interpret finite-measure ?F
proof
  show emeasure ?F (space ?F)  $\neq \infty$ 
    using * by simp
qed
show finite-borel-measure (interval-measure F)
  proof qed simp-all
qed

lemma real-distribution-interval-measure:
  fixes F :: real  $\Rightarrow$  real
  assumes nondecF :  $\bigwedge x y. x \leq y \implies F x \leq F y$  and
    right-cont-F :  $\bigwedge a. \text{continuous (at-right } a) F$  and
    lim-F-at-bot : (F  $\longrightarrow$  0) at-bot and
    lim-F-at-top : (F  $\longrightarrow$  1) at-top
  shows real-distribution (interval-measure F)
proof –
  let ?F = interval-measure F
  interpret prob-space ?F
  proof qed (use interval-measure-UNIV[OF assms] in simp)
  show ?thesis
  proof qed simp-all
qed

lemma
  fixes F :: real  $\Rightarrow$  real
  assumes nondecF :  $\bigwedge x y. x \leq y \implies F x \leq F y$  and
    right-cont-F :  $\bigwedge a. \text{continuous (at-right } a) F$  and
    lim-F-at-bot : (F  $\longrightarrow$  0) at-bot
  shows emeasure-interval-measure-Iic: emeasure (interval-measure F) {.. x} = F x
  and measure-interval-measure-Iic: measure (interval-measure F) {.. x} = F x
  unfolding cdf-def
proof –
  have F-nonneg[simp]:  $0 \leq F y$  for y
    using lim-F-at-bot by (rule tendsto-upperbound) (auto simp: eventually-at-bot-linorder
nondecF intro!: exI[of - y])

  have emeasure (interval-measure F) ( $\bigcup i::\text{nat. } \{-\text{real } i <.. x\}$ ) = F x – ennreal
  0
  proof (intro LIMSEQ-unique[OF Lim-emeasure-incseq])
    have ( $\lambda i. F x - F (-\text{real } i)$ )  $\longrightarrow$  F x – 0
    by (intro tendsto-intros lim-F-at-bot[THEN filterlim-compose] filterlim-real-sequentially)

```

```

      filterlim-uminus-at-top[THEN iffD1])
    from tendsto-ennreal[OF this]
  show  $(\lambda i. \text{emeasure } (interval\text{-measure } F) \{- \text{real } i <.. x\}) \longrightarrow F x - \text{ennreal}$ 
0
    apply (rule filterlim-cong[THEN iffD1, rotated 3])
      apply simp
      apply simp
    apply (rule eventually-sequentiallyI[where c=nat (ceiling (- x))])
    apply (simp add: emeasure-interval-measure-Ioc right-cont-F nondecF)
  done
qed (auto simp: incseq-def)
also have  $(\bigcup i::nat. \{- \text{real } i <.. x\}) = \{..x\}$ 
  by auto (metis minus-minus neg-less-iff-less reals-Archimedean2)
finally show  $\text{emeasure } (interval\text{-measure } F) \{..x\} = F x$ 
  by simp
then show  $\text{measure } (interval\text{-measure } F) \{..x\} = F x$ 
  by (simp add: measure-def)
qed

```

lemma *cdf-interval-measure*:

```

 $(\bigwedge x y. x \leq y \implies F x \leq F y) \implies (\bigwedge a. \text{continuous } (at\text{-right } a) F) \implies (F \longrightarrow$ 
0) at-bot  $\implies \text{cdf } (interval\text{-measure } F) = F$ 
  by (simp add: cdf-def fun-eq-iff measure-interval-measure-Iic)

```

end

3 Weak Convergence of Functions and Distributions

Properties of weak convergence of functions and measures, including the portmanteau theorem.

theory *Weak-Convergence*

imports *Distribution-Functions*

begin

4 Weak Convergence of Functions

definition

```

weak-conv :: (nat  $\Rightarrow$  (real  $\Rightarrow$  real))  $\Rightarrow$  (real  $\Rightarrow$  real)  $\Rightarrow$  bool

```

where

```

weak-conv F-seq F  $\equiv \forall x. \text{isCont } F x \longrightarrow (\lambda n. F\text{-seq } n x) \longrightarrow F x$ 

```

5 Weak Convergence of Distributions

definition

```

weak-conv-m :: (nat  $\Rightarrow$  real measure)  $\Rightarrow$  real measure  $\Rightarrow$  bool

```

where

weak-conv-m $M\text{-seq } M \equiv \text{weak-conv } (\lambda n. \text{cdf } (M\text{-seq } n)) (\text{cdf } M)$

6 Skorohod’s theorem

locale *right-continuous-mono* =

fixes $f :: \text{real} \Rightarrow \text{real}$ and $a\ b :: \text{real}$

assumes *cont*: $\bigwedge x. \text{continuous } (\text{at-right } x) f$

assumes *mono*: *mono* f

assumes *bot*: $(f \longrightarrow a) \text{ at-bot}$

assumes *top*: $(f \longrightarrow b) \text{ at-top}$

begin

abbreviation $I :: \text{real} \Rightarrow \text{real}$ where

$I\ \omega \equiv \text{Inf } \{x. \omega \leq f\ x\}$

lemma *pseudoinverse*: assumes $a < \omega < b$ shows $\omega \leq f\ x \longleftrightarrow I\ \omega \leq x$

proof

let $?F = \{x. \omega \leq f\ x\}$

obtain y where $f\ y < \omega$

by (*metis eventually-happens' trivial-limit-at-bot-linorder order-tendstoD(2) bot*
($a < \omega$))

with *mono* have *bdd*: *bdd-below* $?F$

by (*auto intro!*: *bdd-belowI*[*of* - y] *elim*: *mono-invE*[*OF* - *less-le-trans*])

have *ne*: $?F \neq \{\}$

using *order-tendstoD(1)*[*OF top* ($\omega < b$)]

by (*auto dest!*: *eventually-happens'*[*OF trivial-limit-at-top-linorder*] *intro*: *less-imp-le*)

show $\omega \leq f\ x \implies I\ \omega \leq x$

by (*auto intro!*: *cInf-lower bdd*)

{ assume *: $I\ \omega \leq x$

have $\omega \leq (\text{INF } s:\{x. \omega \leq f\ x\}. f\ s)$

by (*rule cINF-greatest*[*OF ne*]) *auto*

also have $\dots = f\ (I\ \omega)$

using *continuous-at-Inf-mono*[*OF mono cont ne bdd*] ..

also have $\dots \leq f\ x$

using * by (*rule monoD*[*OF* ($\langle \text{mono } f \rangle$)])

finally show $\omega \leq f\ x$. }

qed

lemma *pseudoinverse'*: $\forall \omega \in \{a <..< b\}. \forall x. \omega \leq f\ x \longleftrightarrow I\ \omega \leq x$

by (*intro ballI allI impI pseudoinverse*) *auto*

lemma *mono-I*: *mono-on* $I\ \{a <..< b\}$

unfolding *mono-on-def* by (*metis order.trans order.refl pseudoinverse'*)

end


```

locale cdf-distribution = real-distribution
begin

abbreviation  $C \equiv \text{cdf } M$ 

sublocale right-continuous-mono C 0 1
  by standard
  (auto intro: cdf-nondecreasing cdf-is-right-cont cdf-lim-at-top-prob cdf-lim-at-bot monoI)

lemma measurable-C[measurable]: C ∈ borel-measurable borel
  by (intro borel-measurable-mono mono)

lemma measurable-CI[measurable]: I ∈ borel-measurable (restrict-space borel {0 < .. < 1})
  by (intro borel-measurable-mono-on-fnc mono-I)

lemma emeasure-distr-I: emeasure (distr (restrict-space lborel {0 < .. < 1 :: real}) borel I) UNIV = 1
  by (simp add: emeasure-distr space-restrict-space emeasure-restrict-space)

lemma distr-I-eq-M: distr (restrict-space lborel {0 < .. < 1 :: real}) borel I = M (is ?I = -)
proof (intro cdf-unique ext)
  let  $?\Omega = \text{restrict-space lborel } \{0 < .. < 1\} :: \text{real measure}$ 
  interpret  $\Omega: \text{prob-space } ?\Omega$ 
  by (auto simp add: emeasure-restrict-space space-restrict-space intro!: prob-spaceI)
  show real-distribution ?I
  by auto

  fix  $x$ 
  have  $\text{cdf } ?I \ x = \text{measure lborel } \{\omega \in \{0 < .. < 1\}. \omega \leq C \ x\}$ 
  by (subst cdf-def)
  (auto simp: pseudoinverse[symmetric] measure-distr space-restrict-space measure-restrict-space)
  intro!: arg-cong2[where f=measure]
  also have  $\dots = \text{measure lborel } \{0 < .. < C \ x\}$ 
  using cdf-bounded-prob[of x] AE-lborel-singleton[of C x]
  by (auto intro!: arg-cong[where f=enn2real] emeasure-eq-AE simp: measure-def)
  also have  $\dots = C \ x$ 
  by (simp add: cdf-nonneg)
  finally show  $\text{cdf (distr } ?\Omega \ \text{borel } I) \ x = C \ x$  .
qed standard

end

context
  fixes  $\mu :: \text{nat} \Rightarrow \text{real measure}$ 
  and  $M :: \text{real measure}$ 

```

assumes μ : $\bigwedge n$. *real-distribution* (μ n)
assumes M : *real-distribution* M
assumes μ -to- M : *weak-conv-m* μ M
begin

theorem *Skorohod*:

\exists (Ω :: *real measure*) (Y -seq :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real}$) (Y :: $\text{real} \Rightarrow \text{real}$).
prob-space Ω \wedge
 $(\forall n$. Y -seq $n \in \text{measurable } \Omega \text{ borel}) \wedge$
 $(\forall n$. $\text{distr } \Omega \text{ borel } (Y\text{-seq } n) = \mu$ $n) \wedge$
 $Y \in \text{measurable } \Omega \text{ lborel} \wedge$
 $\text{distr } \Omega \text{ borel } Y = M \wedge$
 $(\forall x \in \text{space } \Omega. (\lambda n. Y\text{-seq } n \ x) \longrightarrow Y \ x)$

proof –

interpret μ : *cdf-distribution* μ n **for** n
unfolding *cdf-distribution-def* **by** (*rule* μ)
interpret M : *cdf-distribution* M
unfolding *cdf-distribution-def* **by** (*rule* M)

have *conv*: *measure* $M \ \{x\} = 0 \implies (\lambda n. \mu.C \ n \ x) \longrightarrow M.C \ x$ **for** x
using μ -to- M *M.isCont-cdf* **by** (*auto simp: weak-conv-m-def weak-conv-def*)

let $? \Omega = \text{restrict-space lborel } \{0 <..< 1\}$:: *real measure*

have *prob-space* $? \Omega$

by (*auto simp: space-restrict-space emeasure-restrict-space intro!: prob-spaceI*)

interpret Ω : *prob-space* $? \Omega$

by *fact*

have Y -*distr*: *distr* $? \Omega$ *borel* $M.I = M$

by (*rule* M .*distr-I-eq-M*)

have Y -*cts-cnvc*: $(\lambda n. \mu.I \ n \ \omega) \longrightarrow M.I \ \omega$

if ω : $\omega \in \{0 <..< 1\}$ *isCont* $M.I \ \omega$ **for** ω :: *real*

proof (*intro limsup-le-liminf-real*)

show *liminf* $(\lambda n. \mu.I \ n \ \omega) \geq M.I \ \omega$

unfolding *le-Liminf-iff*

proof *safe*

fix B :: *ereal* **assume** B : $B < M.I \ \omega$

then show $\forall_F n$ *in sequentially*. $B < \mu.I \ n \ \omega$

proof (*cases* B)

case (*real* r)

with B **have** r : $r < M.I \ \omega$

by *simp*

then obtain x **where** x : $r < x < M.I \ \omega$ *measure* $M \ \{x\} = 0$

using *open-minus-countable[OF M.countable-support, of $\{r <..< M.I \ \omega\}$]*

by *auto*

then have Fx -*less*: $M.C \ x < \omega$

using M .*pseudoinverse'* ω *not-less* **by** *blast*

```

have  $\forall_F n$  in sequentially.  $\mu.C n x < \omega$ 
  using order-tendstoD(2)[OF conv[OF x(3)] Fx-less] .
then have  $\forall_F n$  in sequentially.  $x < \mu.I n \omega$ 
  by eventually-elim (insert  $\omega$   $\mu.pseudoinverse[symmetric]$ , simp add:
not-le[symmetric])
  then show ?thesis
  by eventually-elim (insert x(1), simp add: real)
qed auto
qed

have *:  $\limsup (\lambda n. \mu.I n \omega) \leq M.I \omega'$ 
if  $\omega': 0 < \omega' \omega' < 1 \omega < \omega'$  for  $\omega' :: real$ 
proof (rule dense-ge-bounded)
fix B' assume ereal (M.I  $\omega'$ ) < B' B' < ereal (M.I  $\omega' + 1$ )
then obtain B where M.I  $\omega' < B$  and [simp]: B' = ereal B
  by (cases B') auto
then obtain y where y: M.I  $\omega' < y$  y < B measure M {y} = 0
  using open-minus-countable[OF M.countable-support, of {M.I  $\omega' < .. < B$ }]
by auto
then have  $\omega' \leq M.C (M.I \omega')$ 
  using M.pseudoinverse'  $\omega'$  by (metis greaterThanLessThan-iff order-refl)
also have ...  $\leq M.C y$ 
  using M.mono y unfolding mono-def by auto
finally have Fy-gt:  $\omega < M.C y$ 
  using  $\omega'(3)$  by simp

have  $\forall_F n$  in sequentially.  $\omega \leq \mu.C n y$ 
  using order-tendstoD(1)[OF conv[OF y(3)] Fy-gt] by eventually-elim (rule
less-imp-le)
then have 2:  $\forall_F n$  in sequentially.  $\mu.I n \omega \leq ereal y$ 
  by simp (subst  $\mu.pseudoinverse'$ [rule-format, OF  $\omega(1)$ , symmetric])
then show  $\limsup (\lambda n. \mu.I n \omega) \leq B'$ 
  using  $\langle y < B \rangle$ 
by (intro Limsup-bounded[rotated]) (auto intro: le-less-trans elim: eventually-mono)
qed simp

have **: (M.I  $\longrightarrow$  ereal (M.I  $\omega$ )) (at-right  $\omega$ )
  using  $\omega(2)$  by (auto intro: tendsto-within-subset simp: continuous-at)
show  $\limsup (\lambda n. \mu.I n \omega) \leq M.I \omega$ 
  using  $\omega$ 
  by (intro tendsto-lowerbound[OF **])
  (auto intro!: exI[of - 1] * simp: eventually-at-right[of - 1])
qed

let ?D = { $\omega \in \{0 < .. < 1\}$ .  $\neg isCont M.I \omega$ }
have D-countable: countable ?D
  using mono-on-ctble-discont[OF M.mono-I] by (simp add: at-within-open[of -
{0 < .. < 1}] cong: conj-cong)

```

hence D : $\text{emeasure } ?\Omega \text{ } ?D = 0$
using $\text{emeasure-lborel-countable}[OF \ D\text{-countable}]$
by ($\text{subst emeasure-restrict-space}$) auto

define Y' **where** $Y' \omega = (\text{if } \omega \in ?D \text{ then } 0 \text{ else } M.I \omega)$ **for** ω
have $Y'\text{-AE}$: $AE \ \omega \text{ in } ?\Omega. Y' \omega = M.I \omega$
by ($\text{rule AE-I } [OF - D]$) ($\text{auto simp: space-restrict-space sets-restrict-space-iff } Y'\text{-def}$)

define $Y\text{-seq}'$ **where** $Y\text{-seq}' \ n \ \omega = (\text{if } \omega \in ?D \text{ then } 0 \text{ else } \mu.I \ n \ \omega)$ **for** $n \ \omega$
have $Y\text{-seq}'\text{-AE}$: $\bigwedge n. AE \ \omega \text{ in } ?\Omega. Y\text{-seq}' \ n \ \omega = \mu.I \ n \ \omega$
by ($\text{rule AE-I } [OF - D]$) ($\text{auto simp: space-restrict-space sets-restrict-space-iff } Y\text{-seq}'\text{-def}$)

have $Y'\text{-cnv}$: $\forall \omega \in \{0 < .. < 1\}. (\lambda n. Y\text{-seq}' \ n \ \omega) \longrightarrow Y' \ \omega$
by ($\text{auto simp: } Y'\text{-def } Y\text{-seq}'\text{-def } Y\text{-cts-cnv}$)

have [simp]: $Y\text{-seq}' \ n \in \text{borel-measurable } ?\Omega$ **for** n
by ($\text{rule measurable-discrete-difference}[of \ \mu.I \ n \ - \ ?D]$)
($\text{insert } \mu.\text{measurable-CI}[of \ n] \ D\text{-countable, auto simp: sets-restrict-space } Y\text{-seq}'\text{-def}$)

moreover have $\text{distr } ?\Omega \ \text{borel} \ (Y\text{-seq}' \ n) = \mu \ n$ **for** n
using $\mu.\text{distr-I-eq-M} \ [of \ n] \ Y\text{-seq}'\text{-AE} \ [of \ n]$
by ($\text{subst distr-cong-AE}[\text{where } f = Y\text{-seq}' \ n \ \text{and } g = \mu.I \ n], \text{ auto}$)

moreover have [simp]: $Y' \in \text{borel-measurable } ?\Omega$
by ($\text{rule measurable-discrete-difference}[of \ M.I \ - \ ?D]$)
($\text{insert } M.\text{measurable-CI} \ D\text{-countable, auto simp: sets-restrict-space } Y'\text{-def}$)

moreover have $\text{distr } ?\Omega \ \text{borel} \ Y' = M$
using $M.\text{distr-I-eq-M} \ Y'\text{-AE}$
by ($\text{subst distr-cong-AE}[\text{where } f = Y' \ \text{and } g = M.I], \text{ auto}$)

ultimately have $\text{prob-space } ?\Omega \wedge (\forall n. Y\text{-seq}' \ n \in \text{borel-measurable } ?\Omega) \wedge$
 $(\forall n. \text{distr } ?\Omega \ \text{borel} \ (Y\text{-seq}' \ n) = \mu \ n) \wedge Y' \in \text{measurable } ?\Omega \ \text{lborel} \wedge \text{distr } ?\Omega$
 $\text{borel } Y' = M \wedge$
 $(\forall x \in \text{space } ?\Omega. (\lambda n. Y\text{-seq}' \ n \ x) \longrightarrow Y' \ x)$
using $Y'\text{-cnv} \ \langle \text{prob-space } ?\Omega \rangle$ **by** ($\text{auto simp: space-restrict-space}$)

thus $?thesis$ **by** metis

qed

The Portmanteau theorem, that is, the equivalence of various definitions of weak convergence.

theorem $\text{weak-conv-imp-bdd-ae-continuous-conv}$:

fixes

$f :: \text{real} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$

assumes

$\text{discont-null}: M \ (\{x. \neg \text{isCont } f \ x\}) = 0$ **and**

$f\text{-bdd}: \bigwedge x. \text{norm} \ (f \ x) \leq B$ **and**

[measurable]: $f \in \text{borel-measurable borel}$

shows

$(\lambda n. \text{integral}^L \ (\mu \ n) \ f) \longrightarrow \text{integral}^L \ M \ f$

proof –

have $0 \leq B$

using *norm-ge-zero f-bdd* **by** (*rule order-trans*)

note *Skorohod*

then obtain *Omega Y-seq Y* **where**

ps-Omega [simp]: prob-space Omega **and**

Y-seq-measurable [measurable]: $\bigwedge n. Y\text{-seq } n \in \text{borel-measurable } \text{Omega}$ **and**

distr-Y-seq: $\bigwedge n. \text{distr } \text{Omega borel } (Y\text{-seq } n) = \mu \ n$ **and**

Y-measurable [measurable]: $Y \in \text{borel-measurable } \text{Omega}$ **and**

distr-Y: $\text{distr } \text{Omega borel } Y = M$ **and**

YN Y: $\bigwedge x :: \text{real}. x \in \text{space } \text{Omega} \implies (\lambda n. Y\text{-seq } n \ x) \longrightarrow Y \ x$ **by force**

interpret *prob-space Omega* **by fact**

have $*$: *emeasure Omega* ($Y - \{x. \neg \text{isCont } f \ x\} \cap \text{space } \text{Omega}$) = 0

by (*subst emeasure-distr [symmetric, where N=borel]*) (*auto simp: distr-Y*
discont-null)

have $*$: *AE x in Omega. $(\lambda n. f \ (Y\text{-seq } n \ x)) \longrightarrow f \ (Y \ x)$*

by (*rule AE-I [OF - *]*) (*auto intro: isCont-tendsto-compose YN Y*)

show *?thesis*

by (*auto intro!: integral-dominated-convergence[where w= $\lambda x. B$]*

*simp: f-bdd * integral-distr distr-Y-seq [symmetric] distr-Y [symmetric]*)

qed

theorem *weak-conv-imp-integral-bdd-continuous-conv:*

fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$

assumes

$\bigwedge x. \text{isCont } f \ x$ **and**

$\bigwedge x. \text{norm } (f \ x) \leq B$

shows

$(\lambda n. \text{integral}^L (\mu \ n) \ f) \longrightarrow \text{integral}^L \ M \ f$

using *assms*

by (*intro weak-conv-imp-bdd-ae-continuous-conv*)

(*auto intro!: borel-measurable-continuous-on1 continuous-at-imp-continuous-on*)

theorem *weak-conv-imp-continuity-set-conv:*

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes [*measurable*]: $A \in \text{sets borel}$ **and** $M (\text{frontier } A) = 0$

shows $(\lambda n. \text{measure } (\mu \ n) \ A) \longrightarrow \text{measure } M \ A$

proof –

interpret M : *real-distribution M* **by fact**

interpret μ : *real-distribution $\mu \ n$ for n* **by fact**

have $(\lambda n. (\int x. \text{indicator } A \ x \ \partial \mu \ n) :: \text{real}) \longrightarrow (\int x. \text{indicator } A \ x \ \partial M)$

by (*intro weak-conv-imp-bdd-ae-continuous-conv[where B=1]*)

(*auto intro: assms simp: isCont-indicator*)

then show *?thesis*

by *simp*

qed

end

definition

$cts\text{-}step :: real \Rightarrow real \Rightarrow real \Rightarrow real$

where

$cts\text{-}step\ a\ b\ x \equiv \text{if } x \leq a \text{ then } 1 \text{ else if } x \geq b \text{ then } 0 \text{ else } (b - x) / (b - a)$

lemma *cts-step-uniformly-continuous:*

assumes $[arith]: a < b$

shows *uniformly-continuous-on UNIV* ($cts\text{-}step\ a\ b$)

unfolding *uniformly-continuous-on-def*

proof *clarsimp*

fix $e :: real$ **assume** $[arith]: 0 < e$

let $?d = \min (e * (b - a)) (b - a)$

have $?d > 0$

by (*auto simp add: field-simps*)

moreover have $dist\ x'\ x < ?d \implies dist\ (cts\text{-}step\ a\ b\ x')\ (cts\text{-}step\ a\ b\ x) < e$

for $x\ x'$

by (*auto simp: dist-real-def divide-simps cts-step-def*)

ultimately show $\exists d > 0. \forall x\ x'. dist\ x'\ x < d \implies dist\ (cts\text{-}step\ a\ b\ x')\ (cts\text{-}step\ a\ b\ x) < e$

by *blast*

qed

lemma (*in real-distribution*) *integrable-cts-step:* $a < b \implies integrable\ M\ (cts\text{-}step\ a\ b)$

by (*rule integrable-const-bound [of - 1] (auto simp: cts-step-def[abs-def])*)

lemma (*in real-distribution*) *cdf-cts-step:*

assumes $[arith]: x < y$

shows $cdf\ M\ x \leq integral^L\ M\ (cts\text{-}step\ x\ y)$ **and** $integral^L\ M\ (cts\text{-}step\ x\ y) \leq cdf\ M\ y$

proof –

have $cdf\ M\ x = integral^L\ M\ (indicator\ \{..x\})$

by (*simp add: cdf-def*)

also have $\dots \leq expectation\ (cts\text{-}step\ x\ y)$

by (*intro integral-mono integrable-cts-step*)

(*auto simp: cts-step-def less-top[symmetric] split: split-indicator*)

finally show $cdf\ M\ x \leq expectation\ (cts\text{-}step\ x\ y)$.

next

have $expectation\ (cts\text{-}step\ x\ y) \leq integral^L\ M\ (indicator\ \{..y\})$

by (*intro integral-mono integrable-cts-step*)

(*auto simp: cts-step-def less-top[symmetric] split: split-indicator*)

also have $\dots = cdf\ M\ y$

by (*simp add: cdf-def*)

finally show $expectation\ (cts\text{-}step\ x\ y) \leq cdf\ M\ y$.

qed

context

fixes $M\text{-}seq :: nat \Rightarrow real\ measure$

```

    and M :: real measure
    assumes distr-M-seq [simp]:  $\bigwedge n. \text{real-distribution } (M\text{-seq } n)$ 
    assumes distr-M [simp]:  $\text{real-distribution } M$ 
begin

theorem continuity-set-conv-imp-weak-conv:
  fixes f :: real  $\Rightarrow$  real
  assumes *:  $\bigwedge A. A \in \text{sets borel} \implies M (\text{frontier } A) = 0 \implies (\lambda n. (\text{measure } (M\text{-seq } n) A)) \longrightarrow \text{measure } M A$ 
  shows weak-conv-m M-seq M
proof -
  interpret real-distribution M by simp
  show ?thesis
    by (auto intro!: * simp: frontier-real-Iic isCont-cdf emeasure-eq-measure weak-conv-m-def weak-conv-def cdf-def2)
qed

theorem integral-cts-step-conv-imp-weak-conv:
  assumes integral-conv:  $\bigwedge x y. x < y \implies (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } x y)) \longrightarrow \text{integral}^L M (\text{cts-step } x y)$ 
  shows weak-conv-m M-seq M
  unfolding weak-conv-m-def weak-conv-def
proof (clarsimp)
  interpret real-distribution M by (rule distr-M)
  fix x assume isCont (cdf M) x
  hence left-cont: continuous-at-left x (cdf M)
    unfolding continuous-at-split ..
  { fix y :: real assume [arith]:  $x < y$ 
    have limsup  $(\lambda n. \text{cdf } (M\text{-seq } n) x) \leq \text{limsup } (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } x y))$ 
      by (auto intro!: Limsup-mono always-eventually real-distribution.cdf-cts-step)
    also have  $\dots = \text{integral}^L M (\text{cts-step } x y)$ 
      by (intro lim-imp-Limsup) (auto intro: integral-conv)
    also have  $\dots \leq \text{cdf } M y$ 
      by (simp add: cdf-cts-step)
    finally have  $\text{limsup } (\lambda n. \text{cdf } (M\text{-seq } n) x) \leq \text{cdf } M y$  .
  }
  note * = this
  { fix y :: real assume [arith]:  $x > y$ 
    have  $\text{cdf } M y \leq \text{ereal } (\text{integral}^L M (\text{cts-step } y x))$ 
      by (simp add: cdf-cts-step)
    also have  $\dots = \text{liminf } (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } y x))$ 
      by (intro lim-imp-Liminf[symmetric]) (auto intro: integral-conv)
    also have  $\dots \leq \text{liminf } (\lambda n. \text{cdf } (M\text{-seq } n) x)$ 
      by (auto intro!: Liminf-mono always-eventually real-distribution.cdf-cts-step)
    finally have  $\text{liminf } (\lambda n. \text{cdf } (M\text{-seq } n) x) \geq \text{cdf } M y$  .
  }
  note ** = this

  have  $\text{limsup } (\lambda n. \text{cdf } (M\text{-seq } n) x) \leq \text{cdf } M x$ 
  proof (rule tendsto-lowerbound)

```

```

show  $\forall_F i$  in at-right  $x$ .  $\text{limsup } (\lambda x a. \text{ereal } (\text{cdf } (M\text{-seq } xa) x)) \leq \text{ereal } (\text{cdf } M i)$ 
  by (subst eventually-at-right[of -  $x + 1$ ]) (auto simp: * intro: exI [of -  $x + 1$ ])
qed (insert cdf-is-right-cont, auto simp: continuous-within)
moreover have  $\text{cdf } M x \leq \text{liminf } (\lambda n. \text{cdf } (M\text{-seq } n) x)$ 
proof (rule tendsto-upperbound)
  show  $\forall_F i$  in at-left  $x$ .  $\text{ereal } (\text{cdf } M i) \leq \text{liminf } (\lambda x a. \text{ereal } (\text{cdf } (M\text{-seq } xa) x))$ 
    by (subst eventually-at-left[of  $x - 1$ ]) (auto simp: ** intro: exI [of -  $x - 1$ ])
    qed (insert left-cont, auto simp: continuous-within)
  ultimately show  $(\lambda n. \text{cdf } (M\text{-seq } n) x) \longrightarrow \text{cdf } M x$ 
    by (elim limsup-le-liminf-real)
qed

```

theorem *integral-bdd-continuous-conv-imp-weak-conv:*

```

assumes
   $\bigwedge f. (\bigwedge x. \text{isCont } f x) \implies (\bigwedge x. \text{abs } (f x) \leq 1) \implies (\lambda n. \text{integral}^L (M\text{-seq } n) f :: \text{real}) \longrightarrow \text{integral}^L M f$ 
shows
  weak-conv-m  $M\text{-seq } M$ 
apply (rule integral-cts-step-conv-imp-weak-conv [OF assms])
apply (rule continuous-on-interior)
apply (rule uniformly-continuous-imp-continuous)
apply (rule cts-step-uniformly-continuous)
apply (auto simp: cts-step-def)
done

```

end

end

theory *Giry-Monad*

```

imports Probability-Measure HOL-Library.Monad-Syntax
begin

```

7 Sub-probability spaces

```

locale subprob-space = finite-measure +
  assumes emeasure-space-le-1: emeasure M (space M) ≤ 1
  assumes subprob-not-empty: space M ≠ {}

```

```

lemma subprob-spaceI[Pure.intro!]:
  assumes *: emeasure M (space M) ≤ 1
  assumes space M ≠ {}
  shows subprob-space M
proof –
  interpret finite-measure M
  proof

```


show $\text{emeasure } M (\text{space } M) \neq \infty$ **using** $*$ **by** (*auto simp: top-unique*)
qed
show $\text{subprob-space } M$ **by** *standard fact+*
qed

lemma (*in subprob-space*) *emeasure-subprob-space-less-top*: $\text{emeasure } M A \neq \text{top}$
using *emeasure-finite[of A]* .

lemma *prob-space-imp-subprob-space*:
 $\text{prob-space } M \implies \text{subprob-space } M$
by (*rule subprob-spaceI*) (*simp-all add: prob-space.emeasure-space-1 prob-space.not-empty*)

lemma *subprob-space-imp-sigma-finite*: $\text{subprob-space } M \implies \text{sigma-finite-measure } M$
unfolding *subprob-space-def finite-measure-def* **by** *simp*

sublocale $\text{prob-space} \subseteq \text{subprob-space}$
by (*rule subprob-spaceI*) (*simp-all add: emeasure-space-1 not-empty*)

lemma *subprob-space-sigma* [*simp*]: $\Omega \neq \{\}$ $\implies \text{subprob-space } (\text{sigma } \Omega X)$
by(*rule subprob-spaceI*)(*simp-all add: emeasure-sigma space-measure-of-conv*)

lemma *subprob-space-null-measure*: $\text{space } M \neq \{\}$ $\implies \text{subprob-space } (\text{null-measure } M)$
by(*simp add: null-measure-def*)

lemma (*in subprob-space*) *subprob-space-distr*:
assumes $f: f \in \text{measurable } M M'$ **and** $\text{space } M' \neq \{\}$ **shows** $\text{subprob-space } (\text{distr } M M' f)$
proof (*rule subprob-spaceI*)
have $f - ' \text{space } M' \cap \text{space } M = \text{space } M$ **using** f **by** (*auto dest: measurable-space*)
with f **show** $\text{emeasure } (\text{distr } M M' f) (\text{space } (\text{distr } M M' f)) \leq 1$
by (*auto simp: emeasure-distr emeasure-space-le-1*)
show $\text{space } (\text{distr } M M' f) \neq \{\}$ **by** (*simp add: assms*)
qed

lemma (*in subprob-space*) *subprob-emeasure-le-1*: $\text{emeasure } M X \leq 1$
by (*rule order.trans[OF emeasure-space emeasure-space-le-1]*)

lemma (*in subprob-space*) *subprob-measure-le-1*: $\text{measure } M X \leq 1$
using *subprob-emeasure-le-1[of X]* **by** (*simp add: emeasure-eq-measure*)

lemma (*in subprob-space*) *nn-integral-le-const*:
assumes $0 \leq c$ *AE* x *in* M . $f x \leq c$
shows $(\int^+ x. f x \partial M) \leq c$
proof –
have $(\int^+ x. f x \partial M) \leq (\int^+ x. c \partial M)$
by(*rule nn-integral-mono-AE*) *fact*
also have $\dots \leq c * \text{emeasure } M (\text{space } M)$

using $\langle 0 \leq c \rangle$ **by** *simp*
also have $\dots \leq c * 1$ **using** *emeasure-space-le-1* $\langle 0 \leq c \rangle$ **by** (*rule mult-left-mono*)
finally show *?thesis* **by** *simp*
qed

lemma *emeasure-density-distr-interval*:

fixes $h :: \text{real} \Rightarrow \text{real}$ **and** $g :: \text{real} \Rightarrow \text{real}$ **and** $g' :: \text{real} \Rightarrow \text{real}$
assumes [*simp*]: $a \leq b$
assumes *Mf*[*measurable*]: $f \in \text{borel-measurable borel}$
assumes *Mg*[*measurable*]: $g \in \text{borel-measurable borel}$
assumes *Mg'*[*measurable*]: $g' \in \text{borel-measurable borel}$
assumes *Mh*[*measurable*]: $h \in \text{borel-measurable borel}$
assumes *prob*: *subprob-space* (*density lborel f*)
assumes *nonnegf*: $\bigwedge x. f\ x \geq 0$
assumes *derivg*: $\bigwedge x. x \in \{a..b\} \implies (g \text{ has-real-derivative } g' x) \text{ (at } x)$
assumes *contg'*: *continuous-on* $\{a..b\}$ g'
assumes *mono*: *strict-mono-on* $g \{a..b\}$ **and** *inv*: $\bigwedge x. h\ x \in \{a..b\} \implies g\ (h\ x)$
 $= x$
assumes *range*: $\{a..b\} \subseteq \text{range } h$
shows *emeasure* (*distr* (*density lborel f*) *lborel h*) $\{a..b\} =$
emeasure (*density lborel* ($\lambda x. f\ (g\ x) * g'\ x$)) $\{a..b\}$

proof (*cases a < b*)

assume $a < b$
from *mono* **have** *inj*: *inj-on* $g \{a..b\}$ **by** (*rule strict-mono-on-imp-inj-on*)
from *mono* **have** *mono'*: *mono-on* $g \{a..b\}$ **by** (*rule strict-mono-on-imp-mono-on*)
from *mono'* *derivg* **have** $\bigwedge x. x \in \{a <..< b\} \implies g'\ x \geq 0$
by (*rule mono-on-imp-deriv-nonneg*) *auto*
from *contg'* **this** **have** *derivg-nonneg*: $\bigwedge x. x \in \{a..b\} \implies g'\ x \geq 0$
by (*rule continuous-ge-on-Ioo*) (*simp-all add:* $\langle a < b \rangle$)

from *derivg* **have** *contg*: *continuous-on* $\{a..b\}$ g **by** (*rule has-real-derivative-imp-continuous-on*)

have *A*: $h - \cdot \{a..b\} = \{g\ a..g\ b\}$

proof (*intro equalityI subsetI*)

fix x **assume** $x \in h - \cdot \{a..b\}$

hence $g\ (h\ x) \in \{g\ a..g\ b\}$ **by** (*auto intro: mono-onD[OF mono']*)

with *inv* **and** x **show** $x \in \{g\ a..g\ b\}$ **by** *simp*

next

fix y **assume** $y \in \{g\ a..g\ b\}$

with *IVT'*[*OF* - - - *contg*, *of y*] **obtain** x **where** $x \in \{a..b\}$ $y = g\ x$ **by** *auto*

with *range* **and** *inv* **show** $y \in h - \cdot \{a..b\}$ **by** *auto*

qed

have *prob'*: *subprob-space* (*distr* (*density lborel f*) *lborel h*)

by (*rule subprob-space.subprob-space-distr[OF prob]*) (*simp-all add: Mh*)

have *B*: *emeasure* (*distr* (*density lborel f*) *lborel h*) $\{a..b\} =$

$$\int^+ x. f\ x * \text{indicator}\ (h - \cdot \{a..b\})\ x\ \partial \text{lborel}$$

by (*subst emeasure-distr*) (*simp-all add: emeasure-density Mf Mh measurable-sets-borel[OF Mh]*)

also note *A*

also have $\text{emeasure } (\text{distr } (\text{density } \text{lborel } f) \text{ lborel } h) \{a..b\} \leq 1$
by $(\text{rule } \text{subprob-space.subprob-emeasure-le-1}) (\text{rule } \text{prob}')$
hence $\text{emeasure } (\text{distr } (\text{density } \text{lborel } f) \text{ lborel } h) \{a..b\} \neq \infty$ **by** $(\text{auto } \text{simp: top-unique})$
with assms **have** $(\int^+ x. f x * \text{indicator } \{g a..g b\} x \partial \text{lborel}) =$
 $(\int^+ x. f (g x) * g' x * \text{indicator } \{a..b\} x \partial \text{lborel})$
by $(\text{intro } \text{nn-integral-substitution-aux})$
 $(\text{auto } \text{simp: derivg-nonneg } A \ B \ \text{emeasure-density } \text{mult.commute } \langle a < b \rangle)$
also have $\dots = \text{emeasure } (\text{density } \text{lborel } (\lambda x. f (g x) * g' x)) \{a..b\}$
by $(\text{simp } \text{add: emeasure-density})$
finally show $?thesis$.
next
assume $\neg a < b$
with $\langle a \leq b \rangle$ **have** $[\text{simp}]: b = a$ **by** $(\text{simp } \text{add: not-less } \text{del: } \langle a \leq b \rangle)$
from inv **and** range **have** $h^{-1} \{a\} = \{g a\}$ **by** auto
thus $?thesis$ **by** $(\text{simp-all } \text{add: emeasure-distr } \text{emeasure-density } \text{measurable-sets-borel}[\text{OF } Mh])$
qed

locale $\text{pair-subprob-space} =$
 $\text{pair-sigma-finite } M1 \ M2 + M1: \text{subprob-space } M1 + M2: \text{subprob-space } M2$ **for**
 $M1 \ M2$

sublocale $\text{pair-subprob-space} \subseteq P?: \text{subprob-space } M1 \otimes_M M2$

proof

from $\text{mult-le-one}[\text{OF } M1.\text{emeasure-space-le-1} - M2.\text{emeasure-space-le-1}]$
show $\text{emeasure } (M1 \otimes_M M2) (\text{space } (M1 \otimes_M M2)) \leq 1$
by $(\text{simp } \text{add: } M2.\text{emeasure-pair-measure-Times } \text{space-pair-measure})$
from $M1.\text{subprob-not-empty}$ **and** $M2.\text{subprob-not-empty}$ **show** $\text{space } (M1 \otimes_M M2) \neq \{\}$
by $(\text{simp } \text{add: space-pair-measure})$
qed

lemma $\text{subprob-space-null-measure-iff}$:

$\text{subprob-space } (\text{null-measure } M) \longleftrightarrow \text{space } M \neq \{\}$
by $(\text{auto } \text{intro!: subprob-spaceI } \text{dest: subprob-space.subprob-not-empty})$

lemma $\text{subprob-space-restrict-space}$:

assumes $M: \text{subprob-space } M$
and $A: A \cap \text{space } M \in \text{sets } M \ A \cap \text{space } M \neq \{\}$
shows $\text{subprob-space } (\text{restrict-space } M \ A)$
proof $(\text{rule } \text{subprob-spaceI})$
have $\text{emeasure } (\text{restrict-space } M \ A) (\text{space } (\text{restrict-space } M \ A)) = \text{emeasure } M$
 $(A \cap \text{space } M)$
using A **by** $(\text{simp } \text{add: emeasure-restrict-space } \text{space-restrict-space})$
also have $\dots \leq 1$ **by** $(\text{rule } \text{subprob-space.subprob-emeasure-le-1})(\text{rule } M)$
finally show $\text{emeasure } (\text{restrict-space } M \ A) (\text{space } (\text{restrict-space } M \ A)) \leq 1$.

next

show $\text{space } (\text{restrict-space } M \ A) \neq \{\}$

using A **by** (*simp add: space-restrict-space*)
qed

definition *subprob-algebra* :: 'a measure \Rightarrow 'a measure measure **where**
subprob-algebra $K =$
 (*SUP* $A : \text{sets } K. \text{vimage-algebra } \{M. \text{subprob-space } M \wedge \text{sets } M = \text{sets } K\}$
 ($\lambda M. \text{emeasure } M A$) *borel*)

lemma *space-subprob-algebra*: *space* (*subprob-algebra* A) = $\{M. \text{subprob-space } M \wedge \text{sets } M = \text{sets } A\}$
by (*auto simp add: subprob-algebra-def space-Sup-eq-UN*)

lemma *subprob-algebra-cong*: *sets* $M = \text{sets } N \implies \text{subprob-algebra } M = \text{subprob-algebra } N$
by (*simp add: subprob-algebra-def*)

lemma *measurable-emeasure-subprob-algebra*[*measurable*]:
 $a \in \text{sets } A \implies (\lambda M. \text{emeasure } M a) \in \text{borel-measurable } (\text{subprob-algebra } A)$
by (*auto intro!: measurable-Sup1 measurable-vimage-algebra1 simp: subprob-algebra-def*)

lemma *measurable-measure-subprob-algebra*[*measurable*]:
 $a \in \text{sets } A \implies (\lambda M. \text{measure } M a) \in \text{borel-measurable } (\text{subprob-algebra } A)$
unfolding *measure-def* **by** *measurable*

lemma *subprob-measurableD*:
assumes $N: N \in \text{measurable } M$ (*subprob-algebra* S) **and** $x: x \in \text{space } M$
shows *space* ($N x$) = *space* S
and *sets* ($N x$) = *sets* S
and *measurable* ($N x$) $K = \text{measurable } S K$
and *measurable* K ($N x$) = *measurable* $K S$
using *measurable-space[OF N x]*
by (*auto simp: space-subprob-algebra intro!: measurable-cong-sets dest: sets-eq-imp-space-eq*)

ML \langle

fun *subprob-cong thm ctxt* = (
let
 val *thm'* = *Thm.transfer* (*Proof-Context.theory-of* *ctxt*) *thm*
 val *free* = *thm'* |> *Thm.concl-of* |> *HOLogic.dest-Trueprop* |> *dest-comb* |>
fst |>
 dest-comb |> *snd* |> *strip-abs-body* |> *head-of* |> *is-Free*
in
 if *free* *then* (\square , *Measurable.add-local-cong* (*thm'* *RS* @ $\{thm \text{ subprob-measurableD}(2)\}$)
ctxt)
 else (\square , *ctxt*)
end
handle *THM* - => (\square , *ctxt*) | *TERM* - => (\square , *ctxt*)

\rangle

setup (

Context.theory-map (Measurable.add-preprocessor subprob-cong subprob-cong)

)

context

fixes $K M N$ **assumes** $K: K \in \text{measurable } M \text{ (subprob-algebra } N)$

begin

lemma *subprob-space-kernel*: $a \in \text{space } M \implies \text{subprob-space } (K a)$

using *measurable-space[OF K]* **by** (*simp add: space-subprob-algebra*)

lemma *sets-kernel*: $a \in \text{space } M \implies \text{sets } (K a) = \text{sets } N$

using *measurable-space[OF K]* **by** (*simp add: space-subprob-algebra*)

lemma *measurable-emeasure-kernel[measurable]*:

$A \in \text{sets } N \implies (\lambda a. \text{emeasure } (K a) A) \in \text{borel-measurable } M$

using *measurable-compose[OF K measurable-emeasure-subprob-algebra]* .

end

lemma *measurable-subprob-algebra*:

$(\bigwedge a. a \in \text{space } M \implies \text{subprob-space } (K a)) \implies$

$(\bigwedge a. a \in \text{space } M \implies \text{sets } (K a) = \text{sets } N) \implies$

$(\bigwedge A. A \in \text{sets } N \implies (\lambda a. \text{emeasure } (K a) A) \in \text{borel-measurable } M) \implies$

$K \in \text{measurable } M \text{ (subprob-algebra } N)$

by (*auto intro!: measurable-Sup2 measurable-vimage-algebra2 simp: subprob-algebra-def*)

lemma *measurable-submarkov*:

$K \in \text{measurable } M \text{ (subprob-algebra } M) \longleftrightarrow$

$(\forall x \in \text{space } M. \text{subprob-space } (K x) \wedge \text{sets } (K x) = \text{sets } M) \wedge$

$(\forall A \in \text{sets } M. (\lambda x. \text{emeasure } (K x) A) \in \text{measurable } M \text{ borel})$

proof

assume $(\forall x \in \text{space } M. \text{subprob-space } (K x) \wedge \text{sets } (K x) = \text{sets } M) \wedge$

$(\forall A \in \text{sets } M. (\lambda x. \text{emeasure } (K x) A) \in \text{borel-measurable } M)$

then show $K \in \text{measurable } M \text{ (subprob-algebra } M)$

by (*intro measurable-subprob-algebra auto*)

next

assume $K \in \text{measurable } M \text{ (subprob-algebra } M)$

then show $(\forall x \in \text{space } M. \text{subprob-space } (K x) \wedge \text{sets } (K x) = \text{sets } M) \wedge$

$(\forall A \in \text{sets } M. (\lambda x. \text{emeasure } (K x) A) \in \text{borel-measurable } M)$

by (*auto dest: subprob-space-kernel sets-kernel*)

qed

lemma *measurable-subprob-algebra-generated*:

assumes *eq*: $\text{sets } N = \text{sigma-sets } \Omega G$ **and** *Int-stable* $G G \subseteq \text{Pow } \Omega$

assumes *subsp*: $\bigwedge a. a \in \text{space } M \implies \text{subprob-space } (K a)$

assumes *sets*: $\bigwedge a. a \in \text{space } M \implies \text{sets } (K a) = \text{sets } N$

assumes $\bigwedge A. A \in G \implies (\lambda a. \text{emeasure } (K a) A) \in \text{borel-measurable } M$

```

assumes  $\Omega$ :  $(\lambda a. \text{emeasure } (K \ a) \ \Omega) \in \text{borel-measurable } M$ 
shows  $K \in \text{measurable } M \ (\text{subprob-algebra } N)$ 
proof  $(\text{rule measurable-subprob-algebra})$ 
  fix  $a$  assume  $a \in \text{space } M$  then show  $\text{subprob-space } (K \ a) \ \text{sets } (K \ a) = \text{sets } N$ 
by fact+
next
  interpret  $G$ :  $\text{sigma-algebra } \Omega \ \text{sigma-sets } \Omega \ G$ 
  using  $\langle G \subseteq \text{Pow } \Omega \rangle$  by  $(\text{rule sigma-algebra-sigma-sets})$ 
  fix  $A$  assume  $A \in \text{sets } N$  with  $\text{assms}(2,3)$  show  $(\lambda a. \text{emeasure } (K \ a) \ A) \in \text{borel-measurable } M$ 
  unfolding  $\langle \text{sets } N = \text{sigma-sets } \Omega \ G \rangle$ 
  proof  $(\text{induction rule: sigma-sets-induct-disjoint})$ 
  case  $(\text{basic } A)$  then show  $?case$  by fact
next
  case  $\text{empty}$  then show  $?case$  by simp
next
  case  $(\text{compl } A)$ 
  have  $(\lambda a. \text{emeasure } (K \ a) \ (\Omega - A)) \in \text{borel-measurable } M \longleftrightarrow$ 
     $(\lambda a. \text{emeasure } (K \ a) \ \Omega - \text{emeasure } (K \ a) \ A) \in \text{borel-measurable } M$ 
  using  $G.\text{top } G.\text{sets-into-space sets eq compl subprob-space.emeasure-subprob-space-less-top}[OF \ \text{subsp}]$ 
  by  $(\text{intro measurable-cong emeasure-Diff}) \ \text{auto}$ 
  with  $\text{compl } \Omega$  show  $?case$ 
  by simp
next
  case  $(\text{union } F)$ 
  moreover have  $(\lambda a. \text{emeasure } (K \ a) \ (\bigcup i. F \ i)) \in \text{borel-measurable } M \longleftrightarrow$ 
     $(\lambda a. \sum i. \text{emeasure } (K \ a) \ (F \ i)) \in \text{borel-measurable } M$ 
  using  $\text{sets union eq}$ 
  by  $(\text{intro measurable-cong suminf-emeasure}[symmetric]) \ \text{auto}$ 
  ultimately show  $?case$ 
  by auto
qed
qed

```

lemma $\text{space-subprob-algebra-empty-iff}$:

$\text{space } (\text{subprob-algebra } N) = \{\} \longleftrightarrow \text{space } N = \{\}$

proof

have $\bigwedge x. x \in \text{space } N \implies \text{density } N \ (\lambda-. \ 0) \in \text{space } (\text{subprob-algebra } N)$

by $(\text{auto simp: space-subprob-algebra emeasure-density intro!: subprob-spaceI})$

then show $\text{space } (\text{subprob-algebra } N) = \{\} \implies \text{space } N = \{\}$

by auto

next

assume $\text{space } N = \{\}$

hence $\text{sets } N = \{\{\}\}$ **by** $(\text{simp add: space-empty-iff})$

moreover have $\bigwedge M. \text{subprob-space } M \implies \text{sets } M \neq \{\{\}\}$

by $(\text{simp add: subprob-space.subprob-not-empty space-empty-iff}[symmetric])$

ultimately show $\text{space } (\text{subprob-algebra } N) = \{\}$ **by** $(\text{auto simp: space-subprob-algebra})$

qed

lemma *nn-integral-measurable-subprob-algebra*[*measurable*]:
assumes $f: f \in \text{borel-measurable } N$
shows $(\lambda M. \text{integral}^N M f) \in \text{borel-measurable } (\text{subprob-algebra } N)$ (**is** $- \in ?B$)
using f
proof *induct*
case (*cong f g*)
moreover have $(\lambda M'. \int^+ M''. f M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \int^+ M''. g M'' \partial M') \in ?B$
by (*intro measurable-cong nn-integral-cong cong*)
(auto simp: space-subprob-algebra dest!: sets-eq-imp-space-eq)
ultimately show *?case* **by** *simp*
next
case (*set B*)
then have $(\lambda M'. \int^+ M''. \text{indicator } B M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \text{emeasure } M' B) \in ?B$
by (*intro measurable-cong nn-integral-indicator*) (*simp add: space-subprob-algebra*)
with *set show ?case*
by (*simp add: measurable-emeasure-subprob-algebra*)
next
case (*mult f c*)
then have $(\lambda M'. \int^+ M''. c * f M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. c * \int^+ M''. f M'' \partial M') \in ?B$
by (*intro measurable-cong nn-integral-cmult*) (*auto simp add: space-subprob-algebra*)
with *mult show ?case*
by *simp*
next
case (*add f g*)
then have $(\lambda M'. \int^+ M''. f M'' + g M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. (\int^+ M''. f M'' \partial M') + (\int^+ M''. g M'' \partial M')) \in ?B$
by (*intro measurable-cong nn-integral-add*) (*auto simp add: space-subprob-algebra*)
with *add show ?case*
by (*simp add: ac-simps*)
next
case (*seq F*)
then have $(\lambda M'. \int^+ M''. (\text{SUP } i. F i) M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \text{SUP } i. (\int^+ M''. F i M'' \partial M')) \in ?B$
unfolding *SUP-apply*
by (*intro measurable-cong nn-integral-monotone-convergence-SUP*) (*auto simp add: space-subprob-algebra*)
with *seq show ?case*
by (*simp add: ac-simps*)
qed

lemma *measurable-distr*:

assumes [*measurable*]: $f \in \text{measurable } M N$
shows $(\lambda M'. \text{distr } M' N f) \in \text{measurable } (\text{subprob-algebra } M)$ (*subprob-algebra* N)
proof (*cases space N = {}*)

```

assume not-empty: space N ≠ {}
show ?thesis
proof (rule measurable-subprob-algebra)
  fix A assume A: A ∈ sets N
  then have  $(\lambda M'. \text{emeasure } (\text{distr } M' N f) A) \in \text{borel-measurable } (\text{subprob-algebra } M)$   $\longleftrightarrow$ 
     $(\lambda M'. \text{emeasure } M' (f - ' A \cap \text{space } M)) \in \text{borel-measurable } (\text{subprob-algebra } M)$ 
  by (intro measurable-cong)
    (auto simp: emeasure-distr space-subprob-algebra
      intro!: arg-cong2[where f=emeasure] sets-eq-imp-space-eq arg-cong2[where
f=op  $\cap$ ])
  also have ...
  using A by (intro measurable-emeasure-subprob-algebra simp)
  finally show  $(\lambda M'. \text{emeasure } (\text{distr } M' N f) A) \in \text{borel-measurable } (\text{subprob-algebra } M)$  .
  qed (auto intro!: subprob-space.subprob-space-distr simp: space-subprob-algebra
not-empty cong: measurable-cong-sets)
qed (insert assms, auto simp: measurable-empty-iff space-subprob-algebra-empty-iff)

```

```

lemma emeasure-space-subprob-algebra[measurable]:
   $(\lambda a. \text{emeasure } a (\text{space } a)) \in \text{borel-measurable } (\text{subprob-algebra } N)$ 
proof –
  have  $(\lambda a. \text{emeasure } a (\text{space } N)) \in \text{borel-measurable } (\text{subprob-algebra } N)$  (is ?f
  ∈ ?M)
  by (rule measurable-emeasure-subprob-algebra simp)
  also have ?f ∈ ?M  $\longleftrightarrow$   $(\lambda a. \text{emeasure } a (\text{space } a)) \in ?M$ 
  by (rule measurable-cong) (auto simp: space-subprob-algebra dest: sets-eq-imp-space-eq)
  finally show ?thesis .
qed

```

```

lemma integrable-measurable-subprob-algebra[measurable]:
  fixes f :: 'a ⇒ 'b::{banach, second-countable-topology}
  assumes [measurable]: f ∈ borel-measurable N
  shows Measurable.pred (subprob-algebra N)  $(\lambda M. \text{integrable } M f)$ 
proof (rule measurable-cong[THEN iffD2])
  show  $M \in \text{space } (\text{subprob-algebra } N) \implies \text{integrable } M f \longleftrightarrow (\int^{+x}. \text{norm } (f x) \partial M) < \infty$  for M
  by (auto simp: space-subprob-algebra integrable-iff-bounded)
qed measurable

```

```

lemma integral-measurable-subprob-algebra[measurable]:
  fixes f :: 'a ⇒ 'b::{banach, second-countable-topology}
  assumes f [measurable]: f ∈ borel-measurable N
  shows  $(\lambda M. \text{integral}^L M f) \in \text{subprob-algebra } N \rightarrow_M \text{borel}$ 
proof –
  from borel-measurable-implies-sequence-metric[OF f, of 0]
  obtain F where F:  $\bigwedge i. \text{simple-function } N (F i)$ 
   $\bigwedge x. x \in \text{space } N \implies (\lambda i. F i x) \longrightarrow f x$ 

```


$\bigwedge i x. x \in \text{space } N \implies \text{norm } (F i x) \leq 2 * \text{norm } (f x)$
unfolding *norm-conv-dist* **by** *blast*

have [*measurable*]: $F i \in N \rightarrow_M \text{count-space UNIV}$ **for** i
using $F(1)$ **by** (*rule measurable-simple-function*)

define F' **where** [*abs-def*]:
 $F' M i = (\text{if integrable } M f \text{ then } \text{integral}^L M (F i) \text{ else } 0)$ **for** $M i$

have $(\lambda M. F' M i) \in \text{subprob-algebra } N \rightarrow_M \text{borel}$ **for** i
proof (*rule measurable-cong[THEN iffD2]*)
fix M **assume** $M \in \text{space } (\text{subprob-algebra } N)$
then have [*simp*]: $\text{sets } M = \text{sets } N$ $\text{space } M = \text{space } N$ *subprob-space* M
by (*auto simp: space-subprob-algebra intro!: sets-eq-imp-space-eq*)
interpret *subprob-space* M **by fact**
have $F' M i = (\text{if integrable } M f \text{ then } \text{Bochner-Integration.simple-bochner-integral}$
 $M (F i) \text{ else } 0)$
using $F(1)$
by (*subst simple-bochner-integrable-eq-integral*)
(*auto simp: simple-bochner-integrable.simps simple-function-def F'-def*)
then show $F' M i = (\text{if integrable } M f \text{ then } \sum_{y \in F i} \text{space } N. \text{measure } M$
 $\{x \in \text{space } N. F i x = y\} *_{\mathbb{R}} y \text{ else } 0)$
unfolding *simple-bochner-integral-def* **by** *simp*
qed *measurable*
moreover
have $F' M \longrightarrow \text{integral}^L M f$ **if** $M: M \in \text{space } (\text{subprob-algebra } N)$ **for** M
proof *cases*
from M **have** [*simp*]: $\text{sets } M = \text{sets } N$ $\text{space } M = \text{space } N$
by (*auto simp: space-subprob-algebra intro!: sets-eq-imp-space-eq*)
assume *integrable* $M f$ **then show** *?thesis*
unfolding F' -*def* **using** $F(1)$ [*THEN borel-measurable-simple-function*] F
by (*auto intro!: integral-dominated-convergence[where w= $\lambda x. 2 * \text{norm } (f$*
 $x)]$
cong: measurable-cong-sets)
qed (*auto simp: F'-def not-integrable-integral-eq*)
ultimately show *?thesis*
by (*rule borel-measurable-LIMSEQ-metric*)
qed

lemma *measurable-pair-measure*:

assumes $f: f \in \text{measurable } M (\text{subprob-algebra } N)$
assumes $g: g \in \text{measurable } M (\text{subprob-algebra } L)$
shows $(\lambda x. f x \otimes_M g x) \in \text{measurable } M (\text{subprob-algebra } (N \otimes_M L))$
proof (*rule measurable-subprob-algebra*)
{ fix x **assume** $x \in \text{space } M$
with *measurable-space[OF f]* *measurable-space[OF g]*
have $fx: f x \in \text{space } (\text{subprob-algebra } N)$ **and** $gx: g x \in \text{space } (\text{subprob-algebra } L)$

```

    by auto
  interpret F: subprob-space f x
    using fx by (simp add: space-subprob-algebra)
  interpret G: subprob-space g x
    using gx by (simp add: space-subprob-algebra)

  interpret pair-subprob-space f x g x ..
  show subprob-space (f x  $\otimes_M$  g x) by unfold-locales
  show sets-eq: sets (f x  $\otimes_M$  g x) = sets (N  $\otimes_M$  L)
    using fx gx by (simp add: space-subprob-algebra)

  have 1:  $\bigwedge A B. A \in \text{sets } N \implies B \in \text{sets } L \implies \text{emeasure } (f x \otimes_M g x) (A \times B) = \text{emeasure } (f x) A * \text{emeasure } (g x) B$ 
    using fx gx by (intro G.emeasure-pair-measure-Times) (auto simp: space-subprob-algebra)
  have emeasure (f x  $\otimes_M$  g x) (space (f x  $\otimes_M$  g x)) =
    emeasure (f x) (space (f x)) * emeasure (g x) (space (g x))
  by (subst G.emeasure-pair-measure-Times[symmetric]) (simp-all add: space-pair-measure)
  hence 2:  $\bigwedge A. A \in \text{sets } (N \otimes_M L) \implies \text{emeasure } (f x \otimes_M g x) (\text{space } N \times \text{space } L - A) =$ 
    ... - emeasure (f x  $\otimes_M$  g x) A
    using emeasure-compl[simplified, OF - P.emeasure-finite]
  unfolding sets-eq
  unfolding sets-eq-imp-space-eq[OF sets-eq]
  by (simp add: space-pair-measure G.emeasure-pair-measure-Times)
  note 1 2 sets-eq }
  note Times = this(1) and Compl = this(2) and sets-eq = this(3)

  fix A assume A: A  $\in$  sets (N  $\otimes_M$  L)
  show ( $\lambda a. \text{emeasure } (f a \otimes_M g a) A$ )  $\in$  borel-measurable M
    using Int-stable-pair-measure-generator pair-measure-closed A
    unfolding sets-pair-measure
  proof (induct A rule: sigma-sets-induct-disjoint)
  case (basic A) then show ?case
    by (auto intro!: borel-measurable-times-ennreal simp: Times cong: measurable-cong)
      (auto intro!: measurable-emeasure-kernel f g)
  next
  case (compl A)
  then have A: A  $\in$  sets (N  $\otimes_M$  L)
    by (auto simp: sets-pair-measure)
  have ( $\lambda x. \text{emeasure } (f x) (\text{space } (f x)) * \text{emeasure } (g x) (\text{space } (g x)) -$ 
    emeasure (f x  $\otimes_M$  g x) A)  $\in$  borel-measurable M (is ?f  $\in$  ?M)
    using compl(2) f g by measurable
  thus ?case by (simp add: Compl A cong: measurable-cong)
  next
  case (union A)
  then have range A  $\subseteq$  sets (N  $\otimes_M$  L) disjoint-family A
    by (auto simp: sets-pair-measure)
  then have ( $\lambda a. \text{emeasure } (f a \otimes_M g a) (\bigcup i. A i)$ )  $\in$  borel-measurable M  $\longleftrightarrow$ 
    ( $\lambda a. \sum i. \text{emeasure } (f a \otimes_M g a) (A i)$ )  $\in$  borel-measurable M

```

```

    by (intro measurable-cong suminf-emeasure[symmetric])
      (auto simp: sets-eq)
  also have ...
    using union by auto
  finally show ?case .
qed simp
qed

lemma restrict-space-measurable:
  assumes X: X ≠ {} X ∈ sets K
  assumes N: N ∈ measurable M (subprob-algebra K)
  shows (λx. restrict-space (N x) X) ∈ measurable M (subprob-algebra (restrict-space
  K X))
proof (rule measurable-subprob-algebra)
  fix a assume a: a ∈ space M
  from N[THEN measurable-space, OF this]
  have subprob-space (N a) and [simp]: sets (N a) = sets K space (N a) = space
  K
  by (auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
  then interpret subprob-space N a
  by simp
  show subprob-space (restrict-space (N a) X)
proof
  show space (restrict-space (N a) X) ≠ {}
    using X by (auto simp add: space-restrict-space)
  show emeasure (restrict-space (N a) X) (space (restrict-space (N a) X)) ≤ 1
    using X by (simp add: emeasure-restrict-space space-restrict-space subprob-emeasure-le-1)
qed
  show sets (restrict-space (N a) X) = sets (restrict-space K X)
    by (intro sets-restrict-space-cong) fact
next
fix A assume A: A ∈ sets (restrict-space K X)
show (λa. emeasure (restrict-space (N a) X) A) ∈ borel-measurable M
proof (subst measurable-cong)
  fix a assume a ∈ space M
  from N[THEN measurable-space, OF this]
  have [simp]: sets (N a) = sets K space (N a) = space K
    by (auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
  show emeasure (restrict-space (N a) X) A = emeasure (N a) (A ∩ X)
    using X A by (subst emeasure-restrict-space) (auto simp add: sets-restrict-space
  ac-simps)
next
show (λw. emeasure (N w) (A ∩ X)) ∈ borel-measurable M
  using A X
  by (intro measurable-compose[OF N measurable-emeasure-subprob-algebra])
    (auto simp: sets-restrict-space)
qed
qed

```

8 Properties of return

definition $\text{return} :: 'a \text{ measure} \Rightarrow 'a \Rightarrow 'a \text{ measure}$ **where**
 $\text{return } R \ x = \text{measure-of } (\text{space } R) (\text{sets } R) (\lambda A. \text{indicator } A \ x)$

lemma $\text{space-return[simp]}$: $\text{space } (\text{return } M \ x) = \text{space } M$
by $(\text{simp add: return-def})$

lemma sets-return[simp] : $\text{sets } (\text{return } M \ x) = \text{sets } M$
by $(\text{simp add: return-def})$

lemma $\text{measurable-return1[simp]}$: $\text{measurable } (\text{return } N \ x) \ L = \text{measurable } N \ L$
by $(\text{simp cong: measurable-cong-sets})$

lemma $\text{measurable-return2[simp]}$: $\text{measurable } L \ (\text{return } N \ x) = \text{measurable } L \ N$
by $(\text{simp cong: measurable-cong-sets})$

lemma return-sets-cong : $\text{sets } M = \text{sets } N \Longrightarrow \text{return } M = \text{return } N$
by $(\text{auto dest: sets-eq-imp-space-eq simp: fun-eq-iff return-def})$

lemma return-cong : $\text{sets } A = \text{sets } B \Longrightarrow \text{return } A \ x = \text{return } B \ x$
by $(\text{auto simp add: return-def dest: sets-eq-imp-space-eq})$

lemma $\text{emeasure-return[simp]}$:

assumes $A \in \text{sets } M$

shows $\text{emeasure } (\text{return } M \ x) \ A = \text{indicator } A \ x$

proof $(\text{rule emeasure-measure-of[OF return-def]})$

show $\text{sets } M \subseteq \text{Pow } (\text{space } M)$ **by** $(\text{rule sets.space-closed})$

show $\text{positive } (\text{sets } (\text{return } M \ x)) (\lambda A. \text{indicator } A \ x)$ **by** $(\text{simp add: positive-def})$

from assms **show** $A \in \text{sets } (\text{return } M \ x)$ **unfolding** return-def **by** simp

show $\text{countably-additive } (\text{sets } (\text{return } M \ x)) (\lambda A. \text{indicator } A \ x)$

by $(\text{auto intro!: countably-additiveI suminf-indicator})$

qed

lemma prob-space-return : $x \in \text{space } M \Longrightarrow \text{prob-space } (\text{return } M \ x)$
by rule simp

lemma $\text{subprob-space-return}$: $x \in \text{space } M \Longrightarrow \text{subprob-space } (\text{return } M \ x)$
by $(\text{intro prob-space-return prob-space-imp-subprob-space})$

lemma $\text{subprob-space-return-ne}$:

assumes $\text{space } M \neq \{\}$ **shows** $\text{subprob-space } (\text{return } M \ x)$

proof

show $\text{emeasure } (\text{return } M \ x) (\text{space } (\text{return } M \ x)) \leq 1$

by $(\text{subst emeasure-return}) (\text{auto split: split-indicator})$

qed (simp, fact)

lemma measure-return : **assumes** $X: X \in \text{sets } M$ **shows** $\text{measure } (\text{return } M \ x)$
 $X = \text{indicator } X \ x$

unfolding *measure-def emeasure-return*[*OF X, of x*] **by** (*simp split: split-indicator*)

lemma *AE-return*:

assumes [*simp*]: $x \in \text{space } M$ **and** [*measurable*]: *Measurable.pred M P*

shows (*AE y in return M x. P y*) $\longleftrightarrow P x$

proof –

have (*AE y in return M x. y $\notin \{x \in \text{space } M. \neg P x\}$*) $\longleftrightarrow P x$

by (*subst AE-iff-null-sets[symmetric]*) (*simp-all add: null-sets-def split: split-indicator*)

also have (*AE y in return M x. y $\notin \{x \in \text{space } M. \neg P x\}$*) \longleftrightarrow (*AE y in return M x. P y*)

by (*rule AE-cong*) *auto*

finally show *?thesis* .

qed

lemma *nn-integral-return*:

assumes $x \in \text{space } M$ $g \in \text{borel-measurable } M$

shows ($\int^+ a. g a \partial \text{return } M x$) = $g x$

proof –

interpret *prob-space return M x* **by** (*rule prob-space-return*[*OF $\langle x \in \text{space } M \rangle$*])

have ($\int^+ a. g a \partial \text{return } M x$) = ($\int^+ a. g x \partial \text{return } M x$) **using** *assms*

by (*intro nn-integral-cong-AE*) (*auto simp: AE-return*)

also have ... = $g x$

using *nn-integral-const*[*of return M x*] *emeasure-space-1* **by** *simp*

finally show *?thesis* .

qed

lemma *integral-return*:

fixes $g :: - \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$

assumes $x \in \text{space } M$ $g \in \text{borel-measurable } M$

shows ($\int a. g a \partial \text{return } M x$) = $g x$

proof –

interpret *prob-space return M x* **by** (*rule prob-space-return*[*OF $\langle x \in \text{space } M \rangle$*])

have ($\int a. g a \partial \text{return } M x$) = ($\int a. g x \partial \text{return } M x$) **using** *assms*

by (*intro integral-cong-AE*) (*auto simp: AE-return*)

then show *?thesis*

using *prob-space* **by** *simp*

qed

lemma *return-measurable*[*measurable*]: *return N \in measurable N* (*subprob-algebra N*)

by (*rule measurable-subprob-algebra*) (*auto simp: subprob-space-return*)

lemma *distr-return*:

assumes $f \in \text{measurable } M N$ **and** $x \in \text{space } M$

shows *distr (return M x) N f* = *return N (f x)*

using *assms* **by** (*intro measure-eqI*) (*simp-all add: indicator-def emeasure-distr*)

lemma *return-restrict-space*:

$\Omega \in \text{sets } M \implies \text{return (restrict-space } M \Omega) x = \text{restrict-space (return } M x) \Omega$

by (auto intro!: measure-eqI simp: sets-restrict-space emeasure-restrict-space)

lemma measurable-distr2:

assumes $f[\text{measurable}]$: case-prod $f \in \text{measurable } L \otimes_M M \ N$

assumes $g[\text{measurable}]$: $g \in \text{measurable } L \ (\text{subprob-algebra } M)$

shows $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{measurable } L \ (\text{subprob-algebra } N)$

proof –

have $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{measurable } L \ (\text{subprob-algebra } N)$

$\longleftrightarrow (\lambda x. \text{distr } (\text{return } L \ x \ \otimes_M \ g \ x) \ N \ (\text{case-prod } f)) \in \text{measurable } L \ (\text{subprob-algebra } N)$

proof (rule measurable-cong)

fix x **assume** x : $x \in \text{space } L$

have $g \ x \in \text{space } (\text{subprob-algebra } M)$

using measurable-space[OF $g \ x$].

then have [simp]: $\text{sets } (g \ x) = \text{sets } M$

by (simp add: space-subprob-algebra)

then have [simp]: $\text{space } (g \ x) = \text{space } M$

by (rule sets-eq-imp-space-eq)

let $?R = \text{return } L \ x$

from measurable-compose-Pair1[OF $x \ f$] **have** $f \cdot M'$: $f \ x \in \text{measurable } M \ N$

by simp

interpret subprob-space $g \ x$

using $g \ x$ **by** (simp add: space-subprob-algebra)

have space-pair- M' [simp]: $\bigwedge X. \text{space } (X \ \otimes_M \ g \ x) = \text{space } (X \ \otimes_M \ M)$

by (simp add: space-pair-measure)

show $\text{distr } (g \ x) \ N \ (f \ x) = \text{distr } (?R \ \otimes_M \ g \ x) \ N \ (\text{case-prod } f)$ (**is** $?l = ?r$)

proof (rule measure-eqI)

show $\text{sets } ?l = \text{sets } ?r$

by simp

next

fix A **assume** $A \in \text{sets } ?l$

then have $A[\text{measurable}]$: $A \in \text{sets } N$

by simp

then have $\text{emeasure } ?r \ A = \text{emeasure } (?R \ \otimes_M \ g \ x) \ ((\lambda(x, y). f \ x \ y) - ' A \cap \text{space } (?R \ \otimes_M \ g \ x))$

by (auto simp add: emeasure-distr $f \cdot M'$ cong: measurable-cong-sets)

also have $\dots = (\int^+ M''. \text{emeasure } (g \ x) \ (f \ M'' - ' A \cap \text{space } M) \ \partial ?R)$

apply (subst emeasure-pair-measure-alt)

apply (rule measurable-sets[OF A])

apply (auto simp add: $f \cdot M'$ cong: measurable-cong-sets)

apply (intro nn-integral-cong arg-cong[**where** $f = \text{emeasure } (g \ x)$])

apply (auto simp: space-subprob-algebra space-pair-measure)

done

also have $\dots = \text{emeasure } (g \ x) \ (f \ x - ' A \cap \text{space } M)$

by (subst nn-integral-return)

(auto simp: x intro!: measurable-emeasure)

also have $\dots = \text{emeasure } ?l \ A$

by (simp add: emeasure-distr $f \cdot M'$ cong: measurable-cong-sets)

finally show $\text{emeasure } ?l \ A = \text{emeasure } ?r \ A \ ..$

```

  qed
  qed
  also have ...
  apply (intro measurable-compose[OF measurable-pair-measure measurable-distr])
  apply (rule return-measurable)
  apply measurable
  done
  finally show ?thesis .
  qed

```

lemma *nn-integral-measurable-subprob-algebra2*:

```

  assumes f[measurable]: (λ(x, y). f x y) ∈ borel-measurable (M ⊗M N)
  assumes N[measurable]: L ∈ measurable M (subprob-algebra N)
  shows (λx. integralN (L x) (f x)) ∈ borel-measurable M
  proof -
    note nn-integral-measurable-subprob-algebra[measurable]
    note measurable-distr2[measurable]
    have (λx. integralN (distr (L x) (M ⊗M N) (λy. (x, y))) (λ(x, y). f x y)) ∈
      borel-measurable M
      by measurable
    then show (λx. integralN (L x) (f x)) ∈ borel-measurable M
      by (rule measurable-cong[THEN iffD1, rotated])
         (simp add: nn-integral-distr)
  qed

```

lemma *emeasure-measurable-subprob-algebra2*:

```

  assumes A[measurable]: (SIGMA x:space M. A x) ∈ sets (M ⊗M N)
  assumes L[measurable]: L ∈ measurable M (subprob-algebra N)
  shows (λx. emeasure (L x) (A x)) ∈ borel-measurable M
  proof -
    { fix x assume x ∈ space M
      then have Pair x -‘ Sigma (space M) A = A x
        by auto
      with sets-Pair1[OF A, of x] have A x ∈ sets N
        by auto }
    note ** = this

    have *: ∧x. fst x ∈ space M ⇒ snd x ∈ A (fst x) ↔ x ∈ (SIGMA x:space
      M. A x)
      by (auto simp: fun-eq-iff)
    have (λ(x, y). indicator (A x) y::ennreal) ∈ borel-measurable (M ⊗M N)
      apply measurable
      apply (subst measurable-cong)
      apply (rule *)
      apply (auto simp: space-pair-measure)
      done
    then have (λx. integralN (L x) (indicator (A x))) ∈ borel-measurable M
      by (intro nn-integral-measurable-subprob-algebra2[where N=N] L)
    then show (λx. emeasure (L x) (A x)) ∈ borel-measurable M

```

```

apply (rule measurable-cong[THEN iffD1, rotated])
apply (rule nn-integral-indicator)
apply (simp add: subprob-measurableD[OF L] **)
done
qed

```

```

lemma measure-measurable-subprob-algebra2:
assumes A[measurable]: (SIGMA x:space M. A x) ∈ sets (M ⊗M N)
assumes L[measurable]: L ∈ measurable M (subprob-algebra N)
shows (λx. measure (L x) (A x)) ∈ borel-measurable M
unfolding measure-def
by (intro borel-measurable-enn2real emeasure-measurable-subprob-algebra2[OF assms])

```

```

definition select-sets M = (SOME N. sets M = sets (subprob-algebra N))

```

```

lemma select-sets1:
  sets M = sets (subprob-algebra N) ⇒ sets M = sets (subprob-algebra (select-sets
M))
unfolding select-sets-def by (rule someI)

```

```

lemma sets-select-sets[simp]:
assumes sets: sets M = sets (subprob-algebra N)
shows sets (select-sets M) = sets N
unfolding select-sets-def
proof (rule someI2)
  show sets M = sets (subprob-algebra N)
    by fact
next
  fix L assume sets M = sets (subprob-algebra L)
  with sets have eq: space (subprob-algebra N) = space (subprob-algebra L)
    by (intro sets-eq-imp-space-eq) simp
  show sets L = sets N
  proof cases
    assume space (subprob-algebra N) = {}
    with space-subprob-algebra-empty-iff[of N] space-subprob-algebra-empty-iff[of
L]
    show ?thesis
      by (simp add: eq space-empty-iff)
    next
    assume space (subprob-algebra N) ≠ {}
    with eq show ?thesis
      by (fastforce simp add: space-subprob-algebra)
  qed
qed

```

```

lemma space-select-sets[simp]:
  sets M = sets (subprob-algebra N) ⇒ space (select-sets M) = space N
by (intro sets-eq-imp-space-eq sets-select-sets)

```


9 Join

definition $join :: 'a\ measure\ measure \Rightarrow 'a\ measure$ **where**

$join\ M = measure\ of\ (space\ (select\ sets\ M))\ (sets\ (select\ sets\ M))\ (\lambda B. \int^+ M' . emeasure\ M' B\ \partial M)$

lemma

shows $space\ join[simp]: space\ (join\ M) = space\ (select\ sets\ M)$

and $sets\ join[simp]: sets\ (join\ M) = sets\ (select\ sets\ M)$

by $(simp\ all\ add: join\ def)$

lemma $emeasure\ join:$

assumes $M[simp, measurable\ cong]: sets\ M = sets\ (subprob\ algebra\ N)$ **and** $A \in sets\ N$

shows $emeasure\ (join\ M)\ A = (\int^+ M' . emeasure\ M' A\ \partial M)$

proof $(rule\ emeasure\ measure\ of[OF\ join\ def])$

show $countably\ additive\ (sets\ (join\ M))\ (\lambda B. \int^+ M' . emeasure\ M' B\ \partial M)$

proof $(rule\ countably\ additiveI)$

fix $A :: nat \Rightarrow 'a\ set$ **assume** $A: range\ A \subseteq sets\ (join\ M)$ *disjoint-family* A

have $(\sum i. \int^+ M' . emeasure\ M' (A\ i)\ \partial M) = (\int^+ M' . (\sum i. emeasure\ M' (A\ i))\ \partial M)$

using A **by** $(subst\ nn\ integral\ suminf)\ (auto\ simp: measurable\ emeasure\ subprob\ algebra)$

also **have** $\dots = (\int^+ M' . emeasure\ M' (\bigcup i. A\ i)\ \partial M)$

proof $(rule\ nn\ integral\ cong)$

fix M' **assume** $M' \in space\ M$

then **show** $(\sum i. emeasure\ M' (A\ i)) = emeasure\ M' (\bigcup i. A\ i)$

using A $sets\ eq\ imp\ space\ eq[OF\ M]$ **by** $(simp\ add: suminf\ emeasure\ space\ subprob\ algebra)$

qed

finally **show** $(\sum i. \int^+ M' . emeasure\ M' (A\ i)\ \partial M) = (\int^+ M' . emeasure\ M' (\bigcup i. A\ i)\ \partial M)$.

qed

qed $(auto\ simp: A\ sets.\ space\ closed\ positive\ def)$

lemma $measurable\ join:$

$join \in measurable\ (subprob\ algebra\ (subprob\ algebra\ N))\ (subprob\ algebra\ N)$

proof $(cases\ space\ N \neq \{\}, rule\ measurable\ subprob\ algebra)$

fix A **assume** $A \in sets\ N$

let $?B = borel\ measurable\ (subprob\ algebra\ (subprob\ algebra\ N))$

have $(\lambda M' . emeasure\ (join\ M')\ A) \in ?B \iff (\lambda M' . (\int^+ M'' . emeasure\ M'' A\ \partial M')) \in ?B$

proof $(rule\ measurable\ cong)$

fix M' **assume** $M' \in space\ (subprob\ algebra\ (subprob\ algebra\ N))$

then **show** $emeasure\ (join\ M')\ A = (\int^+ M'' . emeasure\ M'' A\ \partial M')$

by $(intro\ emeasure\ join)\ (auto\ simp: space\ subprob\ algebra\ \langle A \in sets\ N \rangle)$

qed

also **have** $(\lambda M' . \int^+ M'' . emeasure\ M'' A\ \partial M') \in ?B$

using $measurable\ emeasure\ subprob\ algebra[OF\ \langle A \in sets\ N \rangle]$

by $(rule\ nn\ integral\ measurable\ subprob\ algebra)$

finally show $(\lambda M'. \text{emeasure } (\text{join } M') A) \in \text{borel-measurable } (\text{subprob-algebra } (\text{subprob-algebra } N))$.

next

assume $[\text{simp}]$: $\text{space } N \neq \{\}$

fix M **assume** M : $M \in \text{space } (\text{subprob-algebra } (\text{subprob-algebra } N))$

then have $(\int^+ M'. \text{emeasure } M' (\text{space } N) \partial M) \leq (\int^+ M'. 1 \partial M)$

apply $(\text{intro nn-integral-mono})$

apply $(\text{auto simp: space-subprob-algebra})$

$\text{dest!}: \text{sets-eq-imp-space-eq subprob-space.emeasure-space-le-1}$

done

with M **show** $\text{subprob-space } (\text{join } M)$

by $(\text{intro subprob-spaceI})$

$(\text{auto simp: emeasure-join space-subprob-algebra } M \text{ dest: subprob-space.emeasure-space-le-1})$

next

assume $\neg(\text{space } N \neq \{\})$

thus $?thesis$ **by** $(\text{simp add: measurable-empty-iff space-subprob-algebra-empty-iff})$

qed $(\text{auto simp: space-subprob-algebra})$

lemma nn-integral-join :

assumes f : $f \in \text{borel-measurable } N$

and $M[\text{measurable-cong}]$: $\text{sets } M = \text{sets } (\text{subprob-algebra } N)$

shows $(\int^+ x. f x \partial \text{join } M) = (\int^+ M'. \int^+ x. f x \partial M' \partial M)$

using f

proof induct

case $(\text{cong } f g)$

moreover have $\text{integral}^N (\text{join } M) f = \text{integral}^N (\text{join } M) g$

by $(\text{intro nn-integral-cong cong}) (\text{simp add: } M)$

moreover from M **have** $(\int^+ M'. \text{integral}^N M' f \partial M) = (\int^+ M'. \text{integral}^N M' g \partial M)$

by $(\text{intro nn-integral-cong cong})$

$(\text{auto simp add: space-subprob-algebra dest!}: \text{sets-eq-imp-space-eq})$

ultimately show $?case$

by simp

next

case $(\text{set } A)$

with M **have** $(\int^+ M'. \text{integral}^N M' (\text{indicator } A) \partial M) = (\int^+ M'. \text{emeasure } M' A \partial M)$

by $(\text{intro nn-integral-cong nn-integral-indicator})$

$(\text{auto simp: space-subprob-algebra dest!}: \text{sets-eq-imp-space-eq})$

with set **show** $?case$

using M **by** $(\text{simp add: emeasure-join})$

next

case $(\text{mult } f c)$

have $(\int^+ M'. \int^+ x. c * f x \partial M' \partial M) = (\int^+ M'. c * \int^+ x. f x \partial M' \partial M)$

using $\text{mult } M M[\text{THEN sets-eq-imp-space-eq}]$

by $(\text{intro nn-integral-cong nn-integral-cmult}) (\text{auto simp add: space-subprob-algebra})$

also have $\dots = c * (\int^+ M'. \int^+ x. f x \partial M' \partial M)$

using $\text{nn-integral-measurable-subprob-algebra}[\text{OF mult}(2)]$

by $(\text{intro nn-integral-cmult mult}) (\text{simp add: } M)$

```

also have ... = c * (integralN (join M) f)
  by (simp add: mult)
also have ... = (∫+ x. c * f x ∂join M)
  using mult(2,3) by (intro nn-integral-cmult[symmetric] mult) (simp add: M
cong: measurable-cong-sets)
  finally show ?case by simp
next
  case (add f g)
  have (∫+ M'. ∫+ x. f x + g x ∂M' ∂M) = (∫+ M'. (∫+ x. f x ∂M') + (∫+
x. g x ∂M') ∂M)
  using add M M[THEN sets-eq-imp-space-eq]
  by (intro nn-integral-cong nn-integral-add) (auto simp add: space-subprob-algebra)
also have ... = (∫+ M'. ∫+ x. f x ∂M' ∂M) + (∫+ M'. ∫+ x. g x ∂M' ∂M)
  using nn-integral-measurable-subprob-algebra[OF add(1)]
  using nn-integral-measurable-subprob-algebra[OF add(4)]
  by (intro nn-integral-add add) (simp-all add: M)
also have ... = (integralN (join M) f) + (integralN (join M) g)
  by (simp add: add)
also have ... = (∫+ x. f x + g x ∂join M)
  using add by (intro nn-integral-add[symmetric] add) (simp-all add: M cong:
measurable-cong-sets)
  finally show ?case by (simp add: ac-simps)
next
  case (seq F)
  have (∫+ M'. ∫+ x. (SUP i. F i) x ∂M' ∂M) = (∫+ M'. (SUP i. ∫+ x. F i
x ∂M') ∂M)
  using seq M M[THEN sets-eq-imp-space-eq] unfolding SUP-apply
  by (intro nn-integral-cong nn-integral-monotone-convergence-SUP)
(auto simp add: space-subprob-algebra)
also have ... = (SUP i. ∫+ M'. ∫+ x. F i x ∂M' ∂M)
  using nn-integral-measurable-subprob-algebra[OF seq(1)] seq
  by (intro nn-integral-monotone-convergence-SUP)
(simp-all add: M incseq-nn-integral incseq-def le-fun-def nn-integral-mono )
also have ... = (SUP i. integralN (join M) (F i))
  by (simp add: seq)
also have ... = (∫+ x. (SUP i. F i x) ∂join M)
  using seq by (intro nn-integral-monotone-convergence-SUP[symmetric] seq)
(simp-all add: M cong: measurable-cong-sets)
  finally show ?case by (simp add: ac-simps)
qed

```

lemma measurable-join1:

$\llbracket f \in \text{measurable } N \ K; \text{ sets } M = \text{sets } (\text{subprob-algebra } N) \rrbracket$

$\implies f \in \text{measurable } (\text{join } M) \ K$

by(simp add: measurable-def)

lemma

fixes f :: - \implies real

assumes f-measurable [measurable]: f \in borel-measurable N

```

and f-bounded:  $\bigwedge x. x \in \text{space } N \implies |f x| \leq B$ 
and M [measurable-cong]: sets M = sets (subprob-algebra N)
and fin: finite-measure M
and M-bounded: AE M' in M. emeasure M' (space M')  $\leq$  ennreal B'
shows integrable-join: integrable (join M) f (is ?integrable)
and integral-join: integralL (join M) f =  $\int$  M'. integralL M' f  $\partial$ M (is ?integral)
proof(case-tac [!] space N = {})
  assume *: space N = {}
  show ?integrable
    using M * by(simp add: real-integrable-def measurable-def nn-integral-empty)
  have ( $\int$  M'. integralL M' f  $\partial$ M) = ( $\int$  M'. 0  $\partial$ M)
  proof(rule Bochner-Integration.integral-cong)
    fix M'
    assume M'  $\in$  space M
    with sets-eq-imp-space-eq[OF M] have space M' = space N
      by(auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
    with * show ( $\int$  x. f x  $\partial$ M') = 0 by(simp add: Bochner-Integration.integral-empty)
  qed simp
  then show ?integral
    using M * by(simp add: Bochner-Integration.integral-empty)
next
  assume *: space N  $\neq$  {}

  from * have B [simp]: 0  $\leq$  B by(auto dest: f-bounded)

  have [measurable]: f  $\in$  borel-measurable (join M) using f-measurable M
    by(rule measurable-join1)

  { fix f M'
    assume [measurable]: f  $\in$  borel-measurable N
      and f-bounded:  $\bigwedge x. x \in \text{space } N \implies f x \leq B$ 
      and M'  $\in$  space M emeasure M' (space M')  $\leq$  ennreal B'
      have AE x in M'. ennreal (f x)  $\leq$  ennreal B
      proof(rule AE-I2)
        fix x
        assume x  $\in$  space M'
        with  $\langle$ M'  $\in$  space M $\rangle$  sets-eq-imp-space-eq[OF M]
        have x  $\in$  space N by(auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
        from f-bounded[OF this] show ennreal (f x)  $\leq$  ennreal B by simp
      qed
      then have ( $\int$ + x. ennreal (f x)  $\partial$ M')  $\leq$  ( $\int$ + x. ennreal B  $\partial$ M')
        by(rule nn-integral-mono-AE)
      also have ... = ennreal B * emeasure M' (space M') by(simp)
      also have ...  $\leq$  ennreal B * ennreal B' by(rule mult-left-mono)(fact, simp)
      also have ...  $\leq$  ennreal B * ennreal  $|B'|$  by(rule mult-left-mono)(simp-all)
      finally have ( $\int$ + x. ennreal (f x)  $\partial$ M')  $\leq$  ennreal (B *  $|B'|$ ) by (simp add:
ennreal-mult) }
    note bounded1 = this

```

have *bounded*:
 $\bigwedge f. \llbracket f \in \text{borel-measurable } N; \bigwedge x. x \in \text{space } N \implies f x \leq B \rrbracket$
 $\implies (\int^+ x. \text{ennreal } (f x) \partial \text{join } M) \neq \text{top}$
proof –
fix *f*
assume [*measurable*]: $f \in \text{borel-measurable } N$
and *f-bounded*: $\bigwedge x. x \in \text{space } N \implies f x \leq B$
have $(\int^+ x. \text{ennreal } (f x) \partial \text{join } M) = (\int^+ M'. \int^+ x. \text{ennreal } (f x) \partial M' \partial M)$
by (*rule nn-integral-join*[*OF* - *M*]) *simp*
also have $\dots \leq \int^+ M'. B * |B'| \partial M$
using *bounded1*[*OF* $\langle f \in \text{borel-measurable } N \rangle$ *f-bounded*]
by (*rule nn-integral-mono-AE*[*OF* *AE-mp*[*OF* *M-bounded* *AE-I2*], *rule-format*])
also have $\dots = B * |B'| * \text{emeasure } M (\text{space } M)$ **by** *simp*
also have $\dots < \infty$
using *finite-measure.finite-emeasure-space*[*OF* *fin*]
by (*simp add: ennreal-mult-less-top less-top*)
finally show *?thesis f* **by** *simp*
qed
have *f-pos*: $(\int^+ x. \text{ennreal } (f x) \partial \text{join } M) \neq \infty$
and *f-neg*: $(\int^+ x. \text{ennreal } (- f x) \partial \text{join } M) \neq \infty$
using *f-bounded* **by** (*auto del: notI intro!: bounded simp add: abs-le-iff*)

show *?integrable* **using** *f-pos f-neg* **by** (*simp add: real-integrable-def*)

note [*measurable*] = *nn-integral-measurable-subprob-algebra*

have *int-f*: $(\int^+ x. f x \partial \text{join } M) = \int^+ M'. \int^+ x. f x \partial M' \partial M$
by (*simp add: nn-integral-join*[*OF* - *M*])
have *int-mf*: $(\int^+ x. - f x \partial \text{join } M) = (\int^+ M'. \int^+ x. - f x \partial M' \partial M)$
by (*simp add: nn-integral-join*[*OF* - *M*])

have *pos-finite*: *AE* *M'* *in* *M*. $(\int^+ x. f x \partial M') \neq \infty$
using *AE-space* *M-bounded*
proof *eventually-elim*
fix *M'* **assume** $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
then have $(\int^+ x. \text{ennreal } (f x) \partial M') \leq \text{ennreal } (B * |B'|)$
using *f-measurable* **by** (*auto intro!: bounded1 dest: f-bounded*)
then show $(\int^+ x. \text{ennreal } (f x) \partial M') \neq \infty$
by (*auto simp: top-unique*)
qed
hence [*simp*]: $(\int^+ M'. \text{ennreal } (\text{enn2real } (\int^+ x. f x \partial M')) \partial M) = (\int^+ M'. \int^+ x. f x \partial M' \partial M)$
by (*rule nn-integral-cong-AE*[*OF* *AE-mp*]) (*simp add: less-top*)
from *f-pos* **have** [*simp*]: *integrable* *M* $(\lambda M'. \text{enn2real } (\int^+ x. f x \partial M'))$
by (*simp add: int-f real-integrable-def nn-integral-0-iff-AE*[*THEN iffD2*] *ennreal-neg* *enn2real-nonneg*)

have *neg-finite*: *AE* *M'* *in* *M*. $(\int^+ x. - f x \partial M') \neq \infty$
using *AE-space* *M-bounded*

proof *eventually-elim*
fix M' **assume** $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
then have $(\int^+ x. \text{ennreal } (-f x) \partial M') \leq \text{ennreal } (B * |B'|)$
using f -measurable **by**(*auto intro!*: *bounded1 dest: f-bounded*)
then show $(\int^+ x. \text{ennreal } (-f x) \partial M') \neq \infty$
by (*auto simp: top-unique*)
qed
hence [*simp*]: $(\int^+ M'. \text{ennreal } (\text{enn2real } (\int^+ x. -f x \partial M')) \partial M) = (\int^+ M'. \int^+ x. -f x \partial M' \partial M)$
by (*rule nn-integral-cong-AE[OF AE-mp]*) (*simp add: less-top*)
from f -neg **have** [*simp*]: $\text{integrable } M (\lambda M'. \text{enn2real } (\int^+ x. -f x \partial M'))$
by(*simp add: int-mf real-integrable-def nn-integral-0-iff-AE[THEN iffD2]*) *ennreal-neg enn2real-nonneg*

have $(\int x. f x \partial \text{join } M) = \text{enn2real } (\int^+ N. \int^+ x. f x \partial N \partial M) - \text{enn2real } (\int^+ N. \int^+ x. -f x \partial N \partial M)$
unfolding *real-lebesgue-integral-def*[*OF* $\langle ?\text{integrable} \rangle$] **by** (*simp add: nn-integral-join*[*OF* $-M$])
also have $\dots = (\int N. \text{enn2real } (\int^+ x. f x \partial N) \partial M) - (\int N. \text{enn2real } (\int^+ x. -f x \partial N) \partial M)$
using *pos-finite neg-finite* **by** (*subst* (1 2) *integral-eq-nn-integral*) (*auto simp: enn2real-nonneg*)
also have $\dots = (\int N. \text{enn2real } (\int^+ x. f x \partial N) - \text{enn2real } (\int^+ x. -f x \partial N)) \partial M$
by *simp*
also have $\dots = \int M'. \int x. f x \partial M' \partial M$
proof (*rule integral-cong-AE*)
show $AE x$ in M .
 $\text{enn2real } (\int^+ x. \text{ennreal } (f x) \partial x) - \text{enn2real } (\int^+ x. \text{ennreal } (-f x) \partial x)$
 $= \text{integral}^L x f$
using *AE-space M-bounded*
proof *eventually-elim*
fix M' **assume** $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq B'$
then interpret *subprob-space* M'
by (*auto simp: M[THEN sets-eq-imp-space-eq]*) *space-subprob-algebra*

from $\langle M' \in \text{space } M \rangle$ *sets-eq-imp-space-eq*[*OF* M]
have [*measurable-cong*]: $\text{sets } M' = \text{sets } N$ **by**(*simp add: space-subprob-algebra*)
hence [*simp*]: $\text{space } M' = \text{space } N$ **by**(*rule sets-eq-imp-space-eq*)
have *integrable* $M' f$
by(*rule integrable-const-bound*[**where** $B=B$])(*auto simp add: f-bounded*)
then show $\text{enn2real } (\int^+ x. f x \partial M') - \text{enn2real } (\int^+ x. -f x \partial M') = \int x. f x \partial M'$
by(*simp add: real-lebesgue-integral-def*)
qed
qed *simp-all*
finally show $?integral$ **by** *simp*
qed

lemma *join-assoc*:

assumes $M[\text{measurable-cong}]$: *sets* $M = \text{sets} (\text{subprob-algebra} (\text{subprob-algebra } N))$
shows $\text{join} (\text{distr } M (\text{subprob-algebra } N) \text{ join}) = \text{join} (\text{join } M)$
proof (*rule measure-eqI*)
fix A **assume** $A \in \text{sets} (\text{join} (\text{distr } M (\text{subprob-algebra } N) \text{ join}))$
then have $A: A \in \text{sets } N$ **by** *simp*
show $\text{emeasure} (\text{join} (\text{distr } M (\text{subprob-algebra } N) \text{ join})) A = \text{emeasure} (\text{join} (\text{join } M)) A$
using *measurable-join[of N]*
by (*auto simp: M A nn-integral-distr emeasure-join measurable-emeasure-subprob-algebra sets-eq-imp-space-eq[OF M] space-subprob-algebra nn-integral-join[OF - M]*
intro!: nn-integral-cong emeasure-join)
qed (*simp add: M*)

lemma *join-return*:

assumes *sets* $M = \text{sets } N$ **and** *subprob-space* M
shows $\text{join} (\text{return} (\text{subprob-algebra } N) M) = M$
by (*rule measure-eqI*)
(simp-all add: emeasure-join space-subprob-algebra measurable-emeasure-subprob-algebra nn-integral-return assms)

lemma *join-return'*:

assumes *sets* $N = \text{sets } M$
shows $\text{join} (\text{distr } M (\text{subprob-algebra } N) (\text{return } N)) = M$
apply (*rule measure-eqI*)
apply (*simp add: assms*)
apply (*subgoal-tac return N ∈ measurable M (subprob-algebra N)*)
apply (*simp add: emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra assms*)
apply (*subst measurable-cong-sets, rule assms[symmetric], rule refl, rule return-measurable*)
done

lemma *join-distr-distr*:

fixes $f :: 'a \Rightarrow 'b$ **and** $M :: 'a \text{ measure measure}$ **and** $N :: 'b \text{ measure}$
assumes *sets* $M = \text{sets} (\text{subprob-algebra } R)$ **and** $f \in \text{measurable } R N$
shows $\text{join} (\text{distr } M (\text{subprob-algebra } N) (\lambda M. \text{distr } M N f)) = \text{distr} (\text{join } M) N f$ (*is ?r = ?l*)
proof (*rule measure-eqI*)
fix A **assume** $A \in \text{sets } ?r$
hence $A\text{-in-}N: A \in \text{sets } N$ **by** *simp*

from *assms* **have** $f \in \text{measurable} (\text{join } M) N$
by (*simp cong: measurable-cong-sets*)
moreover from *assms* **and** $A\text{-in-}N$ **have** $f - 'A \cap \text{space } R \in \text{sets } R$
by (*intro measurable-sets*) *simp-all*
ultimately have $\text{emeasure} (\text{distr} (\text{join } M) N f) A = \int^+ M'. \text{emeasure } M' (f - 'A \cap \text{space } R) \partial M$

by (*simp-all add: A-in-N emeasure-distr emeasure-join assms*)

also have $\dots = \int^+ x. \text{emeasure } (\text{distr } x \ N \ f) \ A \ \partial M$ **using** *A-in-N*

proof (*intro nn-integral-cong, subst emeasure-distr*)

fix M' **assume** $M' \in \text{space } M$

from *assms* **have** $\text{space } M = \text{space } (\text{subprob-algebra } R)$

using *sets-eq-imp-space-eq* **by** *blast*

with $\langle M' \in \text{space } M \rangle$ **have** [*simp*]: $\text{sets } M' = \text{sets } R$ **using** *space-subprob-algebra*

by *blast*

show $f \in \text{measurable } M' \ N$ **by** (*simp cong: measurable-cong-sets add: assms*)

have $\text{space } M' = \text{space } R$ **by** (*rule sets-eq-imp-space-eq*) *simp*

thus $\text{emeasure } M' (f \text{ -- } A \cap \text{space } R) = \text{emeasure } M' (f \text{ -- } A \cap \text{space } M')$

by *simp*

qed

also have $(\lambda M. \text{distr } M \ N \ f) \in \text{measurable } M \ (\text{subprob-algebra } N)$

by (*simp cong: measurable-cong-sets add: assms measurable-distr*)

hence $(\int^+ x. \text{emeasure } (\text{distr } x \ N \ f) \ A \ \partial M) =$
 $\text{emeasure } (\text{join } (\text{distr } M \ (\text{subprob-algebra } N) \ (\lambda M. \text{distr } M \ N \ f))) \ A$

by (*simp-all add: emeasure-join assms A-in-N nn-integral-distr measurable-emeasure-subprob-algebra*)

finally show $\text{emeasure } ?r \ A = \text{emeasure } ?l \ A \ ..$

qed *simp*

definition *bind* :: $'a \ \text{measure} \Rightarrow ('a \Rightarrow 'b \ \text{measure}) \Rightarrow 'b \ \text{measure}$ **where**

$\text{bind } M \ f = (\text{if } \text{space } M = \{\} \ \text{then } \text{count-space } \{\} \ \text{else}$
 $\text{join } (\text{distr } M \ (\text{subprob-algebra } (f \ (\text{SOME } x. x \in \text{space } M))) \ f))$

adhoc-overloading *Monad-Syntax.bind* *bind*

lemma *bind-empty*:

$\text{space } M = \{\} \Longrightarrow \text{bind } M \ f = \text{count-space } \{\}$

by (*simp add: bind-def*)

lemma *bind-nonempty*:

$\text{space } M \neq \{\} \Longrightarrow \text{bind } M \ f = \text{join } (\text{distr } M \ (\text{subprob-algebra } (f \ (\text{SOME } x. x \in \text{space } M))) \ f)$

by (*simp add: bind-def*)

lemma *sets-bind-empty*: $\text{sets } M = \{\} \Longrightarrow \text{sets } (\text{bind } M \ f) = \{\{\}\}$

by (*auto simp: bind-def*)

lemma *space-bind-empty*: $\text{space } M = \{\} \Longrightarrow \text{space } (\text{bind } M \ f) = \{\}$

by (*simp add: bind-def*)

lemma *sets-bind[simp, measurable-cong]*:

assumes $f: \bigwedge x. x \in \text{space } M \Longrightarrow \text{sets } (f \ x) = \text{sets } N$ **and** $M: \text{space } M \neq \{\}$

shows $\text{sets } (\text{bind } M \ f) = \text{sets } N$

using f [*of SOME x. x \in \text{space } M*] **by** (*simp add: bind-nonempty M some-in-eq*)

lemma *space-bind[simp]*:

assumes $\bigwedge x. x \in \text{space } M \implies \text{sets } (f x) = \text{sets } N$ **and** $\text{space } M \neq \{\}$
shows $\text{space } (\text{bind } M f) = \text{space } N$
using *assms* **by** (*intro sets-eq-imp-space-eq sets-bind*)

lemma *bind-cong-All*:

assumes $\forall x \in \text{space } M. f x = g x$
shows $\text{bind } M f = \text{bind } M g$
proof (*cases space M = {}*)
assume $\text{space } M \neq \{\}$
hence $(\text{SOME } x. x \in \text{space } M) \in \text{space } M$ **by** (*rule-tac someI-ex*) *blast*
with *assms* **have** $f (\text{SOME } x. x \in \text{space } M) = g (\text{SOME } x. x \in \text{space } M)$ **by**
blast
with $\langle \text{space } M \neq \{\} \rangle$ **and** *assms* **show** *?thesis* **by** (*simp add: bind-nonempty*
cong: distr-cong)
qed (*simp add: bind-empty*)

lemma *bind-cong*:

$M = N \implies (\bigwedge x. x \in \text{space } M \implies f x = g x) \implies \text{bind } M f = \text{bind } N g$
using *bind-cong-All[of M f g]* **by** *auto*

lemma *bind-nonempty'*:

assumes $f \in \text{measurable } M$ (*subprob-algebra N*) $x \in \text{space } M$
shows $\text{bind } M f = \text{join } (\text{distr } M$ (*subprob-algebra N*) f)
using *assms*
apply (*subst bind-nonempty, blast*)
apply (*subst subprob-algebra-cong[OF sets-kernel[OF assms(1) someI-ex]], blast*)
apply (*simp add: subprob-algebra-cong[OF sets-kernel[OF assms]]*)
done

lemma *bind-nonempty''*:

assumes $f \in \text{measurable } M$ (*subprob-algebra N*) $\text{space } M \neq \{\}$
shows $\text{bind } M f = \text{join } (\text{distr } M$ (*subprob-algebra N*) f)
using *assms* **by** (*auto intro: bind-nonempty'*)

lemma *emeasure-bind*:

$\llbracket \text{space } M \neq \{\}; f \in \text{measurable } M$ (*subprob-algebra N*); $X \in \text{sets } N \rrbracket$
 $\implies \text{emeasure } (M \ggg f) X = \int^+ x. \text{emeasure } (f x) X \partial M$
by (*simp add: bind-nonempty'' emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra*)

lemma *nn-integral-bind*:

assumes $f: f \in \text{borel-measurable } B$
assumes $N: N \in \text{measurable } M$ (*subprob-algebra B*)
shows $(\int^+ x. f x \partial(M \ggg N)) = (\int^+ x. \int^+ y. f y \partial N x \partial M)$
proof *cases*
assume $M: \text{space } M \neq \{\}$ **show** *?thesis*
unfolding *bind-nonempty''[OF N M]* *nn-integral-join[OF f sets-distr]*
by (*rule nn-integral-distr[OF N]*)
(simp add: f nn-integral-measurable-subprob-algebra)

qed (*simp add: bind-empty space-empty[of M] nn-integral-count-space*)

lemma *AE-bind*:

assumes $N[\text{measurable}]$: $N \in \text{measurable } M$ (*subprob-algebra B*)

assumes $P[\text{measurable}]$: $\text{Measurable.pred } B P$

shows $(AE x \text{ in } M \gg N. P x) \longleftrightarrow (AE x \text{ in } M. AE y \text{ in } N x. P y)$

proof *cases*

assume M : *space* $M = \{\}$ **show** *?thesis*

unfolding *bind-empty[OF M]* **unfolding** *space-empty[OF M]* **by** (*simp add: AE-count-space*)

next

assume M : *space* $M \neq \{\}$

note *sets-kernel[OF N, simp]*

have $*$: $(\int^+ x. \text{indicator } \{x. \neg P x\} x \partial(M \gg N)) = (\int^+ x. \text{indicator } \{x \in \text{space } B. \neg P x\} x \partial(M \gg N))$

by (*intro nn-integral-cong*) (*simp add: space-bind[OF - M] split: split-indicator*)

have $(AE x \text{ in } M \gg N. P x) \longleftrightarrow (\int^+ x. \text{integral}^N (N x) (\text{indicator } \{x \in \text{space } B. \neg P x\} \partial M) = 0)$

by (*simp add: AE-iff-nn-integral sets-bind[OF - M] space-bind[OF - M] * nn-integral-bind[where B=B]*)

del: nn-integral-indicator)

also have $\dots = (AE x \text{ in } M. AE y \text{ in } N x. P y)$

apply (*subst nn-integral-0-iff-AE*)

apply (*rule measurable-compose[OF N nn-integral-measurable-subprob-algebra]*)

apply *measurable*

apply (*intro eventually-subst AE-I2*)

apply (*auto simp add: subprob-measurableD(1)[OF N]*)

intro!: AE-iff-measurable[symmetric])

done

finally show *?thesis* .

qed

lemma *measurable-bind'*:

assumes $M1$: $f \in \text{measurable } M$ (*subprob-algebra N*) **and**

$M2$: *case-prod* $g \in \text{measurable } (M \otimes_M N)$ (*subprob-algebra R*)

shows $(\lambda x. \text{bind } (f x) (g x)) \in \text{measurable } M$ (*subprob-algebra R*)

proof (*subst measurable-cong*)

fix x **assume** $x\text{-in-}M$: $x \in \text{space } M$

with *assms* **have** *space* $(f x) \neq \{\}$

by (*blast dest: subprob-space-kernel subprob-space.subprob-not-empty*)

moreover from $M2$ $x\text{-in-}M$ **have** $g x \in \text{measurable } (f x)$ (*subprob-algebra R*)

by (*subst measurable-cong-sets[OF sets-kernel[OF M1 x-in-M] refl]*)

(*auto dest: measurable-Pair2*)

ultimately show $\text{bind } (f x) (g x) = \text{join } (\text{distr } (f x) (\text{subprob-algebra } R) (g x))$

by (*simp-all add: bind-nonempty'*)

next

show $(\lambda w. \text{join } (\text{distr } (f w) (\text{subprob-algebra } R) (g w))) \in \text{measurable } M$ (*subprob-algebra R*)

```

apply (rule measurable-compose[OF - measurable-join])
apply (rule measurable-distr2[OF M2 M1])
done
qed

```

```

lemma measurable-bind[measurable (raw)]:
  assumes M1:  $f \in \text{measurable } M \text{ (subprob-algebra } N)$  and
    M2:  $(\lambda x. g \text{ (fst } x) \text{ (snd } x)) \in \text{measurable } (M \otimes_M N) \text{ (subprob-algebra } R)$ 
  shows  $(\lambda x. \text{bind } (f \ x) \ (g \ x)) \in \text{measurable } M \text{ (subprob-algebra } R)$ 
  using assms by (auto intro: measurable-bind' simp: measurable-split-conv)

```

```

lemma measurable-bind2:
  assumes  $f \in \text{measurable } M \text{ (subprob-algebra } N)$  and  $g \in \text{measurable } N \text{ (subprob-algebra } R)$ 
  shows  $(\lambda x. \text{bind } (f \ x) \ g) \in \text{measurable } M \text{ (subprob-algebra } R)$ 
  using assms by (intro measurable-bind' measurable-const) auto

```

```

lemma subprob-space-bind:
  assumes subprob-space  $M \ f \in \text{measurable } M \text{ (subprob-algebra } N)$ 
  shows subprob-space  $(M \ggg f)$ 
proof (rule subprob-space-kernel[of  $\lambda x. x \ggg f$ ])
  show  $(\lambda x. x \ggg f) \in \text{measurable } (\text{subprob-algebra } M) \text{ (subprob-algebra } N)$ 
    by (rule measurable-bind, rule measurable-ident-sets, rule refl,
      rule measurable-compose[OF measurable-snd assms(2)])
  from assms(1) show  $M \in \text{space } (\text{subprob-algebra } M)$ 
    by (simp add: space-subprob-algebra)
qed

```

```

lemma
  fixes  $f :: - \Rightarrow \text{real}$ 
  assumes f-measurable [measurable]:  $f \in \text{borel-measurable } K$ 
  and f-bounded:  $\bigwedge x. x \in \text{space } K \implies |f \ x| \leq B$ 
  and N [measurable]:  $N \in \text{measurable } M \text{ (subprob-algebra } K)$ 
  and fin: finite-measure  $M$ 
  and M-bounded:  $\text{AE } x \text{ in } M. \text{emeasure } (N \ x) \ (\text{space } (N \ x)) \leq \text{ennreal } B'$ 
  shows integrable-bind:  $\text{integrable } (\text{bind } M \ N) \ f \ (\text{is } ?\text{integrable})$ 
  and integral-bind:  $\text{integral}^L (\text{bind } M \ N) \ f = \int x. \text{integral}^L (N \ x) \ f \ \partial M \ (\text{is } ?\text{integral})$ 
proof(case-tac [!] space  $M = \{\}$ )
  assume [simp]: space  $M \neq \{\}$ 
  interpret finite-measure  $M$  by(rule fin)

  have integrable  $(\text{join } (\text{distr } M \ (\text{subprob-algebra } K) \ N)) \ f$ 
    using f-measurable f-bounded
  by(rule integrable-join[where  $B'=B'$ ])(simp-all add: finite-measure-distr AE-distr-iff
M-bounded)
  then show ?integrable by(simp add: bind-nonempty'[where  $N=K$ ])

```

```

have  $\text{integral}^L (\text{join } (\text{distr } M \ (\text{subprob-algebra } K) \ N)) \ f = \int M'. \text{integral}^L M'$ 

```

$f \partial \text{distr } M \text{ (subprob-algebra } K) N$
using $f\text{-measurable } f\text{-bounded}$
by(rule *integral-join*[**where** $B'=B'$])(*simp-all add: finite-measure-distr AE-distr-iff*
 $M\text{-bounded}$)
also have $\dots = \int x. \text{integral}^L (N x) f \partial M$
by(rule *integral-distr*)(*simp-all add: integral-measurable-subprob-algebra*[*OF -*])
finally show $? \text{integral}$ **by**(*simp add: bind-nonempty''*[**where** $N=K$])
qed(*simp-all add: bind-def integrable-count-space lebesgue-integral-count-space-finite*
 $\text{Bochner-Integration.integral-empty}$)

lemma (**in** *prob-space*) *prob-space-bind*:
assumes $ae: AE x \text{ in } M. \text{prob-space } (N x)$
and $N[\text{measurable}]: N \in \text{measurable } M \text{ (subprob-algebra } S)$
shows *prob-space* ($M \ggg N$)
proof
have $\text{emeasure } (M \ggg N) (\text{space } (M \ggg N)) = (\int^+ x. \text{emeasure } (N x) (\text{space}$
 $(N x)) \partial M)$
by (*subst emeasure-bind*[**where** $N=S$])
(*auto simp: not-empty space-bind*[*OF sets-kernel*] *subprob-measurableD*[*OF*
 N] *intro!: nn-integral-cong*)
also have $\dots = (\int^+ x. 1 \partial M)$
using ae **by** (*intro nn-integral-cong-AE, eventually-elim*) (*rule prob-space.emeasure-space-1*)
finally show $\text{emeasure } (M \ggg N) (\text{space } (M \ggg N)) = 1$
by (*simp add: emeasure-space-1*)
qed

lemma (**in** *subprob-space*) *bind-in-space*:
 $A \in \text{measurable } M \text{ (subprob-algebra } N) \implies (M \ggg A) \in \text{space } (\text{subprob-algebra}$
 $N)$
by (*auto simp add: space-subprob-algebra subprob-not-empty sets-kernel intro!:*
subprob-space-bind)
unfold-locales

lemma (**in** *subprob-space*) *measure-bind*:
assumes $f: f \in \text{measurable } M \text{ (subprob-algebra } N)$ **and** $X: X \in \text{sets } N$
shows $\text{measure } (M \ggg f) X = \int x. \text{measure } (f x) X \partial M$
proof –
interpret $Mf: \text{subprob-space } M \ggg f$
by (*rule subprob-space-bind*[*OF - f*] *unfold-locales*)
{ **fix** x **assume** $x \in \text{space } M$
from f [*THEN measurable-space, OF this*] **interpret** *subprob-space* $f x$
by (*simp add: space-subprob-algebra*)
have $\text{emeasure } (f x) X = \text{ennreal } (\text{measure } (f x) X) \text{measure } (f x) X \leq 1$
by (*auto simp: emeasure-eq-measure subprob-measure-le-1*) }
note this [*simp*]

have $\text{emeasure } (M \ggg f) X = \int^+ x. \text{emeasure } (f x) X \partial M$
using *subprob-not-empty f X* **by** (*rule emeasure-bind*)

also have $\dots = \int^+ x. \text{ennreal } (\text{measure } (f \ x) \ X) \ \partial M$
by (*intro nn-integral-cong*) *simp*
also have $\dots = \int x. \text{measure } (f \ x) \ X \ \partial M$
by (*intro nn-integral-eq-integral integrable-const-bound*[**where** $B=1$]
measure-measurable-subprob-algebra2[$OF - f$] *pair-measureI* X)
(auto simp: measure-nonneg)
finally show *?thesis*
by (*simp add: Mf.emeasure-eq-measure measure-nonneg integral-nonneg*)
qed

lemma *emeasure-bind-const*:
 $\text{space } M \neq \{\}$ $\implies X \in \text{sets } N \implies \text{subprob-space } N \implies$
 $\text{emeasure } (M \gg (\lambda x. N)) \ X = \text{emeasure } N \ X * \text{emeasure } M \ (\text{space } M)$
by (*simp add: bind-nonempty emeasure-join nn-integral-distr*
space-subprob-algebra measurable-emeasure-subprob-algebra)

lemma *emeasure-bind-const'*:
assumes *subprob-space M subprob-space N*
shows $\text{emeasure } (M \gg (\lambda x. N)) \ X = \text{emeasure } N \ X * \text{emeasure } M \ (\text{space } M)$
using *assms*
proof (*case-tac X \in sets N*)
fix X **assume** $X \in \text{sets } N$
thus $\text{emeasure } (M \gg (\lambda x. N)) \ X = \text{emeasure } N \ X * \text{emeasure } M \ (\text{space } M)$
using *assms*
by (*subst emeasure-bind-const*)
(simp-all add: subprob-space.subprob-not-empty subprob-space.emeasure-space-le-1)
next
fix X **assume** $X \notin \text{sets } N$
with *assms* **show** $\text{emeasure } (M \gg (\lambda x. N)) \ X = \text{emeasure } N \ X * \text{emeasure } M \ (\text{space } M)$
by (*simp add: sets-bind*[*of - - N*] *subprob-space.subprob-not-empty*
space-subprob-algebra emeasure-notin-sets)
qed

lemma *emeasure-bind-const-prob-space*:
assumes *prob-space M subprob-space N*
shows $\text{emeasure } (M \gg (\lambda x. N)) \ X = \text{emeasure } N \ X$
using *assms* **by** (*simp add: emeasure-bind-const' prob-space-imp-subprob-space*
prob-space.emeasure-space-1)

lemma *bind-return*:
assumes $f \in \text{measurable } M \ (\text{subprob-algebra } N)$ **and** $x \in \text{space } M$
shows $\text{bind } (\text{return } M \ x) \ f = f \ x$
using *sets-kernel*[OF *assms*] *assms*
by (*simp-all add: distr-return join-return subprob-space-kernel bind-nonempty'*
cong: subprob-algebra-cong)

lemma *bind-return'*:
shows $\text{bind } M \ (\text{return } M) = M$

by (cases space $M = \{\}$)
 (simp-all add: bind-empty space-empty[symmetric] bind-nonempty join-return'
 cong: subprob-algebra-cong)

lemma *distr-bind*:

assumes $N: N \in \text{measurable } M \text{ (subprob-algebra } K) \text{ space } M \neq \{\}$
 assumes $f: f \in \text{measurable } K R$
 shows $\text{distr } (M \ggg N) R f = (M \ggg (\lambda x. \text{distr } (N x) R f))$
 unfolding bind-nonempty''[OF N]
 apply (subst bind-nonempty''[OF measurable-compose[OF $N(1)$ measurable-distr
 $N(2)$]])
 apply (rule f)
 apply (simp add: join-distr-distr[OF - f , symmetric])
 apply (subst distr-distr[OF measurable-distr, OF $f N(1)$])
 apply (simp add: comp-def)
 done

lemma *bind-distr*:

assumes $f[\text{measurable}]: f \in \text{measurable } M X$
 assumes $N[\text{measurable}]: N \in \text{measurable } X \text{ (subprob-algebra } K) \text{ and space } M \neq \{\}$
 shows $(\text{distr } M X f \ggg N) = (M \ggg (\lambda x. N (f x)))$
 proof –
 have space $X \neq \{\}$ space $M \neq \{\}$
 using $\langle \text{space } M \neq \{\} \rangle f[\text{THEN measurable-space}]$ by auto
 then show ?thesis
 by (simp add: bind-nonempty''[where $N=K$] distr-distr comp-def)
 qed

lemma *bind-count-space-singleton*:

assumes subprob-space $(f x)$
 shows count-space $\{x\} \ggg f = f x$
 proof –
 have $A: \bigwedge A. A \subseteq \{x\} \implies A = \{\} \vee A = \{x\}$ by auto
 have count-space $\{x\} = \text{return } (\text{count-space } \{x\}) x$
 by (intro measure-eqI) (auto dest: A)
 also have ... $\ggg f = f x$
 by (subst bind-return[of - - $f x$]) (auto simp: space-subprob-algebra assms)
 finally show ?thesis .
 qed

lemma *restrict-space-bind*:

assumes $N: N \in \text{measurable } M \text{ (subprob-algebra } K)$
 assumes space $M \neq \{\}$
 assumes $X[\text{simp}]: X \in \text{sets } K X \neq \{\}$
 shows $\text{restrict-space } (bind M N) X = bind M (\lambda x. \text{restrict-space } (N x) X)$
 proof (rule measure-eqI)
 note $N\text{-sets} = \text{sets-bind}[OF \text{sets-kernel}[OF N] \text{assms}(2), \text{simp}]$
 note $N\text{-space} = \text{sets-eq-imp-space-eq}[OF $N\text{-sets}$, \text{simp}]$

```

show sets (restrict-space (bind M N) X) = sets (bind M ( $\lambda x$ . restrict-space (N
x) X))
by (simp add: sets-restrict-space assms(2) sets-bind[OF sets-kernel[OF restrict-space-measurable[OF
assms(4,3,1)]]])
fix A assume A  $\in$  sets (restrict-space (M  $\gg$  N) X)
with X have A  $\in$  sets K A  $\subseteq$  X
by (auto simp: sets-restrict-space)
then show emeasure (restrict-space (M  $\gg$  N) X) A = emeasure (M  $\gg$  ( $\lambda x$ .
restrict-space (N x) X)) A
using assms
apply (subst emeasure-restrict-space)
apply (simp-all add: emeasure-bind[OF assms(2,1)])
apply (subst emeasure-bind[OF - restrict-space-measurable[OF - - N]])
apply (auto simp: sets-restrict-space emeasure-restrict-space space-subprob-algebra
intro!: nn-integral-cong dest!: measurable-space)
done
qed

```

lemma bind-restrict-space:

```

assumes A: A  $\cap$  space M  $\neq$  {} A  $\cap$  space M  $\in$  sets M
and f: f  $\in$  measurable (restrict-space M A) (subprob-algebra N)
shows restrict-space M A  $\gg$  f = M  $\gg$  ( $\lambda x$ . if x  $\in$  A then f x else null-measure
(f (SOME x. x  $\in$  A  $\wedge$  x  $\in$  space M)))
(is ?lhs = ?rhs is - = M  $\gg$  ?f)
proof -
let ?P =  $\lambda x$ . x  $\in$  A  $\wedge$  x  $\in$  space M
let ?x = Eps ?P
let ?c = null-measure (f ?x)
from A have ?P ?x by-(rule someI-ex, blast)
hence ?x  $\in$  space (restrict-space M A) by(simp add: space-restrict-space)
with f have f ?x  $\in$  space (subprob-algebra N) by(rule measurable-space)
hence sps: subprob-space (f ?x)
and sets: sets (f ?x) = sets N
by(simp-all add: space-subprob-algebra)
have space (f ?x)  $\neq$  {} using sps by(rule subprob-space.subprob-not-empty)
moreover have sets ?c = sets N by(simp add: null-measure-def)(simp add:
sets)
ultimately have c: ?c  $\in$  space (subprob-algebra N)
by(simp add: space-subprob-algebra subprob-space-null-measure)
from f A c have f': ?f  $\in$  measurable M (subprob-algebra N)
by(simp add: measurable-restrict-space-iff)

from A have [simp]: space M  $\neq$  {} by blast

have ?lhs = join (distr (restrict-space M A) (subprob-algebra N) f)
using assms by(simp add: space-restrict-space bind-nonempty'')
also have ... = join (distr M (subprob-algebra N) ?f)
by(rule measure-eqI)(auto simp add: emeasure-join nn-integral-distr nn-integral-restrict-space
f f' A intro: nn-integral-cong)

```

also have $\dots = ?rhs$ using f' by (simp add: bind-nonempty')
 finally show ?thesis .
 qed

lemma *bind-const'*: $\llbracket \text{prob-space } M; \text{subprob-space } N \rrbracket \implies M \gg= (\lambda x. N) = N$
 by (intro measure-eqI)
 (simp-all add: space-subprob-algebra prob-space.not-empty emeasure-bind-const-prob-space)

lemma *bind-return-distr*:
 $\text{space } M \neq \{\}$ $\implies f \in \text{measurable } M N \implies \text{bind } M (\text{return } N \circ f) = \text{distr } M N f$
 apply (simp add: bind-nonempty)
 apply (subst subprob-algebra-cong)
 apply (rule sets-return)
 apply (subst distr-distr[symmetric])
 apply (auto intro!: return-measurable simp: distr-distr[symmetric] join-return')
 done

lemma *bind-return-distr'*:
 $\text{space } M \neq \{\}$ $\implies f \in \text{measurable } M N \implies \text{bind } M (\lambda x. \text{return } N (f x)) = \text{distr } M N f$
 using bind-return-distr[of M f N] by (simp add: comp-def)

lemma *bind-assoc*:
 fixes $f :: 'a \Rightarrow 'b$ measure and $g :: 'b \Rightarrow 'c$ measure
 assumes $M1: f \in \text{measurable } M (\text{subprob-algebra } N)$ and $M2: g \in \text{measurable } N (\text{subprob-algebra } R)$
 shows $\text{bind } (\text{bind } M f) g = \text{bind } M (\lambda x. \text{bind } (f x) g)$
 proof (cases $\text{space } M = \{\}$)
 assume [simp]: $\text{space } M \neq \{\}$
 from assms have [simp]: $\text{space } N \neq \{\}$ $\text{space } R \neq \{\}$
 by (auto simp: measurable-empty-iff space-subprob-algebra-empty-iff)
 from assms have sets- fx [simp]: $\bigwedge x. x \in \text{space } M \implies \text{sets } (f x) = \text{sets } N$
 by (simp add: sets-kernel)
 have ex-in: $\bigwedge A. A \neq \{\} \implies \exists x. x \in A$ by blast
 note sets-some[simp] = sets-kernel[OF M1 someI-ex[OF ex-in[OF $\langle \text{space } M \neq \{\} \rangle$]]]
 sets-kernel[OF M2 someI-ex[OF ex-in[OF $\langle \text{space } N \neq \{\} \rangle$]]]
 note space-some[simp] = sets-eq-imp-space-eq[OF this(1)] sets-eq-imp-space-eq[OF this(2)]

have $\text{bind } M (\lambda x. \text{bind } (f x) g) =$
 $\text{join } (\text{distr } M (\text{subprob-algebra } R) (\text{join} \circ (\lambda x. (\text{distr } x (\text{subprob-algebra } R) g)) \circ f))$
 by (simp add: sets-eq-imp-space-eq[OF sets- fx] bind-nonempty o-def cong: subprob-algebra-cong distr-cong)
 also have $\text{distr } M (\text{subprob-algebra } R) (\text{join} \circ (\lambda x. (\text{distr } x (\text{subprob-algebra } R) g)) \circ f) =$
 $\text{distr } (\text{distr } (\text{distr } M (\text{subprob-algebra } N) f)$


```

      (subprob-algebra (subprob-algebra R))
      (λx. distr x (subprob-algebra R) g))
      (subprob-algebra R) join
apply (subst distr-distr,
        (blast intro: measurable-comp measurable-distr measurable-join M1
M2)+)+
apply (simp add: o-assoc)
done
also have join ... = bind (bind M f) g
by (simp add: join-assoc join-distr-distr M2 bind-nonempty cong: subprob-algebra-cong)
finally show ?thesis ..
qed (simp add: bind-empty)

```

lemma *double-bind-assoc*:

```

assumes Mg: g ∈ measurable N (subprob-algebra N')
assumes Mf: f ∈ measurable M (subprob-algebra M')
assumes Mh: case-prod h ∈ measurable (M ⊗M M') N
shows do {x ← M; y ← f x; g (h x y)} = do {x ← M; y ← f x; return N (h x
y)} ≫ g
proof –
have do {x ← M; y ← f x; return N (h x y)} ≫ g =
      do {x ← M; do {y ← f x; return N (h x y)} ≫ g}
using Mh by (auto intro!: bind-assoc measurable-bind[OF Mf] Mf Mg
      measurable-compose[OF - return-measurable] simp: measurable-split-conv)
also from Mh have ∧x. x ∈ space M ⇒ h x ∈ measurable M' N by measurable
hence do {x ← M; do {y ← f x; return N (h x y)} ≫ g} =
      do {x ← M; y ← f x; return N (h x y)} ≫ g}
apply (intro ballI bind-cong refl bind-assoc)
apply (subst measurable-cong-sets[OF sets-kernel[OF Mf] refl], simp)
apply (rule measurable-compose[OF - return-measurable], auto intro: Mg)
done
also have ∧x. x ∈ space M ⇒ space (f x) = space M'
by (intro sets-eq-imp-space-eq sets-kernel[OF Mf])
with measurable-space[OF Mh]
have do {x ← M; y ← f x; return N (h x y)} ≫ g} = do {x ← M; y ← f x;
g (h x y)}
by (intro ballI bind-cong bind-return[OF Mg]) (auto simp: space-pair-measure)
finally show ?thesis ..
qed

```

lemma (**in** *prob-space*) *M-in-subprob*[*measurable (raw)*]: $M \in \text{space (subprob-algebra } M)$

by (simp add: space-subprob-algebra) *unfold-locales*

lemma (**in** *pair-prob-space*) *pair-measure-eq-bind*:

$(M1 \otimes_M M2) = (M1 \gg (\lambda x. M2 \gg (\lambda y. \text{return } (M1 \otimes_M M2) (x, y))))$

proof (rule *measure-eqI*)

have *ps-M2*: *prob-space* M2 **by** *unfold-locales*

note *return-measurable*[*measurable*]

```

show sets (M1  $\otimes_M$  M2) = sets (M1  $\gg$  ( $\lambda x$ . M2  $\gg$  ( $\lambda y$ . return (M1  $\otimes_M$ 
M2) (x, y))))
  by (simp-all add: M1.not-empty M2.not-empty)
  fix A assume [measurable]: A  $\in$  sets (M1  $\otimes_M$  M2)
  show emeasure (M1  $\otimes_M$  M2) A = emeasure (M1  $\gg$  ( $\lambda x$ . M2  $\gg$  ( $\lambda y$ . return
(M1  $\otimes_M$  M2) (x, y)))) A
  by (auto simp: M2.emeasure-pair-measure M1.not-empty M2.not-empty emeasure-bind[where
N=M1  $\otimes_M$  M2]
      intro!: nn-integral-cong)
qed

```

lemma (in pair-prob-space) bind-rotate:

```

assumes C[measurable]: ( $\lambda(x, y)$ . C x y)  $\in$  measurable (M1  $\otimes_M$  M2) (subprob-algebra
N)

```

```

shows (M1  $\gg$  ( $\lambda x$ . M2  $\gg$  ( $\lambda y$ . C x y))) = (M2  $\gg$  ( $\lambda y$ . M1  $\gg$  ( $\lambda x$ . C x
y)))

```

proof –

```

interpret swap: pair-prob-space M2 M1 by unfold-locales
note measurable-bind[where N=M2, measurable]
note measurable-bind[where N=M1, measurable]
have [simp]: M1  $\in$  space (subprob-algebra M1) M2  $\in$  space (subprob-algebra M2)
  by (auto simp: space-subprob-algebra) unfold-locales
have (M1  $\gg$  ( $\lambda x$ . M2  $\gg$  ( $\lambda y$ . C x y))) =
  (M1  $\gg$  ( $\lambda x$ . M2  $\gg$  ( $\lambda y$ . return (M1  $\otimes_M$  M2) (x, y))))  $\gg$  ( $\lambda(x, y)$ . C x
y)
  by (auto intro!: bind-cong simp: bind-return[where N=N] space-pair-measure
bind-assoc[where N=M1  $\otimes_M$  M2 and R=N])
  also have ... = (distr (M2  $\otimes_M$  M1) (M1  $\otimes_M$  M2) ( $\lambda(x, y)$ . (y, x)))  $\gg$ 
( $\lambda(x, y)$ . C x y)
  unfolding pair-measure-eq-bind[symmetric] distr-pair-swap[symmetric] ..
  also have ... = (M2  $\gg$  ( $\lambda x$ . M1  $\gg$  ( $\lambda y$ . return (M2  $\otimes_M$  M1) (x, y))))  $\gg$ 
( $\lambda(y, x)$ . C x y)
  unfolding swap.pair-measure-eq-bind[symmetric]
  by (auto simp add: space-pair-measure M1.not-empty M2.not-empty bind-distr[OF
- C] intro!: bind-cong)
  also have ... = (M2  $\gg$  ( $\lambda y$ . M1  $\gg$  ( $\lambda x$ . C x y)))
  by (auto intro!: bind-cong simp: bind-return[where N=N] space-pair-measure
bind-assoc[where N=M2  $\otimes_M$  M1 and R=N])
  finally show ?thesis .
qed

```

lemma bind-return'': sets $M =$ sets $N \implies M \gg$ return $N = M$

```

by (cases space M = {})

```

```

  (simp-all add: bind-empty space-empty[symmetric] bind-nonempty join-return'
cong: subprob-algebra-cong)

```

lemma (in prob-space) distr-const[simp]:

```

c  $\in$  space N  $\implies$  distr M N ( $\lambda x$ . c) = return N c

```

```

by (rule measure-eqI) (auto simp: emeasure-distr emeasure-space-1)

```

lemma *return-count-space-eq-density*:

return (count-space M) x = density (count-space M) (indicator {x})

by (*rule measure-eqI*)

(*auto simp: indicator-inter-arith[symmetric] emeasure-density split: split-indicator*)

lemma *null-measure-in-space-subprob-algebra [simp]*:

null-measure M ∈ space (subprob-algebra M) ↔ space M ≠ {}

by(*simp add: space-subprob-algebra subprob-space-null-measure-iff*)

9.1 Giry monad on probability spaces

definition *prob-algebra* :: 'a measure ⇒ 'a measure measure **where**

prob-algebra K = restrict-space (subprob-algebra K) {M. prob-space M}

lemma *space-prob-algebra*: *space (prob-algebra M) = {N. sets N = sets M ∧ prob-space N}*

unfolding *prob-algebra-def* **by** (*auto simp: space-subprob-algebra space-restrict-space prob-space-imp-subprob-space*)

lemma *measurable-measure-prob-algebra[measurable]*:

a ∈ sets A ⇒ (λM. Sigma-Algebra.measure M a) ∈ prob-algebra A →_M borel

unfolding *prob-algebra-def* **by** (*intro measurable-restrict-space1 measurable-measure-subprob-algebra*)

lemma *measurable-prob-algebraD*:

f ∈ N →_M prob-algebra M ⇒ f ∈ N →_M subprob-algebra M

unfolding *prob-algebra-def measurable-restrict-space2-iff* **by** *auto*

lemma *measure-measurable-prob-algebra2*:

Sigma (space M) A ∈ sets (M ⊗_M N) ⇒ L ∈ M →_M prob-algebra N ⇒

(λx. Sigma-Algebra.measure (L x) (A x)) ∈ borel-measurable M

using *measure-measurable-subprob-algebra2[of M A N L]* **by** (*auto intro: measurable-prob-algebraD*)

lemma *measurable-prob-algebraI*:

(∧x. x ∈ space N ⇒ prob-space (f x)) ⇒ f ∈ N →_M subprob-algebra M ⇒

f ∈ N →_M prob-algebra M

unfolding *prob-algebra-def* **by** (*intro measurable-restrict-space2*) *auto*

lemma *measurable-distr-prob-space*:

assumes *f: f ∈ M →_M N*

shows *(λM'. distr M' N f) ∈ prob-algebra M →_M prob-algebra N*

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof (*intro conjI measurable-restrict-space1 measurable-distr f*)

show *(λM'. distr M' N f) ∈ space (restrict-space (subprob-algebra M) (Collect prob-space)) → Collect prob-space*

using *f* **by** (*auto simp: space-restrict-space space-subprob-algebra intro!: prob-space.prob-space-distr*)

qed

lemma *measurable-return-prob-space[measurable]*: *return N ∈ N →_M prob-algebra*

N

by (rule measurable-prob-algebraI) (auto simp: prob-space-return)

lemma measurable-distr-prob-space2[measurable (raw)]:

assumes $f: g \in L \rightarrow_M \text{prob-algebra } M$ $(\lambda(x, y). f x y) \in L \otimes_M M \rightarrow_M N$

shows $(\lambda x. \text{distr } (g x) N (f x)) \in L \rightarrow_M \text{prob-algebra } N$

unfolding prob-algebra-def measurable-restrict-space2-iff

proof (intro conjI measurable-restrict-space1 measurable-distr2[where $M=M$] f measurable-prob-algebraD)

show $(\lambda x. \text{distr } (g x) N (f x)) \in \text{space } L \rightarrow \text{Collect prob-space}$

using f subprob-measurableD[OF measurable-prob-algebraD[OF f(1)]]

by (auto simp: measurable-restrict-space2-iff prob-algebra-def
intro!: prob-space.prob-space-distr)

qed

lemma measurable-bind-prob-space:

assumes $f: f \in M \rightarrow_M \text{prob-algebra } N$ and $g: g \in N \rightarrow_M \text{prob-algebra } R$

shows $(\lambda x. \text{bind } (f x) g) \in M \rightarrow_M \text{prob-algebra } R$

unfolding prob-algebra-def measurable-restrict-space2-iff

proof (intro conjI measurable-restrict-space1 measurable-bind2[where $N=N$] f g measurable-prob-algebraD)

show $(\lambda x. f x \ggg g) \in \text{space } M \rightarrow \text{Collect prob-space}$

using g f subprob-measurableD[OF measurable-prob-algebraD[OF f]]

by (auto simp: measurable-restrict-space2-iff prob-algebra-def
intro!: prob-space.prob-space-bind[where $S=R$] AE-I2)

qed

lemma measurable-bind-prob-space2[measurable (raw)]:

assumes $f: f \in M \rightarrow_M \text{prob-algebra } N$ and $g: (\lambda(x, y). g x y) \in (M \otimes_M N) \rightarrow_M \text{prob-algebra } R$

shows $(\lambda x. \text{bind } (f x) (g x)) \in M \rightarrow_M \text{prob-algebra } R$

unfolding prob-algebra-def measurable-restrict-space2-iff

proof (intro conjI measurable-restrict-space1 measurable-bind[where $N=N$] f g measurable-prob-algebraD)

show $(\lambda x. f x \ggg g x) \in \text{space } M \rightarrow \text{Collect prob-space}$

using g f subprob-measurableD[OF measurable-prob-algebraD[OF f]]

using measurable-space[OF g]

by (auto simp: measurable-restrict-space2-iff prob-algebra-def space-pair-measure Pi-iff

intro!: prob-space.prob-space-bind[where $S=R$] AE-I2)

qed (insert g, simp)

lemma measurable-prob-algebra-generated:

assumes eq: sets $N = \text{sigma-sets } \Omega G$ and Int-stable G $G \subseteq \text{Pow } \Omega$

assumes subsp: $\bigwedge a. a \in \text{space } M \implies \text{prob-space } (K a)$

assumes sets: $\bigwedge a. a \in \text{space } M \implies \text{sets } (K a) = \text{sets } N$

assumes $\bigwedge A. A \in G \implies (\lambda a. \text{emeasure } (K a) A) \in \text{borel-measurable } M$

shows $K \in \text{measurable } M$ (prob-algebra N)

unfolding *measurable-restrict-space2-iff prob-algebra-def*
proof
show $K \in M \rightarrow_M \text{subprob-algebra } N$
proof (*rule measurable-subprob-algebra-generated[OF assms(1,2,3) - assms(5,6)]*)
fix a **assume** $a \in \text{space } M$ **then show** *subprob-space (K a)*
using *subsp[of a]* **by** (*intro prob-space-imp-subprob-space*)
next
have $(\lambda a. \text{emeasure } (K a) \Omega) \in \text{borel-measurable } M \longleftrightarrow (\lambda a. 1::\text{ennreal}) \in$
borel-measurable } M
using *sets-eq-imp-space-eq[of sigma \Omega G N] \langle G \subseteq \text{Pow } \Omega \rangle \text{eq sets-eq-imp-space-eq[OF*
sets]
prob-space.emeasure-space-1[OF subsp]
by (*intro measurable-cong*) *auto*
then show $(\lambda a. \text{emeasure } (K a) \Omega) \in \text{borel-measurable } M$ **by** *simp*
qed
qed (*insert subsp, auto*)

lemma *in-space-prob-algebra*:
 $x \in \text{space } (\text{prob-algebra } M) \implies \text{emeasure } x (\text{space } M) = 1$
unfolding *prob-algebra-def space-restrict-space space-subprob-algebra*
by (*auto dest!: prob-space.emeasure-space-1 sets-eq-imp-space-eq*)

lemma *prob-space-pair*:
assumes *prob-space M prob-space N* **shows** *prob-space (M \otimes_M N)*
proof –
interpret M : *prob-space M* **by** *fact*
interpret N : *prob-space N* **by** *fact*
interpret P : *pair-prob-space M N* **proof** **qed**
show *?thesis*
by *unfold-locales*
qed

lemma *measurable-pair-prob[measurable]*:
 $f \in M \rightarrow_M \text{prob-algebra } N \implies g \in M \rightarrow_M \text{prob-algebra } L \implies (\lambda x. f x \otimes_M g$
 $x) \in M \rightarrow_M \text{prob-algebra } (N \otimes_M L)$
unfolding *prob-algebra-def measurable-restrict-space2-iff*
by (*auto intro!: measurable-pair-measure prob-space-pair*)

lemma *emeasure-bind-prob-algebra*:
assumes A : $A \in \text{space } (\text{prob-algebra } N)$
assumes B : $B \in N \rightarrow_M \text{prob-algebra } L$
assumes X : $X \in \text{sets } L$
shows $\text{emeasure } (\text{bind } A B) X = (\int^+ x. \text{emeasure } (B x) X \partial A)$
using $A B$
by (*intro emeasure-bind[OF - - X]*)
(auto simp: space-prob-algebra measurable-prob-algebraD cong: measurable-cong-sets
intro!: prob-space.not-empty)

lemma *prob-space-bind'*:

assumes $A: A \in \text{space (prob-algebra } M)$ **and** $B: B \in M \rightarrow_M \text{prob-algebra } N$
shows $\text{prob-space } (A \gg B)$
using $\text{measurable-bind-prob-space}[OF \text{measurable-const, } OF A B, \text{ THEN measurable-space, of undefined count-space UNIV}]$
by $(\text{simp add: space-prob-algebra})$

lemma $\text{sets-bind}'$:

assumes $A: A \in \text{space (prob-algebra } M)$ **and** $B: B \in M \rightarrow_M \text{prob-algebra } N$
shows $\text{sets } (A \gg B) = \text{sets } N$
using $\text{measurable-bind-prob-space}[OF \text{measurable-const, } OF A B, \text{ THEN measurable-space, of undefined count-space UNIV}]$
by $(\text{simp add: space-prob-algebra})$

lemma $\text{bind-cong-AE}'$:

assumes $M: M \in \text{space (prob-algebra } L)$
and $f: f \in L \rightarrow_M \text{prob-algebra } N$ **and** $g: g \in L \rightarrow_M \text{prob-algebra } N$
and $ae: AE \ x \ \text{in } M. f \ x = g \ x$
shows $\text{bind } M \ f = \text{bind } M \ g$
proof $(\text{rule measure-egI})$
show $\text{sets } (M \gg f) = \text{sets } (M \gg g)$
unfolding $\text{sets-bind}'[OF \ M \ f] \ \text{sets-bind}'[OF \ M \ g] \ ..$
show $A \in \text{sets } (M \gg f) \implies \text{emeasure } (M \gg f) \ A = \text{emeasure } (M \gg g) \ A$
for A
unfolding $\text{sets-bind}'[OF \ M \ f]$
using $\text{emeasure-bind-prob-algebra}[OF \ M \ f, \ \text{of } A] \ \text{emeasure-bind-prob-algebra}[OF \ M \ g, \ \text{of } A] \ ae$
by $(\text{auto intro: nn-integral-cong-AE})$
qed

lemma density-discrete :

$\text{countable } A \implies \text{sets } N = \text{Set.Pow } A \implies (\bigwedge x. f \ x \geq 0) \implies (\bigwedge x. x \in A \implies f \ x = \text{emeasure } N \ \{x\}) \implies$
 $\text{density (count-space } A) \ f = N$
by $(\text{rule measure-egI-countable}[of \ - \ A]) \ (\text{auto simp: emeasure-density})$

lemma $\text{distr-density-discrete}$:

fixes f'
assumes $\text{countable } A$
assumes $f' \in \text{borel-measurable } M$
assumes $g \in \text{measurable } M \ (\text{count-space } A)$
defines $f \equiv \lambda x. \int^+ t. (\text{if } g \ t = x \ \text{then } 1 \ \text{else } 0) * f' \ t \ \partial M$
assumes $\bigwedge x. x \in \text{space } M \implies g \ x \in A$
shows $\text{density (count-space } A) \ (\lambda x. f \ x) = \text{distr (density } M \ f') \ (\text{count-space } A)$
 g
proof $(\text{rule density-discrete})$
fix x **assume** $x: x \in A$
have $f \ x = \int^+ t. \text{indicator } (g \ - \ \{x\} \cap \text{space } M) \ t * f' \ t \ \partial M \ (\text{is - = ?I})$
unfolding $f\text{-def}$
by $(\text{intro nn-integral-cong}) \ (\text{simp split: split-indicator})$

also from x **have** $in\text{-sets}: g - \{x\} \cap \text{space } M \in \text{sets } M$
by $(intro \text{ measurable-sets}[OF \text{ assms}(3)]) \text{ simp}$
have $?I = \text{emeasure } (density \ M \ f') (g - \{x\} \cap \text{space } M)$ **unfolding** $f\text{-def}$
by $(subst \ \text{emeasure-density}[OF \ \text{assms}(2) \ in\text{-sets}], \ \text{subst } \text{mult.commute})$ $(rule \ refl)$
also from $\text{assms}(3) \ x$ **have** $\dots = \text{emeasure } (distr \ (density \ M \ f') \ (\text{count-space } A) \ g) \ \{x\}$
by $(subst \ \text{emeasure-distr}) \ \text{simp-all}$
finally show $f \ x = \text{emeasure } (distr \ (density \ M \ f') \ (\text{count-space } A) \ g) \ \{x\}$.
qed $(insert \ \text{assms}, \ \text{auto})$

lemma $bind\text{-cong-AE}$:

assumes $M = N$
assumes $f: f \in \text{measurable } N \ (\text{subprob-algebra } B)$
assumes $g: g \in \text{measurable } N \ (\text{subprob-algebra } B)$
assumes $ae: AE \ x \ \text{in } N. \ f \ x = g \ x$
shows $bind \ M \ f = bind \ N \ g$

proof $cases$

assume $\text{space } N = \{\}$ **then show** $?thesis$
using $\langle M = N \rangle$ **by** $(simp \ \text{add: } bind\text{-empty})$

next

assume $\text{space } N \neq \{\}$

show $?thesis$ **unfolding** $\langle M = N \rangle$

proof $(rule \ \text{measure-eqI})$

have $*$: $\text{sets } (N \ggg f) = \text{sets } B$

using $\text{sets-bind}[OF \ \text{sets-kernel}[OF \ f] \ \langle \text{space } N \neq \{\} \rangle]$ **by** simp

then show $\text{sets } (N \ggg f) = \text{sets } (N \ggg g)$

using $\text{sets-bind}[OF \ \text{sets-kernel}[OF \ g] \ \langle \text{space } N \neq \{\} \rangle]$ **by** auto

fix A **assume** $A \in \text{sets } (N \ggg f)$

then have $A \in \text{sets } B$

unfolding $*$.

with $ae \ f \ g \ \langle \text{space } N \neq \{\} \rangle$ **show** $\text{emeasure } (N \ggg f) \ A = \text{emeasure } (N \ggg g) \ A$

by $(subst \ (1 \ 2) \ \text{emeasure-bind}[\text{where } N=B]) \ (\text{auto } \text{intro!}: \ \text{nn-integral-cong-AE})$

qed

qed

lemma $bind\text{-cong-strong}$: $M = N \implies (\bigwedge x. x \in \text{space } M = \text{simp} \implies f \ x = g \ x) \implies bind \ M \ f = bind \ N \ g$

by $(\text{auto } \text{simp}: \ \text{simp-implies-def } \text{intro!}: \ \text{bind-cong})$

lemma $\text{sets-bind-measurable}$:

assumes $f: f \in \text{measurable } M \ (\text{subprob-algebra } B)$

assumes $M: \text{space } M \neq \{\}$

shows $\text{sets } (M \ggg f) = \text{sets } B$

using M **by** $(intro \ \text{sets-bind}[OF \ \text{sets-kernel}[OF \ f]]) \ \text{auto}$

lemma $\text{space-bind-measurable}$:

assumes $f: f \in \text{measurable } M \ (\text{subprob-algebra } B)$

assumes M : *space* $M \neq \{\}$
shows *space* $(M \gg f) = \text{space } B$
using M **by** (*intro space-bind*[*OF sets-kernel*[*OF f*]]) *auto*

lemma *bind-distr-return*:

$f \in M \rightarrow_M N \implies g \in N \rightarrow_M L \implies \text{space } M \neq \{\} \implies$
 $\text{distr } M N f \gg (\lambda x. \text{return } L (g x)) = \text{distr } M L (\lambda x. g (f x))$
by (*subst bind-distr*[*OF - measurable-compose*[*OF - return-measurable*]])
(auto intro!: bind-return-distr')

end

10 Projective Family

theory *Projective-Family*

imports *Giry-Monad*

begin

lemma *vimage-restrict-preseve-mono*:

assumes J : $J \subseteq I$
and *sets*: $A \subseteq (\prod_{E \in J} S i)$ $B \subseteq (\prod_{E \in J} S i)$ **and** *ne*: $(\prod_{E \in I} S i) \neq \{\}$
and *eq*: $(\lambda x. \text{restrict } x J) - 'A \cap (\prod_{E \in I} S i) \subseteq (\lambda x. \text{restrict } x J) - 'B \cap$
 $(\prod_{E \in I} S i)$
shows $A \subseteq B$

proof (*intro subsetI*)

fix x **assume** $x \in A$

from *ne* **obtain** y **where** $y: \bigwedge i. i \in I \implies y i \in S i$ **by** *auto*

have $J \cap (I - J) = \{\}$ **by** *auto*

show $x \in B$

proof *cases*

assume $x: x \in (\prod_{E \in J} S i)$

have *merge* $J (I - J) (x, y) \in (\lambda x. \text{restrict } x J) - 'A \cap (\prod_{E \in I} S i)$

using $y x \langle J \subseteq I \rangle$ *PiE-cancel-merge*[*of J I - J x y S*] $\langle x \in A \rangle$

by (*auto simp del: PiE-cancel-merge simp add: Un-absorb1*)

also have $\dots \subseteq (\lambda x. \text{restrict } x J) - 'B \cap (\prod_{E \in I} S i)$ **by** *fact*

finally show $x \in B$

using $y x \langle J \subseteq I \rangle$ *PiE-cancel-merge*[*of J I - J x y S*] $\langle x \in A \rangle$ *eq*

by (*auto simp del: PiE-cancel-merge simp add: Un-absorb1*)

qed (*insert* $\langle x \in A \rangle$ *sets, auto*)

qed

locale *projective-family* =

fixes $I :: 'i \text{ set}$ **and** $P :: 'i \text{ set} \implies ('i \implies 'a) \text{ measure}$ **and** $M :: 'i \implies 'a \text{ measure}$

assumes P : $\bigwedge J H. J \subseteq H \implies \text{finite } H \implies H \subseteq I \implies P J = \text{distr } (P H)$
 $(PiM J M) (\lambda f. \text{restrict } f J)$

assumes *prob-space-P*: $\bigwedge J. \text{finite } J \implies J \subseteq I \implies \text{prob-space } (P J)$

begin

lemma *sets-P*: $\text{finite } J \implies J \subseteq I \implies \text{sets } (P J) = \text{sets } (PiM J M)$

by (subst P[of J J]) simp-all

lemma space-P: finite J $\implies J \subseteq I \implies \text{space } (P J) = \text{space } (PiM J M)$
 using sets-P by (rule sets-eq-imp-space-eq)

lemma not-empty-M: $i \in I \implies \text{space } (M i) \neq \{\}$
 using prob-space-P[THEN prob-space.not-empty] by (auto simp: space-PiM space-P)

lemma not-empty: $\text{space } (PiM I M) \neq \{\}$
 by (simp add: not-empty-M)

abbreviation

emb L K \equiv prod-emb L M K

lemma emb-preserve-mono:

assumes $J \subseteq L \subseteq I$ and sets: $X \in \text{sets } (PiM J M)$ $Y \in \text{sets } (PiM J M)$

assumes $\text{emb } L J X \subseteq \text{emb } L J Y$

shows $X \subseteq Y$

proof (rule vimage-restrict-preseve-mono)

show $X \subseteq (\Pi_E i \in J. \text{space } (M i))$ $Y \subseteq (\Pi_E i \in J. \text{space } (M i))$

using sets[THEN sets.sets-into-space] by (auto simp: space-PiM)

show $(\Pi_E i \in L. \text{space } (M i)) \neq \{\}$

using $\langle L \subseteq I \rangle$ by (auto simp add: not-empty-M space-PiM[symmetric])

show $(\lambda x. \text{restrict } x J) - ' X \cap (\Pi_E i \in L. \text{space } (M i)) \subseteq (\lambda x. \text{restrict } x J) - ' Y \cap (\Pi_E i \in L. \text{space } (M i))$

using $\langle \text{prod-emb } L M J X \subseteq \text{prod-emb } L M J Y \rangle$ by (simp add: prod-emb-def)

qed fact

lemma emb-injective:

assumes $L: J \subseteq L \subseteq I$ and $X: X \in \text{sets } (PiM J M)$ and $Y: Y \in \text{sets } (PiM J M)$

shows $\text{emb } L J X = \text{emb } L J Y \implies X = Y$

by (intro antisym emb-preserve-mono[OF L X Y] emb-preserve-mono[OF L Y X]) auto

lemma emeasure-P: $J \subseteq K \implies \text{finite } K \implies K \subseteq I \implies X \in \text{sets } (PiM J M)$
 $\implies P K (\text{emb } K J X) = P J X$

by (auto intro!: emeasure-distr-restrict[symmetric] simp: P sets-P)

inductive-set generator :: $(i \Rightarrow a)$ set set **where**

finite J $\implies J \subseteq I \implies X \in \text{sets } (PiM J M) \implies \text{emb } I J X \in \text{generator}$

lemma algebra-generator: algebra (space (PiM I M)) generator

unfolding algebra-iff-Int

proof (safe elim!: generator.cases)

fix J X assume $J: \text{finite } J \subseteq I$ and $X: X \in \text{sets } (PiM J M)$

from X [THEN sets.sets-into-space] J show $x \in \text{emb } I J X \implies x \in \text{space } (PiM I M)$ for x

```

  by (auto simp: prod-emb-def space-PiM)

  have emb I J (space (PiM J M) - X) ∈ generator
    by (intro generator.intros J sets.Diff sets.top X)
  with J show space (PiM I M) - emb I J X ∈ generator
    by (simp add: space-PiM prod-emb-PiE)

  fix K Y assume K: finite K K ⊆ I and Y: Y ∈ sets (PiM K M)
  have emb I (J ∪ K) (emb (J ∪ K) J X) ∩ emb I (J ∪ K) (emb (J ∪ K) K
  Y) ∈ generator
    unfolding prod-emb-Int[symmetric]
    by (intro generator.intros sets.Int measurable-prod-emb) (auto intro!: J K X
  Y)
  with J K X Y show emb I J X ∩ emb I K Y ∈ generator
    by simp
  qed (force simp: generator.simps prod-emb-empty[symmetric])

  interpretation generator: algebra space (PiM I M) generator
    by (rule algebra-generator)

  lemma sets-PiM-generator: sets (PiM I M) = sigma-sets (space (PiM I M))
  generator
  proof (intro antisym sets.sigma-sets-subset)
    show sets (PiM I M) ⊆ sigma-sets (space (PiM I M)) generator
      unfolding sets-PiM-single space-PiM[symmetric]
    proof (intro sigma-sets-mono', safe)
      fix i A assume i ∈ I and A: A ∈ sets (M i)
      then have {f ∈ space (PiM I M). f i ∈ A} = emb I {i} (ΠE j∈{i}. A)
        by (auto simp: prod-emb-def space-PiM restrict-def Pi-iff PiE-iff extensional-def)
      with ⟨i∈I⟩ A show {f ∈ space (PiM I M). f i ∈ A} ∈ generator
        by (auto intro!: generator.intros sets-PiM-I-finite)
    qed
  qed (auto elim!: generator.cases)

  definition mu-G (μG) where
    μG A = (THE x. ∀ J⊆I. finite J ⟶ (∀ X∈sets (PiM J M). A = emb I J X
  ⟶ x = emeasure (P J) X))

  definition lim :: ('i ⇒ 'a) measure where
    lim = extend-measure (space (PiM I M)) generator (λx. x) μG

  lemma space-lim[simp]: space lim = space (PiM I M)
    using generator.space-closed
    unfolding lim-def by (intro space-extend-measure) simp

  lemma sets-lim[simp, measurable]: sets lim = sets (PiM I M)
    using generator.space-closed by (simp add: lim-def sets-PiM-generator sets-extend-measure)

  lemma mu-G-spec:

```

assumes J : finite $J \subseteq I \ X \in \text{sets } (Pi_M \ J \ M)$
shows $\mu G \ (emb \ I \ J \ X) = \text{emeasure } (P \ J) \ X$
unfolding $\mu\text{-}G\text{-def}$
proof (*intro the-equality allI impI ballI*)
fix $K \ Y$ **assume** K : finite $K \subseteq I \ Y \in \text{sets } (Pi_M \ K \ M)$
and [*simp*]: $emb \ I \ J \ X = emb \ I \ K \ Y$
have $\text{emeasure } (P \ K) \ Y = \text{emeasure } (P \ (K \cup \ J)) \ (emb \ (K \cup \ J) \ K \ Y)$
using $K \ J$ **by** (*simp add: emeasure-P*)
also have $emb \ (K \cup \ J) \ K \ Y = emb \ (K \cup \ J) \ J \ X$
using $K \ J$ **by** (*simp add: emb-injective[of K \cup J I]*)
also have $\text{emeasure } (P \ (K \cup \ J)) \ (emb \ (K \cup \ J) \ J \ X) = \text{emeasure } (P \ J) \ X$
using $K \ J$ **by** (*subst emeasure-P*) *simp-all*
finally show $\text{emeasure } (P \ J) \ X = \text{emeasure } (P \ K) \ Y \ ..$
qed (*insert J, force*)

lemma *positive-mu-G: positive generator μG*

proof –
show *?thesis*
proof (*safe intro!: positive-def[THEN iffD2]*)
obtain J **where** finite $J \subseteq I$ **by** *auto*
then have $\mu G \ (emb \ I \ J \ \{\}) = 0$
by (*subst mu-G-spec*) *auto*
then show $\mu G \ \{\} = 0$ **by** *simp*
qed
qed

lemma *additive-mu-G: additive generator μG*

proof (*safe intro!: additive-def[THEN iffD2] elim!: generator.cases*)
fix $J \ X \ K \ Y$ **assume** J : finite $J \subseteq I \ X \in \text{sets } (Pi_M \ J \ M)$
and K : finite $K \subseteq I \ Y \in \text{sets } (Pi_M \ K \ M)$
and $emb \ I \ J \ X \cap emb \ I \ K \ Y = \{\}$
then have $JK\text{-disj}$: $emb \ (J \cup \ K) \ J \ X \cap emb \ (J \cup \ K) \ K \ Y = \{\}$
by (*intro emb-injective[of J \cup K I - {}]*) (*auto simp: sets.Int prod-emb-Int*)
have $\mu G \ (emb \ I \ J \ X \cup emb \ I \ K \ Y) = \mu G \ (emb \ I \ (J \cup \ K) \ (emb \ (J \cup \ K) \ J \ X \cup emb \ (J \cup \ K) \ K \ Y))$
using $J \ K$ **by** *simp*
also have $\dots = \text{emeasure } (P \ (J \cup \ K)) \ (emb \ (J \cup \ K) \ J \ X \cup emb \ (J \cup \ K) \ K \ Y)$
using $J \ K$ **by** (*simp add: mu-G-spec sets.Un del: prod-emb-Un*)
also have $\dots = \mu G \ (emb \ I \ J \ X) + \mu G \ (emb \ I \ K \ Y)$
using $J \ K \ JK\text{-disj}$ **by** (*simp add: plus-emeasure[symmetric] mu-G-spec emeasure-P sets-P*)
finally show $\mu G \ (emb \ I \ J \ X \cup emb \ I \ K \ Y) = \mu G \ (emb \ I \ J \ X) + \mu G \ (emb \ I \ K \ Y) \ .$
qed

lemma *emeasure-lim:*

assumes JX : finite $J \subseteq I \ X \in \text{sets } (Pi_M \ J \ M)$
assumes *cont*: $\bigwedge J \ X. (\bigwedge i. J \ i \subseteq I) \implies \text{incseq } J \implies (\bigwedge i. \text{finite } (J \ i)) \implies (\bigwedge i.$

$X i \in \text{sets } (PiM (J i) M) \implies$
 $\text{decseq } (\lambda i. \text{emb } I (J i) (X i)) \implies 0 < (\text{INF } i. P (J i) (X i)) \implies (\bigcap i. \text{emb } I (J i) (X i)) \neq \{\}$
shows $\text{emeasure } \text{lim } (\text{emb } I J X) = P J X$
proof –
have $\exists \mu. (\forall s \in \text{generator}. \mu s = \mu G s) \wedge$
 $\text{measure-space } (\text{space } (PiM I M)) (\text{sigma-sets } (\text{space } (PiM I M)) \text{ generator}) \mu$
proof (*rule generator.caratheodory-empty-continuous[OF positive-mu-G additive-mu-G]*)
show $\bigwedge A. A \in \text{generator} \implies \mu G A \neq \infty$
proof (*clarsimp elim!: generator.cases simp: mu-G-spec del: notI*)
fix J **assume** $\text{finite } J \subseteq I$
then interpret $\text{prob-space } P J$ **by** (*rule prob-space-P*)
show $\bigwedge X. X \in \text{sets } (PiM J M) \implies \text{emeasure } (P J) X \neq \text{top}$
by *simp*
qed
next
fix A **assume** $\text{range } A \subseteq \text{generator}$ **and** $\text{decseq } A (\bigcap i. A i) = \{\}$
then have $\forall i. \exists J X. A i = \text{emb } I J X \wedge \text{finite } J \wedge J \subseteq I \wedge X \in \text{sets } (PiM J M)$
unfolding *image-subset-iff generator.simps* **by** *blast*
then obtain $J X$ **where** $A: \bigwedge i. A i = \text{emb } I (J i) (X i)$
and $*$: $\bigwedge i. \text{finite } (J i) \wedge i. J i \subseteq I \wedge i. X i \in \text{sets } (PiM (J i) M)$
by *metis*
have $(\text{INF } i. P (J i) (X i)) = 0$
proof (*rule ccontr*)
assume $\text{INF-P}: (\text{INF } i. P (J i) (X i)) \neq 0$
have $(\bigcap i. \text{emb } I (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i))) \neq \{\}$
proof (*rule cont*)
show $\text{decseq } (\lambda i. \text{emb } I (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i)))$
using $*$ *<decseq A>* **by** (*subst prod-emb-trans*) (*auto simp: A[abs-def]*)
show $0 < (\text{INF } i. P (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i)))$
using $*$ *INF-P* **by** (*subst emeasure-P*) (*auto simp: less-le intro!*)
INF-greatest
show $\text{incseq } (\lambda i. \bigcup n \leq i. J n)$
by (*force simp: incseq-def*)
qed (*insert *, auto*)
with $\langle (\bigcap i. A i) = \{\} \rangle$ **show** *False*
by (*subst (asm) prod-emb-trans*) (*auto simp: A[abs-def]*)
qed
moreover have $(\lambda i. P (J i) (X i)) \longrightarrow (\text{INF } i. P (J i) (X i))$
proof (*intro LIMSEQ-INF antimonoI*)
fix $x y :: \text{nat}$ **assume** $x \leq y$
have $P (J y \cup J x) (\text{emb } (J y \cup J x) (J y) (X y)) \leq P (J y \cup J x) (\text{emb } (J y \cup J x) (J x) (X x))$
using $\langle \text{decseq } A \rangle$ [*THEN decseqD, OF <x≤y>*] $*$
by (*auto simp: A sets-P del: subsetI intro!: emeasure-mono <x ≤ y>*)
 emb-preserve-mono [*of J y ∪ J x I, where X=emb (J y ∪ J x) (J y) (X y)*])
then show $P (J y) (X y) \leq P (J x) (X x)$

```

    using * by (simp add: emeasure-P)
  qed
  ultimately show  $(\lambda i. \mu G (A i)) \longrightarrow 0$ 
    by (auto simp: A[abs-def] mu-G-spec *)
  qed
  then obtain  $\mu$  where eq:  $\forall s \in \text{generator}. \mu s = \mu G s$ 
    and ms: measure-space (space (PiM I M)) (sets (PiM I M))  $\mu$ 
    by (metis sets-PiM-generator)
  show ?thesis
  proof (subst emeasure-extend-measure[OF lim-def])
    show  $A \in \text{generator} \implies \mu A = \mu G A$  for A
      using eq by simp
    show positive (sets lim)  $\mu$  countably-additive (sets lim)  $\mu$ 
      using ms by (auto simp add: measure-space-def)
    show  $(\lambda x. x) \text{ 'generator} \subseteq \text{Pow} (\text{space} (Pi_M I M))$ 
      using generator.space-closed by simp
    show  $\text{emb } I J X \in \text{generator} \mu G (\text{emb } I J X) = \text{emeasure} (P J) X$ 
      using JX by (auto intro: generator.intros simp: mu-G-spec)
  qed
  qed
end

```

```

sublocale product-prob-space  $\subseteq$  projective-family I  $\lambda J. PiM J M M$ 
  unfolding projective-family-def
proof (intro conjI allI impI distr-restrict)
  show  $\bigwedge J. \text{finite } J \implies \text{prob-space} (Pi_M J M)$ 
    by (intro prob-spaceI) (simp add: space-PiM emeasure-PiM emeasure-space-1)
  qed auto

```

Proof due to Ionescu Tulcea.

```

locale Ionescu-Tulcea =
  fixes  $P :: \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow 'a \text{ measure}$  and  $M :: \text{nat} \Rightarrow 'a \text{ measure}$ 
  assumes  $P[\text{measurable}]: \bigwedge i. P i \in \text{measurable} (PiM \{0..<i\} M)$  (subprob-algebra
  ( $M i$ ))
  assumes prob-space-P:  $\bigwedge i x. x \in \text{space} (PiM \{0..<i\} M) \implies \text{prob-space} (P i x)$ 
begin

```

```

lemma non-empty[simp]:  $\text{space} (M i) \neq \{\}$ 
proof (induction i rule: less-induct)
  case (less i)
  then obtain  $x$  where  $\bigwedge j. j < i \implies x j \in \text{space} (M j)$ 
    unfolding ex-in-conv[symmetric] by metis
  then have  $*$ :  $\text{restrict } x \{0..<i\} \in \text{space} (PiM \{0..<i\} M)$ 
    by (auto simp: space-PiM PiE-iff)
  then interpret prob-space P i (restrict x {0..<i})
    by (rule prob-space-P)
  show ?case

```

using *not-empty subprob-measurableD(1)[OF P, OF *]* **by** *simp*
qed

lemma *space-PiM-not-empty[simp]*: *space (PiM UNIV M) ≠ {}*
unfolding *space-PiM-empty-iff* **by** *auto*

lemma *space-P*: *x ∈ space (PiM {0..<n} M) ⇒ space (P n x) = space (M n)*
by (*simp add: P[THEN subprob-measurableD(1)]*)

lemma *sets-P[measurable-cong]*: *x ∈ space (PiM {0..<n} M) ⇒ sets (P n x) = sets (M n)*
by (*simp add: P[THEN subprob-measurableD(2)]*)

definition *eP :: nat ⇒ (nat ⇒ 'a) ⇒ (nat ⇒ 'a) measure where*
eP n ω = distr (P n ω) (PiM {0..<Suc n} M) (fun-upd ω n)

lemma *measurable-eP[measurable]*:
eP n ∈ measurable (PiM {0..<n} M) (subprob-algebra (PiM {0..<Suc n} M))
by (*auto simp: eP-def[abs-def] measurable-split-conv*
intro!: measurable-fun-upd[where J={0..<n}] measurable-distr2[OF - P])

lemma *space-eP*:
x ∈ space (PiM {0..<n} M) ⇒ space (eP n x) = space (PiM {0..<Suc n} M)
by (*simp add: measurable-eP[THEN subprob-measurableD(1)]*)

lemma *sets-eP[measurable]*:
x ∈ space (PiM {0..<n} M) ⇒ sets (eP n x) = sets (PiM {0..<Suc n} M)
by (*simp add: measurable-eP[THEN subprob-measurableD(2)]*)

lemma *prob-space-eP*: *x ∈ space (PiM {0..<n} M) ⇒ prob-space (eP n x)*
unfolding *eP-def*
by (*intro prob-space.prob-space-distr prob-space-P measurable-fun-upd[where J={0..<n}]*)
auto

lemma *nn-integral-eP*:
ω ∈ space (PiM {0..<n} M) ⇒ f ∈ borel-measurable (PiM {0..<Suc n} M)
 \Rightarrow
 $(\int^+ x. f x \partial eP n \omega) = (\int^+ x. f (\omega(n := x)) \partial P n \omega)$
unfolding *eP-def*
by (*subst nn-integral-distr*) (*auto intro!: measurable-fun-upd[where J={0..<n}]*)
simp: space-PiM PiE-iff)

lemma *emeasure-eP*:
assumes *ω[simp]*: *ω ∈ space (PiM {0..<n} M)* **and** *A[measurable]*: *A ∈ sets (PiM {0..<Suc n} M)*
shows *eP n ω A = P n ω ((λx. ω(n := x)) -' A ∩ space (M n))*
using *nn-integral-eP[of ω n indicator A]*
apply (*simp add: sets-eP nn-integral-indicator[symmetric] sets-P del: nn-integral-indicator*)
apply (*subst nn-integral-indicator[symmetric]*)

using measurable-sets[OF measurable-fun-upd[OF - measurable-const[OF ω] measurable-id]
 A, of n]
apply (auto simp add: sets-P atLeastLessThanSuc space-P simp del: nn-integral-indicator
 intro!: nn-integral-cong split: split-indicator)
done

primrec $C :: \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \text{ measure}$ **where**
 $C\ n\ 0\ \omega = \text{return}\ (PiM\ \{0..<n\}\ M)\ \omega$
 $| C\ n\ (Suc\ m)\ \omega = C\ n\ m\ \omega \ggg eP\ (n + m)$

lemma measurable-C[measurable]:
 $C\ n\ m \in \text{measurable}\ (PiM\ \{0..<n\}\ M)\ (\text{subprob-algebra}\ (PiM\ \{0..<n + m\}\ M))$
by (induction m) auto

lemma space-C:
 $x \in \text{space}\ (PiM\ \{0..<n\}\ M) \implies \text{space}\ (C\ n\ m\ x) = \text{space}\ (PiM\ \{0..<n + m\}\ M)$
by (simp add: measurable-C[THEN subprob-measurableD(1)])

lemma sets-C[measurable-cong]:
 $x \in \text{space}\ (PiM\ \{0..<n\}\ M) \implies \text{sets}\ (C\ n\ m\ x) = \text{sets}\ (PiM\ \{0..<n + m\}\ M)$
by (simp add: measurable-C[THEN subprob-measurableD(2)])

lemma prob-space-C: $x \in \text{space}\ (PiM\ \{0..<n\}\ M) \implies \text{prob-space}\ (C\ n\ m\ x)$
proof (induction m)
case (Suc m) **then show** ?case
by (auto intro!: prob-space.prob-space-bind[**where** $S = PiM\ \{0..<Suc\ (n + m)\}\ M$]
 simp: space-C prob-space-eP)
qed (auto intro!: prob-space-return simp: space-PiM)

lemma split-C:
assumes $\omega: \omega \in \text{space}\ (PiM\ \{0..<n\}\ M)$ **shows** $(C\ n\ m\ \omega \ggg C\ (n + m)\ l) = C\ n\ (m + l)\ \omega$
proof (induction l)
case 0
with ω **show** ?case
by (simp add: bind-return-distr' prob-space-C[THEN prob-space.not-empty]
 distr-cong[OF refl sets-C[symmetric, OF ω]])
next
case (Suc l) **with** ω **show** ?case
by (simp add: bind-assoc[symmetric, OF - measurable-eP]) (simp add: ac-simps)
qed

lemma nn-integral-C:
assumes $m \leq m'$ **and** f [measurable]: $f \in \text{borel-measurable}\ (PiM\ \{0..<n+m\}\ M)$
proof

and nonneg: $\bigwedge x. x \in \text{space } (PiM \{0..<n+m\} M) \implies 0 \leq f x$
and x : $x \in \text{space } (PiM \{0..<n\} M)$
shows $(\int^+ x. f x \partial C n m x) = (\int^+ x. f (\text{restrict } x \{0..<n+m\}) \partial C n m' x)$
using $\langle m \leq m' \rangle$
proof (induction rule: dec-induct)
case (step i)
let $?E = \lambda x. f (\text{restrict } x \{0..<n+m\})$ **and** $?C = \lambda i f. \int^+ x. f x \partial C n i x$
from $\langle m \leq i \rangle x$ **have** $?C i ?E = ?C (\text{Suc } i) ?E$
by (auto simp: nn-integral-bind[**where** $B = PiM \{0..< \text{Suc } (n+i)\} M$] space-C
nn-integral-eP
intro!: nn-integral-cong)
(simp add: space-PiM PiE-iff nonneg prob-space.emeasure-space-1[OF
prob-space-P])
with step show ?case **by** (simp del: restrict-apply)
qed (auto simp: space-PiM space-C[OF x] simp del: restrict-apply intro!: nn-integral-cong)

lemma *emeasure-C*:

assumes $m \leq m'$ **and** $A[\text{measurable}]$: $A \in \text{sets } (PiM \{0..<n+m\} M)$ **and**
 $[simp]$: $x \in \text{space } (PiM \{0..<n\} M)$
shows $\text{emeasure } (C n m' x) (\text{prod-emb } \{0..<n+m\} M \{0..<n+m\} A) =$
 $\text{emeasure } (C n m x) A$
using *assms*
by (subst (1 2) nn-integral-indicator[symmetric])
(auto intro!: nn-integral-cong split: split-indicator simp del: nn-integral-indicator
simp: nn-integral-C[of $m m' - n$] prod-emb-iff space-PiM PiE-iff sets-C
space-C)

lemma *distr-C*:

assumes $m \leq m'$ **and** $[simp]$: $x \in \text{space } (PiM \{0..<n\} M)$
shows $C n m x = \text{distr } (C n m' x) (PiM \{0..<n+m\} M) (\lambda x. \text{restrict } x$
 $\{0..<n+m\})$
proof (rule *measure-eqI*)
fix A **assume** $A \in \text{sets } (C n m x)$
with $\langle m \leq m' \rangle$ **show** $\text{emeasure } (C n m x) A = \text{emeasure } (\text{distr } (C n m' x) (PiM$
 $\{0..<n+m\} M) (\lambda x. \text{restrict } x \{0..<n+m\})) A$
by (subst *emeasure-C*[symmetric, OF $\langle m \leq m' \rangle$] (auto intro!: *emeasure-distr-restrict*[symmetric]
simp: sets-C)
qed (*simp* add: sets-C)

definition *up-to* :: $\text{nat set} \Rightarrow \text{nat}$ **where**

up-to $J = (\text{LEAST } n. \forall i \geq n. i \notin J)$

lemma *up-to-less*: $\text{finite } J \implies i \in J \implies i < \text{up-to } J$

unfolding *up-to-def*

by (rule *LeastI2*[of - *Suc* (*Max* J))] (auto simp: *Suc-le-eq not-le*[symmetric])

lemma *up-to-iff*: $\text{finite } J \implies \text{up-to } J \leq n \iff (\forall i \in J. i < n)$

proof *safe*

show $\text{finite } J \implies \text{up-to } J \leq n \implies i \in J \implies i < n$ **for** i

using *up-to-less*[of J i] **by** *auto*
qed (*auto simp: up-to-def intro!: Least-le*)

lemma *up-to-iff-Ico*: *finite J* \implies *up-to J* $\leq n \iff J \subseteq \{0..<n\}$
by (*auto simp: up-to-iff*)

lemma *up-to*: *finite J* $\implies J \subseteq \{0..< \text{up-to } J\}$
by (*auto simp: up-to-less*)

lemma *up-to-mono*: $J \subseteq H \implies \text{finite } H \implies \text{up-to } J \leq \text{up-to } H$
by (*auto simp add: up-to-iff finite-subset up-to-less*)

definition *CI* :: *nat set* \Rightarrow (*nat* \Rightarrow 'a) *measure* **where**
CI J = distr (C 0 (up-to J) (λx . undefined)) (PiM J M) (λf . restrict f J)

sublocale *PF*: *projective-family UNIV CI*

unfolding *projective-family-def*

proof *safe*

show *finite J* \implies *prob-space (CI J)* **for** J

using *up-to*[of J] **unfolding** *CI-def*

by (*intro prob-space.prob-space-distr prob-space-C measurable-restrict*) *auto*

note *measurable-cong-sets*[*OF sets-C, simp*]

have [*simp*]: $J \subseteq H \implies (\lambda f$. *restrict f J*) \in *measurable (PiM H M) (PiM J M)*

for $H J$

by (*auto intro!: measurable-restrict*)

show $J \subseteq H \implies \text{finite } H \implies \text{CI } J = \text{distr (CI } H) (PiM J M) (\lambda f$. *restrict f J*) **for** $J H$

by (*simp add: CI-def distr-C*[*OF up-to-mono*[of $J H$]] *up-to up-to-mono distr-distr comp-def*

inf.absorb2 finite-subset)

qed

lemma *emeasure-CI'*:

finite J $\implies X \in \text{sets (PiM J M)} \implies \text{CI } J X = \text{C 0 (up-to J) (λ -. undefined)}$
(*PF.emb* $\{0..<\text{up-to } J\} J X$)

unfolding *CI-def* **using** *up-to*[of J] **by** (*rule emeasure-distr-restrict*) (*auto simp: sets-C*)

lemma *emeasure-CI*:

$J \subseteq \{0..<n\} \implies X \in \text{sets (PiM J M)} \implies \text{CI } J X = \text{C 0 } n$ (λ -. *undefined*)
(*PF.emb* $\{0..<n\} J X$)

apply (*subst emeasure-CI'*, *simp-all add: finite-subset*)

apply (*subst emeasure-C*[*symmetric, of up-to J n*])

apply (*auto simp: finite-subset up-to-iff-Ico up-to-less*)

apply (*subst prod-emb-trans*)

apply (*auto simp: up-to-less finite-subset up-to-iff-Ico*)

done

lemma *lim*:

assumes J : finite J **and** X : $X \in \text{sets } (PiM J M)$
shows $\text{emeasure } PF.\text{lim } (PF.\text{emb } UNIV J X) = \text{emeasure } (CI J) X$
proof (rule $PF.\text{emeasure-lim}[OF J \text{ subset-}UNIV X]$)
fix $J X'$ **assume** $J[\text{simp}]$: $\bigwedge i. \text{finite } (J i)$ **and** $X'[\text{measurable}]$: $\bigwedge i. X' i \in \text{sets } (PiM (J i) M)$
and dec : $\text{decseq } (\lambda i. PF.\text{emb } UNIV (J i) (X' i))$
define X **where** $X n =$
 $(\bigcap i \in \{i. J i \subseteq \{0..<n\}\}. PF.\text{emb } \{0..<n\} (J i) (X' i)) \cap \text{space } (PiM \{0..<n\} M)$ **for** n

have $\text{sets-}X[\text{measurable}]$: $X n \in \text{sets } (PiM \{0..<n\} M)$ **for** n
by (cases $\{i. J i \subseteq \{0..<n\}\} = \{\}$)
(auto $\text{simp-}all \text{ add: } X\text{-def}$, auto intro! : $\text{sets.countable-INT' sets.Int}$)

have $\text{dec-}X$: $n \leq m \implies X m \subseteq PF.\text{emb } \{0..<m\} \{0..<n\} (X n)$ **for** $n m$
unfolding $X\text{-def}$ **using** $\text{ivl-subset}[of 0 n 0 m]$
by (cases $\{i. J i \subseteq \{0..<n\}\} = \{\}$)
(auto $\text{simp add: prod-emb-Int prod-emb-}PiE \text{ space-}PiM \text{ simp del: ivl-subset}$)

have $\text{dec-}X'$: $PF.\text{emb } \{0..<n\} (J j) (X' j) \subseteq PF.\text{emb } \{0..<n\} (J i) (X' i)$
if $[\text{simp}]$: $J i \subseteq \{0..<n\} J j \subseteq \{0..<n\} i \leq j$ **for** $n i j$
by (rule $PF.\text{emb-preserve-mono}[of \{0..<n\} UNIV]$) (auto $\text{del: subsetI intro: dec}[THEN \text{antimonoD}]$)

assume $0 < (INF i. CI (J i) (X' i))$
also **have** $\dots \leq (INF i. C 0 i (\lambda x. \text{undefined}) (X i))$
proof (intro $INF\text{-greatest}$)
fix n
interpret C : $\text{prob-space } C 0 n (\lambda x. \text{undefined})$
by (rule $\text{prob-space-}C$) simp
show $(INF i. CI (J i) (X' i)) \leq C 0 n (\lambda x. \text{undefined}) (X n)$
proof *cases*
assume $\{i. J i \subseteq \{0..<n\}\} = \{\}$ **with** $C.\text{emeasure-space-1}$ **show** *?thesis*
by (auto $\text{simp add: } X\text{-def space-}C \text{ intro!}$: $INF\text{-lower2}[of 0] \text{ prob-space.measure-le-1 } PF.\text{prob-space-}P$)
next
assume $*$: $\{i. J i \subseteq \{0..<n\}\} \neq \{\}$
have $(INF i. CI (J i) (X' i)) \leq$
 $(INF i:\{i. J i \subseteq \{0..<n\}\}. C 0 n (\lambda-. \text{undefined}) (PF.\text{emb } \{0..<n\} (J i) (X' i)))$
by (intro $INF\text{-superset-mono}$) (auto $\text{simp: emeasure-}CI$)
also **have** $\dots = C 0 n (\lambda-. \text{undefined}) (\bigcap i \in \{i. J i \subseteq \{0..<n\}\}. (PF.\text{emb } \{0..<n\} (J i) (X' i)))$
using $*$ **by** (intro $\text{emeasure-INT-decseq-subset[symmetric]}$) (auto intro! : $\text{dec-}X' \text{ del: subsetI simp: sets-}C$)
also **have** $\dots = C 0 n (\lambda-. \text{undefined}) (X n)$
using $*$ **by** (auto $\text{simp add: } X\text{-def INT-extend-simps}$)
finally **show** $(INF i. CI (J i) (X' i)) \leq C 0 n (\lambda-. \text{undefined}) (X n)$.

```

qed
qed
finally have  $pos: 0 < (INF\ i.\ C\ 0\ i\ (\lambda x.\ undefined)\ (X\ i))$  .
from  $less-INF-D[OF\ this,\ of\ 0]$  have  $X\ 0 \neq \{\}$ 
by auto

{ fix  $\omega\ n$  assume  $\omega: \omega \in space\ (PiM\ \{0..<n\}\ M)$ 
let  $?C = \lambda i.\ emeasure\ (C\ n\ i\ \omega)\ (X\ (n + i))$ 
let  $?C' = \lambda i\ x.\ emeasure\ (C\ (Suc\ n)\ i\ (\omega(n:=x)))\ (X\ (Suc\ n + i))$ 
have  $M: \bigwedge i.\ ?C'\ i \in borel-measurable\ (P\ n\ \omega)$ 
using  $\omega[measurable,\ simp]$  measurable-fun-upd[where\ J={0..<n}] by measurable\ auto

assume  $0 < (INF\ i.\ ?C\ i)$ 
also have  $\dots \leq (INF\ i.\ emeasure\ (C\ n\ (1 + i)\ \omega)\ (X\ (n + (1 + i))))$ 
by (intro\ INF-greatest\ INF-lower)\ auto
also have  $\dots = (INF\ i.\ \int^{+x}.\ ?C'\ i\ x\ \partial P\ n\ \omega)$ 
using  $\omega\ measurable-C[of\ Suc\ n]$ 
apply (intro\ INF-cong\ refl)
apply (subst\ split-C[symmetric,\ OF\ \omega])
apply (subst\ emeasure-bind[OF\ -\ -\ sets-X])
apply (simp-all\ del:\ C.simps\ add:\ space-C)
apply measurable
apply simp
apply (simp\ add:\ bind-return[OF\ measurable-eP]\ nn-integral-eP)
done
also have  $\dots = (\int^{+x}.\ (INF\ i.\ ?C'\ i\ x)\ \partial P\ n\ \omega)$ 
proof (rule\ nn-integral-monotone-convergence-INF-AE[symmetric])
have  $(\int^{+x}.\ ?C'\ 0\ x\ \partial P\ n\ \omega) \leq (\int^{+x}.\ 1\ \partial P\ n\ \omega)$ 
by (intro\ nn-integral-mono)\ (auto\ split:\ split-indicator)
also have  $\dots < \infty$ 
using prob-space-P[OF\ \omega,\ THEN\ prob-space.emeasure-space-1] by simp
finally show  $(\int^{+x}.\ ?C'\ 0\ x\ \partial P\ n\ \omega) < \infty$  .
next
show AE\ x\ in\ P\ n\ \omega.\ ?C'\ (Suc\ i)\ x \leq ?C'\ i\ x for  $i$ 
proof (rule\ AE-I2)
fix  $x$  assume  $x \in space\ (P\ n\ \omega)$ 
with  $\omega$  have  $\omega': \omega(n := x) \in space\ (PiM\ \{0..<Suc\ n\}\ M)$ 
by (auto\ simp:\ space-P[OF\ \omega]\ space-PiM\ PiE-iff\ extensional-def)
with  $\omega$  show  $?C'\ (Suc\ i)\ x \leq ?C'\ i\ x$ 
apply (subst\ emeasure-C[symmetric,\ of\ i\ Suc\ i])
apply (auto\ intro!\: emeasure-mono[OF\ dec-X]\ del:\ subsetI\ simp:\ sets-C\ space-P)
apply (subst\ sets-bind[OF\ sets-eP])
apply (simp-all\ add:\ space-C\ space-P)
done
qed
qed fact
finally have  $(\int^{+x}.\ (INF\ i.\ ?C'\ i\ x)\ \partial P\ n\ \omega) \neq 0$ 

```

```

    by simp
  with M have  $\exists_F x$  in ae-filter (P n  $\omega$ ).  $0 < (INF i. ?C' i x)$ 
    by (subst (asm) nn-integral-0-iff-AE)
      (auto intro!: borel-measurable-INF simp: Filter.not-eventually not-le
zero-less-iff-neq-zero)
  then have  $\exists_F x$  in ae-filter (P n  $\omega$ ).  $x \in \text{space } (M n) \wedge 0 < (INF i. ?C' i x)$ 
    by (rule frequently-mp[rotated]) (auto simp: space-P  $\omega$ )
  then obtain x where  $x \in \text{space } (M n) \wedge 0 < (INF i. ?C' i x)$ 
    by (auto dest: frequently-ex)
  from this(2)[THEN less-INF-D, of 0] this(2)
  have  $\exists x. \omega(n := x) \in X (Suc n) \wedge 0 < (INF i. ?C' i x)$ 
    by (intro exI[of - x]) (simp split: split-indicator-asm) }
  note step = this

let ? $\omega$  =  $\lambda \omega n x. (\text{restrict } \omega \{0..<n\})(n := x)$ 
let ?L =  $\lambda \omega n r. INF i. \text{emeasure } (C (Suc n) i (\omega \omega n r)) (X (Suc n + i))$ 
have *:  $(\bigwedge i. i < n \implies ?\omega \omega i (\omega i) \in X (Suc i)) \implies$ 
   $\text{restrict } \omega \{0..<n\} \in \text{space } (Pi_M \{0..<n\} M)$  for  $\omega n$ 
  using sets.sets-into-space[OF sets-X, of n]
  by (cases n) (auto simp: atLeastLessThanSuc restrict-def[of - {}])
have  $\exists \omega. \forall n. ?\omega \omega n (\omega n) \in X (Suc n) \wedge 0 < ?L \omega n (\omega n)$ 
proof (rule dependent-wellorder-choice)
  fix n  $\omega$  assume IH:  $\bigwedge i. i < n \implies ?\omega \omega i (\omega i) \in X (Suc i) \wedge 0 < ?L \omega i$ 
  ( $\omega i$ )
  show  $\exists r. ?\omega \omega n r \in X (Suc n) \wedge 0 < ?L \omega n r$ 
  proof (rule step)
    show  $\text{restrict } \omega \{0..<n\} \in \text{space } (Pi_M \{0..<n\} M)$ 
      using IH[THEN conjunct1] by (rule *)
    show  $0 < (INF i. \text{emeasure } (C n i (\text{restrict } \omega \{0..<n\}))) (X (n + i))$ 
      proof (cases n)
        case 0 with pos show ?thesis
          by (simp add: CI-def restrict-def)
        next
          case (Suc i) then show ?thesis
            using IH[of i, THEN conjunct2] by (simp add: atLeastLessThanSuc)
      qed
    qed
  qed (simp cong: restrict-cong)
then obtain  $\omega$  where  $\omega: \bigwedge n. ?\omega \omega n (\omega n) \in X (Suc n)$ 
  by auto
from this[THEN *] have  $\omega$ -space:  $\omega \in \text{space } (Pi_M UNIV M)$ 
  by (auto simp: space-PiM PiE-iff Ball-def)
have *:  $\omega \in PF.emb UNIV \{0..<n\} (X n)$  for n
proof (cases n)
  case 0 with  $\omega$ -space  $\langle X 0 \neq \{\} \rangle$  sets.sets-into-space[OF sets-X, of 0] show
  ?thesis
    by (auto simp add: space-PiM prod-emb-def restrict-def PiE-iff)
  next
    case (Suc i) then show ?thesis

```

```

using  $\omega$ [of  $i$ ]  $\omega$ -space by (auto simp: prod-emb-def space-PiM PiE-iff atLeast-
LessThanSuc)
qed
have **:  $\{i. J i \subseteq \{0..<up-to (J n)\}\} \neq \{\}$  for  $n$ 
by (auto intro!: exI[of -  $n$ ] up-to  $J$ )
have  $\omega \in PF.emb UNIV (J n) (X' n)$  for  $n$ 
using *[of up-to ( $J n$ )] up-to[of  $J n$ ] by (simp add: X-def prod-emb-Int prod-emb-INT[OF
**])
then show  $(\bigcap i. PF.emb UNIV (J i) (X' i)) \neq \{\}$ 
by auto
qed

```

```

lemma distr-lim: assumes  $J[simp]: finite J$  shows  $distr PF.lim (PiM J M) (\lambda x. restrict x J) = CI J$ 
apply (rule measure-egI)
apply (simp add: CI-def)
apply (simp add: emeasure-distr measurable-cong-sets[OF PF.sets-lim] lim[symmetric]
prod-emb-def space-PiM)
done

```

end

```

lemma (in product-prob-space) emeasure-lim-emb:
assumes *:  $finite J J \subseteq I X \in sets (PiM J M)$ 
shows  $emeasure lim (emb I J X) = emeasure (Pi_M J M) X$ 
proof (rule emeasure-lim[OF *], goal-cases)
case (1  $J X$ )

```

```

have  $\exists Q. (\forall i. sets Q = PiM (\bigcup i. J i) M \wedge distr Q (PiM (J i) M) (\lambda x. restrict x (J i)) = Pi_M (J i) M)$ 
proof cases
assume  $finite (\bigcup i. J i)$ 
then have  $distr (PiM (\bigcup i. J i) M) (Pi_M (J i) M) (\lambda x. restrict x (J i)) = Pi_M (J i) M$  for  $i$ 
by (intro distr-restrict[symmetric]) auto
then show ?thesis
by auto
next
assume  $inf: infinite (\bigcup i. J i)$ 
moreover have  $count: countable (\bigcup i. J i)$ 
using 1(3) by (auto intro: countable-finite)
define  $f$  where  $f = from-nat-into (\bigcup i. J i)$ 
define  $t$  where  $t = to-nat-on (\bigcup i. J i)$ 
have  $ft[simp]: x \in J i \implies f (t x) = x$  for  $x i$ 
unfolding  $f$ -def  $t$ -def using  $inf count$  by (intro from-nat-into-to-nat-on) auto
have  $tf[simp]: t (f i) = i$  for  $i$ 
unfolding  $t$ -def  $f$ -def by (intro to-nat-on-from-nat-into-infinite  $inf count$ )
have  $inj-t: inj-on t (\bigcup i. J i)$ 
using  $count$  by (auto simp:  $t$ -def)

```

```

then have inj-t-J: inj-on t (J i) for i
  by (rule subset-inj-on) auto
interpret IT: Ionescu-Tulcea  $\lambda i \omega. M (f i) \lambda i. M (f i)$ 
  by standard auto
interpret Mf: product-prob-space  $\lambda x. M (f x) UNIV$ 
  by standard
have C-eq-PiM: IT.C 0 n ( $\lambda \cdot$ . undefined) = PiM {0.. $n$ } ( $\lambda x. M (f x)$ ) for n
proof (induction n)
  case 0 then show ?case
    by (auto simp: PiM-empty intro!: measure-eqI dest!: subset-singletonD)
  next
    case (Suc n) then show ?case
      apply (auto intro!: measure-eqI simp: sets-bind[OF IT.sets-eP] emeasure-bind[OF
- IT.measurable-eP])
      apply (auto simp: Mf.product-nn-integral-insert nn-integral-indicator[symmetric]
atLeastLessThanSuc IT.emeasure-eP space-PiM
split: split-indicator simp del: nn-integral-indicator intro!:
nn-integral-cong)
      done
      qed
have CI-eq-PiM: IT.CI X = PiM X ( $\lambda x. M (f x)$ ) if X: finite X for X
by (auto simp: IT.up-to-less X IT.CI-def C-eq-PiM intro!: Mf.distr-restrict[symmetric])

let ?Q = distr IT.PF.lim (PiM ( $\bigcup i. J i$ ) M) ( $\lambda \omega. \lambda x \in \bigcup i. J i. \omega (t x)$ )
{ fix i
  have distr ?Q (PiM (J i) M) ( $\lambda x. restrict x (J i)$ ) =
    distr IT.PF.lim (PiM (J i) M) (( $\lambda \omega. \lambda n \in J i. \omega (t n)$ )  $\circ$  ( $\lambda \omega. restrict \omega (t'J$ 
i)))
  proof (subst distr-distr)
    have ( $\lambda \omega. \omega (t x) \in measurable (PiM UNIV (\lambda x. M (f x))) (M x)$ ) if x: x
 $\in J i$  for x i
    using measurable-component-singleton[of t x UNIV  $\lambda x. M (f x)$ ] unfolding
ft[OF x] by simp
    then show ( $\lambda \omega. \lambda x \in \bigcup i. J i. \omega (t x) \in measurable IT.PF.lim (PiM$ 
(UNION UNIV J) M)
    by (auto intro!: measurable-restrict simp: measurable-cong-sets[OF IT.PF.sets-lim
refl])
    qed (auto intro!: distr-cong measurable-restrict measurable-component-singleton)
    also have ... = distr (distr IT.PF.lim (PiM (t'J i) ( $\lambda x. M (f x)$ ))) ( $\lambda \omega. restrict \omega (t'J i)$ )
(PiM (J i) M) ( $\lambda \omega. \lambda n \in J i. \omega (t n)$ )
    proof (intro distr-distr[symmetric])
      have ( $\lambda \omega. \omega (t x) \in measurable (PiM (t'J i) (\lambda x. M (f x))) (M x)$ ) if x: x
 $\in J i$  for x
      using measurable-component-singleton[of t x t'J i  $\lambda x. M (f x)$ ] x unfolding
ft[OF x] by auto
      then show ( $\lambda \omega. \lambda n \in J i. \omega (t n) \in measurable (PiM (t'J i) (\lambda x. M (f$ 
x))) (PiM (J i) M)
      by (auto intro!: measurable-restrict)
    qed (auto intro!: measurable-restrict simp: measurable-cong-sets[OF IT.PF.sets-lim

```

```

refl])
  also have ... = distr (PiM (t'J i) (λx. M (f x))) (PiM (J i) M) (λω. λn∈J
i. ω (t n))
    using ⟨finite (J i)⟩ by (subst IT.distr-lim) (auto simp: CI-eq-PiM)
    also have ... = PiM (J i) M
    using Mf.distr-reorder[of t J i] by (simp add: 1 inj-t-J cong: PiM-cong)
    finally have distr ?Q (PiM (J i) M) (λx. restrict x (J i)) = PiM (J i) M .
}
  then show ∃ Q. ∀ i. sets Q = PiM (∪ i. J i) M ∧ distr Q (PiM (J i) M) (λx.
restrict x (J i)) = PiM (J i) M
    by (intro exI[of - ?Q]) auto
  qed
  then obtain Q where sets-Q: sets Q = PiM (∪ i. J i) M
    and Q: ∧ i. distr Q (PiM (J i) M) (λx. restrict x (J i)) = PiM (J i) M by
blast

from 1 interpret Q: prob-space Q
  by (intro prob-space-distrD[of λx. restrict x (J 0) Q PiM (J 0) M])
    (auto simp: Q measurable-cong-sets[OF sets-Q]
      intro!: prob-space-P measurable-restrict measurable-component-singleton)

have 0 < (INF i. emeasure (PiM (J i) M) (X i)) by fact
also have ... = (INF i. emeasure Q (emb (∪ i. J i) (J i) (X i)))
  by (simp add: emeasure-distr-restrict[OF - sets-Q 1(4), symmetric] SUP-upper
Q)
also have ... = emeasure Q (∩ i. emb (∪ i. J i) (J i) (X i))
proof (rule INF-emeasure-decseq)
  from 1 show decseq (λn. emb (∪ i. J i) (J n) (X n))
  by (intro antimonoI emb-preserve-mono[where X=emb (∪ i. J i) (J n) (X
n) and L=I and J=∪ i. J i for n]
    measurable-prod-emb)
    (auto simp: SUP-least SUP-upper antimono-def)
qed (insert 1, auto simp: sets-Q)
finally have (∩ i. emb (∪ i. J i) (J i) (X i)) ≠ {}
  by auto
moreover have (∩ i. emb I (J i) (X i)) = {} ⇒ (∩ i. emb (∪ i. J i) (J i) (X
i)) = {}
  using 1 by (intro emb-injective[of ∪ i. J i I - {}] sets.countable-INT) (auto
simp: SUP-least SUP-upper)
ultimately show ?case by auto
qed

end

```

11 Infinite Product Measure

```

theory Infinite-Product-Measure
  imports Probability-Measure Projective-Family
begin

```

lemma (in *product-prob-space*) *distr-PiM-restrict-finite*:

assumes *finite J J ⊆ I*
shows $\text{distr } (PiM I M) (PiM J M) (\lambda x. \text{restrict } x J) = PiM J M$
proof (*rule PiM-eqI*)
fix X **assume** $X: \bigwedge i. i \in J \implies X i \in \text{sets } (M i)$
{ fix $J X$ **assume** $J: J \neq \{\} \vee I = \{\}$ *finite J J ⊆ I* **and** $X: \bigwedge i. i \in J \implies X i \in \text{sets } (M i)$
have $\text{emeasure } (PiM I M) (\text{emb } I J (Pi_E J X)) = (\prod_{i \in J}. M i (X i))$
proof (*subst emeasure-extend-measure-Pair[OF PiM-def, where $\mu' = \text{lim}$], goal-cases*)
case 1 then show ?case
by (*simp add: M.emeasure-space-1 emeasure-PiM Pi-iff sets-PiM-I-finite emeasure-lim-emb*)
next
case ($2 J X$)
then have $\text{emb } I J (Pi_E J X) \in \text{sets } (PiM I M)$
by (*intro measurable-prod-emb sets-PiM-I-finite*) *auto*
from this [*THEN sets.sets-into-space*] **show** ?case
by (*simp add: space-PiM*)
qed (*insert assms J X, simp-all del: sets-lim add: M.emeasure-space-1 sets-lim[symmetric] emeasure-countably-additive emeasure-positive*) }
note $*$ = *this*

have $\text{emeasure } (PiM I M) (\text{emb } I J (Pi_E J X)) = (\prod_{i \in J}. M i (X i))$
proof (*cases J ≠ {} ∨ I = {}*)
case False
then obtain i **where** $i: J = \{i\} i \in I$ **by** *auto*
then have $\text{emb } I \{i\} \{\lambda x. \text{undefined}\} = \text{emb } I \{i\} (\Pi_E i \in \{i\}. \text{space } (M i))$
by (*auto simp: space-PiM prod-emb-def*)
with i **show** ?thesis
by (*simp add: * M.emeasure-space-1*)
next
case True
then show ?thesis
by (*simp add: *[OF - assms X]*)
qed
with *assms* **show** $\text{emeasure } (\text{distr } (PiM I M) (PiM J M) (\lambda x. \text{restrict } x J)) (Pi_E J X) = (\prod_{i \in J}. \text{emeasure } (M i) (X i))$
by (*subst emeasure-distr-restrict[OF - refl] (auto intro!: sets-PiM-I-finite X)*)
qed (*insert assms, auto*)

lemma (in *product-prob-space*) *emeasure-PiM-emb'*:

$J \subseteq I \implies \text{finite } J \implies X \in \text{sets } (PiM J M) \implies \text{emeasure } (PiM I M) (\text{emb } I J X) = PiM J M X$
by (*subst distr-PiM-restrict-finite[symmetric, of J] (auto intro!: emeasure-distr-restrict[symmetric])*)

lemma (in *product-prob-space*) *emeasure-PiM-emb*:

$J \subseteq I \implies \text{finite } J \implies (\bigwedge i. i \in J \implies X\ i \in \text{sets } (M\ i)) \implies$
 $\text{emeasure } (Pi_M\ I\ M) (\text{emb } I\ J\ (Pi_E\ J\ X)) = (\prod_{i \in J}. \text{emeasure } (M\ i) (X\ i))$
by (*subst emeasure-PiM-emb'*) (*auto intro!*: *emeasure-PiM*)

sublocale *product-prob-space* \subseteq *P?*: *prob-space* $Pi_M\ I\ M$

proof

have $*$: $\text{emb } I\ \{\}\ \{\lambda x. \text{undefined}\} = \text{space } (Pi_M\ I\ M)$
by (*auto simp*: *prod-emb-def space-PiM*)
show $\text{emeasure } (Pi_M\ I\ M) (\text{space } (Pi_M\ I\ M)) = 1$
using *emeasure-PiM-emb*[*of* $\{\}\ \lambda-. \{\}$] **by** (*simp add*: $*$)

qed

lemma *prob-space-PiM*:

assumes M : $\bigwedge i. i \in I \implies \text{prob-space } (M\ i)$ **shows** *prob-space* $(Pi_M\ I\ M)$

proof –

let $?M = \lambda i. \text{if } i \in I \text{ then } M\ i \text{ else count-space } \{\text{undefined}\}$
interpret M' : *prob-space* $?M\ i$ **for** i
using M **by** (*cases* $i \in I$) (*auto intro!*: *prob-spaceI*)
interpret *product-prob-space* $?M\ I$
by *unfold-locales*
have *prob-space* $(\prod_M\ i \in I. ?M\ i)$
by *unfold-locales*
also have $(\prod_M\ i \in I. ?M\ i) = (\prod_M\ i \in I. M\ i)$
by (*intro PiM-cong*) *auto*
finally show *thesis* .

qed

lemma (in *product-prob-space*) *emeasure-PiM-Collect*:

assumes X : $J \subseteq I$ *finite* J $\bigwedge i. i \in J \implies X\ i \in \text{sets } (M\ i)$
shows $\text{emeasure } (Pi_M\ I\ M) \{x \in \text{space } (Pi_M\ I\ M). \forall i \in J. x\ i \in X\ i\} = (\prod_{i \in J}. \text{emeasure } (M\ i) (X\ i))$

proof –

have $\{x \in \text{space } (Pi_M\ I\ M). \forall i \in J. x\ i \in X\ i\} = \text{emb } I\ J\ (Pi_E\ J\ X)$
unfolding *prod-emb-def* **using** *assms* **by** (*auto simp*: *space-PiM Pi-iff*)
with *emeasure-PiM-emb*[*OF* *assms*] **show** *thesis* **by** *simp*

qed

lemma (in *product-prob-space*) *emeasure-PiM-Collect-single*:

assumes X : $i \in I$ $A \in \text{sets } (M\ i)$
shows $\text{emeasure } (Pi_M\ I\ M) \{x \in \text{space } (Pi_M\ I\ M). x\ i \in A\} = \text{emeasure } (M\ i)$

A

using *emeasure-PiM-Collect*[*of* $\{i\}\ \lambda i. A$] *assms*
by *simp*

lemma (in *product-prob-space*) *measure-PiM-emb*:

assumes $J \subseteq I$ *finite* J $\bigwedge i. i \in J \implies X\ i \in \text{sets } (M\ i)$
shows $\text{measure } (Pi_M\ I\ M) (\text{emb } I\ J\ (Pi_E\ J\ X)) = (\prod_{i \in J}. \text{measure } (M\ i) (X\ i))$

i)

using *emeasure-PiM-emb*[*OF assms*]
unfolding *emeasure-eq-measure* *M.emeasure-eq-measure*
by (*simp add: prod-ennreal measure-nonneg prod-nonneg*)

lemma *sets-Collect-single'*:

$i \in I \implies \{x \in \text{space } (M \ i). \ P \ x\} \in \text{sets } (M \ i) \implies \{x \in \text{space } (PiM \ I \ M). \ P \ (x \ i)\} \in \text{sets } (PiM \ I \ M)$

using *sets-Collect-single*[*of i I {x ∈ space (M i). P x} M*]
by (*simp add: space-PiM PiE-iff cong: conj-cong*)

lemma (**in** *finite-product-prob-space*) *finite-measure-PiM-emb*:

$(\bigwedge i. \ i \in I \implies A \ i \in \text{sets } (M \ i)) \implies \text{measure } (PiM \ I \ M) \ (PiE \ I \ A) = (\prod_{i \in I}. \text{measure } (M \ i) \ (A \ i))$

using *measure-PiM-emb*[*of I A finite-index prod-emb-PiE-same-index*][*OF sets.sets-into-space, of I A M*]
by *auto*

lemma (**in** *product-prob-space*) *PiM-component*:

assumes $i \in I$

shows $\text{distr } (PiM \ I \ M) \ (M \ i) \ (\lambda \omega. \ \omega \ i) = M \ i$

proof (*rule measure-eqI*[*symmetric*])

fix A **assume** $A \in \text{sets } (M \ i)$

moreover have $(\lambda \omega. \ \omega \ i) \ -' \ A \cap \text{space } (PiM \ I \ M) = \{x \in \text{space } (PiM \ I \ M). \ x \ i \in A\}$

by *auto*

ultimately show $\text{emeasure } (M \ i) \ A = \text{emeasure } (\text{distr } (PiM \ I \ M) \ (M \ i) \ (\lambda \omega. \ \omega \ i)) \ A$

by (*auto simp: <i>i ∈ I</i> emeasure-distr measurable-component-singleton emeasure-PiM-Collect-single*)
qed *simp*

lemma (**in** *product-prob-space*) *PiM-eq*:

assumes $M' : \text{sets } M' = \text{sets } (PiM \ I \ M)$

assumes *eq*: $\bigwedge J \ F. \ \text{finite } J \implies J \subseteq I \implies (\bigwedge j. \ j \in J \implies F \ j \in \text{sets } (M \ j))$

\implies

$\text{emeasure } M' \ (\text{prod-emb } I \ M \ J \ (\Pi_E \ j \in J. \ F \ j)) = (\prod_{j \in J}. \ \text{emeasure } (M \ j) \ (F \ j))$

shows $M' = (PiM \ I \ M)$

proof (*rule measure-eqI-PiM-infinite*[*symmetric, OF refl M'*])

show *finite-measure* $(PiM \ I \ M)$

by *standard* (*simp add: P.emeasure-space-1*)

qed (*simp add: eq emeasure-PiM-emb*)

lemma (**in** *product-prob-space*) *AE-component*: $i \in I \implies \text{AE } x \ \text{in } M \ i. \ P \ x \implies \text{AE } x \ \text{in } PiM \ I \ M. \ P \ (x \ i)$

apply (*rule AE-distrD*[*of λω. ω i PiM I M M i P*])

apply *simp*

apply (*subst PiM-component*)

apply *simp-all*

done

lemma *emeasure-PiM-emb*:

assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M\ i)$
assumes $J: J \subseteq I$ *finite* J **and** $A: \bigwedge i. i \in J \implies A\ i \in \text{sets } (M\ i)$
shows $\text{emeasure } (Pi_M\ I\ M) (\text{prod-emb } I\ M\ J\ (Pi_E\ J\ A)) = (\prod_{i \in J}. \text{emeasure } (M\ i) (A\ i))$
proof –
let $?M = \lambda i. \text{if } i \in I \text{ then } M\ i \text{ else count-space } \{\text{undefined}\}$
interpret M' : *prob-space* $?M\ i$ **for** i
using M **by** (*cases* $i \in I$) (*auto intro!*: *prob-spaceI*)
interpret P : *product-prob-space* $?M\ I$
by *unfold-locale*
have $\text{emeasure } (Pi_M\ I\ M) (\text{prod-emb } I\ M\ J\ (Pi_E\ J\ A)) = \text{emeasure } (Pi_M\ I\ ?M) (P.\text{emb } I\ J\ (Pi_E\ J\ A))$
by (*auto simp: prod-emb-def PiE-iff intro!*: *arg-cong2*[**where** $f = \text{emeasure}$]
PiM-cong)
also have $\dots = (\prod_{i \in J}. \text{emeasure } (M\ i) (A\ i))$
using $J\ A$ **by** (*subst P.emeasure-PiM-emb[OF J]*) (*auto intro!*: *prod.cong*)
finally show *?thesis* .
qed

lemma *distr-pair-PiM-eq-PiM*:

fixes $i' :: 'i$ **and** $I :: 'i$ *set* **and** $M :: 'i \Rightarrow 'a$ *measure*
assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M\ i)$ *prob-space* $(M\ i')$
shows $\text{distr } (M\ i' \otimes_M (\prod_{i \in I}. M\ i)) (\prod_{i \in \text{insert } i' I}. M\ i) (\lambda(x, X). X(i' := x)) =$
 $(\prod_{i \in \text{insert } i' I}. M\ i)$ (**is** $?L = -$)
proof (*rule measure-eqI-PiM-infinite[symmetric, OF refl]*)
interpret M' : *prob-space* $M\ i'$ **by** *fact*
interpret I : *prob-space* $(\prod_{i \in I}. M\ i)$
using M **by** (*intro prob-space-PiM*) *auto*
interpret I' : *prob-space* $(\prod_{i \in \text{insert } i' I}. M\ i)$
using M **by** (*intro prob-space-PiM*) *auto*
show *finite-measure* $(\prod_{i \in \text{insert } i' I}. M\ i)$
by *unfold-locale*
fix $J\ A$ **assume** $J: \text{finite } J$ $J \subseteq \text{insert } i' I$ **and** $A: \bigwedge i. i \in J \implies A\ i \in \text{sets } (M\ i)$
let $?X = \text{prod-emb } (\text{insert } i' I)\ M\ J\ (Pi_E\ J\ A)$
have $Pi_M (\text{insert } i' I)\ M\ ?X = (\prod_{i \in J}. M\ i (A\ i))$
using $M\ J\ A$ **by** (*intro emeasure-PiM-emb*) *auto*
also have $\dots = M\ i' (\text{if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) * (\prod_{i \in J - \{i'\}}. M\ i (A\ i))$
using *prod.insert-remove[of J $\lambda i. M\ i (A\ i) i'$]* *J M'.emeasure-space-1*
by (*cases* $i' \in J$) (*auto simp: insert-absorb*)
also have $(\prod_{i \in J - \{i'\}}. M\ i (A\ i)) = Pi_M\ I\ M (\text{prod-emb } I\ M\ (J - \{i'\}) (Pi_E\ (J - \{i'\})\ A))$
using $M\ J\ A$ **by** (*intro emeasure-PiM-emb[symmetric]*) *auto*
also have $M\ i' (\text{if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) * \dots =$
 $(M\ i' \otimes_M Pi_M\ I\ M) ((\text{if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) \times \text{prod-emb } I$

$M (J - \{i'\}) (Pi_E (J - \{i'\}) A)$
using $J A$ **by** (*intro I.emeasure-pair-measure-Times[symmetric] sets-PiM-I*)
auto
also have ((*if* $i' \in J$ *then* $(A i')$ *else* *space* $(M i')$) \times *prod-emb* $I M (J - \{i'\})$
 $(Pi_E (J - \{i'\}) A)$) =
 $(\lambda(x, X). X(i' := x)) - ' ?X \cap \text{space } (M i' \otimes_M Pi_M I M)$
using $A[\text{of } i', \text{ THEN sets.sets-into-space}]$ **unfolding** *set-eq-iff*
by (*simp add: prod-emb-def space-pair-measure space-PiM PiE-fun-upd ac-simps*
cong: conj-cong)
(auto simp add: Pi-iff Ball-def all-conj-distrib)
finally show $Pi_M (\text{insert } i' I) M ?X = ?L ?X$
using $J A$ **by** (*simp add: emeasure-distr*)
qed *simp*

lemma *distr-PiM-reindex:*

assumes $M: \bigwedge i. i \in K \implies \text{prob-space } (M i)$
assumes $f: \text{inj-on } f I f \in I \rightarrow K$
shows $\text{distr } (Pi_M K M) (\Pi_M i \in I. M (f i)) (\lambda\omega. \lambda n \in I. \omega (f n)) = (\Pi_M i \in I. M (f i))$
(is distr ?K ?I ?t = ?I)
proof (*rule measure-eqI-PiM-infinite[symmetric, OF refl]*)
interpret *prob-space ?I*
using $f M$ **by** (*intro prob-space-PiM*) *auto*
show *finite-measure ?I*
by *unfold-locales*
fix $A J$ **assume** $J: \text{finite } J J \subseteq I$ **and** $A: \bigwedge i. i \in J \implies A i \in \text{sets } (M (f i))$
have [*simp*]: $i \in J \implies \text{the-inv-into } I f (f i) = i$ **for** i
using $J f$ **by** (*intro the-inv-into-f-f*) *auto*
have $?I (\text{prod-emb } I (\lambda i. M (f i)) J (Pi_E J A)) = (\prod_{j \in J. M (f j) (A j)})$
using $f J A$ **by** (*intro emeasure-PiM-emb M*) *auto*
also have $\dots = (\prod_{j \in f' J. M j (A (\text{the-inv-into } I f j)))$
using $f J$ **by** (*subst prod.reindex*) (*auto intro!: prod.cong intro: inj-on-subset*
simp: the-inv-into-f-f)
also have $\dots = ?K (\text{prod-emb } K M (f' J) (\Pi_E j \in f' J. A (\text{the-inv-into } I f j)))$
using $f J A$ **by** (*intro emeasure-PiM-emb[symmetric] M*) (*auto simp: the-inv-into-f-f*)
also have $\text{prod-emb } K M (f' J) (\Pi_E j \in f' J. A (\text{the-inv-into } I f j)) = ?t - ' \text{prod-emb } I (\lambda i. M (f i)) J (Pi_E J A) \cap \text{space } ?K$
using $f J A$ **by** (*auto simp: prod-emb-def space-PiM Pi-iff PiE-iff Int-absorb1*)
also have $?K \dots = \text{distr } ?K ?I ?t (\text{prod-emb } I (\lambda i. M (f i)) J (Pi_E J A))$
using $f J A$ **by** (*intro emeasure-distr[symmetric] sets-PiM-I*) (*auto simp: Pi-iff*)
finally show $?I (\text{prod-emb } I (\lambda i. M (f i)) J (Pi_E J A)) = \text{distr } ?K ?I ?t$
 $(\text{prod-emb } I (\lambda i. M (f i)) J (Pi_E J A)) .$
qed *simp*

lemma *distr-PiM-component:*

assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M i)$
assumes $i \in I$
shows $\text{distr } (Pi_M I M) (M i) (\lambda\omega. \omega i) = M i$
proof –

have *: $(\lambda \omega. \omega \ i) - 'A \cap \text{space } (PiM \ I \ M) = \text{prod-emb } I \ M \ \{i\} \ (\Pi_E \ i' \in \{i\}. A)$
for A
by $(\text{auto simp: prod-emb-def space-PiM})$
show $?thesis$
apply $(\text{intro measure-eqI})$
apply $(\text{auto simp add: emeasure-distr } \langle i \in I \rangle * \text{emeasure-PiM-emb } M)$
apply $(\text{subst emeasure-PiM-emb})$
apply $(\text{simp-all add: } M \ \langle i \in I \rangle)$
done
qed

lemma $AE\text{-PiM-component}$:

$(\bigwedge i. i \in I \implies \text{prob-space } (M \ i)) \implies i \in I \implies AE \ x \ \text{in } M \ i. P \ x \implies AE \ x \ \text{in}$
 $PiM \ I \ M. P \ (x \ i)$
using $AE\text{-distrD}[of \ \lambda x. x \ i \ PiM \ I \ M \ M \ i]$
by $(\text{subst } (asm) \ \text{distr-PiM-component}[of \ I \ - \ i]) \ (\text{auto intro: } AE\text{-distrD}[of \ \lambda x. x \ i \ - \ P])$

lemma $decseq\text{-emb-PiE}$:

$incseq \ J \implies decseq \ (\lambda i. \text{prod-emb } I \ M \ (J \ i) \ (\Pi_E \ j \in J \ i. X \ j))$
by $(\text{fastforce simp: decseq-def prod-emb-def incseq-def Pi-iff})$

11.1 Sequence space

definition $comb\text{-seq} :: \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a)$ **where**
 $comb\text{-seq} \ i \ \omega \ \omega' \ j = (\text{if } j < i \ \text{then } \omega \ j \ \text{else } \omega' \ (j - i))$

lemma $split\text{-comb-seq}$: $P \ (comb\text{-seq} \ i \ \omega \ \omega' \ j) \longleftrightarrow (j < i \longrightarrow P \ (\omega \ j)) \wedge (\forall k. j = i + k \longrightarrow P \ (\omega' \ k))$

by $(\text{auto simp: comb-seq-def not-less})$

lemma $split\text{-comb-seq-asm}$: $P \ (comb\text{-seq} \ i \ \omega \ \omega' \ j) \longleftrightarrow \neg ((j < i \wedge \neg P \ (\omega \ j)) \vee (\exists k. j = i + k \wedge \neg P \ (\omega' \ k)))$

by $(\text{auto simp: comb-seq-def})$

lemma $measurable\text{-comb-seq}$:

$(\lambda(\omega, \omega'). \text{comb-seq} \ i \ \omega \ \omega') \in \text{measurable} \ ((\Pi_M \ i \in UNIV. M) \otimes_M (\Pi_M \ i \in UNIV. M)) \ (\Pi_M \ i \in UNIV. M)$

proof $(\text{rule measurable-PiM-single})$

show $(\lambda(\omega, \omega'). \text{comb-seq} \ i \ \omega \ \omega') \in \text{space} \ ((\Pi_M \ i \in UNIV. M) \otimes_M (\Pi_M \ i \in UNIV. M)) \rightarrow (\text{UNIV} \rightarrow_E \text{space } M)$

by $(\text{auto simp: space-pair-measure space-PiM PiE-iff split: split-comb-seq})$

fix $j :: \text{nat}$ **and** **assume** $A: A \in \text{sets } M$

then have *: $\{\omega \in \text{space} \ ((\Pi_M \ i \in UNIV. M) \otimes_M (\Pi_M \ i \in UNIV. M)). \text{case-prod} \ (comb\text{-seq} \ i) \ \omega \ j \in A\} =$

$(\text{if } j < i \ \text{then } \{\omega \in \text{space} \ (\Pi_M \ i \in UNIV. M). \omega \ j \in A\} \times \text{space} \ (\Pi_M \ i \in UNIV. M)$

$\ \text{else } \text{space} \ (\Pi_M \ i \in UNIV. M) \times \{\omega \in \text{space} \ (\Pi_M \ i \in UNIV. M). \omega \ (j - i) \in A\}$

by (*auto simp: space-PiM space-pair-measure comb-seq-def dest: sets.sets-into-space*)
show $\{\omega \in \text{space } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M)). \text{case-prod}$
*(comb-seq i) ω $j \in A\} \in \text{sets } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M))$
unfolding * **by** (*auto simp: A intro!: sets-Collect-single*)
qed*

lemma *measurable-comb-seq'*[*measurable (raw)*]:
assumes $f: f \in \text{measurable } N (\prod_M i \in \text{UNIV}. M)$ **and** $g: g \in \text{measurable } N (\prod_M i \in \text{UNIV}. M)$
shows $(\lambda x. \text{comb-seq } i (f x) (g x)) \in \text{measurable } N (\prod_M i \in \text{UNIV}. M)$
using *measurable-compose[OF measurable-Pair[OF f g] measurable-comb-seq]* **by**
simp

lemma *comb-seq-0*: *comb-seq 0 ω $\omega' = \omega'$*
by (*auto simp add: comb-seq-def*)

lemma *comb-seq-Suc*: *comb-seq (Suc n) ω $\omega' = \text{comb-seq } n \omega (\text{case-nat } (\omega n) \omega')$*
by (*auto simp add: comb-seq-def not-less less-Suc-eq le-imp-diff-is-add intro!: ext split: nat.split*)

lemma *comb-seq-Suc-0*[*simp*]: *comb-seq (Suc 0) $\omega = \text{case-nat } (\omega 0)$*
by (*intro ext (simp add: comb-seq-Suc comb-seq-0)*)

lemma *comb-seq-less*: $i < n \implies \text{comb-seq } n \omega \omega' i = \omega i$
by (*auto split: split-comb-seq*)

lemma *comb-seq-add*: *comb-seq n ω $\omega' (i + n) = \omega' i$*
by (*auto split: nat.split split-comb-seq*)

lemma *case-nat-comb-seq*: *case-nat s' (comb-seq n ω $\omega') (i + n) = \text{case-nat}$
(case-nat s' ω n) $\omega' i$
by (*auto split: nat.split split-comb-seq*)*

lemma *case-nat-comb-seq'*:
*case-nat s (comb-seq i ω $\omega') = \text{comb-seq } (Suc i) (\text{case-nat } s \omega) \omega'$
by (*auto split: split-comb-seq nat.split*)*

locale *sequence-space = product-prob-space* $\lambda i. M \text{ UNIV} :: \text{nat set}$ **for** M
begin

abbreviation $S \equiv \prod_M i \in \text{UNIV} :: \text{nat set}. M$

lemma *infprod-in-sets*[*intro*]:
fixes $E :: \text{nat} \Rightarrow 'a \text{ set}$ **assumes** $E: \bigwedge i. E i \in \text{sets } M$
shows $\Pi i \text{ UNIV } E \in \text{sets } S$
proof –
have $\Pi i \text{ UNIV } E = (\bigcap i. \text{emb UNIV } \{..i\} (\prod_{E j \in \{..i\}} E j))$
using $E E$ [*THEN sets.sets-into-space*]
by (*auto simp: prod-emb-def Pi-iff extensional-def*)

with E **show** *?thesis* **by** *auto*
qed

lemma *measure-PiM-countable*:

fixes $E :: \text{nat} \Rightarrow 'a \text{ set}$ **assumes** $E: \bigwedge i. E i \in \text{sets } M$
shows $(\lambda n. \prod_{i \leq n}. \text{measure } M (E i)) \longrightarrow \text{measure } S (Pi \text{ UNIV } E)$

proof –

let $?E = \lambda n. \text{emb UNIV } \{..n\} (Pi_E \{.. n\} E)$
have $\bigwedge n. (\prod_{i \leq n}. \text{measure } M (E i)) = \text{measure } S (?E n)$
using E **by** (*simp add: measure-PiM-emb*)
moreover have $Pi \text{ UNIV } E = (\bigcap n. ?E n)$
using $E E$ [*THEN sets.sets-into-space*]
by (*auto simp: prod-emb-def extensional-def Pi-iff*)
moreover have $\text{range } ?E \subseteq \text{sets } S$
using E **by** *auto*
moreover have *decseq* $?E$
by (*auto simp: prod-emb-def Pi-iff decseq-def*)
ultimately show *?thesis*
by (*simp add: finite-Lim-measure-decseq*)

qed

lemma *nat-eq-diff-eq*:

fixes $a b c :: \text{nat}$
shows $c \leq b \implies a = b - c \longleftrightarrow a + c = b$
by *auto*

lemma *PiM-comb-seq*:

distr $(S \otimes_M S) S (\lambda(\omega, \omega'). \text{comb-seq } i \omega \omega') = S$ (**is** $?D = -$)

proof (*rule PiM-eq*)

let $?I = \text{UNIV} :: \text{nat set}$ **and** $?M = \lambda n. M$
let *distr - - ?f = ?D*

fix $J E$ **assume** $J: \text{finite } J J \subseteq ?I \bigwedge j. j \in J \implies E j \in \text{sets } M$

let $?X = \text{prod-emb } ?I ?M J (\prod_E j \in J. E j)$

have $\bigwedge j x. j \in J \implies x \in E j \implies x \in \text{space } M$

using J ($?f$) [*THEN sets.sets-into-space*] **by** (*auto simp: space-PiM Pi-iff subset-eq*)

with J **have** $?f - ' ?X \cap \text{space } (S \otimes_M S) =$

$(\text{prod-emb } ?I ?M (J \cap \{..<i\})) (\prod_E j \in J \cap \{..<i\}. E j) \times$

$(\text{prod-emb } ?I ?M ((op + i) - ' J) (\prod_E j \in (op + i) - ' J. E (i + j)))$ (**is** $- =$

$?E \times ?F$)

by (*auto simp: space-pair-measure space-PiM prod-emb-def all-conj-distrib PiE-iff split: split-comb-seq split-comb-seq-asm*)

then have *emeasure* $?D ?X = \text{emeasure } (S \otimes_M S) (?E \times ?F)$

by (*subst emeasure-distr[OF measurable-comb-seq]*)

(*auto intro!: sets-PiM-I simp: split-beta' J*)

also have $\dots = \text{emeasure } S ?E * \text{emeasure } S ?F$

using J **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro!: sets-PiM-I*

finite-vimageI simp: inj-on-def)

also have *emeasure* $S ?F = (\prod_{j \in (op + i) - ' J. \text{emeasure } M (E (i + j))})$

using J **by** (*intro emeasure-PiM-emb*) (*simp-all add: finite-vimageI inj-on-def*)
also have $\dots = (\prod_{j \in J - (J \cap \{..<i\})}. \text{emeasure } M (E j))$
by (*rule prod.reindex-cong [of $\lambda x. x - i$]*)
(auto simp: image-iff Bex-def not-less nat-eq-diff-eq ac-simps cong: conj-cong
intro!: inj-onI)
also have $\text{emeasure } S ?E = (\prod_{j \in J \cap \{..<i\}}. \text{emeasure } M (E j))$
using J **by** (*intro emeasure-PiM-emb*) *simp-all*
also have $(\prod_{j \in J \cap \{..<i\}}. \text{emeasure } M (E j)) * (\prod_{j \in J - (J \cap \{..<i\})}. \text{emeasure } M (E j)) = (\prod_{j \in J}. \text{emeasure } M (E j))$
by (*subst mult.commute*) (*auto simp: J prod.subset-diff[symmetric]*)
finally show $\text{emeasure } ?D ?X = (\prod_{j \in J}. \text{emeasure } M (E j))$.
qed *simp-all*

lemma *PiM-iter*:

distr ($M \otimes_M S$) $S (\lambda(s, \omega). \text{case-nat } s \ \omega) = S$ (**is** $?D = -$)
proof (*rule PiM-eq*)
let $?I = \text{UNIV}::\text{nat set}$ **and** $?M = \lambda n. M$
let $\text{distr } - \text{ } ?f = ?D$

fix $J E$ **assume** $J: \text{finite } J \ J \subseteq ?I \ \bigwedge j. j \in J \implies E j \in \text{sets } M$
let $?X = \text{prod-emb } ?I ?M J (\prod_E j \in J. E j)$
have $\bigwedge j x. j \in J \implies x \in E j \implies x \in \text{space } M$
using $J(3)[\text{THEN sets.sets-into-space}]$ **by** (*auto simp: space-PiM Pi-iff subset-eq*)
with J **have** $?f - ' ?X \cap \text{space } (M \otimes_M S) = (\text{if } 0 \in J \text{ then } E \ 0 \text{ else } \text{space } M)$
 \times
(prod-emb ?I ?M (Suc - ' J) (\prod_E j \in \text{Suc } - ' J. E (Suc j))) (**is** $- = ?E \times ?F$)
by (*auto simp: space-pair-measure space-PiM PiE-iff prod-emb-def all-conj-distrib*
split: nat.split nat.split-asm)
then have $\text{emeasure } ?D ?X = \text{emeasure } (M \otimes_M S) (?E \times ?F)$
by (*subst emeasure-distr*)
(auto intro!: sets-PiM-I simp: split-beta' J)
also have $\dots = \text{emeasure } M ?E * \text{emeasure } S ?F$
using J **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro!: sets-PiM-I*
finite-vimageI)
also have $\text{emeasure } S ?F = (\prod_{j \in \text{Suc } - ' J}. \text{emeasure } M (E (\text{Suc } j)))$
using J **by** (*intro emeasure-PiM-emb*) (*simp-all add: finite-vimageI*)
also have $\dots = (\prod_{j \in J - \{0\}}. \text{emeasure } M (E j))$
by (*rule prod.reindex-cong [of $\lambda x. x - 1$]*)
(auto simp: image-iff Bex-def not-less nat-eq-diff-eq ac-simps cong: conj-cong
intro!: inj-onI)
also have $\text{emeasure } M ?E * (\prod_{j \in J - \{0\}}. \text{emeasure } M (E j)) = (\prod_{j \in J}. \text{emeasure } M (E j))$
by (*auto simp: M.emeasure-space-1 prod.remove J*)
finally show $\text{emeasure } ?D ?X = (\prod_{j \in J}. \text{emeasure } M (E j))$.
qed *simp-all*

end

end

12 Independent families of events, event sets, and random variables

theory *Independent-Family*

imports *Infinite-Product-Measure*

begin

definition (in *prob-space*)

indep-sets $F I \longleftrightarrow (\forall i \in I. F i \subseteq \text{events}) \wedge$
 $(\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow (\forall A \in \text{Pi } J F. \text{prob } (\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob } (A j))))$

definition (in *prob-space*)

indep-set $A B \longleftrightarrow \text{indep-sets } (\text{case-bool } A B) \text{ UNIV}$

definition (in *prob-space*)

indep-events-def-alt: $\text{indep-events } A I \longleftrightarrow \text{indep-sets } (\lambda i. \{A i\}) I$

lemma (in *prob-space*) *indep-events-def*:

$\text{indep-events } A I \longleftrightarrow (A I \subseteq \text{events}) \wedge$
 $(\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{prob } (\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob } (A j)))$

unfolding *indep-events-def-alt indep-sets-def*

apply (*simp add: Ball-def Pi-iff image-subset-iff-funcset*)

apply (*intro conj-cong refl arg-cong[where f=All] ext imp-cong*)

apply *auto*

done

lemma (in *prob-space*) *indep-eventsI*:

$(\bigwedge i. i \in I \implies F i \in \text{sets } M) \implies (\bigwedge J. J \subseteq I \implies \text{finite } J \implies J \neq \{\} \implies \text{prob}$
 $(\bigcap_{i \in J}. F i) = (\prod_{i \in J}. \text{prob } (F i))) \implies \text{indep-events } F I$

by (*auto simp: indep-events-def*)

definition (in *prob-space*)

indep-event $A B \longleftrightarrow \text{indep-events } (\text{case-bool } A B) \text{ UNIV}$

lemma (in *prob-space*) *indep-sets-cong*:

$I = J \implies (\bigwedge i. i \in I \implies F i = G i) \implies \text{indep-sets } F I \longleftrightarrow \text{indep-sets } G J$

by (*simp add: indep-sets-def, intro conj-cong all-cong imp-cong ball-cong*) *blast+*

lemma (in *prob-space*) *indep-events-finite-index-events*:

$\text{indep-events } F I \longleftrightarrow (\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-events } F J)$

by (*auto simp: indep-events-def*)

lemma (in *prob-space*) *indep-sets-finite-index-sets*:

$\text{indep-sets } F I \longleftrightarrow (\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-sets } F J)$

proof (*intro iffI allI impI*)

assume $*$: $\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-sets } F J$

show $\text{indep-sets } F I$ **unfolding** *indep-sets-def*

proof (*intro conjI ballI allI impI*)

```

fix  $i$  assume  $i \in I$ 
with  $*$ [THEN spec, of {i}] show  $F i \subseteq \text{events}$ 
by (auto simp: indep-sets-def)
qed (insert *, auto simp: indep-sets-def)
qed (auto simp: indep-sets-def)

```

```

lemma (in prob-space) indep-sets-mono-index:
 $J \subseteq I \implies \text{indep-sets } F I \implies \text{indep-sets } F J$ 
unfolding indep-sets-def by auto

```

```

lemma (in prob-space) indep-sets-mono-sets:
assumes indep: indep-sets F I
assumes mono:  $\bigwedge i. i \in I \implies G i \subseteq F i$ 
shows indep-sets G I

```

```

proof –
have  $(\forall i \in I. F i \subseteq \text{events}) \implies (\forall i \in I. G i \subseteq \text{events})$ 
using mono by auto
moreover have  $\bigwedge A J. J \subseteq I \implies A \in (\prod_{j \in J}. G j) \implies A \in (\prod_{j \in J}. F j)$ 
using mono by (auto simp: Pi-iff)
ultimately show ?thesis
using indep by (auto simp: indep-sets-def)
qed

```

```

lemma (in prob-space) indep-sets-mono:
assumes indep: indep-sets F I
assumes mono:  $J \subseteq I \bigwedge i. i \in J \implies G i \subseteq F i$ 
shows indep-sets G J
apply (rule indep-sets-mono-sets)
apply (rule indep-sets-mono-index)
apply (fact +)
done

```

```

lemma (in prob-space) indep-setsI:
assumes  $\bigwedge i. i \in I \implies F i \subseteq \text{events}$ 
and  $\bigwedge A J. J \neq \{\} \implies J \subseteq I \implies \text{finite } J \implies (\forall j \in J. A j \in F j) \implies \text{prob}$ 
 $(\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob } (A j))$ 
shows indep-sets F I
using assms unfolding indep-sets-def by (auto simp: Pi-iff)

```

```

lemma (in prob-space) indep-setsD:
assumes indep-sets F I and  $J \subseteq I \ J \neq \{\} \ \text{finite } J \ \forall j \in J. A j \in F j$ 
shows  $\text{prob } (\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob } (A j))$ 
using assms unfolding indep-sets-def by auto

```

```

lemma (in prob-space) indep-setI:
assumes ev: A  $\subseteq$  events B  $\subseteq$  events
and indep:  $\bigwedge a b. a \in A \implies b \in B \implies \text{prob } (a \cap b) = \text{prob } a * \text{prob } b$ 
shows indep-set A B
unfolding indep-set-def

```

```

proof (rule indep-setsI)
  fix  $F J$  assume  $J \neq \{\}$   $J \subseteq UNIV$ 
    and  $F: \forall j \in J. F j \in (case\ j\ of\ True \Rightarrow A \mid False \Rightarrow B)$ 
    have  $J \in Pow\ UNIV$  by auto
    with  $F \langle J \neq \{\} \rangle$  indep[of  $F\ True\ F\ False$ ]
    show  $prob (\bigcap_{j \in J} F j) = (\prod_{j \in J} prob (F j))$ 
      unfolding UNIV-bool Pow-insert by (auto simp: ac-simps)
qed (auto split: bool.split simp: ev)

lemma (in prob-space) indep-setD:
  assumes indep: indep-set  $A B$  and  $ev: a \in A\ b \in B$ 
  shows  $prob (a \cap b) = prob\ a * prob\ b$ 
  using indep[unfolded indep-set-def, THEN indep-setsD, of UNIV case-bool a b]
   $ev$ 
  by (simp add: ac-simps UNIV-bool)

lemma (in prob-space)
  assumes indep: indep-set  $A B$ 
  shows indep-setD-ev1:  $A \subseteq events$ 
    and indep-setD-ev2:  $B \subseteq events$ 
  using indep unfolding indep-set-def indep-sets-def UNIV-bool by auto

lemma (in prob-space) indep-sets-dynkin:
  assumes indep: indep-sets  $F I$ 
  shows indep-sets ( $\lambda i. dynkin (space\ M) (F\ i)$ )  $I$ 
    (is indep-sets ? $F I$ )
proof (subst indep-sets-finite-index-sets, intro allI impI ballI)
  fix  $J$  assume finite  $J\ J \subseteq I\ J \neq \{\}$ 
  with indep have indep-sets  $F J$ 
    by (subst (asm) indep-sets-finite-index-sets) auto
  { fix  $J K$  assume indep-sets  $F K$ 
    let  $?G = \lambda S\ i. if\ i \in S\ then\ ?F\ i\ else\ F\ i$ 
    assume finite  $J\ J \subseteq K$ 
    then have indep-sets ( $?G\ J$ )  $K$ 
    proof induct
      case (insert  $j\ J$ )
      moreover define  $G$  where  $G = ?G\ J$ 
      ultimately have  $G: indep-sets\ G\ K \wedge i. i \in K \Longrightarrow G\ i \subseteq events$  and  $j \in K$ 
        by (auto simp: indep-sets-def)
      let  $?D = \{E \in events. indep-sets (G(j := \{E\}))\ K\}$ 
      { fix  $X$  assume  $X: X \in events$ 
        assume indep:  $\bigwedge J\ A. J \neq \{\} \Longrightarrow J \subseteq K \Longrightarrow finite\ J \Longrightarrow j \notin J \Longrightarrow (\forall i \in J. A\ i \in G\ i)$ 
         $\Longrightarrow prob ((\bigcap_{i \in J} A\ i) \cap X) = prob\ X * (\prod_{i \in J} prob (A\ i))$ 
        have indep-sets ( $G(j := \{X\})\ K$ )
        proof (rule indep-setsI)
          fix  $i$  assume  $i \in K$  then show ( $G(j := \{X\})\ i \subseteq events$ )
            using  $G\ X$  by auto
        next

```

```

fix  $A J$  assume  $J: J \neq \{\}$   $J \subseteq K$  finite  $J \forall i \in J. A i \in (G(j := \{X\})) i$ 
show  $\text{prob} (\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob} (A j))$ 
proof cases
  assume  $j \in J$ 
  with  $J$  have  $A j = X$  by auto
  show ?thesis
  proof cases
    assume  $J = \{j\}$  then show ?thesis by simp
  next
    assume  $J \neq \{j\}$ 
    have  $\text{prob} (\bigcap_{i \in J}. A i) = \text{prob} ((\bigcap_{i \in J - \{j\}}. A i) \cap X)$ 
      using  $\langle j \in J \rangle \langle A j = X \rangle$  by (auto intro!: arg-cong[where f=prob])
    split: if-split-asm)
    also have  $\dots = \text{prob } X * (\prod_{i \in J - \{j\}}. \text{prob} (A i))$ 
    proof (rule indep)
      show  $J - \{j\} \neq \{\}$   $J - \{j\} \subseteq K$  finite  $(J - \{j\}) j \notin J - \{j\}$ 
        using  $J \langle J \neq \{j\} \rangle \langle j \in J \rangle$  by auto
      show  $\forall i \in J - \{j\}. A i \in G i$ 
        using  $J$  by auto
    qed
    also have  $\dots = \text{prob} (A j) * (\prod_{i \in J - \{j\}}. \text{prob} (A i))$ 
      using  $\langle A j = X \rangle$  by simp
    also have  $\dots = (\prod_{i \in J}. \text{prob} (A i))$ 
      unfolding prod.insert-remove[OF  $\langle \text{finite } J \rangle$ , symmetric, of  $\lambda i. \text{prob}$ 
    ( $A i$ )]
      using  $\langle j \in J \rangle$  by (simp add: insert-absorb)
      finally show ?thesis .
    qed
  next
    assume  $j \notin J$ 
    with  $J$  have  $\forall i \in J. A i \in G i$  by (auto split: if-split-asm)
    with  $J$  show ?thesis
      by (intro indep-setsD[OF  $G(1)$ ]) auto
    qed
  qed }
note indep-sets-insert = this
have dynkin-system (space M) ?D
proof (rule dynkin-systemI', simp-all cong del: indep-sets-cong, safe)
  show indep-sets ( $G(j := \{\{\}\})$ )  $K$ 
    by (rule indep-sets-insert) auto
next
fix  $X$  assume  $X: X \in \text{events}$  and  $G': \text{indep-sets} (G(j := \{X\})) K$ 
show indep-sets ( $G(j := \{\text{space } M - X\})$ )  $K$ 
proof (rule indep-sets-insert)
  fix  $J A$  assume  $J: J \neq \{\}$   $J \subseteq K$  finite  $J j \notin J$  and  $A: \forall i \in J. A i \in G i$ 
  then have  $A\text{-sets}: \bigwedge i. i \in J \implies A i \in \text{events}$ 
  using  $G$  by auto
  have  $\text{prob} ((\bigcap_{j \in J}. A j) \cap (\text{space } M - X)) =$ 
     $\text{prob} ((\bigcap_{j \in J}. A j) - (\bigcap_{i \in \text{insert } j J}. (A(j := X)) i))$ 

```

```

    using A-sets sets.sets-into-space[of - M] X ⟨J ≠ {}⟩
    by (auto intro!: arg-cong[where f=prob] split: if-split-asm)
  also have ... = prob (⋂j∈J. A j) - prob (⋂i∈insert j J. (A(j := X)) i)
    using J ⟨J ≠ {}⟩ ⟨j ∉ J⟩ A-sets X sets.sets-into-space
    by (auto intro!: finite-measure-Diff sets.finite-INT split: if-split-asm)
  finally have prob ((⋂j∈J. A j) ∩ (space M - X)) =
    prob (⋂j∈J. A j) - prob (⋂i∈insert j J. (A(j := X)) i) .
  moreover {
    have prob (⋂j∈J. A j) = (∏j∈J. prob (A j))
      using J A ⟨finite J⟩ by (intro indep-setsD[OF G(1)]) auto
    then have prob (⋂j∈J. A j) = prob (space M) * (∏i∈J. prob (A i))
      using prob-space by simp }
  moreover {
    have prob (⋂i∈insert j J. (A(j := X)) i) = (∏i∈insert j J. prob ((A(j
:= X)) i))
      using J A ⟨j ∈ K⟩ by (intro indep-setsD[OF G']) auto
    then have prob (⋂i∈insert j J. (A(j := X)) i) = prob X * (∏i∈J.
prob (A i))
      using ⟨finite J⟩ ⟨j ∉ J⟩ by (auto intro!: prod.cong) }
    ultimately have prob ((⋂j∈J. A j) ∩ (space M - X)) = (prob (space
M) - prob X) * (∏i∈J. prob (A i))
      by (simp add: field-simps)
    also have ... = prob (space M - X) * (∏i∈J. prob (A i))
      using X A by (simp add: finite-measure-compl)
    finally show prob ((⋂j∈J. A j) ∩ (space M - X)) = prob (space M -
X) * (∏i∈J. prob (A i)) .
    qed (insert X, auto)
  next
  fix F :: nat ⇒ 'a set assume disj: disjoint-family F and range F ⊆ ?D
  then have F: ⋀i. F i ∈ events ⋀i. indep-sets (G(j:={F i})) K by auto
  show indep-sets (G(j := {⋃k. F k})) K
  proof (rule indep-sets-insert)
    fix J A assume J: j ∉ J J ≠ {} J ⊆ K finite J and A: ∀i∈J. A i ∈ G i
    then have A-sets: ⋀i. i∈J ⇒ A i ∈ events
      using G by auto
    have prob ((⋂j∈J. A j) ∩ (⋃k. F k)) = prob (⋃k. (⋂i∈insert j J. (A(j
:= F k)) i))
      using ⟨J ≠ {}⟩ ⟨j ∉ J⟩ ⟨j ∈ K⟩ by (auto intro!: arg-cong[where f=prob]
split: if-split-asm)
    moreover have (λk. prob (⋂i∈insert j J. (A(j := F k)) i)) sums prob
(⋃k. (⋂i∈insert j J. (A(j := F k)) i))
      proof (rule finite-measure-UNION)
        show disjoint-family (λk. ⋂i∈insert j J. (A(j := F k)) i)
          using disj by (rule disjoint-family-on-bisimulation) auto
        show range (λk. ⋂i∈insert j J. (A(j := F k)) i) ⊆ events
          using A-sets F ⟨finite J⟩ ⟨J ≠ {}⟩ ⟨j ∉ J⟩ by (auto intro!: sets.Int)
        qed
      moreover { fix k
        from J A ⟨j ∈ K⟩ have prob (⋂i∈insert j J. (A(j := F k)) i) = prob

```

```

(F k) * (∏ i ∈ J. prob (A i))
  by (subst indep-setsD[OF F(2)]) (auto intro!: prod.cong split: if-split-asm)
  also have ... = prob (F k) * prob (∏ i ∈ J. A i)
    using J A ⟨j ∈ K⟩ by (subst indep-setsD[OF G(1)]) auto
  finally have prob (∏ i ∈ insert j J. (A(j := F k)) i) = prob (F k) * prob
(∏ i ∈ J. A i) . }
  ultimately have (λk. prob (F k) * prob (∏ i ∈ J. A i)) sums (prob ((∏ j ∈ J.
A j) ∩ (∪ k. F k)))
    by simp
  moreover
  have (λk. prob (F k) * prob (∏ i ∈ J. A i)) sums (prob (∪ k. F k) * prob
(∏ i ∈ J. A i))
    using disj F(1) by (intro finite-measure-UNION sums-mult2) auto
  then have (λk. prob (F k) * prob (∏ i ∈ J. A i)) sums (prob (∪ k. F k) *
(∏ i ∈ J. prob (A i)))
    using J A ⟨j ∈ K⟩ by (subst indep-setsD[OF G(1), symmetric]) auto
  ultimately
  show prob ((∏ j ∈ J. A j) ∩ (∪ k. F k)) = prob (∪ k. F k) * (∏ j ∈ J. prob
(A j))
    by (auto dest!: sums-unique)
  qed (insert F, auto)
qed (insert sets.sets-into-space, auto)
then have mono: dynkin (space M) (G j) ⊆ {E ∈ events. indep-sets (G(j
:= {E})) K}
proof (rule dynkin-system.dynkin-subset, safe)
  fix X assume X ∈ G j
  then show X ∈ events using G ⟨j ∈ K⟩ by auto
  from ⟨indep-sets G K⟩
  show indep-sets (G(j := {X})) K
    by (rule indep-sets-mono-sets) (insert ⟨X ∈ G j⟩, auto)
qed
have indep-sets (G(j := ?D)) K
proof (rule indep-setsI)
  fix i assume i ∈ K then show (G(j := ?D)) i ⊆ events
    using G(2) by auto
next
  fix A J assume J: J ≠ {} J ⊆ K finite J and A: ∀ i ∈ J. A i ∈ (G(j :=
?D)) i
  show prob (∏ j ∈ J. A j) = (∏ j ∈ J. prob (A j))
  proof cases
    assume j ∈ J
    with A have indep: indep-sets (G(j := {A j})) K by auto
    from J A show ?thesis
      by (intro indep-setsD[OF indep]) auto
  next
    assume j ∉ J
    with J A have ∀ i ∈ J. A i ∈ G i by (auto split: if-split-asm)
    with J show ?thesis
      by (intro indep-setsD[OF G(1)]) auto

```

```

    qed
  qed
  then have indep-sets (G(j := dynkin (space M) (G j))) K
    by (rule indep-sets-mono-sets) (insert mono, auto)
  then show ?case
    by (rule indep-sets-mono-sets) (insert ⟨j ∈ K⟩ ⟨j ∉ J⟩, auto simp: G-def)
  qed (insert ⟨indep-sets F K⟩, simp) }
  from this[OF ⟨indep-sets F J⟩ ⟨finite J⟩ subset-refl]
  show indep-sets ?F J
    by (rule indep-sets-mono-sets) auto
  qed

```

```

lemma (in prob-space) indep-sets-sigma:
  assumes indep: indep-sets F I
  assumes stable:  $\bigwedge i. i \in I \implies \text{Int-stable } (F i)$ 
  shows indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I
proof -
  from indep-sets-dynkin[OF indep]
  show ?thesis
  proof (rule indep-sets-mono-sets, subst sigma-eq-dynkin, simp-all add: stable)
    fix i assume i ∈ I
    with indep have F i ⊆ events by (auto simp: indep-sets-def)
    with sets.sets-into-space show F i ⊆ Pow (space M) by auto
  qed
  qed

```

```

lemma (in prob-space) indep-sets-sigma-sets-iff:
  assumes  $\bigwedge i. i \in I \implies \text{Int-stable } (F i)$ 
  shows indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I  $\longleftrightarrow$  indep-sets F I
proof
  assume indep-sets F I then show indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I
    by (rule indep-sets-sigma) fact
  next
  assume indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I then show indep-sets F I
    by (rule indep-sets-mono-sets) (intro subsetI sigma-sets.Basic)
  qed

```

```

definition (in prob-space)
  indep-vars-def2: indep-vars M' X I  $\longleftrightarrow$ 
  ( $\forall i \in I. \text{random-variable } (M' i) (X i)$ )  $\wedge$ 
  indep-sets ( $\lambda i. \{ X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' i) \}$ ) I

```

```

definition (in prob-space)
  indep-var Ma A Mb B  $\longleftrightarrow$  indep-vars (case-bool Ma Mb) (case-bool A B) UNIV

```

```

lemma (in prob-space) indep-vars-def:
  indep-vars M' X I  $\longleftrightarrow$ 
  ( $\forall i \in I. \text{random-variable } (M' i) (X i)$ )  $\wedge$ 
  indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) \{ X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' i) \}$ ) I

```

```

i)) I
  unfolding indep-vars-def2
  apply (rule conj-cong[OF refl])
  apply (rule indep-sets-sigma-sets-iff[symmetric])
  apply (auto simp: Int-stable-def)
  apply (rule-tac x=A ∩ Aa in exI)
  apply auto
  done

lemma (in prob-space) indep-var-eq:
  indep-var S X T Y  $\longleftrightarrow$ 
    (random-variable S X  $\wedge$  random-variable T Y)  $\wedge$ 
    indep-set
      (sigma-sets (space M) { X -‘ A  $\cap$  space M | A. A  $\in$  sets S })
      (sigma-sets (space M) { Y -‘ A  $\cap$  space M | A. A  $\in$  sets T })
  unfolding indep-var-def indep-vars-def indep-set-def UNIV-bool
  by (intro arg-cong2[where f=op  $\wedge$ ] arg-cong2[where f=indep-sets] ext)
    (auto split: bool.split)

```

```

lemma (in prob-space) indep-sets2-eq:
  indep-set A B  $\longleftrightarrow$  A  $\subseteq$  events  $\wedge$  B  $\subseteq$  events  $\wedge$  ( $\forall a \in A. \forall b \in B. \text{prob } (a \cap b) =$ 
  prob a * prob b)
  unfolding indep-set-def
  proof (intro iffI ballI conjI)
    assume indep: indep-sets (case-bool A B) UNIV
    { fix a b assume a  $\in$  A b  $\in$  B
      with indep-setsD[OF indep, of UNIV case-bool a b]
      show prob (a  $\cap$  b) = prob a * prob b
        unfolding UNIV-bool by (simp add: ac-simps) }
    from indep show A  $\subseteq$  events B  $\subseteq$  events
      unfolding indep-sets-def UNIV-bool by auto
  next
    assume *: A  $\subseteq$  events  $\wedge$  B  $\subseteq$  events  $\wedge$  ( $\forall a \in A. \forall b \in B. \text{prob } (a \cap b) = \text{prob } a$ 
    * prob b)
    show indep-sets (case-bool A B) UNIV
      proof (rule indep-setsI)
        fix i show (case i of True  $\Rightarrow$  A | False  $\Rightarrow$  B)  $\subseteq$  events
          using * by (auto split: bool.split)
      next
        fix J X assume J  $\neq$  {} J  $\subseteq$  UNIV and X:  $\forall j \in J. X j \in$  (case j of True  $\Rightarrow$ 
        A | False  $\Rightarrow$  B)
        then have J = {True}  $\vee$  J = {False}  $\vee$  J = {True,False}
          by (auto simp: UNIV-bool)
        then show prob ( $\bigcap j \in J. X j$ ) = ( $\prod j \in J. \text{prob } (X j)$ )
          using X * by auto
      qed
    qed
  qed

```

```

lemma (in prob-space) indep-set-sigma-sets:

```


assumes *indep-set* $A B$
assumes A : *Int-stable* A **and** B : *Int-stable* B
shows *indep-set* (*sigma-sets* (*space* M) A) (*sigma-sets* (*space* M) B)
proof –
have *indep-sets* ($\lambda i.$ *sigma-sets* (*space* M) (*case* i of *True* $\Rightarrow A$ | *False* $\Rightarrow B$))
UNIV
proof (*rule indep-sets-sigma*)
show *indep-sets* (*case-bool* $A B$) *UNIV*
by (*rule* (*indep-set* $A B$)[*unfolded indep-set-def*])
fix i **show** *Int-stable* (*case* i of *True* $\Rightarrow A$ | *False* $\Rightarrow B$)
using $A B$ **by** (*cases* i) *auto*
qed
then show *?thesis*
unfolding *indep-set-def*
by (*rule indep-sets-mono-sets*) (*auto split: bool.split*)
qed

lemma (*in prob-space*) *indep-eventsI-indep-vars*:
assumes *indep: indep-vars* $N X I$
assumes P : $\bigwedge i. i \in I \Rightarrow \{x \in \text{space } (N i). P i x\} \in \text{sets } (N i)$
shows *indep-events* ($\lambda i. \{x \in \text{space } M. P i (X i x)\}$) I
proof –
have *indep-sets* ($\lambda i. \{X i - 'A \cap \text{space } M \mid A. A \in \text{sets } (N i)\}$) I
using *indep unfolding indep-vars-def2* **by** *auto*
then show *?thesis*
unfolding *indep-events-def-alt*
proof (*rule indep-sets-mono-sets*)
fix i **assume** $i \in I$
then have $\{\{x \in \text{space } M. P i (X i x)\}\} = \{X i - ' \{x \in \text{space } (N i). P i x\}$
 $\cap \text{space } M\}$
using *indep* **by** (*auto simp: indep-vars-def dest: measurable-space*)
also have $\dots \subseteq \{X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (N i)\}$
using P [*OF* ($i \in I$)] **by** *blast*
finally show $\{\{x \in \text{space } M. P i (X i x)\}\} \subseteq \{X i - ' A \cap \text{space } M \mid A. A \in$
 $\text{sets } (N i)\}$.
qed
qed

lemma (*in prob-space*) *indep-sets-collect-sigma*:
fixes $I :: 'j \Rightarrow 'i \text{ set}$ **and** $J :: 'j \text{ set}$ **and** $E :: 'i \Rightarrow 'a \text{ set set}$
assumes *indep: indep-sets* E ($\bigcup j \in J. I j$)
assumes *Int-stable*: $\bigwedge i j. j \in J \Rightarrow i \in I j \Rightarrow \text{Int-stable } (E i)$
assumes *disjoint: disjoint-family-on* $I J$
shows *indep-sets* ($\lambda j. \text{sigma-sets } (\text{space } M) (\bigcup i \in I j. E i)$) J
proof –
let $?E = \lambda j. \{\bigcap k \in K. E' k \mid E' K. \text{finite } K \wedge K \neq \{\} \wedge K \subseteq I j \wedge (\forall k \in K. E'$
 $k \in E k)\}$

from *indep* **have** $E: \bigwedge j i. j \in J \Rightarrow i \in I j \Rightarrow E i \subseteq \text{events}$

```

unfolding indep-sets-def by auto
{ fix j
  let  $?S = \text{sigma-sets}(\text{space } M) (\bigcup_{i \in I} j. E\ i)$ 
  assume  $j \in J$ 
  from  $E[\text{OF this}]$  interpret  $S: \text{sigma-algebra space } M\ ?S$ 
  using sets.sets-into-space[of - M] by (intro sigma-algebra-sigma-sets) auto

  have  $\text{sigma-sets}(\text{space } M) (\bigcup_{i \in I} j. E\ i) = \text{sigma-sets}(\text{space } M) (?E\ j)$ 
  proof (rule sigma-sets-eqI)
    fix  $A$  assume  $A \in (\bigcup_{i \in I} j. E\ i)$ 
    then guess  $i ..$ 
    then show  $A \in \text{sigma-sets}(\text{space } M) (?E\ j)$ 
      by (auto intro!: sigma-sets.intros(2-) exI[of - {i}] exI[of -  $\lambda i. A$ ])
  next
    fix  $A$  assume  $A \in ?E\ j$ 
    then obtain  $E'\ K$  where  $\text{finite } K\ K \neq \{\}\ K \subseteq I\ j \wedge k. k \in K \implies E'\ k \in$ 
     $E\ k$ 
      and  $A: A = (\bigcap_{k \in K}. E'\ k)$ 
      by auto
    then have  $A \in ?S$  unfolding  $A$ 
      by (safe intro!: S.finite-INT) auto
    then show  $A \in \text{sigma-sets}(\text{space } M) (\bigcup_{i \in I} j. E\ i)$ 
      by simp
    qed }
  moreover have  $\text{indep-sets}(\lambda j. \text{sigma-sets}(\text{space } M) (?E\ j))\ J$ 
  proof (rule indep-sets-sigma)
    show  $\text{indep-sets } ?E\ J$ 
    proof (intro indep-setsI)
      fix  $j$  assume  $j \in J$  with  $E$  show  $?E\ j \subseteq \text{events}$  by (force intro!:
    sets.finite-INT)
    next
      fix  $K\ A$  assume  $K: K \neq \{\}\ K \subseteq J$  finite  $K$ 
      and  $\forall j \in K. A\ j \in ?E\ j$ 
      then have  $\forall j \in K. \exists E'\ L. A\ j = (\bigcap_{l \in L}. E'\ l) \wedge \text{finite } L \wedge L \neq \{\} \wedge L \subseteq I$ 
       $j \wedge (\forall l \in L. E'\ l \in E\ l)$ 
      by simp
      from bchoice[OF this] guess  $E' ..$ 
      from bchoice[OF this] obtain  $L$ 
      where  $A: \bigwedge j. j \in K \implies A\ j = (\bigcap_{l \in L} j. E'\ j\ l)$ 
      and  $L: \bigwedge j. j \in K \implies \text{finite } (L\ j) \wedge j. j \in K \implies L\ j \neq \{\} \wedge j. j \in K \implies L\ j$ 
       $\subseteq I\ j$ 
      and  $E': \bigwedge j\ l. j \in K \implies l \in L\ j \implies E'\ j\ l \in E\ l$ 
      by auto

    { fix  $k\ l\ j$  assume  $k \in K\ j \in K\ l \in L\ j\ l \in L\ k$ 
      have  $k = j$ 
      proof (rule ccontr)
        assume  $k \neq j$ 
        with disjoint  $\langle K \subseteq J \rangle \langle k \in K \rangle \langle j \in K \rangle$  have  $I\ k \cap I\ j = \{\}$ 

```

```

      unfolding disjoint-family-on-def by auto
      with L(2,3)[OF ⟨j ∈ K⟩] L(2,3)[OF ⟨k ∈ K⟩]
      show False using ⟨l ∈ L k⟩ ⟨l ∈ L j⟩ by auto
    qed }
  note L-inj = this

  define k where k l = (SOME k. k ∈ K ∧ l ∈ L k) for l
  { fix x j l assume *: j ∈ K l ∈ L j
    have k l = j unfolding k-def
    proof (rule some-equality)
      fix k assume k ∈ K ∧ l ∈ L k
      with * L-inj show k = j by auto
    qed (insert *, simp) }
  note k-simp[simp] = this
  let ?E' = λl. E' (k l) l
  have prob (∏j∈K. A j) = prob (∏l∈(∪k∈K. L k). ?E' l)
    by (auto simp: A intro!: arg-cong[where f=prob])
  also have ... = (∏l∈(∪k∈K. L k). prob (?E' l))
    using L K E' by (intro indep-setsD[OF indep]) (simp-all add: UN-mono)
  also have ... = (∏j∈K. ∏l∈L j. prob (E' j l))
    using K L L-inj by (subst prod.UNION-disjoint) auto
  also have ... = (∏j∈K. prob (A j))
    using K L E' by (auto simp add: A intro!: prod.cong indep-setsD[OF indep,
symmetric]) blast
  finally show prob (∏j∈K. A j) = (∏j∈K. prob (A j)) .
  qed
next
  fix j assume j ∈ J
  show Int-stable (?E j)
  proof (rule Int-stableI)
    fix a assume a ∈ ?E j then obtain Ka Ea
      where a: a = (∏k∈Ka. Ea k) finite Ka Ka ≠ {} Ka ⊆ I j ∧ k. k∈Ka ⇒
Ea k ∈ E k by auto
    fix b assume b ∈ ?E j then obtain Kb Eb
      where b: b = (∏k∈Kb. Eb k) finite Kb Kb ≠ {} Kb ⊆ I j ∧ k. k∈Kb ⇒
Eb k ∈ E k by auto
    let ?f = λk. (if k ∈ Ka ∩ Kb then Ea k ∩ Eb k else if k ∈ Kb then Eb k else
if k ∈ Ka then Ea k else {})
    have Ka ∪ Kb = (Ka ∩ Kb) ∪ (Kb - Ka) ∪ (Ka - Kb)
      by blast
    moreover have (∏x∈Ka ∩ Kb. Ea x ∩ Eb x) ∩
      (∏x∈Kb - Ka. Eb x) ∩ (∏x∈Ka - Kb. Ea x) = (∏k∈Ka. Ea k) ∩
(∏k∈Kb. Eb k)
      by auto
    ultimately have (∏k∈Ka ∪ Kb. ?f k) = (∏k∈Ka. Ea k) ∩ (∏k∈Kb. Eb
k) (is ?lhs = ?rhs)
      by (simp only: image-Un Inter-Un-distrib) simp
    then have a ∩ b = (∏k∈Ka ∪ Kb. ?f k)
      by (simp only: a(1) b(1))

```

```

    with a b ⟨j ∈ J⟩ Int-stableD[OF Int-stable] show a ∩ b ∈ ?E j
      by (intro CollectI exI[of - Ka ∪ Kb] exI[of - ?f]) auto
  qed
  qed
  ultimately show ?thesis
    by (simp cong: indep-sets-cong)
  qed

lemma (in prob-space) indep-vars-restrict:
  assumes ind: indep-vars M' X I and K:  $\bigwedge j. j \in L \implies K j \subseteq I$  and J:
  disjoint-family-on K L
  shows indep-vars ( $\lambda j. PiM (K j) M'$ ) ( $\lambda j \omega. restrict (\lambda i. X i \omega) (K j)$ ) L
  unfolding indep-vars-def
  proof safe
    fix j assume j ∈ L then show random-variable ( $Pi_M (K j) M'$ ) ( $\lambda \omega. \lambda i \in K j. X i \omega$ )
    using K ind by (auto simp: indep-vars-def intro!: measurable-restrict)
  next
    have X:  $\bigwedge i. i \in I \implies X i \in measurable M (M' i)$ 
      using ind by (auto simp: indep-vars-def)
    let ?proj =  $\lambda j S. \{(\lambda \omega. \lambda i \in K j. X i \omega) - ' A \cap space M \mid A. A \in S\}$ 
    let ?UN =  $\lambda j. sigma-sets (space M) (\bigcup i \in K j. \{ X i - ' A \cap space M \mid A. A \in sets (M' i) \})$ 
    show indep-sets ( $\lambda i. sigma-sets (space M) (?proj i (sets (Pi_M (K i) M')))$ ) L
    proof (rule indep-sets-mono-sets)
      fix j assume j: j ∈ L
      have sigma-sets (space M) (?proj j (sets (Pi_M (K j) M'))) =
        sigma-sets (space M) (sigma-sets (space M) (?proj j (prod-algebra (K j) M')))
      using j K X [THEN measurable-space] unfolding sets-PiM
      by (subst sigma-sets-vimage-commute) (auto simp add: Pi-iff)
      also have ... = sigma-sets (space M) (?proj j (prod-algebra (K j) M'))
      by (rule sigma-sets-sigma-sets-eq) auto
      also have ...  $\subseteq$  ?UN j
    proof (rule sigma-sets-mono, safe del: disjE elim!: prod-algebraE)
      fix J E assume J: finite J J  $\neq$  {}  $\vee$  K j = {} J  $\subseteq$  K j and E:  $\forall i. i \in J \implies E i \in sets (M' i)$ 
      show  $(\lambda \omega. \lambda i \in K j. X i \omega) - ' prod-emb (K j) M' J (Pi_E J E) \cap space M \in ?UN j$ 
      proof cases
        assume K j = {} with J show ?thesis
          by (auto simp add: sigma-sets-empty-eq prod-emb-def)
        next
          assume K j  $\neq$  {} with J have J  $\neq$  {}
            by auto
          { interpret sigma-algebra space M ?UN j
            by (rule sigma-algebra-sigma-sets) auto
            have  $\bigwedge A. (\bigwedge i. i \in J \implies A i \in ?UN j) \implies INTER J A \in ?UN j$ 
              using ⟨finite J⟩ ⟨J  $\neq$  {}⟩ by (rule finite-INT) blast }
            note INT = this
          }
      qed
    qed
  qed

```

```

from ⟨ $J \neq \{\}$ ⟩  $J K E$ [rule-format, THEN sets.sets-into-space]  $j$ 
have  $(\lambda\omega. \lambda i \in K j. X i \omega) - 'prod-emb (K j) M' J (Pi_E J E) \cap space M$ 
  =  $(\bigcap i \in J. X i - 'E i \cap space M)$ 
  apply (subst prod-emb-PiE[OF - ])
  apply auto []
  apply auto []
  apply (auto simp add: Pi-iff intro!: X[THEN measurable-space])
  apply (erule-tac x=i in ballE)
  apply auto
  done
also have  $\dots \in ?UN j$ 
  apply (rule INT)
  apply (rule sigma-sets.Basic)
  using  $\langle J \subseteq K j \rangle E$ 
  apply auto
  done
finally show ?thesis .
qed
qed
finally show sigma-sets (space M) (?proj j (sets (Pi_M (K j) M')))  $\subseteq$  ?UN j .
next
show indep-sets ?UN L
proof (rule indep-sets-collect-sigma)
  show indep-sets  $(\lambda i. \{X i - 'A \cap space M \mid A. A \in sets (M' i)\}) (\bigcup j \in L. K$ 
j)
  proof (rule indep-sets-mono-index)
    show indep-sets  $(\lambda i. \{X i - 'A \cap space M \mid A. A \in sets (M' i)\}) I$ 
      using ind unfolding indep-vars-def2 by auto
    show  $(\bigcup l \in L. K l) \subseteq I$ 
      using  $K$  by auto
  qed
next
fix  $l i$  assume  $l \in L i \in K l$ 
show Int-stable  $\{X i - 'A \cap space M \mid A. A \in sets (M' i)\}$ 
  apply (auto simp: Int-stable-def)
  apply (rule-tac x=A  $\cap Aa$  in exI)
  apply auto
  done
qed fact
qed
qed

```

lemma (*in prob-space*) *indep-var-restrict*:

assumes *ind: indep-vars $M' X I$ and $AB: A \cap B = \{\} A \subseteq I B \subseteq I$*

shows *indep-var $(Pi_M A M')$ $(\lambda\omega. restrict (\lambda i. X i \omega) A) (Pi_M B M')$ $(\lambda\omega. restrict (\lambda i. X i \omega) B)$*

proof –

have *:

```

    case-bool (PiM A M') (PiM B M') = (λb. PiM (case-bool A B b) M')
    case-bool (λω. λi∈A. X i ω) (λω. λi∈B. X i ω) = (λb ω. λi∈case-bool A B b.
X i ω)
  by (simp-all add: fun-eq-iff split: bool.split)
  show ?thesis
  unfolding indep-var-def * using AB
  by (intro indep-vars-restrict[OF ind]) (auto simp: disjoint-family-on-def split:
bool.split)
qed

```

```

lemma (in prob-space) indep-vars-subset:
  assumes indep-vars M' X I J ⊆ I
  shows indep-vars M' X J
  using assms unfolding indep-vars-def indep-sets-def
  by auto

```

```

lemma (in prob-space) indep-vars-cong:
  I = J ⇒ (∧i. i ∈ I ⇒ X i = Y i) ⇒ (∧i. i ∈ I ⇒ M' i = N' i) ⇒
indep-vars M' X I ⇔ indep-vars N' Y J
  unfolding indep-vars-def2 by (intro conj-cong indep-sets-cong) auto

```

```

definition (in prob-space) tail-events where
  tail-events A = (∩n. sigma-sets (space M) (UNION {n..} A))

```

```

lemma (in prob-space) tail-events-sets:
  assumes A: ∧i::nat. A i ⊆ events
  shows tail-events A ⊆ events
proof
  fix X assume X: X ∈ tail-events A
  let ?A = (∩n. sigma-sets (space M) (UNION {n..} A))
  from X have ∧n::nat. X ∈ sigma-sets (space M) (UNION {n..} A) by (auto
simp: tail-events-def)
  from this[of 0] have X ∈ sigma-sets (space M) (UNION UNIV A) by simp
  then show X ∈ events
  by induct (insert A, auto)
qed

```

```

lemma (in prob-space) sigma-algebra-tail-events:
  assumes ∧i::nat. sigma-algebra (space M) (A i)
  shows sigma-algebra (space M) (tail-events A)
  unfolding tail-events-def
proof (simp add: sigma-algebra-iff2, safe)
  let ?A = (∩n. sigma-sets (space M) (UNION {n..} A))
  interpret A: sigma-algebra space M A i for i by fact
  { fix X x assume X ∈ ?A x ∈ X
  then have ∧n. X ∈ sigma-sets (space M) (UNION {n..} A) by auto
  from this[of 0] have X ∈ sigma-sets (space M) (UNION UNIV A) by simp
  then have X ⊆ space M
  by induct (insert A.sets-into-space, auto)

```

```

with ⟨ $x \in X$ ⟩ show  $x \in \text{space } M$  by auto }
{ fix  $F :: \text{nat} \Rightarrow 'a \text{ set}$  and  $n$  assume  $\text{range } F \subseteq ?A$ 
  then show  $(\text{UNION UNIV } F) \in \text{sigma-sets } (\text{space } M) (\text{UNION } \{n..\} A)$ 
  by (intro sigma-sets.Union) auto }
qed (auto intro!: sigma-sets.Compl sigma-sets.Empty)

```

lemma (*in prob-space*) *kolmogorov-0-1-law*:

```

fixes  $A :: \text{nat} \Rightarrow 'a \text{ set set}$ 
assumes  $\bigwedge i::\text{nat}. \text{sigma-algebra } (\text{space } M) (A \ i)$ 
assumes indep:  $\text{indep-sets } A \ \text{UNIV}$ 
and  $X: X \in \text{tail-events } A$ 
shows  $\text{prob } X = 0 \vee \text{prob } X = 1$ 

```

proof –

```

have  $A: \bigwedge i. A \ i \subseteq \text{events}$ 
  using indep unfolding indep-sets-def by simp

```

```

let  $?D = \{D \in \text{events}. \text{prob } (X \cap D) = \text{prob } X * \text{prob } D\}$ 
interpret  $A: \text{sigma-algebra } \text{space } M \ A \ i$  for  $i$  by fact
interpret  $T: \text{sigma-algebra } \text{space } M \ \text{tail-events } A$ 
  by (rule sigma-algebra-tail-events) fact
have  $X \subseteq \text{space } M$  using T.space-closed X by auto

```

```

have  $X\text{-in}: X \in \text{events}$ 
  using tail-events-sets A X by auto

```

interpret $D: \text{dynkin-system } \text{space } M \ ?D$

proof (*rule dynkin-systemI*)

```

fix  $D$  assume  $D \in ?D$  then show  $D \subseteq \text{space } M$ 
  using sets.sets-into-space by auto

```

next

```

show  $\text{space } M \in ?D$ 
  using prob-space ⟨X ⊆ space M⟩ by (simp add: Int-absorb2)

```

next

```

fix  $A$  assume  $A: A \in ?D$ 
have  $\text{prob } (X \cap (\text{space } M - A)) = \text{prob } (X - (X \cap A))$ 
  using  $\langle X \subseteq \text{space } M \rangle$  by (auto intro!: arg-cong[where f=prob])
also have  $\dots = \text{prob } X - \text{prob } (X \cap A)$ 
  using  $X\text{-in } A$  by (intro finite-measure-Diff) auto
also have  $\dots = \text{prob } X * \text{prob } (\text{space } M) - \text{prob } X * \text{prob } A$ 
  using  $A \ \text{prob-space}$  by auto
also have  $\dots = \text{prob } X * \text{prob } (\text{space } M - A)$ 
  using  $X\text{-in } A$  sets.sets-into-space
  by (subst finite-measure-Diff) (auto simp: field-simps)
finally show  $\text{space } M - A \in ?D$ 
  using  $A \langle X \subseteq \text{space } M \rangle$  by auto

```

next

```

fix  $F :: \text{nat} \Rightarrow 'a \text{ set}$  assume dis: disjoint-family F and  $\text{range } F \subseteq ?D$ 
then have  $F: \text{range } F \subseteq \text{events} \bigwedge i. \text{prob } (X \cap F \ i) = \text{prob } X * \text{prob } (F \ i)$ 
  by auto

```

```

have (λi. prob (X ∩ F i)) sums prob (⋃i. X ∩ F i)
proof (rule finite-measure-UNION)
  show range (λi. X ∩ F i) ⊆ events
    using F X-in by auto
  show disjoint-family (λi. X ∩ F i)
    using dis by (rule disjoint-family-on-bisimulation) auto
qed
with F have (λi. prob X * prob (F i)) sums prob (X ∩ (⋃i. F i))
  by simp
moreover have (λi. prob X * prob (F i)) sums (prob X * prob (⋃i. F i))
  by (intro sums-mult finite-measure-UNION F dis)
ultimately have prob (X ∩ (⋃i. F i)) = prob X * prob (⋃i. F i)
  by (auto dest!: sums-unique)
with F show (⋃i. F i) ∈ ?D
  by auto
qed

{ fix n
  have indep-sets (λb. sigma-sets (space M) (⋃m∈case-bool {..n} {Suc n..} b.
A m)) UNIV
  proof (rule indep-sets-collect-sigma)
    have *: (⋃b. case b of True ⇒ {..n} | False ⇒ {Suc n..}) = UNIV (is ?U
= -)
      by (simp split: bool.split add: set-eq-iff) (metis not-less-eq-eq)
    with indep show indep-sets A ?U by simp
    show disjoint-family (case-bool {..n} {Suc n..})
      unfolding disjoint-family-on-def by (auto split: bool.split)
    fix m
    show Int-stable (A m)
      unfolding Int-stable-def using A.Int by auto
    qed
  also have (λb. sigma-sets (space M) (⋃m∈case-bool {..n} {Suc n..} b. A m))
=
  case-bool (sigma-sets (space M) (⋃m∈{..n}. A m)) (sigma-sets (space M)
(⋃m∈{Suc n..}. A m))
    by (auto intro!: ext split: bool.split)
    finally have indep: indep-set (sigma-sets (space M) (⋃m∈{..n}. A m))
(sigma-sets (space M) (⋃m∈{Suc n..}. A m))
      unfolding indep-set-def by simp

  have sigma-sets (space M) (⋃m∈{..n}. A m) ⊆ ?D
  proof (simp add: subset-eq, rule)
    fix D assume D: D ∈ sigma-sets (space M) (⋃m∈{..n}. A m)
    have X ∈ sigma-sets (space M) (⋃m∈{Suc n..}. A m)
      using X unfolding tail-events-def by simp
    from indep-setD[OF indep D this] indep-setD-ev1[OF indep] D
    show D ∈ events ∧ prob (X ∩ D) = prob X * prob D
      by (auto simp add: ac-simps)
    qed }

```



```

then have  $(\bigcup n. \text{sigma-sets } (\text{space } M) (\bigcup m \in \{..n\}. A \ m)) \subseteq ?D$  (is ?A  $\subseteq$  -)
by auto

note  $\langle X \in \text{tail-events } A \rangle$ 
also {
  have  $\bigwedge n. \text{sigma-sets } (\text{space } M) (\bigcup i \in \{n..\}. A \ i) \subseteq \text{sigma-sets } (\text{space } M) ?A$ 
    by (intro sigma-sets-subseteq UN-mono) auto
  then have  $\text{tail-events } A \subseteq \text{sigma-sets } (\text{space } M) ?A$ 
    unfolding tail-events-def by auto }
also have  $\text{sigma-sets } (\text{space } M) ?A = \text{dynkin } (\text{space } M) ?A$ 
proof (rule sigma-eq-dynkin)
  { fix  $B \ n$  assume  $B \in \text{sigma-sets } (\text{space } M) (\bigcup m \in \{..n\}. A \ m)$ 
    then have  $B \subseteq \text{space } M$ 
      by induct (insert A sets.sets-into-space[of - M], auto) }
  then show  $?A \subseteq \text{Pow } (\text{space } M)$  by auto
  show Int-stable ?A
  proof (rule Int-stableI)
    fix  $a$  assume  $a \in ?A$  then guess  $n$  .. note  $a = \text{this}$ 
    fix  $b$  assume  $b \in ?A$  then guess  $m$  .. note  $b = \text{this}$ 
    interpret  $\text{Amn}: \text{sigma-algebra space } M \text{ sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max \ m \ n\}. A \ i)$ 
    using  $A \ \text{sets.sets-into-space}[of - M]$  by (intro sigma-algebra-sigma-sets)
  auto
  have  $\text{sigma-sets } (\text{space } M) (\bigcup i \in \{..n\}. A \ i) \subseteq \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max \ m \ n\}. A \ i)$ 
    by (intro sigma-sets-subseteq UN-mono) auto
  with a have  $a \in \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max \ m \ n\}. A \ i)$  by auto
  moreover
  have  $\text{sigma-sets } (\text{space } M) (\bigcup i \in \{..m\}. A \ i) \subseteq \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max \ m \ n\}. A \ i)$ 
    by (intro sigma-sets-subseteq UN-mono) auto
  with b have  $b \in \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max \ m \ n\}. A \ i)$  by auto
  ultimately have  $a \cap b \in \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max \ m \ n\}. A \ i)$ 
    using  $\text{Amn.Int}[of a b]$  by simp
  then show  $a \cap b \in (\bigcup n. \text{sigma-sets } (\text{space } M) (\bigcup i \in \{..n\}. A \ i))$  by auto
  qed
qed
also have  $\text{dynkin } (\text{space } M) ?A \subseteq ?D$ 
  using  $\langle ?A \subseteq ?D \rangle$  by (auto intro!: D.dynkin-subset)
finally show  $?thesis$  by auto
qed

```

lemma *(in prob-space) borel-0-1-law:*

```

fixes  $F :: \text{nat} \Rightarrow 'a \ \text{set}$ 
assumes  $F2: \text{indep-events } F \ \text{UNIV}$ 
shows  $\text{prob } (\bigcap n. \bigcup m \in \{n..\}. F \ m) = 0 \vee \text{prob } (\bigcap n. \bigcup m \in \{n..\}. F \ m) = 1$ 
proof (rule kolmogorov-0-1-law[of  $\lambda i. \text{sigma-sets } (\text{space } M) \{ F \ i \}$ ])
  have  $F1: \text{range } F \subseteq \text{events}$ 
    using  $F2$  by (simp add: indep-events-def subset-eq)

```

```

{ fix i show sigma-algebra (space M) (sigma-sets (space M) {F i})
  using sigma-algebra-sigma-sets[of {F i} space M] F1 sets.sets-into-space
  by auto }
show indep-sets (λi. sigma-sets (space M) {F i}) UNIV
proof (rule indep-sets-sigma)
  show indep-sets (λi. {F i}) UNIV
  unfolding indep-events-def-alt[symmetric] by fact
  fix i show Int-stable {F i}
  unfolding Int-stable-def by simp
qed
let ?Q = λn. ∪ i∈{n..}. F i
show (∩ n. ∪ m∈{n..}. F m) ∈ tail-events (λi. sigma-sets (space M) {F i})
  unfolding tail-events-def
proof
  fix j
  interpret S: sigma-algebra space M sigma-sets (space M) (∪ i∈{j..}. sigma-sets
(space M) {F i})
  using order-trans[OF F1 sets.space-closed]
  by (intro sigma-algebra-sigma-sets) (simp add: sigma-sets-singleton subset-eq)
  have (∩ n. ?Q n) = (∩ n∈{j..}. ?Q n)
  by (intro decseq-SucI INT-decseq-offset UN-mono) auto
  also have ... ∈ sigma-sets (space M) (∪ i∈{j..}. sigma-sets (space M) {F i})
  using order-trans[OF F1 sets.space-closed]
  by (safe intro!: S.countable-INT S.countable-UN)
  (auto simp: sigma-sets-singleton intro!: sigma-sets.Basic bexI)
  finally show (∩ n. ?Q n) ∈ sigma-sets (space M) (∪ i∈{j..}. sigma-sets (space
M) {F i})
  by simp
qed
qed

```

lemma (in prob-space) borel-0-1-law-AE:

```

fixes P :: nat ⇒ 'a ⇒ bool
assumes indep-events (λm. {x∈space M. P m x}) UNIV (is indep-events ?P -)
shows (AE x in M. infinite {m. P m x}) ∨ (AE x in M. finite {m. P m x})
proof -
  have [measurable]: ∧m. {x∈space M. P m x} ∈ sets M
  using assms by (auto simp: indep-events-def)
  have *: (∩ n. ∪ m∈{n..}. {x ∈ space M. P m x}) ∈ events
  by simp
  from assms have prob (∩ n. ∪ m∈{n..}. ?P m) = 0 ∨ prob (∩ n. ∪ m∈{n..}.
?P m) = 1
  by (rule borel-0-1-law)
  also have prob (∩ n. ∪ m∈{n..}. ?P m) = 1 ⟷ (AE x in M. infinite {m. P
m x})
  using * by (simp add: prob-eq-1)
  (simp add: Bex-def infinite-nat-iff-unbounded-le)
  also have prob (∩ n. ∪ m∈{n..}. ?P m) = 0 ⟷ (AE x in M. finite {m. P m
x})

```

```

using * by (simp add: prob-eq-0)
  (auto simp add: Ball-def finite-nat-iff-bounded not-less [symmetric])
finally show ?thesis
  by blast
qed

lemma (in prob-space) indep-sets-finite:
  assumes I: I ≠ {} finite I
  and F:  $\bigwedge i. i \in I \implies F\ i \subseteq \text{events}$   $\bigwedge i. i \in I \implies \text{space } M \in F\ i$ 
  shows indep-sets F I  $\longleftrightarrow (\forall A \in \text{Pi } I\ F. \text{prob } (\bigcap_{j \in I}. A\ j) = (\prod_{j \in I}. \text{prob } (A\ j)))$ 
proof
  assume *: indep-sets F I
  from I show  $\forall A \in \text{Pi } I\ F. \text{prob } (\bigcap_{j \in I}. A\ j) = (\prod_{j \in I}. \text{prob } (A\ j))$ 
    by (intro indep-setsD[OF *] ballI) auto
next
  assume indep:  $\forall A \in \text{Pi } I\ F. \text{prob } (\bigcap_{j \in I}. A\ j) = (\prod_{j \in I}. \text{prob } (A\ j))$ 
  show indep-sets F I
  proof (rule indep-setsI[OF F(1)])
    fix A J assume J: J ≠ {} J  $\subseteq$  I finite J
    assume A:  $\forall j \in J. A\ j \in F\ j$ 
    let ?A =  $\lambda j. \text{if } j \in J \text{ then } A\ j \text{ else space } M$ 
    have  $\text{prob } (\bigcap_{j \in I}. ?A\ j) = \text{prob } (\bigcap_{j \in J}. A\ j)$ 
      using subset-trans[OF F(1) sets.space-closed] J A
      by (auto intro!: arg-cong[where f=prob] split: if-split-asm) blast
    also
    from A F have  $(\lambda j. \text{if } j \in J \text{ then } A\ j \text{ else space } M) \in \text{Pi } I\ F$  (is ?A  $\in$  -)
      by (auto split: if-split-asm)
    with indep have  $\text{prob } (\bigcap_{j \in I}. ?A\ j) = (\prod_{j \in I}. \text{prob } (?A\ j))$ 
      by auto
    also have ... =  $(\prod_{j \in J}. \text{prob } (A\ j))$ 
      unfolding if-distrib prod.If-cases[OF ⟨finite I⟩]
      using prob-space ⟨J  $\subseteq$  I⟩ by (simp add: Int-absorb1 prod.neutral-const)
    finally show  $\text{prob } (\bigcap_{j \in J}. A\ j) = (\prod_{j \in J}. \text{prob } (A\ j))$  ..
  qed
qed

lemma (in prob-space) indep-vars-finite:
  fixes I :: 'i set
  assumes I: I ≠ {} finite I
  and M':  $\bigwedge i. i \in I \implies \text{sets } (M'\ i) = \text{sigma-sets } (\text{space } (M'\ i)) (E\ i)$ 
  and rv:  $\bigwedge i. i \in I \implies \text{random-variable } (M'\ i) (X\ i)$ 
  and Int-stable:  $\bigwedge i. i \in I \implies \text{Int-stable } (E\ i)$ 
  and space:  $\bigwedge i. i \in I \implies \text{space } (M'\ i) \in E\ i$  and closed:  $\bigwedge i. i \in I \implies E\ i \subseteq$ 
  Pow (space (M' i))
  shows indep-vars M' X I  $\longleftrightarrow$ 
  ( $\forall A \in (\text{Pi } i \in I. E\ i). \text{prob } (\bigcap_{j \in I}. X\ j - 'A\ j \cap \text{space } M) = (\prod_{j \in I}. \text{prob } (X\ j - 'A\ j \cap \text{space } M))$ )
proof -

```

```

from rv have  $X: \bigwedge i. i \in I \implies X i \in \text{space } M \rightarrow \text{space } (M' i)$ 
  unfolding measurable-def by simp

{ fix i assume  $i \in I$ 
  from closed[OF  $\langle i \in I \rangle$ ]
  have sigma-sets (space M)  $\{X i -' A \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}$ 
    = sigma-sets (space M)  $\{X i -' A \cap \text{space } M \mid A. A \in E i\}$ 
  unfolding sigma-sets-vimage-commute[OF X, OF  $\langle i \in I \rangle$ , symmetric] M'[OF
 $\langle i \in I \rangle$ ]
  by (subst sigma-sets-sigma-sets-eq) auto }
note sigma-sets-X = this

{ fix i assume  $i \in I$ 
  have Int-stable  $\{X i -' A \cap \text{space } M \mid A. A \in E i\}$ 
  proof (rule Int-stableI)
    fix a assume  $a \in \{X i -' A \cap \text{space } M \mid A. A \in E i\}$ 
    then obtain A where  $a = X i -' A \cap \text{space } M$   $A \in E i$  by auto
    moreover
    fix b assume  $b \in \{X i -' A \cap \text{space } M \mid A. A \in E i\}$ 
    then obtain B where  $b = X i -' B \cap \text{space } M$   $B \in E i$  by auto
    moreover
    have  $(X i -' A \cap \text{space } M) \cap (X i -' B \cap \text{space } M) = X i -' (A \cap B) \cap$ 
space M by auto
    moreover note Int-stable[OF  $\langle i \in I \rangle$ ]
    ultimately
    show  $a \cap b \in \{X i -' A \cap \text{space } M \mid A. A \in E i\}$ 
    by (auto simp del: vimage-Int intro!: exI[of - A \cap B] dest: Int-stableD)
    qed }
note indep-sets-X = indep-sets-sigma-sets-iff[OF this]

{ fix i assume  $i \in I$ 
  { fix A assume  $A \in E i$ 
    with M'[OF  $\langle i \in I \rangle$ ] have  $A \in \text{sets } (M' i)$  by auto
    moreover
    from rv[OF  $\langle i \in I \rangle$ ] have  $X i \in \text{measurable } M (M' i)$  by auto
    ultimately
    have  $X i -' A \cap \text{space } M \in \text{sets } M$  by (auto intro: measurable-sets) }
  with X[OF  $\langle i \in I \rangle$ ] space[OF  $\langle i \in I \rangle$ ]
  have  $\{X i -' A \cap \text{space } M \mid A. A \in E i\} \subseteq \text{events}$ 
    space M  $\in \{X i -' A \cap \text{space } M \mid A. A \in E i\}$ 
    by (auto intro!: exI[of - space (M' i)]) }
note indep-sets-finite-X = indep-sets-finite[OF I this]

have  $(\forall A \in \Pi i \in I. \{X i -' A \cap \text{space } M \mid A. A \in E i\}. \text{prob } (INTER I A) =$ 
 $(\prod j \in I. \text{prob } (A j))) =$ 
 $(\forall A \in \Pi i \in I. E i. \text{prob } ((\bigcap j \in I. X j -' A j) \cap \text{space } M) = (\prod x \in I. \text{prob } (X x$ 
 $-' A x \cap \text{space } M)))$ 
  (is ?L = ?R)
proof safe

```

fix A **assume** $?L$ **and** $A: A \in (\prod_{i \in I}. E\ i)$
from $\langle ?L \rangle [THEN\ bspec, of\ \lambda i. X\ i - ' A\ i \cap\ space\ M] A\ \langle I \neq \{\} \rangle$
show $prob\ ((\bigcap_{j \in I}. X\ j - ' A\ j) \cap\ space\ M) = (\prod_{x \in I}. prob\ (X\ x - ' A\ x \cap\ space\ M))$
by $(auto\ simp\ add: Pi\text{-}iff)$
next
fix A **assume** $?R$ **and** $A: A \in (\prod_{i \in I}. \{X\ i - ' A \cap\ space\ M \mid A. A \in E\ i\})$
from A **have** $\forall i \in I. \exists B. A\ i = X\ i - ' B \cap\ space\ M \wedge B \in E\ i$ **by** $auto$
from $bchoice[OF\ this]$ **obtain** B **where** $B: \forall i \in I. A\ i = X\ i - ' B\ i \cap\ space\ M$
 $B \in (\prod_{i \in I}. E\ i)$ **by** $auto$
from $\langle ?R \rangle [THEN\ bspec, OF\ B(2)] B(1)\ \langle I \neq \{\} \rangle$
show $prob\ (INTER\ I\ A) = (\prod_{j \in I}. prob\ (A\ j))$
by $simp$
qed
then **show** $?thesis$ **using** $\langle I \neq \{\} \rangle$
by $(simp\ add: rv\ indep\ vars\ def\ indep\ sets\ X\ sigma\ sets\ X\ indep\ sets\ finite\ X$
 $cong: indep\ sets\ cong)$
qed

lemma $(in\ prob\ space)\ indep\ vars\ compose:$

assumes $indep\ vars\ M'\ X\ I$
assumes $rv: \bigwedge i. i \in I \implies Y\ i \in measurable\ (M'\ i)\ (N\ i)$
shows $indep\ vars\ N\ (\lambda i. Y\ i \circ X\ i)\ I$
unfolding $indep\ vars\ def$

proof

from $rv\ \langle indep\ vars\ M'\ X\ I \rangle$
show $\forall i \in I. random\ variable\ (N\ i)\ (Y\ i \circ X\ i)$
by $(auto\ simp: indep\ vars\ def)$

have $indep\ sets\ (\lambda i. sigma\ sets\ (space\ M)\ \{X\ i - ' A \cap\ space\ M \mid A. A \in sets\ (M'\ i)\})\ I$

using $\langle indep\ vars\ M'\ X\ I \rangle$ **by** $(simp\ add: indep\ vars\ def)$

then **show** $indep\ sets\ (\lambda i. sigma\ sets\ (space\ M)\ \{(Y\ i \circ X\ i) - ' A \cap\ space\ M \mid A. A \in sets\ (N\ i)\})\ I$

proof $(rule\ indep\ sets\ mono\ sets)$

fix i **assume** $i \in I$

with $\langle indep\ vars\ M'\ X\ I \rangle$ **have** $X: X\ i \in space\ M \rightarrow space\ (M'\ i)$

unfolding $indep\ vars\ def\ measurable\ def$ **by** $auto$

{ fix A **assume** $A \in sets\ (N\ i)$

then **have** $\exists B. (Y\ i \circ X\ i) - ' A \cap\ space\ M = X\ i - ' B \cap\ space\ M \wedge B \in sets\ (M'\ i)$

by $(intro\ exI[of\ -\ Y\ i - ' A \cap\ space\ (M'\ i)])$

$(auto\ simp: vimage\ comp\ intro!: measurable\ sets\ rv\ \langle i \in I \rangle\ funcset\ mem[OF\ X])$ **}**

then **show** $sigma\ sets\ (space\ M)\ \{(Y\ i \circ X\ i) - ' A \cap\ space\ M \mid A. A \in sets\ (N\ i)\} \subseteq$

$sigma\ sets\ (space\ M)\ \{X\ i - ' A \cap\ space\ M \mid A. A \in sets\ (M'\ i)\}$

by $(intro\ sigma\ sets\ subseteq)\ (auto\ simp: vimage\ comp)$

qed

qed

lemma (in *prob-space*) *indep-vars-compose2*:

assumes *indep-vars* $M' X I$

assumes *rv*: $\bigwedge i. i \in I \implies Y i \in \text{measurable } (M' i) (N i)$

shows *indep-vars* $N (\lambda i x. Y i (X i x)) I$

using *indep-vars-compose* [*OF assms*] **by** (*simp add: comp-def*)

lemma (in *prob-space*) *indep-var-compose*:

assumes *indep-var* $M1 X1 M2 X2 Y1 \in \text{measurable } M1 N1 Y2 \in \text{measurable } M2 N2$

shows *indep-var* $N1 (Y1 \circ X1) N2 (Y2 \circ X2)$

proof –

have *indep-vars* (*case-bool* $N1 N2$) ($\lambda b. \text{case-bool } Y1 Y2 b \circ \text{case-bool } X1 X2 b$)
UNIV

using *assms*

by (*intro indep-vars-compose*[**where** $M' = \text{case-bool } M1 M2$])

(*auto simp: indep-var-def split: bool.split*)

also have ($\lambda b. \text{case-bool } Y1 Y2 b \circ \text{case-bool } X1 X2 b = \text{case-bool } (Y1 \circ X1)$
 $(Y2 \circ X2)$)

by (*simp add: fun-eq-iff split: bool.split*)

finally show *?thesis*

unfolding *indep-var-def* .

qed

lemma (in *prob-space*) *indep-vars-Min*:

fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$

assumes *I*: *finite* $I i \notin I$ **and** *indep*: *indep-vars* $(\lambda \cdot. \text{borel}) X (\text{insert } i I)$

shows *indep-var* $\text{borel } (X i) \text{ borel } (\lambda \omega. \text{Min } ((\lambda i. X i \omega)'I))$

proof –

have *indep-var*

$\text{borel } ((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\}))$

$\text{borel } ((\lambda f. \text{Min } (f'I)) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I))$

using *I* **by** (*intro indep-var-compose*[*OF indep-var-restrict*[*OF indep*]] *borel-measurable-Min*)

auto

also have $((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\})) = X i$

by *auto*

also have $((\lambda f. \text{Min } (f'I)) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I)) = (\lambda \omega. \text{Min } ((\lambda i. X i \omega)'I))$

by (*auto cong: rev-conj-cong*)

finally show *?thesis*

unfolding *indep-var-def* .

qed

lemma (in *prob-space*) *indep-vars-sum*:

fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$

assumes *I*: *finite* $I i \notin I$ **and** *indep*: *indep-vars* $(\lambda \cdot. \text{borel}) X (\text{insert } i I)$

shows *indep-var* $\text{borel } (X i) \text{ borel } (\lambda \omega. \sum_{i \in I}. X i \omega)$

proof –

have *indep-var*
 borel $((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\}))$
 borel $((\lambda f. \sum_{i \in I}. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I))$
using *I* **by** $(\text{intro indep-var-compose}[\text{OF indep-var-restrict}[\text{OF indep}]])$ *auto*
also have $((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\})) = X i$
by *auto*
also have $((\lambda f. \sum_{i \in I}. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I)) = (\lambda \omega. \sum_{i \in I}. X i \omega)$
by $(\text{auto cong: rev-conj-cong})$
finally show *?thesis* .
qed

lemma (in prob-space) *indep-vars-prod*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$
assumes *I*: *finite I* $i \notin I$ **and** *indep*: *indep-vars* $(\lambda. \text{borel}) X (\text{insert } i I)$
shows *indep-var borel* $(X i)$ *borel* $(\lambda \omega. \prod_{i \in I}. X i \omega)$
proof –
have *indep-var*
 borel $((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\}))$
 borel $((\lambda f. \prod_{i \in I}. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I))$
using *I* **by** $(\text{intro indep-var-compose}[\text{OF indep-var-restrict}[\text{OF indep}]])$ *auto*
also have $((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\})) = X i$
by *auto*
also have $((\lambda f. \prod_{i \in I}. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I)) = (\lambda \omega. \prod_{i \in I}. X i \omega)$
by $(\text{auto cong: rev-conj-cong})$
finally show *?thesis* .
qed

lemma (in prob-space) *indep-varsD-finite*:
assumes *X*: *indep-vars* $M' X I$
assumes *I*: $I \neq \{\}$ *finite I* $\bigwedge i. i \in I \implies A i \in \text{sets } (M' i)$
shows *prob* $(\bigcap_{i \in I}. X i - 'A i \cap \text{space } M) = (\prod_{i \in I}. \text{prob } (X i - 'A i \cap \text{space } M))$
proof $(\text{rule indep-setsD})$
show *indep-sets* $(\lambda i. \text{sigma-sets } (\text{space } M) \{X i - 'A i \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}) I$
using *X* **by** $(\text{auto simp: indep-vars-def})$
show $I \subseteq I$ $I \neq \{\}$ *finite I* **using** *I* **by** *auto*
show $\forall i \in I. X i - 'A i \cap \text{space } M \in \text{sigma-sets } (\text{space } M) \{X i - 'A i \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}$
using *I* **by** *auto*
qed

lemma (in prob-space) *indep-varsD*:
assumes *X*: *indep-vars* $M' X I$
assumes *I*: $J \neq \{\}$ *finite J* $J \subseteq I$ $\bigwedge i. i \in J \implies A i \in \text{sets } (M' i)$
shows *prob* $(\bigcap_{i \in J}. X i - 'A i \cap \text{space } M) = (\prod_{i \in J}. \text{prob } (X i - 'A i \cap \text{space } M))$
proof $(\text{rule indep-setsD})$

```

show indep-sets ( $\lambda i.$  sigma-sets (space  $M$ )  $\{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}$ )  $I$ 
using  $X$  by (auto simp: indep-vars-def)
show  $\forall i \in J. X\ i - 'A\ i \cap \text{space } M \in \text{sigma-sets } (\text{space } M) \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}$ 
using  $I$  by auto
qed fact+

```

lemma (in prob-space) indep-vars-iff-distr-eq-PiM:

```

fixes  $I :: 'i$  set and  $X :: 'i \Rightarrow 'a \Rightarrow 'b$ 
assumes  $I \neq \{\}$ 
assumes  $rv: \bigwedge i. \text{random-variable } (M' i) (X\ i)$ 
shows indep-vars  $M' X I \longleftrightarrow$ 
 $\text{distr } M (\prod_M i \in I. M' i) (\lambda x. \lambda i \in I. X\ i\ x) = (\prod_M i \in I. \text{distr } M (M' i) (X\ i))$ 

```

proof –

```

let  $?P = \prod_M i \in I. M' i$ 
let  $?X = \lambda x. \lambda i \in I. X\ i\ x$ 
let  $?D = \text{distr } M\ ?P\ ?X$ 
have  $X: \text{random-variable } ?P\ ?X$  by (intro measurable-restrict  $rv$ )
interpret  $D: \text{prob-space } ?D$  by (intro prob-space-distr  $X$ )

```

```

let  $?D' = \lambda i. \text{distr } M (M' i) (X\ i)$ 
let  $?P' = \prod_M i \in I. \text{distr } M (M' i) (X\ i)$ 
interpret  $D': \text{prob-space } ?D' i$  for  $i$  by (intro prob-space-distr  $rv$ )
interpret  $P: \text{product-prob-space } ?D' I ..$ 

```

show ?thesis

proof

```

assume indep-vars  $M' X I$ 
show  $?D = ?P'$ 
proof (rule measure-eqI-generator-eq)
show Int-stable (prod-algebra  $I M'$ )
by (rule Int-stable-prod-algebra)
show prod-algebra  $I M' \subseteq \text{Pow } (\text{space } ?P)$ 
using prod-algebra-sets-into-space by (simp add: space-PiM)
show sets  $?D = \text{sigma-sets } (\text{space } ?P) (\text{prod-algebra } I M')$ 
by (simp add: sets-PiM space-PiM)
show sets  $?P' = \text{sigma-sets } (\text{space } ?P) (\text{prod-algebra } I M')$ 
by (simp add: sets-PiM space-PiM cong: prod-algebra-cong)
let  $?A = \lambda i. \prod_E i \in I. \text{space } (M' i)$ 
show range  $?A \subseteq \text{prod-algebra } I M' (\bigcup i. ?A\ i) = \text{space } (\text{Pi}_M I M')$ 
by (auto simp: space-PiM intro!: space-in-prod-algebra cong: prod-algebra-cong)
{ fix } i show emeasure  $?D (\prod_E i \in I. \text{space } (M' i)) \neq \infty$  by auto }
next
fix  $E$  assume  $E: E \in \text{prod-algebra } I M'$ 
from prod-algebraE[OF  $E$ ] guess  $J Y .$  note  $J = \text{this}$ 

```

```

from  $E$  have  $E \in \text{sets } ?P$  by (auto simp: sets-PiM)
then have emeasure  $?D\ E = \text{emeasure } M\ (?X - 'E \cap \text{space } M)$ 

```



```

    by (simp add: emeasure-distr X)
  also have ?X -‘ E ∩ space M = (∏i∈J. X i -‘ Y i ∩ space M)
    using J ⟨I ≠ {}⟩ measurable-space[OF rv] by (auto simp: prod-emb-def
PiE-iff split: if-split-asm)
  also have emeasure M (∏i∈J. X i -‘ Y i ∩ space M) = (∏i∈J. emeasure
M (X i -‘ Y i ∩ space M))
    using ⟨indep-vars M' X I⟩ J ⟨I ≠ {}⟩ using indep-varsD[of M' X I J]
  by (auto simp: emeasure-eq-measure prod-ennreal measure-nonneg prod-nonneg)
  also have ... = (∏i∈J. emeasure (?D' i) (Y i))
    using rv J by (simp add: emeasure-distr)
  also have ... = emeasure ?P' E
    using P.emeasure-PiM-emb[of J Y] J by (simp add: prod-emb-def)
  finally show emeasure ?D E = emeasure ?P' E .
qed
next
assume ?D = ?P'
show indep-vars M' X I unfolding indep-vars-def
proof (intro conjI indep-setsI ballI rv)
  fix i show sigma-sets (space M) {X i -‘ A ∩ space M | A. A ∈ sets (M' i)}
⊆ events
    by (auto intro!: sets.sigma-sets-subset measurable-sets rv)
next
fix J Y' assume J: J ≠ {} J ⊆ I finite J
assume Y': ∀j∈J. Y' j ∈ sigma-sets (space M) {X j -‘ A ∩ space M | A.
A ∈ sets (M' j)}
have ∀j∈J. ∃Y. Y' j = X j -‘ Y ∩ space M ∧ Y ∈ sets (M' j)
proof
  fix j assume j ∈ J
  from Y'[rule-format, OF this] rv[of j]
  show ∃Y. Y' j = X j -‘ Y ∩ space M ∧ Y ∈ sets (M' j)
    by (subst (asm) sigma-sets-vimage-commute[symmetric, of - - space (M'
j)])
      (auto dest: measurable-space simp: sets.sigma-sets-eq)
qed
from bchoice[OF this] obtain Y where
  Y: ∧j. j ∈ J ⇒ Y' j = X j -‘ Y j ∩ space M ∧ j. j ∈ J ⇒ Y j ∈ sets
(M' j) by auto
let ?E = prod-emb I M' J (PiE J Y)
from Y have (∏j∈J. Y' j) = ?X -‘ ?E ∩ space M
  using J ⟨I ≠ {}⟩ measurable-space[OF rv] by (auto simp: prod-emb-def
PiE-iff split: if-split-asm)
then have emeasure M (∏j∈J. Y' j) = emeasure M (?X -‘ ?E ∩ space
M)
  by simp
also have ... = emeasure ?D ?E
  using Y J by (intro emeasure-distr[symmetric] X sets-PiM-I) auto
also have ... = emeasure ?P' ?E
  using ⟨?D = ?P'⟩ by simp
also have ... = (∏i∈J. emeasure (?D' i) (Y i))

```

```

    using P.emeasure-PiM-emb[of J Y] J Y by (simp add: prod-emb-def)
  also have ... = (∏ i∈J. emeasure M (Y' i))
    using rv J Y by (simp add: emeasure-distr)
  finally have emeasure M (∏ j∈J. Y' j) = (∏ i∈J. emeasure M (Y' i)) .
  then show prob (∏ j∈J. Y' j) = (∏ i∈J. prob (Y' i))
  by (auto simp: emeasure-eq-measure prod-ennreal measure-nonneg prod-nonneg)
qed
qed
qed

```

lemma (in *prob-space*) *indep-varD*:

```

  assumes indep: indep-var Ma A Mb B
  assumes sets: Xa ∈ sets Ma Xb ∈ sets Mb
  shows prob ((λx. (A x, B x)) -‘ (Xa × Xb) ∩ space M) =
    prob (A -‘ Xa ∩ space M) * prob (B -‘ Xb ∩ space M)
proof -
  have prob ((λx. (A x, B x)) -‘ (Xa × Xb) ∩ space M) =
    prob (∏ i∈UNIV. (case-bool A B i -‘ case-bool Xa Xb i ∩ space M))
  by (auto intro!: arg-cong[where f=prob] simp: UNIV-bool)
  also have ... = (∏ i∈UNIV. prob (case-bool A B i -‘ case-bool Xa Xb i ∩ space
M))
  using indep unfolding indep-var-def
  by (rule indep-varsD) (auto split: bool.split intro: sets)
  also have ... = prob (A -‘ Xa ∩ space M) * prob (B -‘ Xb ∩ space M)
  unfolding UNIV-bool by simp
  finally show ?thesis .
qed

```

lemma (in *prob-space*) *prob-indep-random-variable*:

```

  assumes ind[simp]: indep-var N X N Y
  assumes [simp]: A ∈ sets N B ∈ sets N
  shows P(x in M. X x ∈ A ∧ Y x ∈ B) = P(x in M. X x ∈ A) * P(x in M. Y
x ∈ B)
proof -
  have P(x in M. (X x)∈A ∧ (Y x)∈B) = prob ((λx. (X x, Y x)) -‘ (A ×
B) ∩ space M)
  by (auto intro!: arg-cong[where f=prob])
  also have ... = prob (X -‘ A ∩ space M) * prob (Y -‘ B ∩ space M)
  by (auto intro!: indep-varD[where Ma=N and Mb=N])
  also have ... = P(x in M. X x ∈ A) * P(x in M. Y x ∈ B)
  by (auto intro!: arg-cong2[where f=op *] arg-cong[where f=prob])
  finally show ?thesis .
qed

```

lemma (in *prob-space*)

```

  assumes indep-var S X T Y
  shows indep-var-rv1: random-variable S X
  and indep-var-rv2: random-variable T Y
proof -

```

```

have  $\forall i \in UNIV$ . random-variable (case-bool  $S$   $T$   $i$ ) (case-bool  $X$   $Y$   $i$ )
  using assms unfolding indep-var-def indep-vars-def by auto
then show random-variable  $S$   $X$  random-variable  $T$   $Y$ 
  unfolding UNIV-bool by auto
qed

```

lemma (in *prob-space*) *indep-var-distribution-eq*:

```

indep-var  $S$   $X$   $T$   $Y$   $\longleftrightarrow$  random-variable  $S$   $X$   $\wedge$  random-variable  $T$   $Y$   $\wedge$ 
  distr  $M$   $S$   $X$   $\otimes_M$  distr  $M$   $T$   $Y$  = distr  $M$  ( $S$   $\otimes_M$   $T$ ) ( $\lambda x$ . ( $X$   $x$ ,  $Y$   $x$ )) (is -
 $\longleftrightarrow$  -  $\wedge$  -  $\wedge$   $?S$   $\otimes_M$   $?T$  =  $?J$ )

```

proof *safe*

```

assume indep-var  $S$   $X$   $T$   $Y$ 
then show rvs: random-variable  $S$   $X$  random-variable  $T$   $Y$ 
  by (blast dest: indep-var-rv1 indep-var-rv2)+
then have  $XY$ : random-variable ( $S$   $\otimes_M$   $T$ ) ( $\lambda x$ . ( $X$   $x$ ,  $Y$   $x$ ))
  by (rule measurable-Pair)

```

interpret X : *prob-space* $?S$ **by** (*rule prob-space-distr*) *fact*

interpret Y : *prob-space* $?T$ **by** (*rule prob-space-distr*) *fact*

interpret XY : *pair-prob-space* $?S$ $?T$..

show $?S$ \otimes_M $?T$ = $?J$

proof (*rule pair-measure-eqI*)

show *sigma-finite-measure* $?S$..

show *sigma-finite-measure* $?T$..

fix A B **assume** A : $A \in$ *sets* $?S$ **and** B : $B \in$ *sets* $?T$

have *emeasure* $?J$ ($A \times B$) = *emeasure* M ($(\lambda x$. (X x , Y x)) -‘ ($A \times B$) \cap *space* M)

using A B **by** (*intro emeasure-distr[OF XY]*) *auto*

also have ... = *emeasure* M (X -‘ $A \cap$ *space* M) * *emeasure* M (Y -‘ $B \cap$ *space* M)

using *indep-varD[OF (indep-var S X T Y), of A B]* A B

by (*simp add: emeasure-eq-measure measure-nonneg ennreal-mult*)

also have ... = *emeasure* $?S$ A * *emeasure* $?T$ B

using *rvs* A B **by** (*simp add: emeasure-distr*)

finally show *emeasure* $?S$ A * *emeasure* $?T$ B = *emeasure* $?J$ ($A \times B$) **by**

simp

qed *simp*

next

assume *rvs*: *random-variable* S X *random-variable* T Y

then have XY : *random-variable* (S \otimes_M T) (λx . (X x , Y x))

by (*rule measurable-Pair*)

let $?S$ = *distr* M S X **and** $?T$ = *distr* M T Y

interpret X : *prob-space* $?S$ **by** (*rule prob-space-distr*) *fact*

interpret Y : *prob-space* $?T$ **by** (*rule prob-space-distr*) *fact*

interpret XY : *pair-prob-space* $?S$ $?T$..

assume $?S$ \otimes_M $?T$ = $?J$

```

{ fix S and X
  have Int-stable {X -‘ A ∩ space M | A. A ∈ sets S}
  proof (safe intro!: Int-stableI)
    fix A B assume A ∈ sets S B ∈ sets S
    then show ∃ C. (X -‘ A ∩ space M) ∩ (X -‘ B ∩ space M) = (X -‘ C ∩
space M) ∧ C ∈ sets S
      by (intro exI[of - A ∩ B]) auto
    qed }
  note Int-stable = this

show indep-var S X T Y unfolding indep-var-eq
proof (intro conjI indep-set-sigma-sets Int-stable rvs)
  show indep-set {X -‘ A ∩ space M | A. A ∈ sets S} {Y -‘ A ∩ space M | A.
A ∈ sets T}
  proof (safe intro!: indep-setI)
    { fix A assume A ∈ sets S then show X -‘ A ∩ space M ∈ sets M
      using ⟨X ∈ measurable M S⟩ by (auto intro: measurable-sets) }
    { fix A assume A ∈ sets T then show Y -‘ A ∩ space M ∈ sets M
      using ⟨Y ∈ measurable M T⟩ by (auto intro: measurable-sets) }
  next
    fix A B assume ab: A ∈ sets S B ∈ sets T
    then have prob ((X -‘ A ∩ space M) ∩ (Y -‘ B ∩ space M)) = emeasure
?J (A × B)
      using XY by (auto simp add: emeasure-distr emeasure-eq-measure measure-nonneg
intro!: arg-cong[where f=prob])
    also have ... = emeasure (?S ⊗M ?T) (A × B)
      unfolding ⟨?S ⊗M ?T = ?J⟩ ..
    also have ... = emeasure ?S A * emeasure ?T B
      using ab by (simp add: Y.emeasure-pair-measure-Times)
    finally show prob ((X -‘ A ∩ space M) ∩ (Y -‘ B ∩ space M)) =
      prob (X -‘ A ∩ space M) * prob (Y -‘ B ∩ space M)
      using rvs ab by (simp add: emeasure-eq-measure emeasure-distr measure-nonneg
ennreal-mult[symmetric])
    qed
  qed
qed

```

lemma (in prob-space) distributed-joint-indep:

```

assumes S: sigma-finite-measure S and T: sigma-finite-measure T
assumes X: distributed M S X Px and Y: distributed M T Y Py
assumes indep: indep-var S X T Y
shows distributed M (S ⊗M T) (λx. (X x, Y x)) (λ(x, y). Px x * Py y)
using indep-var-distribution-eq[of S X T Y] indep
by (intro distributed-joint-indep'[OF S T X Y]) auto

```

lemma (in prob-space) indep-vars-nn-integral:

```

assumes I: finite I indep-vars (λ-. borel) X I ∧ i ω. i ∈ I ⇒ 0 ≤ X i ω
shows (∫+ω. (∏ i ∈ I. X i ω) ∂M) = (∏ i ∈ I. ∫+ω. X i ω ∂M)

```

proof *cases*

assume $I \neq \{\}$

define Y **where** $[abs-def]: Y\ i\ \omega = (if\ i \in I\ then\ X\ i\ \omega\ else\ 0)$ **for** $i\ \omega$

{ fix i **have** $i \in I \implies random-variable\ borel\ (X\ i)$

using $I(2)$ **by** $(cases\ i \in I)$ $(auto\ simp: indep-vars-def)$ **}**

note $rv-X = this$

{ fix i **have** $random-variable\ borel\ (Y\ i)$

using $I(2)$ **by** $(cases\ i \in I)$ $(auto\ simp: Y-def\ rv-X)$ **}**

note $rv-Y = this[measurable]$

interpret Y : *prob-space distr M borel (Y i) for i*

using $I(2)$ **by** $(cases\ i \in I)$ $(auto\ intro!: prob-space-distr\ simp: indep-vars-def\ prob-space-return)$

interpret *product-sigma-finite* $\lambda i. distr\ M\ borel\ (Y\ i)$

..

have $indep-Y: indep-vars\ (\lambda i. borel)\ Y\ I$

by $(rule\ indep-vars-cong[THEN\ iffD1, OF\ - - -\ I(2)])\ (auto\ simp: Y-def)$

have $(\int^{+\omega}. (\prod_{i \in I}. X\ i\ \omega)\ \partial M) = (\int^{+\omega}. (\prod_{i \in I}. Y\ i\ \omega)\ \partial M)$

using $I(3)$ **by** $(auto\ intro!: nn-integral-cong\ prod.cong\ simp\ add: Y-def\ max-def)$

also have $\dots = (\int^{+\omega}. (\prod_{i \in I}. \omega\ i)\ \partial distr\ M\ (Pi_M\ I\ (\lambda i. borel)))\ (\lambda x. \lambda i \in I. Y\ i\ x)$

by $(subst\ nn-integral-distr)\ auto$

also have $\dots = (\int^{+\omega}. (\prod_{i \in I}. \omega\ i)\ \partial Pi_M\ I\ (\lambda i. distr\ M\ borel\ (Y\ i)))$

unfolding $indep-vars-iff-distr-eq-PiM[THEN\ iffD1, OF\ \langle I \neq \{\} \rangle\ rv-Y\ indep-Y]$

..

also have $\dots = (\prod_{i \in I}. (\int^{+\omega}. \omega\ \partial distr\ M\ borel\ (Y\ i)))$

by $(rule\ product-nn-integral-prod)\ (auto\ intro: \langle finite\ I \rangle)$

also have $\dots = (\prod_{i \in I}. \int^{+\omega}. X\ i\ \omega\ \partial M)$

by $(intro\ prod.cong\ nn-integral-cong)\ (auto\ simp: nn-integral-distr\ Y-def\ rv-X)$

finally show $?thesis$.

qed $(simp\ add: emeasure-space-I)$

lemma **(in** *prob-space*)

fixes $X :: 'i \Rightarrow 'a \Rightarrow 'b :: \{real-normed-field, banach, second-countable-topology\}$

assumes I : *finite I indep-vars* $(\lambda-. borel)\ X\ I \wedge i. i \in I \implies integrable\ M\ (X\ i)$

shows *indep-vars-lebesgue-integral*: $(\int \omega. (\prod_{i \in I}. X\ i\ \omega)\ \partial M) = (\prod_{i \in I}. \int \omega. X\ i\ \omega\ \partial M)$ **(is ?eq)**

and *indep-vars-integrable*: $integrable\ M\ (\lambda \omega. (\prod_{i \in I}. X\ i\ \omega))$ **(is ?int)**

proof $(induct\ rule: case-split)$

assume $I \neq \{\}$

define Y **where** $[abs-def]: Y\ i\ \omega = (if\ i \in I\ then\ X\ i\ \omega\ else\ 0)$ **for** $i\ \omega$

{ fix i **have** $i \in I \implies random-variable\ borel\ (X\ i)$

using $I(2)$ **by** $(cases\ i \in I)$ $(auto\ simp: indep-vars-def)$ **}**

note $rv-X = this[measurable]$

{ fix i **have** $random-variable\ borel\ (Y\ i)$

```

    using I(2) by (cases i∈I) (auto simp: Y-def rv-X) }
    note rv-Y = this[measurable]

{ fix i have integrable M (Y i)
  using I(3) by (cases i∈I) (auto simp: Y-def) }
note int-Y = this

interpret Y: prob-space distr M borel (Y i) for i
  using I(2) by (cases i ∈ I) (auto intro!: prob-space-distr simp: indep-vars-def
prob-space-return)
interpret product-sigma-finite λi. distr M borel (Y i)
..

have indep-Y: indep-vars (λi. borel) Y I
  by (rule indep-vars-cong[THEN iffD1, OF - - - I(2)]) (auto simp: Y-def)

have (∫ ω. (∏ i∈I. X i ω) ∂M) = (∫ ω. (∏ i∈I. Y i ω) ∂M)
  using I(3) by (simp add: Y-def)
also have ... = (∫ ω. (∏ i∈I. ω i) ∂distr M (Pi_M I (λi. borel)) (λx. λi∈I. Y
i x))
  by (subst integral-distr) auto
also have ... = (∫ ω. (∏ i∈I. ω i) ∂Pi_M I (λi. distr M borel (Y i)))
  unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF ⟨I ≠ {}⟩ rv-Y indep-Y]
..
also have ... = (∏ i∈I. (∫ ω. ω ∂distr M borel (Y i)))
  by (rule product-integral-prod) (auto intro: ⟨finite I⟩ simp: integrable-distr-eq
int-Y)
also have ... = (∏ i∈I. ∫ ω. X i ω ∂M)
  by (intro prod.cong integral-cong)
  (auto simp: integral-distr Y-def rv-X)
finally show ?eq .

have integrable (distr M (Pi_M I (λi. borel)) (λx. λi∈I. Y i x)) (λω. (∏ i∈I. ω
i))
  unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF ⟨I ≠ {}⟩ rv-Y indep-Y]
  by (intro product-integrable-prod[OF ⟨finite I⟩])
  (simp add: integrable-distr-eq int-Y)
then show ?int
  by (simp add: integrable-distr-eq Y-def)
qed (simp-all add: prob-space)

lemma (in prob-space)
  fixes X1 X2 :: 'a ⇒ 'b::{real-normed-field, banach, second-countable-topology}
  assumes indep-var borel X1 borel X2 integrable M X1 integrable M X2
  shows indep-var-lebesgue-integral: (∫ ω. X1 ω * X2 ω ∂M) = (∫ ω. X1 ω ∂M)
* (∫ ω. X2 ω ∂M) (is ?eq)
  and indep-var-integrable: integrable M (λω. X1 ω * X2 ω) (is ?int)
unfolding indep-var-def
proof -

```

```

have *: (λω. X1 ω * X2 ω) = (λω. ∏ i ∈ UNIV. (case-bool X1 X2 i ω))
  by (simp add: UNIV-bool mult.commute)
have **: (λ -. borel) = case-bool borel borel
  by (rule ext, metis (full-types) bool.simps(3) bool.simps(4))
show ?eq
  apply (subst *)
  apply (subst indep-vars-lebesgue-integral)
  apply (auto)
  apply (subst **, subst indep-var-def [symmetric], rule assms)
  apply (simp split: bool.split add: assms)
  by (simp add: UNIV-bool mult.commute)
show ?int
  apply (subst *)
  apply (rule indep-vars-integrable)
  apply auto
  apply (subst **, subst indep-var-def [symmetric], rule assms)
  by (simp split: bool.split add: assms)
qed

end

```

13 Convolution Measure

theory Convolution

imports Independent-Family

begin

lemma (in finite-measure) sigma-finite-measure: sigma-finite-measure M

..

definition convolution :: ('a :: ordered-euclidean-space) measure ⇒ 'a measure ⇒
'a measure (infix ★ 50) **where**

convolution M N = distr (M ⊗_M N) borel (λ(x, y). x + y)

lemma

shows space-convolution[simp]: space (convolution M N) = space borel

and sets-convolution[simp]: sets (convolution M N) = sets borel

and measurable-convolution1[simp]: measurable A (convolution M N) = measurable A borel

and measurable-convolution2[simp]: measurable (convolution M N) B = measurable borel B

by (simp-all add: convolution-def)

lemma nn-integral-convolution:

assumes finite-measure M finite-measure N

assumes [measurable-cong]: sets N = sets borel sets M = sets borel

assumes [measurable]: f ∈ borel-measurable borel

shows (∫⁺x. f x ∂convolution M N) = (∫⁺x. ∫⁺y. f (x + y) ∂N ∂M)

proof –

interpret M : *finite-measure* M **by fact**
interpret N : *finite-measure* N **by fact**
interpret *pair-sigma-finite* M N ..
show *?thesis*
unfolding *convolution-def*
by (*simp add: nn-integral-distr N.nn-integral-fst[symmetric]*)
qed

lemma *convolution-emeasure*:

assumes $A \in$ *sets borel finite-measure* M *finite-measure* N
assumes [*simp*]: *sets* $N =$ *sets borel sets* $M =$ *sets borel*
assumes [*simp*]: *space* $M =$ *space* N *space* $N =$ *space borel*
shows *emeasure* $(M \star N)$ $A = \int^+ x. (\text{emeasure } N \{a. a + x \in A\}) \partial N$
using *assms* **by** (*auto intro!: nn-integral-cong simp del: nn-integral-indicator*
simp: nn-integral-convolution
nn-integral-indicator [symmetric] ac-simps split:split-indicator)

lemma *convolution-emeasure'*:

assumes [*simp*]: $A \in$ *sets borel*
assumes [*simp*]: *finite-measure* M *finite-measure* N
assumes [*simp*]: *sets* $N =$ *sets borel sets* $M =$ *sets borel*
shows *emeasure* $(M \star N)$ $A = \int^+ x. \int^+ y. (\text{indicator } A (x + y)) \partial N \partial M$
by (*auto simp del: nn-integral-indicator simp: nn-integral-convolution*
nn-integral-indicator[symmetric] borel-measurable-indicator)

lemma *convolution-finite*:

assumes [*simp*]: *finite-measure* M *finite-measure* N
assumes [*measurable-cong*]: *sets* $N =$ *sets borel sets* $M =$ *sets borel*
shows *finite-measure* $(M \star N)$
unfolding *convolution-def*
by (*intro finite-measure-pair-measure finite-measure.finite-measure-distr*) *auto*

lemma *convolution-emeasure-3*:

assumes [*simp, measurable*]: $A \in$ *sets borel*
assumes [*simp*]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [*simp*]: *sets* $N =$ *sets borel sets* $M =$ *sets borel sets* $L =$ *sets borel*
shows *emeasure* $(L \star (M \star N))$ $A = \int^+ x. \int^+ y. \int^+ z. \text{indicator } A (x + y + z) \partial N \partial M \partial L$
apply (*subst nn-integral-indicator[symmetric], simp*)
apply (*subst nn-integral-convolution,*
auto intro!: borel-measurable-indicator borel-measurable-indicator' convolution-finite)
by (*rule nn-integral-cong*)
+ (*auto simp: semigroup-add-class.add.assoc*)

lemma *convolution-emeasure-3'*:

assumes [*simp, measurable*]: $A \in$ *sets borel*
assumes [*simp*]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [*measurable-cong, simp*]: *sets* $N =$ *sets borel sets* $M =$ *sets borel sets* $L =$ *sets borel*
shows *emeasure* $((L \star M) \star N)$ $A = \int^+ x. \int^+ y. \int^+ z. \text{indicator } A (x + y + z)$

z) $\partial N \partial M \partial L$
apply (*subst nn-integral-indicator[symmetric], simp*)
apply (*subst nn-integral-convolution*)
apply (*simp-all add: convolution-finite*)
apply (*subst nn-integral-convolution*)
apply (*simp-all add: finite-measure.sigma-finite-measure sigma-finite-measure.borel-measurable-nn-integral*)
done

lemma *convolution-commutative:*

assumes [*simp*]: *finite-measure M finite-measure N*
assumes [*measurable-cong, simp*]: *sets N = sets borel sets M = sets borel*
shows $(M \star N) = (N \star M)$
proof (*rule measure-eqI*)
interpret *M: finite-measure M by fact*
interpret *N: finite-measure N by fact*
interpret *pair-sigma-finite M N ..*

show *sets (M \star N) = sets (N \star M) by simp*

fix *A assume A \in sets (M \star N)*
then have *1[measurable]: A \in sets borel by simp*
have *emeasure (M \star N) A = $\int^{+x}. \int^{+y}. \text{indicator } A (x + y) \partial N \partial M$ by (auto intro!: convolution-emeasure')*
also have *... = $\int^{+x}. \int^{+y}. (\lambda(x,y). \text{indicator } A (x + y)) (x, y) \partial N \partial M$ by (auto intro!: nn-integral-cong)*
also have *... = $\int^{+y}. \int^{+x}. (\lambda(x,y). \text{indicator } A (x + y)) (x, y) \partial M \partial N$ by (rule Fubini[symmetric]) simp*
also have *... = emeasure (N \star M) A by (auto intro!: nn-integral-cong simp: add commute convolution-emeasure')*
finally show *emeasure (M \star N) A = emeasure (N \star M) A by simp*
qed

lemma *convolution-associative:*

assumes [*simp*]: *finite-measure M finite-measure N finite-measure L*
assumes [*simp*]: *sets N = sets borel sets M = sets borel sets L = sets borel*
shows $(L \star (M \star N)) = ((L \star M) \star N)$
by (*auto intro!: measure-eqI simp: convolution-emeasure-3 convolution-emeasure-3'*)

lemma (*in prob-space*) *sum-indep-random-variable:*

assumes *ind: indep-var borel X borel Y*
assumes [*simp, measurable*]: *random-variable borel X*
assumes [*simp, measurable*]: *random-variable borel Y*
shows *distr M borel ($\lambda x. X x + Y x$) = convolution (distr M borel X) (distr M borel Y)*
using *ind unfolding indep-var-distribution-eq convolution-def*
by (*auto simp: distr-distr intro!: arg-cong[where f = distr M borel]*)

lemma (*in prob-space*) *sum-indep-random-variable-lborel:*

assumes *ind: indep-var borel X borel Y*

assumes [*simp*, *measurable*]: *random-variable lborel X*
assumes [*simp*, *measurable*]: *random-variable lborel Y*
shows *distr M lborel* ($\lambda x. X x + Y x$) = *convolution* (*distr M lborel X*) (*distr M lborel Y*)
using *ind unfolding indep-var-distribution-eq convolution-def*
by (*auto simp: distr-distr o-def intro!: arg-cong[where f = distr M borel] cong: distr-cong*)

lemma *convolution-density:*

fixes $f g :: \text{real} \Rightarrow \text{ennreal}$
assumes [*measurable*]: $f \in \text{borel-measurable borel}$ $g \in \text{borel-measurable borel}$
assumes [*simp*]: *finite-measure (density lborel f)* *finite-measure (density lborel g)*
shows *density lborel f* \star *density lborel g* = *density lborel* ($\lambda x. \int^+ y. f (x - y) * g y \partial \text{lborel}$)
 (is ?l = ?r)
proof (*intro measure-eqI*)
fix A **assume** $A \in \text{sets } ?l$
then have [*measurable*]: $A \in \text{sets borel}$
by *simp*

have ($\int^+ x. f x * (\int^+ y. g y * \text{indicator } A (x + y) \partial \text{lborel}) \partial \text{lborel}$) =
 ($\int^+ x. (\int^+ y. g y * (f x * \text{indicator } A (x + y))) \partial \text{lborel}$) ∂lborel

proof (*intro nn-integral-cong-AE, eventually-elim*)

fix x

have $f x * (\int^+ y. g y * \text{indicator } A (x + y) \partial \text{lborel}) =$
 ($\int^+ y. f x * (g y * \text{indicator } A (x + y)) \partial \text{lborel}$)

by (*intro nn-integral-cmult[symmetric] auto*)

then show $f x * (\int^+ y. g y * \text{indicator } A (x + y) \partial \text{lborel}) =$
 ($\int^+ y. g y * (f x * \text{indicator } A (x + y)) \partial \text{lborel}$)

by (*simp add: ac-simps*)

qed

also have $\dots = (\int^+ y. (\int^+ x. g y * (f x * \text{indicator } A (x + y)) \partial \text{lborel}) \partial \text{lborel})$
by (*intro lborel-pair.Fubini' simp*)

also have $\dots = (\int^+ y. (\int^+ x. f (x - y) * g y * \text{indicator } A x \partial \text{lborel}) \partial \text{lborel})$

proof (*intro nn-integral-cong-AE, eventually-elim*)

fix y

have ($\int^+ x. g y * (f x * \text{indicator } A (x + y)) \partial \text{lborel}$) =
 $g y * (\int^+ x. f x * \text{indicator } A (x + y) \partial \text{lborel})$

by (*intro nn-integral-cmult auto*)

also have $\dots = g y * (\int^+ x. f (x - y) * \text{indicator } A x \partial \text{lborel})$

by (*subst nn-integral-real-affine[where c=1 and t=-y]*)

(*auto simp add: one-ennreal-def[symmetric]*)

also have $\dots = (\int^+ x. g y * (f (x - y) * \text{indicator } A x) \partial \text{lborel})$

by (*intro nn-integral-cmult[symmetric] auto*)

finally show ($\int^+ x. g y * (f x * \text{indicator } A (x + y)) \partial \text{lborel}$) =
 ($\int^+ x. f (x - y) * g y * \text{indicator } A x \partial \text{lborel}$)

by (*simp add: ac-simps*)

qed

also have $\dots = (\int^+ x. (\int^+ y. f (x - y) * g y * \text{indicator } A x \partial \text{lborel}) \partial \text{lborel})$

by (intro lborel-pair.Fubini') simp
 finally show emeasure ?l A = emeasure ?r A
 by (auto simp: convolution-emeasure' nn-integral-density emeasure-density
 nn-integral-multc)
 qed simp

lemma (in prob-space) distributed-finite-measure-density:
 distributed M N X f \implies finite-measure (density N f)
 using finite-measure-distr[of X N] distributed-distr-eq-density[of M N X f] by
 simp

lemma (in prob-space) distributed-convolution:
 fixes f :: real \Rightarrow -
 fixes g :: real \Rightarrow -
 assumes indep: indep-var borel X borel Y
 assumes X: distributed M lborel X f
 assumes Y: distributed M lborel Y g
 shows distributed M lborel ($\lambda x. X x + Y x$) ($\lambda x. \int^+ y. f (x - y) * g y \partial lborel$)
 unfolding distributed-def
 proof safe
 have fg[measurable]: f \in borel-measurable borel g \in borel-measurable borel
 using distributed-borel-measurable[OF X] distributed-borel-measurable[OF Y]
 by simp-all

show ($\lambda x. \int^+ xa. f (x - xa) * g xa \partial lborel$) \in borel-measurable lborel
 by measurable

have distr M borel ($\lambda x. X x + Y x$) = (distr M borel X \star distr M borel Y)
 using distributed-measurable[OF X] distributed-measurable[OF Y]
 by (intro sum-indep-random-variable) (auto simp: indep)
 also have ... = (density lborel f \star density lborel g)
 using distributed-distr-eq-density[OF X] distributed-distr-eq-density[OF Y]
 by (simp cong: distr-cong)
 also have ... = density lborel ($\lambda x. \int^+ y. f (x - y) * g y \partial lborel$)
 proof (rule convolution-density)
 show finite-measure (density lborel f)
 using X by (rule distributed-finite-measure-density)
 show finite-measure (density lborel g)
 using Y by (rule distributed-finite-measure-density)
 qed fact+
 finally show distr M lborel ($\lambda x. X x + Y x$) = density lborel ($\lambda x. \int^+ y. f (x - y) * g y \partial lborel$)
 by (simp cong: distr-cong)
 show random-variable lborel ($\lambda x. X x + Y x$)
 using distributed-measurable[OF X] distributed-measurable[OF Y] by simp
 qed

lemma prob-space-convolution-density:

```

fixes f:: real  $\Rightarrow$  -
fixes g:: real  $\Rightarrow$  -
assumes [measurable]: f  $\in$  borel-measurable borel
assumes [measurable]: g  $\in$  borel-measurable borel
assumes gt-0[simp]:  $\bigwedge x. 0 \leq f\ x \ \bigwedge x. 0 \leq g\ x$ 
assumes prob-space (density lborel f) (is prob-space ?F)
assumes prob-space (density lborel g) (is prob-space ?G)
shows prob-space (density lborel ( $\lambda x. \int^+ y. f\ (x - y) * g\ y\ \partial$ lborel)) (is prob-space
?D)
proof (subst convolution-density[symmetric])
  interpret F: prob-space ?F by fact
  show finite-measure ?F by unfold-locales
  interpret G: prob-space ?G by fact
  show finite-measure ?G by unfold-locales
  interpret FG: pair-prob-space ?F ?G ..

  show prob-space (density lborel f  $\star$  density lborel g)
    unfolding convolution-def by (rule FG.prob-space-distr) simp
qed simp-all

end

```

14 Information theory

theory Information

imports

Independent-Family

begin

lemma log-le: $1 < a \implies 0 < x \implies x \leq y \implies \log a\ x \leq \log a\ y$
by (subst log-le-cancel-iff) auto

lemma log-less: $1 < a \implies 0 < x \implies x < y \implies \log a\ x < \log a\ y$
by (subst log-less-cancel-iff) auto

lemma sum-cartesian-product':

$(\sum x \in A \times B. f\ x) = (\sum x \in A. \text{sum } (\lambda y. f\ (x, y))\ B)$

unfolding sum.cartesian-product **by** simp

lemma split-pairs:

$((A, B) = X) \longleftrightarrow (\text{fst } X = A \wedge \text{snd } X = B)$ **and**

$(X = (A, B)) \longleftrightarrow (\text{fst } X = A \wedge \text{snd } X = B)$ **by** auto

14.1 Information theory

locale information-space = prob-space +

fixes b :: real **assumes** b-gt-1: $1 < b$

context information-space

begin

Introduce some simplification rules for logarithm of base b .

lemma *log-neg-const*:

assumes $x \leq 0$

shows $\log b x = \log b 0$

proof –

{ **fix** $u :: \text{real}$

have $x \leq 0$ **by** *fact*

also have $0 < \exp u$

using *exp-gt-zero* .

finally have $\exp u \neq x$

by *auto* }

then show $\log b x = \log b 0$

by (*simp add: log-def ln-real-def*)

qed

lemma *log-mult-eq*:

$\log b (A * B) = (\text{if } 0 < A * B \text{ then } \log b |A| + \log b |B| \text{ else } \log b 0)$

using *log-mult[of b |A| |B|] b-gt-1 log-neg-const[of A * B]*

by (*auto simp: zero-less-mult-iff mult-le-0-iff*)

lemma *log-inverse-eq*:

$\log b (\text{inverse } B) = (\text{if } 0 < B \text{ then } - \log b B \text{ else } \log b 0)$

using *log-inverse[of b B] log-neg-const[of inverse B] b-gt-1* **by** *simp*

lemma *log-divide-eq*:

$\log b (A / B) = (\text{if } 0 < A * B \text{ then } \log b |A| - \log b |B| \text{ else } \log b 0)$

unfolding *divide-inverse log-mult-eq log-inverse-eq abs-inverse*

by (*auto simp: zero-less-mult-iff mult-le-0-iff*)

lemmas *log-simps = log-mult-eq log-inverse-eq log-divide-eq*

end

14.2 Kullback–Leibler divergence

The Kullback–Leibler divergence is also known as relative entropy or Kullback–Leibler distance.

definition

$\text{entropy-density } b M N = \log b \circ \text{enn2real} \circ \text{RN-deriv } M N$

definition

$\text{KL-divergence } b M N = \text{integral}^L N (\text{entropy-density } b M N)$

lemma *measurable-entropy-density[measurable]*: $\text{entropy-density } b M N \in \text{borel-measurable } M$

unfolding *entropy-density-def* **by** *auto*

lemma (in *sigma-finite-measure*) *KL-density*:
fixes $f :: 'a \Rightarrow \text{real}$
assumes $1 < b$
assumes $f[\text{measurable}]$: $f \in \text{borel-measurable } M$ **and** nn : $AE\ x\ \text{in } M. 0 \leq f\ x$
shows *KL-divergence* $b\ M\ (\text{density } M\ f) = (\int x. f\ x * \log b\ (f\ x)\ \partial M)$
unfolding *KL-divergence-def*
proof (*subst integral-real-density*)
show $[\text{measurable}]$: *entropy-density* $b\ M\ (\text{density } M\ (\lambda x. \text{ennreal } (f\ x))) \in$
borel-measurable } M
using f
by (*auto simp: comp-def entropy-density-def*)
have *density* $M\ (\text{RN-deriv } M\ (\text{density } M\ f)) = \text{density } M\ f$
using $f\ nn$ **by** (*intro density-RN-deriv-density*) *auto*
then have *eq*: $AE\ x\ \text{in } M. \text{RN-deriv } M\ (\text{density } M\ f)\ x = f\ x$
using $f\ nn$ **by** (*intro density-unique*) *auto*
show $(\int x. f\ x * \text{entropy-density } b\ M\ (\text{density } M\ (\lambda x. \text{ennreal } (f\ x)))\ x\ \partial M) =$
 $(\int x. f\ x * \log b\ (f\ x)\ \partial M)$
apply (*intro integral-cong-AE*)
apply *measurable*
using *eq nn*
apply *eventually-elim*
apply (*auto simp: entropy-density-def*)
done
qed *fact+*

lemma (in *sigma-finite-measure*) *KL-density-density*:
fixes $f\ g :: 'a \Rightarrow \text{real}$
assumes $1 < b$
assumes f : $f \in \text{borel-measurable } M\ AE\ x\ \text{in } M. 0 \leq f\ x$
assumes g : $g \in \text{borel-measurable } M\ AE\ x\ \text{in } M. 0 \leq g\ x$
assumes ac : $AE\ x\ \text{in } M. f\ x = 0 \longrightarrow g\ x = 0$
shows *KL-divergence* $b\ (\text{density } M\ f)\ (\text{density } M\ g) = (\int x. g\ x * \log b\ (g\ x / f\ x)\ \partial M)$
proof –
interpret Mf : *sigma-finite-measure density } M\ f*
using f **by** (*subst sigma-finite-iff-density-finite*) *auto*
have *KL-divergence* $b\ (\text{density } M\ f)\ (\text{density } M\ g) =$
KL-divergence $b\ (\text{density } M\ f)\ (\text{density } (\text{density } M\ f)\ (\lambda x. g\ x / f\ x))$
using $f\ g\ ac$ **by** (*subst density-density-divide*) *simp-all*
also have $\dots = (\int x. (g\ x / f\ x) * \log b\ (g\ x / f\ x)\ \partial \text{density } M\ f)$
using $f\ g\ \langle 1 < b \rangle$ **by** (*intro Mf.KL-density*) (*auto simp: AE-density*)
also have $\dots = (\int x. g\ x * \log b\ (g\ x / f\ x)\ \partial M)$
using $ac\ f\ g\ \langle 1 < b \rangle$ **by** (*subst integral-density*) (*auto intro!: integral-cong-AE*)
finally show *?thesis* .
qed

lemma (in *information-space*) *KL-gt-0*:
fixes $D :: 'a \Rightarrow \text{real}$

```

assumes prob-space (density M D)
assumes D: D ∈ borel-measurable M AE x in M. 0 ≤ D x
assumes int: integrable M (λx. D x * log b (D x))
assumes A: density M D ≠ M
shows 0 < KL-divergence b M (density M D)
proof –
interpret N: prob-space density M D by fact

obtain A where A ∈ sets M emeasure (density M D) A ≠ emeasure M A
using measure-eqI[of density M D M] ⟨density M D ≠ M⟩ by auto

let ?D-set = {x ∈ space M. D x ≠ 0}
have [simp, intro]: ?D-set ∈ sets M
using D by auto

have D-neg: (∫+ x. ennreal (– D x) ∂M) = 0
using D by (subst nn-integral-0-iff-AE) (auto simp: ennreal-neg)

have (∫+ x. ennreal (D x) ∂M) = emeasure (density M D) (space M)
using D by (simp add: emeasure-density cong: nn-integral-cong)
then have D-pos: (∫+ x. ennreal (D x) ∂M) = 1
using N.emeasure-space-1 by simp

have integrable M D
using D D-pos D-neg unfolding real-integrable-def real-lebesgue-integral-def
by simp-all
then have integralL M D = 1
using D D-pos D-neg by (simp add: real-lebesgue-integral-def)

have 0 ≤ 1 – measure M ?D-set
using prob-le-1 by (auto simp: field-simps)
also have ... = (∫ x. D x – indicator ?D-set x ∂M)
using ⟨integrable M D⟩ ⟨integralL M D = 1⟩
by (simp add: emeasure-eq-measure)
also have ... < (∫ x. D x * (ln b * log b (D x)) ∂M)
proof (rule integral-less-AE)
show integrable M (λx. D x – indicator ?D-set x)
using ⟨integrable M D⟩ by (auto simp: less-top[symmetric])
next
from integrable-mult-left(1)[OF int, of ln b]
show integrable M (λx. D x * (ln b * log b (D x)))
by (simp add: ac-simps)
next
show emeasure M {x ∈ space M. D x ≠ 1 ∧ D x ≠ 0} ≠ 0
proof
assume eq-0: emeasure M {x ∈ space M. D x ≠ 1 ∧ D x ≠ 0} = 0
then have disj: AE x in M. D x = 1 ∨ D x = 0
using D(1) by (auto intro!: AE-I[OF subset-refl] sets.sets-Collect)

```

```

have emeasure  $M \{x \in \text{space } M. D x = 1\} = (\int^+ x. \text{indicator } \{x \in \text{space } M. D x = 1\} x \partial M)$ 
using  $D(1)$  by auto
also have  $\dots = (\int^+ x. \text{ennreal } (D x) \partial M)$ 
using disj by (auto intro!: nn-integral-cong-AE simp: indicator-def one-ennreal-def)
finally have AE  $x$  in  $M. D x = 1$ 
using  $D$  D-pos by (intro AE-I-eq-1) auto
then have  $(\int^+ x. \text{indicator } A x \partial M) = (\int^+ x. \text{ennreal } (D x) * \text{indicator } A x \partial M)$ 
by (intro nn-integral-cong-AE) (auto simp: one-ennreal-def[symmetric])
also have  $\dots = \text{density } M D A$ 
using  $\langle A \in \text{sets } M \rangle D$  by (simp add: emeasure-density)
finally show False using  $\langle A \in \text{sets } M \rangle \langle \text{emeasure } (\text{density } M D) A \neq \text{emeasure } M A \rangle$  by simp
qed
show  $\{x \in \text{space } M. D x \neq 1 \wedge D x \neq 0\} \in \text{sets } M$ 
using  $D(1)$  by (auto intro: sets.sets-Collect-conj)

show AE  $t$  in  $M. t \in \{x \in \text{space } M. D x \neq 1 \wedge D x \neq 0\} \longrightarrow D t - \text{indicator } ?D\text{-set } t \neq D t * (\ln b * \log b (D t))$ 
using  $D(2)$ 
proof (eventually-elim, safe)
fix  $t$  assume  $Dt$ :  $t \in \text{space } M D t \neq 1 D t \neq 0 0 \leq D t$ 
and eq:  $D t - \text{indicator } ?D\text{-set } t = D t * (\ln b * \log b (D t))$ 

have  $D t - 1 = D t - \text{indicator } ?D\text{-set } t$ 
using  $Dt$  by simp
also note eq
also have  $D t * (\ln b * \log b (D t)) = - D t * \ln (1 / D t)$ 
using b-gt-1  $\langle D t \neq 0 \rangle \langle 0 \leq D t \rangle$ 
by (simp add: log-def ln-div less-le)
finally have  $\ln (1 / D t) = 1 / D t - 1$ 
using  $\langle D t \neq 0 \rangle$  by (auto simp: field-simps)
from ln-eq-minus-one[OF - this]  $\langle D t \neq 0 \rangle \langle 0 \leq D t \rangle \langle D t \neq 1 \rangle$ 
show False by auto
qed

show AE  $t$  in  $M. D t - \text{indicator } ?D\text{-set } t \leq D t * (\ln b * \log b (D t))$ 
using  $D(2)$  AE-space
proof eventually-elim
fix  $t$  assume  $t \in \text{space } M 0 \leq D t$ 
show  $D t - \text{indicator } ?D\text{-set } t \leq D t * (\ln b * \log b (D t))$ 
proof cases
assume asm:  $D t \neq 0$ 
then have  $0 < D t$  using  $\langle 0 \leq D t \rangle$  by auto
then have  $0 < 1 / D t$  by auto
have  $D t - \text{indicator } ?D\text{-set } t \leq - D t * (1 / D t - 1)$ 
using asm  $\langle t \in \text{space } M \rangle$  by (simp add: field-simps)
also have  $- D t * (1 / D t - 1) \leq - D t * \ln (1 / D t)$ 

```



```

    using ln-le-minus-one (0 < 1 / D t) by (intro mult-left-mono-neg) auto
  also have ... = D t * (ln b * log b (D t))
    using (0 < D t) b-gt-1
    by (simp-all add: log-def ln-div)
  finally show ?thesis by simp
qed simp
qed
qed
also have ... = (∫ x. ln b * (D x * log b (D x)) ∂M)
  by (simp add: ac-simps)
also have ... = ln b * (∫ x. D x * log b (D x) ∂M)
  using int by simp
finally show ?thesis
  using b-gt-1 D by (subst KL-density) (auto simp: zero-less-mult-iff)
qed

```

lemma (in *sigma-finite-measure*) *KL-same-eq-0: KL-divergence b M M = 0*

proof –

have *AE x in M. 1 = RN-deriv M M x*

proof (*rule RN-deriv-unique*)

show *density M (λx. 1) = M*

apply (*auto intro!: measure-eqI emeasure-density*)

apply (*subst emeasure-density*)

apply *auto*

done

qed *auto*

then have *AE x in M. log b (enn2real (RN-deriv M M x)) = 0*

by (*elim AE-mp*) *simp*

from *integral-cong-AE[OF - - this]*

have *integral^L M (entropy-density b M M) = 0*

by (*simp add: entropy-density-def comp-def*)

then show *KL-divergence b M M = 0*

unfolding *KL-divergence-def*

by *auto*

qed

lemma (in *information-space*) *KL-eq-0-iff-eq:*

fixes *D :: 'a ⇒ real*

assumes *prob-space (density M D)*

assumes *D: D ∈ borel-measurable M AE x in M. 0 ≤ D x*

assumes *int: integrable M (λx. D x * log b (D x))*

shows *KL-divergence b M (density M D) = 0 ⟷ density M D = M*

using *KL-same-eq-0[of b] KL-gt-0[OF assms]*

by (*auto simp: less-le*)

lemma (in *information-space*) *KL-eq-0-iff-eq-ac:*

fixes *D :: 'a ⇒ real*

assumes *prob-space N*

assumes *ac: absolutely-continuous M N sets N = sets M*

assumes *int*: integrable N (entropy-density b M N)
shows *KL-divergence* b M $N = 0 \iff N = M$
proof –
interpret N : prob-space N **by** *fact*
have *finite-measure* N **by** *unfold-locales*
from *real-RN-deriv*[*OF this ac*] **guess** D . **note** $D = \text{this}$

have $N = \text{density } M$ (*RN-deriv* M N)
using *ac* **by** (*rule density-RN-deriv*[*symmetric*])
also have $\dots = \text{density } M$ D
using D **by** (*auto intro!*: *density-cong*)
finally have N : $N = \text{density } M$ D .

from *absolutely-continuous-AE*[*OF ac*(2,1) D (2)] D *b-gt-1 ac measurable-entropy-density*
have integrable N ($\lambda x. \log b$ (D x))
by (*intro integrable-cong-AE*[*THEN iffD2*, *OF - - - int*])
(*auto simp*: *N entropy-density-def*)
with D *b-gt-1* **have** integrable M ($\lambda x. D$ $x * \log b$ (D x))
by (*subst integrable-real-density*[*symmetric*]) (*auto simp*: *N*[*symmetric*] *comp-def*)
with (*prob-space* N) D **show** *?thesis*
unfolding N
by (*intro KL-eq-0-iff-eq*) *auto*
qed

lemma (*in information-space*) *KL-nonneg*:
assumes *prob-space* (*density* M D)
assumes D : $D \in \text{borel-measurable } M$ *AE* x *in* M . $0 \leq D$ x
assumes *int*: integrable M ($\lambda x. D$ $x * \log b$ (D x))
shows $0 \leq \text{KL-divergence } b$ (*density* M D)
using *KL-gt-0*[*OF assms*] **by** (*cases density* M $D = M$) (*auto simp*: *KL-same-eq-0*)

lemma (*in sigma-finite-measure*) *KL-density-density-nonneg*:
fixes f g :: ' $a \Rightarrow \text{real}$ '
assumes $1 < b$
assumes f : $f \in \text{borel-measurable } M$ *AE* x *in* M . $0 \leq f$ x *prob-space* (*density* M f)
assumes g : $g \in \text{borel-measurable } M$ *AE* x *in* M . $0 \leq g$ x *prob-space* (*density* M g)
assumes *ac*: *AE* x *in* M . f $x = 0 \implies g$ $x = 0$
assumes *int*: integrable M ($\lambda x. g$ $x * \log b$ (g x / f x))
shows $0 \leq \text{KL-divergence } b$ (*density* M f) (*density* M g)
proof –
interpret Mf : *prob-space density* M f **by** *fact*
interpret Mf : *information-space density* M f b **by** *standard fact*
have *eq*: *density* (*density* M f) ($\lambda x. g$ x / f x) = *density* M g (*is* $?DD = -$)
using f g *ac* **by** (*subst density-density-divide*) *simp-all*

have $0 \leq \text{KL-divergence } b$ (*density* M f) (*density* (*density* M f) ($\lambda x. g$ x / f x))
proof (*rule* Mf .*KL-nonneg*)

```

show prob-space ?DD unfolding eq by fact
from f g show  $(\lambda x. g x / f x) \in \text{borel-measurable (density M f)}$ 
  by auto
show AE x in density M f.  $0 \leq g x / f x$ 
  using f g by (auto simp: AE-density)
show integrable (density M f)  $(\lambda x. g x / f x * \log b (g x / f x))$ 
  using  $\langle 1 < b \rangle$  f g ac
  by (subst integrable-density)
  (auto intro!: integrable-cong-AE[THEN iffD2, OF - - - int] measurable-If)
qed
also have  $\dots = \text{KL-divergence } b \text{ (density M f) (density M g)}$ 
  using f g ac by (subst density-density-divide) simp-all
finally show ?thesis .
qed

```

14.3 Finite Entropy

definition (in *information-space*) *finite-entropy* :: $'b \text{ measure} \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow \text{real}) \Rightarrow \text{bool}$

where

```

finite-entropy S X f  $\longleftrightarrow$ 
  distributed M S X f  $\wedge$ 
  integrable S  $(\lambda x. f x * \log b (f x))$   $\wedge$ 
   $(\forall x \in \text{space } S. 0 \leq f x)$ 

```

lemma (in *information-space*) *finite-entropy-simple-function*:

assumes *X: simple-function M X*

shows *finite-entropy (count-space (X' space M)) X $(\lambda a. \text{measure } M \{x \in \text{space } M. X x = a\})$*

unfolding *finite-entropy-def*

proof *safe*

have $[\text{simp}]: \text{finite } (X \text{ ' space } M)$

using *X by (auto simp: simple-function-def)*

then show *integrable (count-space (X' space M))*

$(\lambda x. \text{prob } \{xa \in \text{space } M. X xa = x\} * \log b (\text{prob } \{xa \in \text{space } M. X xa = x\}))$

by *(rule integrable-count-space)*

have *d: distributed M (count-space (X' space M)) X $(\lambda x. \text{ennreal (if } x \in X' \text{space } M \text{ then prob } \{xa \in \text{space } M. X xa = x\} \text{ else } 0))$*

by *(rule distributed-simple-function-superset[OF X]) (auto intro!: arg-cong[where f=prob])*

show *distributed M (count-space (X' space M)) X $(\lambda x. \text{ennreal (prob } \{xa \in \text{space } M. X xa = x\})$*

by *(rule distributed-cong-density[THEN iffD1, OF - - - d]) auto*

qed *(rule measure-nonneg)*

lemma *ac-fst*:

assumes *sigma-finite-measure T*

shows *absolutely-continuous S (distr (S \otimes_M T) S fst)*

proof –

interpret *sigma-finite-measure T* **by fact**
 { **fix** *A* **assume** *A*: $A \in \text{sets } S \text{ emeasure } S A = 0$
then have *fst* – ‘ $A \cap \text{space } (S \otimes_M T) = A \times \text{space } T$
by (*auto simp: space-pair-measure dest!: sets.sets-into-space*)
with *A* **have** *emeasure* $(S \otimes_M T) (\text{fst} - ' A \cap \text{space } (S \otimes_M T)) = 0$
by (*simp add: emeasure-pair-measure-Times*) }
then show *?thesis*
unfolding *absolutely-continuous-def*
apply (*auto simp: null-sets-distr-iff*)
apply (*auto simp: null-sets-def intro!: measurable-sets*)
done

qed

lemma *ac-snd*:

assumes *sigma-finite-measure T*
shows *absolutely-continuous T (distr (S ⊗_M T) T snd)*

proof –

interpret *sigma-finite-measure T* **by fact**
 { **fix** *A* **assume** *A*: $A \in \text{sets } T \text{ emeasure } T A = 0$
then have *snd* – ‘ $A \cap \text{space } (S \otimes_M T) = \text{space } S \times A$
by (*auto simp: space-pair-measure dest!: sets.sets-into-space*)
with *A* **have** *emeasure* $(S \otimes_M T) (\text{snd} - ' A \cap \text{space } (S \otimes_M T)) = 0$
by (*simp add: emeasure-pair-measure-Times*) }
then show *?thesis*
unfolding *absolutely-continuous-def*
apply (*auto simp: null-sets-distr-iff*)
apply (*auto simp: null-sets-def intro!: measurable-sets*)
done

qed

lemma (*in information-space*) *finite-entropy-integrable*:

finite-entropy S X Px \implies *integrable S (λx. Px x * log b (Px x))*
unfolding *finite-entropy-def* **by** *auto*

lemma (*in information-space*) *finite-entropy-distributed*:

finite-entropy S X Px \implies *distributed M S X Px*
unfolding *finite-entropy-def* **by** *auto*

lemma (*in information-space*) *finite-entropy-nn*:

finite-entropy S X Px \implies $x \in \text{space } S \implies 0 \leq Px x$
by (*auto simp: finite-entropy-def*)

lemma (*in information-space*) *finite-entropy-measurable*:

finite-entropy S X Px \implies $Px \in S \rightarrow_M \text{borel}$
using *distributed-real-measurable[of S Px M X]*
finite-entropy-nn[of S X Px] finite-entropy-distributed[of S X Px] **by** *auto*

lemma (*in information-space*) *subdensity-finite-entropy*:

fixes $g :: 'b \Rightarrow \text{real}$ **and** $f :: 'c \Rightarrow \text{real}$
assumes $T: T \in \text{measurable } P \ Q$
assumes $f: \text{finite-entropy } P \ X \ f$
assumes $g: \text{finite-entropy } Q \ Y \ g$
assumes $Y: Y = T \circ X$
shows $AE \ x \ \text{in } P. \ g \ (T \ x) = 0 \ \longrightarrow \ f \ x = 0$
using $\text{subdensity}[OF \ T, \ \text{of } M \ X \ \lambda x. \ \text{ennreal} \ (f \ x) \ Y \ \lambda x. \ \text{ennreal} \ (g \ x)]$
 $\text{finite-entropy-distributed}[OF \ f] \ \text{finite-entropy-distributed}[OF \ g]$
 $\text{finite-entropy-nn}[OF \ f] \ \text{finite-entropy-nn}[OF \ g]$
 assms
by *auto*

lemma (in *information-space*) *finite-entropy-integrable-transform*:
 $\text{finite-entropy } S \ X \ Px \ \Longrightarrow \ \text{distributed } M \ T \ Y \ Py \ \Longrightarrow \ (\bigwedge x. \ x \in \text{space } T \ \Longrightarrow \ 0 \leq$
 $Py \ x) \ \Longrightarrow$
 $X = (\lambda x. \ f \ (Y \ x)) \ \Longrightarrow \ f \in \text{measurable } T \ S \ \Longrightarrow \ \text{integrable } T \ (\lambda x. \ Py \ x * \log \ b$
 $(Px \ (f \ x)))$
using $\text{distributed-transform-integrable}[of \ M \ T \ Y \ Py \ S \ X \ Px \ f \ \lambda x. \ \log \ b \ (Px \ x)]$
using $\text{distributed-real-measurable}[of \ S \ Px \ M \ X]$
by (*auto simp: finite-entropy-def*)

14.4 Mutual Information

definition (in *prob-space*)
 $\text{mutual-information } b \ S \ T \ X \ Y =$
 $KL\text{-divergence } b \ (\text{distr } M \ S \ X \ \otimes_M \ \text{distr } M \ T \ Y) \ (\text{distr } M \ (S \ \otimes_M \ T) \ (\lambda x. \ (X \ x, \ Y \ x)))$

lemma (in *information-space*) *mutual-information-indep-vars*:
fixes $S \ T \ X \ Y$
defines $P \equiv \text{distr } M \ S \ X \ \otimes_M \ \text{distr } M \ T \ Y$
defines $Q \equiv \text{distr } M \ (S \ \otimes_M \ T) \ (\lambda x. \ (X \ x, \ Y \ x))$
shows $\text{indep-var } S \ X \ T \ Y \ \longleftrightarrow$
 $(\text{random-variable } S \ X \ \wedge \ \text{random-variable } T \ Y \ \wedge$
 $\text{absolutely-continuous } P \ Q \ \wedge \ \text{integrable } Q \ (\text{entropy-density } b \ P \ Q) \ \wedge$
 $\text{mutual-information } b \ S \ T \ X \ Y = 0)$

unfolding *indep-var-distribution-eq*

proof *safe*

assume $rv[\text{measurable}]$: $\text{random-variable } S \ X \ \text{random-variable } T \ Y$

interpret X : *prob-space distr M S X*

by (*rule prob-space-distr*) *fact*

interpret Y : *prob-space distr M T Y*

by (*rule prob-space-distr*) *fact*

interpret XY : *pair-prob-space distr M S X distr M T Y* **by** *standard*

interpret P : *information-space P b* **unfolding** $P\text{-def}$ **by** *standard (rule b-gt-1)*

interpret Q : *prob-space Q* **unfolding** $Q\text{-def}$

by (*rule prob-space-distr*) *simp*

```

{ assume distr M S X  $\otimes_M$  distr M T Y = distr M (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ )
  then have [simp]: Q = P unfolding Q-def P-def by simp

  show ac: absolutely-continuous P Q by (simp add: absolutely-continuous-def)
  then have ed: entropy-density b P Q  $\in$  borel-measurable P
    by simp

  have AE x in P. 1 = RN-deriv P Q x
  proof (rule P.RN-deriv-unique)
    show density P ( $\lambda x. 1$ ) = Q
      unfolding  $\langle Q = P \rangle$  by (intro measure-eqI) (auto simp: emeasure-density)
    qed auto
  then have ae-0: AE x in P. entropy-density b P Q x = 0
    by eventually-elim (auto simp: entropy-density-def)
  then have integrable P (entropy-density b P Q)  $\longleftrightarrow$  integrable Q ( $\lambda x. 0::real$ )
    using ed unfolding  $\langle Q = P \rangle$  by (intro integrable-cong-AE) auto
  then show integrable Q (entropy-density b P Q) by simp

  from ae-0 have mutual-information b S T X Y = ( $\int x. 0 \partial P$ )
  unfolding mutual-information-def KL-divergence-def P-def[symmetric] Q-def[symmetric]
 $\langle Q = P \rangle$ 
    by (intro integral-cong-AE) auto
  then show mutual-information b S T X Y = 0
    by simp }

{ assume ac: absolutely-continuous P Q
  assume int: integrable Q (entropy-density b P Q)
  assume I-eq-0: mutual-information b S T X Y = 0

  have eq: Q = P
  proof (rule P.KL-eq-0-iff-eq-ac[THEN iffD1])
    show prob-space Q by unfold-locales
    show absolutely-continuous P Q by fact
    show integrable Q (entropy-density b P Q) by fact
    show sets Q = sets P by (simp add: P-def Q-def sets-pair-measure)
    show KL-divergence b P Q = 0
      using I-eq-0 unfolding mutual-information-def by (simp add: P-def Q-def)
    qed
  then show distr M S X  $\otimes_M$  distr M T Y = distr M (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ )
    unfolding P-def Q-def .. }
qed

abbreviation (in information-space)
  mutual-information-Pow ( $\mathcal{I}'(-; -')$ ) where
 $\mathcal{I}(X; Y) \equiv$  mutual-information b (count-space (X'space M)) (count-space (Y'space M)) X Y

```

lemma (in *information-space*)

fixes $Pxy :: 'b \times 'c \Rightarrow \text{real}$ **and** $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes Fx : *finite-entropy* S X Px **and** Fy : *finite-entropy* T Y Py
assumes Fxy : *finite-entropy* $(S \otimes_M T)$ $(\lambda x. (X\ x, Y\ x))$ Pxy
defines $f \equiv \lambda x. Pxy\ x * \log b (Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x)))$
shows *mutual-information-distr'*: *mutual-information* b S T X $Y = \text{integral}^L (S \otimes_M T) f$ (is $?M = ?R$)

and *mutual-information-nonneg'*: $0 \leq \text{mutual-information}$ b S T X Y

proof –

have Px : *distributed* M S X Px **and** Px -nn: $\bigwedge x. x \in \text{space } S \Longrightarrow 0 \leq Px\ x$
using Fx **by** (*auto simp: finite-entropy-def*)
have Py : *distributed* M T Y Py **and** Py -nn: $\bigwedge x. x \in \text{space } T \Longrightarrow 0 \leq Py\ x$
using Fy **by** (*auto simp: finite-entropy-def*)
have Pxy : *distributed* M $(S \otimes_M T)$ $(\lambda x. (X\ x, Y\ x))$ Pxy
and Pxy -nn: $\bigwedge x. x \in \text{space } (S \otimes_M T) \Longrightarrow 0 \leq Pxy\ x$
 $\bigwedge x\ y. x \in \text{space } S \Longrightarrow y \in \text{space } T \Longrightarrow 0 \leq Pxy\ (x, y)$
using Fxy **by** (*auto simp: finite-entropy-def space-pair-measure*)

have [*measurable*]: $Px \in S \rightarrow_M \text{borel}$
using Px Px -nn **by** (*intro distributed-real-measurable*)
have [*measurable*]: $Py \in T \rightarrow_M \text{borel}$
using Py Py -nn **by** (*intro distributed-real-measurable*)
have *measurable-Pxy*[*measurable*]: $Pxy \in (S \otimes_M T) \rightarrow_M \text{borel}$
using Pxy Pxy -nn **by** (*intro distributed-real-measurable (auto simp: space-pair-measure)*)

have X [*measurable*]: *random-variable* S X
using Px **by** *auto*
have Y [*measurable*]: *random-variable* T Y
using Py **by** *auto*
interpret S : *sigma-finite-measure* S **by** *fact*
interpret T : *sigma-finite-measure* T **by** *fact*
interpret ST : *pair-sigma-finite* S T ..
interpret X : *prob-space distr* M S X **using** X **by** (*rule prob-space-distr*)
interpret Y : *prob-space distr* M T Y **using** Y **by** (*rule prob-space-distr*)
interpret XY : *pair-prob-space distr* M S X *distr* M T Y ..
let $?P = S \otimes_M T$
let $?D = \text{distr } M\ ?P (\lambda x. (X\ x, Y\ x))$

{ **fix** A **assume** $A \in \text{sets } S$
with X [*THEN measurable-space*] Y [*THEN measurable-space*]
have *emeasure* (*distr* M S X) $A = \text{emeasure } ?D (A \times \text{space } T)$
by (*auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]*) }
note *marginal-eq1 = this*
{ **fix** A **assume** $A \in \text{sets } T$
with X [*THEN measurable-space*] Y [*THEN measurable-space*]
have *emeasure* (*distr* M T Y) $A = \text{emeasure } ?D (\text{space } S \times A)$
by (*auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]*) }

note *marginal-eq2 = this*

have *distr-eq: distr M S X \otimes_M distr M T Y = density ?P (λx . ennreal (Px (fst x) * Py (snd x)))*
unfolding *Px(1)[THEN distributed-distr-eq-density] Py(1)[THEN distributed-distr-eq-density]*
proof (*subst pair-measure-density*)
show (λx . ennreal (Px x)) \in *borel-measurable S (λy . ennreal (Py y)) \in borel-measurable T*
using *Px Py by (auto simp: distributed-def)*
show *sigma-finite-measure (density T Py) unfolding Py(1)[THEN distributed-distr-eq-density, symmetric] ..*
show *density (S \otimes_M T) ($\lambda(x, y)$. ennreal (Px x) * ennreal (Py y)) = density (S \otimes_M T) (λx . ennreal (Px (fst x) * Py (snd x)))*
using *Px-nn Py-nn by (auto intro!: density-cong simp: distributed-def ennreal-mult space-pair-measure)*
qed fact

have *M: ?M = KL-divergence b (density ?P (λx . ennreal (Px (fst x) * Py (snd x)))) (density ?P (λx . ennreal (Pxy x)))*
unfolding *mutual-information-def distr-eq Pxy(1)[THEN distributed-distr-eq-density]*
..

from *Px Py have f: (λx . Px (fst x) * Py (snd x)) \in borel-measurable ?P*
by (*intro borel-measurable-times (auto intro: distributed-real-measurable measurable-fst'' measurable-snd'')*)
have *PxPy-nonneg: AE x in ?P. 0 \leq Px (fst x) * Py (snd x)*
using *Px-nn Py-nn by (auto simp: space-pair-measure)*

have *A: (AE x in ?P. Px (fst x) = 0 \longrightarrow Pxy x = 0)*
by (*rule subdensity-real[OF measurable-fst Pxy Px] (insert Px-nn Pxy-nn, auto simp: space-pair-measure)*)
moreover
have *B: (AE x in ?P. Py (snd x) = 0 \longrightarrow Pxy x = 0)*
by (*rule subdensity-real[OF measurable-snd Pxy Py] (insert Py-nn Pxy-nn, auto simp: space-pair-measure)*)
ultimately have *ac: AE x in ?P. Px (fst x) * Py (snd x) = 0 \longrightarrow Pxy x = 0*
by *eventually-elim auto*

show *?M = ?R*
unfolding *M f-def using Pxy-nn Px-nn Py-nn*
by (*intro ST.KL-density-density b-gt-1 f PxPy-nonneg ac (auto simp: space-pair-measure)*)

have *X: X = fst \circ (λx . (X x, Y x)) and Y: Y = snd \circ (λx . (X x, Y x))*
by *auto*

have *integrable (S \otimes_M T) (λx . Pxy x * log b (Pxy x) - Pxy x * log b (Px (fst x)) - Pxy x * log b (Py (snd x)))*
using *finite-entropy-integrable[OF Fxy]*
using *finite-entropy-integrable-transform[OF Fx Pxy, of fst]*


```

using finite-entropy-integrable-transform[OF Fy Pxy, of snd]
by (simp add: Pxy-nn)
moreover have  $f \in \text{borel-measurable } (S \otimes_M T)$ 
unfolding f-def using Px Py Pxy
by (auto intro: distributed-real-measurable measurable-fst'' measurable-snd''
  intro!: borel-measurable-times borel-measurable-log borel-measurable-divide)
ultimately have int: integrable  $(S \otimes_M T) f$ 
apply (rule integrable-cong-AE-imp)
using A B AE-space
by eventually-elim
  (auto simp: f-def log-divide-eq log-mult-eq field-simps space-pair-measure Px-nn
  Py-nn Pxy-nn
  less-le)

```

```

show  $0 \leq ?M$  unfolding M
proof (intro ST.KL-density-density-nonneg)
  show prob-space (density  $(S \otimes_M T) (\lambda x. \text{ennreal } (Pxy\ x))$ )
    unfolding distributed-distr-eq-density[OF Pxy, symmetric]
    using distributed-measurable[OF Pxy] by (rule prob-space-distr)
  show prob-space (density  $(S \otimes_M T) (\lambda x. \text{ennreal } (Px\ (\text{fst } x) * Py\ (\text{snd } x)))$ )
    unfolding distr-eq[symmetric] by unfold-locales
  show integrable  $(S \otimes_M T) (\lambda x. Pxy\ x * \log b\ (Pxy\ x / (Px\ (\text{fst } x) * Py\ (\text{snd } x))))$ 
    using int unfolding f-def .
qed (insert ac, auto simp: b-gt-1 Px-nn Py-nn Pxy-nn space-pair-measure)
qed

```

lemma (in information-space)

```

fixes Pxy :: 'b  $\times$  'c  $\Rightarrow$  real and Px :: 'b  $\Rightarrow$  real and Py :: 'c  $\Rightarrow$  real
assumes sigma-finite-measure S sigma-finite-measure T
assumes Px: distributed M S X Px and Px-nn:  $\bigwedge x. x \in \text{space } S \Longrightarrow 0 \leq Px\ x$ 
  and Py: distributed M T Y Py and Py-nn:  $\bigwedge y. y \in \text{space } T \Longrightarrow 0 \leq Py\ y$ 
  and Pxy: distributed M  $(S \otimes_M T) (\lambda x. (X\ x, Y\ x)) Pxy$ 
  and Pxy-nn:  $\bigwedge x\ y. x \in \text{space } S \Longrightarrow y \in \text{space } T \Longrightarrow 0 \leq Pxy\ (x, y)$ 
defines f  $\equiv \lambda x. Pxy\ x * \log b\ (Pxy\ x / (Px\ (\text{fst } x) * Py\ (\text{snd } x)))$ 
shows mutual-information-distr: mutual-information b S T X Y = integralL  $(S \otimes_M T) f$  (is ?M = ?R)
and mutual-information-nonneg: integrable  $(S \otimes_M T) f \Longrightarrow 0 \leq \text{mutual-information } b\ S\ T\ X\ Y$ 

```

proof –

```

have X[measurable]: random-variable S X
  using Px by (auto simp: distributed-def)
have Y[measurable]: random-variable T Y
  using Py by (auto simp: distributed-def)
have [measurable]:  $Px \in S \rightarrow_M \text{borel}$ 
  using Px Px-nn by (intro distributed-real-measurable)
have [measurable]:  $Py \in T \rightarrow_M \text{borel}$ 
  using Py Py-nn by (intro distributed-real-measurable)
have measurable-Pxy[measurable]:  $Pxy \in (S \otimes_M T) \rightarrow_M \text{borel}$ 

```

```

using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..
interpret X: prob-space distr M S X using X by (rule prob-space-distr)
interpret Y: prob-space distr M T Y using Y by (rule prob-space-distr)
interpret XY: pair-prob-space distr M S X distr M T Y ..
let ?P = S  $\otimes_M$  T
let ?D = distr M ?P ( $\lambda x. (X x, Y x)$ )

{ fix A assume A  $\in$  sets S
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M S X) A = emeasure ?D (A  $\times$  space T)
  by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq1 = this
{ fix A assume A  $\in$  sets T
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M T Y) A = emeasure ?D (space S  $\times$  A)
  by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq2 = this

have distr-eq: distr M S X  $\otimes_M$  distr M T Y = density ?P ( $\lambda x. \text{ennreal } (Px$ 
( $\text{fst } x) * Py (\text{snd } x))$ )
  unfolding Px(1)[THEN distributed-distr-eq-density] Py(1)[THEN distributed-distr-eq-density]
  proof (subst pair-measure-density)
    show ( $\lambda x. \text{ennreal } (Px x) \in \text{borel-measurable } S$  ( $\lambda y. \text{ennreal } (Py y) \in$ 
borel-measurable T
  using Px Py by (auto simp: distributed-def)
  show sigma-finite-measure (density T Py) unfolding Py(1)[THEN distributed-distr-eq-density,
symmetric] ..
  show density (S  $\otimes_M$  T) ( $\lambda(x, y). \text{ennreal } (Px x) * \text{ennreal } (Py y) =$ 
density (S  $\otimes_M$  T) ( $\lambda x. \text{ennreal } (Px (\text{fst } x) * Py (\text{snd } x))$ )
  using Px-nn Py-nn by (auto intro!: density-cong simp: distributed-def ennreal-mult
space-pair-measure)
  qed fact

have M: ?M = KL-divergence b (density ?P ( $\lambda x. \text{ennreal } (Px (\text{fst } x) * Py (\text{snd } x))$ 
( $\text{density } ?P (\lambda x. \text{ennreal } (Pxy x))$ ))
  unfolding mutual-information-def distr-eq Pxy(1)[THEN distributed-distr-eq-density]
  ..

from Px Py have f: ( $\lambda x. Px (\text{fst } x) * Py (\text{snd } x) \in \text{borel-measurable } ?P$ 
by (intro borel-measurable-times) (auto intro: distributed-real-measurable measurable-fst''
measurable-snd''))
have PxPy-nonneg:  $\text{AE } x \text{ in } ?P. 0 \leq Px (\text{fst } x) * Py (\text{snd } x)$ 
  using Px-nn Py-nn by (auto simp: space-pair-measure)

have ( $\text{AE } x \text{ in } ?P. Px (\text{fst } x) = 0 \longrightarrow Pxy x = 0$ )

```

by (rule *subdensity-real*[*OF measurable-fst Pxy Px*]) (insert *Px-nn Pxy-nn*, auto *simp: space-pair-measure*)

moreover

have (*AE x in ?P. Py (snd x) = 0 \longrightarrow Pxy x = 0*)

by (rule *subdensity-real*[*OF measurable-snd Pxy Py*]) (insert *Py-nn Pxy-nn*, auto *simp: space-pair-measure*)

ultimately have *ac: AE x in ?P. Px (fst x) * Py (snd x) = 0 \longrightarrow Pxy x = 0*
by *eventually-elim auto*

show *?M = ?R*

unfolding *M f-def*

using *b-gt-1 f PxPy-nonneg ac Pxy-nn*

by (*intro ST.KL-density-density*) (auto *simp: space-pair-measure*)

assume *int: integrable (S \otimes_M T) f*

show *0 \leq ?M unfolding M*

proof (*intro ST.KL-density-density-nonneg*)

show *prob-space (density (S \otimes_M T) ($\lambda x. ennreal (Pxy x)$))*

unfolding *distributed-distr-eq-density*[*OF Pxy, symmetric*]

using *distributed-measurable*[*OF Pxy*] **by** (rule *prob-space-distr*)

show *prob-space (density (S \otimes_M T) ($\lambda x. ennreal (Px (fst x) * Py (snd x))$))*

unfolding *distr-eq*[*symmetric*] **by** *unfold-locales*

show *integrable (S \otimes_M T) ($\lambda x. Pxy x * \log b (Pxy x / (Px (fst x) * Py (snd x)))$)*

using *int unfolding f-def .*

qed (*insert ac, auto simp: b-gt-1 Px-nn Py-nn Pxy-nn space-pair-measure*)

qed

lemma (*in information-space*)

fixes *Pxy :: 'b \times 'c \Rightarrow real and Px :: 'b \Rightarrow real and Py :: 'c \Rightarrow real*

assumes *sigma-finite-measure S sigma-finite-measure T*

assumes *Px[measurable]: distributed M S X Px and Px-nn: $\bigwedge x. x \in \text{space } S \implies 0 \leq Px x$*

and *Py[measurable]: distributed M T Y Py and Py-nn: $\bigwedge x. x \in \text{space } T \implies 0 \leq Py x$*

and *Pxy[measurable]: distributed M (S \otimes_M T) ($\lambda x. (X x, Y x)) Pxy$*

and *Pxy-nn: $\bigwedge x. x \in \text{space } (S \otimes_M T) \implies 0 \leq Pxy x$*

assumes *ae: AE x in S. AE y in T. Pxy (x, y) = Px x * Py y*

shows *mutual-information-eq-0: mutual-information b S T X Y = 0*

proof –

interpret *S: sigma-finite-measure S by fact*

interpret *T: sigma-finite-measure T by fact*

interpret *ST: pair-sigma-finite S T ..*

note

distributed-real-measurable[*OF Px-nn Px, measurable*]

distributed-real-measurable[*OF Py-nn Py, measurable*]

distributed-real-measurable[*OF Pxy-nn Pxy, measurable*]

have *AE x in S \otimes_M T. Px (fst x) = 0 \longrightarrow Pxy x = 0*

by (*rule subdensity-real*[*OF measurable-fst Pxy Px*]) (*auto simp: Px-nn Pxy-nn space-pair-measure*)
moreover
have $AE\ x\ in\ S \otimes_M T. Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$
by (*rule subdensity-real*[*OF measurable-snd Pxy Py*]) (*auto simp: Py-nn Pxy-nn space-pair-measure*)
moreover
have $AE\ x\ in\ S \otimes_M T. Pxy\ x = Px\ (fst\ x) * Py\ (snd\ x)$
by (*intro ST.AE-pair-measure*) (*auto simp: ae intro!: measurable-snd'' measurable-fst''*)
ultimately have $AE\ x\ in\ S \otimes_M T. Pxy\ x * \log\ b\ (Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x))) = 0$
by *eventually-elim simp*
then have $(\int x. Pxy\ x * \log\ b\ (Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x))))\ \partial(S \otimes_M T) = (\int x. 0\ \partial(S \otimes_M T))$
by (*intro integral-cong-AE*) *auto*
then show *?thesis*
by (*subst mutual-information-distr*[*OF assms(1-8)*]) (*auto simp add: space-pair-measure*)
qed

lemma (*in information-space*) *mutual-information-simple-distributed:*

assumes $X: simple-distributed\ M\ X\ Px$ **and** $Y: simple-distributed\ M\ Y\ Py$
assumes $XY: simple-distributed\ M\ (\lambda x. (X\ x, Y\ x))\ Pxy$
shows $\mathcal{I}(X ; Y) = (\sum (x, y) \in (\lambda x. (X\ x, Y\ x))\ 'space\ M. Pxy\ (x, y) * \log\ b\ (Pxy\ (x, y) / (Px\ x * Py\ y)))$
proof (*subst mutual-information-distr*[*OF - - simple-distributed*[*OF X*] - *simple-distributed*[*OF Y*] - *simple-distributed-joint*[*OF XY*]])
note $fin = simple-distributed-joint-finite$ [*OF XY, simp*]
show *sigma-finite-measure* (*count-space* ($X\ 'space\ M$))
by (*simp add: sigma-finite-measure-count-space-finite*)
show *sigma-finite-measure* (*count-space* ($Y\ 'space\ M$))
by (*simp add: sigma-finite-measure-count-space-finite*)
let $?Pxy = \lambda x. (if\ x \in (\lambda x. (X\ x, Y\ x))\ 'space\ M\ then\ Pxy\ x\ else\ 0)$
let $?f = \lambda x. ?Pxy\ x * \log\ b\ (?Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x)))$
have $\bigwedge x. ?f\ x = (if\ x \in (\lambda x. (X\ x, Y\ x))\ 'space\ M\ then\ Pxy\ x * \log\ b\ (Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x)))\ else\ 0)$
by *auto*
with fin **show** $(\int x. ?f\ x\ \partial(count-space\ (X\ 'space\ M) \otimes_M\ count-space\ (Y\ 'space\ M))) =$
 $(\sum (x, y) \in (\lambda x. (X\ x, Y\ x))\ 'space\ M. Pxy\ (x, y) * \log\ b\ (Pxy\ (x, y) / (Px\ x * Py\ y)))$
by (*auto simp add: pair-measure-count-space lebesgue-integral-count-space-finite sum.If-cases split-beta'*)
intro!: sum.cong
qed (*insert X Y XY, auto simp: simple-distributed-def*)

lemma (*in information-space*)

fixes $Pxy :: 'b \times 'c \Rightarrow real$ **and** $Px :: 'b \Rightarrow real$ **and** $Py :: 'c \Rightarrow real$
assumes $Px: simple-distributed\ M\ X\ Px$ **and** $Py: simple-distributed\ M\ Y\ Py$
assumes $Pxy: simple-distributed\ M\ (\lambda x. (X\ x, Y\ x))\ Pxy$

assumes $ae: \forall x \in \text{space } M. Pxy (X x, Y x) = Px (X x) * Py (Y x)$
shows *mutual-information-eq-0-simple*: $\mathcal{I}(X ; Y) = 0$
proof (*subst mutual-information-simple-distributed*[*OF Px Py Pxy*])
have $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ 'space } M. Pxy (x, y) * \log b (Pxy (x, y) / (Px x * Py y))) =$
 $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ 'space } M. 0)$
by (*intro sum.cong*) (*auto simp: ae*)
then show $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ 'space } M.$
 $Pxy (x, y) * \log b (Pxy (x, y) / (Px x * Py y))) = 0$ **by** *simp*
qed

14.5 Entropy

definition (*in prob-space*) *entropy* :: $\text{real} \Rightarrow \text{'b measure} \Rightarrow (\text{'a} \Rightarrow \text{'b}) \Rightarrow \text{real}$ **where**
 $\text{entropy } b \text{ } S \text{ } X = - \text{KL-divergence } b \text{ } S \text{ } (\text{distr } M \text{ } S \text{ } X)$

abbreviation (*in information-space*)
 $\text{entropy-Pow } (\mathcal{H}'(-))$ **where**
 $\mathcal{H}(X) \equiv \text{entropy } b \text{ } (\text{count-space } (X \text{ 'space } M)) \text{ } X$

lemma (*in prob-space*) *distributed-RN-deriv*:

assumes $X: \text{distributed } M \text{ } S \text{ } X \text{ } Px$

shows $AE \text{ } x \text{ in } S. \text{RN-deriv } S \text{ } (\text{density } S \text{ } Px) \text{ } x = Px \text{ } x$

proof –

note $D = \text{distributed-measurable}[OF X] \text{ distributed-borel-measurable}[OF X]$

interpret $X: \text{prob-space } \text{distr } M \text{ } S \text{ } X$

using $D(1)$ **by** (*rule prob-space-distr*)

have $sf: \text{sigma-finite-measure } (\text{distr } M \text{ } S \text{ } X)$ **by** *standard*

show *?thesis*

using D

apply (*subst eq-commute*)

apply (*intro RN-deriv-unique-sigma-finite*)

apply (*auto simp: distributed-distr-eq-density[symmetric, OF X] sf*)

done

qed

lemma (*in information-space*)

fixes $X :: \text{'a} \Rightarrow \text{'b}$

assumes $X[\text{measurable}]: \text{distributed } M \text{ } MX \text{ } X \text{ } f$ **and** $nn: \bigwedge x. x \in \text{space } MX \implies 0 \leq f \text{ } x$

shows *entropy-distr*: $\text{entropy } b \text{ } MX \text{ } X = - (\int x. f \text{ } x * \log b (f \text{ } x) \partial MX)$ (*is ?eq*)

proof –

note $D = \text{distributed-measurable}[OF X] \text{ distributed-borel-measurable}[OF X]$

note $ae = \text{distributed-RN-deriv}[OF X]$

note $\text{distributed-real-measurable}[OF nn \text{ } X, \text{measurable}]$

have *ae-eq*: $AE \text{ } x \text{ in } \text{distr } M \text{ } MX \text{ } X. \log b (\text{enn2real } (\text{RN-deriv } MX \text{ } (\text{distr } M \text{ } MX \text{ } X) \text{ } x)) =$

```

log b (f x)
unfolding distributed-distr-eq-density[OF X]
apply (subst AE-density)
using D apply simp
using ae apply eventually-elim
apply auto
done

have int-eq: (∫ x. f x * log b (f x) ∂MX) = (∫ x. log b (f x) ∂distr M MX X)
unfolding distributed-distr-eq-density[OF X]
using D
by (subst integral-density) (auto simp: nn)

show ?eq
unfolding entropy-def KL-divergence-def entropy-density-def comp-def int-eq
neg-equal-iff-equal
using ae-eq by (intro integral-cong-AE) (auto simp: nn)
qed

lemma (in information-space) entropy-le:
fixes Px :: 'b ⇒ real and MX :: 'b measure
assumes X[measurable]: distributed M MX X Px and Px-nn[simp]:  $\bigwedge x. x \in$ 
space MX  $\implies 0 \leq Px x$ 
and fin: emeasure MX  $\{x \in \text{space } MX. Px x \neq 0\} \neq \text{top}$ 
and int: integrable MX  $(\lambda x. - Px x * \log b (Px x))$ 
shows entropy b MX X  $\leq \log b (\text{measure } MX \{x \in \text{space } MX. Px x \neq 0\})$ 
proof -
note Px = distributed-borel-measurable[OF X]
interpret X: prob-space distr M MX X
using distributed-measurable[OF X] by (rule prob-space-distr)

have -  $\log b (\text{measure } MX \{x \in \text{space } MX. Px x \neq 0\}) =$ 
 $-\log b (\int x. \text{indicator } \{x \in \text{space } MX. Px x \neq 0\} x \partial MX)$ 
using Px Px-nn fin by (auto simp: measure-def)
also have  $-\log b (\int x. \text{indicator } \{x \in \text{space } MX. Px x \neq 0\} x \partial MX) = -\log$ 
 $b (\int x. 1 / Px x \partial \text{distr } M MX X)$ 
unfolding distributed-distr-eq-density[OF X] using Px Px-nn
apply (intro arg-cong[where f=log b] arg-cong[where f=uminus])
by (subst integral-density) (auto simp del: integral-indicator intro!: Bochner-Integration.integral-cong)
also have  $\dots \leq (\int x. -\log b (1 / Px x) \partial \text{distr } M MX X)$ 
proof (rule X.jensens-inequality[of  $\lambda x. 1 / Px x \{0<..\} 0 1 \lambda x. -\log b x$ ])
show AE x in distr M MX X.  $1 / Px x \in \{0<..\}$ 
unfolding distributed-distr-eq-density[OF X]
using Px by (auto simp: AE-density)
have [simp]:  $\bigwedge x. x \in \text{space } MX \implies \text{ennreal } (\text{if } Px x = 0 \text{ then } 0 \text{ else } 1) =$ 
indicator  $\{x \in \text{space } MX. Px x \neq 0\} x$ 
by (auto simp: one-ennreal-def)
have  $(\int^+ x. \text{ennreal } (- (\text{if } Px x = 0 \text{ then } 0 \text{ else } 1)) \partial MX) = (\int^+ x. 0 \partial MX)$ 
by (intro nn-integral-cong) (auto simp: ennreal-neg)

```

```

then show integrable (distr M MX X) ( $\lambda x. 1 / Px x$ )
  unfolding distributed-distr-eq-density[OF X] using Px
by (auto simp: nn-integral-density real-integrable-def fin ennreal-neg ennreal-mult[symmetric]
      cong: nn-integral-cong)
have integrable MX ( $\lambda x. Px x * \log b (1 / Px x)$ ) =
  integrable MX ( $\lambda x. - Px x * \log b (Px x)$ )
  using Px
  by (intro integrable-cong-AE) (auto simp: log-divide-eq less-le)
then show integrable (distr M MX X) ( $\lambda x. - \log b (1 / Px x)$ )
  unfolding distributed-distr-eq-density[OF X]
  using Px int
  by (subst integrable-real-density) auto
qed (auto simp: minus-log-convex[OF b-gt-1])
also have ... = ( $\int x. \log b (Px x) \partial \text{distr } M \text{ MX } X$ )
  unfolding distributed-distr-eq-density[OF X] using Px
  by (intro integral-cong-AE) (auto simp: AE-density log-divide-eq)
also have ... = - entropy b MX X
  unfolding distributed-distr-eq-density[OF X] using Px
  by (subst entropy-distr[OF X]) (auto simp: integral-density)
finally show ?thesis
  by simp
qed

```

lemma (in information-space) entropy-le-space:

```

fixes Px :: 'b  $\Rightarrow$  real and MX :: 'b measure
assumes X: distributed M MX X Px and Px-nn[simp]:  $\bigwedge x. x \in \text{space } MX \Rightarrow 0 \leq Px x$ 
and fin: finite-measure MX
and int: integrable MX ( $\lambda x. - Px x * \log b (Px x)$ )
shows entropy b MX X  $\leq \log b (\text{measure } MX (\text{space } MX))$ 
proof -
note Px = distributed-borel-measurable[OF X]
interpret finite-measure MX by fact
have entropy b MX X  $\leq \log b (\text{measure } MX \{x \in \text{space } MX. Px x \neq 0\})$ 
  using int X by (intro entropy-le) auto
also have ...  $\leq \log b (\text{measure } MX (\text{space } MX))$ 
  using Px distributed-imp-emeasure-nonzero[OF X]
  by (intro log-le)
  (auto intro!: finite-measure-mono b-gt-1 less-le[THEN iffD2]
      simp: emeasure-eq-measure cong: conj-cong)
finally show ?thesis .
qed

```

lemma (in information-space) entropy-uniform:

```

assumes X: distributed M MX X ( $\lambda x. \text{indicator } A x / \text{measure } MX A$ ) (is
distributed - - - ?f)
shows entropy b MX X =  $\log b (\text{measure } MX A)$ 
proof (subst entropy-distr[OF X])
have [simp]: emeasure MX A  $\neq \infty$ 

```

```

using uniform-distributed-params[OF X] by (auto simp add: measure-def)
have eq: ( $\int x. \text{indicator } A \ x \ / \ \text{measure } MX \ A \ * \ \log \ b \ (\text{indicator } A \ x \ / \ \text{measure } MX \ A) \ \partial MX$ ) =
  ( $\int x. (- \log \ b \ (\text{measure } MX \ A) \ / \ \text{measure } MX \ A) \ * \ \text{indicator } A \ x \ \partial MX$ )
using uniform-distributed-params[OF X]
by (intro Bochner-Integration.integral-cong) (auto split: split-indicator simp:
log-divide-eq zero-less-measure-iff)
show - ( $\int x. \text{indicator } A \ x \ / \ \text{measure } MX \ A \ * \ \log \ b \ (\text{indicator } A \ x \ / \ \text{measure } MX \ A) \ \partial MX$ ) =
   $\log \ b \ (\text{measure } MX \ A)$ 
unfolding eq using uniform-distributed-params[OF X]
by (subst Bochner-Integration.integral-mult-right) (auto simp: measure-def less-top[symmetric]
intro!: integrable-real-indicator)
qed simp

```

```

lemma (in information-space) entropy-simple-distributed:
  simple-distributed M X f  $\implies \mathcal{H}(X) = - (\sum x \in X \text{'space } M. f \ x \ * \ \log \ b \ (f \ x))$ 
by (subst entropy-distr[OF simple-distributed])
  (auto simp add: lebesgue-integral-count-space-finite)

```

```

lemma (in information-space) entropy-le-card-not-0:
  assumes X: simple-distributed M X f
  shows  $\mathcal{H}(X) \leq \log \ b \ (\text{card } (X \text{'space } M \cap \{x. f \ x \neq 0\}))$ 
proof -
  let ?X = count-space (X'space M)
  have  $\mathcal{H}(X) \leq \log \ b \ (\text{measure } ?X \ \{x \in \text{space } ?X. f \ x \neq 0\})$ 
  by (rule entropy-le[OF simple-distributed[OF X]])
  (insert X, auto simp: simple-distributed-finite[OF X] subset-eq integrable-count-space
emeasure-count-space)
  also have  $\text{measure } ?X \ \{x \in \text{space } ?X. f \ x \neq 0\} = \text{card } (X \text{'space } M \cap \{x. f \ x \neq 0\})$ 
  by (simp-all add: simple-distributed-finite[OF X] subset-eq emeasure-count-space
measure-def Int-def)
  finally show ?thesis .
qed

```

```

lemma (in information-space) entropy-le-card:
  assumes X: simple-distributed M X f
  shows  $\mathcal{H}(X) \leq \log \ b \ (\text{real } (\text{card } (X \text{'space } M)))$ 
proof -
  let ?X = count-space (X'space M)
  have  $\mathcal{H}(X) \leq \log \ b \ (\text{measure } ?X \ (\text{space } ?X))$ 
  by (rule entropy-le-space[OF simple-distributed[OF X]])
  (insert X, auto simp: simple-distributed-finite[OF X] subset-eq integrable-count-space
emeasure-count-space finite-measure-count-space)
  also have  $\text{measure } ?X \ (\text{space } ?X) = \text{card } (X \text{'space } M)$ 
  by (simp-all add: simple-distributed-finite[OF X] subset-eq emeasure-count-space
measure-def)
  finally show ?thesis .

```


qed

14.6 Conditional Mutual Information

definition (in *prob-space*)

conditional-mutual-information b MX MY MZ X Y $Z \equiv$
mutual-information b MX $(MY \otimes_M MZ)$ X $(\lambda x. (Y x, Z x)) -$
mutual-information b MX MZ X Z

abbreviation (in *information-space*)

conditional-mutual-information-Pow $(\mathcal{I}'(-; - | - '))$ **where**
 $\mathcal{I}(X ; Y | Z) \equiv$ *conditional-mutual-information* b
(count-space $(X$ ‘ *space* $M)$ *(count-space* $(Y$ ‘ *space* $M)$ *(count-space* $(Z$ ‘ *space* $M)$ $)$ X Y Z

lemma (in *information-space*)

assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T **and** P : *sigma-finite-measure* P

assumes Px [*measurable*]: *distributed* M S X Px

and Px -*nn[simp]*: $\bigwedge x. x \in \text{space } S \implies 0 \leq Px x$

assumes Pz [*measurable*]: *distributed* M P Z Pz

and Pz -*nn[simp]*: $\bigwedge z. z \in \text{space } P \implies 0 \leq Pz z$

assumes Pyz [*measurable*]: *distributed* M $(T \otimes_M P)$ $(\lambda x. (Y x, Z x))$ Pyz

and Pyz -*nn[simp]*: $\bigwedge y z. y \in \text{space } T \implies z \in \text{space } P \implies 0 \leq Pyz (y, z)$

assumes Pxz [*measurable*]: *distributed* M $(S \otimes_M P)$ $(\lambda x. (X x, Z x))$ Pxz

and Pxz -*nn[simp]*: $\bigwedge x z. x \in \text{space } S \implies z \in \text{space } P \implies 0 \leq Pxz (x, z)$

assumes $Pxyz$ [*measurable*]: *distributed* M $(S \otimes_M T \otimes_M P)$ $(\lambda x. (X x, Y x, Z x))$ $Pxyz$

and $Pxyz$ -*nn[simp]*: $\bigwedge x y z. x \in \text{space } S \implies y \in \text{space } T \implies z \in \text{space } P \implies 0 \leq Pxyz (x, y, z)$

assumes $I1$: *integrable* $(S \otimes_M T \otimes_M P)$ $(\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Px x * Pyz (y, z))))$

assumes $I2$: *integrable* $(S \otimes_M T \otimes_M P)$ $(\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxz (x, z) / (Px x * Pz z)))$

shows *conditional-mutual-information-generic-eq*: *conditional-mutual-information* b S T P X Y Z

$= (\int (x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))) \partial(S \otimes_M T \otimes_M P))$ (**is ?eq**)

and *conditional-mutual-information-generic-nonneg*: $0 \leq$ *conditional-mutual-information* b S T P X Y Z (**is ?nonneg**)

proof –

have [*measurable*]: $Px \in S \rightarrow_M$ *borel*

using Px Px -*nn* **by** (*intro distributed-real-measurable*)

have [*measurable*]: $Pz \in P \rightarrow_M$ *borel*

using Pz Pz -*nn* **by** (*intro distributed-real-measurable*)

have *measurable-Pyz*[*measurable*]: $Pyz \in (T \otimes_M P) \rightarrow_M$ *borel*

using Pyz Pyz -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

have *measurable-Pxz*[*measurable*]: $Pxz \in (S \otimes_M P) \rightarrow_M$ *borel*

using Pxz Pxz -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

have *measurable-Pxyz*[*measurable*]: $Pxyz \in (S \otimes_M T \otimes_M P) \rightarrow_M \text{borel}$
using *Pxyz Pxyz-nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

interpret *S*: *sigma-finite-measure S* **by fact**
interpret *T*: *sigma-finite-measure T* **by fact**
interpret *P*: *sigma-finite-measure P* **by fact**
interpret *TP*: *pair-sigma-finite T P* ..
interpret *SP*: *pair-sigma-finite S P* ..
interpret *ST*: *pair-sigma-finite S T* ..
interpret *SPT*: *pair-sigma-finite S* \otimes_M *P T* ..
interpret *STP*: *pair-sigma-finite S T* \otimes_M *P* ..
interpret *TPS*: *pair-sigma-finite T* \otimes_M *P S* ..
have *TP*: *sigma-finite-measure (T* \otimes_M *P)* ..
have *SP*: *sigma-finite-measure (S* \otimes_M *P)* ..
have *YZ*: *random-variable (T* \otimes_M *P)* ($\lambda x. (Y x, Z x)$)
using *Pyz* **by** (*simp add: distributed-measurable*)

from *Pxz Pxyz* **have** *distr-eq*: $\text{distr } M (S \otimes_M P) (\lambda x. (X x, Z x)) =$
 $\text{distr } (\text{distr } M (S \otimes_M T \otimes_M P) (\lambda x. (X x, Y x, Z x))) (S \otimes_M P) (\lambda(x, y,$
 $z). (x, z))$
by (*simp add: comp-def distr-distr*)

have *mutual-information b S P X Z* =
 $(\int x. Pxz x * \log b (Pxz x / (Px (fst x) * Pz (snd x)))) \partial(S \otimes_M P)$
by (*rule mutual-information-distr[OF S P Px Px-nn Pz Pz-nn Pxz Pxz-nn]*)
also have ... = $(\int (x,y,z). Pxyz (x,y,z) * \log b (Pxz (x,z) / (Px x * Pz z))) \partial(S$
 $\otimes_M T \otimes_M P)$
using *b-gt-1 Pxz Px Pz*
by (*subst distributed-transform-integral[OF Pxyz - Pxz -, where T= $\lambda(x, y, z).$*
 $(x, z)]$)
(auto simp: split-beta' space-pair-measure)

finally have *mi-eq*:
 $\text{mutual-information } b S P X Z = (\int (x,y,z). Pxyz (x,y,z) * \log b (Pxz (x,z) /$
 $(Px x * Pz z))) \partial(S \otimes_M T \otimes_M P)$.

have *ae1*: $AE x \text{ in } S \otimes_M T \otimes_M P. Px (fst x) = 0 \longrightarrow Pxyz x = 0$
by (*intro subdensity-real[of fst, OF - Pxyz Px]*) (*auto simp: space-pair-measure*)
moreover have *ae2*: $AE x \text{ in } S \otimes_M T \otimes_M P. Pz (snd (snd x)) = 0 \longrightarrow$
 $Pxyz x = 0$
by (*intro subdensity-real[of $\lambda x. snd (snd x)$, OF - Pxyz Pz]*) (*auto simp:*
space-pair-measure)
moreover have *ae3*: $AE x \text{ in } S \otimes_M T \otimes_M P. Pxz (fst x, snd (snd x)) = 0$
 $\longrightarrow Pxyz x = 0$
by (*intro subdensity-real[of $\lambda x. (fst x, snd (snd x))$, OF - Pxyz Pxz]*) (*auto*
simp: space-pair-measure)
moreover have *ae4*: $AE x \text{ in } S \otimes_M T \otimes_M P. Pyz (snd x) = 0 \longrightarrow Pxyz x$
 $= 0$
by (*intro subdensity-real[of snd, OF - Pxyz Pyz]*) (*auto simp: space-pair-measure*)
ultimately have *ae*: $AE x \text{ in } S \otimes_M T \otimes_M P.$

```

    Pxyz x * log b (Pxyz x / (Px (fst x) * Pyz (snd x))) -
    Pxyz x * log b (Pxz (fst x, snd (snd x)) / (Px (fst x) * Pz (snd (snd x)))) =
    Pxyz x * log b (Pxyz x * Pz (snd (snd x)) / (Pxz (fst x, snd (snd x)) * Pyz
(snd x)))
  using AE-space
  proof eventually-elim
  case (elim x)
  show ?case
  proof cases
  assume Pxyz x ≠ 0
  with elim have 0 < Px (fst x) 0 < Pz (snd (snd x)) 0 < Pxz (fst x, snd
(snd x))
    0 < Pyz (snd x) 0 < Pxyz x
    by (auto simp: space-pair-measure less-le)
  then show ?thesis
  using b-gt-1 by (simp add: log-simps less-imp-le field-simps)
  qed simp
  qed
  with I1 I2 show ?eq
  unfolding conditional-mutual-information-def
  apply (subst mi-eq)
  apply (subst mutual-information-distr[OF S TP Px Px-nn Pyz - Pxyz])
  apply (auto simp: space-pair-measure)
  apply (subst Bochner-Integration.integral-diff[symmetric])
  apply (auto intro!: integral-cong-AE simp: split-beta' simp del: Bochner-Integration.integral-diff)
  done

  let ?P = density (S ⊗M T ⊗M P) Pxyz
  interpret P: prob-space ?P
  unfolding distributed-distr-eq-density[OF Pxyz, symmetric]
  by (rule prob-space-distr) simp

  let ?Q = density (T ⊗M P) Pyz
  interpret Q: prob-space ?Q
  unfolding distributed-distr-eq-density[OF Pyz, symmetric]
  by (rule prob-space-distr) simp

  let ?f = λ(x, y, z). Pxz (x, z) * (Pyz (y, z) / Pz z) / Pxyz (x, y, z)

  from subdensity-real[of snd, OF - Pyz Pz - AE-I2 AE-I2]
  have aeX1: AE x in T ⊗M P. Pz (snd x) = 0 → Pyz x = 0
  by (auto simp: comp-def space-pair-measure)
  have aeX2: AE x in T ⊗M P. 0 ≤ Pz (snd x)
  using Pz by (intro TP.AE-pair-measure) (auto simp: comp-def)

  have aeX3: AE y in T ⊗M P. (∫+ x. ennreal (Pxz (x, snd y)) ∂S) = ennreal
(Pz (snd y))
  using Pz distributed-marginal-eq-joint2[OF P S Pz Pxz]
  by (intro TP.AE-pair-measure) auto

```

have $(\int^+ x. ?f x \partial?P) \leq (\int^+ (x, y, z). Pxz (x, z) * (Pyz (y, z) / Pz z) \partial(S \otimes_M T \otimes_M P))$
by (*subst nn-integral-density*)
(auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
also have $\dots = (\int^+(y, z). (\int^+ x. ennreal (Pxz (x, z)) * ennreal (Pyz (y, z) / Pz z) \partial S) \partial(T \otimes_M P))$
by (*subst STP.nn-integral-snd[symmetric]*)
(auto simp add: split-beta' ennreal-mult[symmetric] space-pair-measure intro!: nn-integral-cong)
also have $\dots = (\int^+ x. ennreal (Pyz x) * 1 \partial T \otimes_M P)$
apply (*rule nn-integral-cong-AE*)
using *aeX1 aeX2 aeX3 AE-space*
apply *eventually-elim*
proof (*case-tac x, simp add: space-pair-measure*)
fix *a b* **assume** $Pz b = 0 \longrightarrow Pyz (a, b) = 0$ *a* \in *space* *T* \wedge *b* \in *space* *P*
 $(\int^+ x. ennreal (Pxz (x, b)) \partial S) = ennreal (Pz b)$
then show $(\int^+ x. ennreal (Pxz (x, b)) * ennreal (Pyz (a, b) / Pz b) \partial S) =$
 $ennreal (Pyz (a, b))$
by (*subst nn-integral-multc*) (*auto split: prod.split simp: ennreal-mult[symmetric]*)
qed
also have $\dots = 1$
using *Q.emmeasure-space-1 distributed-distr-eq-density[OF Pyz]*
by (*subst nn-integral-density[symmetric]*) *auto*
finally have *le1*: $(\int^+ x. ?f x \partial?P) \leq 1$.
also have $\dots < \infty$ **by** *simp*
finally have *fin*: $(\int^+ x. ?f x \partial?P) \neq \infty$ **by** *simp*

have *pos*: $(\int^+ x. ?f x \partial?P) \neq 0$
apply (*subst nn-integral-density*)
apply (*simp-all add: split-beta'*)
proof
let *?g* $= \lambda x. ennreal (Pxyz x) * (Pxz (fst x, snd (snd x)) * Pyz (snd x) / (Pz (snd (snd x)) * Pxyz x))$
assume $(\int^+ x. ?g x \partial(S \otimes_M T \otimes_M P)) = 0$
then have *AE* *x* *in* *S* \otimes_M *T* \otimes_M *P*. *?g* *x* $= 0$
by (*intro nn-integral-0-iff-AE[THEN iffD1]*) *auto*
then have *AE* *x* *in* *S* \otimes_M *T* \otimes_M *P*. *Pxyz* *x* $= 0$
using *ae1 ae2 ae3 ae4 AE-space*
by *eventually-elim* (*auto split: if-split-asm simp: mult-le-0-iff divide-le-0-iff space-pair-measure*)
then have $(\int^+ x. ennreal (Pxyz x) \partial S \otimes_M T \otimes_M P) = 0$
by (*subst nn-integral-cong-AE[of - $\lambda x. 0$]*) *auto*
with *P.emmeasure-space-1* **show** *False*
by (*subst (asm) emmeasure-density*) (*auto cong: nn-integral-cong*)
qed

have *neg*: $(\int^+ x. - ?f x \partial?P) = 0$
apply (*rule nn-integral-0-iff-AE[THEN iffD2]*)

```

apply simp
apply (subst AE-density)
apply (auto simp: space-pair-measure ennreal-neg)
done

have I3: integrable (S  $\otimes_M$  T  $\otimes_M$  P) ( $\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz$ 
( $x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))))$ )
apply (rule integrable-cong-AE[THEN iffD1, OF - - - Bochner-Integration.integrable-diff[OF
I1 I2]])
using ae
apply (auto simp: split-beta')
done

have  $-\log b 1 \leq -\log b (\text{integral}^L ?P ?f)$ 
proof (intro le-imp-neg-le log-le[OF b-gt-1])
have If: integrable ?P ?f
unfolding real-integrable-def
proof (intro conjI)
from neg show  $(\int^+ x. - ?f x \partial ?P) \neq \infty$ 
by simp
from fin show  $(\int^+ x. ?f x \partial ?P) \neq \infty$ 
by simp
qed simp
then have  $(\int^+ x. ?f x \partial ?P) = (\int x. ?f x \partial ?P)$ 
apply (rule nn-integral-eq-integral)
apply (subst AE-density)
apply simp
apply (auto simp: space-pair-measure ennreal-neg)
done
with pos le1
show  $0 < (\int x. ?f x \partial ?P) (\int x. ?f x \partial ?P) \leq 1$ 
by (simp-all add: one-ennreal.rep-eq zero-less-iff-neq-zero[symmetric])
qed
also have  $-\log b (\text{integral}^L ?P ?f) \leq (\int x. -\log b (?f x) \partial ?P)$ 
proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
show AE x in ?P.  $?f x \in \{0<..\}$ 
unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
using ae1 ae2 ae3 ae4 AE-space
by eventually-elim (auto simp: space-pair-measure less-le)
show integrable ?P ?f
unfolding real-integrable-def
using fin neg by (auto simp: split-beta')
show integrable ?P ( $\lambda x. -\log b (?f x)$ )
apply (subst integrable-real-density)
apply simp
apply (auto simp: space-pair-measure) []
apply simp
apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
apply simp

```

```

apply simp
using ae1 ae2 ae3 ae4 AE-space
apply eventually-elim
apply (auto simp: log-divide-eq log-mult-eq zero-le-mult-iff zero-less-mult-iff
zero-less-divide-iff field-simps
less-le space-pair-measure)
done
qed (auto simp: b-gt-1 minus-log-convex)
also have ... = conditional-mutual-information b S T P X Y Z
unfolding ⟨?eq⟩
apply (subst integral-real-density)
apply simp
apply (auto simp: space-pair-measure) []
apply simp
apply (intro integral-cong-AE)
using ae1 ae2 ae3 ae4
apply (auto simp: log-divide-eq zero-less-mult-iff zero-less-divide-iff field-simps
space-pair-measure less-le)
done
finally show ?nonneg
by simp
qed

```

lemma (*in information-space*)

```

fixes Px :: - => real
assumes S: sigma-finite-measure S and T: sigma-finite-measure T and P:
sigma-finite-measure P
assumes Fx: finite-entropy S X Px
assumes Fz: finite-entropy P Z Pz
assumes Fyz: finite-entropy (T ⊗M P) (λx. (Y x, Z x)) Pyz
assumes Fxz: finite-entropy (S ⊗M P) (λx. (X x, Z x)) Pxz
assumes Fxyz: finite-entropy (S ⊗M T ⊗M P) (λx. (X x, Y x, Z x)) Pxyz
shows conditional-mutual-information-generic-eq': conditional-mutual-information
b S T P X Y Z
= ( $\int (x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))) \partial(S \otimes_M T \otimes_M P)$ ) (is ?eq)
and conditional-mutual-information-generic-nonneg': 0 ≤ conditional-mutual-information
b S T P X Y Z (is ?nonneg)
proof -
note Px = Fx[THEN finite-entropy-distributed, measurable]
note Pz = Fz[THEN finite-entropy-distributed, measurable]
note Pyz = Fyz[THEN finite-entropy-distributed, measurable]
note Pxz = Fxz[THEN finite-entropy-distributed, measurable]
note Pxyz = Fxyz[THEN finite-entropy-distributed, measurable]

note Px-nn = Fx[THEN finite-entropy-nn]
note Pz-nn = Fz[THEN finite-entropy-nn]
note Pyz-nn = Fyz[THEN finite-entropy-nn]
note Pxz-nn = Fxz[THEN finite-entropy-nn]

```

note $P_{xyz-nn} = F_{xyz}[THEN\ finite-entropy-nn]$

note $P_{x'} = F_x[THEN\ finite-entropy-measurable, measurable]$

note $P_{z'} = F_z[THEN\ finite-entropy-measurable, measurable]$

note $P_{yz'} = F_{yz}[THEN\ finite-entropy-measurable, measurable]$

note $P_{xz'} = F_{xz}[THEN\ finite-entropy-measurable, measurable]$

note $P_{xyz'} = F_{xyz}[THEN\ finite-entropy-measurable, measurable]$

interpret S : *sigma-finite-measure* S **by fact**

interpret T : *sigma-finite-measure* T **by fact**

interpret P : *sigma-finite-measure* P **by fact**

interpret TP : *pair-sigma-finite* $T P$..

interpret SP : *pair-sigma-finite* $S P$..

interpret ST : *pair-sigma-finite* $S T$..

interpret SPT : *pair-sigma-finite* $S \otimes_M P T$..

interpret STP : *pair-sigma-finite* $S T \otimes_M P$..

interpret TPS : *pair-sigma-finite* $T \otimes_M P S$..

have TP : *sigma-finite-measure* $(T \otimes_M P)$..

have SP : *sigma-finite-measure* $(S \otimes_M P)$..

from $P_{xz} P_{xyz}$ **have** *distr-eq*: $distr\ M\ (S \otimes_M P)\ (\lambda x. (X\ x, Z\ x)) =$
 $distr\ (distr\ M\ (S \otimes_M T \otimes_M P)\ (\lambda x. (X\ x, Y\ x, Z\ x)))\ (S \otimes_M P)\ (\lambda(x, y,$
 $z). (x, z))$
by (*simp add: distr-distr comp-def*)

have *mutual-information* $b\ S\ P\ X\ Z =$

$(\int x. P_{xz}\ x * \log\ b\ (P_{xz}\ x / (P_x\ (fst\ x) * P_z\ (snd\ x))))\ \partial(S \otimes_M P)$

using $P_x\ P_{x-nn}\ P_z\ P_{z-nn}\ P_{xz}\ P_{xz-nn}$

by (*rule mutual-information-distr[OF S P]*) (*auto simp: space-pair-measure*)

also have ... $= (\int (x,y,z). P_{xyz}\ (x,y,z) * \log\ b\ (P_{xz}\ (x,z) / (P_x\ x * P_z\ z)))\ \partial(S$
 $\otimes_M T \otimes_M P)$

using $b-gt-1\ P_{xz}\ P_{xz-nn}\ P_{xyz}\ P_{xyz-nn}$

by (*subst distributed-transform-integral[OF Pxyz - Pxz, where T= $\lambda(x, y, z).$*
 $(x, z)]$)

(*auto simp: split-beta'*)

finally have *mi-eq*:

mutual-information $b\ S\ P\ X\ Z = (\int (x,y,z). P_{xyz}\ (x,y,z) * \log\ b\ (P_{xz}\ (x,z) /$
 $(P_x\ x * P_z\ z)))\ \partial(S \otimes_M T \otimes_M P)$.

have $ae1$: $AE\ x\ in\ S \otimes_M T \otimes_M P. P_x\ (fst\ x) = 0 \longrightarrow P_{xyz}\ x = 0$

by (*intro subdensity-finite-entropy[of fst, OF - Fxyz Fx]*) *auto*

moreover have $ae2$: $AE\ x\ in\ S \otimes_M T \otimes_M P. P_z\ (snd\ (snd\ x)) = 0 \longrightarrow$
 $P_{xyz}\ x = 0$

by (*intro subdensity-finite-entropy[of $\lambda x. snd\ (snd\ x)$, OF - Fxyz Fz]*) *auto*

moreover have $ae3$: $AE\ x\ in\ S \otimes_M T \otimes_M P. P_{xz}\ (fst\ x, snd\ (snd\ x)) = 0$
 $\longrightarrow P_{xyz}\ x = 0$

by (*intro subdensity-finite-entropy[of $\lambda x. (fst\ x, snd\ (snd\ x))$, OF - Fxyz Fxz]*)
auto

moreover have $ae4$: $AE\ x\ in\ S \otimes_M T \otimes_M P. P_{yz}\ (snd\ x) = 0 \longrightarrow P_{xyz}\ x$

= 0
 by (intro subdensity-finite-entropy[of snd, OF - Fxyz Fyz]) auto
 ultimately have ae: AE x in $S \otimes_M T \otimes_M P$.
 $Pxyz\ x * \log b (Pxyz\ x / (Px\ (fst\ x) * Pyz\ (snd\ x))) -$
 $Pxyz\ x * \log b (Pxz\ (fst\ x, snd\ (snd\ x)) / (Px\ (fst\ x) * Pz\ (snd\ (snd\ x)))) =$
 $Pxyz\ x * \log b (Pxyz\ x * Pz\ (snd\ (snd\ x)) / (Pxz\ (fst\ x, snd\ (snd\ x)) * Pyz$
 (snd x))
 using AE-space
 proof eventually-elim
 case (elim x)
 show ?case
 proof cases
 assume $Pxyz\ x \neq 0$
 with elim have $0 < Px\ (fst\ x)\ 0 < Pz\ (snd\ (snd\ x))\ 0 < Pxz\ (fst\ x, snd$
 (snd x))
 $0 < Pyz\ (snd\ x)\ 0 < Pxyz\ x$
 using Px-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn
 by (auto simp: space-pair-measure less-le)
 then show ?thesis
 using b-gt-1 by (simp add: log-simps less-imp-le field-simps)
 qed simp
 qed
 have integrable ($S \otimes_M T \otimes_M P$)
 ($\lambda x. Pxyz\ x * \log b (Pxyz\ x) - Pxyz\ x * \log b (Px\ (fst\ x)) - Pxyz\ x * \log b$
 ($Pyz\ (snd\ x)$))
 using finite-entropy-integrable[OF Fxyz]
 using finite-entropy-integrable-transform[OF Fx Pxyz Pxyz-nn, of fst]
 using finite-entropy-integrable-transform[OF Fyz Pxyz Pxyz-nn, of snd]
 by simp
 moreover have ($\lambda(x, y, z). Pxyz\ (x, y, z) * \log b (Pxyz\ (x, y, z) / (Px\ x * Pyz$
 (y, z))) \in borel-measurable ($S \otimes_M T \otimes_M P$)
 using Pxyz Px Pyz by simp
 ultimately have I1: integrable ($S \otimes_M T \otimes_M P$) ($\lambda(x, y, z). Pxyz\ (x, y, z)$
 $* \log b (Pxyz\ (x, y, z) / (Px\ x * Pyz\ (y, z)))$)
 apply (rule integrable-cong-AE-imp)
 using ae1 ae4 AE-space
 by eventually-elim
 (insert Px-nn Pyz-nn Pxyz-nn,
 auto simp: log-divide-eq log-mult-eq field-simps zero-less-mult-iff space-pair-measure
 less-le)
 have integrable ($S \otimes_M T \otimes_M P$)
 ($\lambda x. Pxyz\ x * \log b (Pxz\ (fst\ x, snd\ (snd\ x))) - Pxyz\ x * \log b (Px\ (fst\ x)) -$
 $Pxyz\ x * \log b (Pz\ (snd\ (snd\ x)))$)
 using finite-entropy-integrable-transform[OF Fxz Pxyz Pxyz-nn, of $\lambda x. (fst\ x,$
 snd (snd x))]
 using finite-entropy-integrable-transform[OF Fx Pxyz Pxyz-nn, of fst]
 using finite-entropy-integrable-transform[OF Fz Pxyz Pxyz-nn, of snd \circ snd]


```

  by simp
  moreover have  $(\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxz (x, z) / (Px x * Pz z)))$ 
 $\in \text{borel-measurable } (S \otimes_M T \otimes_M P)$ 
  using Pxyz Px Pz
  by auto
  ultimately have I2:  $\text{integrable } (S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z)$ 
 $* \log b (Pxz (x, z) / (Px x * Pz z)))$ 
  apply (rule integrable-cong-AE-imp)
  using ae1 ae2 ae3 ae4 AE-space
  by eventually-elim
  (insert Px-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn,
   auto simp: log-divide-eq log-mult-eq field-simps zero-less-mult-iff less-le
   space-pair-measure)

  from ae I1 I2 show ?eq
  unfolding conditional-mutual-information-def
  apply (subst mi-eq)
  apply (subst mutual-information-distr[OF S TP Px Px-nn Pyz Pyz-nn Pxyz
  Pxyz-nn])
  apply simp
  apply simp
  apply (simp add: space-pair-measure)
  apply (subst Bochner-Integration.integral-diff[symmetric])
  apply (auto intro!: integral-cong-AE simp: split-beta' simp del: Bochner-Integration.integral-diff)
  done

  let ?P = density  $(S \otimes_M T \otimes_M P)$  Pxyz
  interpret P: prob-space ?P
  unfolding distributed-distr-eq-density[OF Pxyz, symmetric] by (rule prob-space-distr)
  simp

  let ?Q = density  $(T \otimes_M P)$  Pyz
  interpret Q: prob-space ?Q
  unfolding distributed-distr-eq-density[OF Pyz, symmetric] by (rule prob-space-distr)
  simp

  let ?f =  $\lambda(x, y, z). Pxz (x, z) * (Pyz (y, z) / Pz z) / Pxyz (x, y, z)$ 

  from subdensity-finite-entropy[of snd, OF - Fyz Fz]
  have aeX1:  $\text{AE } x \text{ in } T \otimes_M P. Pz (snd x) = 0 \longrightarrow Pyz x = 0$  by (auto simp:
  comp-def)
  have aeX2:  $\text{AE } x \text{ in } T \otimes_M P. 0 \leq Pz (snd x)$ 
  using Pz by (intro TP.AE-pair-measure) (auto intro: Pz-nn)

  have aeX3:  $\text{AE } y \text{ in } T \otimes_M P. (\int^+ x. \text{ennreal } (Pxz (x, snd y)) \partial S) = \text{ennreal}$ 
 $(Pz (snd y))$ 
  using Pz distributed-marginal-eq-joint2[OF P S Pz Pxz]
  by (intro TP.AE-pair-measure) (auto)
  have  $(\int^+ x. ?f x \partial ?P) \leq (\int^+ (x, y, z). Pxz (x, z) * (Pyz (y, z) / Pz z) \partial S)$ 

```

$\otimes_M T \otimes_M P$)
using $Px\text{-nn } Pz\text{-nn } Pxz\text{-nn } Pyz\text{-nn } Pxyz\text{-nn}$
by (*subst nn-integral-density*)
(auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
also have $\dots = (\int^+ (y, z). \int^+ x. \text{ennreal } (Pxz (x, z)) * \text{ennreal } (Pyz (y, z) /$
 $Pz z) \partial S \partial T \otimes_M P)$
using $Px\text{-nn } Pz\text{-nn } Pxz\text{-nn } Pyz\text{-nn } Pxyz\text{-nn}$
by (*subst STP.nn-integral-snd[symmetric]*)
(auto simp add: split-beta' ennreal-mult[symmetric] space-pair-measure intro!:
nn-integral-cong)
also have $\dots = (\int^+ x. \text{ennreal } (Pyz x) * 1 \partial T \otimes_M P)$
apply (*rule nn-integral-cong-AE*)
using $aeX1 aeX2 aeX3 AE\text{-space}$
apply *eventually-elim*
proof (*case-tac x, simp add: space-pair-measure*)
fix $a b$ **assume** $Pz b = 0 \longrightarrow Pyz (a, b) = 0 \ 0 \leq Pz b \ a \in \text{space } T \wedge b \in$
 $\text{space } P$
 $(\int^+ x. \text{ennreal } (Pxz (x, b)) \partial S) = \text{ennreal } (Pz b)$
then show $(\int^+ x. \text{ennreal } (Pxz (x, b)) * \text{ennreal } (Pyz (a, b) / Pz b) \partial S) =$
 $\text{ennreal } (Pyz (a, b))$
using $Pyz\text{-nn}[of (a,b)]$
by (*subst nn-integral-multc*) (*auto simp: space-pair-measure ennreal-mult[symmetric]*)
qed
also have $\dots = 1$
using $Q.\text{emeasure-space-1 } Pyz\text{-nn } \text{distributed-distr-eq-density}[OF Pyz]$
by (*subst nn-integral-density[symmetric]*) *auto*
finally have $le1: (\int^+ x. ?f x \partial ?P) \leq 1 .$
also have $\dots < \infty$ **by** *simp*
finally have $fin: (\int^+ x. ?f x \partial ?P) \neq \infty$ **by** *simp*

have $(\int^+ x. ?f x \partial ?P) \neq 0$
using $Pxyz\text{-nn}$
apply (*subst nn-integral-density*)
apply (*simp-all add: split-beta' ennreal-mult'[symmetric] cong: nn-integral-cong*)
proof
let $?g = \lambda x. \text{ennreal } (\text{if } Pxyz \ x = 0 \ \text{then } 0 \ \text{else } Pxz (fst \ x, snd (snd \ x)) * Pyz$
 $(snd \ x) / Pz (snd (snd \ x)))$
assume $(\int^+ x. ?g x \partial (S \otimes_M T \otimes_M P)) = 0$
then have $AE \ x \ \text{in } S \otimes_M T \otimes_M P. ?g \ x = 0$
by (*intro nn-integral-0-iff-AE[THEN iffD1]*) *auto*
then have $AE \ x \ \text{in } S \otimes_M T \otimes_M P. Pxyz \ x = 0$
using $ae1 ae2 ae3 ae4 AE\text{-space}$
by *eventually-elim*
(insert Px-nn Pz-nn Pxz-nn Pyz-nn,
auto split: if-split-asm simp: mult-le-0-iff divide-le-0-iff space-pair-measure)
then have $(\int^+ x. \text{ennreal } (Pxyz \ x) \partial S \otimes_M T \otimes_M P) = 0$
by (*subst nn-integral-cong-AE[of - $\lambda x. 0$]*) *auto*
with $P.\text{emeasure-space-1}$ **show** *False*
by (*subst (asm) emeasure-density*) (*auto cong: nn-integral-cong*)

```

qed
then have pos:  $0 < (\int^+ x. ?f x \partial?P)$ 
  by (simp add: zero-less-iff-neq-zero)

have neg:  $(\int^+ x. - ?f x \partial?P) = 0$ 
  using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  by (intro nn-integral-0-iff-AE[THEN iffD2])
  (auto simp: split-beta' AE-density space-pair-measure intro!: AE-I2 ennreal-neg)

have I3: integrable  $(S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz$ 
 $(x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))))$ 
  apply (rule integrable-cong-AE[THEN iffD1, OF --- Bochner-Integration.integrable-diff[OF
  I1 I2]])
  using ae
  apply (auto simp: split-beta')
  done

have  $-\log b 1 \leq -\log b (\text{integral}^L ?P ?f)$ 
proof (intro le-imp-neg-le log-le[OF b-gt-1])
  have If: integrable  $?P ?f$ 
    unfolding real-integrable-def
  proof (intro conjI)
    from neg show  $(\int^+ x. - ?f x \partial?P) \neq \infty$ 
      by simp
    from fin show  $(\int^+ x. ?f x \partial?P) \neq \infty$ 
      by simp
  qed simp
then have  $(\int^+ x. ?f x \partial?P) = (\int x. ?f x \partial?P)$ 
  using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  by (intro nn-integral-eq-integral)
  (auto simp: AE-density space-pair-measure)
with pos le1
show  $0 < (\int x. ?f x \partial?P) (\int x. ?f x \partial?P) \leq 1$ 
  by (simp-all add: )
qed
also have  $-\log b (\text{integral}^L ?P ?f) \leq (\int x. -\log b (?f x) \partial?P)$ 
proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
  show AE x in ?P.  $?f x \in \{0<..\}$ 
    unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
    using ae1 ae2 ae3 ae4 AE-space
  by eventually-elim (insert Pxyz-nn Pyz-nn Pz-nn Pxz-nn, auto simp: space-pair-measure
  less-le)
  show integrable  $?P ?f$ 
    unfolding real-integrable-def
    using fin neg by (auto simp: split-beta')
  show integrable  $?P (\lambda x. -\log b (?f x))$ 
    using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
    apply (subst integrable-real-density)
    apply simp

```

```

apply simp
apply simp
apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
apply simp
apply simp
using ae1 ae2 ae3 ae4 AE-space
apply eventually-elim
apply (auto simp: log-divide-eq log-mult-eq zero-le-mult-iff zero-less-mult-iff
        zero-less-divide-iff field-simps space-pair-measure less-le)
done
qed (auto simp: b-gt-1 minus-log-convex)
also have ... = conditional-mutual-information b S T P X Y Z
unfolding (?eq)
using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
apply (subst integral-real-density)
apply simp
apply simp
apply simp
apply (intro integral-cong-AE)
using ae1 ae2 ae3 ae4 AE-space
apply (auto simp: log-divide-eq zero-less-mult-iff zero-less-divide-iff
        field-simps space-pair-measure less-le)
done
finally show ?nonneg
by simp
qed

lemma (in information-space) conditional-mutual-information-eq:
assumes Pz: simple-distributed M Z Pz
assumes Pyz: simple-distributed M (λx. (Y x, Z x)) Pyz
assumes Pxz: simple-distributed M (λx. (X x, Z x)) Pxz
assumes Pxyz: simple-distributed M (λx. (X x, Y x, Z x)) Pxyz
shows  $I(X ; Y \mid Z) =$ 
 $(\sum_{(x, y, z) \in (\lambda x. (X x, Y x, Z x))' \text{space } M. Pxyz(x, y, z) * \log b (Pxyz(x, y,$ 
 $z) / (Pxz(x, z) * (Pyz(y, z) / Pz z)))))$ 
proof (subst conditional-mutual-information-generic-eq[OF - - - -
        simple-distributed[OF Pz] - simple-distributed-joint[OF Pyz] - simple-distributed-joint[OF
Pxz] -
        simple-distributed-joint2[OF Pxyz]])
note simple-distributed-joint2-finite[OF Pxyz, simp]
show sigma-finite-measure (count-space (X ' space M))
by (simp add: sigma-finite-measure-count-space-finite)
show sigma-finite-measure (count-space (Y ' space M))
by (simp add: sigma-finite-measure-count-space-finite)
show sigma-finite-measure (count-space (Z ' space M))
by (simp add: sigma-finite-measure-count-space-finite)
have count-space (X ' space M)  $\otimes_M$  count-space (Y ' space M)  $\otimes_M$  count-space
(Z ' space M) =
count-space (X'space M  $\times$  Y'space M  $\times$  Z'space M)

```

```

(is ?P = ?C)
by (simp add: pair-measure-count-space)

let ?Px =  $\lambda x.$  measure  $M$  ( $X - \{x\} \cap$  space  $M$ )
have ( $\lambda x.$  ( $X$   $x$ ,  $Z$   $x$ ))  $\in$  measurable  $M$  (count-space ( $X$  ‘ space  $M$ )  $\otimes_M$ 
count-space ( $Z$  ‘ space  $M$ ))
using simple-distributed-joint[OF Pxz] by (rule distributed-measurable)
from measurable-comp[OF this measurable-fst]
have random-variable (count-space ( $X$  ‘ space  $M$ ))  $X$ 
by (simp add: comp-def)
then have simple-function  $M$   $X$ 
unfolding simple-function-def by (auto simp: measurable-count-space-eq2)
then have simple-distributed  $M$   $X$  ?Px
by (rule simple-distributedI) (auto simp: measure-nonneg)
then show distributed  $M$  (count-space ( $X$  ‘ space  $M$ ))  $X$  ?Px
by (rule simple-distributed)

let ?f = ( $\lambda x.$  if  $x \in$  ( $\lambda x.$  ( $X$   $x$ ,  $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then  $Pxyz$   $x$  else 0)
let ?g = ( $\lambda x.$  if  $x \in$  ( $\lambda x.$  ( $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then  $Pyz$   $x$  else 0)
let ?h = ( $\lambda x.$  if  $x \in$  ( $\lambda x.$  ( $X$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then  $Pxz$   $x$  else 0)
show
integrable ?P ( $\lambda(x, y, z).$  ?f ( $x, y, z$ ) * log  $b$  (?f ( $x, y, z$ ) / (?Px  $x$  * ?g ( $y,$ 
 $z$ ))))
integrable ?P ( $\lambda(x, y, z).$  ?f ( $x, y, z$ ) * log  $b$  (?h ( $x, z$ ) / (?Px  $x$  * Pz  $z$ )))
by (auto intro!: integrable-count-space simp: pair-measure-count-space)
let ?i =  $\lambda x$   $y$   $z.$  ?f ( $x, y, z$ ) * log  $b$  (?f ( $x, y, z$ ) / (?h ( $x, z$ ) * (?g ( $y, z$ ) / Pz
 $z$ )))
let ?j =  $\lambda x$   $y$   $z.$   $Pxyz$  ( $x, y, z$ ) * log  $b$  ( $Pxyz$  ( $x, y, z$ ) / ( $Pxz$  ( $x, z$ ) * ( $Pyz$  ( $y, z$ )
/ Pz  $z$ )))
have ( $\lambda(x, y, z).$  ?i  $x$   $y$   $z$ ) = ( $\lambda x.$  if  $x \in$  ( $\lambda x.$  ( $X$   $x$ ,  $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then
?j (fst  $x$ ) (fst (snd  $x$ )) (snd (snd  $x$ )) else 0)
by (auto intro!: ext)
then show ( $\int$  ( $x, y, z$ ). ?i  $x$   $y$   $z$   $\partial$ ?P) = ( $\sum$  ( $x, y, z$ ) $\in$ ( $\lambda x.$  ( $X$   $x$ ,  $Y$   $x$ ,  $Z$   $x$ )) ‘
space  $M.$  ?j  $x$   $y$   $z$ )
by (auto intro!: sum.cong simp add: (?P = ?C) lebesgue-integral-count-space-finite
simple-distributed-finite sum.If-cases split-beta)
qed (insert Pz Pyz Pxz Pxyz, auto intro: measure-nonneg)

lemma (in information-space) conditional-mutual-information-nonneg:
assumes  $X$ : simple-function  $M$   $X$  and  $Y$ : simple-function  $M$   $Y$  and  $Z$ : simple-function
 $M$   $Z$ 
shows  $0 \leq \mathcal{I}(X ; Y | Z)$ 
proof -
have [simp]: count-space ( $X$  ‘ space  $M$ )  $\otimes_M$  count-space ( $Y$  ‘ space  $M$ )  $\otimes_M$ 
count-space ( $Z$  ‘ space  $M$ ) =
count-space ( $X$ ‘space  $M$   $\times$   $Y$ ‘space  $M$   $\times$   $Z$ ‘space  $M$ )
by (simp add: pair-measure-count-space  $X$   $Y$   $Z$  simple-functionD)
note sf = sigma-finite-measure-count-space-finite[OF simple-functionD(1)]
note sd = simple-distributedI[OF - - refl]

```

```

note sp = simple-function-Pair
show ?thesis
apply (rule conditional-mutual-information-generic-nonneg[OF sf[OF X] sf[OF
Y] sf[OF Z]])
apply (rule simple-distributed[OF sd[OF X]])
apply simp
apply simp
apply (rule simple-distributed[OF sd[OF Z]])
apply simp
apply simp
apply (rule simple-distributed-joint[OF sd[OF sp[OF Y Z]])]
apply simp
apply simp
apply (rule simple-distributed-joint[OF sd[OF sp[OF X Z]])]
apply simp
apply simp
apply (rule simple-distributed-joint2[OF sd[OF sp[OF X sp[OF Y Z]])]
apply simp
apply simp
apply (auto intro!: integrable-count-space simp: X Y Z simple-functionD)
done
qed

```

14.7 Conditional Entropy

definition (in *prob-space*)

conditional-entropy $b S T X Y = - (\int (x, y). \log b (\text{enn2real } (\text{RN-deriv } (S \otimes_M T) (\text{distr } M (S \otimes_M T) (\lambda x. (X x, Y x)))) (x, y)) / \text{enn2real } (\text{RN-deriv } T (\text{distr } M T Y) y)) \partial \text{distr } M (S \otimes_M T) (\lambda x. (X x, Y x)))$

abbreviation (in *information-space*)

conditional-entropy-Pow $(\mathcal{H}'(- | -))$ **where**
 $\mathcal{H}(X | Y) \equiv \text{conditional-entropy } b (\text{count-space } (X' \text{space } M)) (\text{count-space } (Y' \text{space } M)) X Y$

lemma (in *information-space*) *conditional-entropy-generic-eq:*

fixes $Pxy :: - \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes $Py[\text{measurable}]$: *distributed* $M T Y Py$ **and** $Py\text{-nn}[\text{simp}]$: $\bigwedge x. x \in \text{space } T \Rightarrow 0 \leq Py x$
assumes $Pxy[\text{measurable}]$: *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$
and $Pxy\text{-nn}[\text{simp}]$: $\bigwedge x y. x \in \text{space } S \Rightarrow y \in \text{space } T \Rightarrow 0 \leq Pxy (x, y)$
shows *conditional-entropy* $b S T X Y = - (\int (x, y). Pxy (x, y) * \log b (Pxy (x, y) / Py y) \partial (S \otimes_M T))$
proof –
interpret S : *sigma-finite-measure* S **by fact**
interpret T : *sigma-finite-measure* T **by fact**
interpret ST : *pair-sigma-finite* $S T$..

```

have [measurable]:  $P_y \in T \rightarrow_M \text{borel}$ 
  using  $P_y$   $P_y$ -nn by (intro distributed-real-measurable)
have measurable- $P_{xy}$ [measurable]:  $P_{xy} \in (S \otimes_M T) \rightarrow_M \text{borel}$ 
  using  $P_{xy}$   $P_{xy}$ -nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

have AE  $x$  in density  $(S \otimes_M T)$   $(\lambda x. \text{ennreal } (P_{xy} x))$ .  $P_{xy} x = \text{enn2real}$ 
(RN-deriv  $(S \otimes_M T)$  (distr  $M$   $(S \otimes_M T)$   $(\lambda x. (X x, Y x))$ )  $x$ )
  unfolding AE-density[OF distributed-borel-measurable, OF  $P_{xy}$ ]
  unfolding distributed-distr-eq-density[OF  $P_{xy}$ ]
  using distributed-RN-deriv[OF  $P_{xy}$ ]
  by auto
moreover
have AE  $x$  in density  $(S \otimes_M T)$   $(\lambda x. \text{ennreal } (P_y x))$ .  $P_y (\text{snd } x) = \text{enn2real}$ 
(RN-deriv  $T$  (distr  $M$   $T$   $Y$ ) ( $\text{snd } x$ ))
  unfolding AE-density[OF distributed-borel-measurable, OF  $P_y$ ]
  unfolding distributed-distr-eq-density[OF  $P_y$ ]
  apply (rule  $ST$ .AE-pair-measure)
  apply auto
  using distributed-RN-deriv[OF  $P_y$ ]
  apply auto
  done
ultimately
have conditional-entropy  $b$   $S$   $T$   $X$   $Y = - \int x. P_{xy} x * \log b (P_{xy} x / P_y (\text{snd } x)) \partial(S \otimes_M T)$ 
  unfolding conditional-entropy-def neg-equal-iff-equal
  apply (subst integral-real-density[symmetric])
  apply (auto simp: distributed-distr-eq-density[OF  $P_{xy}$ ] space-pair-measure
    intro!: integral-cong-AE)
  done
then show ?thesis by (simp add: split-beta')
qed

```

lemma (in information-space) conditional-entropy-eq-entropy:

```

fixes  $P_x :: 'b \Rightarrow \text{real}$  and  $P_y :: 'c \Rightarrow \text{real}$ 
assumes  $S$ : sigma-finite-measure  $S$  and  $T$ : sigma-finite-measure  $T$ 
assumes  $P_y$ [measurable]: distributed  $M$   $T$   $Y$   $P_y$ 
  and  $P_y$ -nn[simp]:  $\bigwedge x. x \in \text{space } T \implies 0 \leq P_y x$ 
assumes  $P_{xy}$ [measurable]: distributed  $M$   $(S \otimes_M T)$   $(\lambda x. (X x, Y x))$   $P_{xy}$ 
  and  $P_{xy}$ -nn[simp]:  $\bigwedge x y. x \in \text{space } S \implies y \in \text{space } T \implies 0 \leq P_{xy} (x, y)$ 
assumes  $I1$ : integrable  $(S \otimes_M T)$   $(\lambda x. P_{xy} x * \log b (P_{xy} x))$ 
assumes  $I2$ : integrable  $(S \otimes_M T)$   $(\lambda x. P_{xy} x * \log b (P_y (\text{snd } x)))$ 
shows conditional-entropy  $b$   $S$   $T$   $X$   $Y = \text{entropy } b (S \otimes_M T)$   $(\lambda x. (X x, Y x))$ 
– entropy  $b$   $T$   $Y$ 
proof –
  interpret  $S$ : sigma-finite-measure  $S$  by fact
  interpret  $T$ : sigma-finite-measure  $T$  by fact
  interpret  $ST$ : pair-sigma-finite  $S$   $T$  ..

```

have $[measurable]: Py \in T \rightarrow_M \text{borel}$
using $Py Py\text{-nn}$ **by** $(\text{intro distributed-real-measurable})$
have $measurable\text{-}Pxy[measurable]: Pxy \in (S \otimes_M T) \rightarrow_M \text{borel}$
using $Pxy Py\text{-nn}$ **by** $(\text{intro distributed-real-measurable})$ $(\text{auto simp: space-pair-measure})$

have $entropy\ b\ T\ Y = - (\int y. Py\ y * \log\ b\ (Py\ y)\ \partial T)$
by $(\text{rule entropy-distr}[OF\ Py\ Py\text{-nn}])$
also have $\dots = - (\int (x,y). Pxy\ (x,y) * \log\ b\ (Py\ y)\ \partial(S \otimes_M T))$
using $b\text{-gt-1}$
by $(\text{subst distributed-transform-integral}[OF\ Pxy - Py, \textbf{where } T=\text{snd}])$
 $(\text{auto intro!: Bochner-Integration.integral-cong simp: space-pair-measure})$
finally have $e\text{-eq}: entropy\ b\ T\ Y = - (\int (x,y). Pxy\ (x,y) * \log\ b\ (Py\ y)\ \partial(S \otimes_M T))$.

have $**:$ $\bigwedge x. x \in \text{space } (S \otimes_M T) \implies 0 \leq Pxy\ x$
by $(\text{auto simp: space-pair-measure})$

have $ae2:$ $AE\ x\ \text{in } S \otimes_M T. Py\ (\text{snd } x) = 0 \longrightarrow Pxy\ x = 0$
by $(\text{intro subdensity-real}[of\ \text{snd}, OF - Pxy\ Py])$
 $(\text{auto intro: measurable-Pair simp: space-pair-measure})$
moreover have $ae4:$ $AE\ x\ \text{in } S \otimes_M T. 0 \leq Py\ (\text{snd } x)$
using Py **by** $(\text{intro ST.AE-pair-measure})$ $(\text{auto simp: comp-def intro!: measurable-snd''})$
ultimately have $AE\ x\ \text{in } S \otimes_M T. 0 \leq Pxy\ x \wedge 0 \leq Py\ (\text{snd } x) \wedge$
 $(Pxy\ x = 0 \vee (Pxy\ x \neq 0 \longrightarrow 0 < Pxy\ x \wedge 0 < Py\ (\text{snd } x)))$
using $AE\text{-space}$ **by** eventually-elim $(\text{auto simp: space-pair-measure less-le})$
then have $ae:$ $AE\ x\ \text{in } S \otimes_M T.$
 $Pxy\ x * \log\ b\ (Pxy\ x) - Pxy\ x * \log\ b\ (Py\ (\text{snd } x)) = Pxy\ x * \log\ b\ (Pxy\ x /$
 $Py\ (\text{snd } x))$
by eventually-elim $(\text{auto simp: log-simps field-simps b-gt-1})$
have $\text{conditional-entropy } b\ S\ T\ X\ Y =$
 $-\ (\int x. Pxy\ x * \log\ b\ (Pxy\ x) - Pxy\ x * \log\ b\ (Py\ (\text{snd } x))\ \partial(S \otimes_M T))$
unfolding $\text{conditional-entropy-generic-eq}[OF\ S\ T\ Py\ Py\text{-nn}\ Pxy\ Py\text{-nn}, \text{ simplified}]$
 $\text{neg-equal-iff-equal}$
apply $(\text{intro integral-cong-AE})$
using ae
apply auto
done
also have $\dots = - (\int x. Pxy\ x * \log\ b\ (Pxy\ x)\ \partial(S \otimes_M T)) - - (\int x. Pxy\ x$
 $* \log\ b\ (Py\ (\text{snd } x))\ \partial(S \otimes_M T))$
by $(\text{simp add: Bochner-Integration.integral-diff}[OF\ I1\ I2])$
finally show $?thesis$
using $\text{conditional-entropy-generic-eq}[OF\ S\ T\ Py\ Py\text{-nn}\ Pxy\ Py\text{-nn}, \text{ simplified}]$
 $\text{entropy-distr}[OF\ Pxy\ **, \text{ simplified}]$ $e\text{-eq}$
by $(\text{simp add: split-beta'})$
qed

lemma $(\text{in information-space})$ $\text{conditional-entropy-eq-entropy-simple}:$

assumes $X:$ $\text{simple-function } M\ X$ **and** $Y:$ $\text{simple-function } M\ Y$

shows $\mathcal{H}(X \mid Y) = \text{entropy } b\ (\text{count-space } (X\ \text{space } M) \otimes_M \text{count-space})$

$(Y \text{'space } M)) (\lambda x. (X \ x, Y \ x)) - \mathcal{H}(Y)$
proof –
have $\text{count-space } (X \ \text{'space } M) \otimes_M \text{count-space } (Y \ \text{'space } M) = \text{count-space } (X \ \text{'space } M \times Y \ \text{'space } M)$
(is $?P = ?C$) **using** $X \ Y$ **by** (*simp add: simple-functionD pair-measure-count-space*)
show *?thesis*
by (*rule conditional-entropy-eq-entropy sigma-finite-measure-count-space-finite simple-functionD X Y simple-distributed simple-distributedI[OF - - refl] simple-distributed-joint simple-function-Pair integrable-count-space measure-nonneg*)
(auto simp: (<?P = ?C> measure-nonneg intro!: integrable-count-space simple-functionD X Y)
qed

lemma (*in information-space*) *conditional-entropy-eq*:
assumes Y : *simple-distributed* $M \ Y \ P_y$
assumes XY : *simple-distributed* $M \ (\lambda x. (X \ x, Y \ x)) \ P_{xy}$
shows $\mathcal{H}(X \mid Y) = - (\sum (x, y) \in (\lambda x. (X \ x, Y \ x)) \ \text{'space } M. P_{xy} (x, y) * \log b (P_{xy} (x, y) / P_y y))$
proof (*subst conditional-entropy-generic-eq[OF - - simple-distributed[OF Y] - simple-distributed-joint[OF XY]]*)
have *finite* $((\lambda x. (X \ x, Y \ x)) \ \text{'space } M)$
using XY **unfolding** *simple-distributed-def* **by** *auto*
from *finite-imageI[OF this, of fst]*
have [*simp*]: *finite* $(X \ \text{'space } M)$
by (*simp add: image-comp comp-def*)
note Y [*THEN simple-distributed-finite, simp*]
show *sigma-finite-measure* $(\text{count-space } (X \ \text{'space } M))$
by (*simp add: sigma-finite-measure-count-space-finite*)
show *sigma-finite-measure* $(\text{count-space } (Y \ \text{'space } M))$
by (*simp add: sigma-finite-measure-count-space-finite*)
let $?f = (\lambda x. \text{if } x \in (\lambda x. (X \ x, Y \ x)) \ \text{'space } M \text{ then } P_{xy} \ x \ \text{else } 0)$
have $\text{count-space } (X \ \text{'space } M) \otimes_M \text{count-space } (Y \ \text{'space } M) = \text{count-space } (X \ \text{'space } M \times Y \ \text{'space } M)$
(is $?P = ?C$)
using Y **by** (*simp add: simple-distributed-finite pair-measure-count-space*)
have *eq*: $(\lambda(x, y). ?f (x, y) * \log b (?f (x, y) / P_y y)) = (\lambda x. \text{if } x \in (\lambda x. (X \ x, Y \ x)) \ \text{'space } M \text{ then } P_{xy} \ x * \log b (P_{xy} \ x / P_y (snd \ x)) \ \text{else } 0)$
by *auto*
from Y **show** $-(\int (x, y). ?f (x, y) * \log b (?f (x, y) / P_y y) \ \partial ?P) = -(\sum (x, y) \in (\lambda x. (X \ x, Y \ x)) \ \text{'space } M. P_{xy} (x, y) * \log b (P_{xy} (x, y) / P_y y))$
by (*auto intro!: sum.cong simp add: (<?P = ?C> lebesgue-integral-count-space-finite simple-distributed-finite eq sum.If-cases split-beta')*)
qed (*insert Y XY, auto*)

lemma (*in information-space*) *conditional-mutual-information-eq-conditional-entropy*:
assumes X : *simple-function* $M \ X$ **and** Y : *simple-function* $M \ Y$

```

shows  $\mathcal{I}(X ; X \mid Y) = \mathcal{H}(X \mid Y)$ 
proof –
  define  $P_y$  where  $P_y x = (if\ x \in Y'space\ M\ then\ measure\ M\ (Y - \{x\} \cap space\ M)\ else\ 0)$  for  $x$ 
  define  $P_{xy}$  where  $P_{xy} x =$ 
     $(if\ x \in (\lambda x. (X\ x, Y\ x))'space\ M\ then\ measure\ M\ ((\lambda x. (X\ x, Y\ x)) - \{x\} \cap space\ M)\ else\ 0)$ 
  for  $x$ 
  define  $P_{xxy}$  where  $P_{xxy} x =$ 
     $(if\ x \in (\lambda x. (X\ x, X\ x, Y\ x))'space\ M\ then\ measure\ M\ ((\lambda x. (X\ x, X\ x, Y\ x)) - \{x\} \cap space\ M)\ else\ 0)$ 
  for  $x$ 
  let  $?M = X'space\ M \times X'space\ M \times Y'space\ M$ 

  note  $XY = simple-function-Pair[OF\ X\ Y]$ 
  note  $XXY = simple-function-Pair[OF\ X\ XY]$ 
  have  $P_y$ : simple-distributed  $M\ Y\ P_y$ 
    using  $Y$  by (rule simple-distributedI) (auto simp: P_y-def measure-nonneg)
  have  $P_{xy}$ : simple-distributed  $M\ (\lambda x. (X\ x, Y\ x))\ P_{xy}$ 
    using  $XY$  by (rule simple-distributedI) (auto simp: P_xy-def measure-nonneg)
  have  $P_{xxy}$ : simple-distributed  $M\ (\lambda x. (X\ x, X\ x, Y\ x))\ P_{xxy}$ 
    using  $XXY$  by (rule simple-distributedI) (auto simp: P_xxy-def measure-nonneg)
  have  $eq$ :  $(\lambda x. (X\ x, X\ x, Y\ x))'space\ M = (\lambda(x, y). (x, x, y))'(\lambda x. (X\ x, Y\ x))'space\ M$ 
    by auto
  have  $inj$ :  $\bigwedge A. inj-on\ (\lambda(x, y). (x, x, y))\ A$ 
    by (auto simp: inj-on-def)
  have  $P_{xxy-eq}$ :  $\bigwedge x\ y. P_{xxy}\ (x, x, y) = P_{xy}\ (x, y)$ 
    by (auto simp: P_xxy-def P_xy-def intro!: arg-cong[where f=prob])
  have  $AE\ x$  in count-space  $((\lambda x. (X\ x, Y\ x))'space\ M). P_y\ (snd\ x) = 0 \longrightarrow P_{xy}\ x = 0$ 
    using  $P_y\ P_{xy}$ 
    by (intro subdensity-real[of snd, OF - P_xy[THEN simple-distributed] P_y[THEN simple-distributed]])
    (auto intro: measurable-Pair simp: AE-count-space)
  then show  $?thesis$ 
    apply (subst conditional-mutual-information-eq[OF P_y P_xy P_xy P_xxy])
    apply (subst conditional-entropy-eq[OF P_y P_xy])
    apply (auto intro!: sum.cong simp: P_xxy-eq sum-negf[symmetric] eq sum.reindex[OF inj])
    (log-simps zero-less-mult-iff zero-le-mult-iff field-simps mult-less-0-iff AE-count-space)
    using  $P_y[THEN\ simple-distributed]\ P_{xy}[THEN\ simple-distributed]$ 
    apply (auto simp add: not-le AE-count-space less-le antisym simple-distributed-nonneg[OF P_y] simple-distributed-nonneg[OF P_xy])
  done
qed

```

lemma (in *information-space*) *conditional-entropy-nonneg*:

assumes X : *simple-function* $M X$ **and** Y : *simple-function* $M Y$ **shows** $0 \leq$

$\mathcal{H}(X \mid Y)$

using *conditional-mutual-information-eq-conditional-entropy*[$OF X Y$] *conditional-mutual-information-nonneg*
 $X X Y$]

by *simp*

14.8 Equalities

lemma (in *information-space*) *mutual-information-eq-entropy-conditional-entropy-distr*:

fixes $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \text{real}$

assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T

assumes Px [*measurable*]: *distributed* $M S X Px$

and Px -*nn*[*simp*]: $\bigwedge x. x \in \text{space } S \implies 0 \leq Px x$

and Pxy [*measurable*]: *distributed* $M T Y Py$

and Pxy -*nn*[*simp*]: $\bigwedge x. x \in \text{space } T \implies 0 \leq Py x$

and Pxy [*measurable*]: *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$

and Pxy -*nn*[*simp*]: $\bigwedge x y. x \in \text{space } S \implies y \in \text{space } T \implies 0 \leq Pxy (x, y)$

assumes Ix : *integrable*($S \otimes_M T$) ($\lambda x. Pxy x * \log b (Px (fst x))$)

assumes Iy : *integrable*($S \otimes_M T$) ($\lambda x. Pxy x * \log b (Py (snd x))$)

assumes Ixy : *integrable*($S \otimes_M T$) ($\lambda x. Pxy x * \log b (Pxy x)$)

shows *mutual-information* $b S T X Y = \text{entropy } b S X + \text{entropy } b T Y -$
entropy $b (S \otimes_M T) (\lambda x. (X x, Y x))$

proof –

have [*measurable*]: $Px \in S \rightarrow_M \text{borel}$

using Px Px -*nn* **by** (*intro distributed-real-measurable*)

have [*measurable*]: $Py \in T \rightarrow_M \text{borel}$

using Py Py -*nn* **by** (*intro distributed-real-measurable*)

have *measurable-Pxy*[*measurable*]: $Pxy \in (S \otimes_M T) \rightarrow_M \text{borel}$

using Pxy Pxy -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

have X : *entropy* $b S X = - (\int x. Pxy x * \log b (Px (fst x)) \partial(S \otimes_M T))$

using *b-gt-1*

apply (*subst entropy-distr*[$OF Px Px$ -*nn*], *simp*)

apply (*subst distributed-transform-integral*[$OF Pxy - Px$, **where** $T=fst$])

apply (*auto intro!*: *integral-cong simp: space-pair-measure*)

done

have Y : *entropy* $b T Y = - (\int x. Pxy x * \log b (Py (snd x)) \partial(S \otimes_M T))$

using *b-gt-1*

apply (*subst entropy-distr*[$OF Py Py$ -*nn*], *simp*)

apply (*subst distributed-transform-integral*[$OF Pxy - Py$, **where** $T=snd$])

apply (*auto intro!*: *integral-cong simp: space-pair-measure*)

done

interpret S : *sigma-finite-measure* S **by fact**

interpret T : *sigma-finite-measure* T **by fact**

interpret ST : *pair-sigma-finite* $S T$..

have ST : *sigma-finite-measure* ($S \otimes_M T$) ..

have XY : *entropy* $b (S \otimes_M T) (\lambda x. (X x, Y x)) = - (\int x. Pxy x * \log b (Pxy x)) \partial(S \otimes_M T)$
by (*subst entropy-distr*[*OF Pxy*]) (*auto intro!*: *integral-cong simp: space-pair-measure*)

have $AE x$ in $S \otimes_M T$. $Px (fst x) = 0 \longrightarrow Pxy x = 0$
by (*intro subdensity-real*[*of fst, OF - Pxy Px*]) (*auto intro: measurable-Pair simp: space-pair-measure*)

moreover have $AE x$ in $S \otimes_M T$. $P_y (snd x) = 0 \longrightarrow Pxy x = 0$
by (*intro subdensity-real*[*of snd, OF - Pxy Py*]) (*auto intro: measurable-Pair simp: space-pair-measure*)

moreover have $AE x$ in $S \otimes_M T$. $0 \leq Px (fst x)$
using Px **by** (*intro ST.AE-pair-measure*) (*auto simp: comp-def intro!: measurable-fst''*)

moreover have $AE x$ in $S \otimes_M T$. $0 \leq P_y (snd x)$
using P_y **by** (*intro ST.AE-pair-measure*) (*auto simp: comp-def intro!: measurable-snd''*)

ultimately have $AE x$ in $S \otimes_M T$. $Pxy x * \log b (Pxy x) - Pxy x * \log b (Px (fst x)) - Pxy x * \log b (P_y (snd x)) = Pxy x * \log b (Pxy x / (Px (fst x) * P_y (snd x)))$
(is $AE x$ in $-. ?f x = ?g x$)
using *AE-space*

proof *eventually-elim*
case (*elim x*)
show *?case*
proof *cases*
assume $Pxy x \neq 0$
with *elim* **have** $0 < Px (fst x) 0 < P_y (snd x) 0 < Pxy x$
by (*auto simp: space-pair-measure less-le*)
then show *?thesis*
using *b-gt-1* **by** (*simp add: log-simps less-imp-le field-simps*)

qed *simp*
qed

have *entropy* $b S X + \text{entropy } b T Y - \text{entropy } b (S \otimes_M T) (\lambda x. (X x, Y x)) = \text{integral}^L (S \otimes_M T) ?f$
unfolding $X Y XY$
apply (*subst Bochner-Integration.integral-diff*)
apply (*intro Bochner-Integration.integrable-diff Ixy Ix Iy*)
apply (*subst Bochner-Integration.integral-diff*)
apply (*intro Ixy Ix Iy*)
apply (*simp add: field-simps*)
done

also have $\dots = \text{integral}^L (S \otimes_M T) ?g$
using $\langle AE x \text{ in } -. ?f x = ?g x \rangle$ **by** (*intro integral-cong-AE*) *auto*

also have $\dots = \text{mutual-information } b S T X Y$
by (*rule mutual-information-distr*[*OF S T Px - Py - Pxy - , symmetric*])
(auto simp: space-pair-measure)

finally show *?thesis ..*
qed

lemma (in *information-space*) *mutual-information-eq-entropy-conditional-entropy'*:
fixes $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \text{real}$
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes Px : *distributed* M S X $Px \wedge x. x \in \text{space } S \implies 0 \leq Px\ x$
and Py : *distributed* M T Y $Px \wedge x. x \in \text{space } T \implies 0 \leq Py\ x$
assumes Pxy : *distributed* M $(S \otimes_M T)$ $(\lambda x. (X\ x, Y\ x))$ Pxy
 $\wedge x. x \in \text{space } (S \otimes_M T) \implies 0 \leq Pxy\ x$
assumes Ix : *integrable* $(S \otimes_M T)$ $(\lambda x. Pxy\ x * \log b (Px\ (\text{fst } x)))$
assumes Iy : *integrable* $(S \otimes_M T)$ $(\lambda x. Pxy\ x * \log b (Py\ (\text{snd } x)))$
assumes Ixy : *integrable* $(S \otimes_M T)$ $(\lambda x. Pxy\ x * \log b (Pxy\ x))$
shows *mutual-information* b S T X Y = *entropy* b S X - *conditional-entropy*
 b S T X Y
using
mutual-information-eq-entropy-conditional-entropy-distr[OF S T Px Px Pxy Ix
 Iy Ixy]
conditional-entropy-eq-entropy[OF S T Px Pxy Ixy Iy]
by (*simp add: space-pair-measure*)

lemma (in *information-space*) *mutual-information-eq-entropy-conditional-entropy*:
assumes $sf-X$: *simple-function* M X **and** $sf-Y$: *simple-function* M Y
shows $\mathcal{I}(X ; Y) = \mathcal{H}(X) - \mathcal{H}(X | Y)$
proof -
have X : *simple-distributed* M X $(\lambda x. \text{measure } M (X - \{x\} \cap \text{space } M))$
using $sf-X$ **by** (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
have Y : *simple-distributed* M Y $(\lambda x. \text{measure } M (Y - \{x\} \cap \text{space } M))$
using $sf-Y$ **by** (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
have $sf-XY$: *simple-function* M $(\lambda x. (X\ x, Y\ x))$
using $sf-X$ $sf-Y$ **by** (*rule simple-function-Pair*)
then have XY : *simple-distributed* M $(\lambda x. (X\ x, Y\ x))$ $(\lambda x. \text{measure } M ((\lambda x. (X\ x, Y\ x)) - \{x\} \cap \text{space } M))$
by (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
from *simple-distributed-joint-finite*[OF $this$, *simp*]
have eq : *count-space* $(X - \text{space } M) \otimes_M \text{count-space } (Y - \text{space } M) = \text{count-space}$
 $(X - \text{space } M \times Y - \text{space } M)$
by (*simp add: pair-measure-count-space*)

have $\mathcal{I}(X ; Y) = \mathcal{H}(X) + \mathcal{H}(Y) - \text{entropy } b$ (*count-space* $(X - \text{space } M) \otimes_M$
count-space $(Y - \text{space } M)$) $(\lambda x. (X\ x, Y\ x))$
using *sigma-finite-measure-count-space-finite*
sigma-finite-measure-count-space-finite
simple-distributed[OF X] *measure-nonneg simple-distributed*[OF Y] *measure-nonneg*
simple-distributed-joint[OF XY]
by (*rule mutual-information-eq-entropy-conditional-entropy-distr*)
(*auto simp: eq integrable-count-space measure-nonneg*)
then show *?thesis*
unfolding *conditional-entropy-eq-entropy-simple*[OF $sf-X$ $sf-Y$] **by** *simp*
qed

lemma (in *information-space*) *mutual-information-nonneg-simple*:

assumes $sf\text{-}X$: simple-function $M X$ **and** $sf\text{-}Y$: simple-function $M Y$
shows $0 \leq \mathcal{I}(X ; Y)$
proof –
have X : simple-distributed $M X$ ($\lambda x. \text{measure } M (X - \{x\} \cap \text{space } M)$)
using $sf\text{-}X$ **by** (rule simple-distributedI) (auto simp: measure-nonneg)
have Y : simple-distributed $M Y$ ($\lambda x. \text{measure } M (Y - \{x\} \cap \text{space } M)$)
using $sf\text{-}Y$ **by** (rule simple-distributedI) (auto simp: measure-nonneg)

have $sf\text{-}XY$: simple-function M ($\lambda x. (X x, Y x)$)
using $sf\text{-}X$ $sf\text{-}Y$ **by** (rule simple-function-Pair)
then have XY : simple-distributed M ($\lambda x. (X x, Y x)$) ($\lambda x. \text{measure } M ((\lambda x. (X x, Y x)) - \{x\} \cap \text{space } M)$)
by (rule simple-distributedI) (auto simp: measure-nonneg)

from simple-distributed-joint-finite[OF this, simp]
have eq : count-space $(X \text{ 'space } M) \otimes_M \text{count-space } (Y \text{ 'space } M) = \text{count-space } (X \text{ 'space } M \times Y \text{ 'space } M)$
by (simp add: pair-measure-count-space)

show ?thesis
by (rule mutual-information-nonneg[OF - - simple-distributed[OF X] - simple-distributed[OF Y] - simple-distributed-joint[OF XY]])
(simp-all add: eq integrable-count-space sigma-finite-measure-count-space-finite measure-nonneg)
qed

lemma (in information-space) conditional-entropy-less-eq-entropy:
assumes X : simple-function $M X$ **and** Z : simple-function $M Z$
shows $\mathcal{H}(X | Z) \leq \mathcal{H}(X)$
proof –
have $0 \leq \mathcal{I}(X ; Z)$ **using** $X Z$ **by** (rule mutual-information-nonneg-simple)
also have $\mathcal{I}(X ; Z) = \mathcal{H}(X) - \mathcal{H}(X | Z)$ **using** mutual-information-eq-entropy-conditional-entropy[OF assms].
finally show ?thesis **by** auto
qed

lemma (in information-space)
fixes $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \text{real}$
assumes S : sigma-finite-measure S **and** T : sigma-finite-measure T
assumes Px : finite-entropy $S X Px$ **and** Py : finite-entropy $T Y Py$
assumes Pxy : finite-entropy $(S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$
shows conditional-entropy $b S T X Y \leq \text{entropy } b S X$
proof –

have $0 \leq \text{mutual-information } b S T X Y$
by (rule mutual-information-nonneg') fact+
also have $\dots = \text{entropy } b S X - \text{conditional-entropy } b S T X Y$
apply (rule mutual-information-eq-entropy-conditional-entropy')
using assms

by (*auto intro!*: *finite-entropy-integrable finite-entropy-distributed*
finite-entropy-integrable-transform[*OF Px*]
finite-entropy-integrable-transform[*OF Py*]
intro: finite-entropy-nn)
finally show *?thesis* **by** *auto*
qed

lemma (*in information-space*) *entropy-chain-rule*:
assumes *X: simple-function M X* **and** *Y: simple-function M Y*
shows $\mathcal{H}(\lambda x. (X x, Y x)) = \mathcal{H}(X) + \mathcal{H}(Y|X)$
proof –
note *XY = simple-distributedI*[*OF simple-function-Pair*[*OF X Y*] *measure-nonneg*
refl]
note *YX = simple-distributedI*[*OF simple-function-Pair*[*OF Y X*] *measure-nonneg*
refl]
note *simple-distributed-joint-finite*[*OF this, simp*]
let *?f = λx. prob ((λx. (X x, Y x)) - ‘ {x} ∩ space M)*
let *?g = λx. prob ((λx. (Y x, X x)) - ‘ {x} ∩ space M)*
let *?h = λx. if x ∈ (λx. (Y x, X x)) ‘ space M then prob ((λx. (Y x, X x)) - ‘*
{x} ∩ space M) else 0
have $\mathcal{H}(\lambda x. (X x, Y x)) = - (\sum x \in (\lambda x. (X x, Y x)) \text{ ‘ space } M. ?f x * \log b (?f$
x))
using *XY by (rule entropy-simple-distributed)*
also have $\dots = - (\sum x \in (\lambda(x, y). (y, x)) \text{ ‘ } (\lambda x. (X x, Y x)) \text{ ‘ space } M. ?g x *$
log b (?g x))
by (*subst* (*?*) *sum.reindex*) (*auto simp: inj-on-def intro!: sum.cong arg-cong*[**where**
*f = λA. prob A * log b (prob A)*])
also have $\dots = - (\sum x \in (\lambda x. (Y x, X x)) \text{ ‘ space } M. ?h x * \log b (?h x))$
by (*auto intro!: sum.cong*)
also have $\dots = \text{entropy } b (\text{count-space } (Y \text{ ‘ space } M) \otimes_M \text{count-space } (X \text{ ‘$
*space } M)) (\lambda x. (Y x, X x))
by (*subst entropy-distr*[*OF simple-distributed-joint*[*OF YX*]])
(auto simp: pair-measure-count-space sigma-finite-measure-count-space-finite
lebesgue-integral-count-space-finite
cong del: sum.strong-cong intro!: sum.mono-neutral-left measure-nonneg)
finally have $\mathcal{H}(\lambda x. (X x, Y x)) = \text{entropy } b (\text{count-space } (Y \text{ ‘ space } M) \otimes_M$
*count-space } (X \text{ ‘ space } M)) (\lambda x. (Y x, X x)) .
then show *?thesis*
unfolding *conditional-entropy-eq-entropy-simple*[*OF Y X*] **by** *simp*
qed**

lemma (*in information-space*) *entropy-partition*:
assumes *X: simple-function M X*
shows $\mathcal{H}(X) = \mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$
proof –
note *fX = simple-function-compose*[*OF X, of f*]
have *eq: (λx. ((f ∘ X) x, X x)) ‘ space M = (λx. (f x, x)) ‘ X ‘ space M* **by**
auto
have *inj: ∧A. inj-on (λx. (f x, x)) A*

```

  by (auto simp: inj-on-def)
show ?thesis
  apply (subst entropy-chain-rule[symmetric, OF fX X])
  apply (subst entropy-simple-distributed[OF simple-distributedI[OF simple-function-Pair[OF
fX X] measure-nonneg refl]])
  apply (subst entropy-simple-distributed[OF simple-distributedI[OF X measure-nonneg
refl]])
  unfolding eq
  apply (subst sum.reindex[OF inj])
  apply (auto intro!: sum.cong arg-cong[where f= $\lambda A. \text{prob } A * \log b (\text{prob } A)$ ])
done
qed

```

```

corollary (in information-space) entropy-data-processing:
  assumes X: simple-function M X shows  $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$ 
proof -
  note fX = simple-function-compose[OF X, of f]
  from X have  $\mathcal{H}(X) = \mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$  by (rule entropy-partition)
  then show  $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$ 
    by (auto intro: conditional-entropy-nonneg[OF X fX])
qed

```

```

corollary (in information-space) entropy-of-inj:
  assumes X: simple-function M X and inj: inj-on f (X'space M)
  shows  $\mathcal{H}(f \circ X) = \mathcal{H}(X)$ 
proof (rule antisym)
  show  $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$  using entropy-data-processing[OF X] .
next
  have sf: simple-function M (f  $\circ$  X)
    using X by auto
  have  $\mathcal{H}(X) = \mathcal{H}(\text{the-inv-into } (X'space M) f \circ (f \circ X))$ 
    unfolding o-assoc
  apply (subst entropy-simple-distributed[OF simple-distributedI[OF X measure-nonneg
refl]])
  apply (subst entropy-simple-distributed[OF simple-distributedI[OF simple-function-compose[OF
X], where f= $\lambda x. \text{prob } (X - \{x\} \cap \text{space } M)$ ])
  apply (auto intro!: sum.cong arg-cong[where f=prob] image-eqI simp: the-inv-into-f-f[OF
inj] comp-def measure-nonneg)
  done
  also have ...  $\leq \mathcal{H}(f \circ X)$ 
    using entropy-data-processing[OF sf] .
  finally show  $\mathcal{H}(X) \leq \mathcal{H}(f \circ X)$  .
qed

```

end

15 Properties of Various Distributions

theory *Distributions*

imports *Convolution Information*
begin

lemma (in *prob-space*) *distributed-affine*:

fixes $f :: \text{real} \Rightarrow \text{ennreal}$

assumes f : *distributed M lborel X f*

assumes c : $c \neq 0$

shows *distributed M lborel* $(\lambda x. t + c * X x)$ $(\lambda x. f ((x - t) / c) / |c|)$

unfolding *distributed-def*

proof *safe*

have [*measurable*]: $f \in \text{borel-measurable borel}$

using f **by** (*simp add: distributed-def*)

have [*measurable*]: $X \in \text{borel-measurable M}$

using f **by** (*simp add: distributed-def*)

show $(\lambda x. f ((x - t) / c) / |c|) \in \text{borel-measurable lborel}$

by *simp*

show *random-variable lborel* $(\lambda x. t + c * X x)$

by *simp*

have *eq*: $\text{ennreal } |c| * (f x / \text{ennreal } |c|) = f x$ **for** x

using c

by (*cases f x*)

(*auto simp: divide-ennreal ennreal-mult[symmetric] ennreal-top-divide ennreal-mult-top*)

have *density lborel f = distr M lborel X*

using f **by** (*simp add: distributed-def*)

with c **show** *distr M lborel* $(\lambda x. t + c * X x) = \text{density lborel } (\lambda x. f ((x - t) / c) / \text{ennreal } |c|)$

by (*subst (2) lborel-real-affine[where c=c and t=t]*)

(*simp-all add: density-density-eq density-distr distr-distr field-simps eq cong: distr-cong*)

qed

lemma (in *prob-space*) *distributed-affineI*:

fixes $f :: \text{real} \Rightarrow \text{ennreal}$ **and** $c :: \text{real}$

assumes f : *distributed M lborel* $(\lambda x. (X x - t) / c)$ $(\lambda x. |c| * f (x * c + t))$

assumes c : $c \neq 0$

shows *distributed M lborel X f*

proof –

have *eq*: $f x * \text{ennreal } |c| / \text{ennreal } |c| = f x$ **for** x

using c **by** (*simp add: ennreal-times-divide[symmetric]*)

show *?thesis*

using *distributed-affine[OF f c, where t=t] c*

by (*simp add: field-simps eq*)

qed

lemma (in *prob-space*) *distributed-AE2*:

assumes $[measurable]: distributed\ M\ N\ X\ f\ Measurable.pred\ N\ P$
shows $(AE\ x\ in\ M. P\ (X\ x)) \longleftrightarrow (AE\ x\ in\ N. 0 < f\ x \longrightarrow P\ x)$
proof –
have $(AE\ x\ in\ M. P\ (X\ x)) \longleftrightarrow (AE\ x\ in\ distr\ M\ N\ X. P\ x)$
by $(simp\ add: AE-distr-iff)$
also have $\dots \longleftrightarrow (AE\ x\ in\ density\ N\ f. P\ x)$
unfolding $distributed-distr-eq-density[OF\ assms(1)] \dots$
also have $\dots \longleftrightarrow (AE\ x\ in\ N. 0 < f\ x \longrightarrow P\ x)$
by $(rule\ AE-density)\ simp$
finally show $?thesis\ .$
qed

15.1 Erlang

lemma $nn-integral-power-times-exp-Icc:$

assumes $[arith]: 0 \leq a$
shows $(\int^{+x}. ennreal\ (x^k * exp\ (-x)) * indicator\ \{0..a\}\ x\ \partial lborel) =$
 $(1 - (\sum_{n \leq k}. (a^n * exp\ (-a)) / fact\ n)) * fact\ k\ (is\ ?I = -)$
proof –
let $?f = \lambda k\ x. x^k * exp\ (-x) / fact\ k$
let $?F = \lambda k\ x. - (\sum_{n \leq k}. (x^n * exp\ (-x)) / fact\ n)$
have $?I * (inverse\ (real-of-nat\ (fact\ k))) =$
 $(\int^{+x}. ennreal\ (x^k * exp\ (-x)) * indicator\ \{0..a\}\ x * (inverse\ (real-of-nat\ (fact\ k))))\ \partial lborel)$
by $(intro\ nn-integral-mult[symmetric])\ auto$
also have $\dots = (\int^{+x}. ennreal\ (?f\ k\ x) * indicator\ \{0..a\}\ x\ \partial lborel)$
by $(intro\ nn-integral-cong)$
 $(simp\ add: field-simps\ ennreal-mult'[symmetric]\ indicator-mult-ennreal)$
also have $\dots = ennreal\ (?F\ k\ a - ?F\ k\ 0)$
proof $(rule\ nn-integral-FTC-Icc)$
fix x **assume** $x \in \{0..a\}$
show $DERIV\ (?F\ k)\ x\ :=\ ?f\ k\ x$
proof $(induction\ k)$
case 0 **show** $?case$ **by** $(auto\ intro!: derivative-eq-intros)$
next
case $(Suc\ k)$
have $DERIV\ (\lambda x. ?F\ k\ x - (x^{Suc\ k} * exp\ (-x)) / fact\ (Suc\ k))\ x\ :=$
 $?f\ k\ x - ((real\ (Suc\ k) - x) * x^k * exp\ (-x)) / (fact\ (Suc\ k))$
by $(intro\ DERIV-diff\ Suc)$
 $(auto\ intro!: derivative-eq-intros\ simp\ del: fact-Suc\ power-Suc$
 $simp\ add: field-simps\ power-Suc[symmetric])$
also have $(\lambda x. ?F\ k\ x - (x^{Suc\ k} * exp\ (-x)) / fact\ (Suc\ k)) = ?F\ (Suc\ k)$
by $simp$
also have $?f\ k\ x - ((real\ (Suc\ k) - x) * x^k * exp\ (-x)) / (fact\ (Suc\ k))$
 $= ?f\ (Suc\ k)\ x$
by $(auto\ simp: field-simps\ simp\ del: fact-Suc)$
 $(simp-all\ add: of-nat-Suc\ field-simps)$
finally show $?case\ .$
qed

qed *auto*
also have $\dots = \text{ennreal } (1 - (\sum_{n \leq k}. (a^n * \exp(-a)) / \text{fact } n))$
by (*auto simp: power-0-left if-distrib[where f= $\lambda x. x / a$ for a] sum.If-cases*)
also have $\dots = \text{ennreal } ((1 - (\sum_{n \leq k}. (a^n * \exp(-a)) / \text{fact } n)) * \text{fact } k)$
 $* \text{ennreal } (\text{inverse } (\text{fact } k))$
by (*subst ennreal-mult''[symmetric]*) (*auto intro!: arg-cong[where f=ennreal]*)
finally show *?thesis*
by (*auto simp add: mult-right-ennreal-cancel le-less*)
qed

lemma *nn-integral-power-times-exp-Ici*:

shows $(\int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots\} x \ \partial \text{lborel}) = \text{real-of-nat } (\text{fact } k)$

proof (*rule LIMSEQ-unique*)

let $?X = \lambda n. \int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots \text{real } n\} x \ \partial \text{lborel}$

show $?X \longrightarrow (\int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots\} x \ \partial \text{lborel})$

apply (*intro nn-integral-LIMSEQ*)

apply (*auto simp: incseq-def le-fun-def eventually-sequentially*
split: split-indicator intro!: Lim-eventually)

apply (*metis nat-ceiling-le-eq*)

done

have $((\lambda x::\text{real}. (1 - (\sum_{n \leq k}. (x^n / \exp x) / (\text{fact } n))) * \text{fact } k) \longrightarrow$
 $(1 - (\sum_{n \leq k}. 0 / (\text{fact } n))) * \text{fact } k)$ *at-top*

by (*intro tendsto-intros tendsto-power-div-exp-0 simp*)

then show $?X \longrightarrow \text{real-of-nat } (\text{fact } k)$

by (*subst nn-integral-power-times-exp-Icc*)

(*auto simp: exp-minus field-simps intro!: filterlim-compose[OF - filterlim-real-sequentially]*)

qed

definition *erlang-density* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**

erlang-density $k \ l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } (l^{(\text{Suc } k)} * x^k * \exp(-l * x)) / \text{fact } k)$

definition *erlang-CDF* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**

erlang-CDF $k \ l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } 1 - (\sum_{n \leq k}. ((l * x)^n * \exp(-l * x)) / \text{fact } n))$

lemma *erlang-density-nonneg[simp]*: $0 \leq l \Longrightarrow 0 \leq \text{erlang-density } k \ l \ x$

by (*simp add: erlang-density-def*)

lemma *borel-measurable-erlang-density[measurable]*: $\text{erlang-density } k \ l \in \text{borel-measurable borel}$

by (*auto simp add: erlang-density-def[abs-def]*)

lemma *erlang-CDF-transform*: $0 < l \Longrightarrow \text{erlang-CDF } k \ l \ a = \text{erlang-CDF } k \ 1 \ (l * a)$

by (*auto simp add: erlang-CDF-def mult-less-0-iff*)

lemma *erlang-CDF-nonneg*[simp]: **assumes** $0 < l$ **shows** $0 \leq \text{erlang-CDF } k \ l \ x$
unfolding *erlang-CDF-def*

proof (*clarsimp simp: not-less*)

assume $0 \leq x$

have $(\sum_{n \leq k}. (l * x) ^ n * \exp(- (l * x)) / \text{fact } n) =$
 $\exp(- (l * x)) * (\sum_{n \leq k}. (l * x) ^ n / \text{fact } n)$

unfolding *sum-distrib-left* **by** (*intro sum.cong*) (*auto simp: field-simps*)

also have $\dots = (\sum_{n \leq k}. (l * x) ^ n / \text{fact } n) / \exp(l * x)$
by (*simp add: exp-minus field-simps*)

also have $\dots \leq 1$

proof (*subst divide-le-eq-1-pos*)

show $(\sum_{n \leq k}. (l * x) ^ n / \text{fact } n) \leq \exp(l * x)$

using $\langle 0 < l \rangle \langle 0 \leq x \rangle$ *summable-exp-generic*[of $l * x$]

by (*auto simp: exp-def divide-inverse ac-simps intro!: sum-le-suminf*)

qed *simp*

finally show $(\sum_{n \leq k}. (l * x) ^ n * \exp(- (l * x)) / \text{fact } n) \leq 1$.

qed

lemma *nn-integral-erlang-density*:

assumes [*arith*]: $0 < l$

shows $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. a\} \ x \ \partial \text{lborel}) =$
erlang-CDF } k \ l \ a

proof (*cases* $0 \leq a$)

case [*arith*]: *True*

have *eq*: $\bigwedge x. \text{indicator } \{0..a\} (x / l) = \text{indicator } \{0..a * l\} x$

by (*simp add: field-simps split: split-indicator*)

have $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. a\} \ x \ \partial \text{lborel}) =$

$(\int^+ x. (l / \text{fact } k) * (\text{ennreal } ((l * x) ^ k * \exp(- (l * x))) * \text{indicator } \{0 .. a\} \ x) \ \partial \text{lborel})$

by (*intro nn-integral-cong*)

(auto simp: erlang-density-def power-mult-distrib ennreal-mult[*symmetric*]
split: split-indicator)

also have $\dots = (l / \text{fact } k) * (\int^+ x. \text{ennreal } ((l * x) ^ k * \exp(- (l * x))) * \text{indicator } \{0 .. a\} \ x \ \partial \text{lborel})$

by (*intro nn-integral-cmult*) *auto*

also have $\dots = \text{ennreal } (l / \text{fact } k) * ((1 / l) * (\int^+ x. \text{ennreal } (x ^ k * \exp(- x)) * \text{indicator } \{0 .. l * a\} \ x \ \partial \text{lborel}))$

by (*subst nn-integral-real-affine*[**where** $c=1 / l$ **and** $t=0$]) (*auto simp: field-simps eq*)

also have $\dots = (1 - (\sum_{n \leq k}. ((l * a) ^ n * \exp(- (l * a))) / \text{fact } n))$

by (*subst nn-integral-power-times-exp-Icc*) (*auto simp: ennreal-mult*[*symmetric*])

also have $\dots = \text{erlang-CDF } k \ l \ a$

by (*auto simp: erlang-CDF-def*)

finally show *?thesis* .

next

case *False*

then have $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. a\} \ x \ \partial \text{lborel}) =$
 $(\int^+ x. 0 \ \partial (\text{lborel}::\text{real measure}))$

by (*intro nn-integral-cong*) (*auto simp: erlang-density-def*)

with *False* **show** *?thesis*
by (*simp add: erlang-CDF-def*)
qed

lemma *emeasure-erlang-density*:

$0 < l \implies \text{emeasure } (\text{density } \text{lborel } (\text{erlang-density } k \ l)) \ \{.. \ a\} = \text{erlang-CDF } k \ l$
 a

by (*simp add: emeasure-density nn-integral-erlang-density*)

lemma *nn-integral-erlang-ith-moment*:

fixes $k \ i :: \text{nat}$ **and** $l :: \text{real}$

assumes [*arith*]: $0 < l$

shows $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x * x^i) \ \partial \text{lborel}) = \text{fact } (k + i) /$
 $(\text{fact } k * l^i)$

proof –

have *eq*: $\bigwedge x. \text{indicator } \{0..\} (x / l) = \text{indicator } \{0..\} x$

by (*simp add: field-simps split: split-indicator*)

have $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x * x^i) \ \partial \text{lborel}) =$

$(\int^+ x. (l / (\text{fact } k * l^i)) * (\text{ennreal } ((l*x)^{(k+i)} * \exp(- (l*x))) * \text{indicator}$
 $\{0 ..\} x) \ \partial \text{lborel})$

by (*intro nn-integral-cong*)

(*auto simp: erlang-density-def power-mult-distrib power-add ennreal-mult'[symmetric]*)

split: split-indicator)

also have $\dots = (l / (\text{fact } k * l^i)) * (\int^+ x. \text{ennreal } ((l*x)^{(k+i)} * \exp(- (l*x)))$

$* \text{indicator } \{0 ..\} x \ \partial \text{lborel})$

by (*intro nn-integral-cmult*) *auto*

also have $\dots = \text{ennreal } (l / (\text{fact } k * l^i)) * ((1/l) * (\int^+ x. \text{ennreal } (x^{(k+i)} * \exp(- x)) * \text{indicator } \{0 ..\} x \ \partial \text{lborel}))$

by (*subst nn-integral-real-affine[where c=1 / l and t=0]*) (*auto simp: field-simps eq*)

also have $\dots = \text{fact } (k + i) / (\text{fact } k * l^i)$

by (*subst nn-integral-power-times-exp-Ici*) (*auto simp: ennreal-mult'[symmetric]*)

finally show *?thesis* .

qed

lemma *prob-space-erlang-density*:

assumes [*arith*]: $0 < l$

shows *prob-space* (*density* *lborel* (*erlang-density* $k \ l$)) (**is** *prob-space* *?D*)

proof

show *emeasure* *?D* (*space* *?D*) = 1

using *nn-integral-erlang-ith-moment*[*OF* l , **where** $k=k$ **and** $i=0$] **by** (*simp add: emeasure-density*)

qed

lemma (**in** *prob-space*) *erlang-distributed-le*:

assumes D : *distributed* $M \ \text{lborel } X$ (*erlang-density* $k \ l$)

assumes [*simp, arith*]: $0 < l \leq a$

shows $\mathcal{P}(x \text{ in } M. X \ x \leq a) = \text{erlang-CDF } k \ l \ a$

proof –

```

have emeasure  $M \{x \in \text{space } M. X x \leq a\} = \text{emeasure } (\text{distr } M \text{ lborel } X) \{.. a\}$ 
using distributed-measurable[OF D]
by (subst emeasure-distr) (auto intro!: arg-cong2[where f=emeasure])
also have ... = emeasure (density lborel (erlang-density k l)) {.. a}
unfolding distributed-distr-eq-density[OF D] ..
also have ... = erlang-CDF k l a
by (auto intro!: emeasure-erlang-density)
finally show ?thesis
by (auto simp: emeasure-eq-measure measure-nonneg)
qed

```

```

lemma (in prob-space) erlang-distributed-gt:
  assumes D[simp]: distributed M lborel X (erlang-density k l)
  assumes [arith]:  $0 < l \ 0 \leq a$ 
  shows  $\mathcal{P}(x \text{ in } M. a < X x) = 1 - (\text{erlang-CDF } k \ l \ a)$ 
proof -
  have  $1 - (\text{erlang-CDF } k \ l \ a) = 1 - \mathcal{P}(x \text{ in } M. X x \leq a)$  by (subst erlang-distributed-le)
  auto
  also have ... = prob (space M - {x ∈ space M. X x ≤ a})
  using distributed-measurable[OF D] by (auto simp: prob-compl)
  also have ... =  $\mathcal{P}(x \text{ in } M. a < X x)$  by (auto intro!: arg-cong[where f=prob])
  simp: not-le
  finally show ?thesis by simp
qed

```

```

lemma erlang-CDF-at0: erlang-CDF k l 0 = 0
by (induction k) (auto simp: erlang-CDF-def)

```

```

lemma erlang-distributedI:
  assumes X[measurable]:  $X \in \text{borel-measurable } M$  and [arith]:  $0 < l$ 
  and X-distr:  $\bigwedge a. 0 \leq a \implies \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{erlang-CDF } k \ l \ a$ 
  shows distributed M lborel X (erlang-density k l)
proof (rule distributedI-borel-atMost)
  fix a :: real
  { assume  $a \leq 0$ 
    with X have emeasure M {x ∈ space M. X x ≤ a} ≤ emeasure M {x ∈ space M. X x ≤ 0}
    by (intro emeasure-mono) auto
    also have ... = 0 by (auto intro!: erlang-CDF-at0 simp: X-distr[of 0])
    finally have emeasure M {x ∈ space M. X x ≤ a} ≤ 0 by simp
    then have emeasure M {x ∈ space M. X x ≤ a} = 0 by simp
  }
  note eq-0 = this

```

```

show  $(\int^+ x. \text{erlang-density } k \ l \ x * \text{indicator } \{..a\} x \ \partial \text{lborel}) = \text{ennreal } (\text{erlang-CDF } k \ l \ a)$ 
using nn-integral-erlang-density[of l k a]

```

by (*simp add: ennreal-indicator ennreal-mult*)

show $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal } (\text{erlang-CDF } k \ l \ a)$
 using *X-distr[of a] eq-0* by (*auto simp: one-ennreal-def erlang-CDF-def*)
 qed *simp-all*

lemma (*in prob-space*) *erlang-distributed-iff*:

assumes [*arith*]: $0 < l$

shows $\text{distributed } M \ \text{lborel } X \ (\text{erlang-density } k \ l) \longleftrightarrow$

$(X \in \text{borel-measurable } M \wedge 0 < l \wedge (\forall a \geq 0. \mathcal{P}(x \text{ in } M. X x \leq a) = \text{erlang-CDF } k \ l \ a))$

using

distributed-measurable[of M lborel X erlang-density k l]

emeasure-erlang-density[of l]

erlang-distributed-le[of X k l]

by (*auto intro!: erlang-distributedI simp: one-ennreal-def emeasure-eq-measure*)

lemma (*in prob-space*) *erlang-distributed-mult-const*:

assumes *erlX*: $\text{distributed } M \ \text{lborel } X \ (\text{erlang-density } k \ l)$

assumes *a-pos[arith]*: $0 < \alpha \ 0 < l$

shows $\text{distributed } M \ \text{lborel } (\lambda x. \alpha * X x) \ (\text{erlang-density } k \ (l / \alpha))$

proof (*subst erlang-distributed-iff, safe*)

have [*measurable*]: *random-variable borel X* and [*arith*]: $0 < l$

and [*simp*]: $\bigwedge a. 0 \leq a \implies \text{prob } \{x \in \text{space } M. X x \leq a\} = \text{erlang-CDF } k \ l \ a$
 by(*insert erlX, auto simp: erlang-distributed-iff*)

show *random-variable borel* $(\lambda x. \alpha * X x) \ 0 < l / \alpha \ 0 < l / \alpha$

by (*auto simp: field-simps*)

fix *a*:: *real* assume [*arith*]: $0 \leq a$

obtain *b*:: *real* where [*simp, arith*]: $b = a / \alpha$ by *blast*

have [*arith*]: $0 \leq b$ by (*auto simp: divide-nonneg-pos*)

have $\text{prob } \{x \in \text{space } M. \alpha * X x \leq a\} = \text{prob } \{x \in \text{space } M. X x \leq b\}$

by (*rule arg-cong[where f=prob]*) (*auto simp: field-simps*)

moreover have $\text{prob } \{x \in \text{space } M. X x \leq b\} = \text{erlang-CDF } k \ l \ b$ by *auto*

moreover have $\text{erlang-CDF } k \ (l / \alpha) \ a = \text{erlang-CDF } k \ l \ b$ **unfolding** *erlang-CDF-def*
 by *auto*

ultimately show $\text{prob } \{x \in \text{space } M. \alpha * X x \leq a\} = \text{erlang-CDF } k \ (l / \alpha) \ a$

by *fastforce*

qed

lemma (*in prob-space*) *has-bochner-integral-erlang-ith-moment*:

fixes *k i* :: *nat* and *l* :: *real*

assumes [*arith*]: $0 < l$ and *D*: $\text{distributed } M \ \text{lborel } X \ (\text{erlang-density } k \ l)$

shows *has-bochner-integral* $M \ (\lambda x. X x \wedge i) \ (\text{fact } (k + i) / (\text{fact } k * l \wedge i))$

proof (*rule has-bochner-integral-nn-integral*)

show $AE\ x\ in\ M.\ 0 \leq X\ x\ ^i$
by (*subst distributed-AE2*[*OF D*]) (*auto simp: erlang-density-def*)
show $(\int^+ x.\ ennreal\ (X\ x\ ^i)\ \partial M) = ennreal\ (fact\ (k + i) / (fact\ k * l\ ^i))$
using *nn-integral-erlang-ith-moment*[*of l k i*]
by (*subst distributed-nn-integral*[*symmetric, OF D*]) (*auto simp: ennreal-mult'*)
qed (*insert distributed-measurable*[*OF D*], *auto*)

lemma (*in prob-space*) *erlang-ith-moment-integrable*:
 $0 < l \implies distributed\ M\ lborel\ X\ (erlang-density\ k\ l) \implies integrable\ M\ (\lambda x.\ X\ x\ ^i)$
by *rule* (*rule has-bochner-integral-erlang-ith-moment*)

lemma (*in prob-space*) *erlang-ith-moment*:
 $0 < l \implies distributed\ M\ lborel\ X\ (erlang-density\ k\ l) \implies expectation\ (\lambda x.\ X\ x\ ^i) = fact\ (k + i) / (fact\ k * l\ ^i)$
by (*rule has-bochner-integral-integral-eq*) (*rule has-bochner-integral-erlang-ith-moment*)

lemma (*in prob-space*) *erlang-distributed-variance*:
assumes [*arith*]: $0 < l$ **and** *distributed M lborel X (erlang-density k l)*
shows *variance X = (k + 1) / l²*
proof (*subst variance-eq*)
show *integrable M X integrable M (\lambda x. (X x)²)*
using *erlang-ith-moment-integrable*[*OF assms, of 1*] *erlang-ith-moment-integrable*[*OF assms, of 2*]
by *auto*

show $expectation\ (\lambda x.\ (X\ x)^2) - (expectation\ X)^2 = real\ (k + 1) / l^2$
using *erlang-ith-moment*[*OF assms, of 1*] *erlang-ith-moment*[*OF assms, of 2*]
by *simp* (*auto simp: power2-eq-square field-simps of-nat-Suc*)
qed

15.2 Exponential distribution

abbreviation *exponential-density* :: *real* \Rightarrow *real* \Rightarrow *real* **where**
exponential-density \equiv *erlang-density 0*

lemma *exponential-density-def*:
 $exponential-density\ l\ x = (if\ x < 0\ then\ 0\ else\ l * exp\ (-x * l))$
by (*simp add: fun-eq-iff erlang-density-def*)

lemma *erlang-CDF-0*: $erlang-CDF\ 0\ l\ a = (if\ 0 \leq a\ then\ 1 - exp\ (-l * a)\ else\ 0)$
by (*simp add: erlang-CDF-def*)

lemma *prob-space-exponential-density*: $0 < l \implies prob-space\ (density\ lborel\ (exponential-density\ l))$
by (*rule prob-space-erlang-density*)

lemma (*in prob-space*) *exponential-distributedD-le*:

assumes D : distributed M lborel X (exponential-density l) **and** a : $0 \leq a$ **and** l : $0 < l$
shows $\mathcal{P}(x \text{ in } M. X x \leq a) = 1 - \exp(-a * l)$
using erlang-distributed-le[OF D l a] **by** (simp add: erlang-CDF-def)

lemma (in prob-space) exponential-distributedD-gt:
assumes D : distributed M lborel X (exponential-density l) **and** a : $0 \leq a$ **and** l : $0 < l$
shows $\mathcal{P}(x \text{ in } M. a < X x) = \exp(-a * l)$
using erlang-distributed-gt[OF D l a] **by** (simp add: erlang-CDF-def)

lemma (in prob-space) exponential-distributed-memoryless:
assumes D : distributed M lborel X (exponential-density l) **and** a : $0 \leq a$ **and** l : $0 < l$ **and** t : $0 \leq t$
shows $\mathcal{P}(x \text{ in } M. a + t < X x \mid a < X x) = \mathcal{P}(x \text{ in } M. t < X x)$
proof –
have $\mathcal{P}(x \text{ in } M. a + t < X x \mid a < X x) = \mathcal{P}(x \text{ in } M. a + t < X x) / \mathcal{P}(x \text{ in } M. a < X x)$
using $\langle 0 \leq t \rangle$ **by** (auto simp: cond-prob-def intro!: arg-cong[where $f=prob$] arg-cong2[where $f=op$ /])
also have $\dots = \exp(-(a + t) * l) / \exp(-a * l)$
using a t **by** (simp add: exponential-distributedD-gt[OF D - l])
also have $\dots = \exp(-t * l)$
using l **by** (auto simp: field-simps exp-add[symmetric])
finally show ?thesis
using t **by** (simp add: exponential-distributedD-gt[OF D - l])
qed

lemma exponential-distributedI:
assumes X [measurable]: $X \in$ borel-measurable M **and** [arith]: $0 < l$
and X -distr: $\bigwedge a. 0 \leq a \implies \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = 1 - \exp(-a * l)$
shows distributed M lborel X (exponential-density l)
proof (rule erlang-distributedI)
fix a :: real **assume** $0 \leq a$ **then show** $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal}(\text{erlang-CDF } 0 \ l \ a)$
using X -distr[of a] **by** (simp add: erlang-CDF-def ennreal-minus ennreal-1[symmetric] del: ennreal-1)
qed fact+

lemma (in prob-space) exponential-distributed-iff:
assumes $0 < l$
shows distributed M lborel X (exponential-density l) \iff
 $(X \in \text{borel-measurable } M \wedge (\forall a \geq 0. \mathcal{P}(x \text{ in } M. X x \leq a) = 1 - \exp(-a * l)))$
using assms erlang-distributed-iff[of l X 0] **by** (auto simp: erlang-CDF-0)

lemma (in prob-space) exponential-distributed-expectation:

$0 < l \implies \text{distributed } M \text{ lborel } X \text{ (exponential-density } l) \implies \text{expectation } X = 1 / l$

using *erlang-ith-moment*[of l X 0 1] **by** *simp*

lemma *exponential-density-nonneg*: $0 < l \implies 0 \leq \text{exponential-density } l \ x$
by (*auto simp: exponential-density-def*)

lemma (*in prob-space*) *exponential-distributed-min*:

assumes $0 < l \ 0 < u$

assumes *expX*: *distributed M lborel X (exponential-density l)*

assumes *expY*: *distributed M lborel Y (exponential-density u)*

assumes *ind*: *indep-var borel X borel Y*

shows *distributed M lborel* $(\lambda x. \text{min } (X \ x) \ (Y \ x)) \text{ (exponential-density } (l + u))$

proof (*subst exponential-distributed-iff, safe*)

have *randX*: *random-variable borel X*

using *expX* $\langle 0 < l \rangle$ **by** (*simp add: exponential-distributed-iff*)

moreover have *randY*: *random-variable borel Y*

using *expY* $\langle 0 < u \rangle$ **by** (*simp add: exponential-distributed-iff*)

ultimately show *random-variable borel* $(\lambda x. \text{min } (X \ x) \ (Y \ x))$ **by** *auto*

show $0 < l + u$

using $\langle 0 < l \rangle \ \langle 0 < u \rangle$ **by** *auto*

fix *a::real* **assume** *a[arith]*: $0 \leq a$

have *gt1*[*simp*]: $\mathcal{P}(x \text{ in } M. a < X \ x) = \text{exp } (- a * l)$

by (*rule exponential-distributedD-gt*[*OF expX a*]) *fact*

have *gt2*[*simp*]: $\mathcal{P}(x \text{ in } M. a < Y \ x) = \text{exp } (- a * u)$

by (*rule exponential-distributedD-gt*[*OF expY a*]) *fact*

have $\mathcal{P}(x \text{ in } M. a < (\text{min } (X \ x) \ (Y \ x))) = \mathcal{P}(x \text{ in } M. a < (X \ x) \wedge a < (Y \ x))$
by (*auto intro!: arg-cong*[*where f=prob*])

also have $\dots = \mathcal{P}(x \text{ in } M. a < (X \ x)) * \mathcal{P}(x \text{ in } M. a < (Y \ x))$

using *prob-indep-random-variable*[*OF ind, of* $\{a <..\} \ \{a <..\}$] **by** *simp*

also have $\dots = \text{exp } (- a * (l + u))$ **by** (*auto simp: field-simps mult-exp-exp*)

finally have *indep-prob*: $\mathcal{P}(x \text{ in } M. a < (\text{min } (X \ x) \ (Y \ x))) = \text{exp } (- a * (l + u))$.

have $\{x \in \text{space } M. (\text{min } (X \ x) \ (Y \ x)) \leq a\} = (\text{space } M - \{x \in \text{space } M. a < (\text{min } (X \ x) \ (Y \ x))\})$

by *auto*

then have $1 - \text{prob } \{x \in \text{space } M. a < (\text{min } (X \ x) \ (Y \ x))\} = \text{prob } \{x \in \text{space } M. (\text{min } (X \ x) \ (Y \ x)) \leq a\}$

using *randX randY* **by** (*auto simp: prob-compl*)

then show $\text{prob } \{x \in \text{space } M. (\text{min } (X \ x) \ (Y \ x)) \leq a\} = 1 - \text{exp } (- a * (l + u))$

using *indep-prob* **by** *auto*

qed

lemma (in prob-space) exponential-distributed-Min:
assumes finI: finite I
assumes A: $I \neq \{\}$
assumes l: $\bigwedge i. i \in I \implies 0 < l\ i$
assumes expX: $\bigwedge i. i \in I \implies$ distributed M lborel (X i) (exponential-density (l i))
assumes ind: indep-vars ($\lambda i. \text{borel}$) X I
shows distributed M lborel ($\lambda x. \text{Min } ((\lambda i. X\ i\ x)^I)$) (exponential-density ($\sum i \in I. l\ i$))
using assms
proof (induct rule: finite-ne-induct)
case (singleton i) **then show** ?case **by simp**
next
case (insert i I)
then have distributed M lborel ($\lambda x. \text{min } (X\ i\ x) (\text{Min } ((\lambda i. X\ i\ x)^I))$) (exponential-density ($l\ i + (\sum i \in I. l\ i)$))
by (intro exponential-distributed-min indep-vars-Min insert)
(auto intro: indep-vars-subset sum-pos)
then show ?case
using insert by simp
qed

lemma (in prob-space) exponential-distributed-variance:
 $0 < l \implies$ distributed M lborel X (exponential-density l) \implies variance X = $1 / l^2$
using erlang-distributed-variance[of l X 0] **by simp**

lemma nn-integral-zero': AE x in M. $f\ x = 0 \implies (\int^{+x}. f\ x\ \partial M) = 0$
by (simp cong: nn-integral-cong-AE)

lemma convolution-erlang-density:
fixes $k_1\ k_2 :: \text{nat}$
assumes [simp, arith]: $0 < l$
shows ($\lambda x. \int^{+y}. \text{ennreal } (\text{erlang-density } k_1\ l\ (x - y)) * \text{ennreal } (\text{erlang-density } k_2\ l\ y)\ \partial \text{lborel}$) =
(erlang-density (Suc $k_1 + \text{Suc } k_2 - 1$) l)
(is ?LHS = ?RHS)
proof
fix $x :: \text{real}$
have $x \leq 0 \vee 0 < x$
by arith
then show ?LHS $x =$?RHS x
proof
assume $x \leq 0$ **then show** ?thesis
apply (subst nn-integral-zero')
apply (rule AE-I[where N={0}])
apply (auto simp add: erlang-density-def not-less)
done
next

```

note zero-le-mult-iff [simp] zero-le-divide-iff [simp]

have I-eq1: integralN lborel (erlang-density (Suc k1 + Suc k2 - 1) l) = 1
  using nn-integral-erlang-ith-moment[of l Suc k1 + Suc k2 - 1 0] by (simp
del: fact-Suc)

have 1: (∫+ x. ennreal (erlang-density (Suc k1 + Suc k2 - 1) l x * indicator
{0<..} x) ∂lborel) = 1
  apply (subst I-eq1[symmetric])
  unfolding erlang-density-def
  by (auto intro!: nn-integral-cong split:split-indicator)

have prob-space (density lborel ?LHS)
  by (intro prob-space-convolution-density)
  (auto intro!: prob-space-erlang-density erlang-density-nonneg)
then have 2: integralN lborel ?LHS = 1
  by (auto dest!: prob-space.emeasure-space-1 simp: emeasure-density)

let ?I = (integralN lborel (λy. ennreal ((1 - y)k1 * yk2 * indicator {0..1}
y)))
let ?C = (fact (Suc (k1 + k2))) / ((fact k1) * (fact k2))
let ?s = Suc k1 + Suc k2 - 1
let ?L = (λx. ∫+ y. ennreal (erlang-density k1 l (x - y) * erlang-density k2 l y
* indicator {0..x} y) ∂lborel)

{ fix x :: real assume [arith]: 0 < x
  have *: (∧x y n. (x - y * x::real)n = xn * (1 - y)n
    unfolding power-mult-distrib[symmetric] by (simp add: field-simps)

  have ?LHS x = ?L x
    unfolding erlang-density-def
    by (auto intro!: nn-integral-cong simp: ennreal-mult split:split-indicator)
  also have ... = (λx. ennreal ?C * ?I * erlang-density ?s l x) x
    apply (subst nn-integral-real-affine[where c=x and t = 0])
  apply (simp-all add: nn-integral-cmult[symmetric] nn-integral-multc[symmetric]
del: fact-Suc)
    apply (intro nn-integral-cong)
    apply (auto simp add: erlang-density-def mult-less-0-iff exp-minus field-simps
exp-diff power-add *
      ennreal-mult[symmetric]
      simp del: fact-Suc split: split-indicator)
  done
  finally have (∫+ y. ennreal (erlang-density k1 l (x - y) * erlang-density k2
l y) ∂lborel) =
    (λx. ennreal ?C * ?I * erlang-density ?s l x) x
    by (simp add: ennreal-mult) }
note * = this

assume [arith]: 0 < x

```

```

have 3: 1 = integralN lborel (λx. ?LHS xa * indicator {0<..} xa)
  by (subst 2[symmetric])
    (auto intro!: nn-integral-cong-AE AE-I[where N={0}])
      simp: erlang-density-def nn-integral-multc[symmetric] indicator-def
split: if-split-asm)
  also have ... = integralN lborel (λx. (ennreal (?C) * ?I) * ((erlang-density ?s
l x) * indicator {0<..} x))
  by (auto intro!: nn-integral-cong simp: ennreal-mult[symmetric] * split:
split-indicator)
  also have ... = ennreal (?C) * ?I
  using 1
  by (auto simp: nn-integral-cmult)
finally have ennreal (?C) * ?I = 1 by presburger

then show ?thesis
  using * by (simp add: ennreal-mult)
qed
qed

```

```

lemma (in prob-space) sum-indep-erlang:
  assumes indep: indep-var borel X borel Y
  assumes [simp, arith]: 0 < l
  assumes erlX: distributed M lborel X (erlang-density k1 l)
  assumes erlY: distributed M lborel Y (erlang-density k2 l)
  shows distributed M lborel (λx. X x + Y x) (erlang-density (Suc k1 + Suc k2 -
1) l)
  using assms
  apply (subst convolution-erlang-density[symmetric, OF <0<l])
  apply (intro distributed-convolution)
  apply auto
  done

```

```

lemma (in prob-space) erlang-distributed-sum:
  assumes finI : finite I
  assumes A: I ≠ {}
  assumes [simp, arith]: 0 < l
  assumes expX: ∧i. i ∈ I ⇒ distributed M lborel (X i) (erlang-density (k i) l)
  assumes ind: indep-vars (λi. borel) X I
  shows distributed M lborel (λx. ∑ i∈I. X i x) (erlang-density ((∑ i∈I. Suc (k
i)) - 1) l)
  using assms
  proof (induct rule: finite-ne-induct)
  case (singleton i) then show ?case by auto
  next
  case (insert i I)
  then have distributed M lborel (λx. (X i x) + (∑ i∈I. X i x)) (erlang-density
(Suc (k i) + Suc ((∑ i∈I. Suc (k i)) - 1) - 1) l)
  by (intro sum-indep-erlang indep-vars-sum) (auto intro!: indep-vars-subset)
  also have (λx. (X i x) + (∑ i∈I. X i x)) = (λx. ∑ i∈insert i I. X i x)

```

using *insert by auto*
also have $Suc(k\ i) + Suc((\sum i \in I. Suc(k\ i)) - 1) - 1 = (\sum i \in insert\ i\ I. Suc(k\ i)) - 1$
using *insert by (auto intro!: Suc-pred simp: ac-simps)*
finally show *?case by fast*
qed

lemma (in *prob-space*) *exponential-distributed-sum*:

assumes *finI: finite I*
assumes *A: I ≠ {}*
assumes *l: 0 < l*
assumes *expX: $\bigwedge i. i \in I \implies distributed\ M\ lborel\ (X\ i)$ (exponential-density l)*
assumes *ind: indep-vars ($\lambda i. borel$) X I*
shows *distributed M lborel ($\lambda x. \sum i \in I. X\ i\ x$) (erlang-density ((card I) - 1) l)*
using *erlang-distributed-sum[OF assms] by simp*

lemma (in *information-space*) *entropy-exponential*:

assumes *l[simp, arith]: 0 < l*
assumes *D: distributed M lborel X (exponential-density l)*
shows *entropy b lborel X = log b (exp 1 / l)*

proof –

have *[simp]: integrable lborel (exponential-density l)*
using *distributed-integrable[OF D, of $\lambda-. 1$] by simp*

have *[simp]: $\int^L lborel (exponential-density l) = 1$*
using *distributed-integral[OF D, of $\lambda-. 1$] by (simp add: prob-space)*

have *[simp]: integrable lborel ($\lambda x. exponential-density\ l\ x * x$)*
using *erlang-ith-moment-integrable[OF l D, of 1] distributed-integrable[OF D, of $\lambda x. x$] by simp*

have *[simp]: $\int^L lborel (\lambda x. exponential-density\ l\ x * x) = 1 / l$*
using *erlang-ith-moment[OF l D, of 1] distributed-integral[OF D, of $\lambda x. x$] by simp*

have *entropy b lborel X = - ($\int x. exponential-density\ l\ x * log\ b (exponential-density\ l\ x)$) $\partial lborel$*

using *D by (rule entropy-distr) simp*

also have *($\int x. exponential-density\ l\ x * log\ b (exponential-density\ l\ x)$) $\partial lborel$*

=

*($\int x. (ln\ l * exponential-density\ l\ x - l * (exponential-density\ l\ x * x)) / ln\ b$) $\partial lborel$*

by (*intro Bochner-Integration.integral-cong*) (*auto simp: log-def ln-mult exponential-density-def field-simps*)

also have *... = (ln l - 1) / ln b*

by *simp*

finally show *?thesis*

by (*simp add: log-def divide-simps ln-div*)

qed

15.3 Uniform distribution

lemma *uniform-distrI*:

assumes $X: X \in \text{measurable } M \ M'$
and $A: A \in \text{sets } M' \ \text{emeasure } M' \ A \neq \infty \ \text{emeasure } M' \ A \neq 0$
assumes $\text{distr}: \bigwedge B. B \in \text{sets } M' \implies \text{emeasure } M \ (X \text{ -' } B \cap \text{space } M) =$
 $\text{emeasure } M' \ (A \cap B) / \text{emeasure } M' \ A$
shows $\text{distr } M \ M' \ X = \text{uniform-measure } M' \ A$
unfolding *uniform-measure-def*
proof (*intro measure-eqI*)
let $?f = \lambda x. \text{indicator } A \ x / \text{emeasure } M' \ A$
fix B **assume** $B: B \in \text{sets } (\text{distr } M \ M' \ X)$
with X **have** $\text{emeasure } M \ (X \text{ -' } B \cap \text{space } M) = \text{emeasure } M' \ (A \cap B) /$
 $\text{emeasure } M' \ A$
by (*simp add: distr[of B] measurable-sets*)
also have $\dots = (1 / \text{emeasure } M' \ A) * \text{emeasure } M' \ (A \cap B)$
by (*simp add: divide-ennreal-def ac-simps*)
also have $\dots = (\int^+ x. (1 / \text{emeasure } M' \ A) * \text{indicator } (A \cap B) \ x \ \partial M')$
using $A \ B$
by (*intro nn-integral-cmult-indicator[symmetric]*) (*auto intro!*)
also have $\dots = (\int^+ x. ?f \ x * \text{indicator } B \ x \ \partial M')$
by (*rule nn-integral-cong*) (*auto split: split-indicator*)
finally show $\text{emeasure } (\text{distr } M \ M' \ X) \ B = \text{emeasure } (\text{density } M' \ ?f) \ B$
using $A \ B \ X$ **by** (*auto simp add: emeasure-distr emeasure-density*)
qed *simp*

lemma *uniform-distrI-borel*:

fixes $A :: \text{real set}$
assumes $X[\text{measurable}]: X \in \text{borel-measurable } M$ **and** $A: \text{emeasure } \text{lborel } A =$
 $\text{ennreal } r \ 0 < r$
and $[\text{measurable}]: A \in \text{sets borel}$
assumes $\text{distr}: \bigwedge a. \text{emeasure } M \ \{x \in \text{space } M. X \ x \leq a\} = \text{emeasure } \text{lborel } (A$
 $\cap \{.. a\}) / r$
shows $\text{distributed } M \ \text{lborel } X \ (\lambda x. \text{indicator } A \ x / \text{measure } \text{lborel } A)$
proof (*rule distributedI-borel-atMost*)
let $?f = \lambda x. 1 / r * \text{indicator } A \ x$
fix a
have $\text{emeasure } \text{lborel } (A \cap \{..a\}) \leq \text{emeasure } \text{lborel } A$
using A **by** (*intro emeasure-mono*) *auto*
also have $\dots < \infty$
using A **by** *simp*
finally have $\text{fin}: \text{emeasure } \text{lborel } (A \cap \{..a\}) \neq \text{top}$
by *simp*
from *emeasure-eq-ennreal-measure[OF this]*
have $\text{fin-eq}: \text{emeasure } \text{lborel } (A \cap \{..a\}) / r = \text{ennreal } (\text{measure } \text{lborel } (A \cap$
 $\{..a\}) / r)$
using A **by** (*simp add: divide-ennreal measure-nonneg*)
then show $\text{emeasure } M \ \{x \in \text{space } M. X \ x \leq a\} = \text{ennreal } (\text{measure } \text{lborel } (A$
 $\cap \{..a\}) / r)$
using distr **by** *simp*

```

have ( $\int^+ x. \text{ennreal} (\text{indicator } A \ x / \text{measure } \text{lborel } A * \text{indicator } \{..a\} \ x)$ 
 $\partial \text{lborel}$ ) =
  ( $\int^+ x. \text{ennreal} (1 / \text{measure } \text{lborel } A) * \text{indicator} (A \cap \{..a\}) \ x \ \partial \text{lborel}$ )
  by (auto intro! nn-integral-cong split: split-indicator)
also have ... =  $\text{ennreal} (1 / \text{measure } \text{lborel } A) * \text{emeasure } \text{lborel} (A \cap \{..a\})$ 
  using  $\langle A \in \text{sets } \text{borel} \rangle$ 
  by (intro nn-integral-cmult-indicator) (auto simp: measure-nonneg)
also have ... =  $\text{ennreal} (\text{measure } \text{lborel} (A \cap \{..a\}) / r)$ 
  unfolding emeasure-eq-ennreal-measure[OF fin] using  $A$ 
  by (simp add: measure-def ennreal-mult'[symmetric])
finally show ( $\int^+ x. \text{ennreal} (\text{indicator } A \ x / \text{measure } \text{lborel } A * \text{indicator } \{..a\}$ 
 $x) \ \partial \text{lborel}$ ) =
   $\text{ennreal} (\text{measure } \text{lborel} (A \cap \{..a\}) / r)$  .
qed (auto simp: measure-nonneg)

```

lemma (*in prob-space*) *uniform-distrI-borel-atLeastAtMost*:

```

fixes  $a \ b :: \text{real}$ 
assumes  $X: X \in \text{borel-measurable } M$  and  $a < b$ 
assumes distr:  $\bigwedge t. a \leq t \implies t \leq b \implies \mathcal{P}(x \text{ in } M. X \ x \leq t) = (t - a) / (b - a)$ 
shows distributed  $M \ \text{lborel } X \ (\lambda x. \text{indicator } \{a..b\} \ x / \text{measure } \text{lborel } \{a..b\})$ 
proof (rule uniform-distrI-borel)
  fix  $t$ 
  have  $t < a \vee (a \leq t \wedge t \leq b) \vee b < t$ 
  by auto
  then show  $\text{emeasure } M \ \{x \in \text{space } M. X \ x \leq t\} = \text{emeasure } \text{lborel} (\{a .. b\} \cap \{..t\}) / (b - a)$ 
  proof (elim disjE conjE)
    assume  $t < a$ 
    then have  $\text{emeasure } M \ \{x \in \text{space } M. X \ x \leq t\} \leq \text{emeasure } M \ \{x \in \text{space } M. X \ x \leq a\}$ 
    using  $X$  by (auto intro!: emeasure-mono measurable-sets)
    also have ... = 0
    using distr[of a]  $\langle a < b \rangle$  by (simp add: emeasure-eq-measure)
    finally have  $\text{emeasure } M \ \{x \in \text{space } M. X \ x \leq t\} = 0$ 
    by (simp add: antisym measure-nonneg)
    with  $\langle t < a \rangle$  show ?thesis by simp
  next
    assume  $\text{bnds: } a \leq t \leq b$ 
    have  $\{a..b\} \cap \{..t\} = \{a..t\}$ 
    using  $\text{bnds}$  by auto
    then show ?thesis using  $\langle a \leq t \rangle \langle a < b \rangle$ 
    using distr[OF bnds] by (simp add: emeasure-eq-measure divide-ennreal)
  next
    assume  $b < t$ 
    have  $1 = \text{emeasure } M \ \{x \in \text{space } M. X \ x \leq b\}$ 
    using distr[of b]  $\langle a < b \rangle$  by (simp add: one-ennreal-def emeasure-eq-measure)
    also have ...  $\leq \text{emeasure } M \ \{x \in \text{space } M. X \ x \leq t\}$ 

```



```

    using X ⟨b < t⟩ by (auto intro!: emeasure-mono measurable-sets)
    finally have emeasure M {x∈space M. X x ≤ t} = 1
      by (simp add: antisym emeasure-eq-measure)
    with ⟨b < t⟩ ⟨a < b⟩ show ?thesis by (simp add: measure-def divide-ennreal)
  qed
qed (insert X ⟨a < b⟩, auto)

```

lemma (in prob-space) uniform-distributed-measure:

```

  fixes a b :: real
  assumes D: distributed M lborel X (λx. indicator {a .. b} x / measure lborel {a
  .. b})
  assumes t: a ≤ t t ≤ b
  shows P(x in M. X x ≤ t) = (t - a) / (b - a)
proof -
  have emeasure M {x ∈ space M. X x ≤ t} = emeasure (distr M lborel X) {.. t}
    using distributed-measurable[OF D]
    by (subst emeasure-distr) (auto intro!: arg-cong2[where f=emeasure])
  also have ... = (∫+x. ennreal (1 / (b - a)) * indicator {a .. t} x ∂lborel)
    using distributed-borel-measurable[OF D] ⟨a ≤ t⟩ ⟨t ≤ b⟩
    unfolding distributed-distr-eq-density[OF D]
    by (subst emeasure-density)
    (auto intro!: nn-integral-cong simp: measure-def split: split-indicator)
  also have ... = ennreal (1 / (b - a)) * (t - a)
    using ⟨a ≤ t⟩ ⟨t ≤ b⟩
    by (subst nn-integral-cmult-indicator) auto
  finally show ?thesis
    using t by (simp add: emeasure-eq-measure ennreal-mult''[symmetric] measure-nonneg)
qed

```

lemma (in prob-space) uniform-distributed-bounds:

```

  fixes a b :: real
  assumes D: distributed M lborel X (λx. indicator {a .. b} x / measure lborel {a
  .. b})
  shows a < b
proof (rule ccontr)
  assume ¬ a < b
  then have {a .. b} = {} ∨ {a .. b} = {a .. a} by simp
  with uniform-distributed-params[OF D] show False
    by (auto simp: measure-def)
qed

```

lemma (in prob-space) uniform-distributed-iff:

```

  fixes a b :: real
  shows distributed M lborel X (λx. indicator {a..b} x / measure lborel {a..b})
  ↔
  (X ∈ borel-measurable M ∧ a < b ∧ (∀ t∈{a .. b}. P(x in M. X x ≤ t) = (t -
  a) / (b - a)))
  using
    uniform-distributed-bounds[of X a b]

```

uniform-distributed-measure[of X a b]
distributed-measurable[of M $\text{lborel } X$]
by (*auto intro!*: *uniform-distrI-borel-atLeastAtMost simp del: content-real-if*)

lemma (*in prob-space*) *uniform-distributed-expectation*:

fixes a b :: *real*
assumes D : *distributed* M $\text{lborel } X$ (λx . *indicator* $\{a .. b\} x / \text{measure lborel } \{a .. b\}$)
shows *expectation* $X = (a + b) / 2$
proof (*subst distributed-integral[OF D, of λx . x , symmetric]*)
have $a < b$
using *uniform-distributed-bounds[OF D]* .

have ($\int x$. *indicator* $\{a .. b\} x / \text{measure lborel } \{a .. b\} * x \partial \text{lborel}$) =
 $(\int x$. $(x / \text{measure lborel } \{a .. b\}) * \text{indicator } \{a .. b\} x \partial \text{lborel}$)
by (*intro Bochner-Integration.integral-cong*) *auto*
also have ($\int x$. $(x / \text{measure lborel } \{a .. b\}) * \text{indicator } \{a .. b\} x \partial \text{lborel}$) =
 $(a + b) / 2$
proof (*subst integral-FTC-Icc-real*)

fix x
show *DERIV* (λx . $x^2 / (2 * \text{measure lborel } \{a..b\})$) x :> $x / \text{measure lborel } \{a..b\}$
using *uniform-distributed-params[OF D]*
by (*auto intro!*: *derivative-eq-intros simp del: content-real-if*)
show *isCont* (λx . $x / \text{Sigma-Algebra.measure lborel } \{a..b\}$) x
using *uniform-distributed-params[OF D]*
by (*auto intro!*: *isCont-divide*)
have *: $b^2 / (2 * \text{measure lborel } \{a..b\}) - a^2 / (2 * \text{measure lborel } \{a..b\}) =$
 $(b*b - a * a) / (2 * (b - a))$
using $\langle a < b \rangle$
by (*auto simp: measure-def power2-eq-square diff-divide-distrib[symmetric]*)
show $b^2 / (2 * \text{measure lborel } \{a..b\}) - a^2 / (2 * \text{measure lborel } \{a..b\}) = (a$
 $+ b) / 2$
using $\langle a < b \rangle$
unfolding * *square-diff-square-factored* **by** (*auto simp: field-simps*)
qed (*insert* $\langle a < b \rangle$, *simp*)
finally show ($\int x$. *indicator* $\{a .. b\} x / \text{measure lborel } \{a .. b\} * x \partial \text{lborel}$) =
 $(a + b) / 2$.
qed (*auto simp: measure-nonneg*)

lemma (*in prob-space*) *uniform-distributed-variance*:

fixes a b :: *real*
assumes D : *distributed* M $\text{lborel } X$ (λx . *indicator* $\{a .. b\} x / \text{measure lborel } \{a .. b\}$)
shows *variance* $X = (b - a)^2 / 12$
proof (*subst distributed-variance*)
have [*arith*]: $a < b$ **using** *uniform-distributed-bounds[OF D]* .
let $?\mu = \text{expectation } X$ **let** $?D = \lambda x$. *indicator* $\{a..b\} (x + ?\mu) / \text{measure lborel } \{a..b\}$

have $(\int x. x^2 * (?D x) \partial\text{lborel}) = (\int x. x^2 * (\text{indicator } \{a - ?\mu .. b - ?\mu\} x)$
 $/ \text{measure } \text{lborel } \{a .. b\} \partial\text{lborel})$
by $(\text{intro } \text{Bochner-Integration.integral-cong}) (\text{auto split: split-indicator})$
also have $\dots = (b - a)^2 / 12$
by $(\text{simp add: integral-power uniform-distributed-expectation}[OF D])$
 $(\text{simp add: eval-nat-numeral field-simps})$
finally show $(\int x. x^2 * ?D x \partial\text{lborel}) = (b - a)^2 / 12 .$
qed $(\text{auto intro: } D \text{ simp del: content-real-if})$

15.4 Normal distribution

definition $\text{normal-density} :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**
 $\text{normal-density } \mu \sigma x = 1 / \text{sqrt } (2 * \text{pi} * \sigma^2) * \text{exp } (-(x - \mu)^2 / (2 * \sigma^2))$

abbreviation $\text{std-normal-density} :: \text{real} \Rightarrow \text{real}$ **where**
 $\text{std-normal-density} \equiv \text{normal-density } 0 \ 1$

lemma $\text{std-normal-density-def}: \text{std-normal-density } x = (1 / \text{sqrt } (2 * \text{pi})) * \text{exp}$
 $(- x^2 / 2)$
unfolding $\text{normal-density-def}$ **by** simp

lemma $\text{normal-density-nonneg}[simp]: 0 \leq \text{normal-density } \mu \sigma x$
by $(\text{auto simp: normal-density-def})$

lemma $\text{normal-density-pos}: 0 < \sigma \implies 0 < \text{normal-density } \mu \sigma x$
by $(\text{auto simp: normal-density-def})$

lemma $\text{borel-measurable-normal-density}[measurable]: \text{normal-density } \mu \sigma \in \text{borel-measurable}$
 borel
by $(\text{auto simp: normal-density-def}[abs-def])$

lemma $\text{gaussian-moment-0}: \text{has-bochner-integral } \text{lborel } (\lambda x. \text{indicator } \{0..\} x *_{\mathbb{R}} \text{exp } (- x^2)) (\text{sqrt } \text{pi} / 2)$

proof –

let $?pI = \lambda f. (\int^+ s. f (s::\text{real}) * \text{indicator } \{0..\} s \partial\text{lborel})$
let $?gauss = \lambda x. \text{exp } (- x^2)$

let $?I = \text{indicator } \{0<..\} :: \text{real} \Rightarrow \text{real}$
let $?ff = \lambda x s. x * \text{exp } (- x^2 * (1 + s^2)) :: \text{real}$

have $*: ?pI ?gauss = (\int^+ x. ?gauss x * ?I x \partial\text{lborel})$
by $(\text{intro nn-integral-cong-AE AE-I}[\text{where } N=\{0\}]) (\text{auto split: split-indicator})$

have $?pI ?gauss * ?pI ?gauss = (\int^+ x. \int^+ s. ?gauss x * ?gauss s * ?I s * ?I x$
 $\partial\text{lborel } \partial\text{lborel})$

by $(\text{auto simp: nn-integral-cmult}[symmetric] \text{nn-integral-multc}[symmetric] * \text{ennreal-mult}[symmetric])$
 $\text{intro!: nn-integral-cong split: split-indicator})$

also have $\dots = (\int^+ x. \int^+ s. ?ff x s * ?I s * ?I x \partial\text{lborel } \partial\text{lborel})$

```

proof (rule nn-integral-cong, cases)
  fix x :: real assume x ≠ 0
  then show ( $\int^{+s} ?gauss\ x * ?gauss\ s * ?I\ s * ?I\ x\ \partial lborel$ ) = ( $\int^{+s} ?ff\ x\ s$ 
*  $?I\ s * ?I\ x\ \partial lborel$ )
    by (subst nn-integral-real-affine[where t=0 and c=x])
      (auto simp: mult-exp-exp nn-integral-cmult[symmetric] field-simps zero-less-mult-iff
ennreal-mult[symmetric]
intro!: nn-integral-cong split: split-indicator)
qed simp
also have ... =  $\int^{+s} \int^{+x} ?ff\ x\ s * ?I\ s * ?I\ x\ \partial lborel\ \partial lborel$ 
  by (rule lborel-pair.Fubini'[symmetric]) auto
also have ... =  $?pI\ (\lambda s. ?pI\ (\lambda x. ?ff\ x\ s))$ 
  by (rule nn-integral-cong-AE)
    (auto intro!: nn-integral-cong-AE AE-I[where N={0}] split: split-indicator-asm)
also have ... =  $?pI\ (\lambda s. ennreal\ (1 / (2 * (1 + s^2))))$ 
proof (intro nn-integral-cong ennreal-mult-right-cong)
  fix s :: real show  $?pI\ (\lambda x. ?ff\ x\ s) = ennreal\ (1 / (2 * (1 + s^2)))$ 
  proof (subst nn-integral-FTC-atLeast)
    have (( $\lambda a. - (exp\ (- (a^2 * (1 + s^2))) / (2 + 2 * s^2))$ )  $\longrightarrow$  ( $- (0 / (2 +$ 
 $2 * s^2))$ )) at-top
    apply (intro tendsto-intros filterlim-compose[OF exp-at-bot] filterlim-compose[OF
filterlim-uminus-at-bot-at-top])
      apply (subst mult.commute)
      apply (auto intro!: filterlim-tendsto-pos-mult-at-top
filterlim-at-top-mult-at-top[OF filterlim-ident filterlim-ident]
simp: add-pos-nonneg power2-eq-square add-nonneg-eq-0-iff)
    done
  then show (( $\lambda a. - (exp\ (- a^2 - s^2 * a^2) / (2 + 2 * s^2))$ )  $\longrightarrow$  0) at-top
  by (simp add: field-simps)
qed (auto intro!: derivative-eq-intros simp: field-simps add-nonneg-eq-0-iff)
qed
also have ... = ennreal (pi / 4)
proof (subst nn-integral-FTC-atLeast)
  show (( $\lambda a. arctan\ a / 2$ )  $\longrightarrow$  (pi / 2) / 2) at-top
  by (intro tendsto-intros) (simp-all add: tendsto-arctan-at-top)
qed (auto intro!: derivative-eq-intros simp: add-nonneg-eq-0-iff field-simps power2-eq-square)
finally have  $?pI\ ?gauss^2 = pi / 4$ 
  by (simp add: power2-eq-square)
then have  $?pI\ ?gauss = sqrt\ (pi / 4)$ 
  using power-eq-iff-eq-base[of 2 enn2real (?pI ?gauss) sqrt (pi / 4)]
  by (cases ?pI ?gauss) (auto simp: power2-eq-square ennreal-mult[symmetric]
ennreal-top-mult)
also have  $?pI\ ?gauss = (\int^{+x} indicator\ \{0..\} x *_{\mathbb{R}} exp\ (- x^2)\ \partial lborel)$ 
  by (intro nn-integral-cong) (simp split: split-indicator)
also have  $sqrt\ (pi / 4) = sqrt\ pi / 2$ 
  by (simp add: real-sqrt-divide)
finally show ?thesis
  by (rule has-bochner-integral-nn-integral[rotated 3])
    auto

```

qed

lemma gaussian-moment-1:

has-bochner-integral lborel ($\lambda x::\text{real. indicator } \{0..\} x *_R (\exp (- x^2) * x)$) ($1 / 2$)

proof –

have ($\int^+ x. \text{indicator } \{0..\} x *_R (\exp (- x^2) * x) \partial \text{lborel}$) =
 $(\int^+ x. \text{ennreal } (x * \exp (- x^2)) * \text{indicator } \{0..\} x \partial \text{lborel})$
by (*intro nn-integral-cong*)

(*auto simp: ac-simps split: split-indicator*)

also have $\dots = \text{ennreal } (0 - (- \exp (- 0^2) / 2))$

proof (*rule nn-integral-FTC-atLeast*)

have ($(\lambda x::\text{real.} - \exp (- x^2) / 2) \longrightarrow - 0 / 2$) *at-top*

by (*intro tendsto-divide tendsto-minus filterlim-compose [OF exp-at-bot]*
filterlim-compose [OF filterlim-uminus-at-bot-at-top]
filterlim-pow-at-top filterlim-ident)

auto

then show ($(\lambda a::\text{real.} - \exp (- a^2) / 2) \longrightarrow 0$) *at-top*

by *simp*

qed (*auto intro!: derivative-eq-intros*)

also have $\dots = \text{ennreal } (1 / 2)$

by *simp*

finally show *?thesis*

by (*rule has-bochner-integral-nn-integral [rotated 3]*)
(auto split: split-indicator)

qed

lemma

fixes $k :: \text{nat}$

shows *gaussian-moment-even-pos:*

has-bochner-integral lborel ($\lambda x::\text{real. indicator } \{0..\} x *_R (\exp (-x^2) * x^{(2 * k)})$)

$((\text{sqrt } \pi / 2) * (\text{fact } (2 * k) / (2^{(2 * k)} * \text{fact } k)))$

(is ?even)

and *gaussian-moment-odd-pos:*

has-bochner-integral lborel ($\lambda x::\text{real. indicator } \{0..\} x *_R (\exp (-x^2) * x^{(2 * k + 1)})$) ($\text{fact } k / 2$)

(is ?odd)

proof –

let $?M = \lambda k x. \exp (- x^2) * x^k :: \text{real}$

{ fix $k I$ **assume** $Mk:$ *has-bochner-integral lborel* ($\lambda x. \text{indicator } \{0..\} x *_R ?M k x$) I

have $2 \neq (0::\text{real})$

by *linarith*

let $?f = \lambda b. \int x. \text{indicator } \{0..\} x *_R ?M (k + 2) x * \text{indicator } \{..b\} x \partial \text{lborel}$

have ($(\lambda b. (k + 1) / 2 * (\int x. \text{indicator } \{..b\} x *_R (\text{indicator } \{0..\} x *_R ?M k x) \partial \text{lborel}) - ?M (k + 1) b / 2) \longrightarrow$

$(k + 1) / 2 * (\int x. \text{indicator } \{0..\} x *_R ?M k x \partial \text{lborel}) - 0 / 2$) *at-top*

(is ?tendsto)
proof (intro tendsto-intros (2 ≠ 0) tendsto-integral-at-top sets-lborel Mk[THEN integrable.intros])
show (?M (k + 1) → 0) at-top
proof cases
assume even k
have ((λx. ((x²)^(k div 2 + 1) / exp (x²)) * (1 / x) :: real) → 0 * 0)
at-top
by (intro tendsto-intros tendsto-divide-0[OF tendsto-const] filterlim-compose[OF tendsto-power-div-exp-0]
filterlim-at-top-imp-at-infinity filterlim-ident filterlim-pow-at-top
filterlim-ident)
auto
also have (λx. ((x²)^(k div 2 + 1) / exp (x²)) * (1 / x) :: real) = ?M (k + 1)
using (even k) **by** (auto simp: fun-eq-iff exp-minus field-simps power2-eq-square power-mult elim: evenE)
finally show ?thesis **by simp**
next
assume odd k
have ((λx. ((x²)^{((k - 1) div 2 + 1)} / exp (x²)) :: real) → 0) at-top
by (intro filterlim-compose[OF tendsto-power-div-exp-0] filterlim-at-top-imp-at-infinity
filterlim-ident filterlim-pow-at-top)
auto
also have (λx. ((x²)^{((k - 1) div 2 + 1)} / exp (x²)) :: real) = ?M (k + 1)
using (odd k) **by** (auto simp: fun-eq-iff exp-minus field-simps power2-eq-square power-mult elim: oddE)
finally show ?thesis **by simp**
qed
qed
also have ?tendsto ↔ ((?f → (k + 1) / 2 * (∫ x. indicator {0..} x *_R ?M k x ∂lborel) - 0 / 2) at-top)
proof (intro filterlim-cong refl eventually-at-top-linorder[THEN iffD2] exI[of - 0] allI impI)
fix b :: real **assume** b: 0 ≤ b
have Suc k * (∫ x. indicator {0..b} x *_R ?M k x ∂lborel) = (∫ x. indicator {0..b} x *_R (exp (- x²) * ((Suc k) * x^k)) ∂lborel)
unfolding integral-mult-right-zero[symmetric] **by** (intro Bochner-Integration.integral-cong) auto
also have ... = exp (- b²) * b^(Suc k) - exp (- 0²) * 0^(Suc k) -
(∫ x. indicator {0..b} x *_R (- 2 * x * exp (- x²) * x^(Suc k)) ∂lborel)
by (rule integral-by-parts')
(auto intro!: derivative-eq-intros b
simp: diff-Suc of-nat-Suc field-simps split: nat.split)
also have (∫ x. indicator {0..b} x *_R (- 2 * x * exp (- x²) * x^(Suc k)) ∂lborel) =
(∫ x. indicator {0..b} x *_R (- 2 * (exp (- x²) * x<sup>(k + 2))) ∂lborel)
by (intro Bochner-Integration.integral-cong) auto</sup>

```

finally have Suc k * (∫ x. indicator {0..b} x *R ?M k x ∂lborel) =
  exp (- b2) * b ^ (Suc k) + 2 * (∫ x. indicator {0..b} x *R ?M (k + 2) x
∂lborel)
by (simp del: real-scaleR-def integral-mult-right add: integral-mult-right[symmetric])
then show (k + 1) / 2 * (∫ x. indicator {..b} x *R (indicator {0..} x *R
?M k x)∂lborel) - exp (- b2) * b ^ (k + 1) / 2 = ?f b
by (simp add: field-simps atLeastAtMost-def indicator-inter-arith)
qed
finally have int-M-at-top: ((?f → (k + 1) / 2 * (∫ x. indicator {0..} x *R
?M k x ∂lborel)) at-top)
by simp

have has-bochner-integral lborel (λx. indicator {0..} x *R ?M (k + 2) x) ((k
+ 1) / 2 * I)
proof (rule has-bochner-integral-monotone-convergence-at-top)
fix y :: real
have *: (λx. indicator {0..} x *R ?M (k + 2) x * indicator {..y} x :: real) =
  (λx. indicator {0..y} x *R ?M (k + 2) x)
by rule (simp split: split-indicator)
show integrable lborel (λx. indicator {0..} x *R (?M (k + 2) x) * indicator
{..y} x :: real)
unfolding * by (rule borel-integrable-compact) (auto intro!: continuous-intros)
show ((?f → (k + 1) / 2 * I) at-top)
using int-M-at-top has-bochner-integral-integral-eq[OF Mk] by simp
qed (auto split: split-indicator) }
note step = this

show ?even
proof (induct k)
case (Suc k)
note step[OF this]
also have (real (2 * k + 1) / 2 * (sqrt pi / 2 * ((fact (2 * k)) / ((2::real)^(2*k)
* fact k)))) =
  sqrt pi / 2 * ((fact (2 * Suc k)) / ((2::real)^(2 * Suc k) * fact (Suc k)))
apply (simp add: field-simps del: fact-Suc)
apply (simp add: of-nat-mult field-simps)
done
finally show ?case
by simp
qed (insert gaussian-moment-0, simp)

show ?odd
proof (induct k)
case (Suc k)
note step[OF this]
also have (real (2 * k + 1 + 1) / (2::real) * ((fact k) / 2)) = (fact (Suc k))
/ 2
by (simp add: field-simps of-nat-Suc divide-simps del: fact-Suc) (simp add:
field-simps)

```

finally show ?case
by simp
qed (insert gaussian-moment-1, simp)
qed

context

fixes $k :: \text{nat}$ **and** $\mu \sigma :: \text{real}$ **assumes** [arith]: $0 < \sigma$
begin

lemma normal-moment-even:

has-bochner-integral lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu) ^ (2 * k)$) (*fact*
 $(2 * k) / ((2 / \sigma^2) ^ k * \text{fact } k)$)

proof –

have eq: $\bigwedge x :: \text{real}. x^2 ^ k = (x ^ k)^2$
by (simp add: power-mult[symmetric] ac-simps)

have *has-bochner-integral lborel* ($\lambda x. \exp (-x^2) * x ^ (2 * k)$)
 $(\text{sqrt } \pi * (\text{fact } (2 * k) / (2 ^ (2 * k) * \text{fact } k)))$

using *has-bochner-integral-even-function* [OF gaussian-moment-even-pos [where
 $k=k$]] **by** simp

then have *has-bochner-integral lborel* ($\lambda x. (\exp (-x^2) * x ^ (2 * k)) * ((2 * \sigma^2) ^ k /$
 $\text{sqrt } (2 * \pi * \sigma^2))$)

$((\text{sqrt } \pi * (\text{fact } (2 * k) / (2 ^ (2 * k) * \text{fact } k))) * ((2 * \sigma^2) ^ k / \text{sqrt } (2 * \pi$
 $* \sigma^2)))$

by (rule *has-bochner-integral-mult-left*)

also have ($\lambda x. (\exp (-x^2) * x ^ (2 * k)) * ((2 * \sigma^2) ^ k / \text{sqrt } (2 * \pi * \sigma^2))$) =

$(\lambda x. \exp (- ((\text{sqrt } 2 * \sigma) * x)^2 / (2 * \sigma^2)) * ((\text{sqrt } 2 * \sigma) * x) ^ (2 * k) / \text{sqrt}$
 $(2 * \pi * \sigma^2))$

by (auto simp: fun-eq-iff field-simps real-sqrt-power[symmetric] real-sqrt-mult
real-sqrt-divide power-mult eq)

also have $((\text{sqrt } \pi * (\text{fact } (2 * k) / (2 ^ (2 * k) * \text{fact } k))) * ((2 * \sigma^2) ^ k / \text{sqrt}$
 $(2 * \pi * \sigma^2))) =$

$(\text{inverse } (\text{sqrt } 2 * \sigma) * ((\text{fact } (2 * k))) / ((2 / \sigma^2) ^ k * (\text{fact } k)))$

by (auto simp: fun-eq-iff power-mult field-simps real-sqrt-power[symmetric]
real-sqrt-mult

power2-eq-square)

finally show ?thesis

unfolding normal-density-def

by (subst lborel-has-bochner-integral-real-affine-iff [where $c=\text{sqrt } 2 * \sigma$ **and**
 $t=\mu$]) simp-all

qed

lemma normal-moment-abs-odd:

has-bochner-integral lborel ($\lambda x. \text{normal-density } \mu \sigma x * |x - \mu| ^ (2 * k + 1)$)
 $(2 ^ k * \sigma ^ (2 * k + 1) * \text{fact } k * \text{sqrt } (2 / \pi))$

proof –

have *has-bochner-integral lborel* ($\lambda x :: \text{real}. \text{indicator } \{0..\} x *_{\mathbb{R}} (\exp (-x^2) * |x| ^ (2$
 $* k + 1))$) (*fact* $k / 2$)

by (rule *has-bochner-integral-cong* [THEN iffD1, OF - - - gaussian-moment-odd-pos [of

k]]) *auto*
from *has-bochner-integral-even-function*[*OF this*]
have *has-bochner-integral lborel* ($\lambda x::\text{real. exp } (-x^2) * |x|^{(2 * k + 1)}$) (*fact k*)
by *simp*
then have *has-bochner-integral lborel* ($\lambda x. (\text{exp } (-x^2) * |x|^{(2 * k + 1)}) * (2^k * \sigma^{(2 * k + 1)} / \text{sqrt } (\text{pi} * \sigma^2))$)
(*fact k * (2^k * \sigma^{(2 * k + 1)} / \text{sqrt } (\text{pi} * \sigma^2))*)
by (*rule has-bochner-integral-mult-left*)
also have ($\lambda x. (\text{exp } (-x^2) * |x|^{(2 * k + 1)}) * (2^k * \sigma^{(2 * k + 1)} / \text{sqrt } (\text{pi} * \sigma^2))$) =
($\lambda x. \text{exp } (- ((\text{sqrt } 2 * \sigma) * x)^2 / (2 * \sigma^2)) * |\text{sqrt } 2 * \sigma * x|^{(2 * k + 1)}$)
/
 $\text{sqrt } (2 * \text{pi} * \sigma^2)$)
by (*simp add: field-simps abs-mult real-sqrt-power[symmetric] power-mult real-sqrt-mult*)
also have (*fact k * (2^k * \sigma^{(2 * k + 1)} / \text{sqrt } (\text{pi} * \sigma^2))*) =
(*inverse (sqrt 2) * inverse \sigma * (2^k * (\sigma * \sigma^{(2 * k)})) * (fact k) * sqrt (2*
/
 $\text{pi})$)
by (*auto simp: fun-eq-iff power-mult field-simps real-sqrt-power[symmetric]*
real-sqrt-divide
real-sqrt-mult)
finally show *?thesis*
unfolding *normal-density-def*
by (*subst lborel-has-bochner-integral-real-affine-iff* [**where** $c = \text{sqrt } 2 * \sigma$ **and**
 $t = \mu$])
simp-all
qed

lemma *normal-moment-odd:*

has-bochner-integral lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{(2 * k + 1)}$) 0
proof –
have *has-bochner-integral lborel* ($\lambda x. \text{exp } (-x^2) * x^{(2 * k + 1)}$::*real*) 0
using *gaussian-moment-odd-pos* **by** (*rule has-bochner-integral-odd-function*)
simp
then have *has-bochner-integral lborel* ($\lambda x. (\text{exp } (-x^2) * x^{(2 * k + 1)}) * (2^k * \sigma^{(2 * k)} / \text{sqrt } \text{pi})$)
($0 * (2^k * \sigma^{(2 * k)} / \text{sqrt } \text{pi})$)
by (*rule has-bochner-integral-mult-left*)
also have ($\lambda x. (\text{exp } (-x^2) * x^{(2 * k + 1)}) * (2^k * \sigma^{(2 * k)} / \text{sqrt } \text{pi})$) =
($\lambda x. \text{exp } (- ((\text{sqrt } 2 * \sigma * x)^2 / (2 * \sigma^2)) * (\text{sqrt } 2 * \sigma * x * (\text{sqrt } 2 * \sigma * x)^{(2 * k)} / \text{sqrt } (2 * \text{pi} * \sigma^2))$)
unfolding *real-sqrt-mult*
by (*simp add: field-simps abs-mult real-sqrt-power[symmetric] power-mult fun-eq-iff*)
finally show *?thesis*
unfolding *normal-density-def*
by (*subst lborel-has-bochner-integral-real-affine-iff* [**where** $c = \text{sqrt } 2 * \sigma$ **and**
 $t = \mu$]) *simp-all*
qed

lemma *integral-normal-moment-even:*

$\text{integral}^L \text{lborel } (\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{(2 * k)}) = \text{fact } (2 * k) / ((2 / \sigma^2)^k * \text{fact } k)$

using *normal-moment-even* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-normal-moment-abs-odd*:

$\text{integral}^L \text{lborel } (\lambda x. \text{normal-density } \mu \sigma x * |x - \mu|^{(2 * k + 1)}) = 2^k * \sigma^{2 * k + 1} * \text{fact } k * \text{sqrt } (2 / \text{pi})$

using *normal-moment-abs-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-normal-moment-odd*:

$\text{integral}^L \text{lborel } (\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{(2 * k + 1)}) = 0$

using *normal-moment-odd* **by** (*rule has-bochner-integral-integral-eq*)

end

context

fixes $\sigma :: \text{real}$

assumes $\sigma\text{-pos}[\text{arith}]: 0 < \sigma$

begin

lemma *normal-moment-nz-1*: *has-bochner-integral lborel* ($\lambda x. \text{normal-density } \mu \sigma x * x$) μ

proof –

note *normal-moment-even*[*OF* $\sigma\text{-pos}$, *of* μ 0]

note *normal-moment-odd*[*OF* $\sigma\text{-pos}$, *of* μ 0] *has-bochner-integral-mult-left*[*of* μ , *OF this*]

note *has-bochner-integral-add*[*OF this*]

then show *?thesis*

by (*simp add: power2-eq-square field-simps*)

qed

lemma *integral-normal-moment-nz-1*:

$\text{integral}^L \text{lborel } (\lambda x. \text{normal-density } \mu \sigma x * x) = \mu$

using *normal-moment-nz-1* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-normal-moment-nz-1*: *integrable lborel* ($\lambda x. \text{normal-density } \mu \sigma x * x$)

using *normal-moment-nz-1* **by** *rule*

lemma *integrable-normal-moment*: *integrable lborel* ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^k$)

proof *cases*

assume *even k* **then show** *?thesis*

using *integrable.intros*[*OF normal-moment-even*] **by** (*auto elim: evenE*)

next

assume *odd k* **then show** *?thesis*

using *integrable.intros*[*OF normal-moment-odd*] **by** (*auto elim: oddE*)

qed

lemma *integrable-normal-moment-abs*: *integrable lborel* ($\lambda x. \text{normal-density } \mu \sigma x * |x - \mu|^k$)
proof *cases*
 assume *even k then show ?thesis*
 using *integrable.intros[OF normal-moment-even]* **by** (*auto simp add: power-even-abs elim: evenE*)
next
 assume *odd k then show ?thesis*
 using *integrable.intros[OF normal-moment-abs-odd]* **by** (*auto elim: oddE*)
qed

lemma *integrable-normal-density[simp, intro]*: *integrable lborel* (*normal-density* $\mu \sigma$)
using *integrable-normal-moment[of μ 0]* **by** *simp*

lemma *integral-normal-density[simp]*: ($\int x. \text{normal-density } \mu \sigma x \partial \text{lborel}$) = 1
using *integral-normal-moment-even[of $\sigma \mu$ 0]* **by** *simp*

lemma *prob-space-normal-density*:
 prob-space (*density lborel* (*normal-density* $\mu \sigma$))
proof **qed** (*simp add: emeasure-density nn-integral-eq-integral normal-density-nonneg*)

end

context
 fixes *k :: nat*
begin

lemma *std-normal-moment-even*:
 has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k)}$) (*fact* ($2 * k$) / ($2^k * \text{fact } k$))
using *normal-moment-even[of 1 0 k]* **by** *simp*

lemma *std-normal-moment-abs-odd*:
 has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * |x|^{(2 * k + 1)}$) (*sqrt* ($2/\text{pi}$) * $2^k * \text{fact } k$)
using *normal-moment-abs-odd[of 1 0 k]* **by** (*simp add: ac-simps*)

lemma *std-normal-moment-odd*:
 has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k + 1)}$) 0
using *normal-moment-odd[of 1 0 k]* **by** *simp*

lemma *integral-std-normal-moment-even*:
 integral^L lborel ($\lambda x. \text{std-normal-density } x * x^{(2*k)}$) = *fact* ($2 * k$) / ($2^k * \text{fact } k$)
using *std-normal-moment-even* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-std-normal-moment-abs-odd:*

integral^L lborel ($\lambda x. \text{std-normal-density } x * |x|^{(2 * k + 1)} = \text{sqrt } (2 / \text{pi}) * 2^k * \text{fact } k$)
using *std-normal-moment-abs-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-std-normal-moment-odd:*

integral^L lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k + 1)} = 0$)
using *std-normal-moment-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-std-normal-moment-abs: integrable lborel* ($\lambda x. \text{std-normal-density } x * |x|^k$)

using *integrable-normal-moment-abs[of 1 0 k]* **by** *simp*

lemma *integrable-std-normal-moment: integrable lborel* ($\lambda x. \text{std-normal-density } x * x^k$)

using *integrable-normal-moment[of 1 0 k]* **by** *simp*

end

lemma (*in prob-space*) *normal-density-affine:*

assumes *X: distributed M lborel X* (*normal-density* $\mu \sigma$)

assumes [*simp, arith*]: $0 < \sigma \wedge \alpha \neq 0$

shows *distributed M lborel* ($\lambda x. \beta + \alpha * X x$) (*normal-density* $(\beta + \alpha * \mu) (|\alpha| * \sigma)$)

proof –

have *eq:* $\bigwedge x. |\alpha| * \text{normal-density } (\beta + \alpha * \mu) (|\alpha| * \sigma) (x * \alpha + \beta) = \text{normal-density } \mu \sigma x$

by (*simp add: normal-density-def real-sqrt-mult field-simps*)

(*simp add: power2-eq-square field-simps*)

show *?thesis*

by (*rule distributed-affineI[OF - « $\alpha \neq 0$ », where $t = \beta$]*)

(*simp-all add: eq X ennreal-mult'[symmetric]*)

qed

lemma (*in prob-space*) *normal-standard-normal-convert:*

assumes *pos-var[simp, arith]*: $0 < \sigma$

shows *distributed M lborel X* (*normal-density* $\mu \sigma$) = *distributed M lborel* ($\lambda x. (X x - \mu) / \sigma$) *std-normal-density*

proof *auto*

assume *distributed M lborel X* ($\lambda x. \text{ennreal } (\text{normal-density } \mu \sigma x)$)

then have *distributed M lborel* ($\lambda x. -\mu / \sigma + (1/\sigma) * X x$) ($\lambda x. \text{ennreal } (\text{normal-density } (-\mu / \sigma + (1/\sigma) * \mu) (|1/\sigma| * \sigma) x)$)

by(*rule normal-density-affine*) *auto*

then show *distributed M lborel* ($\lambda x. (X x - \mu) / \sigma$) ($\lambda x. \text{ennreal } (\text{std-normal-density } x)$)

by (*simp add: diff-divide-distrib[symmetric] field-simps*)

next

```

assume *: distributed M lborel ( $\lambda x. (X x - \mu) / \sigma$ ) ( $\lambda x. \text{ennreal (std-normal-density } x)$ )
have distributed M lborel ( $\lambda x. \mu + \sigma * ((X x - \mu) / \sigma)$ ) ( $\lambda x. \text{ennreal (normal-density } \mu \mid \sigma \mid x)$ )
using normal-density-affine[OF *, of  $\sigma \mu$ ] by simp
then show distributed M lborel X ( $\lambda x. \text{ennreal (normal-density } \mu \sigma \mid x)$ ) by simp
qed

```

lemma conv-normal-density-zero-mean:

```

assumes [simp, arith]:  $0 < \sigma \ 0 < \tau$ 
shows ( $\lambda x. \int^+ y. \text{ennreal (normal-density } 0 \ \sigma \ (x - y) * \text{normal-density } 0 \ \tau \ y)$ 
 $\partial \text{lborel}$ ) =
  normal-density 0 (sqrt ( $\sigma^2 + \tau^2$ )) (is ?LHS = ?RHS)

```

proof –

```

define  $\sigma' \ \tau'$  where  $\sigma' = \sigma^2$  and  $\tau' = \tau^2$ 
then have [simp, arith]:  $0 < \sigma' \ 0 < \tau'$ 
by simp-all
let ? $\sigma = \text{sqrt} ((\sigma' * \tau') / (\sigma' + \tau'))$ 
have sqrt: ( $\text{sqrt} (2 * \text{pi} * (\sigma' + \tau')) * \text{sqrt} (2 * \text{pi} * (\sigma' * \tau') / (\sigma' + \tau'))$ ) =
  ( $\text{sqrt} (2 * \text{pi} * \sigma') * \text{sqrt} (2 * \text{pi} * \tau')$ )
by (subst power-eq-iff-eq-base[symmetric, where  $n=2$ ])
  (simp-all add: real-sqrt-mult[symmetric] power2-eq-square)
have ?LHS =
  ( $\lambda x. \int^+ y. \text{ennreal}((\text{normal-density } 0 \ (\text{sqrt} (\sigma' + \tau')) \ x) * \text{normal-density } (\tau' * x / (\sigma' + \tau')) \ ?\sigma \ y) \ \partial \text{lborel}$ )
apply (intro ext nn-integral-cong)
apply (simp add: normal-density-def  $\sigma'$ -def[symmetric]  $\tau'$ -def[symmetric] sqrt
mult-exp-exp)
apply (simp add: divide-simps power2-eq-square)
apply (simp add: field-simps)
done

```

also have ... =

```

  ( $\lambda x. (\text{normal-density } 0 \ (\text{sqrt} (\sigma^2 + \tau^2)) \ x) * \int^+ y. \text{ennreal}(\text{normal-density } (\tau^2 * x / (\sigma^2 + \tau^2)) \ ?\sigma \ y) \ \partial \text{lborel}$ )
by (subst nn-integral-cmult[symmetric])
  (auto simp:  $\sigma'$ -def  $\tau'$ -def normal-density-def ennreal-mult'[symmetric])

```

also have ... = ($\lambda x. (\text{normal-density } 0 \ (\text{sqrt} (\sigma^2 + \tau^2)) \ x)$)

by (subst nn-integral-eq-integral) (auto simp: normal-density-nonneg)

finally show ?thesis **by** fast

qed

lemma conv-std-normal-density:

```

  ( $\lambda x. \int^+ y. \text{ennreal (std-normal-density } (x - y) * \text{std-normal-density } y) \ \partial \text{lborel}$ )
=
  (normal-density 0 (sqrt 2))
by (subst conv-normal-density-zero-mean) simp-all

```

lemma (in *prob-space*) *add-indep-normal*:
assumes *indep*: *indep-var* borel *X* borel *Y*
assumes *pos-var*[*arith*]: $0 < \sigma$ $0 < \tau$
assumes *normalX*[*simp*]: *distributed M lborel X* (*normal-density* μ σ)
assumes *normalY*[*simp*]: *distributed M lborel Y* (*normal-density* ν τ)
shows *distributed M lborel* ($\lambda x. X\ x + Y\ x$) (*normal-density* $(\mu + \nu)$ (*sqrt* ($\sigma^2 + \tau^2$)))
proof –
have *ind*[*simp*]: *indep-var* borel ($\lambda x. -\mu + X\ x$) borel ($\lambda x. -\nu + Y\ x$)
proof –
have *indep-var* borel ($(\lambda x. -\mu + x) \circ X$) borel ($(\lambda x. -\nu + x) \circ Y$)
by (*auto intro!*: *indep-var-compose* *assms*)
then show *?thesis* **by** (*simp add: o-def*)
qed

have *distributed M lborel* ($\lambda x. -\mu + 1 * X\ x$) (*normal-density* $(-\mu + 1 * \mu)$ (*|1| * sigma*))
by(*rule normal-density-affine*[*OF normalX pos-var*(1), *of 1 -mu*]) *simp*
then have *1*[*simp*]: *distributed M lborel* ($\lambda x. -\mu + X\ x$) (*normal-density* 0 σ)
by *simp*

have *distributed M lborel* ($\lambda x. -\nu + 1 * Y\ x$) (*normal-density* $(-\nu + 1 * \nu)$ (*|1| * tau*))
by(*rule normal-density-affine*[*OF normalY pos-var*(2), *of 1 -nu*]) *simp*
then have *2*[*simp*]: *distributed M lborel* ($\lambda x. -\nu + Y\ x$) (*normal-density* 0 τ)
by *simp*

have ***: *distributed M lborel* ($\lambda x. (-\mu + X\ x) + (-\nu + Y\ x)$) ($\lambda x. \text{ennreal}$ (*normal-density* 0 (*sqrt* ($\sigma^2 + \tau^2$)) *x*))
using *distributed-convolution*[*OF ind 1 2*] *conv-normal-density-zero-mean*[*OF pos-var*]
by (*simp add: ennreal-mult* [*symmetric*] *normal-density-nonneg*)

have *distributed M lborel* ($\lambda x. \mu + \nu + 1 * (-\mu + X\ x + (-\nu + Y\ x))$)
($\lambda x. \text{ennreal}$ (*normal-density* $(\mu + \nu + 1 * 0)$ (*|1| * sqrt* ($\sigma^2 + \tau^2$)) *x*))
by (*rule normal-density-affine*[*OF **, *of 1 mu + nu*]) (*auto simp: add-pos-pos*)

then show *?thesis* **by** *auto*
qed

lemma (in *prob-space*) *diff-indep-normal*:
assumes *indep*[*simp*]: *indep-var* borel *X* borel *Y*
assumes [*simp, arith*]: $0 < \sigma$ $0 < \tau$
assumes *normalX*[*simp*]: *distributed M lborel X* (*normal-density* μ σ)
assumes *normalY*[*simp*]: *distributed M lborel Y* (*normal-density* ν τ)
shows *distributed M lborel* ($\lambda x. X\ x - Y\ x$) (*normal-density* $(\mu - \nu)$ (*sqrt* ($\sigma^2 + \tau^2$))
 $+ \tau^2$)))
proof –

```

have distributed M lborel ( $\lambda x. 0 + - 1 * Y x$ ) ( $\lambda x. \text{ennreal (normal-density (0 + - 1 * \nu) (|- 1| * \tau) x)}$ )
by(rule normal-density-affine, auto)
then have [simp]:distributed M lborel ( $\lambda x. - Y x$ ) ( $\lambda x. \text{ennreal (normal-density (- \nu) \tau x)}$ ) by simp

```

```

have distributed M lborel ( $\lambda x. X x + (- Y x)$ ) ( $\text{normal-density } (\mu + - \nu) (\text{sqrt } (\sigma^2 + \tau^2))$ )
apply (rule add-indep-normal)
apply (rule indep-var-compose[unfolding comp-def, of borel - borel -  $\lambda x. x - \lambda x. - x$ ])
apply simp-all
done
then show ?thesis by simp
qed

```

lemma (in prob-space) sum-indep-normal:

```

assumes finite I I  $\neq \{\}$  indep-vars ( $\lambda i. \text{borel}$ ) X I
assumes  $\bigwedge i. i \in I \implies 0 < \sigma i$ 
assumes normal:  $\bigwedge i. i \in I \implies \text{distributed M lborel (X i) (normal-density } (\mu i) (\sigma i))$ 
shows distributed M lborel ( $\lambda x. \sum i \in I. X i x$ ) ( $\text{normal-density } (\sum i \in I. \mu i) (\text{sqrt } (\sum i \in I. (\sigma i)^2))$ )
using assms
proof (induct I rule: finite-ne-induct)
case (singleton i) then show ?case by (simp add : abs-of-pos)
next
case (insert i I)
then have 1: distributed M lborel ( $\lambda x. (X i x) + (\sum i \in I. X i x)$ )
( $\text{normal-density } (\mu i + \text{sum } \mu I) (\text{sqrt } ((\sigma i)^2 + (\text{sqrt } (\sum i \in I. (\sigma i)^2))^2))$ )
apply (intro add-indep-normal indep-vars-sum insert real-sqrt-gt-zero)
apply (auto intro: indep-vars-subset intro!: sum-pos)
apply fastforce
done
have 2: ( $\lambda x. (X i x) + (\sum i \in I. X i x)$ ) = ( $\lambda x. (\sum j \in \text{insert } i I. X j x)$ )
 $\mu i + \text{sum } \mu I = \text{sum } \mu (\text{insert } i I)$ 
using insert by auto

have 3: ( $\text{sqrt } ((\sigma i)^2 + (\text{sqrt } (\sum i \in I. (\sigma i)^2))^2)$ ) = ( $\text{sqrt } (\sum i \in \text{insert } i I. (\sigma i)^2)$ )
using insert by (simp add: sum-nonneg)

show ?case using 1 2 3 by simp
qed

```

lemma (in prob-space) standard-normal-distributed-expectation:

```

assumes D: distributed M lborel X std-normal-density
shows expectation X = 0

```

using *integral-std-normal-moment-odd*[of 0]
distributed-integral[OF D , of $\lambda x. x$, symmetric]
by (*auto simp*:)

lemma (**in** *prob-space*) *normal-distributed-expectation*:
assumes σ [*arith*]: $0 < \sigma$
assumes D : *distributed M lborel X* (*normal-density* μ σ)
shows *expectation* $X = \mu$
using *integral-normal-moment-nz-1*[OF σ , of μ] *distributed-integral*[OF D , of
 $\lambda x. x$, symmetric]
by (*auto simp*: *field-simps*)

lemma (**in** *prob-space*) *normal-distributed-variance*:
fixes $a b$:: *real*
assumes [*simp*, *arith*]: $0 < \sigma$
assumes D : *distributed M lborel X* (*normal-density* μ σ)
shows *variance* $X = \sigma^2$
using *integral-normal-moment-even*[of σ μ 1]
by (*subst distributed-integral*[OF D , symmetric])
(*simp-all add*: *eval-nat-numeral normal-distributed-expectation*[OF *assms*])

lemma (**in** *prob-space*) *standard-normal-distributed-variance*:
distributed M lborel X std-normal-density \implies *variance* $X = 1$
using *normal-distributed-variance*[of 1 X 0] **by** *simp*

lemma (**in** *information-space*) *entropy-normal-density*:
assumes [*arith*]: $0 < \sigma$
assumes D : *distributed M lborel X* (*normal-density* μ σ)
shows *entropy* b *lborel X* = $\log b (2 * \pi * \exp 1 * \sigma^2) / 2$
proof –
have *entropy* b *lborel X* = $-\left(\int x. \text{normal-density } \mu \sigma x * \log b (\text{normal-density } \mu \sigma x) \partial \text{lborel}\right)$
using D **by** (*rule entropy-distr*) *simp*
also have $\dots = -\left(\int x. \text{normal-density } \mu \sigma x * (-\ln (2 * \pi * \sigma^2) - (x - \mu)^2 / \sigma^2) / (2 * \ln b) \partial \text{lborel}\right)$
by (*intro arg-cong*[**where** $f = \text{uminus}$] *Bochner-Integration.integral-cong*)
(*auto simp*: *normal-density-def field-simps ln-mult log-def ln-div ln-sqrt*)
also have $\dots = -\left(\int x. -(\text{normal-density } \mu \sigma x * (\ln (2 * \pi * \sigma^2)) + (\text{normal-density } \mu \sigma x * (x - \mu)^2) / \sigma^2) / (2 * \ln b) \partial \text{lborel}\right)$
by (*intro arg-cong*[**where** $f = \text{uminus}$] *Bochner-Integration.integral-cong*) (*auto simp*: *divide-simps field-simps*)
also have $\dots = \left(\int x. \text{normal-density } \mu \sigma x * (\ln (2 * \pi * \sigma^2)) + (\text{normal-density } \mu \sigma x * (x - \mu)^2) / \sigma^2 \partial \text{lborel}\right) / (2 * \ln b)$
by (*simp del*: *minus-add-distrib*)
also have $\dots = (\ln (2 * \pi * \sigma^2) + 1) / (2 * \ln b)$
using *integral-normal-moment-even*[of σ μ 1] **by** (*simp add*: *integrable-normal-moment fact-numeral*)
also have $\dots = \log b (2 * \pi * \exp 1 * \sigma^2) / 2$
by (*simp add*: *log-def field-simps ln-mult*)

finally show ?thesis .
 qed
 end

16 Characteristic Functions

theory Characteristic-Functions
 imports Weak-Convergence Independent-Family Distributions
 begin

lemma mult-min-right: $a \geq 0 \implies (a :: \text{real}) * \min b c = \min (a * b) (a * c)$
 by (metis min.absorb-iff2 min-def mult-left-mono)

lemma sequentially-even-odd:
 assumes E : eventually $(\lambda n. P (2 * n))$ sequentially and O : eventually $(\lambda n. P (2 * n + 1))$ sequentially
 shows eventually P sequentially

proof –
 from E obtain $n\text{-}e$ where [intro]: $\bigwedge n. n \geq n\text{-}e \implies P (2 * n)$
 by (auto simp: eventually-sequentially)
 moreover
 from O obtain $n\text{-}o$ where [intro]: $\bigwedge n. n \geq n\text{-}o \implies P (Suc (2 * n))$
 by (auto simp: eventually-sequentially)
 show ?thesis
 unfolding eventually-sequentially
 proof (intro exI allI impI)
 fix n assume $\max (2 * n\text{-}e) (2 * n\text{-}o + 1) \leq n$ then show $P n$
 by (cases even n) (auto elim!: evenE oddE)
 qed
 qed

lemma limseq-even-odd:
 assumes $(\lambda n. f (2 * n)) \longrightarrow (l :: 'a :: \text{topological-space})$
 and $(\lambda n. f (2 * n + 1)) \longrightarrow l$
 shows $f \longrightarrow l$
 using assms by (auto simp: filterlim-iff intro: sequentially-even-odd)

16.1 Application of the FTC: integrating $e^i x$

abbreviation $iexp :: \text{real} \Rightarrow \text{complex}$ where
 $iexp \equiv (\lambda x. exp (i * \text{complex-of-real } x))$

lemma isCont-iexp [simp]: isCont $iexp x$
 by (intro continuous-intros)

lemma has-vector-derivative-iexp[derivative-intros]:
 $(iexp \text{ has-vector-derivative } i * iexp x)$ (at x within s)
 by (auto intro!: derivative-eq-intros simp: Re-exp Im-exp has-vector-derivative-complex-iff)

lemma *interval-integral-iexp*:
fixes $a\ b :: \text{real}$
shows $(\text{CLBINT } x=a..b. \text{iexp } x) = i * \text{iexp } a - i * \text{iexp } b$
by (*subst interval-integral-FTC-finite* [**where** $F = \lambda x. -i * \text{iexp } x$])
(auto intro!: derivative-eq-intros continuous-intros)

16.2 The Characteristic Function of a Real Measure.

definition

$\text{char} :: \text{real measure} \Rightarrow \text{real} \Rightarrow \text{complex}$

where

$\text{char } M\ t = \text{CLINT } x|M. \text{iexp } (t * x)$

lemma (*in real-distribution*) *char-zero*: $\text{char } M\ 0 = 1$
unfolding *char-def* **by** (*simp del: space-eq-univ add: prob-space*)

lemma (*in prob-space*) *integrable-iexp*:
assumes $f: f \in \text{borel-measurable } M \wedge x. \text{Im } (f\ x) = 0$
shows $\text{integrable } M (\lambda x. \text{exp } (i * (f\ x)))$
proof (*intro integrable-const-bound* [*of - 1*])
from f **have** $\wedge x. \text{of-real } (\text{Re } (f\ x)) = f\ x$
by (*simp add: complex-eq-iff*)
then show $\text{AE } x \text{ in } M. \text{cmod } (\text{exp } (i * f\ x)) \leq 1$
using *norm-exp-i-times*[*of Re (f x) for x*] **by** *simp*
qed (*insert f, simp*)

lemma (*in real-distribution*) *cmod-char-le-1*: $\text{norm } (\text{char } M\ t) \leq 1$

proof –

have $\text{norm } (\text{char } M\ t) \leq (\int x. \text{norm } (\text{iexp } (t * x)) \partial M)$

unfolding *char-def* **by** (*intro integral-norm-bound*)

also have $\dots \leq 1$

by (*simp del: of-real-mult*)

finally show *?thesis* .

qed

lemma (*in real-distribution*) *isCont-char*: $\text{isCont } (\text{char } M)\ t$

unfolding *continuous-at-sequentially*

proof safe

fix X **assume** $X: X \longrightarrow t$

show $(\text{char } M \circ X) \longrightarrow \text{char } M\ t$

unfolding *comp-def char-def*

by (*rule integral-dominated-convergence*[**where** $w=\lambda-. 1$]) (*auto intro!: tendsto-intros X*)

qed

lemma (*in real-distribution*) *char-measurable* [*measurable*]: $\text{char } M \in \text{borel-measurable borel}$

by (*auto intro!: borel-measurable-continuous-on1 continuous-at-imp-continuous-on*)

isCont-char)

16.3 Independence

lemma (in *prob-space*) *char-distr-add*:

fixes $X1\ X2 :: 'a \Rightarrow \text{real}$ **and** $t :: \text{real}$

assumes *indep-var borel X1 borel X2*

shows $\text{char} (\text{distr } M \text{ borel } (\lambda\omega. X1\ \omega + X2\ \omega))\ t =$

$\text{char} (\text{distr } M \text{ borel } X1)\ t * \text{char} (\text{distr } M \text{ borel } X2)\ t$

proof –

from *assms* **have** [*measurable*]: *random-variable borel X1* **by** (*elim indep-var-rv1*)

from *assms* **have** [*measurable*]: *random-variable borel X2* **by** (*elim indep-var-rv2*)

have $\text{char} (\text{distr } M \text{ borel } (\lambda\omega. X1\ \omega + X2\ \omega))\ t = (\text{CLINT } x|M. \text{iexp } (t * (X1\ x + X2\ x)))$

by (*simp add: char-def integral-distr*)

also have $\dots = (\text{CLINT } x|M. \text{iexp } (t * (X1\ x)) * \text{iexp } (t * (X2\ x)))$

by (*simp add: field-simps exp-add*)

also have $\dots = (\text{CLINT } x|M. \text{iexp } (t * (X1\ x))) * (\text{CLINT } x|M. \text{iexp } (t * (X2\ x)))$

by (*intro indep-var-lebesgue-integral indep-var-compose[unfolded comp-def, OF assms]*)

(*auto intro!: integrable-iexp*)

also have $\dots = \text{char} (\text{distr } M \text{ borel } X1)\ t * \text{char} (\text{distr } M \text{ borel } X2)\ t$

by (*simp add: char-def integral-distr*)

finally show *?thesis* .

qed

lemma (in *prob-space*) *char-distr-sum*:

indep-vars ($\lambda i. \text{borel } X\ A \Longrightarrow$

$\text{char} (\text{distr } M \text{ borel } (\lambda\omega. \sum_{i \in A} X\ i\ \omega))\ t = (\prod_{i \in A} \text{char} (\text{distr } M \text{ borel } (X\ i))\ t)$

proof (*induct A rule: infinite-finite-induct*)

case (*insert x F*) **with** *indep-vars-subset*[of $\lambda-. \text{borel } X\ \text{insert } x\ F\ F$] **show** *?case*

by (*auto simp add: char-distr-add indep-vars-sum*)

qed (*simp-all add: char-def integral-distr prob-space del: distr-const*)

16.4 Approximations to e^{ix}

Proofs from Billingsley, page 343.

lemma *CLBINT-I0c-power-mirror-iexp*:

fixes $x :: \text{real}$ **and** $n :: \text{nat}$

defines $f\ s\ m \equiv \text{complex-of-real } ((x - s) \wedge m)$

shows $(\text{CLBINT } s=0..x. f\ s\ n * \text{iexp } s) =$

$x \wedge \text{Suc } n / \text{Suc } n + (i / \text{Suc } n) * (\text{CLBINT } s=0..x. f\ s\ (\text{Suc } n) * \text{iexp } s)$

proof –

have *1*:

$((\lambda s. \text{complex-of-real } (-(x - s) \wedge (\text{Suc } n) / (\text{Suc } n))) * \text{iexp } s)$

has-vector-derivative complex-of-real(($x - s$) ^{n} * *iexp* s + (i * *iexp* s) * *complex-of-real*(-(($x - s$) ^{$(Suc\ n)$} / ($Suc\ n$))))

(*at* s *within* A) **for** $s\ A$

by (*intro derivative-eq-intros*) *auto*

let $?F = \lambda s.$ *complex-of-real*(-(($x - s$) ^{$(Suc\ n)$} / ($Suc\ n$))) * *iexp* s

have $x^{(Suc\ n) / (Suc\ n)} = (CLBINT\ s=0..x. (f\ s\ n * iexp\ s + (i * iexp\ s) * - (f\ s\ (Suc\ n) / (Suc\ n))))$ (**is** $?LHS = ?RHS$)

proof -

have $?RHS = (CLBINT\ s=0..x. (f\ s\ n * iexp\ s + (i * iexp\ s) * *complex-of-real*(-(($x - s$) ^{$(Suc\ n)$} / ($Suc\ n$))))$

by (*cases* $0 \leq x$) (*auto intro!*: *simp: f-def[abs-def]*)

also have $\dots = ?F\ x - ?F\ 0$

unfolding *zero-ereal-def* **using** 1

by (*intro interval-integral-FTC-finite*)

(*auto simp: f-def add-nonneg-eq-0-iff complex-eq-iff*

intro!: *continuous-at-imp-continuous-on continuous-intros*)

finally show $?thesis$

by *auto*

qed

show $?thesis$

unfolding $\langle ?LHS = ?RHS \rangle$ *f-def interval-lebesgue-integral-mult-right [symmetric]*

by (*subst interval-lebesgue-integral-add(2) [symmetric]*)

(*auto intro!*: *interval-integrable-isCont continuous-intros simp: zero-ereal-def complex-eq-iff*)

qed

lemma *iexp-eq1*:

fixes $x :: real$

defines $f\ s\ m \equiv *complex-of-real* ((x - s)^m)$

shows $iexp\ x =$

$(\sum k \leq n. (i * x)^k / (fact\ k)) + ((i^{(Suc\ n)} / (fact\ n)) * (CLBINT\ s=0..x. (f\ s\ n) * (iexp\ s)))$ (**is** $?P\ n$)

proof (*induction* n)

show $?P\ 0$

by (*auto simp add: field-simps interval-integral-iexp f-def zero-ereal-def*)

next

fix n **assume** $ih: ?P\ n$

have $** : \bigwedge a\ b :: real. a = -b \longleftrightarrow a + b = 0$

by *linarith*

have $*$: $of\text{-nat}\ n * of\text{-nat}\ (fact\ n) \neq - (of\text{-nat}\ (fact\ n) :: complex)$

unfolding *of-nat-mult[symmetric]*

by (*simp add: complex-eq-iff ** of-nat-add[symmetric] del: of-nat-mult of-nat-fact of-nat-add*)

show $?P\ (Suc\ n)$

unfolding *sum-atMost-Suc ih f-def CLBINT-I0c-power-mirror-iexp[of - n]*

by (*simp add: divide-simps add-eq-0-iff **) (*simp add: field-simps*)

qed

lemma *iexp-eq2*:
fixes $x :: \text{real}$
defines $f\ s\ m \equiv \text{complex-of-real } ((x - s) \wedge m)$
shows $iexp\ x = (\sum_{k \leq \text{Suc } n} (i * x) \wedge k / \text{fact } k) + i \wedge \text{Suc } n / \text{fact } n * (\text{CLBINT } s=0..x. f\ s\ n * (iexp\ s - 1))$
proof –
have *isCont-f*: $isCont\ (\lambda s. f\ s\ n)\ x$ **for** $n\ x$
by (*auto simp: f-def*)
let $?F = \lambda s. \text{complex-of-real } (-(x - s) \wedge (\text{Suc } n) / \text{real } (\text{Suc } n))$
have *calc1*: $(\text{CLBINT } s=0..x. f\ s\ n * (iexp\ s - 1)) =$
 $(\text{CLBINT } s=0..x. f\ s\ n * iexp\ s) - (\text{CLBINT } s=0..x. f\ s\ n)$
unfolding *zero-ereal-def*
by (*subst interval-lebesgue-integral-diff(2) [symmetric]*)
(simp-all add: interval-integrable-isCont isCont-f field-simps)

have *calc2*: $(\text{CLBINT } s=0..x. f\ s\ n) = x \wedge \text{Suc } n / \text{Suc } n$
unfolding *zero-ereal-def*
proof (*subst interval-integral-FTC-finite [where a = 0 and b = x and f = $\lambda s. f\ s\ n$ and F = ?F]*)
show (*?F has-vector-derivative f y n*) (*at y within {min 0 x..max 0 x}*) **for** y
unfolding *f-def*
by (*intro has-vector-derivative-of-real*)
(auto intro!: derivative-eq-intros simp del: power-Suc simp add: add-nonneg-eq-0-iff)
qed (*auto intro: continuous-at-imp-continuous-on isCont-f*)

have *calc3*: $i \wedge (\text{Suc } (\text{Suc } n)) / (\text{fact } (\text{Suc } n)) = (i \wedge (\text{Suc } n) / (\text{fact } n)) * (i / (\text{Suc } n))$
by (*simp add: field-simps*)

show *?thesis*
unfolding *iexp-eq1 [where n = Suc n and x=x] calc1 calc2 calc3* **unfolding** *f-def*
by (*subst CLBINT-I0c-power-mirror-iexp [where n = n] auto*)
qed

lemma *abs-LBINT-I0c-abs-power-diff*:
 $|\text{LBINT } s=0..x. |(x - s) \wedge n| = |x \wedge (\text{Suc } n) / (\text{Suc } n)|$
proof –
have $|\text{LBINT } s=0..x. |(x - s) \wedge n| = |\text{LBINT } s=0..x. (x - s) \wedge n|$
proof *cases*
assume $0 \leq x \vee \text{even } n$
then have $(\text{LBINT } s=0..x. |(x - s) \wedge n|) = \text{LBINT } s=0..x. (x - s) \wedge n$
by (*auto simp add: zero-ereal-def power-even-abs power-abs min-absorb1 max-absorb2*)
intro!: interval-integral-cong
then show *?thesis* **by** *simp*
next
assume $\neg (0 \leq x \vee \text{even } n)$
then have $(\text{LBINT } s=0..x. |(x - s) \wedge n|) = \text{LBINT } s=0..x. -((x - s) \wedge n)$

by (auto simp add: zero-ereal-def power-abs min-absorb1 max-absorb2
 ereal-min[symmetric] ereal-max[symmetric] power-minus-odd[symmetric]
 simp del: ereal-min ereal-max intro!: interval-integral-cong)
 also have ... = - (LBINT s=0..x. (x - s) ^ n)
 by (subst interval-lebesgue-integral-uminus, rule refl)
 finally show ?thesis by simp
 qed
 also have LBINT s=0..x. (x - s) ^ n = x ^ Suc n / Suc n
 proof -
 let ?F = $\lambda t. - ((x - t) ^ (Suc n) / Suc n)$
 have LBINT s=0..x. (x - s) ^ n = ?F x - ?F 0
 unfolding zero-ereal-def
 by (intro interval-integral-FTC-finite continuous-at-imp-continuous-on
 has-field-derivative-iff-has-vector-derivative[THEN iffD1])
 (auto simp del: power-Suc intro!: derivative-eq-intros simp add: add-nonneg-eq-0-iff)
 also have ... = x ^ (Suc n) / (Suc n) by simp
 finally show ?thesis .
 qed
 finally show ?thesis .
 qed

lemma *iexp-approx1*: $cmod (iexp x - (\sum k \leq n. (i * x) ^ k / fact k)) \leq |x| ^ (Suc n) / fact (Suc n)$

proof -
 have $iexp x - (\sum k \leq n. (i * x) ^ k / fact k) =$
 $((i ^ (Suc n)) / (fact n)) * (CLBINT s=0..x. (x - s) ^ n * (iexp s))$ (is ?t1
 = ?t2)
 by (subst iexp-eq1 [of - n], simp add: field-simps)
 then have $cmod (?t1) = cmod (?t2)$
 by simp
 also have ... = $(1 / of-nat (fact n)) * cmod (CLBINT s=0..x. (x - s) ^ n * (iexp s))$
 by (simp add: norm-mult norm-divide norm-power)
 also have ... $\leq (1 / of-nat (fact n)) * |LBINT s=0..x. cmod ((x - s) ^ n * (iexp s))|$
 by (intro mult-left-mono interval-integral-norm2)
 (auto simp: zero-ereal-def intro: interval-integrable-isCont)
 also have ... $\leq (1 / of-nat (fact n)) * |LBINT s=0..x. |(x - s) ^ n|$
 by (simp add: norm-mult del: of-real-diff of-real-power)
 also have ... $\leq (1 / of-nat (fact n)) * |x ^ (Suc n) / (Suc n)|$
 by (simp add: abs-LBINT-I0c-abs-power-diff)
 also have $1 / real-of-nat (fact n::nat) * |x ^ Suc n / real (Suc n)| =$
 $|x| ^ Suc n / fact (Suc n)$
 by (simp add: abs-mult power-abs)
 finally show ?thesis .
 qed

lemma *iexp-approx2*: $cmod (iexp x - (\sum k \leq n. (i * x) ^ k / fact k)) \leq 2 * |x| ^ n / fact n$

proof (*induction n*) — really cases

case (*Suc n*)
have *: $\bigwedge a b. \text{interval-lebesgue-integrable lborel } a b f \implies \text{interval-lebesgue-integrable lborel } a b g \implies$
 $|\text{LBINT } s=a..b. f s| \leq |\text{LBINT } s=a..b. g s|$
if $f: \bigwedge s. 0 \leq f s$ **and** $g: \bigwedge s. f s \leq g s$ **for** $f g :: - \Rightarrow \text{real}$
using *order-trans[OF f g]* **unfolding** *interval-lebesgue-integral-def interval-lebesgue-integrable-def*
by (*auto simp: integral-nonneg-AE[OF AE-I2] intro!: integral-mono mult-mono*)

have $iexp x - (\sum k \leq \text{Suc } n. (i * x) ^ k / \text{fact } k) =$
 $((i ^ (\text{Suc } n)) / (\text{fact } n)) * (\text{CLBINT } s=0..x. (x - s) ^ n * (iexp s - 1))$ (**is**
 $?t1 = ?t2$)
unfolding *iexp-eq2 [of x n]* **by** (*simp add: field-simps*)
then have $cmod (?t1) = cmod (?t2)$
by *simp*
also have $\dots = (1 / (\text{fact } n)) * cmod (\text{CLBINT } s=0..x. (x - s) ^ n * (iexp s - 1))$
by (*simp add: norm-mult norm-divide norm-power*)
also have $\dots \leq (1 / (\text{fact } n)) * |\text{LBINT } s=0..x. cmod ((x - s) ^ n * (iexp s - 1))|$
by (*intro mult-left-mono interval-integral-norm2*)
(auto intro!: interval-integrable-isCont simp: zero-ereal-def)
also have $\dots = (1 / (\text{fact } n)) * |\text{LBINT } s=0..x. \text{abs } ((x - s) ^ n) * cmod((iexp s - 1))|$
by (*simp add: norm-mult del: of-real-diff of-real-power*)
also have $\dots \leq (1 / (\text{fact } n)) * |\text{LBINT } s=0..x. \text{abs } ((x - s) ^ n) * 2|$
by (*intro mult-left-mono * order-trans [OF norm-triangle-ineq4]*)
(auto simp: mult-ac zero-ereal-def intro!: interval-integrable-isCont)
also have $\dots = (2 / (\text{fact } n)) * |x ^ (\text{Suc } n) / (\text{Suc } n)|$
by (*simp add: abs-LBINT-I0c-abs-power-diff abs-mult*)
also have $2 / \text{fact } n * |x ^ \text{Suc } n / \text{real } (\text{Suc } n)| = 2 * |x| ^ \text{Suc } n / (\text{fact } (\text{Suc } n))$
by (*simp add: abs-mult power-abs*)
finally show $?case .$
qed (*insert norm-triangle-ineq4 [of iexp x 1], simp*)

lemma (*in real-distribution*) *char-approx1*:

assumes *integrable-moments*: $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. x ^ k)$
shows $cmod (\text{char } M t - (\sum k \leq n. ((i * t) ^ k / \text{fact } k) * \text{expectation } (\lambda x. x ^ k)))$
 \leq
 $(2 * |t| ^ n / \text{fact } n) * \text{expectation } (\lambda x. |x| ^ n)$ (**is** $cmod (\text{char } M t - ?t1) \leq -$)

proof —

have *integ-iexp*: *integrable M (λx. iexp (t * x))*
by (*intro integrable-const-bound auto*)

define *c* **where** [*abs-def*]: $c k x = (i * t) ^ k / \text{fact } k * \text{complex-of-real } (x ^ k)$ **for**
 $k x$

have *integ-c*: $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. c k x)$
unfolding *c-def* **by** (*intro integrable-mult-right integrable-of-real integrable-moments*)

have $k \leq n \implies \text{expectation } (c \ k) = (i * t) \wedge k * (\text{expectation } (\lambda x. x \wedge k)) / \text{fact } k$ **for** k
unfolding *c-def integral-mult-right-zero integral-complex-of-real by simp*
then have $\text{norm } (\text{char } M \ t - ?t1) = \text{norm } (\text{char } M \ t - (\text{CLINT } x \mid M. (\sum k \leq n. c \ k \ x)))$
by (*simp add: integ-c*)
also have $\dots = \text{norm } ((\text{CLINT } x \mid M. \text{iexp } (t * x) - (\sum k \leq n. c \ k \ x)))$
unfolding *char-def by (subst Bochner-Integration.integral-diff[OF integ-iexp])*
(*auto intro!: integ-c*)
also have $\dots \leq \text{expectation } (\lambda x. \text{cmod } (\text{iexp } (t * x) - (\sum k \leq n. c \ k \ x)))$
by (*intro integral-norm-bound*)
also have $\dots \leq \text{expectation } (\lambda x. 2 * |t| \wedge n / \text{fact } n * |x| \wedge n)$
proof (*rule integral-mono*)
show *integrable* $M (\lambda x. \text{cmod } (\text{iexp } (t * x) - (\sum k \leq n. c \ k \ x)))$
by (*intro integrable-norm Bochner-Integration.integrable-diff integ-iexp Bochner-Integration.integrable-sum integ-c*) *simp*
show *integrable* $M (\lambda x. 2 * |t| \wedge n / \text{fact } n * |x| \wedge n)$
unfolding *power-abs[symmetric]*
by (*intro integrable-mult-right integrable-abs integrable-moments*) *simp*
show $\text{cmod } (\text{iexp } (t * x) - (\sum k \leq n. c \ k \ x)) \leq 2 * |t| \wedge n / \text{fact } n * |x| \wedge n$
for x
using *iexp-approx2[of t * x n]* **by** (*auto simp add: abs-mult field-simps c-def*)
qed
finally show *?thesis*
unfolding *integral-mult-right-zero* .
qed

lemma (*in real-distribution*) *char-approx2*:
assumes *integrable-moments*: $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. x \wedge k)$
shows $\text{cmod } (\text{char } M \ t - (\sum k \leq n. ((i * t) \wedge k / \text{fact } k) * \text{expectation } (\lambda x. x \wedge k)))$
 \leq
 $(|t| \wedge n / \text{fact } (\text{Suc } n)) * \text{expectation } (\lambda x. \min (2 * |x| \wedge n * \text{Suc } n) (|t| * |x| \wedge \text{Suc } n))$
(*is cmod (char M t - ?t1) ≤ -*)

proof –
have *integ-iexp*: *integrable* $M (\lambda x. \text{iexp } (t * x))$
by (*intro integrable-const-bound*) *auto*

define c **where** [*abs-def*]: $c \ k \ x = (i * t) \wedge k / \text{fact } k * \text{complex-of-real } (x \wedge k)$ **for** $k \ x$

have *integ-c*: $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. c \ k \ x)$
unfolding *c-def* **by** (*intro integrable-mult-right integrable-of-real integrable-moments*)

have $*$: $\min (2 * |t * x| \wedge n / \text{fact } n) (|t * x| \wedge \text{Suc } n / \text{fact } (\text{Suc } n)) =$
 $|t| \wedge n / \text{fact } (\text{Suc } n) * \min (2 * |x| \wedge n * \text{real } (\text{Suc } n)) (|t| * |x| \wedge (\text{Suc } n))$ **for** x
apply (*subst mult-min-right*)
apply *simp*
apply (*rule arg-cong2[where f=min]*)


```

apply (simp-all add: field-simps abs-mult del: fact-Suc)
apply (simp-all add: field-simps)
done

have  $k \leq n \implies \text{expectation } (c k) = (i * t) ^ k * (\text{expectation } (\lambda x. x ^ k)) / \text{fact } k$ 
for  $k$ 
  unfolding c-def integral-mult-right-zero integral-complex-of-real by simp
  then have  $\text{norm } (\text{char } M t - ?t1) = \text{norm } (\text{char } M t - (\text{CLINT } x \mid M. (\sum_{k \leq n} c k x)))$ 
  by (simp add: integ-c)
  also have  $\dots = \text{norm } ((\text{CLINT } x \mid M. \text{iexp } (t * x) - (\sum_{k \leq n} c k x)))$ 
  unfolding char-def by (subst Bochner-Integration.integral-diff[OF integ-iexp])
  (auto intro!: integ-c)
  also have  $\dots \leq \text{expectation } (\lambda x. \text{cmod } (\text{iexp } (t * x) - (\sum_{k \leq n} c k x)))$ 
  by (rule integral-norm-bound)
  also have  $\dots \leq \text{expectation } (\lambda x. \min (2 * |t * x| ^ n / \text{fact } n) (|t * x| ^ (\text{Suc } n)) / \text{fact } (\text{Suc } n))$ 
  (is -  $\leq \text{expectation } ?f$ )
  proof (rule integral-mono)
    show integrable  $M (\lambda x. \text{cmod } (\text{iexp } (t * x) - (\sum_{k \leq n} c k x)))$ 
    by (intro integrable-norm Bochner-Integration.integrable-diff integ-iexp Bochner-Integration.integrable-sum
  integ-c) simp
    show integrable  $M ?f$ 
    by (rule Bochner-Integration.integrable-bound[where  $f = \lambda x. 2 * |t * x| ^ n / \text{fact } n$ ])
    (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
    show  $\text{cmod } (\text{iexp } (t * x) - (\sum_{k \leq n} c k x)) \leq ?f x$  for  $x$ 
    using iexp-approx1[of  $t * x n$ ] iexp-approx2[of  $t * x n$ ]
    by (auto simp add: abs-mult field-simps c-def intro!: min.boundedI)
  qed
  also have  $\dots = (|t| ^ n / \text{fact } (\text{Suc } n)) * \text{expectation } (\lambda x. \min (2 * |x| ^ n * \text{Suc } n) (|t| * |x| ^ (\text{Suc } n)))$ 
  unfolding *
  proof (rule Bochner-Integration.integral-mult-right)
    show integrable  $M (\lambda x. \min (2 * |x| ^ n * \text{real } (\text{Suc } n)) (|t| * |x| ^ (\text{Suc } n)))$ 
    by (rule Bochner-Integration.integrable-bound[where  $f = \lambda x. 2 * |x| ^ n * \text{real } (\text{Suc } n)$ ])
    (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
  qed
  finally show ?thesis
  unfolding integral-mult-right-zero .
qed

lemma (in real-distribution) char-approx3:
fixes  $t$ 
assumes
  integrable-1: integrable  $M (\lambda x. x)$  and
  integral-1:  $\text{expectation } (\lambda x. x) = 0$  and
  integrable-2: integrable  $M (\lambda x. x ^ 2)$  and

```

```

    integral-2: variance ( $\lambda x. x$ ) =  $\sigma^2$ 
  shows cmod (char M t - (1 - t^2 *  $\sigma^2$  / 2)) ≤
    (t^2 / 6) * expectation ( $\lambda x. \min (6 * x^2) (abs t * (abs x)^3)$ )
  proof -
  note real-distribution.char-approx2 [of M 2 t, simplified]
  have [simp]: prob UNIV = 1 by (metis prob-space space-eq-univ)
  from integral-2 have [simp]: expectation ( $\lambda x. x * x$ ) =  $\sigma^2$ 
    by (simp add: integral-1 numeral-eq-Suc)
  have 1:  $k \leq 2 \implies$  integrable M ( $\lambda x. x^k$ ) for k
    using assms by (auto simp: eval-nat-numeral le-Suc-eq)
  note char-approx1
  note 2 = char-approx1 [of 2 t, OF 1, simplified]
  have cmod (char M t - ( $\sum_{k \leq 2}. (i * t)^k * (expectation (\lambda x. x^k)) / (fact k)$ )) ≤
    t^2 * expectation ( $\lambda x. \min (6 * x^2) (|t| * |x|^3)$ ) / fact (3::nat)
    using char-approx2 [of 2 t, OF 1] by simp
  also have ( $\sum_{k \leq 2}. (i * t)^k * expectation (\lambda x. x^k) / (fact k)$ ) = 1 - t^2
    *  $\sigma^2 / 2$ 
    by (simp add: complex-eq-iff numeral-eq-Suc integral-1 Re-divide Im-divide)
  also have fact 3 = 6 by (simp add: eval-nat-numeral)
  also have t^2 * expectation ( $\lambda x. \min (6 * x^2) (|t| * |x|^3)$ ) / 6 =
    t^2 / 6 * expectation ( $\lambda x. \min (6 * x^2) (|t| * |x|^3)$ ) by (simp add: field-simps)
  finally show ?thesis .
qed

```

This is a more familiar textbook formulation in terms of random variables, but we will use the previous version for the CLT.

```

lemma (in prob-space) char-approx3':
  fixes  $\mu ::$  real measure and X
  assumes rv-X [simp]: random-variable borel X
    and [simp]: integrable M X integrable M ( $\lambda x. (X x)^2$ ) expectation X = 0
    and var-X: variance X =  $\sigma^2$ 
    and  $\mu$ -def:  $\mu =$  distr M borel X
  shows cmod (char  $\mu$  t - (1 - t^2 *  $\sigma^2$  / 2)) ≤
    (t^2 / 6) * expectation ( $\lambda x. \min (6 * (X x)^2) (|t| * |X x|^3)$ )
  using var-X unfolding  $\mu$ -def
  apply (subst integral-distr [symmetric, OF rv-X], simp)
  apply (intro real-distribution.char-approx3)
  apply (auto simp add: integrable-distr-eq integral-distr)
  done

```

this is the formulation in the book – in terms of a random variable *with* the distribution, rather the distribution itself. I don't know which is more useful, though in principal we can go back and forth between them.

```

lemma (in prob-space) char-approx1':
  fixes  $\mu ::$  real measure and X
  assumes integrable-moments :  $\bigwedge k. k \leq n \implies$  integrable M ( $\lambda x. X x^k$ )
    and rv-X[measurable]: random-variable borel X
    and  $\mu$ -distr : distr M borel X =  $\mu$ 

```

```

shows cmod (char  $\mu$  t - ( $\sum k \leq n. ((i * t) ^k / \text{fact } k) * \text{expectation } (\lambda x. (X$ 
x) ^k)))  $\leq$ 
  (2 * |t| ^n / fact n) * expectation ( $\lambda x. |X x| ^n$ )
unfolding  $\mu$ -distr[symmetric]
apply (subst (1 2) integral-distr [symmetric, OF rv-X], simp, simp)
apply (intro real-distribution.char-approx1[of distr M borel X n t] real-distribution-distr
rv-X)
apply (auto simp: integrable-distr-eq integrable-moments)
done

```

16.5 Calculation of the Characteristic Function of the Standard Distribution

abbreviation

std-normal-distribution \equiv density lborel std-normal-density

lemma *real-dist-normal-dist: real-distribution std-normal-distribution*

using prob-space-normal-density **by** (auto simp: real-distribution-def real-distribution-axioms-def)

lemma *std-normal-distribution-even-moments:*

fixes $k :: \text{nat}$

shows (LINT $x | \text{std-normal-distribution}. x^{(2 * k)} = \text{fact } (2 * k) / (2^k * \text{fact } k)$)

and integrable std-normal-distribution ($\lambda x. x^{(2 * k)}$)

using integral-std-normal-moment-even[of k]

by (subst integral-density)

(auto simp: normal-density-nonneg integrable-density

intro: integrable.intros std-normal-moment-even)

lemma *integrable-std-normal-distribution-moment: integrable std-normal-distribution*

($\lambda x. x^k$)

by (auto simp: normal-density-nonneg integrable-std-normal-moment integrable-density)

lemma *integral-std-normal-distribution-moment-odd:*

odd $k \implies \text{integral}^L \text{std-normal-distribution } (\lambda x. x^k) = 0$

using integral-std-normal-moment-odd[of $(k - 1) \text{ div } 2$]

by (auto simp: integral-density normal-density-nonneg elim: oddE)

lemma *std-normal-distribution-even-moments-abs:*

fixes $k :: \text{nat}$

shows (LINT $x | \text{std-normal-distribution}. |x|^{(2 * k)} = \text{fact } (2 * k) / (2^k * \text{fact } k)$)

using integral-std-normal-moment-even[of k]

by (subst integral-density) (auto simp: normal-density-nonneg power-even-abs)

lemma *std-normal-distribution-odd-moments-abs:*

fixes $k :: \text{nat}$

shows (LINT $x | \text{std-normal-distribution}. |x|^{(2 * k + 1)} = \text{sqrt } (2 / \text{pi}) * 2^k$)

```

k * fact k
  using integral-std-normal-moment-abs-odd[of k]
  by (subst integral-density) (auto simp: normal-density-nonneg)

theorem char-std-normal-distribution:
  char std-normal-distribution = ( $\lambda t$ . complex-of-real (exp (- (t2) / 2)))
proof (intro ext LIMSEQ-unique)
  fix t :: real
  let ?f' =  $\lambda k$ . (i * t)k / fact k * (LINT x | std-normal-distribution. xk)
  let ?f =  $\lambda n$ . ( $\sum k \leq n$ . ?f' k)
  show ?f  $\longrightarrow$  exp (- (t2) / 2)
  proof (rule limseq-even-odd)
    have (i * complex-of-real t)(2 * a) / (2a * fact a) = - ((complex-of-real t)2 / 2)a / fact a for a
      by (subst power-mult) (simp add: field-simps uminus-power-if power-mult)
    then have *: ?f (2 * n) = complex-of-real ( $\sum k < \text{Suc } n$ . (1 / fact k) * (- (t2) / 2)k) for n :: nat
      unfolding of-real-sum
      by (intro sum.reindex-bij-witness-not-neutral[symmetric, where
        i= $\lambda n$ . n div 2 and j= $\lambda n$ . 2 * n and T'={i. i  $\leq$  2 * n  $\wedge$  odd i} and
        S'={}])
      (auto simp: integral-std-normal-distribution-moment-odd std-normal-distribution-even-moments)
    show ( $\lambda n$ . ?f (2 * n))  $\longrightarrow$  exp (- (t2) / 2)
      unfolding * using exp-converges[where 'a=real]
      by (intro tendsto-of-real LIMSEQ-Suc) (auto simp: inverse-eq-divide sums-def
        [symmetric])
    have **: ?f (2 * n + 1) = ?f (2 * n) for n
      proof -
        have ?f (2 * n + 1) = ?f (2 * n) + ?f' (2 * n + 1)
          by simp
        also have ?f' (2 * n + 1) = 0
          by (subst integral-std-normal-distribution-moment-odd) simp-all
        finally show ?f (2 * n + 1) = ?f (2 * n)
          by simp
      qed
    show ( $\lambda n$ . ?f (2 * n + 1))  $\longrightarrow$  exp (- (t2) / 2)
      unfolding ** by fact
  qed

have **: ( $\lambda n$ . xn / fact n)  $\longrightarrow$  0 for x :: real
  using summable-LIMSEQ-zero [OF summable-exp] by (auto simp add: inverse-eq-divide)

let ?F =  $\lambda n$ . 2 * |t|n / fact n * (LINT x | std-normal-distribution. |x|n)

show ?f  $\longrightarrow$  char std-normal-distribution t
proof (rule metric-tendsto-imp-tendsto[OF limseq-even-odd])
  show ( $\lambda n$ . ?F (2 * n))  $\longrightarrow$  0
  proof (rule Lim-transform-eventually)
    show  $\forall F$  n in sequentially. 2 * ((t2 / 2)n / fact n) = ?F (2 * n)

```

unfolding *std-normal-distribution-even-moments-abs* **by** (*simp add: power-mult power-divide*)

qed (*intro tendsto-mult-right-zero ***)

have *: $?F (2 * n + 1) = (2 * |t| * \text{sqrt } (2 / \text{pi})) * ((2 * t^2)^n * \text{fact } n / \text{fact } (2 * n + 1))$ **for** n

unfolding *std-normal-distribution-odd-moments-abs*

by (*simp add: field-simps power-mult[symmetric] power-even-abs*)

have *norm* $((2 * t^2)^n * \text{fact } n / \text{fact } (2 * n + 1)) \leq (2 * t^2)^n / \text{fact } n$ **for** n

using *mult-mono[OF - square-fact-le-2-fact, of 1 1 + 2 * real n n]*

by (*auto simp add: divide-simps intro!: mult-left-mono*)

then show $(\lambda n. ?F (2 * n + 1)) \longrightarrow 0$

unfolding * **by** (*intro tendsto-mult-right-zero Lim-null-comparison [OF - ** [of 2 * t^2]]*) *auto*

show $\forall_F n$ *in sequentially. dist (?f n) (char std-normal-distribution t) \leq dist (?F n) 0*

using *real-distribution.char-approx1[OF real-dist-normal-dist integrable-std-normal-distribution-moment]*

by (*auto simp: dist-norm integral-nonneg-AE norm-minus-commute*)

qed

qed

end

17 Helly’s selection theorem

The set of bounded, monotone, right continuous functions is sequentially compact

theory *Helly-Selection*

imports *HOL-Library.Diagonal-Subsequence Weak-Convergence*

begin

lemma *minus-one-less: $x - 1 < (x::real)$*

by *simp*

theorem *Helly-selection:*

fixes $f :: \text{nat} \Rightarrow \text{real} \Rightarrow \text{real}$

assumes *rcont: $\bigwedge n x. \text{continuous } (\text{at-right } x) (f n)$*

assumes *mono: $\bigwedge n. \text{mono } (f n)$*

assumes *bdd: $\bigwedge n x. |f n x| \leq M$*

shows $\exists s. \text{strict-mono } (s::\text{nat} \Rightarrow \text{nat}) \wedge (\exists F. (\forall x. \text{continuous } (\text{at-right } x) F) \wedge \text{mono } F \wedge (\forall x. |F x| \leq M) \wedge$

$(\forall x. \text{continuous } (\text{at } x) F \longrightarrow (\lambda n. f (s n) x) \longrightarrow F x))$

proof –

obtain $m :: \text{real} \Rightarrow \text{nat}$ **where** *bij-betw m \mathbb{Q} UNIV*

using *countable-rat Rats-infinite* **by** (*erule countableE-infinite*)

then obtain $r :: \text{nat} \Rightarrow \text{real}$ **where** *bij: bij-betw r UNIV \mathbb{Q}*

```

using bij-betw-inv by blast

have dense-r:  $\bigwedge x y. x < y \implies \exists n. x < r n \wedge r n < y$ 
by (metis Rats-dense-in-real bij f-the-inv-into-f bij-betw-def)

let ?P =  $\lambda n. \lambda s. \text{convergent } (\lambda k. f (s k) (r n))$ 
interpret nat: subseqs ?P
proof (unfold convergent-def, unfold subseqs-def, auto)
  fix n :: nat and s :: nat  $\implies$  nat assume s: strict-mono s
  have bounded  $\{-M..M\}$ 
    using bounded-closed-interval by auto
  moreover have  $\bigwedge k. f (s k) (r n) \in \{-M..M\}$ 
    using bdd by (simp add: abs-le-iff minus-le-iff)
  ultimately have  $\exists l s'. \text{strict-mono } s' \wedge ((\lambda k. f (s k) (r n)) \circ s') \longrightarrow l$ 
    using compact-Icc compact-imp-seq-compact seq-compactE by metis
  thus  $\exists s'. \text{strict-mono } (s'::\text{nat}\implies\text{nat}) \wedge (\exists l. (\lambda k. f (s (s' k)) (r n)) \longrightarrow l)$ 
    by (auto simp: comp-def)
qed
define d where d = nat.diagseq
have subseq: strict-mono d
  unfolding d-def using nat.subseq-diagseq by auto
have rat-cnv: ?P n d for n
proof –
  have Pn-seqseq: ?P n (nat.seqseq (Suc n))
    by (rule nat.seqseq-holds)
  have 1:  $(\lambda k. f ((\text{nat.seqseq } (Suc n) \circ (\lambda k. \text{nat.fold-reduce } (Suc n) k (Suc n + k))) k) (r n)) = (\lambda k. f (\text{nat.seqseq } (Suc n) k) (r n)) \circ (\lambda k. \text{nat.fold-reduce } (Suc n) k (Suc n + k))$ 
    by auto
  have 2: ?P n (d \circ (op + (Suc n)))
    unfolding d-def nat.diagseq-seqseq 1
    by (intro convergent-subseq-convergent Pn-seqseq nat.subseq-diagonal-rest)
  then obtain L where 3:  $(\lambda na. f (d (na + Suc n)) (r n)) \longrightarrow L$ 
    by (auto simp: add.commute dest: convergentD)
  then have  $(\lambda k. f (d k) (r n)) \longrightarrow L$ 
    by (rule LIMSEQ-offset)
  then show ?thesis
    by (auto simp: convergent-def)
qed
let ?f =  $\lambda n. \lambda k. f (d k) (r n)$ 
have lim-f: ?f n  $\longrightarrow$  lim (?f n) for n
  using rat-cnv convergent-LIMSEQ-iff by auto
have lim-bdd: lim (?f n)  $\in \{-M..M\}$  for n
proof –
  have closed  $\{-M..M\}$  using closed-real-atLeastAtMost by auto
  hence  $(\forall i. ?f n i \in \{-M..M\}) \wedge ?f n \longrightarrow \text{lim } (?f n) \longrightarrow \text{lim } (?f n) \in \{-M..M\}$ 
    unfolding closed-sequential-limits by (drule-tac x = \lambda k. f (d k) (r n) in spec) blast

```

moreover have $\forall i. ?f\ n\ i \in \{-M..M\}$
using *bdd* **by** (*simp add: abs-le-iff minus-le-iff*)
ultimately show $\lim\ (?f\ n) \in \{-M..M\}$
using *lim-f* **by** *auto*
qed
then have *limset-bdd*: $\bigwedge x. \{\lim\ (?f\ n) \mid n. x < r\ n\} \subseteq \{-M..M\}$
by *auto*
then have *bdd-below*: *bdd-below* $\{\lim\ (?f\ n) \mid n. x < r\ n\}$ **for** *x*
by (*metis (mono-tags) bdd-below-Icc bdd-below-mono*)
have *r-unbdd*: $\exists n. x < r\ n$ **for** *x*
using *dense-r*[*OF less-add-one, of x*] **by** *auto*
then have *nonempty*: $\{\lim\ (?f\ n) \mid n. x < r\ n\} \neq \{\}$ **for** *x*
by *auto*

define *F* **where** $F\ x = \text{Inf}\ \{\lim\ (?f\ n) \mid n. x < r\ n\}$ **for** *x*
have *F-eq*: *ereal* $(F\ x) = (\text{INF}\ n:\{n. x < r\ n\}. \text{ereal}\ (\lim\ (?f\ n)))$ **for** *x*
unfolding *F-def* **by** (*subst ereal-Inf'*[*OF bdd-below nonempty*]) (*simp add: setcompr-eq-image*)
have *mono-F*: *mono F*
using *nonempty* **by** (*auto intro!*: *cInf-superset-mono simp: F-def bdd-below mono-def*)
moreover have $\bigwedge x. \text{continuous}\ (\text{at-right}\ x)\ F$
unfolding *continuous-within order-tendsto-iff eventually-at-right*[*OF less-add-one*]
proof *safe*
show $F\ x < u \implies \exists b > x. \forall y > x. y < b \implies F\ y < u$ **for** *x u*
unfolding *F-def cInf-less-iff*[*OF nonempty bdd-below*] **by** *auto*
next
show $\exists b > x. \forall y > x. y < b \implies l < F\ y$ **if** *l*: $l < F\ x$ **for** *x l*
using *less-le-trans*[*OF l mono-F*[*THEN monoD, of x*]] **by** (*auto intro: less-add-one*)
qed
moreover
{ fix *x*
have $F\ x \in \{-M..M\}$
unfolding *F-def* **using** *limset-bdd bdd-below r-unbdd* **by** (*intro closed-subset-contains-Inf*)
auto
then have $|F\ x| \leq M$ **by** *auto* }
moreover have $(\lambda n. f\ (d\ n)\ x) \longrightarrow F\ x$ **if** *cts*: *continuous (at x) F* **for** *x*
proof (*rule limsup-le-liminf-real*)
show $\text{limsup}\ (\lambda n. f\ (d\ n)\ x) \leq F\ x$
proof (*rule tendsto-lowerbound*)
show $(F \longrightarrow \text{ereal}\ (F\ x))\ (\text{at-right}\ x)$
using *cts* **unfolding** *continuous-at-split* **by** (*auto simp: continuous-within*)
show $\forall_F\ i\ \text{in}\ \text{at-right}\ x. \text{limsup}\ (\lambda n. f\ (d\ n)\ x) \leq F\ i$
unfolding *eventually-at-right*[*OF less-add-one*]
proof (*rule, rule, rule less-add-one, safe*)
fix *y* **assume** *y*: $x < y$
with *dense-r* **obtain** *N* **where** $x < r\ N\ r\ N < y$ **by** *auto*
have $*$: $y < r\ n' \implies \lim\ (?f\ N) \leq \lim\ (?f\ n')$ **for** *n'*

```

      using ⟨r N < y⟩ by (intro LIMSEQ-le[OF lim-f lim-f]) (auto intro!:
mono[THEN monoD])
      have limsup (λn. f (d n) x) ≤ limsup (?f N)
        using ⟨x < r N⟩ by (auto intro!: Limsup-mono always-eventually
mono[THEN monoD])
      also have ... = lim (λn. ereal (?f N n))
      using rat-cnv[of N] by (force intro!: convergent-limsup-cl simp: convergent-def)
      also have ... ≤ F y
        by (auto intro!: INF-greatest * simp: convergent-real-imp-convergent-ereal
rat-cnv F-eq)
      finally show limsup (λn. f (d n) x) ≤ F y .
    qed
  qed simp
  show F x ≤ liminf (λn. f (d n) x)
  proof (rule tendsto-upperbound)
    show (F ⟶ ereal (F x)) (at-left x)
      using cts unfolding continuous-at-split by (auto simp: continuous-within)
    show ∀ F i in at-left x. F i ≤ liminf (λn. f (d n) x)
      unfolding eventually-at-left[OF minus-one-less]
    proof (rule, rule, rule minus-one-less, safe)
      fix y assume y: y < x
      with dense-r obtain N where y < r N r N < x by auto
      have F y ≤ liminf (?f N)
        using ⟨y < r N⟩ by (auto simp: F-eq convergent-real-imp-convergent-ereal
rat-cnv convergent-liminf-cl intro!: INF-lower2)
      also have ... ≤ liminf (λn. f (d n) x)
        using ⟨r N < x⟩ by (auto intro!: Liminf-mono monoD[OF mono]
always-eventually)
      finally show F y ≤ liminf (λn. f (d n) x) .
    qed
  qed simp
  qed
  ultimately show ?thesis using subseq by auto
  qed

```

definition

$tight :: (nat \Rightarrow real\ measure) \Rightarrow bool$

where

$tight\ \mu \equiv (\forall n. real\text{-distribution}\ (\mu\ n)) \wedge (\forall (\varepsilon::real)>0. \exists a\ b::real. a < b \wedge (\forall n. measure\ (\mu\ n)\ \{a<..b\} > 1 - \varepsilon))$

theorem *tight-imp-convergent-subsubsequence*:

assumes μ : *tight* μ *strict-mono* s

shows $\exists r\ M. strict\text{-mono}\ (r :: nat \Rightarrow nat) \wedge real\text{-distribution}\ M \wedge weak\text{-conv}\text{-}m$
 $(\mu \circ s \circ r)\ M$

proof –

define f **where** $f k = \text{cdf } (\mu (s k))$ **for** k
interpret μ : *real-distribution* μk **for** k
using μ **unfolding** *tight-def* **by** *auto*

have $rcont$: $\bigwedge x. \text{continuous } (\text{at-right } x) (f k)$
and $mono$: $\text{mono } (f k)$
and top : $(f k \longrightarrow 1)$ *at-top*
and bot : $(f k \longrightarrow 0)$ *at-bot* **for** k
unfolding *f-def mono-def*
using $\mu.cdf\text{-nondecreasing}$ $\mu.cdf\text{-is-right-cont}$ $\mu.cdf\text{-lim-at-top-prob}$ $\mu.cdf\text{-lim-at-bot}$
by *auto*

have bdd : $|f k x| \leq 1$ **for** $k x$
by (*auto simp add: abs-le-iff minus-le-iff f-def* $\mu.cdf\text{-nonneg}$ $\mu.cdf\text{-bounded-prob}$)

from *Helly-selection*[*OF* $rcont$ $mono$ bdd , *of* $\lambda x. x$] **obtain** $r F$
where F : *strict-mono* $r \bigwedge x. \text{continuous } (\text{at-right } x) F \text{ mono } F \bigwedge x. |F x| \leq 1$
and $\text{lim-}F$: $\bigwedge x. \text{continuous } (\text{at } x) F \implies (\lambda n. f (r n) x) \longrightarrow F x$
by *blast*

have $0 \leq f n x$ **for** $n x$
unfolding *f-def* **by** (*rule* $\mu.cdf\text{-nonneg}$)
have $F\text{-nonneg}$: $0 \leq F x$ **for** x
proof –
obtain y **where** $y < x$ *isCont* $F y$
using *open-minus-countable*[*OF* $\text{mono-ctble-discont}$ [*OF* $\langle \text{mono } F \rangle$], *of* $\{.. < x\}$] **by** *auto*
then **have** $0 \leq F y$
by (*intro* *LIMSEQ-le-const*[*OF* $\text{lim-}F$]) (*auto simp: f-def* $\mu.cdf\text{-nonneg}$)
also **have** $\dots \leq F x$
using $\langle y < x \rangle$ **by** (*auto intro!: monoD*[*OF* $\langle \text{mono } F \rangle$])
finally **show** $0 \leq F x$.
qed

have Fab : $\exists a b. (\forall x \geq b. F x \geq 1 - \varepsilon) \wedge (\forall x \leq a. F x \leq \varepsilon)$ **if** $\varepsilon: 0 < \varepsilon$ **for** ε
proof *auto*
obtain $a' b'$ **where** $a' b'$: $a' < b' \bigwedge k. \text{measure } (\mu k) \{a' < .. b'\} > 1 - \varepsilon$
using $\varepsilon \mu$ **by** (*auto simp: tight-def*)
obtain a **where** $a: a < a'$ *isCont* $F a$
using *open-minus-countable*[*OF* $\text{mono-ctble-discont}$ [*OF* $\langle \text{mono } F \rangle$], *of* $\{.. < a'\}$] **by** *auto*
obtain b **where** $b: b' < b$ *isCont* $F b$
using *open-minus-countable*[*OF* $\text{mono-ctble-discont}$ [*OF* $\langle \text{mono } F \rangle$], *of* $\{b' < ..\}$] **by** *auto*
have $a < b$
using $a b a' b'$ **by** *simp*

let $?\mu = \lambda k. \text{measure } (\mu (s (r k)))$
have ab : $?\mu k \{a < .. b\} > 1 - \varepsilon$ **for** k
proof –

have $?μ k \{a' < ..b'\} \leq ?μ k \{a < ..b\}$
using $a b$ **by** (*intro μ.finite-measure-mono*) *auto*
then show *?thesis*
using $a'b'(2)$ **by** (*metis less-eq-real-def less-trans*)
qed

have $(λk. ?μ k \{..b\}) \longrightarrow F b$
using $b(2)$ *lim-F unfolding f-def cdf-def o-def* **by** *auto*
then have $1 - ε \leq F b$
proof (*rule tendsto-lowerbound, intro always-eventually allI*)
fix k
have $1 - ε < ?μ k \{a < ..b\}$
using ab **by** *auto*
also have $... \leq ?μ k \{..b\}$
by (*auto intro!: μ.finite-measure-mono*)
finally show $1 - ε \leq ?μ k \{..b\}$
by (*rule less-imp-le*)
qed (*rule sequentially-bot*)
then show $\exists b. \forall x \geq b. 1 - ε \leq F x$
using F *unfolding mono-def* **by** (*metis order.trans*)

have $(λk. ?μ k \{..a\}) \longrightarrow F a$
using $a(2)$ *lim-F unfolding f-def cdf-def o-def* **by** *auto*
then have $F a: F a \leq ε$
proof (*rule tendsto-upperbound, intro always-eventually allI*)
fix k
have $?μ k \{..a\} + ?μ k \{a < ..b\} \leq 1$
by (*subst μ.finite-measure-Union[symmetric]*) *auto*
then show $?μ k \{..a\} \leq ε$
using $ab[of k]$ **by** *simp*
qed (*rule sequentially-bot*)
then show $\exists a. \forall x \leq a. F x \leq ε$
using F *unfolding mono-def* **by** (*metis order.trans*)
qed

have $(F \longrightarrow 1)$ *at-top*
proof (*rule order-tendstoI*)
show $1 < y \implies \forall_F x \text{ in } at-top. F x < y$ **for** y
using $\langle \bigwedge x. |F x| \leq 1 \rangle \langle \bigwedge x. 0 \leq F x \rangle$ **by** (*auto intro: le-less-trans always-eventually*)
fix $y :: real$ **assume** $y < 1$
then obtain z **where** $y < z < 1$
using *dense[of y 1]* **by** *auto*
with $Fab[of 1 - z]$ **show** $\forall_F x \text{ in } at-top. y < F x$
by (*auto simp: eventually-at-top-linorder intro: less-le-trans*)
qed
moreover
have $(F \longrightarrow 0)$ *at-bot*
proof (*rule order-tendstoI*)
show $y < 0 \implies \forall_F x \text{ in } at-bot. y < F x$ **for** y

```

    using  $\langle \bigwedge x. 0 \leq F x \rangle$  by (auto intro: less-le-trans always-eventually)
  fix  $y :: \text{real}$  assume  $0 < y$ 
  then obtain  $z$  where  $0 < z < y$ 
    using dense[of 0 y] by auto
  with Fab[of z] show  $\forall_F x$  in at-bot.  $F x < y$ 
    by (auto simp: eventually-at-bot-linorder intro: le-less-trans)
  qed
  ultimately have  $M: \text{real-distribution (interval-measure } F) \text{ cdf (interval-measure } F) = F$ 
    using  $F$  by (auto intro!: real-distribution-interval-measure cdf-interval-measure
  simp: mono-def)
  with  $\text{lim-}F \text{ LIMSEQ-subseq-LIMSEQ } M$  have  $\text{weak-conv-}m (\mu \circ s \circ r)$  (interval-measure
   $F$ )
    by (auto simp: weak-conv-def weak-conv-m-def f-def comp-def)
  then show  $\exists r M. \text{strict-mono } (r :: \text{nat} \Rightarrow \text{nat}) \wedge (\text{real-distribution } M \wedge$ 
   $\text{weak-conv-}m (\mu \circ s \circ r) M)$ 
    using  $F M$  by auto
  qed

```

corollary *tight-subseq-weak-converge:*

```

  fixes  $\mu :: \text{nat} \Rightarrow \text{real measure}$  and  $M :: \text{real measure}$ 
  assumes  $\bigwedge n. \text{real-distribution } (\mu n) \text{ real-distribution } M$  and tight: tight  $\mu$  and
    subseq:  $\bigwedge s \nu. \text{strict-mono } s \implies \text{real-distribution } \nu \implies \text{weak-conv-}m (\mu \circ s) \nu$ 
     $\implies \text{weak-conv-}m (\mu \circ s) M$ 
  shows  $\text{weak-conv-}m \mu M$ 
  proof (rule ccontr)
    define  $f$  where  $f n = \text{cdf } (\mu n)$  for  $n$ 
    define  $F$  where  $F = \text{cdf } M$ 

    assume  $\neg \text{weak-conv-}m \mu M$ 
    then obtain  $x$  where  $x: \text{isCont } F x \wedge (\lambda n. f n x) \not\longrightarrow F x$ 
      by (auto simp: weak-conv-m-def weak-conv-def f-def F-def)
    then obtain  $\varepsilon$  where  $\varepsilon > 0$  and infinite  $\{n. \neg \text{dist } (f n x) (F x) < \varepsilon\}$ 
      by (auto simp: tendsto-iff not-eventually INFM-iff-infinite cofinite-eq-sequentially[symmetric])
    then obtain  $s :: \text{nat} \Rightarrow \text{nat}$  where  $s: \bigwedge n. \neg \text{dist } (f (s n) x) (F x) < \varepsilon$  and
      strict-mono  $s$ 
      using enumerate-in-set enumerate-mono by (fastforce simp: strict-mono-def)
    then obtain  $r \nu$  where  $r: \text{strict-mono } r \text{ real-distribution } \nu \text{ weak-conv-}m (\mu \circ s$ 
     $\circ r) \nu$ 
      using tight-imp-convergent-subsubsequence[OF tight] by blast
    then have  $\text{weak-conv-}m (\mu \circ (s \circ r)) M$ 
      using (strict-mono  $s$ )  $r$  by (intro subseq strict-mono-o) (auto simp: comp-assoc)
    then have  $(\lambda n. f (s (r n)) x) \longrightarrow F x$ 
      using  $x$  by (auto simp: weak-conv-m-def weak-conv-def F-def f-def)
    then show False
      using  $s (\varepsilon > 0)$  by (auto dest: tendstoD)
  qed
end

```

18 Integral of sinc

```
theory Sinc-Integral
  imports Distributions
begin
```

18.1 Various preparatory integrals

Naming convention The theorem name consists of the following parts:

- Kind of integral: *has-bochner-integral / integrable / LBINT*
- Interval: Interval (0 / infinity / open / closed) (infinity / open / closed)
- Name of the occurring constants: power, exp, m (for minus), scale, sin, ...

lemma *has-bochner-integral-I0i-power-exp-m'*:
*has-bochner-integral lborel ($\lambda x. x^k * \exp(-x) * \text{indicator } \{0 \dots\} x::\text{real}$) (fact k)*

```
using nn-integral-power-times-exp-Ici[of k]
by (intro has-bochner-integral-nn-integral)
   (auto simp: nn-integral-set-ennreal split: split-indicator)
```

lemma *has-bochner-integral-I0i-power-exp-m*:
*has-bochner-integral lborel ($\lambda x. x^k * \exp(-x) * \text{indicator } \{0 < \dots\} x::\text{real}$) (fact k)*

```
using AE-lborel-singleton[of 0]
by (intro has-bochner-integral-cong-AE[THEN iffD1, OF - - - has-bochner-integral-I0i-power-exp-m'])
   (auto split: split-indicator)
```

lemma *integrable-I0i-exp-mscale*: $0 < (u::\text{real}) \implies \text{set-integrable lborel } \{0 < \dots\} (\lambda x. \exp(-(x * u)))$

```
using lborel-integrable-real-affine-iff[of u  $\lambda x. \text{indicator } \{0 < \dots\} x *_{\mathbb{R}} \exp(-x)$  0]
   has-bochner-integral-I0i-power-exp-m[of 0]
by (simp add: indicator-def zero-less-mult-iff mult-ac integrable.intros)
```

lemma *LBINT-I0i-exp-mscale*: $0 < (u::\text{real}) \implies \text{LBINT } x=0..\infty. \exp(-(x * u)) = 1 / u$

```
using lborel-integral-real-affine[of u  $\lambda x. \text{indicator } \{0 < \dots\} x *_{\mathbb{R}} \exp(-x)$  0]
   has-bochner-integral-I0i-power-exp-m[of 0]
by (auto simp: indicator-def zero-less-mult-iff interval-lebesgue-integral-0-infity
   field-simps
   dest!: has-bochner-integral-integral-eq)
```

lemma *LBINT-I0c-exp-mscale-sin*:

```
LBINT x=0..t.  $\exp(-(u * x)) * \sin x =$ 
  (1 / (1 + u^2)) * (1 -  $\exp(-(u * t)) * (u * \sin t + \cos t)$ ) (is - = ?F t)
```

unfolding *zero-ereal-def*
proof (*subst interval-integral-FTC-finite*)
show ($?F$ *has-vector-derivative* $\exp(-u * x) * \sin x$) (*at* x *within* $\{ \min 0 t .. \max 0 t \}$) **for** x
by (*auto intro!*: *derivative-eq-intros*
simp: has-field-derivative-iff-has-vector-derivative[symmetric] power2-eq-square)
(simp-all add: field-simps add-nonneg-eq-0-iff)
qed (*auto intro: continuous-at-imp-continuous-on*)

lemma *LBINT-I0i-exp-mscale-sin*:

assumes $0 < x$
shows *LBINT* $u=0..∞. |\exp(-u * x) * \sin x| = |\sin x| / x$
proof (*subst interval-integral-FTC-nonneg*)
let $?F = \lambda u. 1 / x * (1 - \exp(-u * x)) * |\sin x|$
show $\bigwedge t. (?F$ *has-real-derivative* $|\exp(-t * x) * \sin x|$) (*at* t)
using $\langle 0 < x \rangle$ **by** (*auto intro!*: *derivative-eq-intros simp: abs-mult*)
show ($(?F \circ \text{real-of-ereal}) \longrightarrow 0$) (*at-right* 0)
using $\langle 0 < x \rangle$ **by** (*auto simp: zero-ereal-def ereal-tendsto-simps intro!: tendsto-eq-intros*)
have $*$: ($\lambda t. \exp(-t * x) \longrightarrow 0$) *at-top*
using $\langle 0 < x \rangle$
by (*auto intro!*: *exp-at-bot[THEN filterlim-compose] filterlim-tendsto-pos-mult-at-top filterlim-ident*
simp: filterlim-uminus-at-bot mult.commute[of - x])
show ($(?F \circ \text{real-of-ereal}) \longrightarrow |\sin x| / x$) (*at-left* ∞)
using $\langle 0 < x \rangle$ **unfolding** *ereal-tendsto-simps*
by (*intro filterlim-compose[OF - *]*) (*auto intro!: tendsto-eq-intros filterlim-ident*)
qed *auto*

lemma

shows *integrable-inverse-1-plus-square*:
set-integrable lborel (einterval $(-\infty) \infty$) ($\lambda x. \text{inverse}(1 + x^2)$)
and *LBINT-inverse-1-plus-square*:
LBINT $x=-∞..∞. \text{inverse}(1 + x^2) = \pi$
proof –
have 1 : $-(\pi / 2) < x \implies x * 2 < \pi \implies (\tan$ *has-real-derivative* $1 + (\tan x)^2$) (*at* x) **for** x
using *cos-gt-zero-pi[of x]* **by** (*subst tan-sec*) (*auto intro!*: *DERIV-tan simp: power-inverse*)
have 2 : $-(\pi / 2) < x \implies x * 2 < \pi \implies \text{isCont}(\lambda x. 1 + (\tan x)^2) x$ **for** x
using *cos-gt-zero-pi[of x]* **by** *auto*
show *LBINT* $x=-∞..∞. \text{inverse}(1 + x^2) = \pi$
by (*subst interval-integral-substitution-nonneg[of $-\pi/2 \ \pi/2 \ \tan \lambda x. 1 + (\tan x)^2$]*)
(auto intro: derivative-eq-intros 1 2 filterlim-tan-at-right
simp add: ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff)
show *set-integrable lborel (einterval $(-\infty) \infty$) ($\lambda x. \text{inverse}(1 + x^2)$)*
by (*subst interval-integral-substitution-nonneg[of $-\pi/2 \ \pi/2 \ \tan \lambda x. 1 + (\tan x)^2$]*)
(auto intro: derivative-eq-intros 1 2 filterlim-tan-at-right

simp add: ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff)

qed

lemma

shows *integrable-I0i-1-div-plus-square:*

interval-lebesgue-integrable lborel 0 ∞ (λx. 1 / (1 + x²))

and *LBINT-I0i-1-div-plus-square:*

LBINT x=0..∞. 1 / (1 + x²) = pi / 2

proof –

have 1: $0 < x \implies x * 2 < \pi \implies (\text{tan has-real-derivative } 1 + (\text{tan } x)^2) \text{ (at } x)$
for x

using *cos-gt-zero-pi[of x]* **by** (*subst tan-sec*) (*auto intro!: DERIV-tan simp: power-inverse*)

have 2: $0 < x \implies x * 2 < \pi \implies \text{isCont } (\lambda x. 1 + (\text{tan } x)^2) \text{ } x \text{ for } x$

using *cos-gt-zero-pi[of x]* **by** *auto*

show *LBINT x=0..∞. 1 / (1 + x²) = pi / 2*

by (*subst interval-integral-substitution-nonneg[of 0 pi/2 tan λx. 1 + (tan x)²]*)
(auto intro: derivative-eq-intros 1 2 tendsto-eq-intros

simp add: ereal-tendsto-simps filterlim-tan-at-left zero-ereal-def add-nonneg-eq-0-iff)

show *interval-lebesgue-integrable lborel 0 ∞ (λx. 1 / (1 + x²))*

unfolding *interval-lebesgue-integrable-def*

by (*subst interval-integral-substitution-nonneg[of 0 pi/2 tan λx. 1 + (tan x)²]*)
(auto intro: derivative-eq-intros 1 2 tendsto-eq-intros

simp add: ereal-tendsto-simps filterlim-tan-at-left zero-ereal-def add-nonneg-eq-0-iff)

qed

19 The sinc function, and the sine integral (Si)

abbreviation *sinc* :: *real* \Rightarrow *real* **where**

sinc $\equiv (\lambda x. \text{if } x = 0 \text{ then } 1 \text{ else } \sin x / x)$

lemma *sinc-at-0*: $((\lambda x. \sin x / x :: \text{real}) \longrightarrow 1) \text{ (at } 0)$

using *DERIV-sin [of 0]* **by** (*auto simp add: has-field-derivative-def field-has-derivative-at*)

lemma *isCont-sinc*: *isCont sinc x*

proof *cases*

assume $x = 0$ **then show** *?thesis*

using *LIM-equal [where g = λx. sin x / x and a=0 and f=sinc and l=1]*

by (*auto simp: isCont-def sinc-at-0*)

next

assume $x \neq 0$ **show** *?thesis*

by (*rule continuous-transform-within [where d = abs x and f = λx. sin x / x]*)

(auto simp add: dist-real-def (x ≠ 0))

qed

lemma *continuous-on-sinc[continuous-intros]*:

continuous-on S f \implies continuous-on S (λx. sinc (f x))

using *continuous-on-compose[of S f sinc, OF - continuous-at-imp-continuous-on]*

by (auto simp: isCont-sinc)

lemma borel-measurable-sinc[measurable]: sinc \in borel-measurable borel

by (intro borel-measurable-continuous-on1 continuous-at-imp-continuous-on ballI isCont-sinc)

lemma sinc-AE: AE x in lborel. sin x / x = sinc x

by (rule AE-I [where $N = \{0\}$], auto)

definition Si :: real \Rightarrow real **where** Si $t \equiv$ LBINT $x=0..t$. sin x / x

lemma sinc-neg [simp]: sinc $(-x) =$ sinc x

by auto

lemma Si-alt-def : Si $t =$ LBINT $x=0..t$. sinc x

proof cases

assume $0 \leq t$ **then show** ?thesis

using AE-lborel-singleton[of 0]

by (auto simp: Si-def intro!: interval-lebesgue-integral-cong-AE)

next

assume $\neg 0 \leq t$ **then show** ?thesis

unfolding Si-def using AE-lborel-singleton[of 0]

by (subst (1 2) interval-integral-endpoints-reverse)

(auto simp: Si-def intro!: interval-lebesgue-integral-cong-AE)

qed

lemma Si-neg:

assumes $T \geq 0$ **shows** Si $(-T) = -$ Si T

proof –

have LBINT $x=ereal 0..T$. $-1 *_{\mathbb{R}}$ sinc $(-x) =$ LBINT $y=ereal (-0)..ereal (-T)$. sinc y

by (rule interval-integral-substitution-finite [OF assms])

(auto intro: derivative-intros continuous-at-imp-continuous-on isCont-sinc)

also have (LBINT $x=ereal 0..T$. $-1 *_{\mathbb{R}}$ sinc $(-x)) = -$ (LBINT $x=ereal 0..T$. sinc x)

by (subst sinc-neg) (simp-all add: interval-lebesgue-integral-uminus)

finally have *: $-$ (LBINT $x=ereal 0..T$. sinc $x) =$ LBINT $y=ereal 0..ereal (-T)$. sinc y

by simp

show ?thesis

using assms **unfolding** Si-alt-def

by (subst zero-ereal-def)+ (auto simp add: * [symmetric])

qed

lemma integrable-sinc':

interval-lebesgue-integrable lborel (ereal 0) (ereal T) (λt . sin $(t * \vartheta)$ / t)

proof –

have *: interval-lebesgue-integrable lborel (ereal 0) (ereal T) (λt . $\vartheta *$ sinc $(t *$

ϑ))
by (*intro interval-lebesgue-integrable-mult-right interval-integrable-isCont continuous-within-compose3*
[OF isCont-sinc])
auto
show *?thesis*
by (*rule interval-lebesgue-integrable-cong-AE[THEN iffD1, OF - - - *]*)
(insert AE-lborel-singleton[of 0], auto)
qed

lemma *DERIV-Si: (Si has-real-derivative sinc x) (at x)*

proof –

have (*at x within {min 0 (x - 1)..max 0 (x + 1)}*) = *at x*
by (*intro at-within-interior*) *auto*
moreover have $\min 0 (x - 1) \leq x \leq \max 0 (x + 1)$
by *auto*
ultimately show *?thesis*
using *interval-integral-FTC2[of min 0 (x - 1) 0 max 0 (x + 1) sinc x]*
by (*auto simp: continuous-at-imp-continuous-on isCont-sinc Si-alt-def[abs-def]*)
zero-ereal-def
has-field-derivative-iff-has-vector-derivative
split del: if-split

qed

lemma *isCont-Si: isCont Si x*

using *DERIV-Si by (rule DERIV-isCont)*

lemma *borel-measurable-Si[measurable]: Si ∈ borel-measurable borel*

by (*auto intro: isCont-Si continuous-at-imp-continuous-on borel-measurable-continuous-on1*)

lemma *Si-at-top-LBINT:*

$((\lambda t. (LBINT x=0..\infty. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))) \longrightarrow$
 $0) \text{ at-top}$

proof –

let $?F = \lambda x t. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2) :: \text{real}$
have *int: set-integrable lborel {0<..} (λx. exp(-x) * (x + 1) :: real)*
unfolding *distrib-left*
using *has-bochner-integral-I0i-power-exp-m[of 0] has-bochner-integral-I0i-power-exp-m[of 1]*
by (*intro set-integral-add*) (*auto dest!: integrable.intros simp: ac-simps*)

have $((\lambda t :: \text{real}. LBINT x:\{0<..\}. ?F x t) \longrightarrow LBINT x :: \text{real}:\{0<..\}. 0) \text{ at-top}$

proof (*rule integral-dominated-convergence-at-top[OF - - int], simp-all del: abs-divide split: split-indicator*)

have $*: 0 < x \implies |x * \sin t + \cos t| / (1 + x^2) \leq (x * 1 + 1) / 1 \text{ for } x t :: \text{real}$

by (*intro frac-le abs-triangle-ineq[THEN order-trans] add-mono*)

(auto simp add: abs-mult simp del: mult-1-right intro!: mult-mono)

then have $1 \leq t \implies 0 < x \implies |?F x t| \leq \exp(-x) * (x + 1) \text{ for } x t :: \text{real}$


```

    by (auto simp add: abs-mult times-divide-eq-right[symmetric] simp del:
times-divide-eq-right
      intro!: mult-mono)
  then show  $\forall_F i$  in at-top.  $\text{AE } x$  in lborel.  $0 < x \longrightarrow |?F x i| \leq \exp(-x) * (x + 1)$ 
    by (auto intro: eventually-mono eventually-ge-at-top[of 1::real])
  show  $\text{AE } x$  in lborel.  $0 < x \longrightarrow (?F x \longrightarrow 0)$  at-top
  proof (intro always-eventually impI allI)
    fix  $x :: \text{real}$  assume  $0 < x$ 
    show  $((\lambda t. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2)) \longrightarrow 0)$  at-top
    proof (rule Lim-null-comparison)
      show  $\forall_F t$  in at-top.  $\text{norm } (?F x t) \leq \exp(-(x * t)) * (x + 1)$ 
      using eventually-ge-at-top[of 1::real] *  $\langle 0 < x \rangle$ 
      by (auto simp add: abs-mult times-divide-eq-right[symmetric] simp del:
times-divide-eq-right
        intro!: mult-mono elim: eventually-mono)
    show  $((\lambda t. \exp(-(x * t)) * (x + 1)) \longrightarrow 0)$  at-top
      by (auto simp: filterlim-uminus-at-top [symmetric]
        intro!: filterlim-tendsto-pos-mult-at-top[OF tendsto-const]  $\langle 0 < x \rangle$ )
  filterlim-ident
    tendsto-mult-left-zero filterlim-compose[OF exp-at-bot])

  qed
  qed
  qed
  then show  $((\lambda t. (\text{LBINT } x=0..\infty. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))) \longrightarrow 0)$  at-top
    by (simp add: interval-lebesgue-integral-0-infty)
  qed

lemma Si-at-top-integrable:
  assumes  $t \geq 0$ 
  shows interval-lebesgue-integrable lborel  $0 \infty (\lambda x. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))$ 
  using  $\langle 0 \leq t \rangle$  unfolding le-less
  proof
    assume  $0 = t$  then show ?thesis
      using integrable-I0i-1-div-plus-square by simp
  next
    assume [arith]:  $0 < t$ 
    have *:  $0 \leq a \implies 0 < x \implies a / (1 + x^2) \leq a$  for  $a x :: \text{real}$ 
      using zero-le-power2[of x, arith] using divide-left-mono[of 1  $1+x^2$  a] by auto
    have set-integrable lborel  $\{0 <..\}$   $(\lambda x. (\exp(-x) * x) * (\sin t / t) + \exp(-x) * \cos t)$ 
      using has-bochner-integral-I0i-power-exp-m[of 0] has-bochner-integral-I0i-power-exp-m[of 1]
    by (intro set-integral-add set-integrable-mult-left)
      (auto dest!: integrable.intros simp: ac-simps)
    from lborel-integrable-real-affine[OF this, of t 0]
    show ?thesis

```

unfolding *interval-lebesgue-integral-0-infty*
by (rule *Bochner-Integration.integrable-bound*) (auto simp: *field-simps * split: split-indicator*)
qed

lemma *Si-at-top*: ($Si \longrightarrow pi / 2$) *at-top*

proof –

have $\forall_F t$ in *at-top*. $pi / 2 - (LBINT u=0..\infty. \exp(-(u * t)) * (u * \sin t + \cos t) / (1 + u^2)) = Si\ t$

using *eventually-ge-at-top[of 0]*

proof *eventually-elim*

fix $t :: real$ **assume** $t \geq 0$

have $Si\ t = LBINT x=0..t. \sin x * (LBINT u=0..\infty. \exp(-(u * x)))$

unfolding *Si-def* **using** $\langle 0 \leq t \rangle$

by (intro *interval-integral-discrete-difference[where X={0}]*) (auto simp: *LBINT-I0i-exp-mscale*)

also have $\dots = LBINT x. (LBINT u=ereal\ 0..\infty. \text{indicator } \{0 <.. < t\} x *_R \sin x * \exp(-(u * x)))$

using $\langle 0 \leq t \rangle$ **by** (simp add: *zero-ereal-def interval-lebesgue-integral-le-eq mult-ac*)

also have $\dots = LBINT x. (LBINT u. \text{indicator } (\{0 <..\} \times \{0 <.. < t\}) (u, x) *_R (\sin x * \exp(-(u * x))))$

by (subst *interval-integral-I0i*) (simp-all add: *indicator-times ac-simps*)

also have $\dots = LBINT u. (LBINT x. \text{indicator } (\{0 <..\} \times \{0 <.. < t\}) (u, x) *_R (\sin x * \exp(-(u * x))))$

proof (intro *lborel-pair.Fubini-integral[symmetric] lborel-pair.Fubini-integrable*)

show $(\lambda(x, y). \text{indicator } (\{0 <..\} \times \{0 <.. < t\}) (y, x) *_R (\sin x * \exp(-(y * x))))$

$\in \text{borel-measurable } (lborel \otimes_M lborel)$ (**is** $?f \in \text{borel-measurable}$ -)

by *measurable*

have $AE\ x$ in *lborel*. $\text{indicator } \{0..t\} x *_R \text{abs } (\text{sinc } x) = LBINT\ y. \text{norm } (?f\ (x, y))$

using *AE-lborel-singleton[of 0] AE-lborel-singleton[of t]*

proof *eventually-elim*

fix $x :: real$ **assume** $x \neq 0$ $x \neq t$

have $LBINT\ y. |\text{indicator } (\{0 <..\} \times \{0 <.. < t\}) (y, x) *_R (\sin x * \exp(-(y * x)))| =$

$LBINT\ y. |\sin x| * \exp(-(y * x)) * \text{indicator } \{0 <..\} y * \text{indicator } \{0 <.. < t\} x$

by (intro *Bochner-Integration.integral-cong*) (auto split: *split-indicator simp: abs-mult*)

also have $\dots = |\sin x| * \text{indicator } \{0 <.. < t\} x * (LBINT\ y=0..\infty. \exp(-(y * x)))$

by (*cases x > 0*)

(auto simp: *field-simps interval-lebesgue-integral-0-infty split: split-indicator*)

also have $\dots = |\sin x| * \text{indicator } \{0 <.. < t\} x * (1 / x)$

by (*cases x > 0*) (auto simp add: *LBINT-I0i-exp-mscale*)

also have $\dots = \text{indicator } \{0..t\} x *_R |\text{sinc } x|$

```

    using x by (simp add: field-simps split: split-indicator)
    finally show indicator {0..t} x *R abs (sinc x) = LBINT y. norm (?f (x,
y))
    by simp
qed
moreover have set-integrable lborel {0 .. t} (λx. abs (sinc x))
by (auto intro!: borel-integrable-compact continuous-intros simp del: real-scaleR-def)
ultimately show integrable lborel (λx. LBINT y. norm (?f (x, y)))
by (rule integrable-cong-AE-imp[rotated 2]) simp

have 0 < x ⇒ set-integrable lborel {0<..} (λy. sin x * exp (-(y * x))) for
x :: real
by (intro set-integrable-mult-right integrable-I0i-exp-mscale)
then show AE x in lborel. integrable lborel (λy. ?f (x, y))
by (intro AE-I2) (auto simp: indicator-times split: split-indicator)
qed
also have ... = LBINT u=0..∞. (LBINT x=0..t. exp (-(u * x)) * sin x)
using <0≤t>
by (auto simp: interval-lebesgue-integral-def zero-ereal-def ac-simps
split: split-indicator intro!: Bochner-Integration.integral-cong)
also have ... = LBINT u=0..∞. 1 / (1 + u2) - 1 / (1 + u2) * (exp (-(u
* t)) * (u * sin t + cos t))
by (auto simp: divide-simps LBINT-I0c-exp-mscale-sin intro!: interval-integral-cong)
also have ... = pi / 2 - (LBINT u=0..∞. exp (-(u * t)) * (u * sin t + cos
t) / (1 + u2))
using Si-at-top-integrable[OF <0≤t>] by (simp add: integrable-I0i-1-div-plus-square
LBINT-I0i-1-div-plus-square)
finally show pi / 2 - (LBINT u=0..∞. exp (-(u * t)) * (u * sin t + cos t)
/ (1 + u2)) = Si t ..
qed
then show ?thesis
by (rule Lim-transform-eventually)
(auto intro!: tendsto-eq-intros Si-at-top-LBINT)
qed

```

19.1 The final theorems: boundedness and scalability

lemma *bounded-Si*: $\exists B. \forall T. |Si\ T| \leq B$

proof –

```

have *: 0 ≤ y ⇒ dist x y < z ⇒ abs x ≤ y + z for x y z :: real
by (simp add: dist-real-def)

```

```

have eventually (λT. dist (Si T) (pi / 2) < 1) at-top
using Si-at-top by (elim tendstoD) simp
then have eventually (λT. 0 ≤ T ∧ |Si T| ≤ pi / 2 + 1) at-top
using eventually-ge-at-top[of 0] by eventually-elim (insert *[of pi/2 Si - 1],
auto)
then have ∃ T. 0 ≤ T ∧ (∀ t ≥ T. |Si t| ≤ pi / 2 + 1)
by (auto simp add: eventually-at-top-linorder)

```

then obtain T where $T: 0 \leq T \wedge t. t \geq T \implies |Si\ t| \leq pi / 2 + 1$
 by *auto*
moreover have $t \leq -T \implies |Si\ t| \leq pi / 2 + 1$ for t
 using $T(1)\ T(2)[of\ -t]\ Si\ neg[of\ -t]$ by *simp*
moreover have $\exists M. \forall t. -T \leq t \wedge t \leq T \longrightarrow |Si\ t| \leq M$
 by (rule *isCont-bounded*) (auto *intro!*: *isCont-Si continuous-intros* $\langle 0 \leq T \rangle$)
then obtain M where $M: \bigwedge t. -T \leq t \wedge t \leq T \implies |Si\ t| \leq M$
 by *auto*
ultimately show *?thesis*
 by (*intro exI*[*of - max M (pi/2 + 1)*]) (*meson le-max-iff-disj linorder-not-le order-le-less*)
qed

lemma *LBINT-I0c-sin-scale-divide*:

assumes $T \geq 0$
 shows $LBINT\ t=0..T. \sin\ (t * \vartheta) / t = \text{sgn}\ \vartheta * Si\ (T * |\vartheta|)$
proof –
 { **assume $0 < \vartheta$**
 have $(LBINT\ t=ereal\ 0..T. \sin\ (t * \vartheta) / t) = (LBINT\ t=ereal\ 0..T. \vartheta *_R \text{ sinc}\ (t * \vartheta))$
 by (rule *interval-integral-discrete-difference*[*of* $\{0\}$]) *auto*
 also have $\dots = (LBINT\ t=ereal\ (0 * \vartheta)..T * \vartheta. \text{ sinc}\ t)$
 apply (rule *interval-integral-substitution-finite* [*OF assms*])
 apply (subst *mult.commute*, rule *DERIV-subset*)
 by (auto *intro!*: *derivative-intros continuous-at-imp-continuous-on isCont-sinc*)
 also have $\dots = (LBINT\ t=ereal\ (0 * \vartheta)..T * \vartheta. \sin\ t / t)$
 by (rule *interval-integral-discrete-difference*[*of* $\{0\}$]) *auto*
 finally have $(LBINT\ t=ereal\ 0..T. \sin\ (t * \vartheta) / t) = (LBINT\ t=ereal\ 0..T * \vartheta. \sin\ t / t)$
 by *simp*
 hence $LBINT\ x. \text{ indicator}\ \{0 <.. < T\}\ x * \sin\ (x * \vartheta) / x =$
 $LBINT\ x. \text{ indicator}\ \{0 <.. < T * \vartheta\}\ x * \sin\ x / x$
 using *assms* $\langle 0 < \vartheta \rangle$ **unfolding** *interval-lebesgue-integral-def einterval-eq zero-ereal-def*
 by (auto *simp: ac-simps*)
 } **note** *aux1 = this*
 { **assume $0 > \vartheta$**
 have [*simp*]: $\text{integrable\ lborel}\ (\lambda x. \sin\ (x * \vartheta) * \text{ indicator}\ \{0 <.. < T\}\ x / x)$
 using *integrable-sinc'* [*of T* ϑ] *assms*
 by (*simp add: interval-lebesgue-integrable-def ac-simps*)
 have $(LBINT\ t=ereal\ 0..T. \sin\ (t * -\vartheta) / t) = (LBINT\ t=ereal\ 0..T. -\vartheta *_R \text{ sinc}\ (t * -\vartheta))$
 by (rule *interval-integral-discrete-difference*[*of* $\{0\}$]) *auto*
 also have $\dots = (LBINT\ t=ereal\ (0 * -\vartheta)..T * -\vartheta. \text{ sinc}\ t)$
 apply (rule *interval-integral-substitution-finite* [*OF assms*])
 apply (subst *mult.commute*, rule *DERIV-subset*)
 by (auto *intro!*: *derivative-intros continuous-at-imp-continuous-on isCont-sinc*)
 also have $\dots = (LBINT\ t=ereal\ (0 * -\vartheta)..T * -\vartheta. \sin\ t / t)$
 by (rule *interval-integral-discrete-difference*[*of* $\{0\}$]) *auto*

```

finally have (LBINT t=ereal 0..T. sin (t * -∅) / t) = (LBINT t=ereal 0..T
* -∅. sin t / t)
  by simp
  hence LBINT x. indicator {0<..using assms (0 > ∅) unfolding interval-lebesgue-integral-def einterval-eq
zero-ereal-def
    by (auto simp add: field-simps mult-le-0-iff split: if-split-asm)
  } note aux2 = this
  show ?thesis
    using assms unfolding Si-def interval-lebesgue-integral-def sgn-real-def einterval-eq
zero-ereal-def
    apply auto
    apply (erule aux1)
    apply (rule aux2)
    apply auto
    done
qed
end

```

20 The Levy inversion theorem, and the Levy continuity theorem.

```

theory Levy
  imports Characteristic-Functions Helly-Selection Sinc-Integral
begin

```

20.1 The Levy inversion theorem

```

lemma Levy-Inversion-aux1:

```

```

  fixes a b :: real

```

```

  assumes a ≤ b

```

```

  shows ((λt. (iexp -(t * a)) - iexp -(t * b))) / (i * t) ⟶ b - a (at 0)
    (is (?F ⟶ -) (at -))

```

```

proof -

```

```

  have 1: cmod (?F t - (b - a)) ≤ a^2 / 2 * abs t + b^2 / 2 * abs t if t ≠ 0
for t

```

```

  proof -

```

```

    have cmod (?F t - (b - a)) = cmod (
      (iexp -(t * a) - (1 + i * -(t * a))) / (i * t) -
      (iexp -(t * b) - (1 + i * -(t * b))) / (i * t)
      (is - = cmod (?one / (i * t) - ?two / (i * t)))
    )

```

```

    using (t ≠ 0) by (intro arg-cong[where f=norm]) (simp add: field-simps)

```

```

  also have ... ≤ cmod (?one / (i * t)) + cmod (?two / (i * t))

```

```

  by (rule norm-triangle-ineq4)

```

```

  also have cmod (?one / (i * t)) = cmod ?one / abs t

```

```

  by (simp add: norm-divide norm-mult)

```

```

also have  $cmod ( ?two / (i * t) ) = cmod ?two / abs t$ 
  by (simp add: norm-divide norm-mult)
also have  $cmod ?one / abs t + cmod ?two / abs t \leq$ 
   $((- (a * t))^2 / 2) / abs t + ((- (b * t))^2 / 2) / abs t$ 
  apply (rule add-mono)
  apply (rule divide-right-mono)
using iexp-approx1 [of  $-(t * a)$  1] apply (simp add: field-simps eval-nat-numeral)
  apply force
  apply (rule divide-right-mono)
using iexp-approx1 [of  $-(t * b)$  1] apply (simp add: field-simps eval-nat-numeral)
  by force
also have  $\dots = a^2 / 2 * abs t + b^2 / 2 * abs t$ 
  using  $\langle t \neq 0 \rangle$  apply (case-tac  $t \geq 0$ , simp add: field-simps power2-eq-square)
  using  $\langle t \neq 0 \rangle$  by (subst (1 2) abs-of-neg, auto simp add: field-simps power2-eq-square)
finally show  $cmod (?F t - (b - a)) \leq a^2 / 2 * abs t + b^2 / 2 * abs t .$ 
qed
show ?thesis
  apply (rule LIM-zero-cancel)
  apply (rule tendsto-norm-zero-cancel)
  apply (rule real-LIM-sandwich-zero [OF - - 1])
  apply (auto intro!: tendsto-eq-intros)
done
qed

```

```

lemma Levy-Inversion-aux2:
  fixes  $a b t :: real$ 
  assumes  $a \leq b$  and  $t \neq 0$ 
  shows  $cmod ((iexp (t * b) - iexp (t * a)) / (i * t)) \leq b - a$  (is ?F  $\leq -$ )
proof -
  have ?F =  $cmod (iexp (t * a) * (iexp (t * (b - a)) - 1) / (i * t))$ 
    using  $\langle t \neq 0 \rangle$  by (intro arg-cong[where  $f=norm$ ]) (simp add: field-simps
  exp-diff exp-minus)
  also have  $\dots = cmod (iexp (t * (b - a)) - 1) / abs t$ 
    unfolding norm-divide norm-mult norm-exp-i-times using  $\langle t \neq 0 \rangle$ 
    by (simp add: complex-eq-iff norm-mult)
  also have  $\dots \leq abs (t * (b - a)) / abs t$ 
    using iexp-approx1 [of  $t * (b - a)$  0]
    by (intro divide-right-mono) (auto simp add: field-simps eval-nat-numeral)
  also have  $\dots = b - a$ 
    using assms by (auto simp add: abs-mult)
  finally show ?thesis .
qed

```

```

theorem (in real-distribution) Levy-Inversion:
  fixes  $a b :: real$ 
  assumes  $a \leq b$ 
  defines  $\mu \equiv measure M$  and  $\varphi \equiv char M$ 
  assumes  $\mu \{a\} = 0$  and  $\mu \{b\} = 0$ 

```

shows $(\lambda T. 1 / (2 * \pi) * (CLBINT t=-T..T. (iexp (-(t * a)) - iexp (-(t * b))) / (i * t) * \varphi t))$
 $\longrightarrow \mu \{a<..b\}$
(is $(\lambda T. 1 / (2 * \pi) * (CLBINT t=-T..T. ?F t * \varphi t)) \longrightarrow of-real (\mu \{a<..b\}))$

proof -

interpret P : pair-sigma-finite lborel M ..
from bounded-Si **obtain** B **where** $Bprop$: $\bigwedge T. abs (Si T) \leq B$ **by** auto
from $Bprop$ [of 0] **have** [simp]: $B \geq 0$ **by** auto
let $?f = \lambda t x :: real. (iexp (t * (x - a)) - iexp(t * (x - b))) / (i * t)$
{ fix $T :: real$
assume $T \geq 0$
let $?f' = \lambda(t, x). indicator \{-T <..< T\} t *R ?f t x$
{ fix x
have 1: complex-interval-lebesgue-integrable lborel $u v (\lambda t. ?f t x)$ **for** $u v :: real$
using Levy-Inversion-aux2[of $x - b x - a$]
apply (simp add: interval-lebesgue-integrable-def del: times-divide-eq-left)
apply (intro integrableI-bounded-set-indicator[**where** $B=b - a$] conjI impI)
apply (auto intro!: AE-I [of - - {0}] simp: assms)
done
have $(CLBINT t. ?f' (t, x)) = (CLBINT t=-T..T. ?f t x)$
using $\langle T \geq 0 \rangle$ **by** (simp add: interval-lebesgue-integral-def)
also have ... = $(CLBINT t=-T..(0 :: real). ?f t x) + (CLBINT t=(0 :: real)..T. ?f t x)$
(is - = - + $?t$)
using 1 **by** (intro interval-integral-sum[symmetric]) (simp add: min-absorb1 max-absorb2 $\langle T \geq 0 \rangle$)
also have $(CLBINT t=-T..(0 :: real). ?f t x) = (CLBINT t=(0::real)..T. ?f (-t) x)$
by (subst interval-integral-reflect) auto
also have ... + $?t = (CLBINT t=(0::real)..T. ?f (-t) x + ?f t x)$
using 1
by (intro interval-lebesgue-integral-add(2) [symmetric] interval-integrable-mirror[THEN iffD2]) simp-all
also have ... = $(CLBINT t=(0::real)..T. ((iexp(t * (x - a)) - iexp (-(t * (x - a)))) - (iexp(t * (x - b)) - iexp (-(t * (x - b))))) / (i * t))$
using $\langle T \geq 0 \rangle$ **by** (intro interval-integral-cong) (auto simp add: divide-simps)
also have ... = $(CLBINT t=(0::real)..T. complex-of-real(2 * (sin (t * (x - a)) / t) - 2 * (sin (t * (x - b)) / t)))$
using $\langle T \geq 0 \rangle$
apply (intro interval-integral-cong)
apply (simp add: field-simps cis.ctr Im-divide Re-divide Im-exp Re-exp complex-eq-iff)
unfolding minus-diff-eq[symmetric, of $y * x y * a$ **for** $y a$] sin-minus cos-minus
apply (simp add: field-simps power2-eq-square)
done

```

also have ... = complex-of-real (LBINT t=(0::real)..T.
  2 * (sin (t * (x - a)) / t) - 2 * (sin (t * (x - b)) / t))
by (rule interval-lebesgue-integral-of-real)
also have ... = complex-of-real (2 * (sgn (x - a) * Si (T * abs (x - a)) -
  sgn (x - b) * Si (T * abs (x - b))))
apply (subst interval-lebesgue-integral-diff)
apply (rule interval-lebesgue-integrable-mult-right, rule integrable-sinc) +
apply (subst interval-lebesgue-integral-mult-right) +
apply (simp add: zero-ereal-def[symmetric] LBINT-I0c-sin-scale-divide[OF
(T ≥ 0)])
done
finally have (CLBINT t. ?f' (t, x)) =
  2 * (sgn (x - a) * Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x
- b))) .
} note main-eq = this
have (CLBINT t=-T..T. ?F t * φ t) =
  (CLBINT t. (CLINT x | M. ?F t * iexp (t * x) * indicator {-T<..using (T ≥ 0) unfolding φ-def char-def interval-lebesgue-integral-def
by (auto split: split-indicator intro!: Bochner-Integration.integral-cong)
also have ... = (CLBINT t. (CLINT x | M. ?f' (t, x)))
by (auto intro!: Bochner-Integration.integral-cong simp: field-simps exp-diff
exp-minus split: split-indicator)
also have ... = (CLINT x | M. (CLBINT t. ?f' (t, x)))
proof (intro P.Fubini-integral [symmetric] integrableI-bounded-set [where B=b
- a])
show emeasure (lborel ⊗M M) ({- T<..using (T ≥ 0)
by (subst emeasure-pair-measure-Times)
(auto simp: ennreal-mult-less-top not-less top-unique)
show AE x ∈ {- T<..M M. cmod (case x of (t,
x) ⇒ ?f' (t, x)) ≤ b - a
using Levy-Inversion-aux2[of x - b x - a for x] (a ≤ b)
by (intro AE-I [of - - {0} × UNIV]) (force simp: emeasure-pair-measure-Times) +
qed (auto split: split-indicator split-indicator-asm)
also have ... = (CLINT x | M. (complex-of-real (2 * (sgn (x - a) *
Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b))))))
using main-eq by (intro Bochner-Integration.integral-cong, auto)
also have ... = complex-of-real (LINT x | M. (2 * (sgn (x - a) *
Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b))))))
by (rule integral-complex-of-real)
finally have (CLBINT t=-T..T. ?F t * φ t) =
  complex-of-real (LINT x | M. (2 * (sgn (x - a) *
Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))))) .
} note main-eq2 = this

have (λT :: nat. LINT x | M. (2 * (sgn (x - a) *
Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))) →
(LINT x | M. 2 * pi * indicator {a<..b} x)
proof (rule integral-dominated-convergence [where w=λx. 4 * B])

```



```

show integrable  $M$  ( $\lambda x. 4 * B$ )
  by (rule integrable-const-bound [of - 4 * B]) auto
next
  let ?S =  $\lambda n::nat. \lambda x. \text{sgn } (x - a) * Si (n * |x - a|) - \text{sgn } (x - b) * Si (n * |x - b|)$ 
  { fix  $n x$ 
    have  $\text{norm } (?S n x) \leq \text{norm } (\text{sgn } (x - a) * Si (n * |x - a|)) + \text{norm } (\text{sgn } (x - b) * Si (n * |x - b|))$ 
    by (rule norm-triangle-ineq4)
    also have  $\dots \leq B + B$ 
    using  $Bprop$  by (intro add-mono) (auto simp: abs-mult abs-sgn-eq)
    finally have  $\text{norm } (2 * ?S n x) \leq 4 * B$ 
    by simp }
  then show  $\bigwedge n. AE x \text{ in } M. \text{norm } (2 * ?S n x) \leq 4 * B$ 
  by auto
  have  $AE x \text{ in } M. x \neq a \wedge AE x \text{ in } M. x \neq b$ 
  using prob-eq-0[of {a}] prob-eq-0[of {b}]  $\langle \mu \{a\} = 0 \rangle \langle \mu \{b\} = 0 \rangle$  by (auto simp:  $\mu$ -def)
  then show  $AE x \text{ in } M. (\lambda n. 2 * ?S n x) \longrightarrow 2 * pi * \text{indicator } \{a <.. b\} x$ 
  proof eventually-elim
    fix  $x$  assume  $x: x \neq a \wedge x \neq b$ 
    then have  $(\lambda n. 2 * (\text{sgn } (x - a) * Si (|x - a| * n) - \text{sgn } (x - b) * Si (|x - b| * n)))$ 
       $\longrightarrow 2 * (\text{sgn } (x - a) * (pi / 2) - \text{sgn } (x - b) * (pi / 2))$ 
    by (intro tendsto-intros filterlim-compose[OF Si-at-top] filterlim-tendsto-pos-mult-at-top[OF tendsto-const] filterlim-real-sequentially)
    auto
    also have  $(\lambda n. 2 * (\text{sgn } (x - a) * Si (|x - a| * n) - \text{sgn } (x - b) * Si (|x - b| * n))) = (\lambda n. 2 * ?S n x)$ 
    by (auto simp: ac-simps)
    also have  $2 * (\text{sgn } (x - a) * (pi / 2) - \text{sgn } (x - b) * (pi / 2)) = 2 * pi * \text{indicator } \{a <.. b\} x$ 
    using  $x \langle a \leq b \rangle$  by (auto split: split-indicator)
    finally show  $(\lambda n. 2 * ?S n x) \longrightarrow 2 * pi * \text{indicator } \{a <.. b\} x$  .
  qed
qed simp-all
also have  $(LINT x \mid M. 2 * pi * \text{indicator } \{a <.. b\} x) = 2 * pi * \mu \{a <.. b\}$ 
by (simp add:  $\mu$ -def)
finally have  $(\lambda T. LINT x \mid M. (2 * (\text{sgn } (x - a) * Si (T * \text{abs } (x - a)) - \text{sgn } (x - b) * Si (T * \text{abs } (x - b)))) \longrightarrow 2 * pi * \mu \{a <.. b\}$  .
with main-eq2 show ?thesis
by (auto intro!: tendsto-eq-intros)
qed

```

theorem Levy-uniqueness:

fixes $M1 M2 :: \text{real measure}$

assumes real-distribution $M1$ real-distribution $M2$ **and**

$\text{char } M1 = \text{char } M2$

shows $M1 = M2$
proof –
interpret $M1$: *real-distribution* $M1$ **by** (*rule assms*)
interpret $M2$: *real-distribution* $M2$ **by** (*rule assms*)
have *countable* ($\{x. \text{measure } M1 \{x\} \neq 0\} \cup \{x. \text{measure } M2 \{x\} \neq 0\}$)
by (*intro countable-Un* $M2.\text{countable-support } M1.\text{countable-support}$)
then have *count*: *countable* $\{x. \text{measure } M1 \{x\} \neq 0 \vee \text{measure } M2 \{x\} \neq 0\}$
by (*simp add: Un-def*)

have *cdf* $M1 = \text{cdf } M2$
proof (*rule ext*)
fix x
let $?D = \lambda x. |\text{cdf } M1 \ x - \text{cdf } M2 \ x|$

{ **fix** $\varepsilon :: \text{real}$
assume $\varepsilon > 0$
have ($?D \longrightarrow 0$) *at-bot*
using $\langle \text{cdf } M1 \longrightarrow 0 \rangle \text{ at-bot} \langle \text{cdf } M2 \longrightarrow 0 \rangle \text{ at-bot}$ **by** (*intro tendsto-eq-intros*) *auto*
then have *eventually* $(\lambda y. ?D \ y < \varepsilon / 2 \wedge y \leq x)$ *at-bot*
using $\langle \varepsilon > 0 \rangle$ **by** (*simp only: tendsto-iff dist-real-def diff-0-right eventually-conj eventually-le-at-bot abs-idempotent*)
then obtain M **where** $\bigwedge y. y \leq M \implies ?D \ y < \varepsilon / 2 \ M \leq x$
unfolding *eventually-at-bot-linorder* **by** *auto*
with *open-minus-countable*[*OF count, of* $\{.. < M\}$] **obtain** a **where**
measure $M1 \ \{a\} = 0$ *measure* $M2 \ \{a\} = 0$ $a < M$ $a \leq x$ **and** $1: ?D \ a < \varepsilon / 2$
by *auto*

have ($?D \longrightarrow ?D \ x$) (*at-right* x)
using $M1.\text{cdf-is-right-cont} \ [of \ x] \ M2.\text{cdf-is-right-cont} \ [of \ x]$
by (*intro tendsto-intros*) (*auto simp add: continuous-within*)
then have *eventually* $(\lambda y. |?D \ y - ?D \ x| < \varepsilon / 2)$ (*at-right* x)
using $\langle \varepsilon > 0 \rangle$ **by** (*simp only: tendsto-iff dist-real-def eventually-conj eventually-at-right-less*)
then obtain N **where** $N > x \ \bigwedge y. x < y \implies y < N \implies |?D \ y - ?D \ x| < \varepsilon / 2$
by (*auto simp add: eventually-at-right*[*OF less-add-one*])
with *open-minus-countable*[*OF count, of* $\{x <.. < N\}$] **obtain** b **where** $x < b < N$
measure $M1 \ \{b\} = 0$ *measure* $M2 \ \{b\} = 0$ **and** $2: |?D \ b - ?D \ x| < \varepsilon / 2$
by (*auto simp: abs-minus-commute*)
from $\langle a \leq x \rangle \langle x < b \rangle$ **have** $a < b$ $a \leq b$ **by** *auto*

from $\langle \text{char } M1 = \text{char } M2 \rangle$
 $M1.\text{Levy-Inversion} \ [OF \ \langle a \leq b \rangle \langle \text{measure } M1 \ \{a\} = 0 \rangle \langle \text{measure } M1 \ \{b\} = 0 \rangle]$
 $M2.\text{Levy-Inversion} \ [OF \ \langle a \leq b \rangle \langle \text{measure } M2 \ \{a\} = 0 \rangle \langle \text{measure } M2 \ \{b\} = 0 \rangle]$

```

have complex-of-real (measure M1 {a<..b}) = complex-of-real (measure M2
{a<..b})
  by (intro LIMSEQ-unique) auto
then have ?D a = ?D b
unfolding of-real-eq-iff M1.cdf-diff-eq [OF ⟨a < b⟩, symmetric] M2.cdf-diff-eq
[OF ⟨a < b⟩, symmetric] by simp
then have ?D x = |( ?D b - ?D x) - ?D a|
  by simp
also have ... ≤ |?D b - ?D x| + |?D a|
  by (rule abs-triangle-ineq4)
also have ... ≤ ε / 2 + ε / 2
  using 1 2 by (intro add-mono) auto
finally have ?D x ≤ ε by simp }
then show cdf M1 x = cdf M2 x
  by (metis abs-le-zero-iff dense-ge eq-iff-diff-eq-0)
qed
thus ?thesis
by (rule cdf-unique [OF ⟨real-distribution M1⟩ ⟨real-distribution M2⟩])
qed

```

20.2 The Levy continuity theorem

theorem *levy-continuity1:*

```

fixes M :: nat ⇒ real measure and M' :: real measure
assumes ∧n. real-distribution (M n) real-distribution M' weak-conv-m M M'
shows (λn. char (M n) t) ⟶ char M' t
unfolding char-def using assms by (rule weak-conv-imp-integral-bdd-continuous-conv)
auto

```

theorem *levy-continuity:*

```

fixes M :: nat ⇒ real measure and M' :: real measure
assumes real-distr-M : ∧n. real-distribution (M n)
  and real-distr-M': real-distribution M'
  and char-conv: ∧t. (λn. char (M n) t) ⟶ char M' t
shows weak-conv-m M M'

```

proof –

```

interpret Mn: real-distribution M n for n by fact
interpret M': real-distribution M' by fact

```

```

have *: ∧u x. u > 0 ⟹ x ≠ 0 ⟹ (CLBINT t;{-u..u}. 1 - iexp (t * x)) =
  2 * (u - sin (u * x) / x)

```

proof –

```

fix u :: real and x :: real
assume u > 0 and x ≠ 0
hence (CLBINT t;{-u..u}. 1 - iexp (t * x)) = (CLBINT t=-u..u. 1 - iexp
(t * x))
  by (subst interval-integral-Icc, auto)
also have ... = (CLBINT t=-u..0. 1 - iexp (t * x)) + (CLBINT t=0..u. 1
- iexp (t * x))

```

```

using ⟨ $u > 0$ ⟩
apply (subst interval-integral-sum)
apply (simp add: min-absorb1 min-absorb2 max-absorb1 max-absorb2)
apply (rule interval-integrable-isCont)
apply auto
done
also have ... = (CLBINT  $t=ereal\ 0..u.$   $1 - iexp\ (t * -x)$ ) + (CLBINT
 $t=ereal\ 0..u.$   $1 - iexp\ (t * x)$ )
apply (subgoal-tac  $0 = eréal\ 0,$  erule ssubst)
by (subst interval-integral-reflect, auto)
also have ... = (LBINT  $t=ereal\ 0..u.$   $2 - 2 * cos\ (t * x)$ )
apply (subst interval-lebesgue-integral-add (2) [symmetric])
apply ((rule interval-integrable-isCont, auto)+) [2]
unfolding exp-Euler cos-of-real
apply (simp add: of-real-mult interval-lebesgue-integral-of-real[symmetric])
done
also have ... =  $2 * u - 2 * sin\ (u * x) / x$ 
by (subst interval-lebesgue-integral-diff)
      (auto intro!: interval-integrable-isCont
        simp: interval-lebesgue-integral-of-real integral-cos [OF ⟨ $x \neq 0$ ⟩]
        mult.commute[of - x])
finally show (CLBINT  $t:\{-u..u\}.$   $1 - iexp\ (t * x)$ ) =  $2 * (u - sin\ (u * x) / x)$ 
by (simp add: field-simps)
qed
have main-bound:  $\bigwedge u\ n.$   $u > 0 \implies Re\ (CLBINT\ t:\{-u..u\}.$   $1 - char\ (M\ n)$ 
 $t) \geq$ 
 $u * measure\ (M\ n)\ \{x. abs\ x \geq 2 / u\}$ 
proof -
fix  $u :: real$  and  $n$ 
assume  $u > 0$ 
interpret  $P:$  pair-sigma-finite  $M\ n$  lborel ..

have Mn1 [simp]:  $measure\ (M\ n)\ UNIV = 1$  by (metis Mn.prob-space Mn.space-eq-univ)

have Mn2 [simp]:  $\bigwedge x.$  complex-integrable  $(M\ n)\ (\lambda t. exp\ (i * complex-of-real\ (x * t)))$ 
by (rule Mn.integrable-const-bound [where  $B = 1$ ], auto)
have Mn3: set-integrable  $(M\ n \otimes_M\ lborel)\ (UNIV \times \{-u..u\})\ (\lambda a. 1 - exp\ (i * complex-of-real\ (snd\ a * fst\ a)))$ 
using ⟨ $0 < u$ ⟩
by (intro integrableI-bounded-set-indicator [where  $B=2$ ])
      (auto simp: lborel.emeasure-pair-measure-Times ennreal-mult-less-top not-less-top-unique
        split: split-indicator
        intro!: order-trans [OF norm-triangle-ineq4])
have (CLBINT  $t:\{-u..u\}.$   $1 - char\ (M\ n)\ t) =$ 
 $(CLBINT\ t:\{-u..u\}.$   $(CLINT\ x \mid M\ n.$   $1 - iexp\ (t * x)))$ 
unfolding char-def by (rule set-lebesgue-integral-cong, auto simp del: of-real-mult)

```

also have ... = (CLBINT t. (CLINT x | M n. indicator {-u..u} t *_R (1 - iexp (t * x))))
by (rule Bochner-Integration.integral-cong) (auto split: split-indicator)
also have ... = (CLINT x | M n. (CLBINT t:{-u..u}. 1 - iexp (t * x)))
using Mn3 **by** (subst P.Fubini-integral) (auto simp: indicator-times split-beta')
also have ... = (CLINT x | M n. (if x = 0 then 0 else 2 * (u - sin (u * x) / x)))
using ⟨u > 0⟩ **by** (intro Bochner-Integration.integral-cong, auto simp add: * simp del: of-real-mult)
also have ... = (LINT x | M n. (if x = 0 then 0 else 2 * (u - sin (u * x) / x)))
by (rule integral-complex-of-real)
finally have Re (CLBINT t:{-u..u}. 1 - char (M n) t) =
(LINT x | M n. (if x = 0 then 0 else 2 * (u - sin (u * x) / x))) **by** simp
also have ... ≥ (LINT x : {x. abs x ≥ 2 / u} | M n. u)
proof -
have complex-integrable (M n) (λx. CLBINT t:{-u..u}. 1 - iexp (snd (x, t) * fst (x, t)))
using Mn3 **by** (intro P.integrable-fst) (simp add: indicator-times split-beta')
hence complex-integrable (M n) (λx. if x = 0 then 0 else 2 * (u - sin (u * x) / x))
using ⟨u > 0⟩ **by** (subst integrable-cong) (auto simp add: * simp del: of-real-mult)
hence **: integrable (M n) (λx. if x = 0 then 0 else 2 * (u - sin (u * x) / x))
unfolding complex-of-real-integrable-eq .
have 2 * sin x ≤ x **if** 2 ≤ x **for** x :: real
by (rule order-trans[OF - ⟨2 ≤ x⟩]) auto
moreover have x ≤ 2 * sin x **if** x ≤ - 2 **for** x :: real
by (rule order-trans[OF ⟨x ≤ - 2⟩]) auto
moreover have x < 0 ⇒ x ≤ sin x **for** x :: real
using sin-x-le-x[of -x] **by** simp
ultimately show ?thesis
using ⟨u > 0⟩
by (intro integral-mono [OF - **])
(auto simp: divide-simps sin-x-le-x mult.commute[of u] mult-neg-pos top-unique less-top[symmetric] split: split-indicator)
qed
also (xtrans) **have** (LINT x : {x. abs x ≥ 2 / u} | M n. u) =
u * measure (M n) {x. abs x ≥ 2 / u}
by (simp add: Mn.emmeasure-eq-measure)
finally show Re (CLBINT t:{-u..u}. 1 - char (M n) t) ≥ u * measure (M n) {x. abs x ≥ 2 / u} .
qed
have tight-aux: ∧ε. ε > 0 ⇒ ∃ a b. a < b ∧ (∀ n. 1 - ε < measure (M n) {a<..b})
proof -

```

fix  $\varepsilon :: \text{real}$ 
assume  $\varepsilon > 0$ 
note  $M'.\text{isCont-char}$  [of 0]
hence  $\exists d > 0. \forall t. \text{abs } t < d \longrightarrow \text{cmod } (\text{char } M' t - 1) < \varepsilon / 4$ 
  apply (subst (asm) continuous-at-eps-delta)
  apply (drule-tac  $x = \varepsilon / 4$  in spec)
  using ( $\varepsilon > 0$ ) by (auto simp add: dist-real-def dist-complex-def  $M'.\text{char-zero}$ )
then obtain  $d$  where  $d > 0 \wedge (\forall t. (\text{abs } t < d \longrightarrow \text{cmod } (\text{char } M' t - 1) < \varepsilon / 4))$  ..
  hence  $d0: d > 0$  and  $d1: \bigwedge t. \text{abs } t < d \implies \text{cmod } (\text{char } M' t - 1) < \varepsilon / 4$ 
by auto
  have  $1: \bigwedge x. \text{cmod } (1 - \text{char } M' x) \leq 2$ 
  by (rule order-trans [OF norm-triangle-ineq4], auto simp add:  $M'.\text{cmod-char-le-1}$ )
  then have  $2: \bigwedge u v. \text{complex-set-integrable lborel } \{u..v\} (\lambda x. 1 - \text{char } M' x)$ 
  by (intro integrableI-bounded-set-indicator[where  $B=2$ ]) (auto simp: emeasure-lborel-Icc-eq)
  have  $3: \bigwedge u v. \text{set-integrable lborel } \{u..v\} (\lambda x. \text{cmod } (1 - \text{char } M' x))$ 
  by (intro borel-integrable-compact[OF compact-Icc] continuous-at-imp-continuous-on
    continuous-intros ballI  $M'.\text{isCont-char}$  continuous-intros)
  have  $\text{cmod } (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t) \leq \text{LBINT } t:\{-d/2..d/2\}. \text{cmod } (1 - \text{char } M' t)$ 
  using integral-norm-bound[of -  $\lambda x. \text{indicator } \{u..v\} x *_R (1 - \text{char } M' x)$ ]
for  $u v$  by simp
  also have  $4: \dots \leq \text{LBINT } t:\{-d/2..d/2\}. \varepsilon / 4$ 
  apply (rule integral-mono [OF 3])
  apply (simp add: emeasure-lborel-Icc-eq)
  apply (case-tac  $x \in \{-d/2..d/2\}$ )
  apply auto
  apply (subst norm-minus-commute)
  apply (rule less-imp-le)
  apply (rule d1 [simplified])
  using  $d0$  apply auto
  done
also from  $d0$   $4$  have  $\dots = d * \varepsilon / 4$ 
  by simp
  finally have  $\text{bound: cmod } (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t) \leq d * \varepsilon / 4$  .
  have  $\text{cmod } (1 - \text{char } (M n) x) \leq 2$  for  $n x$ 
  by (rule order-trans [OF norm-triangle-ineq4], auto simp add:  $Mn.\text{cmod-char-le-1}$ )
  then have  $(\lambda n. \text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) \longrightarrow (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)$ 
  apply (intro integral-dominated-convergence[where  $w = \lambda x. \text{indicator } \{-d/2..d/2\} x *_R 2$ ])
  apply (auto intro!: char-conv tendsto-intros
    simp: emeasure-lborel-Icc-eq
    split: split-indicator)
  done
hence eventually  $(\lambda n. \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4)$  sequentially
  using  $d0$  ( $\varepsilon > 0$ ) apply (subst (asm) tendsto-iff)

```

```

    by (subst (asm) dist-complex-def, drule spec, erule mp, auto)
  hence  $\exists N. \forall n \geq N. \text{cmod} ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) -$ 
     $(\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4$  by (simp add:
eventually-sequentially)
  then guess  $N ..$ 
  hence  $N: \bigwedge n. n \geq N \implies \text{cmod} ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n)$ 
 $t) -$ 
     $(\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4$  by auto
  { fix  $n$ 
    assume  $n \geq N$ 
    have  $\text{cmod} (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) =$ 
       $\text{cmod} ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t))$ 
       $+ (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)$  by simp
    also have  $\dots \leq \text{cmod} ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) -$ 
       $(\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)) + \text{cmod}(\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' t)$ 
      by (rule norm-triangle-ineq)
    also have  $\dots < d * \varepsilon / 4 + d * \varepsilon / 4$ 
      by (rule add-less-le-mono [OF N [OF  $\langle n \geq N \rangle$ ] bound])
    also have  $\dots = d * \varepsilon / 2$  by auto
    finally have  $\text{cmod} (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t) < d * \varepsilon / 2$  .
    hence  $d * \varepsilon / 2 > \text{Re} (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M n) t)$ 
      by (rule order-le-less-trans [OF complex-Re-le-cmod])
    hence  $d * \varepsilon / 2 > \text{Re} (\text{CLBINT } t:\{-(d/2)..d/2\}. 1 - \text{char } (M n) t)$  (is -
 $> ?lhs$ ) by simp
    also have  $?lhs \geq (d / 2) * \text{measure } (M n) \{x. \text{abs } x \geq 2 / (d / 2)\}$ 
      using  $d0$  by (intro main-bound, simp)
    finally (xtrans) have  $d * \varepsilon / 2 > (d / 2) * \text{measure } (M n) \{x. \text{abs } x \geq 2 / (d / 2)\}$  .
    with  $d0 \langle \varepsilon > 0 \rangle$  have  $\varepsilon > \text{measure } (M n) \{x. \text{abs } x \geq 2 / (d / 2)\}$  by (simp
add: field-simps)
    hence  $\varepsilon > 1 - \text{measure } (M n) (\text{UNIV} - \{x. \text{abs } x \geq 2 / (d / 2)\})$ 
      apply (subst Mn.borel-UNIV [symmetric])
      by (subst Mn.prob-compl, auto)
    also have  $\text{UNIV} - \{x. \text{abs } x \geq 2 / (d / 2)\} = \{x. -(4 / d) < x \wedge x < (4 / d)\}$ 
      using  $d0$  apply (auto simp add: field-simps)

    apply (case-tac  $x \geq 0$ , auto simp add: field-simps)
    apply (subgoal-tac  $0 \leq x * d$ , arith, rule mult-nonneg-nonneg, auto)
    apply (case-tac  $x \geq 0$ , auto simp add: field-simps)
    apply (subgoal-tac  $x * d \leq 0$ , arith)
    apply (rule mult-nonpos-nonneg, auto)
    by (case-tac  $x \geq 0$ , auto simp add: field-simps)
    finally have  $\text{measure } (M n) \{x. -(4 / d) < x \wedge x < (4 / d)\} > 1 - \varepsilon$ 
      by auto
  } note 6 = this

```

```

{ fix n :: nat
  have *: (UN (k :: nat). {− real k<..real k}) = UNIV
  by (auto, metis leI le-less-trans less-imp-le minus-less-iff reals-Archimedean2)
  have (λk. measure (M n) {− real k<..real k}) ⟶
    measure (M n) (UN (k :: nat). {− real k<..real k})
  by (rule Mn.finite-Lim-measure-incseq, auto simp add: incseq-def)
  hence (λk. measure (M n) {− real k<..real k}) ⟶ 1
  using Mn.prob-space unfolding * Mn.borel-UNIV by simp
  hence eventually (λk. measure (M n) {− real k<..real k} > 1 − ε) sequentially
    apply (elim order-tendstoD (1))
    using ⟨ε > 0⟩ by auto
} note 7 = this
{ fix n :: nat
  have eventually (λk. ∀ m < n. measure (M m) {− real k<..real k} > 1 − ε)
sequentially
  (is ?P n)
  proof (induct n)
    case (Suc n) with 7[of n] show ?case
      by eventually-elim (auto simp add: less-Suc-eq)
  qed simp
} note 8 = this
from 8 [of N] have ∃ K :: nat. ∀ k ≥ K. ∀ m < N. 1 − ε <
  Sigma-Algebra.measure (M m) {− real k<..real k}
  by (auto simp add: eventually-sequentially)
  hence ∃ K :: nat. ∀ m < N. 1 − ε < Sigma-Algebra.measure (M m) {− real
K<..real K} by auto
then obtain K :: nat where
  ∀ m < N. 1 − ε < Sigma-Algebra.measure (M m) {− real K<..real K} ..
  hence K: ∧m. m < N ⟹ 1 − ε < Sigma-Algebra.measure (M m) {− real
K<..real K}
  by auto
let ?K' = max K (4 / d)
have −?K' < ?K' ∧ (∀ n. 1 − ε < measure (M n) {−?K'<..?K'})
  using d0 apply auto
  apply (rule max.strict-coboundedI2, auto)
proof −
  fix n
  show 1 − ε < measure (M n) {− max (real K) (4 / d)<..max (real K) (4
/ d)}
    apply (case-tac n < N)
    apply (rule order-less-le-trans)
    apply (erule K)
    apply (rule Mn.finite-measure-mono, auto)
    apply (rule order-less-le-trans)
    apply (rule 6, erule leI)
    by (rule Mn.finite-measure-mono, auto)
qed
thus ∃ a b. a < b ∧ (∀ n. 1 − ε < measure (M n) {a<..b}) by (intro exI)
qed

```



```

have tight: tight M
  by (auto simp: tight-def intro: assms tight-aux)
show ?thesis
proof (rule tight-subseq-weak-converge [OF real-distr-M real-distr-M' tight])
  fix s  $\nu$ 
  assume s: strict-mono (s :: nat  $\Rightarrow$  nat)
  assume nu: weak-conv-m (M  $\circ$  s)  $\nu$ 
  assume *: real-distribution  $\nu$ 
  have 2:  $\bigwedge n$ . real-distribution ((M  $\circ$  s) n) unfolding comp-def by (rule assms)
  have 3:  $\bigwedge t$ . ( $\lambda n$ . char ((M  $\circ$  s) n) t)  $\longrightarrow$  char  $\nu$  t by (intro levy-continuity1
[OF 2 * nu])
  have 4:  $\bigwedge t$ . ( $\lambda n$ . char ((M  $\circ$  s) n) t) = (( $\lambda n$ . char (M n) t)  $\circ$  s) by (rule ext,
simp)
  have 5:  $\bigwedge t$ . ( $\lambda n$ . char ((M  $\circ$  s) n) t)  $\longrightarrow$  char M' t
    by (subst 4, rule LIMSEQ-subseq-LIMSEQ [OF - s], rule assms)
  hence char  $\nu$  = char M' by (intro ext, intro LIMSEQ-unique [OF 3 5])
  hence  $\nu$  = M' by (rule Levy-uniqueness [OF * (real-distribution M')])
  thus weak-conv-m (M  $\circ$  s) M'
    by (elim subst) (rule nu)
qed
qed
end

```

21 The Central Limit Theorem

theory *Central-Limit-Theorem*

imports *Levy*

begin

theorem (in *prob-space*) *central-limit-theorem-zero-mean*:

fixes $X :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

and $\mu :: \text{real measure}$

and $\sigma :: \text{real}$

and $S :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes X -indep: *indep-vars* (λi . borel) X UNIV

and X -integrable: $\bigwedge n$. integrable M (X n)

and X -mean-0: $\bigwedge n$. expectation (X n) = 0

and σ -pos: $\sigma > 0$

and X -square-integrable: $\bigwedge n$. integrable M (λx . (X n x)²)

and X -variance: $\bigwedge n$. variance (X n) = σ^2

and X -distrib: $\bigwedge n$. distr M borel (X n) = μ

defines S n $\equiv \lambda x$. $\sum_{i < n} X$ i x

shows weak-conv-m (λn . distr M borel (λx . S n x / sqrt ($n * \sigma^2$))) *std-normal-distribution*

proof –

let ? S' = λn x. S n x / sqrt ($\text{real } n * \sigma^2$)

define φ **where** φ n = char (distr M borel (? S' n)) **for** n

define ψ **where** ψ n t = char μ (t / sqrt ($\sigma^2 * n$)) **for** n t

```

have  $X$ -rv [simp, measurable]:  $\bigwedge n$ . random-variable borel ( $X$   $n$ )
  using  $X$ -indep unfolding indep-vars-def2 by simp
interpret  $\mu$ : real-distribution  $\mu$ 
  by (subst  $X$ -distrib [symmetric, of 0], rule real-distribution-distr, simp)

have  $\mu$ -integrable [simp]: integrable  $\mu$  ( $\lambda x$ .  $x$ )
  and  $\mu$ -mean-integrable [simp]:  $\mu$ .expectation ( $\lambda x$ .  $x$ ) = 0
  and  $\mu$ -square-integrable [simp]: integrable  $\mu$  ( $\lambda x$ .  $x^2$ )
  and  $\mu$ -variance [simp]:  $\mu$ .expectation ( $\lambda x$ .  $x^2$ ) =  $\sigma^2$ 
  using assms by (simp-all add:  $X$ -distrib [symmetric, of 0] integrable-distr-eq
integral-distr)

have main:  $\forall_F n$  in sequentially.
  cmod ( $\varphi$   $n$   $t$  - (1 + (- $t^2$ ) / 2) /  $n$ )  $\wedge n$   $\leq$ 
   $t^2$  / (6 *  $\sigma^2$ ) * (LINT  $x$  |  $\mu$ . min (6 *  $x^2$ ) (| $t$  / sqrt ( $\sigma^2$  *  $n$ )| * | $x$ |  $\wedge$  3)) for  $t$ 
proof (rule eventually-sequentiallyI)
  fix  $n$  :: nat
  assume  $n \geq$  nat (ceiling ( $t^2$  / 4))
  hence  $n$ :  $n \geq t^2$  / 4 by (subst nat-ceiling-le-eq [symmetric])
  let ? $t$  =  $t$  / sqrt ( $\sigma^2$  *  $n$ )

  define  $\psi'$  where  $\psi' n i$  = char (distr  $M$  borel ( $\lambda x$ .  $X i x$  / sqrt ( $\sigma^2$  *  $n$ ))) for
 $n i$ 
  have *:  $\bigwedge n i t$ .  $\psi' n i t$  =  $\psi n t$ 
    unfolding  $\psi$ -def  $\psi'$ -def char-def
    by (subst  $X$ -distrib [symmetric]) (auto simp: integral-distr)

  have  $\varphi n t$  = char (distr  $M$  borel ( $\lambda x$ .  $\sum i < n$ .  $X i x$  / sqrt ( $\sigma^2$  * real  $n$ )))  $t$ 
    by (auto simp:  $\varphi$ -def  $S$ -def sum-divide-distrib ac-simps)
  also have ... = ( $\prod i < n$ .  $\psi' n i t$ )
    unfolding  $\psi'$ -def
    apply (rule char-distr-sum)
    apply (rule indep-vars-compose2[where  $X=X$ ])
    apply (rule indep-vars-subset)
    apply (rule  $X$ -indep)
    apply auto
    done
  also have ... = ( $\psi n t$ )  $\wedge n$ 
    by (auto simp add: * prod-constant)
  finally have  $\varphi$ -eq:  $\varphi n t$  = ( $\psi n t$ )  $\wedge n$  .

  have norm ( $\psi n t$  - (1 - ? $t^2$  *  $\sigma^2$  / 2))  $\leq$  ? $t^2$  / 6 * (LINT  $x$  |  $\mu$ . min (6 *
 $x^2$ ) (|? $t$ | * | $x$ |  $\wedge$  3))
    unfolding  $\psi$ -def by (rule  $\mu$ .char-approx3, auto)
  also have ? $t^2$  *  $\sigma^2$  =  $t^2$  /  $n$ 
    using  $\sigma$ -pos by (simp add: power-divide)
  also have  $t^2$  /  $n$  / 2 = ( $t^2$  / 2) /  $n$ 
    by simp

```

finally have $∗∗$: $norm (\psi \ n \ t - (1 + (-t^2) / 2) / n) \leq$
 $?t^2 / 6 * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| ^ 3))$ **by** *simp*

have $norm (\varphi \ n \ t - (complex-of-real (1 + (-t^2) / 2) / n) ^ n) \leq$
 $n * norm (\psi \ n \ t - (complex-of-real (1 + (-t^2) / 2) / n))$
using *n*
by (*auto intro!*: *norm-power-diff* $\mu.cmod-char-le-1$ *abs-leI*
simp del: *of-real-diff simp: of-real-diff[symmetric] divide-le-eq* φ -*eq*

ψ -def)

also have $\dots \leq n * (?t^2 / 6 * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| ^ 3)))$
by (*rule mult-left-mono* [*OF* $∗∗$], *simp*)
also have $\dots = (t^2 / (6 * \sigma^2) * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| ^ 3)))$
using σ -*pos* **by** (*simp add*: *field-simps min-absorb2*)
finally show $norm (\varphi \ n \ t - (1 + (-t^2) / 2) / n) ^ n \leq$
 $(t^2 / (6 * \sigma^2) * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| ^ 3)))$
by *simp*

qed

show *?thesis*

proof (*rule levy-continuity*)

fix *t*

let $?t = \lambda n. t / \text{sqrt} (\sigma^2 * n)$

have $\bigwedge x. (\lambda n. \min (6 * x^2) (|t| * |x| ^ 3 / |\text{sqrt} (\sigma^2 * \text{real } n)|)) \longrightarrow 0$
using σ -*pos*
by (*auto simp*: *real-sqrt-mult min-absorb2*
intro!: *tendsto-min[THEN tendsto-eq-rhs] sqrt-at-top[THEN filterlim-compose]*
filterlim-tendsto-pos-mult-at-top filterlim-at-top-imp-at-infinity
tendsto-divide-0 filterlim-real-sequentially)

then have $(\lambda n. LINT \ x|\mu. \min (6 * x^2) (|?t \ n| * |x| ^ 3)) \longrightarrow (LINT \ x|\mu.$
 $0)$
by (*intro integral-dominated-convergence* [**where** $w = \lambda x. 6 * x^2$]) *auto*
then have $∗$: $(\lambda n. t^2 / (6 * \sigma^2) * (LINT \ x|\mu. \min (6 * x^2) (|?t \ n| * |x| ^ 3)))$
 $\longrightarrow 0$
by (*simp only*: *integral-zero tendsto-mult-right-zero*)

have $(\lambda n. complex-of-real ((1 + (-t^2) / 2) / n) ^ n) \longrightarrow complex-of-real$
 $(\exp (-t^2) / 2)$
by (*rule isCont-tendsto-compose* [*OF* - *tendsto-exp-limit-sequentially*]) *auto*
then have $(\lambda n. \varphi \ n \ t) \longrightarrow complex-of-real (\exp (-t^2) / 2)$
by (*rule Lim-transform*) (*rule Lim-null-comparison* [*OF main* $∗$])
then show $(\lambda n. \text{char} (\text{distr } M \ \text{borel} \ (?S' \ n)) \ t) \longrightarrow \text{char } \text{std-normal-distribution}$
 t
by (*simp add*: φ -*def char-std-normal-distribution*)
qed (*auto intro!*: *real-dist-normal-dist simp: S-def*)
qed

theorem (*in prob-space*) *central-limit-theorem*:

fixes $X :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$
and $\mu :: \text{real measure}$

```

    and  $\sigma :: \text{real}$ 
    and  $S :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$ 
  assumes  $X\text{-indep}: \text{indep-vars } (\lambda i. \text{borel}) X \text{ UNIV}$ 
    and  $X\text{-integrable}: \bigwedge n. \text{integrable } M (X n)$ 
    and  $X\text{-mean}: \bigwedge n. \text{expectation } (X n) = m$ 
    and  $\sigma\text{-pos}: \sigma > 0$ 
    and  $X\text{-square-integrable}: \bigwedge n. \text{integrable } M (\lambda x. (X n x)^2)$ 
    and  $X\text{-variance}: \bigwedge n. \text{variance } (X n) = \sigma^2$ 
    and  $X\text{-distrib}: \bigwedge n. \text{distr } M \text{ borel } (X n) = \mu$ 
  defines  $X' i x \equiv X i x - m$ 
  shows  $\text{weak-conv-m } (\lambda n. \text{distr } M \text{ borel } (\lambda x. (\sum_{i < n}. X' i x) / \text{sqrt } (n * \sigma^2)))$ 
  std-normal-distribution
  proof (intro central-limit-theorem-zero-mean)
    show  $\text{indep-vars } (\lambda i. \text{borel}) X' \text{ UNIV}$ 
      unfolding  $X'\text{-def}[abs-def]$  using  $X\text{-indep}$  by (rule indep-vars-compose2) auto
    show  $\text{integrable } M (X' n) \text{ expectation } (X' n) = 0$  for  $n$ 
      using  $X\text{-integrable } X\text{-mean}$  by (auto simp:  $X'\text{-def}[abs-def]$  prob-space)
    show  $\sigma > 0 \text{ integrable } M (\lambda x. (X' n x)^2) \text{ variance } (X' n) = \sigma^2$  for  $n$ 
      using  $\langle 0 < \sigma \rangle X\text{-integrable } X\text{-mean } X\text{-square-integrable } X\text{-variance}$  unfolding
     $X'\text{-def}$ 
      by (auto simp: prob-space power2-diff)
    show  $\text{distr } M \text{ borel } (X' n) = \text{distr } \mu \text{ borel } (\lambda x. x - m)$  for  $n$ 
      unfolding  $X\text{-distrib}[of n, symmetric]$  using  $X\text{-integrable}$ 
      by (subst distr-distr) (auto simp:  $X'\text{-def}[abs-def]$  comp-def)
  qed
end

```

```

theory Discrete-Topology
imports HOL-Analysis.Analysis
begin

```

Copy of discrete types with discrete topology. This space is polish.

```

typedef 'a discrete = UNIV::'a set
morphisms of-discrete discrete
..

```

```

instantiation discrete :: (type) metric-space
begin

```

```

definition dist-discrete :: 'a discrete  $\Rightarrow$  'a discrete  $\Rightarrow$  real
  where  $\text{dist-discrete } n m = (\text{if } n = m \text{ then } 0 \text{ else } 1)$ 

```

```

definition uniformity-discrete :: ('a discrete  $\times$  'a discrete) filter where
  (uniformity::('a discrete  $\times$  'a discrete) filter) = (INF e:{0 <..}. principal {(x,
  y). dist x y < e})

```

```

definition open-discrete :: 'a discrete set  $\Rightarrow$  bool where

```

$(open :: 'a \text{ discrete set} \Rightarrow bool) \ U \longleftrightarrow (\forall x \in U. \text{eventually } (\lambda(x', y). x' = x \longrightarrow y \in U) \text{ uniformity})$

instance proof qed (*auto simp: uniformity-discrete-def open-discrete-def dist-discrete-def*
intro: exI[where x=1])
end

lemma open-discrete: *open (S :: 'a discrete set)*
unfolding *open-dist dist-discrete-def* **by** (*auto intro!: exI[of - 1 / 2]*)

instance discrete :: (type) complete-space
proof
fix *X :: nat \Rightarrow 'a discrete* **assume** *Cauchy X*
hence $\exists n. \forall m \geq n. X \ n = X \ m$
by (*force simp: dist-discrete-def Cauchy-def split: if-split-asm dest: spec[where x=1]*)
then guess *n ..*
thus *convergent X*
by (*intro convergentI[where L=X n] tendstoI eventually-sequentiallyI[of n]*
(simp add: dist-discrete-def))
qed

instance discrete :: (countable) countable
proof
have *inj (λc :: 'a discrete. to-nat (of-discrete c))*
by (*simp add: inj-on-def of-discrete-inject*)
thus $\exists f :: 'a \text{ discrete} \Rightarrow \text{nat}. \text{inj } f$ **by** *blast*
qed

instance discrete :: (countable) second-countable-topology
proof
let *?B = range (λn :: 'a discrete. {n})*
have $\bigwedge S. \text{generate-topology } ?B (\bigcup x \in S. \{x\})$
by (*intro generate-topology-Union*) (*auto intro: generate-topology.intros*)
then have *open = generate-topology ?B*
by (*auto intro!: ext simp: open-discrete*)
moreover have *countable ?B* **by** *simp*
ultimately show $\exists B :: 'a \text{ discrete set set}. \text{countable } B \wedge \text{open} = \text{generate-topology } B$ **by** *blast*
qed

instance discrete :: (countable) polish-space ..

end

22 Probability mass function

theory *Probability-Mass-Function*
imports

Giry-Monad
HOL-Library.Multiset

begin

lemma *AE-emeasure-singleton*:

assumes x : *emeasure* $M \{x\} \neq 0$ **and** ae : *AE* x *in* M . $P x$ **shows** $P x$

proof –

from x **have** x - M : $\{x\} \in$ *sets* M

by (*auto intro: emeasure-notin-sets*)

from ae **obtain** N **where** N : $\{x \in$ *space* $M. \neg P x\} \subseteq N$ *emeasure* $M N = 0$ $N \in$ *sets* M

by (*auto elim: AE-E*)

{ **assume** $\neg P x$

with x - M [*THEN sets.sets-into-space*] N **have** *emeasure* $M \{x\} \leq$ *emeasure* $M N$

by (*intro emeasure-mono*) *auto*

with $x N$ **have** *False*

by (*auto simp:*) }

then show $P x$ **by** *auto*

qed

lemma *AE-measure-singleton*: *measure* $M \{x\} \neq 0 \implies$ *AE* x *in* M . $P x \implies P x$

by (*metis AE-emeasure-singleton measure-def emeasure-empty measure-empty*)

lemma (*in finite-measure*) *AE-support-countable*:

assumes [*simp*]: *sets* $M = UNIV$

shows (*AE* x *in* M . *measure* $M \{x\} \neq 0$) \longleftrightarrow ($\exists S$. *countable* $S \wedge$ (*AE* x *in* M . $x \in S$))

proof

assume $\exists S$. *countable* $S \wedge$ (*AE* x *in* M . $x \in S$)

then obtain S **where** S [*intro*]: *countable* S **and** ae : *AE* x *in* M . $x \in S$

by *auto*

then have *emeasure* $M (\bigcup x \in \{x \in S. \text{emeasure } M \{x\} \neq 0\}. \{x\}) =$

$(\int^+ x. \text{emeasure } M \{x\} * \text{indicator } \{x \in S. \text{emeasure } M \{x\} \neq 0\} x \partial \text{count-space } UNIV)$

by (*subst emeasure-UN-countable*)

(*auto simp: disjoint-family-on-def nn-integral-restrict-space[symmetric] restrict-count-space*)

also have $\dots = (\int^+ x. \text{emeasure } M \{x\} * \text{indicator } S x \partial \text{count-space } UNIV)$

by (*auto intro!: nn-integral-cong split: split-indicator*)

also have $\dots = \text{emeasure } M (\bigcup x \in S. \{x\})$

by (*subst emeasure-UN-countable*)

(*auto simp: disjoint-family-on-def nn-integral-restrict-space[symmetric] restrict-count-space*)

also have $\dots = \text{emeasure } M (\text{space } M)$

using ae **by** (*intro emeasure-eq-AE*) *auto*

finally have *emeasure* $M \{x \in \text{space } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0\} =$ *emeasure* $M (\text{space } M)$

by (*simp add: emeasure-single-in-space cong: rev-conj-cong*)

with *finite-measure-compl*[*of* $\{x \in \text{space } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0\}$]

have *AE* x *in* M . $x \in S \wedge \text{emeasure } M \{x\} \neq 0$

```

  by (intro AE-I[OF order-refl]) (auto simp: emeasure-eq-measure measure-nonneg
  set-diff-eq cong: conj-cong)
  then show AE x in M. measure M {x} ≠ 0
    by (auto simp: emeasure-eq-measure)
qed (auto intro!: exI[of - {x. measure M {x} ≠ 0}] countable-support)

```

22.1 PMF as measure

```

typedef 'a pmf = {M :: 'a measure. prob-space M ∧ sets M = UNIV ∧ (AE x
in M. measure M {x} ≠ 0)}

```

```

  morphisms measure-pmf Abs-pmf

```

```

  by (intro exI[of - uniform-measure (count-space UNIV) {undefined}])
    (auto intro!: prob-space-uniform-measure AE-uniform-measureI)

```

```

declare [[coercion measure-pmf]]

```

```

lemma prob-space-measure-pmf: prob-space (measure-pmf p)
  using pmf.measure-pmf[of p] by auto

```

```

interpretation measure-pmf: prob-space measure-pmf M for M
  by (rule prob-space-measure-pmf)

```

```

interpretation measure-pmf: subprob-space measure-pmf M for M
  by (rule prob-space-imp-subprob-space) unfold-locales

```

```

lemma subprob-space-measure-pmf: subprob-space (measure-pmf x)
  by unfold-locales

```

```

locale pmf-as-measure
begin

```

```

setup-lifting type-definition-pmf

```

```

end

```

```

context
begin

```

```

interpretation pmf-as-measure .

```

```

lemma sets-measure-pmf[simp]: sets (measure-pmf p) = UNIV
  by transfer blast

```

```

lemma sets-measure-pmf-count-space[measurable-cong]:
  sets (measure-pmf M) = sets (count-space UNIV)
  by simp

```

```

lemma space-measure-pmf[simp]: space (measure-pmf p) = UNIV
  using sets-eq-imp-space-eq[of measure-pmf p count-space UNIV] by simp

```

lemma *measure-pmf-UNIV* [*simp*]: *measure* (*measure-pmf* *p*) *UNIV* = 1
using *measure-pmf.prob-space*[*of p*] **by** *simp*

lemma *measure-pmf-in-subprob-algebra*[*measurable (raw)*]: *measure-pmf* *x* ∈ *space* (*subprob-algebra* (*count-space UNIV*))
by (*simp add: space-subprob-algebra subprob-space-measure-pmf*)

lemma *measurable-pmf-measure1*[*simp*]: *measurable* (*M* :: 'a *pmf*) *N* = *UNIV* → *space N*
by (*auto simp: measurable-def*)

lemma *measurable-pmf-measure2*[*simp*]: *measurable N* (*M* :: 'a *pmf*) = *measurable N* (*count-space UNIV*)
by (*intro measurable-cong-sets simp-all*)

lemma *measurable-pair-restrict-pmf2*:

assumes *countable A*

assumes [*measurable*]: $\bigwedge y. y \in A \implies (\lambda x. f(x, y)) \in \text{measurable } M L$

shows $f \in \text{measurable } (M \otimes_M \text{restrict-space } (\text{measure-pmf } N) A) L$ (**is** $f \in \text{measurable } ?M$ -)

proof –

have [*measurable-cong*]: *sets* (*restrict-space* (*count-space UNIV*) *A*) = *sets* (*count-space A*)

by (*simp add: restrict-count-space*)

show *?thesis*

by (*intro measurable-compose-countable'*[**where** $f = \lambda a b. f(\text{fst } b, a)$ **and** $g = \text{snd}$ **and** $I = A$,

unfolded prod.collapse] *assms*)

measurable

qed

lemma *measurable-pair-restrict-pmf1*:

assumes *countable A*

assumes [*measurable*]: $\bigwedge x. x \in A \implies (\lambda y. f(x, y)) \in \text{measurable } N L$

shows $f \in \text{measurable } (\text{restrict-space } (\text{measure-pmf } M) A \otimes_M N) L$

proof –

have [*measurable-cong*]: *sets* (*restrict-space* (*count-space UNIV*) *A*) = *sets* (*count-space A*)

by (*simp add: restrict-count-space*)

show *?thesis*

by (*intro measurable-compose-countable'*[**where** $f = \lambda a b. f(a, \text{snd } b)$ **and** $g = \text{fst}$ **and** $I = A$,

unfolded prod.collapse] *assms*)

measurable

qed

lift-definition $pmf :: 'a pmf \Rightarrow 'a \Rightarrow real$ **is** $\lambda M x. measure M \{x\}$.

lift-definition $set-pmf :: 'a pmf \Rightarrow 'a set$ **is** $\lambda M. \{x. measure M \{x\} \neq 0\}$.
declare $[[coercion set-pmf]]$

lemma $AE-measure-pmf$: $AE x$ in $(M::'a pmf)$. $x \in M$
by $transfer simp$

lemma $emeasure-pmf-single-eq-zero-iff$:
fixes $M :: 'a pmf$
shows $emeasure M \{y\} = 0 \longleftrightarrow y \notin M$
unfolding $set-pmf.rep-eq$ **by** $(simp add: measure-pmf.emeasure-eq-measure)$

lemma $AE-measure-pmf-iff$: $(AE x$ in $measure-pmf M. P x) \longleftrightarrow (\forall y \in M. P y)$
using $AE-measure-singleton[of M]$ $AE-measure-pmf[of M]$
by $(auto simp: set-pmf.rep-eq)$

lemma $AE-pmfI$: $(\bigwedge y. y \in set-pmf M \Longrightarrow P y) \Longrightarrow almost-everywhere (measure-pmf M) P$
by $(simp add: AE-measure-pmf-iff)$

lemma $countable-set-pmf [simp]$: $countable (set-pmf p)$
by $transfer (metis prob-space.finite-measure finite-measure.countable-support)$

lemma $pmf-positive$: $x \in set-pmf p \Longrightarrow 0 < pmf p x$
by $transfer (simp add: less-le)$

lemma $pmf-nonneg[simp]$: $0 \leq pmf p x$
by $transfer simp$

lemma $pmf-not-neg [simp]$: $\neg pmf p x < 0$
by $(simp add: not-less pmf-nonneg)$

lemma $pmf-pos [simp]$: $pmf p x \neq 0 \Longrightarrow pmf p x > 0$
using $pmf-nonneg[of p x]$ **by** $linarith$

lemma $pmf-le-1$: $pmf p x \leq 1$
by $(simp add: pmf.rep-eq)$

lemma $set-pmf-not-empty$: $set-pmf M \neq \{\}$
using $AE-measure-pmf[of M]$ **by** $(intro notI) simp$

lemma $set-pmf-iff$: $x \in set-pmf M \longleftrightarrow pmf M x \neq 0$
by $transfer simp$

lemma $pmf-positive-iff$: $0 < pmf p x \longleftrightarrow x \in set-pmf p$
unfolding $less-le$ **by** $(simp add: set-pmf-iff)$

lemma $set-pmf-eq$: $set-pmf M = \{x. pmf M x \neq 0\}$

by (auto simp: set-pmf-iff)

lemma *set-pmf-eq'*: $\text{set-pmf } p = \{x. \text{pmf } p \ x > 0\}$

proof safe

fix x assume $x \in \text{set-pmf } p$

hence $\text{pmf } p \ x \neq 0$ by (auto simp: set-pmf-eq)

with $\text{pmf-nonneg}[of \ p \ x]$ show $\text{pmf } p \ x > 0$ by simp

qed (auto simp: set-pmf-eq)

lemma *emeasure-pmf-single*:

fixes $M :: 'a \ \text{pmf}$

shows $\text{emeasure } M \ \{x\} = \text{pmf } M \ x$

by transfer (simp add: finite-measure.emeasure-eq-measure[OF prob-space.finite-measure])

lemma *measure-pmf-single*: $\text{measure } (\text{measure-pmf } M) \ \{x\} = \text{pmf } M \ x$

using *emeasure-pmf-single*[of $M \ x$] by (simp add: measure-pmf.emeasure-eq-measure pmf-nonneg measure-nonneg)

lemma *emeasure-measure-pmf-finite*: $\text{finite } S \implies \text{emeasure } (\text{measure-pmf } M) \ S$

$= (\sum s \in S. \text{pmf } M \ s)$

by (subst *emeasure-eq-sum-singleton*) (auto simp: *emeasure-pmf-single pmf-nonneg*)

lemma *measure-measure-pmf-finite*: $\text{finite } S \implies \text{measure } (\text{measure-pmf } M) \ S = \text{sum } (\text{pmf } M) \ S$

using *emeasure-measure-pmf-finite*[of $S \ M$]

by (simp add: measure-pmf.emeasure-eq-measure measure-nonneg sum-nonneg pmf-nonneg)

lemma *sum-pmf-eq-1*:

assumes $\text{finite } A \ \text{set-pmf } p \subseteq A$

shows $(\sum x \in A. \text{pmf } p \ x) = 1$

proof –

have $(\sum x \in A. \text{pmf } p \ x) = \text{measure-pmf.prob } p \ A$

by (simp add: *measure-measure-pmf-finite assms*)

also from *assms* have $\dots = 1$

by (subst *measure-pmf.prob-eq-1*) (auto simp: *AE-measure-pmf-iff*)

finally show ?thesis .

qed

lemma *nn-integral-measure-pmf-support*:

fixes $f :: 'a \Rightarrow \text{ennreal}$

assumes f : $\text{finite } A$ and nn : $\bigwedge x. x \in A \implies 0 \leq f \ x \ \bigwedge x. x \in \text{set-pmf } M \implies x \notin A \implies f \ x = 0$

shows $(\int^+ x. f \ x \ \partial \text{measure-pmf } M) = (\sum x \in A. f \ x * \text{pmf } M \ x)$

proof –

have $(\int^+ x. f \ x \ \partial M) = (\int^+ x. f \ x * \text{indicator } A \ x \ \partial M)$

using nn by (*intro nn-integral-cong-AE*) (auto simp: *AE-measure-pmf-iff split: split-indicator*)

also have $\dots = (\sum x \in A. f \ x * \text{emeasure } M \ \{x\})$

using *assms* **by** (*intro nn-integral-indicator-finite*) *auto*
finally show *?thesis*
by (*simp add: emeasure-measure-pmf-finite*)
qed

lemma *nn-integral-measure-pmf-finite*:

fixes $f :: 'a \Rightarrow \text{ennreal}$

assumes f : *finite* (*set-pmf* M) **and** nn : $\bigwedge x. x \in \text{set-pmf } M \implies 0 \leq f x$

shows $(\int^+ x. f x \ \partial \text{measure-pmf } M) = (\sum_{x \in \text{set-pmf } M}. f x * \text{pmf } M x)$

using *assms* **by** (*intro nn-integral-measure-pmf-support*) *auto*

lemma *integrable-measure-pmf-finite*:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

shows *finite* (*set-pmf* M) \implies *integrable* $M f$

by (*auto intro!*: *integrableI-bounded simp: nn-integral-measure-pmf-finite ennreal-mult-less-top*)

lemma *integral-measure-pmf-real*:

assumes [*simp*]: *finite* A **and** $\bigwedge a. a \in \text{set-pmf } M \implies f a \neq 0 \implies a \in A$

shows $(\int x. f x \ \partial \text{measure-pmf } M) = (\sum_{a \in A}. f a * \text{pmf } M a)$

proof –

have $(\int x. f x \ \partial \text{measure-pmf } M) = (\int x. f x * \text{indicator } A x \ \partial \text{measure-pmf } M)$

using *assms*(2) **by** (*intro integral-cong-AE*) (*auto split: split-indicator simp: AE-measure-pmf-iff*)

also have $\dots = (\sum_{a \in A}. f a * \text{pmf } M a)$

by (*subst integral-indicator-finite-real*)

(*auto simp: measure-def emeasure-measure-pmf-finite pmf-nonneg*)

finally show *?thesis* .

qed

lemma *integrable-pmf: integrable* (*count-space* X) (*pmf* M)

proof –

have $(\int^+ x. \text{pmf } M x \ \partial \text{count-space } X) = (\int^+ x. \text{pmf } M x \ \partial \text{count-space } (M \cap X))$

by (*auto simp add: nn-integral-count-space-indicator set-pmf-iff intro!*: *nn-integral-cong split: split-indicator*)

then have *integrable* (*count-space* X) (*pmf* M) = *integrable* (*count-space* ($M \cap X$)) (*pmf* M)

by (*simp add: integrable-iff-bounded pmf-nonneg*)

then show *?thesis*

by (*simp add: pmf.rep-eq measure-pmf.integrable-measure disjoint-family-on-def*)

qed

lemma *integral-pmf*: $(\int x. \text{pmf } M x \ \partial \text{count-space } X) = \text{measure } M X$

proof –

have $(\int x. \text{pmf } M x \ \partial \text{count-space } X) = (\int^+ x. \text{pmf } M x \ \partial \text{count-space } X)$

by (*simp add: pmf-nonneg integrable-pmf nn-integral-eq-integral*)

also have $\dots = (\int^+ x. \text{emeasure } M \{x\} \ \partial \text{count-space } (X \cap M))$

by (*auto intro!*: *nn-integral-cong-AE split: split-indicator*)

simp: pmf.rep-eq measure-pmf.emeasure-eq-measure nn-integral-count-space-indicator

AE-count-space set-pmf-iff)
also have ... = *emeasure* M ($X \cap M$)
by (*rule emeasure-countable-singleton[symmetric]*) (*auto intro: countable-set-pmf*)
also have ... = *emeasure* M X
by (*auto intro!: emeasure-eq-AE simp: AE-measure-pmf-iff*)
finally show ?*thesis*
by (*simp add: measure-pmf.emeasure-eq-measure measure-nonneg integral-nonneg pmf-nonneg*)
qed

lemma *integral-pmf-restrict*:
 $(f::'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}) \in \text{borel-measurable } (\text{count-space } UNIV) \implies$
 $(\int x. f x \partial \text{measure-pmf } M) = (\int x. f x \partial \text{restrict-space } M M)$
by (*auto intro!: integral-cong-AE simp add: integral-restrict-space AE-measure-pmf-iff*)

lemma *emeasure-pmf*: *emeasure* ($M::'a$ *pmf*) $M = 1$
proof –
have *emeasure* ($M::'a$ *pmf*) $M = \text{emeasure } (M::'a$ *pmf*) (*space* M)
by (*intro emeasure-eq-AE*) (*simp-all add: AE-measure-pmf*)
then show ?*thesis*
using *measure-pmf.emeasure-space-1* **by** *simp*
qed

lemma *emeasure-pmf-UNIV* [*simp*]: *emeasure* (*measure-pmf* M) $UNIV = 1$
using *measure-pmf.emeasure-space-1[of M]* **by** *simp*

lemma *in-null-sets-measure-pmfI*:
 $A \cap \text{set-pmf } p = \{\} \implies A \in \text{null-sets } (\text{measure-pmf } p)$
using *emeasure-eq-0-AE*[**where** ? $P = \lambda x. x \in A$ **and** $M = \text{measure-pmf } p$]
by(*auto simp add: null-sets-def AE-measure-pmf-iff*)

lemma *measure-subprob*: *measure-pmf* $M \in \text{space } (\text{subprob-algebra } (\text{count-space } UNIV))$
by (*simp add: space-subprob-algebra subprob-space-measure-pmf*)

22.2 Monad Interpretation

lemma *measurable-measure-pmf*[*measurable*]:
 $(\lambda x. \text{measure-pmf } (M x)) \in \text{measurable } (\text{count-space } UNIV) (\text{subprob-algebra } (\text{count-space } UNIV))$
by (*auto simp: space-subprob-algebra intro!: prob-space-imp-subprob-space*) *unfold-locales*

lemma *bind-measure-pmf-cong*:
assumes $\bigwedge x. A x \in \text{space } (\text{subprob-algebra } N) \wedge x. B x \in \text{space } (\text{subprob-algebra } N)$
assumes $\bigwedge i. i \in \text{set-pmf } x \implies A i = B i$
shows *bind* (*measure-pmf* x) $A = \text{bind } (\text{measure-pmf } x) B$
proof (*rule measure-eqI*)

```

show sets (measure-pmf x  $\gg$  A) = sets (measure-pmf x  $\gg$  B)
  using assms by (subst (1 2) sets-bind) (auto simp: space-subprob-algebra)
next
fix X assume X  $\in$  sets (measure-pmf x  $\gg$  A)
then have X: X  $\in$  sets N
  using assms by (subst (asm) sets-bind) (auto simp: space-subprob-algebra)
show emeasure (measure-pmf x  $\gg$  A) X = emeasure (measure-pmf x  $\gg$  B) X
  using assms
  by (subst (1 2) emeasure-bind[where N=N, OF - - X])
    (auto intro!: nn-integral-cong-AE simp: AE-measure-pmf-iff)
qed

```

lift-definition bind-pmf :: 'a pmf \Rightarrow ('a \Rightarrow 'b pmf) \Rightarrow 'b pmf **is** bind

proof (*clarify, intro conjI*)

fix f :: 'a measure **and** g :: 'a \Rightarrow 'b measure

assume prob-space f

then interpret f: prob-space f .

assume sets f = UNIV **and** ae-f: AE x in f. measure f {x} \neq 0

then have s-f[*simp*]: sets f = sets (count-space UNIV)

by *simp*

assume g: $\bigwedge x$. prob-space (g x) \wedge sets (g x) = UNIV \wedge (AE y in g x. measure (g x) {y} \neq 0)

then have g: $\bigwedge x$. prob-space (g x) **and** s-g[*simp*]: $\bigwedge x$. sets (g x) = sets (count-space UNIV)

and ae-g: $\bigwedge x$. AE y in g x. measure (g x) {y} \neq 0

by *auto*

have [*measurable*]: g \in measurable f (subprob-algebra (count-space UNIV))

by (auto simp: measurable-def space-subprob-algebra prob-space-imp-subprob-space g)

show prob-space (f \gg g)

using g **by** (intro f.prob-space-bind[**where** S=count-space UNIV]) *auto*

then interpret fg: prob-space f \gg g .

show [*simp*]: sets (f \gg g) = UNIV

using sets-eq-imp-space-eq[OF s-f]

by (subst sets-bind[**where** N=count-space UNIV]) *auto*

show AE x in f \gg g. measure (f \gg g) {x} \neq 0

apply (simp add: fg.prob-eq-0 AE-bind[**where** B=count-space UNIV])

using ae-f

apply eventually-elim

using ae-g

apply eventually-elim

apply (auto dest: AE-measure-singleton)

done

qed

adhoc-overloading Monad-Syntax.bind bind-pmf

lemma *ennreal-pmf-bind*: $\text{pmf } (\text{bind-pmf } N f) i = (\int^+ x. \text{pmf } (f x) i \partial \text{measure-pmf } N)$
unfolding *pmf.rep-eq bind-pmf.rep-eq*
by (*auto simp: measure-pmf.measure-bind*[**where** $N = \text{count-space UNIV}$] *measure-subprob measure-nonneg*
intro!: *nn-integral-eq-integral[symmetric] measure-pmf.integrable-const-bound*[**where** $B=I$])

lemma *pmf-bind*: $\text{pmf } (\text{bind-pmf } N f) i = (\int x. \text{pmf } (f x) i \partial \text{measure-pmf } N)$
using *ennreal-pmf-bind*[*of* $N f i$]
by (*subst (asm) nn-integral-eq-integral*)
(auto simp: pmf-nonneg pmf-le-1 pmf-nonneg integral-nonneg
intro!: *nn-integral-eq-integral[symmetric] measure-pmf.integrable-const-bound*[**where** $B=I$])

lemma *bind-pmf-const*[*simp*]: $\text{bind-pmf } M (\lambda x. c) = c$
by *transfer (simp add: bind-const' prob-space-imp-subprob-space)*

lemma *set-bind-pmf*[*simp*]: $\text{set-pmf } (\text{bind-pmf } M N) = (\bigcup M \in \text{set-pmf } M. \text{set-pmf } (N M))$

proof –

have $\text{set-pmf } (\text{bind-pmf } M N) = \{x. \text{ennreal } (\text{pmf } (\text{bind-pmf } M N) x) \neq 0\}$
by (*simp add: set-pmf-eq pmf-nonneg*)
also have $\dots = (\bigcup M \in \text{set-pmf } M. \text{set-pmf } (N M))$
unfolding *ennreal-pmf-bind*
by (*subst nn-integral-0-iff-AE*) (*auto simp: AE-measure-pmf-iff pmf-nonneg set-pmf-eq*)
finally show *?thesis* .
qed

lemma *bind-pmf-cong* [*fundef-cong*]:

assumes $p = q$
shows $(\bigwedge x. x \in \text{set-pmf } q \implies f x = g x) \implies \text{bind-pmf } p f = \text{bind-pmf } q g$
unfolding $\langle p = q \rangle$ [*symmetric*] *measure-pmf-inject[symmetric] bind-pmf.rep-eq*
by (*auto simp: AE-measure-pmf-iff Pi-iff space-subprob-algebra subprob-space-measure-pmf sets-bind*[**where** $N = \text{count-space UNIV}$] *emeasure-bind*[**where** $N = \text{count-space UNIV}$]
intro!: *nn-integral-cong-AE measure-eqI*)

lemma *bind-pmf-cong-simp*:

$p = q \implies (\bigwedge x. x \in \text{set-pmf } q = \text{simp} \implies f x = g x) \implies \text{bind-pmf } p f = \text{bind-pmf } q g$
by (*simp add: simp-implies-def cong: bind-pmf-cong*)

lemma *measure-pmf-bind*: $\text{measure-pmf } (\text{bind-pmf } M f) = (\text{measure-pmf } M \gg (\lambda x. \text{measure-pmf } (f x)))$

by *transfer simp*

lemma *nn-integral-bind-pmf*[*simp*]: $(\int^+ x. f x \partial \text{bind-pmf } M N) = (\int^+ x. \int^+ y. f$

```

y ∂N x ∂M)
  using measurable-measure-pmf[of N]
  unfolding measure-pmf-bind
  apply (intro nn-integral-bind[where B=count-space UNIV])
  apply auto
  done

```

lemma *emeasure-bind-pmf*[simp]: $\text{emeasure } (\text{bind-pmf } M \ N) \ X = (\int^+ x. \text{emeasure } (N \ x) \ X \ \partial M)$

```

  using measurable-measure-pmf[of N]
  unfolding measure-pmf-bind
  by (subst emeasure-bind[where N=count-space UNIV]) auto

```

lift-definition *return-pmf* :: 'a \Rightarrow 'a pmf **is** *return* (count-space UNIV)

```

  by (auto intro!: prob-space-return simp: AE-return measure-return)

```

lemma *bind-return-pmf*: $\text{bind-pmf } (\text{return-pmf } x) \ f = f \ x$

```

  by transfer
  (auto intro!: prob-space-imp-subprob-space bind-return[where N=count-space UNIV]
  simp: space-subprob-algebra)

```

lemma *set-return-pmf*[simp]: $\text{set-pmf } (\text{return-pmf } x) = \{x\}$

```

  by transfer (auto simp add: measure-return split: split-indicator)

```

lemma *bind-return-pmf'*: $\text{bind-pmf } N \ \text{return-pmf} = N$

proof (*transfer, clarify*)

fix $N :: 'a \text{ measure}$ **assume** sets $N = \text{UNIV}$ **then show** $N \ggg \text{return } (\text{count-space UNIV}) = N$

```

  by (subst return-sets-cong[where N=N]) (simp-all add: bind-return')

```

qed

lemma *bind-assoc-pmf*: $\text{bind-pmf } (\text{bind-pmf } A \ B) \ C = \text{bind-pmf } A \ (\lambda x. \text{bind-pmf } (B \ x) \ C)$

```

  by transfer

```

```

  (auto intro!: bind-assoc[where N=count-space UNIV and R=count-space UNIV]
  simp: measurable-def space-subprob-algebra prob-space-imp-subprob-space)

```

definition *map-pmf* $f \ M = \text{bind-pmf } M \ (\lambda x. \text{return-pmf } (f \ x))$

lemma *map-bind-pmf*: $\text{map-pmf } f \ (\text{bind-pmf } M \ g) = \text{bind-pmf } M \ (\lambda x. \text{map-pmf } f \ (g \ x))$

```

  by (simp add: map-pmf-def bind-assoc-pmf)

```

lemma *bind-map-pmf*: $\text{bind-pmf } (\text{map-pmf } f \ M) \ g = \text{bind-pmf } M \ (\lambda x. g \ (f \ x))$

```

  by (simp add: map-pmf-def bind-assoc-pmf bind-return-pmf)

```

lemma *map-pmf-transfer*[*transfer-rule*]:

$rel\text{-}fun\ op = (rel\text{-}fun\ cr\text{-}pmf\ cr\text{-}pmf)\ (\lambda f\ M.\ distr\ M\ (count\text{-}space\ UNIV)\ f)$
 $map\text{-}pmf$

proof –

have $rel\text{-}fun\ op = (rel\text{-}fun\ pmf\text{-}as\text{-}measure.\ cr\text{-}pmf\ pmf\text{-}as\text{-}measure.\ cr\text{-}pmf)$

$(\lambda f\ M.\ M \gg\! = (return\ (count\text{-}space\ UNIV)\ o\ f))\ map\text{-}pmf$

unfolding $map\text{-}pmf\text{-}def[abs\text{-}def]\ comp\text{-}def$ **by** $transfer\text{-}prover$

then show $?thesis$

by $(force\ simp:\ rel\text{-}fun\text{-}def\ cr\text{-}pmf\text{-}def\ bind\text{-}return\text{-}distr)$

qed

lemma $map\text{-}pmf\text{-}rep\text{-}eq$:

$measure\text{-}pmf\ (map\text{-}pmf\ f\ M) = distr\ (measure\text{-}pmf\ M)\ (count\text{-}space\ UNIV)\ f$

unfolding $map\text{-}pmf\text{-}def\ bind\text{-}pmf.\ rep\text{-}eq\ comp\text{-}def\ return\text{-}pmf.\ rep\text{-}eq$

using $bind\text{-}return\text{-}distr[of\ M\ f\ count\text{-}space\ UNIV]$ **by** $(simp\ add:\ comp\text{-}def)$

lemma $map\text{-}pmf\text{-}id[simp]$: $map\text{-}pmf\ id = id$

by $(rule,\ transfer)\ (auto\ simp:\ emeasure\text{-}distr\ measurable\text{-}def\ intro!\ : measure\text{-}eqI)$

lemma $map\text{-}pmf\text{-}ident[simp]$: $map\text{-}pmf\ (\lambda x.\ x) = (\lambda x.\ x)$

using $map\text{-}pmf\text{-}id$ **unfolding** $id\text{-}def$.

lemma $map\text{-}pmf\text{-}compose$: $map\text{-}pmf\ (f\ o\ g) = map\text{-}pmf\ f\ o\ map\text{-}pmf\ g$

by $(rule,\ transfer)\ (simp\ add:\ distr\text{-}distr[symmetric,\ where\ N=count\text{-}space\ UNIV]\ measurable\text{-}def)$

lemma $map\text{-}pmf\text{-}comp$: $map\text{-}pmf\ f\ (map\text{-}pmf\ g\ M) = map\text{-}pmf\ (\lambda x.\ f\ (g\ x))\ M$

using $map\text{-}pmf\text{-}compose[of\ f\ g]$ **by** $(simp\ add:\ comp\text{-}def)$

lemma $map\text{-}pmf\text{-}cong$: $p = q \implies (\bigwedge x.\ x \in set\text{-}pmf\ q \implies f\ x = g\ x) \implies map\text{-}pmf\ f\ p = map\text{-}pmf\ f\ q$

unfolding $map\text{-}pmf\text{-}def$ **by** $(rule\ bind\text{-}pmf\text{-}cong)\ auto$

lemma $pmf\text{-}set\text{-}map$: $set\text{-}pmf\ o\ map\text{-}pmf\ f = op\ 'f\ o\ set\text{-}pmf$

by $(auto\ simp\ add:\ comp\text{-}def\ fun\text{-}eq\text{-}iff\ map\text{-}pmf\text{-}def)$

lemma $set\text{-}map\text{-}pmf[simp]$: $set\text{-}pmf\ (map\text{-}pmf\ f\ M) = f\ set\text{-}pmf\ M$

using $pmf\text{-}set\text{-}map[of\ f]$ **by** $(auto\ simp:\ comp\text{-}def\ fun\text{-}eq\text{-}iff)$

lemma $emeasure\text{-}map\text{-}pmf[simp]$: $emeasure\ (map\text{-}pmf\ f\ M)\ X = emeasure\ M\ (f\ -\ 'X)$

unfolding $map\text{-}pmf\text{-}rep\text{-}eq$ **by** $(subst\ emeasure\text{-}distr)\ auto$

lemma $measure\text{-}map\text{-}pmf[simp]$: $measure\ (map\text{-}pmf\ f\ M)\ X = measure\ M\ (f\ -\ 'X)$

using $emeasure\text{-}map\text{-}pmf[of\ f\ M\ X]$ **by** $(simp\ add:\ measure\text{-}pmf.\ emeasure\text{-}eq\text{-}measure\ measure\text{-}nonneg)$

lemma $nn\text{-}integral\text{-}map\text{-}pmf[simp]$: $(\int^{+x}.\ f\ x\ \partial map\text{-}pmf\ g\ M) = (\int^{+x}.\ f\ (g\ x)\ \partial M)$

unfolding *map-pmf-rep-eq* **by** (*intro nn-integral-distr*) *auto*

lemma *ennreal-pmf-map*: $\text{pmf } (\text{map-pmf } f \ p) \ x = (\int^+ y. \text{indicator } (f - \{x\}) \ y \ \partial \text{measure-pmf } p)$

proof (*transfer fixing: f x*)

fix $p :: 'b \ \text{measure}$

presume *prob-space p*

then interpret *prob-space p* .

presume $\text{sets } p = \text{UNIV}$

then show $\text{ennreal } (\text{measure } (\text{distr } p \ (\text{count-space } \text{UNIV}) \ f) \ \{x\}) = \text{integral}^N \ p \ (\text{indicator } (f - \{x\}))$

by (*simp add: measure-distr measurable-def emeasure-eq-measure*)

qed *simp-all*

lemma *pmf-map*: $\text{pmf } (\text{map-pmf } f \ p) \ x = \text{measure } p \ (f - \{x\})$

proof (*transfer fixing: f x*)

fix $p :: 'b \ \text{measure}$

presume *prob-space p*

then interpret *prob-space p* .

presume $\text{sets } p = \text{UNIV}$

then show $\text{measure } (\text{distr } p \ (\text{count-space } \text{UNIV}) \ f) \ \{x\} = \text{measure } p \ (f - \{x\})$

by (*simp add: measure-distr measurable-def emeasure-eq-measure*)

qed *simp-all*

lemma *nn-integral-pmf*: $(\int^+ x. \text{pmf } p \ x \ \partial \text{count-space } A) = \text{emeasure } (\text{measure-pmf } p) \ A$

proof –

have $(\int^+ x. \text{pmf } p \ x \ \partial \text{count-space } A) = (\int^+ x. \text{pmf } p \ x \ \partial \text{count-space } (A \cap \text{set-pmf } p))$

by (*auto simp add: nn-integral-count-space-indicator indicator-def set-pmf-iff intro: nn-integral-cong*)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) \ (\bigcup x \in A \cap \text{set-pmf } p. \ \{x\})$

by (*subst emeasure-UN-countable*) (*auto simp add: emeasure-pmf-single disjoint-family-on-def*)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) \ ((\bigcup x \in A \cap \text{set-pmf } p. \ \{x\}) \cup \{x. \ x \in A \wedge x \notin \text{set-pmf } p\})$

by (*rule emeasure-Un-null-set[symmetric]*) (*auto intro: in-null-sets-measure-pmfI*)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) \ A$

by (*auto intro: arg-cong2[where f=emeasure]*)

finally show *?thesis* .

qed

lemma *integral-map-pmf[simp]*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$

shows $\text{integral}^L \ (\text{map-pmf } g \ p) \ f = \text{integral}^L \ p \ (\lambda x. \ f \ (g \ x))$

by (*simp add: integral-distr map-pmf-rep-eq*)

lemma *pmf-abs-summable [intro]*: *pmf p abs-summable-on A*

by (*rule abs-summable-on-subset[OF - subset-UNIV]*)

(*auto simp: abs-summable-on-def integrable-iff-bounded nn-integral-pmf*)

lemma *measure-pmf-conv-infsetsum*: $\text{measure} (\text{measure-pmf } p) A = \text{infsetsum} (\text{pmf } p) A$

unfolding *infsetsum-def* **by** (*simp add: integral-eq-nn-integral nn-integral-pmf measure-def*)

lemma *infsetsum-pmf-eq-1*:

assumes $\text{set-pmf } p \subseteq A$

shows $\text{infsetsum} (\text{pmf } p) A = 1$

proof –

have $\text{infsetsum} (\text{pmf } p) A = \text{lebesgue-integral} (\text{count-space UNIV}) (\text{pmf } p)$

using *assms unfolding infsetsum-altdef*

by (*intro Bochner-Integration.integral-cong*) (*auto simp: indicator-def set-pmf-eq*)

also have $\dots = 1$

by (*subst integral-eq-nn-integral*) (*auto simp: nn-integral-pmf*)

finally show *?thesis* .

qed

lemma *map-return-pmf [simp]*: $\text{map-pmf } f (\text{return-pmf } x) = \text{return-pmf } (f x)$

by *transfer (simp add: distr-return)*

lemma *map-pmf-const [simp]*: $\text{map-pmf } (\lambda \cdot. c) M = \text{return-pmf } c$

by *transfer (auto simp: prob-space.distr-const)*

lemma *pmf-return [simp]*: $\text{pmf} (\text{return-pmf } x) y = \text{indicator } \{y\} x$

by *transfer (simp add: measure-return)*

lemma *nn-integral-return-pmf [simp]*: $0 \leq f x \implies (\int^{+x}. f x \partial \text{return-pmf } x) = f x$

unfolding *return-pmf.rep-eq* **by** (*intro nn-integral-return*) *auto*

lemma *emeasure-return-pmf [simp]*: $\text{emeasure} (\text{return-pmf } x) X = \text{indicator } X x$

unfolding *return-pmf.rep-eq* **by** (*intro emeasure-return*) *auto*

lemma *measure-return-pmf [simp]*: $\text{measure-pmf.prob} (\text{return-pmf } x) A = \text{indicator } A x$

proof –

have $\text{ennreal} (\text{measure-pmf.prob} (\text{return-pmf } x) A) =$

$\text{emeasure} (\text{measure-pmf} (\text{return-pmf } x)) A$

by (*simp add: measure-pmf.emeasure-eq-measure*)

also have $\dots = \text{ennreal} (\text{indicator } A x)$ **by** (*simp add: ennreal-indicator*)

finally show *?thesis* **by** *simp*

qed

lemma *return-pmf-inj [simp]*: $\text{return-pmf } x = \text{return-pmf } y \iff x = y$

by (*metis insertI1 set-return-pmf singletonD*)

lemma *map-pmf-eq-return-pmf-iff*:

$\text{map-pmf } f p = \text{return-pmf } x \iff (\forall y \in \text{set-pmf } p. f y = x)$

proof

assume $\text{map-pmf } f \ p = \text{return-pmf } x$
 then have $\text{set-pmf } (\text{map-pmf } f \ p) = \text{set-pmf } (\text{return-pmf } x)$ **by** *simp*
 then show $\forall y \in \text{set-pmf } p. f \ y = x$ **by** *auto*
next
 assume $\forall y \in \text{set-pmf } p. f \ y = x$
 then show $\text{map-pmf } f \ p = \text{return-pmf } x$
 unfolding $\text{map-pmf-const}[\text{symmetric, of } - \ p]$ **by** (*intro map-pmf-cong*) *auto*
qed

definition $\text{pair-pmf } A \ B = \text{bind-pmf } A \ (\lambda x. \text{bind-pmf } B \ (\lambda y. \text{return-pmf } (x, y)))$

lemma *pmf-pair*: $\text{pmf } (\text{pair-pmf } M \ N) \ (a, b) = \text{pmf } M \ a * \text{pmf } N \ b$
 unfolding *pair-pmf-def pmf-bind pmf-return*
 apply (*subst integral-measure-pmf-real* [**where** $A = \{b\}$])
 apply (*auto simp: indicator-eq-0-iff*)
 apply (*subst integral-measure-pmf-real* [**where** $A = \{a\}$])
 apply (*auto simp: indicator-eq-0-iff sum-nonneg-eq-0-iff pmf-nonneg*)
 done

lemma *set-pair-pmf*[*simp*]: $\text{set-pmf } (\text{pair-pmf } A \ B) = \text{set-pmf } A \times \text{set-pmf } B$
 unfolding *pair-pmf-def set-bind-pmf set-return-pmf* **by** *auto*

lemma *measure-pmf-in-subprob-space*[*measurable (raw)*]:
 $\text{measure-pmf } M \in \text{space } (\text{subprob-algebra } (\text{count-space } \text{UNIV}))$
 by (*simp add: space-subprob-algebra intro-locales*)

lemma *nn-integral-pair-pmf'*: $(\int^{+x}. f \ x \ \partial \text{pair-pmf } A \ B) = (\int^{+a}. \int^{+b}. f \ (a, b) \ \partial B \ \partial A)$

proof –

have $(\int^{+x}. f \ x \ \partial \text{pair-pmf } A \ B) = (\int^{+x}. f \ x * \text{indicator } (A \times B) \ x \ \partial \text{pair-pmf } A \ B)$
 by (*auto simp: AE-measure-pmf-iff intro!: nn-integral-cong-AE*)
 also have $\dots = (\int^{+a}. \int^{+b}. f \ (a, b) * \text{indicator } (A \times B) \ (a, b) \ \partial B \ \partial A)$
 by (*simp add: pair-pmf-def*)
 also have $\dots = (\int^{+a}. \int^{+b}. f \ (a, b) \ \partial B \ \partial A)$
 by (*auto intro!: nn-integral-cong-AE simp: AE-measure-pmf-iff*)
 finally show *?thesis* .

qed

lemma *bind-pair-pmf*:

assumes $M[\text{measurable}] : M \in \text{measurable } (\text{count-space } \text{UNIV} \otimes_M \text{count-space } \text{UNIV}) \ (\text{subprob-algebra } N)$
 shows $\text{measure-pmf } (\text{pair-pmf } A \ B) \ggg M = (\text{measure-pmf } A \ggg (\lambda x. \text{measure-pmf } B \ggg (\lambda y. M \ (x, y))))$
 (is ?L = ?R)

proof (*rule measure-eqI*)

have $M'[\text{measurable}] : M \in \text{measurable } (\text{pair-pmf } A \ B) \ (\text{subprob-algebra } N)$
 using $M[\text{THEN measurable-space}]$ **by** (*simp-all add: space-pair-measure*)

```

note measurable-bind[where  $N = \text{count-space UNIV}$ , measurable]
note measure-pmf-in-subprob-space[simp]

have sets-eq-N: sets ?L = N
  by (subst sets-bind[OF sets-kernel[OF M']]) auto
show sets ?L = sets ?R
  using measurable-space[OF M]
  by (simp add: sets-eq-N space-pair-measure space-subprob-algebra)
fix X assume X ∈ sets ?L
then have X[measurable]: X ∈ sets N
  unfolding sets-eq-N .
then show emeasure ?L X = emeasure ?R X
  apply (simp add: emeasure-bind[OF - M' X])
  apply (simp add: nn-integral-bind[where  $B = \text{count-space UNIV}$ ] pair-pmf-def
measure-pmf-bind[of A]
      nn-integral-measure-pmf-finite)
  apply (subst emeasure-bind[OF - - X])
  apply measurable
  apply (subst emeasure-bind[OF - - X])
  apply measurable
  done
qed

lemma map-fst-pair-pmf: map-pmf fst (pair-pmf A B) = A
  by (simp add: pair-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf')

lemma map-snd-pair-pmf: map-pmf snd (pair-pmf A B) = B
  by (simp add: pair-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf')

lemma nn-integral-pmf':
  inj-on f A  $\implies$   $(\int^+ x. \text{pmf } p (f x) \partial \text{count-space } A) = \text{emeasure } p (f \text{' } A)$ 
  by (subst nn-integral-bij-count-space[where  $g=f$  and  $B=f \text{' } A$ ])
  (auto simp: bij-betw-def nn-integral-pmf)

lemma pmf-le-0-iff[simp]: pmf M p ≤ 0  $\iff$  pmf M p = 0
  using pmf-nonneg[of M p] by arith

lemma min-pmf-0[simp]: min (pmf M p) 0 = 0 min 0 (pmf M p) = 0
  using pmf-nonneg[of M p] by arith+

lemma pmf-eq-0-set-pmf: pmf M p = 0  $\iff$  p ∉ set-pmf M
  unfolding set-pmf-iff by simp

lemma pmf-map-inj: inj-on f (set-pmf M)  $\implies$  x ∈ set-pmf M  $\implies$  pmf (map-pmf
f M) (f x) = pmf M x
  by (auto simp: pmf.rep-eq map-pmf.rep-eq measure-distr AE-measure-pmf-iff
inj-onD
      intro!: measure-pmf.finite-measure-eq-AE)

```

```

lemma pmf-map-inj': inj f  $\implies$  pmf (map-pmf f M) (f x) = pmf M x
apply(cases x  $\in$  set-pmf M)
  apply(simp add: pmf-map-inj[OF subset-inj-on])
apply(simp add: pmf-eq-0-set-pmf[symmetric])
apply(auto simp add: pmf-eq-0-set-pmf dest: injD)
done

```

```

lemma pmf-map-outside: x  $\notin$  f ` set-pmf M  $\implies$  pmf (map-pmf f M) x = 0
  unfolding pmf-eq-0-set-pmf by simp

```

```

lemma measurable-set-pmf[measurable]: Measurable.pred (count-space UNIV) ( $\lambda x.$ 
x  $\in$  set-pmf M)
  by simp

```

22.3 PMFs as function

```

context
  fixes f :: 'a  $\Rightarrow$  real
  assumes nonneg:  $\bigwedge x. 0 \leq f x$ 
  assumes prob: ( $\int^+ x. f x \partial$ count-space UNIV) = 1
begin

```

```

lift-definition embed-pmf :: 'a pmf is density (count-space UNIV) (ennreal  $\circ$  f)

```

```

proof (intro conjI)

```

```

  have *[simp]:  $\bigwedge x y. \text{ennreal } (f y) * \text{indicator } \{x\} y = \text{ennreal } (f x) * \text{indicator } \{x\} y$ 

```

```

    by (simp split: split-indicator)

```

```

  show AE x in density (count-space UNIV) (ennreal  $\circ$  f).

```

```

    measure (density (count-space UNIV) (ennreal  $\circ$  f)) {x}  $\neq$  0

```

```

    by (simp add: AE-density nonneg measure-def emeasure-density max-def)

```

```

  show prob-space (density (count-space UNIV) (ennreal  $\circ$  f))

```

```

    by standard (simp add: emeasure-density prob)

```

```

qed simp

```

```

lemma pmf-embed-pmf: pmf embed-pmf x = f x

```

```

proof transfer

```

```

  have *[simp]:  $\bigwedge x y. \text{ennreal } (f y) * \text{indicator } \{x\} y = \text{ennreal } (f x) * \text{indicator } \{x\} y$ 

```

```

    by (simp split: split-indicator)

```

```

  fix x show measure (density (count-space UNIV) (ennreal  $\circ$  f)) {x} = f x

```

```

    by transfer (simp add: measure-def emeasure-density nonneg max-def)

```

```

qed

```

```

lemma set-embed-pmf: set-pmf embed-pmf = {x. f x  $\neq$  0}

```

```

by(auto simp add: set-pmf-eq pmf-embed-pmf)

```

```

end

```

lemma *embed-pmf-transfer*:

rel-fun (eq-onp ($\lambda f. (\forall x. 0 \leq f x) \wedge (\int^+ x. \text{ennreal } (f x) \partial \text{count-space UNIV} = 1)$) pmf-as-measure.cr-pmf ($\lambda f. \text{density } (\text{count-space UNIV}) (\text{ennreal } \circ f)$) embed-pmf

by (auto simp: rel-fun-def eq-onp-def embed-pmf.transfer)

lemma *measure-pmf-eq-density*: *measure-pmf p = density (count-space UNIV) (pmf p)*

proof (*transfer, elim conjE*)

fix *M* :: 'a *measure* **assume** [*simp*]: *sets M = UNIV and ae: AE x in M. measure M {x} ≠ 0*

assume *prob-space M then interpret prob-space M .*

show *M = density (count-space UNIV) ($\lambda x. \text{ennreal } (\text{measure } M \{x\})$)*

proof (*rule measure-eqI*)

fix *A* :: 'a *set*

have ($\int^+ x. \text{ennreal } (\text{measure } M \{x\}) * \text{indicator } A x \partial \text{count-space UNIV} = (\int^+ x. \text{emeasure } M \{x\} * \text{indicator } (A \cap \{x. \text{measure } M \{x\} \neq 0\}) x \partial \text{count-space UNIV}$)

by (*auto intro!: nn-integral-cong simp: emeasure-eq-measure split: split-indicator*)

also have $\dots = (\int^+ x. \text{emeasure } M \{x\} \partial \text{count-space } (A \cap \{x. \text{measure } M \{x\} \neq 0\}))$

by (*subst nn-integral-restrict-space[symmetric] (auto simp: restrict-count-space)*)

also have $\dots = \text{emeasure } M (\bigcup x \in (A \cap \{x. \text{measure } M \{x\} \neq 0\}). \{x\})$

by (*intro emeasure-UN-countable[symmetric] countable-Int2 countable-support (auto simp: disjoint-family-on-def)*)

also have $\dots = \text{emeasure } M A$

using *ae by (intro emeasure-eq-AE) auto*

finally show *emeasure M A = emeasure (density (count-space UNIV) ($\lambda x. \text{ennreal } (\text{measure } M \{x\})$)) A*

using *emeasure-space-1 by (simp add: emeasure-density)*

qed *simp*

qed

lemma *td-pmf-embed-pmf*:

type-definition pmf embed-pmf {f::'a \Rightarrow real. ($\forall x. 0 \leq f x$) \wedge ($\int^+ x. \text{ennreal } (f x) \partial \text{count-space UNIV} = 1$)}

unfolding *type-definition-def*

proof *safe*

fix *p* :: 'a *pmf*

have ($\int^+ x. 1 \partial \text{measure-pmf } p = 1$)

using *measure-pmf.emeasure-space-1[of p] by simp*

then show *: ($\int^+ x. \text{ennreal } (\text{pmf } p x) \partial \text{count-space UNIV} = 1$)

by (*simp add: measure-pmf-eq-density nn-integral-density pmf-nonneg del: nn-integral-const*)

show *embed-pmf (pmf p) = p*

by (*intro measure-pmf-inject[THEN iffD1]*)

*(simp add: * embed-pmf.rep-eq pmf-nonneg measure-pmf-eq-density[of p] comp-def)*

next

fix $f :: 'a \Rightarrow \text{real}$ **assume** $\forall x. 0 \leq f x$ $(\int^+ x. f x \ \partial \text{count-space UNIV}) = 1$
then show $\text{pmf} (\text{embed-pmf } f) = f$
by $(\text{auto intro!}: \text{pmf-embed-pmf})$
qed (rule pmf-nonneg)

end

lemma $\text{nn-integral-measure-pmf}: (\int^+ x. f x \ \partial \text{measure-pmf } p) = \int^+ x. \text{ennreal}$
 $(\text{pmf } p \ x) * f x \ \partial \text{count-space UNIV}$
by $(\text{simp add}: \text{measure-pmf-eq-density nn-integral-density pmf-nonneg})$

lemma $\text{integral-measure-pmf}:$

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$
assumes $A: \text{finite } A$
shows $(\bigwedge a. a \in \text{set-pmf } M \implies f a \neq 0 \implies a \in A) \implies (\text{LINT } x|M. f x) =$
 $(\sum a \in A. \text{pmf } M \ a *_{\mathbb{R}} f a)$
unfolding $\text{measure-pmf-eq-density}$
apply $(\text{simp add}: \text{integral-density})$
apply $(\text{subst lebesgue-integral-count-space-finite-support})$
apply $(\text{auto intro!}: \text{finite-subset}[OF - \langle \text{finite } A \rangle] \text{sum.mono-neutral-left simp}:$
 $\text{pmf-eq-0-set-pmf})$
done

lemma $\text{expectation-return-pmf} [\text{simp}]:$

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$
shows $\text{measure-pmf.expectation} (\text{return-pmf } x) f = f x$
by $(\text{subst integral-measure-pmf}[of \{x\}]) \text{simp-all}$

lemma $\text{pmf-expectation-bind}:$

fixes $p :: 'a \ \text{pmf}$ **and** $f :: 'a \Rightarrow 'b \ \text{pmf}$
and $h :: 'b \Rightarrow 'c::\{\text{banach, second-countable-topology}\}$
assumes $\text{finite } A \ \bigwedge x. x \in A \implies \text{finite} (\text{set-pmf} (f x)) \ \text{set-pmf } p \subseteq A$
shows $\text{measure-pmf.expectation} (p \gg f) h =$
 $(\sum a \in A. \text{pmf } p \ a *_{\mathbb{R}} \text{measure-pmf.expectation} (f a) h)$

proof –

have $\text{measure-pmf.expectation} (p \gg f) h = (\sum a \in (\bigcup x \in A. \text{set-pmf} (f x)). \text{pmf}$
 $(p \gg f) \ a *_{\mathbb{R}} h \ a)$

using $\text{assms by} (\text{intro integral-measure-pmf}) \text{auto}$

also have $\dots = (\sum x \in (\bigcup x \in A. \text{set-pmf} (f x)). (\sum a \in A. (\text{pmf } p \ a * \text{pmf} (f a)$
 $x) *_{\mathbb{R}} h \ x))$

proof $(\text{intro sum.cong refl, goal-cases})$

case $(1 \ x)$

thus $?case$

by $(\text{subst pmf-bind, subst integral-measure-pmf}[of A])$

$(\text{insert assms, auto simp}: \text{scaleR-sum-left})$

qed

also have $\dots = (\sum j \in A. \text{pmf } p \ j *_{\mathbb{R}} (\sum i \in (\bigcup x \in A. \text{set-pmf} (f x)). \text{pmf} (f j) \ i$
 $*_{\mathbb{R}} h \ i))$

```

    by (subst sum.commute) (simp add: scaleR-sum-right)
  also have ... = (∑ j∈A. pmf p j *R measure-pmf.expectation (f j) h)
  proof (intro sum.cong refl, goal-cases)
    case (1 x)
    thus ?case
      by (subst integral-measure-pmf[of (∪ x∈A. set-pmf (f x))])
         (insert assms, auto simp: scaleR-sum-left)
  qed
  finally show ?thesis .
qed

lemma continuous-on-LINT-pmf: — This is dominated convergence!
  fixes f :: 'i ⇒ 'a::topological-space ⇒ 'b::{banach, second-countable-topology}
  assumes f: ∧i. i ∈ set-pmf M ⇒ continuous-on A (f i)
  and bnd: ∧a i. a ∈ A ⇒ i ∈ set-pmf M ⇒ norm (f i a) ≤ B
  shows continuous-on A (λa. LINT i|M. f i a)
proof cases
  assume finite M with f show ?thesis
    using integral-measure-pmf[OF ⟨finite M⟩]
    by (subst integral-measure-pmf[OF ⟨finite M⟩])
       (auto intro!: continuous-on-sum continuous-on-scaleR continuous-on-const)
next
  assume infinite M
  let ?f = λi x. pmf (map-pmf (to-nat-on M) M) i *R f (from-nat-into M i) x

  show ?thesis
  proof (rule uniform-limit-theorem)
    show ∀F n in sequentially. continuous-on A (λa. ∑ i<n. ?f i a)
    by (intro always-eventually allI continuous-on-sum continuous-on-scaleR
        continuous-on-const f
            from-nat-into set-pmf-not-empty)
    show uniform-limit A (λn a. ∑ i<n. ?f i a) (λa. LINT i|M. f i a) sequentially
    proof (subst uniform-limit-cong[where g=λn a. ∑ i<n. ?f i a])
      fix a assume a ∈ A
      have 1: (LINT i|M. f i a) = (LINT i|map-pmf (to-nat-on M) M. f
        (from-nat-into M i) a)
      by (auto intro!: integral-cong-AE AE-pmfI)
      have 2: ... = (LINT i|count-space UNIV. pmf (map-pmf (to-nat-on M) M)
        i *R f (from-nat-into M i) a)
      by (simp add: measure-pmf-eq-density integral-density)
      have (λn. ?f n a) sums (LINT i|M. f i a)
      unfolding 1 2
    proof (intro sums-integral-count-space-nat)
      have A: integrable M (λi. f i a)
      using ⟨a∈A⟩ by (auto intro!: measure-pmf.integrable-const-bound AE-pmfI
        bnd)
      have integrable (map-pmf (to-nat-on M) M) (λi. f (from-nat-into M i) a)
      by (auto simp add: map-pmf-rep-eq integrable-distr-eq intro!: AE-pmfI
        integrable-cong-AE-imp[OF A])
    end
  end
end

```



```

    then show integrable (count-space UNIV) (λn. ?f n a)
      by (simp add: measure-pmf-eq-density integrable-density)
    qed
    then show (LINT i|M. f i a) = (∑ n. ?f n a)
      by (simp add: sums-unique)
  next
  show uniform-limit A (λn a. ∑ i<n. ?f i a) (λa. (∑ n. ?f n a)) sequentially
  proof (rule weierstrass-m-test)
    fix n a assume a∈A
    then show norm (?f n a) ≤ pmf (map-pmf (to-nat-on M) M) n * B
    using bnd by (auto intro!: mult-mono simp: from-nat-into set-pmf-not-empty)
  next
  have integrable (map-pmf (to-nat-on M) M) (λn. B)
    by auto
  then show summable (λn. pmf (map-pmf (to-nat-on (set-pmf M)) M) n
* B)
    by (simp add: measure-pmf-eq-density integrable-density integrable-count-space-nat-iff
summable-rabs-cancel)
  qed
  qed simp
  qed simp
qed

```

lemma continuous-on-LBINT:

```

  fixes f :: real ⇒ real
  assumes f: ∧b. a ≤ b ⇒ set-integrable lborel {a..b} f
  shows continuous-on UNIV (λb. LBINT x:{a..b}. f x)
proof (subst set-borel-integral-eq-integral)
  { fix b :: real assume a ≤ b
    from f[OF this] have continuous-on {a..b} (λb. integral {a..b} f)
      by (intro indefinite-integral-continuous-1 set-borel-integral-eq-integral) }
  note * = this

  have continuous-on (∪ b∈{a..}. {a <..}) (λb. integral {a..b} f)
  proof (intro continuous-on-open-UN)
    show b ∈ {a..} ⇒ continuous-on {a<..}) (λb. integral {a..b} f) for b
      using *[of b] by (rule continuous-on-subset) auto
  qed simp
  also have (∪ b∈{a..}. {a <..}) = {a <..}}
    by (auto simp: lt-ex gt-ex less-imp-le) (simp add: Bex-def less-imp-le gt-ex cong:
rev-conj-cong)
  finally have continuous-on {a+1 ..} (λb. integral {a..b} f)
    by (rule continuous-on-subset) auto
  moreover have continuous-on {a..a+1} (λb. integral {a..b} f)
    by (rule *) simp
  moreover
  have x ≤ a ⇒ {a..x} = (if a = x then {a} else {}) for x
    by auto
  then have continuous-on {..a} (λb. integral {a..b} f)

```

```

  by (subst continuous-on-cong[OF refl, where g= $\lambda x. 0$ ]) (auto intro!: continuous-on-const)
  ultimately have continuous-on ( $\{..a\} \cup \{a..a+1\} \cup \{a+1 ..\}$ ) ( $\lambda b. \text{integral } \{a..b\} f$ )
  by (intro continuous-on-closed-Un) auto
  also have  $\{..a\} \cup \{a..a+1\} \cup \{a+1 ..\} = \text{UNIV}$ 
  by auto
  finally show continuous-on UNIV ( $\lambda b. \text{integral } \{a..b\} f$ )
  by auto
next
  show set-integrable lborel  $\{a..b\} f$  for  $b$ 
  using  $f$  by (cases  $a \leq b$ ) auto
qed

```

```

locale pmf-as-function
begin

```

```

  setup-lifting td-pmf-embed-pmf

```

```

lemma set-pmf-transfer[transfer-rule]:
  assumes bi-total A
  shows rel-fun (pcr-pmf A) (rel-set A) ( $\lambda f. \{x. f x \neq 0\}$ ) set-pmf
  using (bi-total A)
  by (auto simp: pcr-pmf-def cr-pmf-def rel-fun-def rel-set-def bi-total-def Bex-def
  set-pmf-iff)
  metis+

```

```

end

```

```

context
begin

```

```

interpretation pmf-as-function .

```

```

lemma pmf-eqI: ( $\bigwedge i. \text{pmf } M i = \text{pmf } N i$ )  $\implies M = N$ 
  by transfer auto

```

```

lemma pmf-eq-iff:  $M = N \iff (\forall i. \text{pmf } M i = \text{pmf } N i)$ 
  by (auto intro: pmf-eqI)

```

```

lemma pmf-neq-exists-less:
  assumes  $M \neq N$ 
  shows  $\exists x. \text{pmf } M x < \text{pmf } N x$ 
proof (rule ccontr)
  assume  $\neg(\exists x. \text{pmf } M x < \text{pmf } N x)$ 
  hence  $ge: \text{pmf } M x \geq \text{pmf } N x$  for  $x$  by (auto simp: not-less)
  from  $assms$  obtain  $x$  where  $\text{pmf } M x \neq \text{pmf } N x$  by (auto simp: pmf-eq-iff)
  with  $ge[of x]$  have  $gt: \text{pmf } M x > \text{pmf } N x$  by simp
  have  $1 = \text{measure } (\text{measure-pmf } M) \text{ UNIV}$  by simp
  also have  $\dots = \text{measure } (\text{measure-pmf } N) \{x\} + \text{measure } (\text{measure-pmf } N)$ 

```

$(UNIV - \{x\})$
by (*subst measure-pmf.finite-measure-Union [symmetric]*) *simp-all*
also from *gt have measure (measure-pmf N) {x} < measure (measure-pmf M) {x}*
by (*simp add: measure-pmf-single*)
also have *measure (measure-pmf N) (UNIV - {x}) ≤ measure (measure-pmf M) (UNIV - {x})*
by (*subst (1 2) integral-pmf [symmetric]*)
(intro integral-mono integrable-pmf, simp-all add: ge)
also have *measure (measure-pmf M) {x} + ... = 1*
by (*subst measure-pmf.finite-measure-Union [symmetric]*) *simp-all*
finally show *False by simp-all*
qed

lemma *bind-commute-pmf: bind-pmf A (λx. bind-pmf B (C x)) = bind-pmf B (λy. bind-pmf A (λx. C x y))*

unfolding *pmf-eq-iff pmf-bind*

proof

fix *i*

interpret *B: prob-space restrict-space B B*

by (*intro prob-space-restrict-space measure-pmf.emmeasure-eq-1-AE*)
(auto simp: AE-measure-pmf-iff)

interpret *A: prob-space restrict-space A A*

by (*intro prob-space-restrict-space measure-pmf.emmeasure-eq-1-AE*)
(auto simp: AE-measure-pmf-iff)

interpret *AB: pair-prob-space restrict-space A A restrict-space B B*

by *unfold-locales*

have $(\int x. \int y. \text{pmf } (C x y) \ i \ \partial B \ \partial A) = (\int x. (\int y. \text{pmf } (C x y) \ i \ \partial \text{restrict-space } B \ B) \ \partial A)$

by (*rule Bochner-Integration.integral-cong*) *(auto intro!: integral-pmf-restrict)*

also have $\dots = (\int x. (\int y. \text{pmf } (C x y) \ i \ \partial \text{restrict-space } B \ B) \ \partial \text{restrict-space } A \ A)$

by (*intro integral-pmf-restrict B.borel-measurable-lebesgue-integral measurable-pair-restrict-pmf2 countable-set-pmf borel-measurable-count-space*)

also have $\dots = (\int y. \int x. \text{pmf } (C x y) \ i \ \partial \text{restrict-space } A \ A \ \partial \text{restrict-space } B \ B)$

by (*rule AB.Fubini-integral[symmetric]*)

(auto intro!: AB.integrable-const-bound[where B=1] measurable-pair-restrict-pmf2 simp: pmf-nonneg pmf-le-1 measurable-restrict-space1)

also have $\dots = (\int y. \int x. \text{pmf } (C x y) \ i \ \partial \text{restrict-space } A \ A \ \partial B)$

by (*intro integral-pmf-restrict[symmetric] A.borel-measurable-lebesgue-integral measurable-pair-restrict-pmf2 countable-set-pmf borel-measurable-count-space*)

also have $\dots = (\int y. \int x. \text{pmf } (C x y) \ i \ \partial A \ \partial B)$

by (*rule Bochner-Integration.integral-cong*) *(auto intro!: integral-pmf-restrict[symmetric])*

finally show $(\int x. \int y. \text{pmf } (C x y) \ i \ \partial B \ \partial A) = (\int y. \int x. \text{pmf } (C x y) \ i \ \partial A \ \partial B)$.

qed

lemma *pair-map-pmf1*: $\text{pair-pmf } (\text{map-pmf } f \ A) \ B = \text{map-pmf } (\text{apfst } f) \ (\text{pair-pmf } A \ B)$

proof (*safe intro!*: *pmf-eqI*)

fix $a :: 'a$ **and** $b :: 'b$

have [*simp*]: $\bigwedge c \ d. \text{indicator } (\text{apfst } f \ -' \ \{(a, b)\}) \ (c, d) = \text{indicator } (f \ -' \ \{a\}) \ c \ * \ (\text{indicator } \{b\} \ d :: \text{ennreal})$

by (*auto split*: *split-indicator*)

have $\text{ennreal } (\text{pmf } (\text{pair-pmf } (\text{map-pmf } f \ A) \ B) \ (a, b)) = \text{ennreal } (\text{pmf } (\text{map-pmf } (\text{apfst } f) \ (\text{pair-pmf } A \ B)) \ (a, b))$

unfolding *pmf-pair ennreal-pmf-map*

by (*simp add*: *nn-integral-pair-pmf' max-def emeasure-pmf-single nn-integral-multc pmf-nonneg*)

emeasure-map-pmf[symmetric] ennreal-mult del: emeasure-map-pmf)

then show $\text{pmf } (\text{pair-pmf } (\text{map-pmf } f \ A) \ B) \ (a, b) = \text{pmf } (\text{map-pmf } (\text{apfst } f) \ (\text{pair-pmf } A \ B)) \ (a, b)$

by (*simp add*: *pmf-nonneg*)

qed

lemma *pair-map-pmf2*: $\text{pair-pmf } A \ (\text{map-pmf } f \ B) = \text{map-pmf } (\text{apsnd } f) \ (\text{pair-pmf } A \ B)$

proof (*safe intro!*: *pmf-eqI*)

fix $a :: 'a$ **and** $b :: 'b$

have [*simp*]: $\bigwedge c \ d. \text{indicator } (\text{apsnd } f \ -' \ \{(a, b)\}) \ (c, d) = \text{indicator } \{a\} \ c \ * \ (\text{indicator } (f \ -' \ \{b\}) \ d :: \text{ennreal})$

by (*auto split*: *split-indicator*)

have $\text{ennreal } (\text{pmf } (\text{pair-pmf } A \ (\text{map-pmf } f \ B)) \ (a, b)) = \text{ennreal } (\text{pmf } (\text{map-pmf } (\text{apsnd } f) \ (\text{pair-pmf } A \ B)) \ (a, b))$

unfolding *pmf-pair ennreal-pmf-map*

by (*simp add*: *nn-integral-pair-pmf' max-def emeasure-pmf-single nn-integral-cmult nn-integral-multc pmf-nonneg*)

emeasure-map-pmf[symmetric] ennreal-mult del: emeasure-map-pmf)

then show $\text{pmf } (\text{pair-pmf } A \ (\text{map-pmf } f \ B)) \ (a, b) = \text{pmf } (\text{map-pmf } (\text{apsnd } f) \ (\text{pair-pmf } A \ B)) \ (a, b)$

by (*simp add*: *pmf-nonneg*)

qed

lemma *map-pair*: $\text{map-pmf } (\lambda(a, b). (f \ a, \ g \ b)) \ (\text{pair-pmf } A \ B) = \text{pair-pmf } (\text{map-pmf } f \ A) \ (\text{map-pmf } g \ B)$

by (*simp add*: *pair-map-pmf2 pair-map-pmf1 map-pmf-comp split-beta'*)

end

lemma *pair-return-pmf1*: $\text{pair-pmf } (\text{return-pmf } x) \ y = \text{map-pmf } (\text{Pair } x) \ y$

by (*simp add*: *pair-pmf-def bind-return-pmf map-pmf-def*)

lemma *pair-return-pmf2*: $\text{pair-pmf } x \text{ (return-pmf } y) = \text{map-pmf } (\lambda x. (x, y)) \ x$
by(*simp add: pair-pmf-def bind-return-pmf map-pmf-def*)

lemma *pair-pair-pmf*: $\text{pair-pmf } (\text{pair-pmf } u \ v) \ w = \text{map-pmf } (\lambda(x, (y, z)). ((x, y), z)) \ (\text{pair-pmf } u \ (\text{pair-pmf } v \ w))$
by(*simp add: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf*)

lemma *pair-commute-pmf*: $\text{pair-pmf } x \ y = \text{map-pmf } (\lambda(x, y). (y, x)) \ (\text{pair-pmf } y \ x)$

unfolding *pair-pmf-def* **by**(*subst bind-commute-pmf*)(*simp add: map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *set-pmf-subset-singleton*: $\text{set-pmf } p \subseteq \{x\} \longleftrightarrow p = \text{return-pmf } x$

proof(*intro iffI pmf-eqI*)

fix *i*

assume *x*: $\text{set-pmf } p \subseteq \{x\}$

hence *: $\text{set-pmf } p = \{x\}$ **using** *set-pmf-not-empty*[of *p*] **by** *auto*

have *ennreal* (*pmf* *p* *x*) = $\int^+ i. \text{indicator } \{x\} \ i \ \partial p$ **by**(*simp add: emeasure-pmf-single*)

also have $\dots = \int^+ i. 1 \ \partial p$ **by**(*rule nn-integral-cong-AE*)(*simp add: AE-measure-pmf-iff **)

also have $\dots = 1$ **by** *simp*

finally show *pmf* *p* *i* = *pmf* (*return-pmf* *x*) *i* **using** *x*

by(*auto split: split-indicator simp add: pmf-eq-0-set-pmf*)

qed *auto*

lemma *bind-eq-return-pmf*:

$\text{bind-pmf } p \ f = \text{return-pmf } x \longleftrightarrow (\forall y \in \text{set-pmf } p. f \ y = \text{return-pmf } x)$

(*is ?lhs* \longleftrightarrow *?rhs*)

proof(*intro iffI strip*)

fix *y*

assume *y*: $y \in \text{set-pmf } p$

assume *?lhs*

hence $\text{set-pmf } (\text{bind-pmf } p \ f) = \{x\}$ **by** *simp*

hence $(\bigcup y \in \text{set-pmf } p. \text{set-pmf } (f \ y)) = \{x\}$ **by** *simp*

hence $\text{set-pmf } (f \ y) \subseteq \{x\}$ **using** *y* **by** *auto*

thus $f \ y = \text{return-pmf } x$ **by**(*simp add: set-pmf-subset-singleton*)

next

assume *: *?rhs*

show *?lhs*

proof(*rule pmf-eqI*)

fix *i*

have *ennreal* (*pmf* (*bind-pmf* *p* *f*) *i*) = $\int^+ y. \text{ennreal } (\text{pmf } (f \ y) \ i) \ \partial p$

by (*simp add: ennreal-pmf-bind*)

also have $\dots = \int^+ y. \text{ennreal } (\text{pmf } (\text{return-pmf } x) \ i) \ \partial p$

by(*rule nn-integral-cong-AE*)(*simp add: AE-measure-pmf-iff **)

also have $\dots = \text{ennreal } (\text{pmf } (\text{return-pmf } x) \ i)$

by *simp*

finally show *pmf* (*bind-pmf* *p* *f*) *i* = *pmf* (*return-pmf* *x*) *i*

by (*simp add: pmf-nonneg*)

qed
qed

lemma *pmf-False-conv-True*: $\text{pmf } p \text{ False} = 1 - \text{pmf } p \text{ True}$

proof –

have $\text{pmf } p \text{ False} + \text{pmf } p \text{ True} = \text{measure } p \{ \text{False} \} + \text{measure } p \{ \text{True} \}$

by (*simp add: measure-pmf-single*)

also have $\dots = \text{measure } p (\{ \text{False} \} \cup \{ \text{True} \})$

by (*subst measure-pmf.finite-measure-Union*) *simp-all*

also have $\{ \text{False} \} \cup \{ \text{True} \} = \text{space } p$ **by** *auto*

finally show *?thesis* **by** *simp*

qed

lemma *pmf-True-conv-False*: $\text{pmf } p \text{ True} = 1 - \text{pmf } p \text{ False}$

by (*simp add: pmf-False-conv-True*)

22.4 Conditional Probabilities

lemma *measure-pmf-zero-iff*: $\text{measure } (\text{measure-pmf } p) s = 0 \iff \text{set-pmf } p \cap s = \{ \}$

by (*subst measure-pmf.prob-eq-0*) (*auto simp: AE-measure-pmf-iff*)

context

fixes $p :: 'a \text{ pmf}$ **and** $s :: 'a \text{ set}$

assumes *not-empty*: $\text{set-pmf } p \cap s \neq \{ \}$

begin

interpretation *pmf-as-measure* .

lemma *emeasure-measure-pmf-not-zero*: $\text{emeasure } (\text{measure-pmf } p) s \neq 0$

proof

assume $\text{emeasure } (\text{measure-pmf } p) s = 0$

then have *AE* x *in* $\text{measure-pmf } p$. $x \notin s$

by (*rule AE-I[rotated]*) *auto*

with *not-empty* **show** *False*

by (*auto simp: AE-measure-pmf-iff*)

qed

lemma *measure-measure-pmf-not-zero*: $\text{measure } (\text{measure-pmf } p) s \neq 0$

using *emeasure-measure-pmf-not-zero* **by** (*simp add: measure-pmf.emeasure-eq-measure measure-nonneg*)

lift-definition *cond-pmf* $:: 'a \text{ pmf}$ **is**

uniform-measure ($\text{measure-pmf } p$) s

proof (*intro conjI*)

show *prob-space* ($\text{uniform-measure } (\text{measure-pmf } p) s$)

by (*intro prob-space-uniform-measure*) (*auto simp: emeasure-measure-pmf-not-zero*)

show *AE* x *in* $\text{uniform-measure } (\text{measure-pmf } p) s$. $\text{measure } (\text{uniform-measure } (\text{measure-pmf } p) s) \{x\} \neq 0$

by (*simp add: emeasure-measure-pmf-not-zero measure-measure-pmf-not-zero AE-uniform-measure*

AE-measure-pmf-iff set-pmf.rep-eq less-top[symmetric])

qed *simp*

lemma *pmf-cond*: $\text{pmf cond-pmf } x = (\text{if } x \in s \text{ then pmf } p \ x / \text{measure } p \ s \text{ else } 0)$

by *transfer (simp add: emeasure-measure-pmf-not-zero pmf.rep-eq)*

lemma *set-cond-pmf*[*simp*]: $\text{set-pmf cond-pmf} = \text{set-pmf } p \cap s$

by (*auto simp add: set-pmf-iff pmf-cond measure-measure-pmf-not-zero split: if-split-asm*)

end

lemma *measure-pmf-posI*: $x \in \text{set-pmf } p \implies x \in A \implies \text{measure-pmf.prob } p \ A > 0$

using *measure-measure-pmf-not-zero[of p A]* **by** (*subst zero-less-measure-iff*) *blast*

lemma *cond-map-pmf*:

assumes *set-pmf p* $\cap f \text{ -' } s \neq \{\}$

shows $\text{cond-pmf (map-pmf } f \ p) \ s = \text{map-pmf } f \ (\text{cond-pmf } p \ (f \text{ -' } s))$

proof –

have $*$: $\text{set-pmf (map-pmf } f \ p) \cap s \neq \{\}$

using *assms by auto*

{ **fix** x

have $\text{ennreal (pmf (map-pmf } f \ (\text{cond-pmf } p \ (f \text{ -' } s))) \ x) = \text{emeasure } p \ (f \text{ -' } s \cap f \text{ -' } \{x\}) / \text{emeasure } p \ (f \text{ -' } s)$

unfolding *ennreal-pmf-map cond-pmf.rep-eq[OF assms]* **by** (*simp add: nn-integral-uniform-measure*)

also have $f \text{ -' } s \cap f \text{ -' } \{x\} = (\text{if } x \in s \text{ then } f \text{ -' } \{x\} \text{ else } \{\})$

by *auto*

also have $\text{emeasure } p \ (\text{if } x \in s \text{ then } f \text{ -' } \{x\} \text{ else } \{\}) / \text{emeasure } p \ (f \text{ -' } s)$

=

$\text{ennreal (pmf (cond-pmf (map-pmf } f \ p) \ s) \ x)$

using *measure-measure-pmf-not-zero[OF *]*

by (*simp add: pmf-cond[OF *] ennreal-pmf-map measure-pmf.emeasure-eq-measure divide-ennreal pmf-nonneg measure-nonneg zero-less-measure-iff*

pmf-map)

finally have $\text{ennreal (pmf (cond-pmf (map-pmf } f \ p) \ s) \ x) = \text{ennreal (pmf (map-pmf } f \ (\text{cond-pmf } p \ (f \text{ -' } s))) \ x)$

by *simp* }

then show *?thesis*

by (*intro pmf-eqI*) (*simp add: pmf-nonneg*)

qed

lemma *bind-cond-pmf-cancel*:

assumes [*simp*]: $\bigwedge x. x \in \text{set-pmf } p \implies \text{set-pmf } q \cap \{y. R \ x \ y\} \neq \{\}$

assumes [*simp*]: $\bigwedge y. y \in \text{set-pmf } q \implies \text{set-pmf } p \cap \{x. R \ x \ y\} \neq \{\}$

assumes [*simp*]: $\bigwedge x \ y. x \in \text{set-pmf } p \implies y \in \text{set-pmf } q \implies R \ x \ y \implies \text{measure } q \ \{y. R \ x \ y\} = \text{measure } p \ \{x. R \ x \ y\}$

```

shows bind-pmf p (λx. cond-pmf q {y. R x y}) = q
proof (rule pmf-eqI)
  fix i
  have ennreal (pmf (bind-pmf p (λx. cond-pmf q {y. R x y})) i) =
    (∫+x. ennreal (pmf q i / measure p {x. R x i}) * ennreal (indicator {x. R x
i} x) ∂p)
  by (auto simp add: ennreal-pmf-bind AE-measure-pmf-iff pmf-cond pmf-eq-0-set-pmf
pmf-nonneg measure-nonneg
      intro!: nn-integral-cong-AE)
  also have ... = (pmf q i * measure p {x. R x i}) / measure p {x. R x i}
  by (simp add: pmf-nonneg measure-nonneg zero-ennreal-def[symmetric] ennreal-indicator
      nn-integral-cmult measure-pmf.emmeasure-eq-measure ennreal-mult[symmetric])
  also have ... = pmf q i
  by (cases pmf q i = 0)
    (simp-all add: pmf-eq-0-set-pmf measure-measure-pmf-not-zero pmf-nonneg)
  finally show pmf (bind-pmf p (λx. cond-pmf q {y. R x y})) i = pmf q i
  by (simp add: pmf-nonneg)
qed

```

22.5 Relator

```

inductive rel-pmf :: ('a ⇒ 'b ⇒ bool) ⇒ 'a pmf ⇒ 'b pmf ⇒ bool
for R p q
where
  [ [ ∧x y. (x, y) ∈ set-pmf pq ⇒ R x y;
    map-pmf fst pq = p; map-pmf snd pq = q ] ]
  ⇒ rel-pmf R p q

```

lemma rel-pmfI:

```

assumes R: rel-set R (set-pmf p) (set-pmf q)
assumes eq: ∧x y. x ∈ set-pmf p ⇒ y ∈ set-pmf q ⇒ R x y ⇒
  measure p {x. R x y} = measure q {y. R x y}
shows rel-pmf R p q

```

proof

```

let ?pq = bind-pmf p (λx. bind-pmf (cond-pmf q {y. R x y}) (λy. return-pmf
(x, y)))

```

```

have ∧x. x ∈ set-pmf p ⇒ set-pmf q ∩ {y. R x y} ≠ {}

```

```

  using R by (auto simp: rel-set-def)

```

```

then show ∧x y. (x, y) ∈ set-pmf ?pq ⇒ R x y

```

```

  by auto

```

```

show map-pmf fst ?pq = p

```

```

  by (simp add: map-bind-pmf bind-return-pmf')

```

```

show map-pmf snd ?pq = q

```

```

  using R eq

```

```

  apply (simp add: bind-cond-pmf-cancel map-bind-pmf bind-return-pmf')

```

```

  apply (rule bind-cond-pmf-cancel)

```

```

  apply (auto simp: rel-set-def)

```

```

done

```


qed

lemma *rel-pmf-imp-rel-set*: $rel\text{-}pmf\ R\ p\ q \implies rel\text{-}set\ R\ (set\text{-}pmf\ p)\ (set\text{-}pmf\ q)$
 by (*force simp add: rel-pmf.simps rel-set-def*)

lemma *rel-pmfD-measure*:

assumes *rel-R*: $rel\text{-}pmf\ R\ p\ q$ **and** *R*: $\bigwedge a\ b. R\ a\ b \implies R\ a\ y \longleftrightarrow R\ x\ b$

assumes $x \in set\text{-}pmf\ p\ y \in set\text{-}pmf\ q$

shows $measure\ p\ \{x. R\ x\ y\} = measure\ q\ \{y. R\ x\ y\}$

proof –

from *rel-R* **obtain** *pq* **where** $pq: \bigwedge x\ y. (x, y) \in set\text{-}pmf\ pq \implies R\ x\ y$

and *eq*: $p = map\text{-}pmf\ fst\ pq\ q = map\text{-}pmf\ snd\ pq$

by (*auto elim: rel-pmf.cases*)

have $measure\ p\ \{x. R\ x\ y\} = measure\ pq\ \{x. R\ (fst\ x)\ y\}$

by (*simp add: eq map-pmf-rep-eq measure-distr*)

also have $\dots = measure\ pq\ \{y. R\ x\ (snd\ y)\}$

by (*intro measure-pmf.finite-measure-eq-AE*)

(*auto simp: AE-measure-pmf-iff R dest!: pq*)

also have $\dots = measure\ q\ \{y. R\ x\ y\}$

by (*simp add: eq map-pmf-rep-eq measure-distr*)

finally show $measure\ p\ \{x. R\ x\ y\} = measure\ q\ \{y. R\ x\ y\}$.

qed

lemma *rel-pmf-measureD*:

assumes $rel\text{-}pmf\ R\ p\ q$

shows $measure\ (measure\text{-}pmf\ p)\ A \leq measure\ (measure\text{-}pmf\ q)\ \{y. \exists x \in A. R\ x\ y\}$ (*is ?lhs \leq ?rhs*)

using *assms*

proof *cases*

fix *pq*

assume *R*: $\bigwedge x\ y. (x, y) \in set\text{-}pmf\ pq \implies R\ x\ y$

and *p[symmetric]*: $map\text{-}pmf\ fst\ pq = p$

and *q[symmetric]*: $map\text{-}pmf\ snd\ pq = q$

have $?lhs = measure\ (measure\text{-}pmf\ pq)\ (fst\ -\ 'A)$ **by** (*simp add: p*)

also have $\dots \leq measure\ (measure\text{-}pmf\ pq)\ \{y. \exists x \in A. R\ x\ (snd\ y)\}$

by (*rule measure-pmf.finite-measure-mono-AE*) (*auto 4 3 simp add: AE-measure-pmf-iff dest: R*)

also have $\dots = ?rhs$ **by** (*simp add: q*)

finally show *thesis* .

qed

lemma *rel-pmf-iff-measure*:

assumes *symp R transp R*

shows $rel\text{-}pmf\ R\ p\ q \longleftrightarrow$

$rel\text{-}set\ R\ (set\text{-}pmf\ p)\ (set\text{-}pmf\ q) \wedge$

$(\forall x \in set\text{-}pmf\ p. \forall y \in set\text{-}pmf\ q. R\ x\ y \longrightarrow measure\ p\ \{x. R\ x\ y\} = measure\ q\ \{y. R\ x\ y\})$

by (*safe intro!: rel-pmf-imp-rel-set rel-pmfI*)

(*auto intro!: rel-pmfD-measure dest: sympD[OF (symp R)] transpD[OF (transp*

R))

lemma *quotient-rel-set-disjoint*:

$equivp\ R \implies C \in UNIV // \{(x, y). R\ x\ y\} \implies rel\text{-}set\ R\ A\ B \implies A \cap C = \{\}$
 $\longleftrightarrow B \cap C = \{\}$

using *in-quotient-imp-closed*[of $UNIV\ \{(x, y). R\ x\ y\}\ C$]

by (*auto* $0\ 0\ simp$: *equivp-equiv rel-set-def set-eq-iff elim: equivpE*)

(*blast dest: equivp-symp*) $+$

lemma *quotientD*: $equiv\ X\ R \implies A \in X // R \implies x \in A \implies A = R\ \{\ x\}$

by (*metis Image-singleton-iff equiv-class-eq-iff quotientE*)

lemma *rel-pmf-iff-equivp*:

assumes *equivp* R

shows $rel\text{-}pmf\ R\ p\ q \longleftrightarrow (\forall C \in UNIV // \{(x, y). R\ x\ y\}. measure\ p\ C = measure\ q\ C)$

(*is* $- \longleftrightarrow (\forall C \in // ?R. -)$)

proof (*subst rel-pmf-iff-measure, safe*)

show *symp* $R\ transp\ R$

using *assms* **by** (*auto simp: equivp-reflp-symp-transp*)

next

fix C **assume** $C: C \in UNIV // ?R$ **and** $R: rel\text{-}set\ R\ (set\text{-}pmf\ p)\ (set\text{-}pmf\ q)$

assume $eq: \forall x \in set\text{-}pmf\ p. \forall y \in set\text{-}pmf\ q. R\ x\ y \longrightarrow measure\ p\ \{x.\ R\ x\ y\} = measure\ q\ \{y.\ R\ x\ y\}$

show $measure\ p\ C = measure\ q\ C$

proof (*cases* $p \cap C = \{\}$)

case *True*

then have $q \cap C = \{\}$

using *quotient-rel-set-disjoint*[*OF assms C R*] **by** *simp*

with *True* **show** *?thesis*

unfolding *measure-pmf-zero-iff*[*symmetric*] **by** *simp*

next

case *False*

then have $q \cap C \neq \{\}$

using *quotient-rel-set-disjoint*[*OF assms C R*] **by** *simp*

with *False* **obtain** $x\ y$ **where** *in-set*: $x \in set\text{-}pmf\ p\ y \in set\text{-}pmf\ q$ **and** *in-C*:
 $x \in C\ y \in C$

by *auto*

then have $R\ x\ y$

using *in-quotient-imp-in-rel*[of $UNIV\ ?R\ C\ x\ y$] C *assms*

by (*simp add: equivp-equiv*)

with *in-set eq* **have** $measure\ p\ \{x.\ R\ x\ y\} = measure\ q\ \{y.\ R\ x\ y\}$

by *auto*

moreover have $\{y.\ R\ x\ y\} = C$

using *assms* ($x \in C$) C *quotientD*[of $UNIV\ ?R\ C\ x$] **by** (*simp add: equivp-equiv*)

moreover have $\{x.\ R\ x\ y\} = C$

using *assms* ($y \in C$) C *quotientD*[of $UNIV\ ?R\ C\ y$] *sympD*[of R]

by (*auto simp add: equivp-equiv elim: equivpE*)

```

    ultimately show ?thesis
      by auto
  qed
next
assume eq:  $\forall C \in UNIV // ?R. \text{measure } p \ C = \text{measure } q \ C$ 
show rel-set  $R$  (set-pmf  $p$ ) (set-pmf  $q$ )
  unfolding rel-set-def
proof safe
  fix  $x$  assume  $x: x \in \text{set-pmf } p$ 
  have  $\{y. R \ x \ y\} \in UNIV // ?R$ 
    by (auto simp: quotient-def)
  with eq have *:  $\text{measure } q \ \{y. R \ x \ y\} = \text{measure } p \ \{y. R \ x \ y\}$ 
    by auto
  have  $\text{measure } q \ \{y. R \ x \ y\} \neq 0$ 
    using  $x$  assms unfolding * by (auto simp: measure-pmf-zero-iff set-eq-iff
dest: equivp-reflp)
  then show  $\exists y \in \text{set-pmf } q. R \ x \ y$ 
    unfolding measure-pmf-zero-iff by auto
next
  fix  $y$  assume  $y: y \in \text{set-pmf } q$ 
  have  $\{x. R \ x \ y\} \in UNIV // ?R$ 
    using assms by (auto simp: quotient-def dest: equivp-symp)
  with eq have *:  $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } q \ \{x. R \ x \ y\}$ 
    by auto
  have  $\text{measure } p \ \{x. R \ x \ y\} \neq 0$ 
    using  $y$  assms unfolding * by (auto simp: measure-pmf-zero-iff set-eq-iff
dest: equivp-reflp)
  then show  $\exists x \in \text{set-pmf } p. R \ x \ y$ 
    unfolding measure-pmf-zero-iff by auto
qed

fix  $x \ y$  assume  $x \in \text{set-pmf } p \ y \in \text{set-pmf } q \ R \ x \ y$ 
have  $\{y. R \ x \ y\} \in UNIV // ?R \ \{x. R \ x \ y\} = \{y. R \ x \ y\}$ 
  using assms  $\langle R \ x \ y \rangle$  by (auto simp: quotient-def dest: equivp-symp equivp-transp)
with eq show  $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } q \ \{y. R \ x \ y\}$ 
  by auto
qed

bnf pmf: 'a pmf map: map-pmf sets: set-pmf bd : natLeq rel: rel-pmf
proof -
  show map-pmf id = id by (rule map-pmf-id)
  show  $\bigwedge f \ g. \text{map-pmf } (f \circ g) = \text{map-pmf } f \circ \text{map-pmf } g$  by (rule map-pmf-compose)
  show  $\bigwedge f \ g :: 'a \Rightarrow 'b. \bigwedge p. (\bigwedge x. x \in \text{set-pmf } p \Longrightarrow f \ x = g \ x) \Longrightarrow \text{map-pmf } f \ p =$ 
    map-pmf  $g \ p$ 
    by (intro map-pmf-cong refl)

  show  $\bigwedge f :: 'a \Rightarrow 'b. \text{set-pmf } \circ \text{map-pmf } f = \text{op } ' f \circ \text{set-pmf}$ 
    by (rule pmf-set-map)

```

```

show (card-of (set-pmf p), natLeq) ∈ ordLeq for p :: 's pmf
proof –
  have (card-of (set-pmf p), card-of (UNIV :: nat set)) ∈ ordLeq
    by (rule card-of-ordLeqI[where f=to-nat-on (set-pmf p)])
      (auto intro: countable-set-pmf)
  also have (card-of (UNIV :: nat set), natLeq) ∈ ordLeq
    by (metis Field-natLeq card-of-least natLeq-Well-order)
  finally show ?thesis .
qed

show  $\bigwedge R. \text{rel-pmf } R = (\lambda x y. \exists z. \text{set-pmf } z \subseteq \{(x, y). R x y\} \wedge$ 
   $\text{map-pmf fst } z = x \wedge \text{map-pmf snd } z = y)$ 
  by (auto simp add: fun-eq-iff rel-pmf.simps)

show rel-pmf R OO rel-pmf S ≤ rel-pmf (R OO S)
for R :: 'a ⇒ 'b ⇒ bool and S :: 'b ⇒ 'c ⇒ bool
proof –
  { fix p q r
    assume pq: rel-pmf R p q
      and qr: rel-pmf S q r
    from pq obtain pq where pq:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R x y$ 
      and p: p = map-pmf fst pq and q: q = map-pmf snd pq by cases auto
    from qr obtain qr where qr:  $\bigwedge y z. (y, z) \in \text{set-pmf } qr \implies S y z$ 
      and q': q = map-pmf fst qr and r: r = map-pmf snd qr by cases auto

    define pr where pr =
      bind-pmf pq ( $\lambda xy. \text{bind-pmf } (\text{cond-pmf } qr \{yz. \text{fst } yz = \text{snd } xy\})$ 
        ( $\lambda yz. \text{return-pmf } (\text{fst } xy, \text{snd } yz)$ ))
    have pr-welldefined:  $\bigwedge y. y \in q \implies qr \cap \{yz. \text{fst } yz = y\} \neq \{\}$ 
      by (force simp: q')

    have rel-pmf (R OO S) p r
    proof (rule rel-pmf.intros)
      fix x z assume (x, z) ∈ pr
      then have  $\exists y. (x, y) \in pq \wedge (y, z) \in qr$ 
        by (auto simp: q pr-welldefined pr-def split-beta)
      with pq qr show (R OO S) x z
        by blast
    next
      have map-pmf snd pr = map-pmf snd (bind-pmf q ( $\lambda y. \text{cond-pmf } qr \{yz. \text{fst } yz = y\}$ ))
        by (simp add: pr-def q split-beta bind-map-pmf map-pmf-def[symmetric]
          map-bind-pmf map-pmf-comp)
      then show map-pmf snd pr = r
        unfolding r q' bind-map-pmf by (subst (asm) bind-cond-pmf-cancel) (auto
          simp: eq-commute)
      qed (simp add: pr-def map-bind-pmf split-beta map-pmf-def[symmetric] p
        map-pmf-comp)
    }
  }

```

then show *?thesis*
by(*auto simp add: le-fun-def*)
qed
qed (*fact natLeq-card-order natLeq-cinfinite*)+

lemma *map-pmf-idI*: $(\bigwedge x. x \in \text{set-pmf } p \implies f x = x) \implies \text{map-pmf } f p = p$
by(*simp cong: pmf.map-cong*)

lemma *rel-pmf-conj*[*simp*]:
 $\text{rel-pmf } (\lambda x y. P \wedge Q x y) x y \longleftrightarrow P \wedge \text{rel-pmf } Q x y$
 $\text{rel-pmf } (\lambda x y. Q x y \wedge P) x y \longleftrightarrow P \wedge \text{rel-pmf } Q x y$
using *set-pmf-not-empty* **by** (*fastforce simp: pmf.in-rel subset-eq*)+

lemma *rel-pmf-top*[*simp*]: $\text{rel-pmf } \text{top} = \text{top}$
by (*auto simp: pmf.in-rel[abs-def] fun-eq-iff map-fst-pair-pmf map-snd-pair-pmf*
intro: exI[of - pair-pmf x y for x y])

lemma *rel-pmf-return-pmf1*: $\text{rel-pmf } R (\text{return-pmf } x) M \longleftrightarrow (\forall a \in M. R x a)$
proof *safe*

fix *a* **assume** $a \in M \text{ rel-pmf } R (\text{return-pmf } x) M$
then obtain *pq* **where** $*$: $\bigwedge a b. (a, b) \in \text{set-pmf } pq \implies R a b$
and *eq*: $\text{return-pmf } x = \text{map-pmf } \text{fst } pq M = \text{map-pmf } \text{snd } pq M$
by (*force elim: rel-pmf.cases*)
moreover have $\text{set-pmf } (\text{return-pmf } x) = \{x\}$
by *simp*
with $\langle a \in M \rangle$ **have** $(x, a) \in pq$
by (*force simp: eq*)
with $*$ **show** $R x a$
by *auto*

qed (*auto intro!: rel-pmf.intros[where pq=pair-pmf (return-pmf x) M]*
simp: map-fst-pair-pmf map-snd-pair-pmf)

lemma *rel-pmf-return-pmf2*: $\text{rel-pmf } R M (\text{return-pmf } x) \longleftrightarrow (\forall a \in M. R a x)$
by (*subst pmf.rel-flip[symmetric]*) (*simp add: rel-pmf-return-pmf1*)

lemma *rel-return-pmf*[*simp*]: $\text{rel-pmf } R (\text{return-pmf } x1) (\text{return-pmf } x2) = R x1 x2$

unfolding *rel-pmf-return-pmf2 set-return-pmf* **by** *simp*

lemma *rel-pmf-False*[*simp*]: $\text{rel-pmf } (\lambda x y. \text{False}) x y = \text{False}$
unfolding *pmf.in-rel fun-eq-iff* **using** *set-pmf-not-empty* **by** *fastforce*

lemma *rel-pmf-rel-prod*:

$\text{rel-pmf } (\text{rel-prod } R S) (\text{pair-pmf } A A') (\text{pair-pmf } B B') \longleftrightarrow \text{rel-pmf } R A B \wedge \text{rel-pmf } S A' B'$

proof *safe*

assume $\text{rel-pmf } (\text{rel-prod } R S) (\text{pair-pmf } A A') (\text{pair-pmf } B B')$
then obtain *pq* **where** pq : $\bigwedge a b c d. ((a, c), (b, d)) \in \text{set-pmf } pq \implies R a b \wedge S c d$

```

and eq: map-pmf fst pq = pair-pmf A A' map-pmf snd pq = pair-pmf B B'
by (force elim: rel-pmf.cases)
show rel-pmf R A B
proof (rule rel-pmf.intros)
  let ?f =  $\lambda(a, b). (fst\ a, fst\ b)$ 
  have [simp]:  $(\lambda x. fst\ (?f\ x)) = fst\ o\ fst\ (\lambda x. snd\ (?f\ x)) = fst\ o\ snd$ 
    by auto

  show map-pmf fst (map-pmf ?f pq) = A
    by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-fst-pair-pmf)
  show map-pmf snd (map-pmf ?f pq) = B
    by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-fst-pair-pmf)

  fix a b assume (a, b)  $\in$  set-pmf (map-pmf ?f pq)
  then obtain c d where ((a, c), (b, d))  $\in$  set-pmf pq
    by auto
  from pq[OF this] show R a b ..
qed
show rel-pmf S A' B'
proof (rule rel-pmf.intros)
  let ?f =  $\lambda(a, b). (snd\ a, snd\ b)$ 
  have [simp]:  $(\lambda x. fst\ (?f\ x)) = snd\ o\ fst\ (\lambda x. snd\ (?f\ x)) = snd\ o\ snd$ 
    by auto

  show map-pmf fst (map-pmf ?f pq) = A'
    by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-snd-pair-pmf)
  show map-pmf snd (map-pmf ?f pq) = B'
    by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-snd-pair-pmf)

  fix c d assume (c, d)  $\in$  set-pmf (map-pmf ?f pq)
  then obtain a b where ((a, c), (b, d))  $\in$  set-pmf pq
    by auto
  from pq[OF this] show S c d ..
qed
next
assume rel-pmf R A B rel-pmf S A' B'
then obtain Rpq Spq
  where Rpq:  $\bigwedge a\ b. (a, b) \in set-pmf\ Rpq \implies R\ a\ b$ 
    map-pmf fst Rpq = A map-pmf snd Rpq = B
    and Spq:  $\bigwedge a\ b. (a, b) \in set-pmf\ Spq \implies S\ a\ b$ 
    map-pmf fst Spq = A' map-pmf snd Spq = B'
  by (force elim: rel-pmf.cases)

  let ?f =  $(\lambda((a, c), (b, d)). ((a, b), (c, d)))$ 
  let ?pq = map-pmf ?f (pair-pmf Rpq Spq)
  have [simp]:  $(\lambda x. fst\ (?f\ x)) = (\lambda(a, b). (fst\ a, fst\ b))\ (\lambda x. snd\ (?f\ x)) = (\lambda(a,$ 
  b). (snd a, snd b))
    by auto

```

```

show rel-pmf (rel-prod R S) (pair-pmf A A') (pair-pmf B B')
  by (rule rel-pmf.intros[where pq=?pq])
      (auto simp: map-snd-pair-pmf map-fst-pair-pmf map-pmf-comp Rpq Spq
                map-pair)
qed

```

```

lemma rel-pmf-refl:
  assumes  $\bigwedge x. x \in \text{set-pmf } p \implies P x x$ 
  shows rel-pmf P p p
  by (rule rel-pmf.intros[where pq=map-pmf ( $\lambda x. (x, x)$ ) p])
      (auto simp add: pmf.map-comp o-def assms)

```

```

lemma rel-pmf-bij-betw:
  assumes f: bij-betw f (set-pmf p) (set-pmf q)
  and eq:  $\bigwedge x. x \in \text{set-pmf } p \implies \text{pmf } p x = \text{pmf } q (f x)$ 
  shows rel-pmf ( $\lambda x y. f x = y$ ) p q
proof(rule rel-pmf.intros)
  let ?pq = map-pmf ( $\lambda x. (x, f x)$ ) p
  show map-pmf fst ?pq = p by(simp add: pmf.map-comp o-def)

```

```

have map-pmf f p = q
proof(rule pmf-eqI)
  fix i
  show pmf (map-pmf f p) i = pmf q i
  proof(cases i  $\in$  set-pmf q)
    case True
    with f obtain j where i = f j j  $\in$  set-pmf p
    by(auto simp add: bij-betw-def image-iff)
    thus ?thesis using f by(simp add: bij-betw-def pmf-map-inj eq)
  next
    case False thus ?thesis
    by(subst pmf-map-outside)(auto simp add: set-pmf-iff eq[symmetric])
  qed
qed
then show map-pmf snd ?pq = q by(simp add: pmf.map-comp o-def)
qed auto

```

```

context
begin

```

```

interpretation pmf-as-measure .

```

```

definition join-pmf M = bind-pmf M ( $\lambda x. x$ )

```

```

lemma bind-eq-join-pmf: bind-pmf M f = join-pmf (map-pmf f M)
  unfolding join-pmf-def bind-map-pmf ..

```

```

lemma join-eq-bind-pmf: join-pmf M = bind-pmf M id
  by (simp add: join-pmf-def id-def)

```

lemma *pmf-join*: $\text{pmf } (\text{join-pmf } N) \ i = (\int M. \text{pmf } M \ i \ \partial\text{measure-pmf } N)$
unfolding *join-pmf-def pmf-bind ..*

lemma *ennreal-pmf-join*: $\text{ennreal } (\text{pmf } (\text{join-pmf } N) \ i) = (\int^+ M. \text{pmf } M \ i \ \partial\text{measure-pmf } N)$
unfolding *join-pmf-def ennreal-pmf-bind ..*

lemma *set-pmf-join-pmf[simp]*: $\text{set-pmf } (\text{join-pmf } f) = (\bigcup p \in \text{set-pmf } f. \text{set-pmf } p)$
by (*simp add: join-pmf-def*)

lemma *join-return-pmf*: $\text{join-pmf } (\text{return-pmf } M) = M$
by (*simp add: integral-return pmf-eq-iff pmf-join return-pmf.rep-eq*)

lemma *map-join-pmf*: $\text{map-pmf } f \ (\text{join-pmf } AA) = \text{join-pmf } (\text{map-pmf } (\text{map-pmf } f) \ AA)$
by (*simp add: join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *join-map-return-pmf*: $\text{join-pmf } (\text{map-pmf } \text{return-pmf } A) = A$
by (*simp add: join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf' ^*)

end

lemma *rel-pmf-joinI*:

assumes *rel-pmf* (*rel-pmf* *P*) *p q*
shows *rel-pmf* *P* (*join-pmf* *p*) (*join-pmf* *q*)

proof –

from *assms* **obtain** *pq* **where** *p*: $p = \text{map-pmf } \text{fst } pq$
and *q*: $q = \text{map-pmf } \text{snd } pq$
and *P*: $\bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies \text{rel-pmf } P \ x \ y$
by *cases auto*

from *P* **obtain** *PQ*

where *PQ*: $\bigwedge x \ y \ a \ b. \llbracket (x, y) \in \text{set-pmf } pq; (a, b) \in \text{set-pmf } (PQ \ x \ y) \rrbracket \implies P \ a \ b$

and *x*: $\bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies \text{map-pmf } \text{fst } (PQ \ x \ y) = x$
and *y*: $\bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies \text{map-pmf } \text{snd } (PQ \ x \ y) = y$
by(*metis rel-pmf.simps*)

let *?r* = $\text{bind-pmf } pq \ (\lambda(x, y). PQ \ x \ y)$

have $\bigwedge a \ b. (a, b) \in \text{set-pmf } ?r \implies P \ a \ b$ **by** (*auto intro: PQ*)

moreover **have** $\text{map-pmf } \text{fst } ?r = \text{join-pmf } p \ \text{map-pmf } \text{snd } ?r = \text{join-pmf } q$

by (*simp-all add: p q x y join-pmf-def map-bind-pmf bind-map-pmf split-def cong: bind-pmf-cong*)

ultimately show *?thesis ..*

qed

lemma *rel-pmf-bindI*:

assumes *pq*: *rel-pmf* *R* *p q*

and $fg: \bigwedge x y. R x y \implies \text{rel-pmf } P (f x) (g y)$
shows $\text{rel-pmf } P (\text{bind-pmf } p f) (\text{bind-pmf } q g)$
unfolding bind-eq-join-pmf
by $(\text{rule rel-pmf-joinI})$
 $(\text{auto simp add: pmf.rel-map intro: pmf.rel-mono}[\text{THEN le-funD}, \text{ THEN le-funD}, \text{ THEN le-boolD}, \text{ THEN mp}, \text{ OF - pq}] fg)$

Proof that rel-pmf preserves orders. Antisymmetry proof follows Thm. 1 in N. Saheb-Djahromi, Cpo’s of measures for nondeterminism, Theoretical Computer Science 12(1):19–37, 1980, [http://dx.doi.org/10.1016/0304-3975\(80\)90003-1](http://dx.doi.org/10.1016/0304-3975(80)90003-1)

lemma

assumes $*$: $\text{rel-pmf } R p q$
and refl : $\text{reflp } R$ **and** trans : $\text{transp } R$
shows measure-Ici : $\text{measure } p \{y. R x y\} \leq \text{measure } q \{y. R x y\}$ (**is** $?thesis1$)
and measure-Ioi : $\text{measure } p \{y. R x y \wedge \neg R y x\} \leq \text{measure } q \{y. R x y \wedge \neg R y x\}$ (**is** $?thesis2$)

proof –

from $*$ **obtain** pq

where pq : $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R x y$

and p : $p = \text{map-pmf fst } pq$

and q : $q = \text{map-pmf snd } pq$

by cases auto

show $?thesis1$ $?thesis2$ **unfolding** $p q \text{map-pmf-rep-eq}$ **using** refl trans

by $(\text{auto } 4 \ 3 \text{ simp add: measure-distr reflpD AE-measure-pmf-iff intro!: measure-pmf.finite-measure-mono-AL dest!: pq elim: transpE})$

qed

lemma rel-pmf-inf :

fixes $p q$:: ‘a pmf

assumes 1: $\text{rel-pmf } R p q$

assumes 2: $\text{rel-pmf } R q p$

and refl : $\text{reflp } R$ **and** trans : $\text{transp } R$

shows $\text{rel-pmf } (\text{inf } R R^{-1-1}) p q$

proof $(\text{subst rel-pmf-iff-equivp}, \text{ safe})$

show $\text{equivp } (\text{inf } R R^{-1-1})$

using trans refl **by** $(\text{auto simp: equivp-reflp-symp-transp intro: sympI transpI reflpI dest: transpD reflpD})$

fix C **assume** $C \in \text{UNIV} // \{(x, y). \text{inf } R R^{-1-1} x y\}$

then obtain x **where** C : $C = \{y. R x y \wedge R y x\}$

by $(\text{auto elim: quotientE})$

let $?R = \lambda x y. R x y \wedge R y x$

let $?pR = \lambda y. \text{measure } q \{x. ?R x y\}$

have $\text{measure } p \{y. ?R x y\} = \text{measure } p (\{y. R x y\} - \{y. R x y \wedge \neg R y x\})$

by $(\text{auto intro!: arg-cong}[\text{where } f = \text{measure } p])$

also have $\dots = \text{measure } p \{y. R x y\} - \text{measure } p \{y. R x y \wedge \neg R y x\}$

by $(\text{rule measure-pmf.finite-measure-Diff}) \text{ auto}$

```

also have measure p {y. R x y  $\wedge$   $\neg$  R y x} = measure q {y. R x y  $\wedge$   $\neg$  R y x}
  using 1 2 refl trans by(auto intro!: Orderings.antisym measure-Ioi)
also have measure p {y. R x y} = measure q {y. R x y}
  using 1 2 refl trans by(auto intro!: Orderings.antisym measure-Ici)
also have measure q {y. R x y} - measure q {y. R x y  $\wedge$   $\neg$  R y x} =
  measure q ({y. R x y} - {y. R x y  $\wedge$   $\neg$  R y x})
  by(rule measure-pmf.finite-measure-Diff[symmetric]) auto
also have ... = ? $\mu$ R x
  by(auto intro!: arg-cong[where f=measure q])
finally show measure p C = measure q C
  by (simp add: C conj-commute)
qed

```

```

lemma rel-pmf-antisym:
  fixes p q :: 'a pmf
  assumes 1: rel-pmf R p q
  assumes 2: rel-pmf R q p
  and refl: reflp R and trans: transp R and antisym: antisymp R
  shows p = q
proof -
  from 1 2 refl trans have rel-pmf (inf R R-1-1) p q by(rule rel-pmf-inf)
  also have inf R R-1-1 = op =
    using refl antisym by (auto intro!: ext simp add: reflpD dest: antisympD)
  finally show ?thesis unfolding pmf.rel-eq .
qed

```

```

lemma reflp-rel-pmf: reflp R  $\implies$  reflp (rel-pmf R)
  by (fact pmf.rel-reflp)

```

```

lemma antisymp-rel-pmf:
  [ reflp R; transp R; antisymp R ]
   $\implies$  antisymp (rel-pmf R)
by(rule antisympI)(blast intro: rel-pmf-antisym)

```

```

lemma transp-rel-pmf:
  assumes transp R
  shows transp (rel-pmf R)
  using assms by (fact pmf.rel-transp)

```

22.6 Distributions

```

context
begin

```

```

interpretation pmf-as-function .

```

22.6.1 Bernoulli Distribution

```

lift-definition bernoulli-pmf :: real  $\implies$  bool pmf is
   $\lambda p b. ((\lambda p. \text{if } b \text{ then } p \text{ else } 1 - p) \circ \text{min } 1 \circ \text{max } 0) p$ 

```

by (*auto simp: nn-integral-count-space-finite* [**where** $A = \{False, True\}$] *UNIV-bool split: split-max split-min*)

lemma *pmf-bernoulli-True*[*simp*]: $0 \leq p \implies p \leq 1 \implies \text{pmf } (\text{bernoulli-pmf } p) \text{ True} = p$
by *transfer simp*

lemma *pmf-bernoulli-False*[*simp*]: $0 \leq p \implies p \leq 1 \implies \text{pmf } (\text{bernoulli-pmf } p) \text{ False} = 1 - p$
by *transfer simp*

lemma *set-pmf-bernoulli*[*simp*]: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{bernoulli-pmf } p) = \text{UNIV}$
by (*auto simp add: set-pmf-iff UNIV-bool*)

lemma *nn-integral-bernoulli-pmf*[*simp*]:
assumes [*simp*]: $0 \leq p \ p \leq 1 \ \wedge x. 0 \leq f x$
shows $(\int^+ x. f x \ \partial \text{bernoulli-pmf } p) = f \text{ True} * p + f \text{ False} * (1 - p)$
by (*subst nn-integral-measure-pmf-support* [*of UNIV*])
(auto simp: UNIV-bool field-simps)

lemma *integral-bernoulli-pmf*[*simp*]:
assumes [*simp*]: $0 \leq p \ p \leq 1$
shows $(\int x. f x \ \partial \text{bernoulli-pmf } p) = f \text{ True} * p + f \text{ False} * (1 - p)$
by (*subst integral-measure-pmf* [*of UNIV*]) *(auto simp: UNIV-bool)*

lemma *pmf-bernoulli-half* [*simp*]: $\text{pmf } (\text{bernoulli-pmf } (1 / 2)) x = 1 / 2$
by (*cases x*) *simp-all*

lemma *measure-pmf-bernoulli-half*: $\text{measure-pmf } (\text{bernoulli-pmf } (1 / 2)) = \text{uniform-count-measure UNIV}$
by (*rule measure-eqI*)
(simp-all add: nn-integral-pmf[symmetric] emeasure-uniform-count-measure ennreal-divide-numeral[symmetric] nn-integral-count-space-finite sets-uniform-count-measure divide-ennreal-def mult-ac ennreal-of-nat-eq-real-of-nat)

22.6.2 Geometric Distribution

context

fixes $p :: \text{real}$ **assumes** $p[\text{arith}]: 0 < p \ p \leq 1$

begin

lift-definition *geometric-pmf* $:: \text{nat pmf is } \lambda n. (1 - p) ^ n * p$

proof

have $(\sum i. \text{ennreal } (p * (1 - p) ^ i)) = \text{ennreal } (p * (1 / (1 - (1 - p))))$

by (*intro suminf-ennreal-eq sums-mult geometric-sums*) *auto*

then show $(\int^+ x. \text{ennreal } ((1 - p) ^ x * p) \ \partial \text{count-space UNIV}) = 1$

by (*simp add: nn-integral-count-space-nat field-simps*)
qed simp

lemma pmf-geometric[*simp*]: $\text{pmf } \text{geometric-pmf } n = (1 - p)^n * p$
 by *transfer rule*

end

lemma set-pmf-geometric: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{geometric-pmf } p) =$
 UNIV
 by (*auto simp: set-pmf-iff*)

22.6.3 Uniform Multiset Distribution

context

fixes $M :: 'a \text{ multiset}$ **assumes** *M-not-empty*: $M \neq \{\#\}$

begin

lift-definition pmf-of-multiset :: $'a \text{ pmf}$ **is** $\lambda x. \text{count } M x / \text{size } M$

proof

show $(\int^+ x. \text{ennreal } (\text{real } (\text{count } M x) / \text{real } (\text{size } M))) \partial \text{count-space UNIV} =$
 1

using *M-not-empty*

by (*simp add: zero-less-divide-iff nn-integral-count-space nonempty-has-size*
sum-divide-distrib[symmetric])

(*auto simp: size-multiset-overloaded-eq intro!: sum.cong*)

qed simp

lemma pmf-of-multiset[*simp*]: $\text{pmf } \text{pmf-of-multiset } x = \text{count } M x / \text{size } M$
 by *transfer rule*

lemma set-pmf-of-multiset[*simp*]: $\text{set-pmf } \text{pmf-of-multiset} = \text{set-mset } M$
 by (*auto simp: set-pmf-iff*)

end

22.6.4 Uniform Distribution

context

fixes $S :: 'a \text{ set}$ **assumes** *S-not-empty*: $S \neq \{\}$ **and** *S-finite*: *finite S*

begin

lift-definition pmf-of-set :: $'a \text{ pmf}$ **is** $\lambda x. \text{indicator } S x / \text{card } S$

proof

show $(\int^+ x. \text{ennreal } (\text{indicator } S x / \text{real } (\text{card } S))) \partial \text{count-space UNIV} = 1$

using *S-not-empty S-finite*

by (*subst nn-integral-count-space'[of S]*)

(*auto simp: ennreal-of-nat-eq-real-of-nat ennreal-mult[symmetric]*)

qed simp

lemma *pmf-of-set*[simp]: $\text{pmf pmf-of-set } x = \text{indicator } S \ x / \text{card } S$
by *transfer rule*

lemma *set-pmf-of-set*[simp]: $\text{set-pmf pmf-of-set} = S$
using *S-finite S-not-empty* **by** (*auto simp: set-pmf-iff*)

lemma *emeasure-pmf-of-set-space*[simp]: $\text{emeasure pmf-of-set } S = 1$
by (*rule measure-pmf.emeasure-eq-1-AE*) (*auto simp: AE-measure-pmf-iff*)

lemma *nn-integral-pmf-of-set*: $\text{nn-integral } (\text{measure-pmf pmf-of-set}) \ f = \text{sum } f \ S / \text{card } S$
by (*subst nn-integral-measure-pmf-finite*)
(simp-all add: sum-distrib-right[symmetric] card-gt-0-iff S-not-empty S-finite divide-ennreal-def divide-ennreal[symmetric] ennreal-of-nat-eq-real-of-nat[symmetric] ennreal-times-divide)

lemma *integral-pmf-of-set*: $\text{integral}^L (\text{measure-pmf pmf-of-set}) \ f = \text{sum } f \ S / \text{card } S$
by (*subst integral-measure-pmf[of S]*) (*auto simp: S-finite sum-divide-distrib*)

lemma *emeasure-pmf-of-set*: $\text{emeasure } (\text{measure-pmf pmf-of-set}) \ A = \text{card } (S \cap A) / \text{card } S$
by (*subst nn-integral-indicator[symmetric], simp*)
(simp add: S-finite S-not-empty card-gt-0-iff indicator-def sum.If-cases divide-ennreal ennreal-of-nat-eq-real-of-nat nn-integral-pmf-of-set)

lemma *measure-pmf-of-set*: $\text{measure } (\text{measure-pmf pmf-of-set}) \ A = \text{card } (S \cap A) / \text{card } S$
using *emeasure-pmf-of-set[of A]*
by (*simp add: measure-nonneg measure-pmf.emeasure-eq-measure*)

end

lemma *pmf-expectation-bind-pmf-of-set*:
fixes $A :: 'a \text{ set}$ **and** $f :: 'a \Rightarrow 'b \text{ pmf}$
and $h :: 'b \Rightarrow 'c :: \{\text{banach, second-countable-topology}\}$
assumes $A \neq \{\}$ *finite* $A \wedge x. x \in A \implies \text{finite } (\text{set-pmf } (f \ x))$
shows $\text{measure-pmf.expectation } (\text{pmf-of-set } A \gg f) \ h =$
 $(\sum a \in A. \text{measure-pmf.expectation } (f \ a) \ h /_{\mathbb{R}} \text{real } (\text{card } A))$
using *assms* **by** (*subst pmf-expectation-bind[of A]*) (*auto simp: divide-simps*)

lemma *map-pmf-of-set*:
assumes *finite* $A \ A \neq \{\}$
shows $\text{map-pmf } f \ (\text{pmf-of-set } A) = \text{pmf-of-multiset } (\text{image-mset } f \ (\text{mset-set } A))$
(is ?lhs = ?rhs)

proof (*intro pmf-eqI*)
fix x

from *assms* **have** $\text{ennreal } (\text{pmf } ?\text{lhs } x) = \text{ennreal } (\text{pmf } ?\text{rhs } x)$
by (*subst ennreal-pmf-map*)
(simp-all add: emeasure-pmf-of-set mset-set-empty-iff count-image-mset
Int-commute)
thus $\text{pmf } ?\text{lhs } x = \text{pmf } ?\text{rhs } x$ **by** *simp*
qed

lemma *pmf-bind-pmf-of-set*:
assumes $A \neq \{\}$ *finite A*
shows $\text{pmf } (\text{bind-pmf } (\text{pmf-of-set } A) f) x =$
 $(\sum_{xa \in A. \text{pmf } (f \text{ } xa) x} / \text{real-of-nat } (\text{card } A))$ (**is** $?lhs = ?rhs$)
proof –
from *assms* **have** $\text{card } A > 0$ **by** *auto*
with *assms* **have** $\text{ennreal } ?lhs = \text{ennreal } ?rhs$
by (*subst ennreal-pmf-bind*)
(simp-all add: nn-integral-pmf-of-set max-def pmf-nonneg divide-ennreal
[symmetric]
sum-nonneg ennreal-of-nat-eq-real-of-nat)
thus $?thesis$ **by** (*subst (asm) ennreal-inj*) (*auto intro!: sum-nonneg divide-nonneg-nonneg*)
qed

lemma *pmf-of-set-singleton*: $\text{pmf-of-set } \{x\} = \text{return-pmf } x$
by(*rule pmf-eqI*)(*simp add: indicator-def*)

lemma *map-pmf-of-set-inj*:
assumes $f: \text{inj-on } f \ A$
and [*simp*]: $A \neq \{\}$ *finite A*
shows $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } (f \text{ ' } A)$ (**is** $?lhs = ?rhs$)
proof(*rule pmf-eqI*)
fix i
show $\text{pmf } ?lhs \ i = \text{pmf } ?rhs \ i$
proof(*cases i ∈ f ' A*)
case *True*
then obtain i' **where** $i = f \ i'$ $i' \in A$ **by** *auto*
thus $?thesis$ **using** f **by**(*simp add: card-image pmf-map-inj*)
next
case *False*
hence $\text{pmf } ?lhs \ i = 0$ **by**(*simp add: pmf-eq-0-set-pmf set-map-pmf*)
moreover **have** $\text{pmf } ?rhs \ i = 0$ **using** *False* **by** *simp*
ultimately show $?thesis$ **by** *simp*
qed
qed

lemma *map-pmf-of-set-bij-betw*:
assumes *bij-betw f A B A ≠ {}* *finite A*
shows $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } B$
proof –
have $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } (f \text{ ' } A)$
by (*intro map-pmf-of-set-inj assms bij-betw-imp-inj-on[OF assms(1)]*)

also from *assms* **have** $f \text{ ‘ } A = B \text{ by (simp add: bij-betw-def)}$
finally show *?thesis* .
qed

Choosing an element uniformly at random from the union of a disjoint family of finite non-empty sets with the same size is the same as first choosing a set from the family uniformly at random and then choosing an element from the chosen set uniformly at random.

lemma *pmf-of-set-UN*:

assumes *finite* $(\text{UNION } A \text{ } f) \ A \neq \{\}$ $\bigwedge x. x \in A \implies f \ x \neq \{\}$
 $\bigwedge x. x \in A \implies \text{card } (f \ x) = n$ *disjoint-family-on* $f \ A$
shows $\text{pmf-of-set } (\text{UNION } A \text{ } f) = \text{do } \{x \leftarrow \text{pmf-of-set } A; \text{pmf-of-set } (f \ x)\}$
(is ?lhs = ?rhs)

proof *(intro pmf-eqI)*

fix x

from *assms* **have** $[\text{simp}]: \text{finite } A$

using *infinite-disjoint-family-imp-infinite-UNION* $[\text{of } A \text{ } f]$ **by** *blast*

from *assms* **have** $\text{ereal } (\text{pmf } (\text{pmf-of-set } (\text{UNION } A \text{ } f)) \ x) =$
 $\text{ereal } (\text{indicator } (\bigcup x \in A. f \ x) \ x / \text{real } (\text{card } (\bigcup x \in A. f \ x)))$

by *(subst pmf-of-set) auto*

also from *assms* **have** $\text{card } (\bigcup x \in A. f \ x) = \text{card } A * n$

by *(subst card-UN-disjoint) (auto simp: disjoint-family-on-def)*

also from *assms*

have $\text{indicator } (\bigcup x \in A. f \ x) \ x / \text{real } \dots =$

$\text{indicator } (\bigcup x \in A. f \ x) \ x / (n * \text{real } (\text{card } A))$

by *(simp add: sum-divide-distrib [symmetric] mult-ac)*

also from *assms* **have** $\text{indicator } (\bigcup x \in A. f \ x) \ x = (\sum y \in A. \text{indicator } (f \ y) \ x)$

by *(intro indicator-UN-disjoint) simp-all*

also from *assms* **have** $\text{ereal } ((\sum y \in A. \text{indicator } (f \ y) \ x) / (\text{real } n * \text{real } (\text{card } A))) =$

$\text{ereal } (\text{pmf } ?\text{rhs } x)$

by *(subst pmf-bind-pmf-of-set) (simp-all add: sum-divide-distrib)*

finally show $\text{pmf } ?\text{lhs } x = \text{pmf } ?\text{rhs } x$ **by** *simp*

qed

lemma *bernoulli-pmf-half-conv-pmf-of-set*: $\text{bernoulli-pmf } (1 / 2) = \text{pmf-of-set } \text{UNIV}$

by *(rule pmf-eqI) simp-all*

22.6.5 Poisson Distribution

context

fixes $\text{rate} :: \text{real}$ **assumes** *rate-pos*: $0 < \text{rate}$

begin

lift-definition *poisson-pmf* $:: \text{nat pmf}$ **is** $\lambda k. \text{rate} ^ k / \text{fact } k * \exp (-\text{rate})$

proof

have *summable*: $\text{summable } (\lambda x :: \text{nat}. \text{rate} ^ x / \text{fact } x)$ **using** *summable-exp*

by *(simp add: field-simps divide-inverse [symmetric])*

have $(\int ^+(x :: \text{nat}). \text{rate} ^ x / \text{fact } x * \exp (-\text{rate}) \ \partial \text{count-space } \text{UNIV}) =$

$\text{exp } (-\text{rate}) * (\int^+(x::\text{nat}). \text{rate} \wedge x / \text{fact } x \partial\text{count-space UNIV})$
by (*simp add: field-simps nn-integral-cmult[symmetric] ennreal-mult'[symmetric]*)
also from *rate-pos* **have** $(\int^+(x::\text{nat}). \text{rate} \wedge x / \text{fact } x \partial\text{count-space UNIV}) =$
 $(\sum x. \text{rate} \wedge x / \text{fact } x)$
by (*simp-all add: nn-integral-count-space-nat suminf-ennreal summable ennreal-suminf-neq-top*)
also have $\dots = \text{exp rate}$ **unfolding** *exp-def*
by (*simp add: field-simps divide-inverse [symmetric]*)
also have $\text{ennreal } (\text{exp } (-\text{rate})) * \text{ennreal } (\text{exp rate}) = 1$
by (*simp add: mult-exp-exp ennreal-mult[symmetric]*)
finally show $(\int^+ x. \text{ennreal } (\text{rate} \wedge x / (\text{fact } x) * \text{exp } (-\text{rate})) \partial\text{count-space UNIV}) = 1$.
qed (*simp add: rate-pos[THEN less-imp-le]*)

lemma *pmf-poisson[simp]*: $\text{pmf poisson-pmf } k = \text{rate} \wedge k / \text{fact } k * \text{exp } (-\text{rate})$
by *transfer rule*

lemma *set-pmf-poisson[simp]*: $\text{set-pmf poisson-pmf} = \text{UNIV}$
using *rate-pos* **by** (*auto simp: set-pmf-iff*)

end

22.6.6 Binomial Distribution

context

fixes $n :: \text{nat}$ **and** $p :: \text{real}$ **assumes** *p-nonneg*: $0 \leq p$ **and** *p-le-1*: $p \leq 1$

begin

lift-definition *binomial-pmf* $:: \text{nat pmf}$ **is** $\lambda k. (n \text{ choose } k) * p \wedge k * (1 - p) \wedge (n - k)$

proof

have $(\int^+ k. \text{ennreal } (\text{real } (n \text{ choose } k) * p \wedge k * (1 - p) \wedge (n - k)) \partial\text{count-space UNIV}) =$

$\text{ennreal } (\sum k \leq n. \text{real } (n \text{ choose } k) * p \wedge k * (1 - p) \wedge (n - k))$

using *p-le-1 p-nonneg* **by** (*subst nn-integral-count-space'*) *auto*

also have $(\sum k \leq n. \text{real } (n \text{ choose } k) * p \wedge k * (1 - p) \wedge (n - k)) = (p + (1 - p)) \wedge n$

by (*subst binomial-ring*) (*simp add: atLeast0AtMost*)

finally show $(\int^+ x. \text{ennreal } (\text{real } (n \text{ choose } x) * p \wedge x * (1 - p) \wedge (n - x)) \partial\text{count-space UNIV}) = 1$

by *simp*

qed (*insert p-nonneg p-le-1, simp*)

lemma *pmf-binomial[simp]*: $\text{pmf binomial-pmf } k = (n \text{ choose } k) * p \wedge k * (1 - p) \wedge (n - k)$

by *transfer rule*

lemma *set-pmf-binomial-eq*: $\text{set-pmf binomial-pmf} = (\text{if } p = 0 \text{ then } \{0\} \text{ else if } p = 1 \text{ then } \{n\} \text{ else } \{.. n\})$

using *p-nonneg p-le-1* **unfolding** *set-eq-iff set-pmf-iff pmf-binomial* **by** (*auto*)

simp: *set-pmf-iff*)

end

end

lemma *set-pmf-binomial-0*[*simp*]: *set-pmf* (*binomial-pmf* *n* 0) = {0}
by (*simp add: set-pmf-binomial-eq*)

lemma *set-pmf-binomial-1*[*simp*]: *set-pmf* (*binomial-pmf* *n* 1) = {*n*}
by (*simp add: set-pmf-binomial-eq*)

lemma *set-pmf-binomial*[*simp*]: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{binomial-pmf } n \ p) = \{..n\}$
by (*simp add: set-pmf-binomial-eq*)

context includes *lifting-syntax*

begin

lemma *bind-pmf-parametric* [*transfer-rule*]:
 $(\text{rel-pmf } A \implies (A \implies \text{rel-pmf } B) \implies \text{rel-pmf } B) \text{ bind-pmf bind-pmf}$
by(*blast intro: rel-pmf-bindI dest: rel-funD*)

lemma *return-pmf-parametric* [*transfer-rule*]: $(A \implies \text{rel-pmf } A) \text{ return-pmf}$
 return-pmf
by(*rule rel-funI*) *simp*

end

primrec *replicate-pmf* :: $\text{nat} \Rightarrow 'a \text{ pmf} \Rightarrow 'a \text{ list pmf}$ **where**
 $\text{replicate-pmf } 0 = \text{return-pmf } []$
 $| \text{replicate-pmf } (\text{Suc } n) \ p = \text{do } \{x \leftarrow p; xs \leftarrow \text{replicate-pmf } n \ p; \text{return-pmf } (x \# xs)\}$

lemma *replicate-pmf-1*: $\text{replicate-pmf } 1 \ p = \text{map-pmf } (\lambda x. [x]) \ p$
by (*simp add: map-pmf-def bind-return-pmf*)

lemma *set-replicate-pmf*:
 $\text{set-pmf } (\text{replicate-pmf } n \ p) = \{xs \in \text{lists } (\text{set-pmf } p). \text{length } xs = n\}$
by (*induction n*) (*auto simp: length-Suc-conv*)

lemma *replicate-pmf-distrib*:
 $\text{replicate-pmf } (m + n) \ p =$
 $\text{do } \{xs \leftarrow \text{replicate-pmf } m \ p; ys \leftarrow \text{replicate-pmf } n \ p; \text{return-pmf } (xs @ ys)\}$
by (*induction m*) (*simp-all add: bind-return-pmf bind-return-pmf' bind-assoc-pmf*)

lemma *power-diff'*:
assumes $b \leq a$

shows $x \wedge (a - b) = (\text{if } x = 0 \wedge a = b \text{ then } 1 \text{ else } x \wedge a / (x::'a::\text{field}) \wedge b)$
proof (cases $x = 0$)
case *True*
with *assms* **show** *?thesis* **by** (cases $a - b$) *simp-all*
qed (insert *assms*, *simp-all* add: *power-diff*)

lemma *binomial-pmf-Suc*:

assumes $p \in \{0..1\}$
shows $\text{binomial-pmf } (\text{Suc } n) p =$
 $\text{do } \{b \leftarrow \text{bernoulli-pmf } p;$
 $k \leftarrow \text{binomial-pmf } n p;$
 $\text{return-pmf } ((\text{if } b \text{ then } 1 \text{ else } 0) + k)\} (\text{is } - = ?\text{rhs})$
proof (intro *pmf-eqI*)
fix k
have $A: \text{indicator } \{\text{Suc } a\} (\text{Suc } b) = \text{indicator } \{a\} b$ **for** $a b$
by (*simp* add: *indicator-def*)
show $\text{pmf } (\text{binomial-pmf } (\text{Suc } n) p) k = \text{pmf } ?\text{rhs } k$
by (cases k ; cases $k > n$)
(insert *assms*, auto *simp*: *pmf-bind* *measure-pmf-single* *A* *divide-simps*
algebra-simps
not-less *less-eq-Suc-le* [*symmetric*] *power-diff'*)
qed

lemma *binomial-pmf-0*: $p \in \{0..1\} \implies \text{binomial-pmf } 0 p = \text{return-pmf } 0$
by (*rule* *pmf-eqI*) (*simp-all* add: *indicator-def*)

lemma *binomial-pmf-altdef*:

assumes $p \in \{0..1\}$
shows $\text{binomial-pmf } n p = \text{map-pmf } (\text{length} \circ \text{filter } \text{id}) (\text{replicate-pmf } n$
 $(\text{bernoulli-pmf } p))$
by (*induction* n)
(insert *assms*, auto *simp*: *binomial-pmf-Suc* *map-pmf-def* *bind-return-pmf*
bind-assoc-pmf
bind-return-pmf' *binomial-pmf-0* *intro!*: *bind-pmf-cong*)

22.7 PMFs from association lists

definition *pmf-of-list* :: $('a \times \text{real}) \text{ list} \Rightarrow 'a \text{ pmf}$ **where**
 $\text{pmf-of-list } xs = \text{embed-pmf } (\lambda x. \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) xs)))$

definition *pmf-of-list-wf* **where**

$\text{pmf-of-list-wf } xs \longleftrightarrow (\forall x \in \text{set } (\text{map } \text{snd } xs) . x \geq 0) \wedge \text{sum-list } (\text{map } \text{snd } xs) = 1$

lemma *pmf-of-list-wfI*:

$(\bigwedge x. x \in \text{set } (\text{map } \text{snd } xs) \implies x \geq 0) \implies \text{sum-list } (\text{map } \text{snd } xs) = 1 \implies$
pmf-of-list-wf xs
unfolding *pmf-of-list-wf-def* **by** *simp*

context

begin

private lemma *pmf-of-list-aux*:

assumes $\bigwedge x. x \in \text{set } (\text{map } \text{snd } xs) \implies x \geq 0$

assumes $\text{sum-list } (\text{map } \text{snd } xs) = 1$

shows $(\int^+ x. \text{ennreal } (\text{sum-list } (\text{map } \text{snd } [z \leftarrow xs . \text{fst } z = x]))) \partial \text{count-space UNIV} = 1$

proof –

have $(\int^+ x. \text{ennreal } (\text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) xs)))) \partial \text{count-space UNIV} =$

$(\int^+ x. \text{ennreal } (\text{sum-list } (\text{map } (\lambda(x',p). \text{indicator } \{x'\} x * p) xs))) \partial \text{count-space UNIV}$

by (*intro nn-integral-cong ennreal-cong, subst sum-list-map-filter'*) (*auto intro: sum-list-cong*)

also have $\dots = (\sum (x',p) \leftarrow xs. (\int^+ x. \text{ennreal } (\text{indicator } \{x'\} x * p) \partial \text{count-space UNIV}))$

using *assms(1)*

proof (*induction xs*)

case (*Cons x xs*)

from *Cons.prem*s **have** $\text{snd } x \geq 0$ **by** *simp*

moreover have $b \geq 0$ **if** $(a,b) \in \text{set } xs$ **for** $a b$

using *Cons.prem*s[*of b*] **that** **by** *force*

ultimately have $(\int^+ y. \text{ennreal } (\sum (x', p) \leftarrow x \# xs. \text{indicator } \{x'\} y * p)) \partial \text{count-space UNIV} =$

$(\int^+ y. \text{ennreal } (\text{indicator } \{\text{fst } x\} y * \text{snd } x) +$

$\text{ennreal } (\sum (x', p) \leftarrow xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV})$

by (*intro nn-integral-cong, subst ennreal-plus [symmetric]*)

(*auto simp: case-prod-unfold indicator-def intro!: sum-list-nonneg*)

also have $\dots = (\int^+ y. \text{ennreal } (\text{indicator } \{\text{fst } x\} y * \text{snd } x) \partial \text{count-space UNIV}) +$

$(\int^+ y. \text{ennreal } (\sum (x', p) \leftarrow xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV})$

by (*intro nn-integral-add*)

(*force intro!: sum-list-nonneg AE-I2 intro: Cons simp: indicator-def*)+

also have $(\int^+ y. \text{ennreal } (\sum (x', p) \leftarrow xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV}) =$

$(\sum (x', p) \leftarrow xs. (\int^+ y. \text{ennreal } (\text{indicator } \{x'\} y * p) \partial \text{count-space UNIV}))$

using *Cons(1)* **by** (*intro Cons simp-all*)

finally show *?case* **by** (*simp add: case-prod-unfold*)

qed *simp*

also have $\dots = (\sum (x',p) \leftarrow xs. \text{ennreal } p * (\int^+ x. \text{indicator } \{x'\} x \partial \text{count-space UNIV}))$

using *assms(1)*

by (*intro sum-list-cong, simp only: case-prod-unfold, subst nn-integral-cmult [symmetric]*)

(*auto intro!: assms(1) simp: max-def times-ereal.simps [symmetric] mult-ac*)

ereal-indicator

simp del: times-ereal.simps)+
also from *assms* **have** $\dots = \text{sum-list } (\text{map } \text{snd } xs)$ **by** (*simp add: case-prod-unfold sum-list-ennreal*)
also have $\dots = 1$ **using** *assms*(2) **by** *simp*
finally show *?thesis* .
qed

lemma *pmf-pmf-of-list*:

assumes *pmf-of-list-wf xs*
shows $\text{pmf } (\text{pmf-of-list } xs) x = \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) xs))$
using *assms pmf-of-list-aux*[of *xs*] **unfolding** *pmf-of-list-def pmf-of-list-wf-def*
by (*subst pmf-embed-pmf*) (*auto intro!: sum-list-nonneg*)

end

lemma *set-pmf-of-list*:

assumes *pmf-of-list-wf xs*
shows $\text{set-pmf } (\text{pmf-of-list } xs) \subseteq \text{set } (\text{map } \text{fst } xs)$
proof *clarify*
fix *x* **assume** *A: x ∈ set-pmf (pmf-of-list xs)*
show $x \in \text{set } (\text{map } \text{fst } xs)$
proof (*rule ccontr*)
assume $x \notin \text{set } (\text{map } \text{fst } xs)$
hence $[z \leftarrow xs . \text{fst } z = x] = []$ **by** (*auto simp: filter-empty-conv*)
with *A* **assms** **show** *False* **by** (*simp add: pmf-pmf-of-list set-pmf-eq*)
qed
qed

lemma *finite-set-pmf-of-list*:

assumes *pmf-of-list-wf xs*
shows $\text{finite } (\text{set-pmf } (\text{pmf-of-list } xs))$
using *assms* **by** (*rule finite-subset[OF set-pmf-of-list]*) *simp-all*

lemma *emeasure-Int-set-pmf*:

$\text{emeasure } (\text{measure-pmf } p) (A \cap \text{set-pmf } p) = \text{emeasure } (\text{measure-pmf } p) A$
by (*rule emeasure-eq-AE*) (*auto simp: AE-measure-pmf-iff*)

lemma *measure-Int-set-pmf*:

$\text{measure } (\text{measure-pmf } p) (A \cap \text{set-pmf } p) = \text{measure } (\text{measure-pmf } p) A$
using *emeasure-Int-set-pmf*[of *p A*] **by** (*simp add: Sigma-Algebra.measure-def*)

lemma *measure-prob-cong-0*:

assumes $\bigwedge x. x \in A - B \implies \text{pmf } p x = 0$
assumes $\bigwedge x. x \in B - A \implies \text{pmf } p x = 0$
shows $\text{measure } (\text{measure-pmf } p) A = \text{measure } (\text{measure-pmf } p) B$
proof –
have $\text{measure-pmf.prob } p A = \text{measure-pmf.prob } p (A \cap \text{set-pmf } p)$
by (*simp add: measure-Int-set-pmf*)

also have $A \cap \text{set-pmf } p = B \cap \text{set-pmf } p$
using *assms* **by** (*auto simp: set-pmf-eq*)
also have $\text{measure-pmf.prob } p \dots = \text{measure-pmf.prob } p B$
by (*simp add: measure-Int-set-pmf*)
finally show *?thesis* .
qed

lemma *emeasure-pmf-of-list*:

assumes *pmf-of-list-wf xs*
shows $\text{emeasure } (\text{pmf-of-list } xs) A = \text{ennreal } (\text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda x. \text{fst } x \in A) xs)))$
proof –
have $\text{emeasure } (\text{pmf-of-list } xs) A = \text{nn-integral } (\text{measure-pmf } (\text{pmf-of-list } xs))$
(indicator A)
by *simp*
also from *assms*
have $\dots = (\sum_{x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{ennreal } (\text{sum-list } (\text{map } \text{snd } [z \leftarrow xs . \text{fst } z = x])))$
by (*subst nn-integral-measure-pmf-finite*) (*simp-all add: finite-set-pmf-of-list pmf-pmf-of-list Int-def*)
also from *assms*
have $\dots = \text{ennreal } (\sum_{x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{sum-list } (\text{map } \text{snd } [z \leftarrow xs . \text{fst } z = x]))$
by (*subst sum-ennreal*) (*auto simp: pmf-of-list-wf-def intro!: sum-list-nonneg*)
also have $\dots = \text{ennreal } (\sum_{x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{indicator } A x * \text{pmf } (\text{pmf-of-list } xs) x)$ (*is - = ennreal ?S*)
using *assms* **by** (*intro ennreal-cong sum.cong*) (*auto simp: pmf-pmf-of-list*)
also have $?S = (\sum_{x \in \text{set-pmf } (\text{pmf-of-list } xs). \text{indicator } A x * \text{pmf } (\text{pmf-of-list } xs) x)$
using *assms* **by** (*intro sum.mono-neutral-left set-pmf-of-list finite-set-pmf-of-list*)
auto
also have $\dots = (\sum_{x \in \text{set } (\text{map } \text{fst } xs). \text{indicator } A x * \text{pmf } (\text{pmf-of-list } xs) x)$
using *assms* **by** (*intro sum.mono-neutral-left set-pmf-of-list*) (*auto simp: set-pmf-eq*)
also have $\dots = (\sum_{x \in \text{set } (\text{map } \text{fst } xs). \text{indicator } A x * \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) xs)))$
using *assms* **by** (*simp add: pmf-pmf-of-list*)
also have $\dots = (\sum_{x \in \text{set } (\text{map } \text{fst } xs). \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x \wedge x \in A) xs)))$
by (*intro sum.cong*) (*auto simp: indicator-def*)
also have $\dots = (\sum_{x \in \text{set } (\text{map } \text{fst } xs). (\sum_{xa = 0..<\text{length } xs. \text{if } \text{fst } (xs ! xa) = x \wedge x \in A \text{ then } \text{snd } (xs ! xa) \text{ else } 0))$
by (*intro sum.cong refl, subst sum-list-map-filter', subst sum-list-sum-nth*) *simp*
also have $\dots = (\sum_{xa = 0..<\text{length } xs. (\sum_{x \in \text{set } (\text{map } \text{fst } xs). \text{if } \text{fst } (xs ! xa) = x \wedge x \in A \text{ then } \text{snd } (xs ! xa) \text{ else } 0))$
by (*rule sum commute*)
also have $\dots = (\sum_{xa = 0..<\text{length } xs. \text{if } \text{fst } (xs ! xa) \in A \text{ then } (\sum_{x \in \text{set } (\text{map } \text{fst } xs). \text{if } x = \text{fst } (xs ! xa) \text{ then } \text{snd } (xs ! xa) \text{ else } 0) \text{ else } 0)$
by (*auto intro!: sum.cong sum.neutral simp del: sum.delta*)

also have $\dots = (\sum xa = 0..<length\ xs.\ if\ fst\ (xs\ !\ xa) \in A\ then\ snd\ (xs\ !\ xa)\ else\ 0)$
by (*intro sum.cong refl*) (*simp-all add: sum.delta*)
also have $\dots = sum-list\ (map\ snd\ (filter\ (\lambda x.\ fst\ x \in A)\ xs))$
by (*subst sum-list-map-filter'*, *subst sum-list-sum-nth*) *simp-all*
finally show *?thesis* .
qed

lemma *measure-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows $measure\ (pmf-of-list\ xs)\ A = sum-list\ (map\ snd\ (filter\ (\lambda x.\ fst\ x \in A)\ xs))$
using *assms unfolding pmf-of-list-wf-def Sigma-Algebra.measure-def*
by (*subst emeasure-pmf-of-list [OF assms]*, *subst enn2real-ennreal*) (*auto intro!: sum-list-nonneg*)

lemma *sum-list-nonneg-eq-zero-iff*:
fixes $xs :: 'a :: linordered-ab-group-add\ list$
shows $(\bigwedge x.\ x \in set\ xs \implies x \geq 0) \implies sum-list\ xs = 0 \longleftrightarrow set\ xs \subseteq \{0\}$
proof (*induction xs*)
case (*Cons x xs*)
from *Cons.prem*s **have** $sum-list\ (x\#\!xs) = 0 \longleftrightarrow x = 0 \wedge sum-list\ xs = 0$
unfolding *sum-list-simps* **by** (*subst add-nonneg-eq-0-iff*) (*auto intro: sum-list-nonneg*)
with *Cons.IH Cons.prem*s **show** *?case* **by** *simp*
qed *simp-all*

lemma *sum-list-filter-nonzero*:
 $sum-list\ (filter\ (\lambda x.\ x \neq 0)\ xs) = sum-list\ xs$
by (*induction xs*) *simp-all*

lemma *set-pmf-of-list-eq*:
assumes *pmf-of-list-wf xs* $\bigwedge x.\ x \in snd\ 'set\ xs \implies x > 0$
shows $set-pmf\ (pmf-of-list\ xs) = fst\ 'set\ xs$
proof
{
fix x **assume** $A: x \in fst\ 'set\ xs$ **and** $B: x \notin set-pmf\ (pmf-of-list\ xs)$
then obtain y **where** $y: (x, y) \in set\ xs$ **by** *auto*
from B **have** $sum-list\ (map\ snd\ [z\leftarrow xs.\ fst\ z = x]) = 0$
by (*simp add: pmf-pmf-of-list [OF assms(1)] set-pmf-eq*)
moreover from y **have** $y \in snd\ '\{xa \in set\ xs.\ fst\ xa = x\}$ **by** *force*
ultimately have $y = 0$ **using** *assms(1)*
by (*subst (asm) sum-list-nonneg-eq-zero-iff*) (*auto simp: pmf-of-list-wf-def*)
with *assms(2) y* **have** *False* **by** *force*
}
thus $fst\ 'set\ xs \subseteq set-pmf\ (pmf-of-list\ xs)$ **by** *blast*
qed (*insert set-pmf-of-list [OF assms(1)]*, *simp-all*)

```

lemma pmf-of-list-remove-zeros:
  assumes pmf-of-list-wf xs
  defines xs'  $\equiv$  filter ( $\lambda z. \text{snd } z \neq 0$ ) xs
  shows pmf-of-list-wf xs' pmf-of-list xs' = pmf-of-list xs
proof –
  have map snd [z $\leftarrow$ xs . snd z  $\neq$  0] = filter ( $\lambda x. x \neq 0$ ) (map snd xs)
    by (induction xs) simp-all
  with assms(1) show wf: pmf-of-list-wf xs'
    by (auto simp: pmf-of-list-wf-def xs'-def sum-list-filter-nonzero)
  have sum-list (map snd [z $\leftarrow$ xs' . fst z = i]) = sum-list (map snd [z $\leftarrow$ xs . fst z
= i]) for i
    unfolding xs'-def by (induction xs) simp-all
  with assms(1) wf show pmf-of-list xs' = pmf-of-list xs
    by (intro pmf-eqI) (simp-all add: pmf-pmf-of-list)
qed

end

```

23 Code generation for PMFs

```

theory PMF-Impl
imports Probability-Mass-Function HOL-Library.AList-Mapping
begin

```

23.1 General code generation setup

```

definition pmf-of-mapping :: ('a, real) mapping  $\Rightarrow$  'a pmf where
  pmf-of-mapping m = embed-pmf (Mapping.lookup-default 0 m)

```

```

lemma nn-integral-lookup-default:
  fixes m :: ('a, real) mapping
  assumes finite (Mapping.keys m) All-mapping m ( $\lambda x. x \geq 0$ )
  shows nn-integral (count-space UNIV) ( $\lambda k. \text{ennreal } (\text{Mapping.lookup-default } 0 \text{ } m \text{ } k)$ ) =
    ennreal ( $\sum k \in \text{Mapping.keys } m. \text{Mapping.lookup-default } 0 \text{ } m \text{ } k$ )
proof –
  have nn-integral (count-space UNIV) ( $\lambda k. \text{ennreal } (\text{Mapping.lookup-default } 0 \text{ } m \text{ } k)$ ) =
    ( $\sum x \in \text{Mapping.keys } m. \text{ennreal } (\text{Mapping.lookup-default } 0 \text{ } m \text{ } x)$ ) using
  assms
  by (subst nn-integral-count-space'[of Mapping.keys m])
    (auto simp: Mapping.lookup-default-def keys-is-none-rep Option.is-none-def)
  also from assms have ... = ennreal ( $\sum k \in \text{Mapping.keys } m. \text{Mapping.lookup-default } 0 \text{ } m \text{ } k$ )
  by (intro sum-ennreal)
    (auto simp: Mapping.lookup-default-def All-mapping-def split: option.splits)
  finally show ?thesis .
qed

```

lemma *pmf-of-mapping*:

assumes *finite* (*Mapping.keys m*) *All-mapping m* ($\lambda p. p \geq 0$)

assumes $(\sum_{x \in \text{Mapping.keys } m}. \text{Mapping.lookup-default } 0 \text{ } m \text{ } x) = 1$

shows $\text{pmf } (\text{pmf-of-mapping } m) \text{ } x = \text{Mapping.lookup-default } 0 \text{ } m \text{ } x$

unfolding *pmf-of-mapping-def*

proof (*intro pmf-embed-pmf*)

from *assms* **show** $(\int^+ x. \text{ennreal } (\text{Mapping.lookup-default } 0 \text{ } m \text{ } x) \text{ } \partial \text{count-space UNIV}) = 1$

by (*subst nn-integral-lookup-default*) (*simp-all*)

qed (*insert assms, simp add: All-mapping-def Mapping.lookup-default-def split: option.splits*)

lemma *pmf-of-set-pmf-of-mapping*:

assumes $A \neq \{\}$ *set xs = A distinct xs*

shows $\text{pmf-of-set } A = \text{pmf-of-mapping } (\text{Mapping.tabulate } xs \text{ } (\lambda x. 1 / \text{real } (\text{length } xs)))$

(**is** *?lhs = ?rhs*)

by (*rule pmf-eqI, subst pmf-of-mapping*)

(*insert assms, auto intro!: All-mapping-tabulate*

simp: Mapping.lookup-default-def lookup-tabulate distinct-card)

lift-definition *mapping-of-pmf* :: $'a \text{ pmf} \Rightarrow ('a, \text{real}) \text{ mapping}$ **is**

$\lambda p \text{ } x. \text{if } \text{pmf } p \text{ } x = 0 \text{ then None else Some } (\text{pmf } p \text{ } x) .$

lemma *lookup-default-mapping-of-pmf*:

$\text{Mapping.lookup-default } 0 \text{ } (\text{mapping-of-pmf } p) \text{ } x = \text{pmf } p \text{ } x$

by (*simp add: mapping-of-pmf.abs-eq lookup-default-def Mapping.lookup.abs-eq*)

context

begin

interpretation *pmf-as-function* .

lemma *nn-integral-pmf-eq-1*: $(\int^+ x. \text{ennreal } (\text{pmf } p \text{ } x) \text{ } \partial \text{count-space UNIV}) = 1$

by *transfer simp-all*

end

lemma *pmf-of-mapping-mapping-of-pmf* [*code abstype*]:

$\text{pmf-of-mapping } (\text{mapping-of-pmf } p) = p$

unfolding *pmf-of-mapping-def*

by (*rule pmf-eqI, subst pmf-embed-pmf*)

(*insert nn-integral-pmf-eq-1[of p],*

auto simp: lookup-default-mapping-of-pmf split: option.splits)

lemma *mapping-of-pmfI*:

assumes $\bigwedge x. x \in \text{Mapping.keys } m \implies \text{Mapping.lookup } m \text{ } x = \text{Some } (\text{pmf } p \text{ } x)$

assumes $\text{Mapping.keys } m = \text{set-pmf } p$

shows $\text{mapping-of-pmf } p = m$

using *assms* **by** *transfer (rule ext, auto simp: set-pmf-eq)*

lemma *mapping-of-pmfI'*:

assumes $\bigwedge x. x \in \text{Mapping.keys } m \implies \text{Mapping.lookup-default } 0 \ m \ x = \text{pmf } p$
 x
assumes $\text{Mapping.keys } m = \text{set-pmf } p$
shows $\text{mapping-of-pmf } p = m$
using *assms* **unfolding** $\text{Mapping.lookup-default-def}$
by *transfer (rule ext, force simp: set-pmf-eq)*

lemma *return-pmf-code* [*code abstract*]:

$\text{mapping-of-pmf } (\text{return-pmf } x) = \text{Mapping.update } x \ 1 \ \text{Mapping.empty}$
by (*intro mapping-of-pmfI*) (*auto simp: lookup-update'*)

lemma *pmf-of-set-code-aux*:

assumes $A \neq \{\}$ *set* $xs = A$ *distinct* xs
shows $\text{mapping-of-pmf } (\text{pmf-of-set } A) = \text{Mapping.tabulate } xs \ (\lambda-. \ 1 \ / \ \text{real}$
 $(\text{length } xs))$
using *assms*
by (*intro mapping-of-pmfI, subst pmf-of-set*)
(auto simp: lookup-tabulate distinct-card)

definition *pmf-of-set-impl* **where**

$\text{pmf-of-set-impl } A = \text{mapping-of-pmf } (\text{pmf-of-set } A)$

lemma *pmf-of-set-impl-code-alt*:

assumes $A \neq \{\}$ *finite* A
shows $\text{pmf-of-set-impl } A =$
 $(\text{let } p = 1 \ / \ \text{real } (\text{card } A)$
 $\text{in } \text{Finite-Set.fold } (\lambda x. \ \text{Mapping.update } x \ p) \ \text{Mapping.empty } A)$

proof –

define p **where** $p = 1 \ / \ \text{real } (\text{card } A)$
let $?m = \text{Finite-Set.fold } (\lambda x. \ \text{Mapping.update } x \ p) \ \text{Mapping.empty } A$
interpret *comp-fun-idem* $\lambda x. \ \text{Mapping.update } x \ p$
by *standard (transfer, force simp: fun-eq-iff)+*
have *keys*: $\text{Mapping.keys } ?m = A$
using *assms(2)* **by** (*induction A rule: finite-induct*) *simp-all*
have *lookup*: $\text{Mapping.lookup } ?m \ x = \text{Some } p$ **if** $x \in A$ **for** x
using *assms(2)* **that** **by** (*induction A rule: finite-induct*) (*auto simp: lookup-update'*)
from *keys lookup assms* **show** *?thesis* **unfolding** *pmf-of-set-impl-def*
by (*intro mapping-of-pmfI*) (*simp-all add: Let-def p-def*)

qed

lemma *pmf-of-set-impl-code* [*code*]:

$\text{pmf-of-set-impl } (\text{set } xs) =$
 $(\text{if } xs = [] \ \text{then}$
 $\text{Code.abort } (\text{STR } \text{"pmf-of-set of empty set"}) \ (\lambda-. \ \text{mapping-of-pmf } (\text{pmf-of-set}$
 $(\text{set } xs)))$
 $\text{else let } xs' = \text{remdups } xs; \ p = 1 \ / \ \text{real } (\text{length } xs') \ \text{in}$

Mapping.tabulate xs' (λ-. p)
unfolding *pmf-of-set-impl-def*
using *pmf-of-set-code-aux*[of set xs remdups xs] **by** (*simp add: Let-def*)

lemma *pmf-of-set-code* [code abstract]:
mapping-of-pmf (pmf-of-set A) = pmf-of-set-impl A
by (*simp add: pmf-of-set-impl-def*)

lemma *pmf-of-multiset-pmf-of-mapping*:
assumes $A \neq \{\#\}$ *set xs = set-mset A distinct xs*
shows *mapping-of-pmf (pmf-of-multiset A) = Mapping.tabulate xs (λx. count A x / real (size A))*
using *assms* **by** (*intro mapping-of-pmfI*) (*auto simp: lookup-tabulate*)

definition *pmf-of-multiset-impl* **where**
pmf-of-multiset-impl A = mapping-of-pmf (pmf-of-multiset A)

lemma *pmf-of-multiset-impl-code-alt*:
assumes $A \neq \{\#\}$
shows *pmf-of-multiset-impl A =*
(let p = 1 / real (size A)
in fold-mset (λx. Mapping.map-default x 0 (op + p)) Mapping.empty
A)

proof –

define *p* **where** $p = 1 / \text{real (size A)}$
interpret *comp-fun-commute* $\lambda x. \text{Mapping.map-default } x \ 0 \ (op + p)$
unfolding *Mapping.map-default-def* [abs-def]
by (*standard, intro mapping-eqI ext*)
(simp-all add: o-def lookup-map-entry' lookup-default' lookup-default-def)
let $?m = \text{fold-mset } (\lambda x. \text{Mapping.map-default } x \ 0 \ (op + p)) \ \text{Mapping.empty } A$
have *keys: Mapping.keys ?m = set-mset A* **by** (*induction A*) *simp-all*
have *lookup: Mapping.lookup-default 0 ?m x = real (count A x) * p* **for** *x*
by (*induction A*)
(simp-all add: lookup-map-default' lookup-default-def lookup-empty ring-distrib)
from *keys lookup assms* **show** *?thesis* **unfolding** *pmf-of-multiset-impl-def*
by (*intro mapping-of-pmfI'*) (*simp-all add: Let-def p-def*)

qed

lemma *pmf-of-multiset-impl-code* [code]:
pmf-of-multiset-impl (mset xs) =
(if xs = [] then
Code.abort (STR "pmf-of-multiset of empty multiset")
(λ-. mapping-of-pmf (pmf-of-multiset (mset xs)))
else let xs' = remdups xs; p = 1 / real (length xs) in
*Mapping.tabulate xs' (λx. real (count (mset xs) x) * p)*
using *pmf-of-multiset-pmf-of-mapping*[of mset xs remdups xs]
by (*simp add: pmf-of-multiset-impl-def*)

lemma *pmf-of-multiset-code* [code abstract]:
mapping-of-pmf (*pmf-of-multiset* A) = *pmf-of-multiset-impl* A
by (*simp add: pmf-of-multiset-impl-def*)

lemma *bernoulli-pmf-code* [code abstract]:
mapping-of-pmf (*bernoulli-pmf* p) =
 (if $p \leq 0$ then *Mapping.update* False 1 *Mapping.empty*
 else if $p \geq 1$ then *Mapping.update* True 1 *Mapping.empty*
 else *Mapping.update* False (1 - p) (*Mapping.update* True p *Mapping.empty*))
by (*intro mapping-of-pmfI*) (*auto simp: bernoulli-pmf.rep-eq lookup-update' set-pmf-eq*)

lemma *pmf-code* [code]: *pmf* p x = *Mapping.lookup-default* 0 (*mapping-of-pmf* p)
 x
unfolding *mapping-of-pmf-def Mapping.lookup-default-def*
by (*auto split: option.splits simp: id-def Mapping.lookup.abs-eq*)

lemma *set-pmf-code* [code]: *set-pmf* p = *Mapping.keys* (*mapping-of-pmf* p)
by *transfer* (*auto simp: dom-def set-pmf-eq*)

lemma *keys-mapping-of-pmf* [simp]: *Mapping.keys* (*mapping-of-pmf* p) = *set-pmf*
 p
by *transfer* (*auto simp: dom-def set-pmf-eq*)

definition *fold-combine-plus* **where**
fold-combine-plus = *comm-monoid-set.F* (*Mapping.combine* (*op* + :: *real* \Rightarrow -))
Mapping.empty

context
begin

interpretation *fold-combine-plus*: *combine-mapping-abel-semigroup* *op* + :: *real*
 \Rightarrow -
by *unfold-locales* (*simp-all add: add-ac*)

qualified lemma *lookup-default-fold-combine-plus*:
fixes A :: 'b set **and** f :: 'b \Rightarrow ('a, real) *mapping*
assumes *finite* A
shows *Mapping.lookup-default* 0 (*fold-combine-plus* f A) x =
 ($\sum_{y \in A} \text{Mapping.lookup-default } 0 \text{ (f y) x}$)
unfolding *fold-combine-plus-def* **using** *assms*
by (*induction* A *rule: finite-induct*)
 (*simp-all add: lookup-default-empty lookup-default-neutral-combine*)

qualified lemma *keys-fold-combine-plus*:
finite A \implies *Mapping.keys* (*fold-combine-plus* f A) = ($\bigcup_{x \in A} \text{Mapping.keys}$ (f

x)

by (*simp add: fold-combine-plus-def fold-combine-plus.keys-fold-combine*)

qualified lemma *fold-combine-plus-code* [*code*]:

fold-combine-plus g (*set* xs) = *foldr* ($\lambda x. \text{Mapping.combine } op + (g \ x)$) (*remdups* xs) *Mapping.empty*

by (*simp add: fold-combine-plus-def fold-combine-plus.fold-combine-code*)

private lemma *lookup-default-0-map-values*:

assumes $f \ x \ 0 = 0$

shows *Mapping.lookup-default* 0 (*Mapping.map-values* $f \ m$) $x = f \ x$ (*Mapping.lookup-default* $0 \ m \ x$)

unfolding *Mapping.lookup-default-def*

using *assms* **by** *transfer (auto split: option.splits)*

qualified lemma *mapping-of-bind-pmf*:

assumes *finite (set-pmf* p)

shows *mapping-of-pmf* (*bind-pmf* $p \ f$) =

fold-combine-plus ($\lambda x. \text{Mapping.map-values } (\lambda \cdot. op * (pmf \ p \ x))$)
(*mapping-of-pmf* ($f \ x$))) (*set-pmf* p)

using *assms*

by (*intro mapping-of-pmfI'*)

(*auto simp: keys-fold-combine-plus lookup-default-fold-combine-plus*
pmf-bind integral-measure-pmf lookup-default-0-map-values
lookup-default-mapping-of-pmf mult-ac)

lift-definition *bind-pmf-aux* :: $'a \ pmf \Rightarrow ('a \Rightarrow 'b \ pmf) \Rightarrow 'a \ set \Rightarrow ('b, \ real)$
mapping is

$\lambda(p :: 'a \ pmf) (f :: 'a \Rightarrow 'b \ pmf) (A :: 'a \ set) (x :: 'b).$

if $x \in (\bigcup y \in A. \text{set-pmf } (f \ y))$ *then*

Some (measure-pmf.expectation p ($\lambda y. \text{indicator } A \ y * pmf \ (f \ y) \ x$))

else None .

lemma *keys-bind-pmf-aux* [*simp*]:

Mapping.keys (*bind-pmf-aux* $p \ f \ A$) = $(\bigcup x \in A. \text{set-pmf } (f \ x))$

by *transfer (auto split: if-splits)*

lemma *lookup-default-bind-pmf-aux*:

Mapping.lookup-default 0 (*bind-pmf-aux* $p \ f \ A$) $x =$

(*if* $x \in (\bigcup y \in A. \text{set-pmf } (f \ y))$ *then*

measure-pmf.expectation p ($\lambda y. \text{indicator } A \ y * pmf \ (f \ y) \ x$) *else* 0)

unfolding *lookup-default-def* **by** *transfer' simp-all*

lemma *lookup-default-bind-pmf-aux'* [*simp*]:

Mapping.lookup-default 0 (*bind-pmf-aux* $p \ f$ (*set-pmf* p)) $x = pmf$ (*bind-pmf* $p \ f$) x

unfolding *lookup-default-def*

by *transfer (auto simp: pmf-bind AE-measure-pmf-iff set-pmf-eq*
intro!: integral-cong-AE integral-eq-zero-AE)

lemma *bind-pmf-aux-correct*:
 $mapping-of-pmf (bind-pmf p f) = bind-pmf-aux p f (set-pmf p)$
by (*intro mapping-of-pmfI'*) *simp-all*

lemma *bind-pmf-aux-code-aux*:
assumes *finite A*
shows $bind-pmf-aux p f A =$
 $fold-combine-plus (\lambda x. Mapping.map-values (\lambda-. op * (pmf p x))$
 $(mapping-of-pmf (f x))) A$ (**is** *?lhs = ?rhs*)

proof (*intro mapping-eqI'* [**where** $d = 0$])
fix x **assume** $x \in Mapping.keys ?lhs$
then obtain y **where** $y: y \in A \ x \in set-pmf (f y)$ **by** *auto*
hence $Mapping.lookup-default 0 ?lhs x =$
 $measure-pmf.expectation p (\lambda y. indicator A y * pmf (f y) x)$
by (*auto simp: lookup-default-bind-pmf-aux*)
also from *assms* **have** $\dots = (\sum y \in A. pmf p y * pmf (f y) x)$
by (*subst integral-measure-pmf [of A]*)
(auto simp: set-pmf-eq indicator-def mult-ac split: if-splits)
also from *assms* **have** $\dots = Mapping.lookup-default 0 ?rhs x$
by (*simp add: lookup-default-fold-combine-plus lookup-default-0-map-values*
 $lookup-default-mapping-of-pmf$)
finally show $Mapping.lookup-default 0 ?lhs x = Mapping.lookup-default 0 ?rhs$
 x .

qed (*insert assms, simp-all add: keys-fold-combine-plus*)

lemma *bind-pmf-aux-code* [*code*]:
 $bind-pmf-aux p f (set xs) =$
 $fold-combine-plus (\lambda x. Mapping.map-values (\lambda-. op * (pmf p x))$
 $(mapping-of-pmf (f x))) (set xs)$
by (*rule bind-pmf-aux-code-aux*) *simp-all*

lemmas *bind-pmf-code* [*code abstract*] = *bind-pmf-aux-correct*

end

hide-const (**open**) *fold-combine-plus*

lift-definition *cond-pmf-impl* :: $'a pmf \Rightarrow 'a set \Rightarrow ('a, real) mapping option$ **is**
 $\lambda p A. if A \cap set-pmf p = \{\}$ *then None else*
 $Some (\lambda x. if x \in A \cap set-pmf p$ *then* $Some (pmf p x / measure-pmf.prob p A)$
else None) .

lemma *cond-pmf-impl-code-alt*:
assumes *finite A*
shows $cond-pmf-impl p A = ($
 $let C = A \cap set-pmf p;$
 $prob = (\sum x \in C. pmf p x)$

```

      in if prob = 0 then
        None
      else
        Some (Mapping.map-values (λ- y. y / prob)
              (Mapping.filter (λk -. k ∈ C) (mapping-of-pmf p)))
proof –
  define C where C = A ∩ set-pmf p
  define prob where prob = (∑ x∈C. pmf p x)
  also note C-def
  also from assms have (∑ x∈A ∩ set-pmf p. pmf p x) = (∑ x∈A. pmf p x)
    by (intro sum.mono-neutral-left) (auto simp: set-pmf-eq)
  finally have prob1: prob = (∑ x∈A. pmf p x) .
  hence prob2: prob = measure-pmf.prob p A
    using assms by (subst measure-measure-pmf-finite) simp-all
  have prob3: prob = 0 ↔ A ∩ set-pmf p = {}
    by (subst prob1, subst sum-nonneg-eq-0-iff) (auto simp: set-pmf-eq assms)
  from assms have prob4: prob = measure-pmf.prob p C
    unfolding prob-def by (intro measure-measure-pmf-finite [symmetric]) (simp-all
add: C-def)

  show ?thesis
  proof (cases prob = 0)
    case True
      hence A ∩ set-pmf p = {} by (subst (asm) prob3)
      with True show ?thesis by (simp add: Let-def prob-def C-def cond-pmf-impl.abs-eq)
    next
      case False
        hence A: C ≠ {} unfolding C-def by (subst (asm) prob3) auto
        with prob3 have prob-nz: prob ≠ 0 by (auto simp: C-def)
        fix x
        have cond-pmf-impl p A =
          Some (mapping.Mapping (λx. if x ∈ C then
            Some (pmf p x / measure-pmf.prob p C) else None))
          (is - = Some ?m)
          using A prob2 prob4 unfolding C-def by transfer (auto simp: fun-eq-iff)
        also have ?m = Mapping.map-values (λ- y. y / prob)
          (Mapping.filter (λk -. k ∈ C) (mapping-of-pmf p))
          using prob-nz prob4 assms unfolding C-def
          by transfer (auto simp: fun-eq-iff set-pmf-eq)
        finally show ?thesis using False by (simp add: Let-def prob-def C-def)
  qed
qed

lemma cond-pmf-impl-code [code]:
  cond-pmf-impl p (set xs) = (
    let C = set xs ∩ set-pmf p;
    prob = (∑ x∈C. pmf p x)
    in if prob = 0 then
      None

```

```

else
  Some (Mapping.map-values ( $\lambda$ - y. y / prob)
        (Mapping.filter ( $\lambda$ k -. k  $\in$  C) (mapping-of-pmf p))))
by (rule cond-pmf-impl-code-alt) simp-all

```

lemma *cond-pmf-code* [code abstract]:

```

mapping-of-pmf (cond-pmf p A) =
  (case cond-pmf-impl p A of
    None  $\Rightarrow$  Code.abort (STR "cond-pmf with set of probability 0")
      ( $\lambda$ -. mapping-of-pmf (cond-pmf p A))
    | Some m  $\Rightarrow$  m)

```

proof (cases cond-pmf-impl p A)

```

case (Some m)
hence A: set-pmf p  $\cap$  A  $\neq$  {} by transfer (auto split: if-splits)
from Some have B: Mapping.keys m = set-pmf (cond-pmf p A)
  by (subst set-cond-pmf[OF A], transfer) (auto split: if-splits)
with Some A have mapping-of-pmf (cond-pmf p A) = m
  by (intro mapping-of-pmfI[OF B], transfer) (auto split: if-splits simp: pmf-cond)
with Some show ?thesis by simp

```

qed simp-all

lemma *binomial-pmf-code* [code abstract]:

```

mapping-of-pmf (binomial-pmf n p) = (
  if p < 0  $\vee$  p > 1 then
    Code.abort (STR "binomial-pmf with invalid probability")
      ( $\lambda$ -. mapping-of-pmf (binomial-pmf n p))
  else if p = 0 then Mapping.update 0 1 Mapping.empty
  else if p = 1 then Mapping.update n 1 Mapping.empty
  else Mapping.tabulate [0.. $\text{Suc } n$ ] ( $\lambda$ k. real (n choose k) * p ^ k * (1 - p) ^
(n - k)))
by (cases p < 0  $\vee$  p > 1)
  (simp, intro mapping-of-pmfI,
   auto simp: lookup-update' lookup-empty set-pmf-binomial-eq lookup-tabulate
split: if-splits)

```

lemma *pred-pmf-code* [code]:

```

pred-pmf P p = ( $\forall$  x  $\in$  set-pmf p. P x)
by (auto simp: pred-pmf-def)

```

lemma *mapping-of-pmf-pmf-of-list*:

```

assumes  $\bigwedge$ x. x  $\in$  snd ' set xs  $\Longrightarrow$  x > 0 sum-list (map snd xs) = 1
shows mapping-of-pmf (pmf-of-list xs) =
  Mapping.tabulate (remdups (map fst xs))
  ( $\lambda$ x. sum-list (map snd (filter ( $\lambda$ z. fst z = x) xs)))

```

proof –

```

from assms have wf: pmf-of-list-wf xs by (intro pmf-of-list-wfI) force

```

```

with assms have set-pmf (pmf-of-list xs) = fst ‘ set xs
  by (intro set-pmf-of-list-eq) auto
with wf show ?thesis
  by (intro mapping-of-pmfI) (auto simp: lookup-tabulate pmf-pmf-of-list)
qed

```

```

lemma mapping-of-pmf-pmf-of-list':
  assumes pmf-of-list-wf xs
  defines xs'  $\equiv$  filter ( $\lambda z. \text{snd } z \neq 0$ ) xs
  shows mapping-of-pmf (pmf-of-list xs) =
    Mapping.tabulate (remdups (map fst xs'))
      ( $\lambda x. \text{sum-list}$  (map snd (filter ( $\lambda z. \text{fst } z = x$ ) xs'))) (is - = ?rhs)

```

```

proof –
  have wf: pmf-of-list-wf xs' unfolding xs'-def by (rule pmf-of-list-remove-zeros)
  fact
  have pos:  $\forall x \in \text{snd}'\text{set } xs'. x > 0$  using assms(1) unfolding xs'-def
    by (force simp: pmf-of-list-wf-def)
  from assms have pmf-of-list xs = pmf-of-list xs'
    unfolding xs'-def by (subst pmf-of-list-remove-zeros) simp-all
  also from wf pos have mapping-of-pmf ... = ?rhs
    by (intro mapping-of-pmf-pmf-of-list) (auto simp: pmf-of-list-wf-def)
  finally show ?thesis .
qed

```

```

lemma pmf-of-list-wf-code [code]:
  pmf-of-list-wf xs  $\longleftrightarrow$  list-all ( $\lambda z. \text{snd } z \geq 0$ ) xs  $\wedge$  sum-list (map snd xs) = 1
  by (auto simp add: pmf-of-list-wf-def list-all-def)

```

```

lemma pmf-of-list-code [code abstract]:
  mapping-of-pmf (pmf-of-list xs) = (
    if pmf-of-list-wf xs then
      let xs' = filter ( $\lambda z. \text{snd } z \neq 0$ ) xs
      in Mapping.tabulate (remdups (map fst xs'))
        ( $\lambda x. \text{sum-list}$  (map snd (filter ( $\lambda z. \text{fst } z = x$ ) xs')))
    else
      Code.abort (STR "Invalid list for pmf-of-list") ( $\lambda-. \text{mapping-of-pmf}$  (pmf-of-list
xs)))
  using mapping-of-pmf-pmf-of-list'[of xs] by (simp add: Let-def)

```

```

lemma mapping-of-pmf-eq-iff [simp]:
  mapping-of-pmf p = mapping-of-pmf q  $\longleftrightarrow$  p = (q :: 'a pmf)
proof (transfer, intro iffI pmf-eqI)
  fix p q :: 'a pmf and x :: 'a
  assume ( $\lambda x. \text{if } \text{pmf } p \ x = 0 \text{ then None else Some } (\text{pmf } p \ x)$ ) =
    ( $\lambda x. \text{if } \text{pmf } q \ x = 0 \text{ then None else Some } (\text{pmf } q \ x)$ )
  hence ( $\text{if } \text{pmf } p \ x = 0 \text{ then None else Some } (\text{pmf } p \ x)$ ) =
    ( $\text{if } \text{pmf } q \ x = 0 \text{ then None else Some } (\text{pmf } q \ x)$ ) for x
  by (simp add: fun-eq-iff)
from this[of x] show pmf p x = pmf q x by (auto split: if-splits)

```


qed (*simp-all cong: if-cong*)

23.2 Code abbreviations for integrals and probabilities

Integrals and probabilities are defined for general measures, so we cannot give any code equations directly. We can, however, specialise these constants them to PMFs, give code equations for these specialised constants, and tell the code generator to unfold the original constants to the specialised ones whenever possible.

definition *pmf-integral* **where**

pmf-integral $p f = \text{lebesgue-integral } (\text{measure-pmf } p) (f :: - \Rightarrow \text{real})$

definition *pmf-set-integral* **where**

pmf-set-integral $p f A = \text{lebesgue-integral } (\text{measure-pmf } p) (\lambda x. \text{indicator } A x * f x :: \text{real})$

definition *pmf-prob* **where**

pmf-prob $p A = \text{measure-pmf.prob } p A$

lemma *pmf-prob-compl*: $\text{pmf-prob } p (-A) = 1 - \text{pmf-prob } p A$

using *measure-pmf.prob-compl[of A p]* **by** (*simp add: pmf-prob-def Compl-eq-Diff-UNIV*)

lemma *pmf-integral-pmf-set-integral* [code]:

pmf-integral $p f = \text{pmf-set-integral } p f (\text{set-pmf } p)$

unfolding *pmf-integral-def pmf-set-integral-def*

by (*intro integral-cong-AE*) (*simp-all add: AE-measure-pmf-iff*)

lemma *pmf-prob-pmf-set-integral*:

pmf-prob $p A = \text{pmf-set-integral } p (\lambda-. 1) A$

by (*simp add: pmf-prob-def pmf-set-integral-def*)

lemma *pmf-set-integral-code-alt-finite*:

finite $A \implies \text{pmf-set-integral } p f A = (\sum x \in A. \text{pmf } p x * f x)$

unfolding *pmf-set-integral-def*

by (*subst integral-measure-pmf[of A]*) (*auto simp: indicator-def mult-ac split: if-splits*)

lemma *pmf-set-integral-code* [code]:

pmf-set-integral $p f (\text{set } xs) = (\sum x \in \text{set } xs. \text{pmf } p x * f x)$

by (*rule pmf-set-integral-code-alt-finite*) *simp-all*

lemma *pmf-prob-code-alt-finite*:

finite $A \implies \text{pmf-prob } p A = (\sum x \in A. \text{pmf } p x)$

by (*simp add: pmf-prob-pmf-set-integral pmf-set-integral-code-alt-finite*)

lemma *pmf-prob-code* [code]:

pmf-prob $p (\text{set } xs) = (\sum x \in \text{set } xs. \text{pmf } p x)$

$pmf\text{-}prob\ p\ (List.\text{coset}\ xs) = 1 - (\sum_{x \in set\ xs} pmf\ p\ x)$
by (*simp-all add: pmf-prob-code-alt-finite pmf-prob-compl*)

lemma *pmf-prob-code-unfold* [*code-abbrev*]: $pmf\text{-}prob\ p = measure\text{-}pmf.\text{prob}\ p$
by (*intro ext*) (*simp add: pmf-prob-def*)

lemma *pmf-integral-code-unfold* [*code-abbrev*]: $pmf\text{-}integral\ p = measure\text{-}pmf.\text{expectation}\ p$
by (*intro ext*) (*simp add: pmf-integral-def*)

definition *pmf-of-alist* $xs = embed\text{-}pmf\ (\lambda x. case\ map\text{-}of\ xs\ x\ of\ Some\ p \Rightarrow p \mid None \Rightarrow 0)$

lemma *pmf-of-mapping-Mapping* [*code-post*]:
 $pmf\text{-}of\text{-}mapping\ (Mapping\ xs) = pmf\text{-}of\text{-}alist\ xs$
unfolding *pmf-of-mapping-def Mapping.lookup-default-def [abs-def] pmf-of-alist-def*
by *transfer simp-all*

instantiation *pmf* :: (*equal*) *equal*
begin

definition *equal-pmf* $p\ q = (mapping\text{-}of\text{-}pmf\ p = mapping\text{-}of\text{-}pmf\ (q :: 'a\ pmf))$

instance **by** *standard* (*simp add: equal-pmf-def*)
end

definition *single* :: '*a* \Rightarrow '*a* *multiset* **where**
single $s = \{\#s\# \}$

definition (**in** *term-syntax*)
 $pmfify :: ('a::\text{typerep}\ \text{multiset} \times (\text{unit} \Rightarrow \text{Code-Evaluation.term})) \Rightarrow$
 $'a \times (\text{unit} \Rightarrow \text{Code-Evaluation.term}) \Rightarrow$
 $'a\ pmf \times (\text{unit} \Rightarrow \text{Code-Evaluation.term})$ **where**
[*code-unfold*]: $pmfify\ A\ x =$
 $Code\text{-}Evaluation.\text{valtermify}\ pmf\text{-}of\text{-}multiset\ \{\cdot\}$
 $(Code\text{-}Evaluation.\text{valtermify}\ (op\ +)\ \{\cdot\}\ A\ \{\cdot\})$
 $(Code\text{-}Evaluation.\text{valtermify}\ single\ \{\cdot\}\ x)$

notation *fcomp* (**infixl** $\circ >$ 60)

notation *scomp* (**infixl** $\circ \rightarrow$ 60)

instantiation *pmf* :: (*random*) *random*
begin

definition

```

Quickcheck-Random.random i =
  Quickcheck-Random.random i ◦→ (λA.
    Quickcheck-Random.random i ◦→ (λx. Pair (pmfify A x)))

```

instance ..**end**

```

no-notation fcomp (infixl ◦> 60)
no-notation scomp (infixl ◦→ 60)

```

```

instantiation pmf :: (full-exhaustive) full-exhaustive
begin

```

```

definition full-exhaustive-pmf :: ('a pmf × (unit ⇒ term) ⇒ (bool × term list)
option) ⇒ natural ⇒ (bool × term list) option
where

```

```

  full-exhaustive-pmf f i =
    Quickcheck-Exhaustive.full-exhaustive (λA.
      Quickcheck-Exhaustive.full-exhaustive (λx. f (pmfify A x)) i) i

```

instance ..**end****end**

24 Finite Maps

theory *Fin-Map*

```

imports HOL-Analysis.Finite-Product-Measure HOL-Library.Finite-Map
begin

```

The *fmap* type can be instantiated to *polish-space*, needed for the proof of projective limit. *extensional* functions are used for the representation in order to stay close to the developments of (finite) products Pi_E and their sigma-algebra Pi_M .

```

type-notation fmap ((- ⇒F /-) [22, 21] 21)

```

```

unbundle fmap.lifting

```

24.1 Domain and Application

```

lift-definition domain::('i ⇒F 'a) ⇒ 'i set is dom .

```

```

lemma finite-domain[simp, intro]: finite (domain P)

```

by *transfer simp*

lift-definition *proj* :: ($'i \Rightarrow_F 'a$) \Rightarrow $'i \Rightarrow 'a$ ($'((-)')_F$ [0] 1000) **is**
 $\lambda f x.$ if $x \in \text{dom } f$ then the $(f x)$ else *undefined* .

declare [[*coercion proj*]]

lemma *extensional-proj*[*simp, intro*]: $(P)_F \in \text{extensional } (\text{domain } P)$
 by *transfer (auto simp: extensional-def)*

lemma *proj-undefined*[*simp, intro*]: $i \notin \text{domain } P \Longrightarrow P i = \text{undefined}$
 using *extensional-proj*[*of P*] **unfolding** *extensional-def* **by** *auto*

lemma *finmap-eq-iff*: $P = Q \iff (\text{domain } P = \text{domain } Q \wedge (\forall i \in \text{domain } P. P i = Q i))$
 apply *transfer*
 apply (*safe intro!*: *ext*)
 subgoal for $P Q x$
 by (*cases* $x \in \text{dom } P$; *cases* $P x$) (*auto dest!*: *bspec*[**where** $x=x$])
 done

24.2 Constructor of Finite Maps

lift-definition *finmap-of*:: $'i \text{ set} \Rightarrow ('i \Rightarrow 'a) \Rightarrow ('i \Rightarrow_F 'a)$ **is**
 $\lambda I f x.$ if $x \in I \wedge \text{finite } I$ then *Some* $(f x)$ else *None*
 by (*simp add: dom-def*)

lemma *proj-finmap-of*[*simp*]:
 assumes *finite inds*
 shows $(\text{finmap-of } \text{inds } f)_F = \text{restrict } f \text{ inds}$
 using *assms*
 by *transfer force*

lemma *domain-finmap-of*[*simp*]:
 assumes *finite inds*
 shows $\text{domain } (\text{finmap-of } \text{inds } f) = \text{inds}$
 using *assms*
 by *transfer (auto split: if-splits)*

lemma *finmap-of-eq-iff*[*simp*]:
 assumes *finite i finite j*
 shows $\text{finmap-of } i \text{ m} = \text{finmap-of } j \text{ n} \iff i = j \wedge (\forall k \in i. m k = n k)$
 using *assms* **by** (*auto simp: finmap-eq-iff*)

lemma *finmap-of-inj-on-extensional-finite*:
 assumes *finite K*
 assumes $S \subseteq \text{extensional } K$
 shows *inj-on* $(\text{finmap-of } K) S$
proof (*rule inj-onI*)

fix $x y :: 'a \Rightarrow 'b$
assume $\text{finmap-of } K x = \text{finmap-of } K y$
hence $(\text{finmap-of } K x)_F = (\text{finmap-of } K y)_F$ **by** *simp*
moreover
assume $x \in S y \in S$ **hence** $x \in \text{extensional } K y \in \text{extensional } K$ **using** *assms*
by *auto*
ultimately
show $x = y$ **using** *assms* **by** (*simp add: extensional-restrict*)
qed

24.3 Product set of Finite Maps

This is Pi for Finite Maps, most of this is copied

definition $Pi' :: 'i \text{ set} \Rightarrow ('i \Rightarrow 'a \text{ set}) \Rightarrow ('i \Rightarrow_F 'a) \text{ set}$ **where**

$$Pi' I A = \{ P. \text{domain } P = I \wedge (\forall i. i \in I \longrightarrow (P)_F i \in A i) \}$$

syntax

$$-Pi' :: [pttrn, 'a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Rightarrow 'b) \text{ set} \quad ((\exists \Pi' \text{ -} \in \text{ -} / \text{ -}) \quad 10)$$

translations

$$\Pi' x \in A. B == \text{CONST } Pi' A (\lambda x. B)$$

24.3.1 Basic Properties of Pi'

lemma $Pi'-I[\text{intro!}]$: $\text{domain } f = A \Longrightarrow (\bigwedge x. x \in A \Longrightarrow f x \in B x) \Longrightarrow f \in Pi' A B$

by (*simp add: Pi'-def*)

lemma $Pi'-I[\text{simp}]$: $\text{domain } f = A \Longrightarrow (\bigwedge x. x \in A \longrightarrow f x \in B x) \Longrightarrow f \in Pi' A B$

by (*simp add: Pi'-def*)

lemma $Pi'-\text{mem}$: $f \in Pi' A B \Longrightarrow x \in A \Longrightarrow f x \in B x$

by (*simp add: Pi'-def*)

lemma $Pi'-\text{iff}$: $f \in Pi' I X \longleftrightarrow \text{domain } f = I \wedge (\forall i \in I. f i \in X i)$

unfolding $Pi'-\text{def}$ **by** *auto*

lemma $Pi'-E[\text{elim}]$:

$$f \in Pi' A B \Longrightarrow (f x \in B x \Longrightarrow \text{domain } f = A \Longrightarrow Q) \Longrightarrow (x \notin A \Longrightarrow Q) \Longrightarrow Q$$

by(*auto simp: Pi'-def*)

lemma $\text{in-}Pi'-\text{cong}$:

$$\text{domain } f = \text{domain } g \Longrightarrow (\bigwedge w. w \in A \Longrightarrow f w = g w) \Longrightarrow f \in Pi' A B \longleftrightarrow g \in Pi' A B$$

by (*auto simp: Pi'-def*)

lemma $Pi'-\text{eq-empty}[\text{simp}]$:

$$\text{assumes } \text{finite } A \text{ shows } (Pi' A B) = \{\} \longleftrightarrow (\exists x \in A. B x = \{\})$$

```

using assms
apply (simp add: Pi'-def, auto)
apply (drule-tac x = finmap-of A (λu. SOME y. y ∈ B u) in spec, auto)
apply (cut-tac P= %y. y ∈ B i in some-eq-ex, auto)
done

```

```

lemma Pi'-mono: (λx. x ∈ A ⇒ B x ⊆ C x) ⇒ Pi' A B ⊆ Pi' A C
by (auto simp: Pi'-def)

```

```

lemma Pi-Pi': finite A ⇒ (PiE A B) = proj ' Pi' A B
apply (auto simp: Pi'-def Pi-def extensional-def)
apply (rule-tac x = finmap-of A (restrict x A) in image-eqI)
apply auto
done

```

24.4 Topological Space of Finite Maps

```

instantiation fmap :: (type, topological-space) topological-space
begin

```

```

definition open-fmap :: ('a ⇒F 'b) set ⇒ bool where
  [code del]: open-fmap = generate-topology {Pi' a b | a b. ∀ i ∈ a. open (b i)}

```

```

lemma open-Pi'I: (λi. i ∈ I ⇒ open (A i)) ⇒ open (Pi' I A)
by (auto intro: generate-topology.Basis simp: open-fmap-def)

```

```

instance using topological-space-generate-topology
by intro-classes (auto simp: open-fmap-def class.topological-space-def)

```

```

end

```

```

lemma open-restricted-space:
  shows open {m. P (domain m)}

```

```

proof –

```

```

  have {m. P (domain m)} = (∪ i ∈ Collect P. {m. domain m = i}) by auto
  also have open ...

```

```

  proof (rule, safe, cases)

```

```

    fix i::'a set

```

```

    assume finite i

```

```

    hence {m. domain m = i} = Pi' i (λ-. UNIV) by (auto simp: Pi'-def)

```

```

    also have open ... by (auto intro: open-Pi'I simp: (finite i))

```

```

    finally show open {m. domain m = i} .

```

```

  next

```

```

    fix i::'a set

```

```

    assume ¬ finite i hence {m. domain m = i} = {} by auto

```

```

    also have open ... by simp

```

```

    finally show open {m. domain m = i} .

```

```

  qed

```

```

  finally show ?thesis .

```

qed

lemma *closed-restricted-space*:

shows *closed* $\{m. P (\text{domain } m)\}$
using *open-restricted-space*[of $\lambda x. \neg P x$]
unfolding *closed-def* **by** (*rule back-subst*) *auto*

lemma *tendsto-proj*: $((\lambda x. x) \longrightarrow a) F \Longrightarrow ((\lambda x. (x)_F i) \longrightarrow (a)_F i) F$
unfolding *tendsto-def*

proof *safe*

fix $S :: 'b$ *set*
let $?S = Pi' (\text{domain } a) (\lambda x. \text{if } x = i \text{ then } S \text{ else } UNIV)$
assume *open* S **hence** *open* $?S$ **by** (*auto intro!*: *open-Pi'I*)
moreover **assume** $\forall S. \text{open } S \longrightarrow a \in S \longrightarrow \text{eventually } (\lambda x. x \in S) F a i \in S$
ultimately **have** *eventually* $(\lambda x. (x)_F i \in S) F$ **by** *auto*
thus *eventually* $(\lambda x. (x)_F i \in S) F$
by *eventually-elim* (*insert* $\langle a i \in S \rangle$, *force simp*: *Pi'-iff split*: *if-split-asm*)

qed

lemma *continuous-proj*:

shows *continuous-on* $s (\lambda x. (x)_F i)$
unfolding *continuous-on-def* **by** (*safe intro!*: *tendsto-proj tendsto-ident-at*)

instance *fmap* :: (*type*, *first-countable-topology*) *first-countable-topology*

proof

fix $x :: 'a \Rightarrow_F 'b$
have $\forall i. \exists A. \text{countable } A \wedge (\forall a \in A. x i \in a) \wedge (\forall a \in A. \text{open } a) \wedge$
 $(\forall S. \text{open } S \wedge x i \in S \longrightarrow (\exists a \in A. a \subseteq S)) \wedge (\forall a b. a \in A \longrightarrow b \in A \longrightarrow a$
 $\cap b \in A)$ (**is** $\forall i. ?th i$)

proof

fix i **from** *first-countable-basis-Int-stableE*[of $x i$] **guess** A .

thus $?th i$ **by** (*intro exI*[**where** $x=A$]) *simp*

qed

then **guess** A **unfolding** *choice-iff* **.. note** $A = \text{this}$

hence *open-sub*: $\bigwedge i S. i \in \text{domain } x \Longrightarrow \text{open } (S i) \Longrightarrow x i \in (S i) \Longrightarrow (\exists a \in A. i.$
 $a \subseteq (S i))$ **by** *auto*

have *A-notempty*: $\bigwedge i. i \in \text{domain } x \Longrightarrow A i \neq \{\}$ **using** *open-sub*[of $\lambda-. UNIV$]
by *auto*

let $?A = (\lambda f. Pi' (\text{domain } x) f) ' (Pi_E (\text{domain } x) A)$

show $\exists A :: \text{nat} \Rightarrow ('a \Rightarrow_F 'b)$ *set*. $(\forall i. x \in (A i) \wedge \text{open } (A i)) \wedge (\forall S. \text{open } S \wedge$
 $x \in S \longrightarrow (\exists i. A i \subseteq S))$

proof (*rule first-countableI*[**where** $A=?A$], *safe*)

show *countable* $?A$ **using** A **by** (*simp add*: *countable-PiE*)

next

fix $S :: ('a \Rightarrow_F 'b)$ *set* **assume** *open* S $x \in S$

thus $\exists a \in ?A. a \subseteq S$ **unfolding** *open-fmap-def*

proof (*induct rule*: *generate-topology.induct*)

case *UNIV* **thus** $?case$ **by** (*auto simp add*: *ex-in-conv PiE-eq-empty-iff*
 $A\text{-notempty}$)

```

next
  case (Int a b)
  then obtain f g where
    f ∈ PiE (domain x) A Pi' (domain x) f ⊆ a g ∈ PiE (domain x) A Pi'
    (domain x) g ⊆ b
  by auto
  thus ?case using A
  by (auto simp: Pi'-iff PiE-iff extensional-def Int-stable-def
    intro!: bexI[where x=λi. f i ∩ g i])
next
  case (UN B)
  then obtain b where x ∈ b b ∈ B by auto
  hence ∃ a ∈ ?A. a ⊆ b using UN by simp
  thus ?case using ⟨b ∈ B⟩ by blast
next
  case (Basis s)
  then obtain a b where xs: x ∈ Pi' a b s = Pi' a b ∧ i. i ∈ a ⇒ open (b i)
by auto
  have ∀ i. ∃ a. (i ∈ domain x ∧ open (b i) ∧ (x)F i ∈ b i) → (a ∈ A i ∧ a ⊆
  b i)
  using open-sub[of - b] by auto
  then obtain b'
  where ∧ i. i ∈ domain x ⇒ open (b i) ⇒ (x)F i ∈ b i ⇒ (b' i ∈ A i ∧
  b' i ⊆ b i)
  unfolding choice-iff by auto
  with xs have ∧ i. i ∈ a ⇒ (b' i ∈ A i ∧ b' i ⊆ b i) Pi' a b' ⊆ Pi' a b
  by (auto simp: Pi'-iff intro!: Pi'-mono)
  thus ?case using xs
  by (intro bexI[where x=Pi' a b']
    (auto simp: Pi'-iff intro!: image-eqI[where x=restrict b' (domain x)]))
qed
qed (insert A, auto simp: PiE-iff intro!: open-Pi'I)
qed

```

24.5 Metric Space of Finite Maps

```

instantiation fmap :: (type, metric-space) dist
begin

```

```

definition dist-fmap where

```

```

  dist P Q = Max (range (λi. dist ((P)F i) ((Q)F i))) + (if domain P = domain
  Q then 0 else 1)

```

```

instance ..

```

```

end

```

```

instantiation fmap :: (type, metric-space) uniformity-dist
begin

```


definition *[code del]*:

(*uniformity* :: (('a, 'b) fmap × ('a ⇒_F 'b)) filter) =
 (INF e:{0 <..}. principal {(x, y). dist x y < e})

instance

by *standard* (rule *uniformity-fmap-def*)

end

declare *uniformity-Abort*[**where** 'a=(*'a* ⇒_F 'b)::*metric-space*), *code*]

instantiation *fmap* :: (*type*, *metric-space*) *metric-space*

begin

lemma *finite-proj-image'*: $x \notin \text{domain } P \implies \text{finite } ((P)_F \text{ ' } S)$

by (rule *finite-subset*[of - proj *P* ' (*domain P* ∩ *S* ∪ {*x*})]) *auto*

lemma *finite-proj-image*: *finite* ((*P*)_F ' *S*)

by (*cases* ∃*x*. $x \notin \text{domain } P$) (*auto intro: finite-proj-image' finite-subset*[**where** *B=domain P*])

lemma *finite-proj-diag*: *finite* ((λ*i*. *d* ((*P*)_F *i*) ((*Q*)_F *i*)) ' *S*)

proof –

have (λ*i*. *d* ((*P*)_F *i*) ((*Q*)_F *i*)) ' *S* = (λ(*i*, *j*). *d* *i* *j*) ' ((λ*i*. ((*P*)_F *i*, (*Q*)_F *i*)) ' *S*) by *auto*

moreover have ((λ*i*. ((*P*)_F *i*, (*Q*)_F *i*)) ' *S*) ⊆ (λ*i*. (*P*)_F *i*) ' *S* × (λ*i*. (*Q*)_F *i*) ' *S* by *auto*

moreover have *finite* ... using *finite-proj-image*[of *P S*] *finite-proj-image*[of *Q S*]

by (*intro finite-cartesian-product simp-all*)

ultimately show ?*thesis* by (*simp add: finite-subset*)

qed

lemma *dist-le-1-imp-domain-eq*:

shows $\text{dist } P \ Q < 1 \implies \text{domain } P = \text{domain } Q$

by (*simp add: dist-fmap-def finite-proj-diag split: if-split-asm*)

lemma *dist-proj*:

shows $\text{dist } ((x)_F \ i) \ ((y)_F \ i) \leq \text{dist } x \ y$

proof –

have $\text{dist } (x \ i) \ (y \ i) \leq \text{Max } (\text{range } (\lambda i. \text{dist } (x \ i) \ (y \ i)))$

by (*simp add: Max-ge-iff finite-proj-diag*)

also have ... ≤ *dist x y* by (*simp add: dist-fmap-def*)

finally show ?*thesis* .

qed

lemma *dist-finmap-lessI*:

assumes $\text{domain } P = \text{domain } Q$

assumes $0 < e$

assumes $\bigwedge i. i \in \text{domain } P \implies \text{dist } (P \ i) \ (Q \ i) < e$

```

shows  $\text{dist } P \ Q < e$ 
proof -
  have  $\text{dist } P \ Q = \text{Max } (\text{range } (\lambda i. \text{dist } (P \ i) \ (Q \ i)))$ 
    using assms by (simp add: dist-fmap-def finite-proj-diag)
  also have  $\dots < e$ 
  proof (subst Max-less-iff, safe)
    fix  $i$ 
    show  $\text{dist } ((P)_F \ i) \ ((Q)_F \ i) < e$  using assms
      by (cases i ∈ domain P simp-all)
    qed (simp add: finite-proj-diag)
  finally show ?thesis .
qed

instance
proof
  fix  $S :: ('a \Rightarrow_F 'b) \ \text{set}$ 
  have  $*$ :  $\text{open } S = (\forall x \in S. \exists e > 0. \forall y. \text{dist } y \ x < e \longrightarrow y \in S)$  (is - = ?od)
  proof
    assume open S
    thus ?od
      unfolding open-fmap-def
    proof (induct rule: generate-topology.induct)
      case UNIV thus ?case by (auto intro: zero-less-one)
    next
      case (Int a b)
      show ?case
      proof safe
        fix  $x$  assume  $x: x \in a \ x \in b$ 
        with Int obtain  $e1 \ e2$  where
           $e1 > 0 \ \forall y. \text{dist } y \ x < e1 \longrightarrow y \in a \ e2 > 0 \ \forall y. \text{dist } y \ x < e2 \longrightarrow y \in b$  by
force
        thus  $\exists e > 0. \forall y. \text{dist } y \ x < e \longrightarrow y \in a \cap b$ 
          by (auto intro!: exI[where x=min e1 e2])
        qed
      next
        case (UN K)
        show ?case
        proof safe
          fix  $x \ X$  assume  $x \in X$  and  $X: X \in K$ 
          with UN obtain  $e$  where  $e > 0 \ \wedge y. \text{dist } y \ x < e \longrightarrow y \in X$  by force
          with  $X$  show  $\exists e > 0. \forall y. \text{dist } y \ x < e \longrightarrow y \in \bigcup K$  by auto
          qed
        next
          case (Basis s) then obtain  $a \ b$  where  $s: s = P \ i' \ a \ b$  and  $b: \bigwedge i. i \in a \implies$ 
open (b i) by auto
          show ?case
          proof safe
            fix  $x$  assume  $x \in s$ 
            hence [simp]: finite a and a-dom: a = domain x using  $s$  by (auto simp:

```

Pi'-iff)
obtain *es* **where** $es: \forall i \in a. es\ i > 0 \wedge (\forall y. dist\ y\ (proj\ x\ i) < es\ i \longrightarrow y \in b\ i)$
using $b\ \langle x \in s \rangle$ **by** *atomize-elim* (*intro bchoice, auto simp: open-dist s*)
hence *in-b*: $\bigwedge i y. i \in a \implies dist\ y\ (proj\ x\ i) < es\ i \implies y \in b\ i$ **by** *auto*
show $\exists e > 0. \forall y. dist\ y\ x < e \longrightarrow y \in s$
proof (*cases, rule, safe*)
assume $a \neq \{\}$
show $0 < min\ 1\ (Min\ (es\ 'a))$ **using** *es* **by** (*auto simp: 'a \neq \{\}*)
fix *y* **assume** $d: dist\ y\ x < min\ 1\ (Min\ (es\ 'a))$
show $y \in s$ **unfolding** *s*
proof
show $domain\ y = a$ **using** $d\ s\ \langle a \neq \{\} \rangle$ **by** (*auto simp: dist-le-1-imp-domain-eq*
a-dom)
fix *i* **assume** $i \in a$
hence $dist\ ((y)_F\ i)\ ((x)_F\ i) < es\ i$ **using** *d*
by (*auto simp: dist-fmap-def 'a \neq \{\}*) *intro!*: *le-less-trans[OF dist-proj]*
with *i* **show** $y\ i \in b\ i$ **by** (*rule in-b*)
qed
next
assume $\neg a \neq \{\}$
thus $\exists e > 0. \forall y. dist\ y\ x < e \longrightarrow y \in s$
using $s\ \langle x \in s \rangle$ **by** (*auto simp: Pi'-def dist-le-1-imp-domain-eq intro!*:
exI[where x=1])
qed
qed
qed
next
assume $\forall x \in S. \exists e > 0. \forall y. dist\ y\ x < e \longrightarrow y \in S$
then obtain *e* **where** *e-pos*: $\bigwedge x. x \in S \implies e\ x > 0$ **and**
e-in: $\bigwedge x y. x \in S \implies dist\ y\ x < e\ x \implies y \in S$
unfolding *bchoice-iff*
by *auto*
have *S-eq*: $S = \bigcup \{Pi'\ a\ b \mid a\ b. \exists x \in S. domain\ x = a \wedge b = (\lambda i. ball\ (x\ i)\ (e\ x))\}$
proof *safe*
fix *x* **assume** $x \in S$
thus $x \in \bigcup \{Pi'\ a\ b \mid a\ b. \exists x \in S. domain\ x = a \wedge b = (\lambda i. ball\ (x\ i)\ (e\ x))\}$
using *e-pos* **by** (*auto intro!*: *exI[where x=Pi' (domain x) (\lambda i. ball (x i) (e x))]*)
next
fix *x y*
assume $y \in S$
moreover
assume $x \in (\Pi'\ i \in domain\ y. ball\ (y\ i)\ (e\ y))$
hence $dist\ x\ y < e\ y$ **using** *e-pos* ($y \in S$)
by (*auto simp: dist-fmap-def Pi'-iff finite-proj-diag dist-commute*)
ultimately show $x \in S$ **by** (*rule e-in*)
qed

```

also have open ...
  unfolding open-fmap-def
  by (intro generate-topology.UN) (auto intro: generate-topology.Basis)
  finally show open S .
qed
show open S = ( $\forall x \in S. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in S$ )
  unfolding * eventually-uniformity-metric
  by (simp del: split-paired-All add: dist-fmap-def dist-commute eq-commute)
next
  fix P Q :: 'a  $\Rightarrow_F$  'b
  have Max-eq-iff:  $\bigwedge A m. \text{ finite } A \implies A \neq \{\} \implies (\text{Max } A = m) = (m \in A \wedge (\forall a \in A. a \leq m))$ 
  by (auto intro: Max-in Max-eqI)
  show dist P Q = 0  $\longleftrightarrow$  P = Q
  by (auto simp: finmap-eq-iff dist-fmap-def Max-ge-iff finite-proj-diag Max-eq-iff
    add-nonneg-eq-0-iff
    intro!: Max-eqI image-eqI[where x=undefined])
next
  fix P Q R :: 'a  $\Rightarrow_F$  'b
  let ?dists =  $\lambda P Q i. \text{ dist } ((P)_F i) ((Q)_F i)$ 
  let ?dpq = ?dists P Q and ?dpr = ?dists P R and ?dqr = ?dists Q R
  let ?dom =  $\lambda P Q. (\text{if domain } P = \text{domain } Q \text{ then } 0 \text{ else } 1::\text{real})$ 
  have dist P Q = Max (range ?dpq) + ?dom P Q
  by (simp add: dist-fmap-def)
  also obtain t where  $t \in \text{range } ?dpq \ t = \text{Max } (\text{range } ?dpq)$  by (simp add:
    finite-proj-diag)
  then obtain i where  $\text{Max } (\text{range } ?dpq) = ?dpq i$  by auto
  also have  $?dpq i \leq ?dpr i + ?dqr i$  by (rule dist-triangle2)
  also have  $?dpr i \leq \text{Max } (\text{range } ?dpr)$  by (simp add: finite-proj-diag)
  also have  $?dqr i \leq \text{Max } (\text{range } ?dqr)$  by (simp add: finite-proj-diag)
  also have  $?dom P Q \leq ?dom P R + ?dom Q R$  by simp
  finally show  $\text{dist } P Q \leq \text{dist } P R + \text{dist } Q R$  by (simp add: dist-fmap-def
    ac-simps)
qed
end

```

24.6 Complete Space of Finite Maps

lemma tendsto-finmap:

```

fixes f :: nat  $\Rightarrow$  ('i  $\Rightarrow_F$  ('a :: metric-space))
assumes ind-f:  $\bigwedge n. \text{ domain } (f n) = \text{domain } g$ 
assumes proj-g:  $\bigwedge i. i \in \text{domain } g \implies (\lambda n. (f n) i) \longrightarrow g i$ 
shows  $f \longrightarrow g$ 
unfolding tendsto-iff

```

proof safe

```

fix e :: real assume 0 < e
let ?dists =  $\lambda x i. \text{ dist } ((f x)_F i) ((g)_F i)$ 
have eventually ( $\lambda x. \forall i \in \text{domain } g. ?dists x i < e$ ) sequentially

```

```

    using finite-domain[of g] proj-g
  proof induct
    case (insert i G)
    with ⟨0 < e⟩ have eventually (λx. ?dists x i < e) sequentially by (auto simp
add: tendsto-iff)
    moreover
    from insert have eventually (λx. ∀ i ∈ G. dist ((f x)F i) ((g)F i) < e) sequen-
tially by simp
    ultimately show ?case by eventually-elim auto
  qed simp
  thus eventually (λx. dist (f x) g < e) sequentially
  by eventually-elim (auto simp add: dist-fmap-def finite-proj-diag ind-f ⟨0 < e⟩)
qed

instance fmap :: (type, complete-space) complete-space
proof
  fix P::nat ⇒ 'a ⇒F 'b
  assume Cauchy P
  then obtain Nd where Nd: ∧n. n ≥ Nd ⇒ dist (P n) (P Nd) < 1
  by (force simp: Cauchy-altdef2)
  define d where d = domain (P Nd)
  with Nd have dim: ∧n. n ≥ Nd ⇒ domain (P n) = d using dist-le-1-imp-domain-eq
by auto
  have [simp]: finite d unfolding d-def by simp
  define p where p i n = P n i for i n
  define q where q i = lim (p i) for i
  define Q where Q = finmap-of d q
  have q: ∧i. i ∈ d ⇒ q i = Q i by (auto simp add: Q-def Abs-fmap-inverse)
  {
    fix i assume i ∈ d
    have Cauchy (p i) unfolding Cauchy-altdef2 p-def
  proof safe
    fix e::real assume 0 < e
    with ⟨Cauchy P⟩ obtain N where N: ∧n. n ≥ N ⇒ dist (P n) (P N) <
min e 1
    by (force simp: Cauchy-altdef2 min-def)
    hence ∧n. n ≥ N ⇒ domain (P n) = domain (P N) using dist-le-1-imp-domain-eq
  by auto
    with dim have dim: ∧n. n ≥ N ⇒ domain (P n) = d by (metis
nat-le-linear)
  show ∃ N. ∀ n ≥ N. dist ((P n) i) ((P N) i) < e
  proof (safe intro!: exI[where x=N])
    fix n assume N ≤ n have N ≤ N by simp
    have dist ((P n) i) ((P N) i) ≤ dist (P n) (P N)
    using dim[OF ⟨N ≤ n⟩] dim[OF ⟨N ≤ N⟩] ⟨i ∈ d⟩
    by (auto intro!: dist-proj)
    also have ... < e using N[OF ⟨N ≤ n⟩] by simp
    finally show dist ((P n) i) ((P N) i) < e .
  qed

```

```

qed
hence convergent (p i) by (metis Cauchy-convergent-iff)
hence p i  $\longrightarrow$  q i unfolding q-def convergent-def by (metis limI)
} note p = this
have P  $\longrightarrow$  Q
proof (rule metric-LIMSEQ-I)
  fix e::real assume 0 < e
  have  $\exists ni. \forall i \in d. \forall n \geq ni. i. dist (p i n) (q i) < e$ 
  proof (safe intro!: bchoice)
    fix i assume i  $\in$  d
    from p[OF  $\langle i \in d \rangle$ , THEN metric-LIMSEQ-D, OF  $\langle 0 < e \rangle$ ]
    show  $\exists no. \forall n \geq no. dist (p i n) (q i) < e$  .
  qed then guess ni .. note ni = this
  define N where N = max Nd (Max (ni ' d))
  show  $\exists N. \forall n \geq N. dist (P n) Q < e$ 
  proof (safe intro!: exI[where x=N])
    fix n assume N  $\leq$  n
    hence dom: domain (P n) = d domain Q = d domain (P n) = domain Q
      using dim by (simp-all add: N-def Q-def dim-def Abs-fmap-inverse)
    show dist (P n) Q < e
    proof (rule dist-finmap-lessI[OF dom(3)  $\langle 0 < e \rangle$ ])
      fix i
      assume i  $\in$  domain (P n)
      hence ni i  $\leq$  Max (ni ' d) using dom by simp
      also have ...  $\leq$  N by (simp add: N-def)
      finally show dist ((P n)F i) ((Q)F i) < e using ni  $\langle i \in$  domain (P n) $\rangle \langle N$ 
         $\leq n \rangle$  dom
        by (auto simp: p-def q N-def less-imp-le)
    qed
  qed
  qed
  thus convergent P by (auto simp: convergent-def)
qed

```

24.7 Second Countable Space of Finite Maps

instantiation fmap :: (countable, second-countable-topology) second-countable-topology
begin

definition basis-proj::'b set set

where basis-proj = (SOME B. countable B \wedge topological-basis B)

lemma countable-basis-proj: countable basis-proj and basis-proj: topological-basis
basis-proj

unfolding basis-proj-def by (intro is-basis countable-basis)+

definition basis-finmap::('a \Rightarrow_F 'b) set set

where basis-finmap = {Pi' I S | I S. finite I \wedge ($\forall i \in I. S i \in$ basis-proj)}

lemma *in-basis-finmapI*:
assumes *finite I* **assumes** $\bigwedge i. i \in I \implies S i \in \text{basis-proj}$
shows $Pi' I S \in \text{basis-finmap}$
using *assms* **unfolding** *basis-finmap-def* **by** *auto*

lemma *basis-finmap-eq*:
assumes $\text{basis-proj} \neq \{\}$
shows $\text{basis-finmap} = (\lambda f. Pi' (\text{domain } f) (\lambda i. \text{from-nat-into basis-proj } ((f)_F i)))$
(UNIV::('a \Rightarrow_F nat) set) (is - = ?f ' -)
unfolding *basis-finmap-def*

proof *safe*
fix $I::'a \text{ set}$ **and** $S::'a \Rightarrow 'b \text{ set}$
assume $\text{finite } I \forall i \in I. S i \in \text{basis-proj}$
hence $Pi' I S = ?f (\text{finmap-of } I (\lambda x. \text{to-nat-on basis-proj } (S x)))$
by (*force simp: Pi'-def countable-basis-proj*)
thus $Pi' I S \in \text{range } ?f$ **by** *simp*

next
fix x **and** $f::'a \Rightarrow_F \text{nat}$
show $\exists I S. (\Pi' i \in \text{domain } f. \text{from-nat-into basis-proj } ((f)_F i)) = Pi' I S \wedge$
 $\text{finite } I \wedge (\forall i \in I. S i \in \text{basis-proj})$
using *assms* **by** (*auto intro: from-nat-into*)

qed

lemma *basis-finmap-eq-empty*: $\text{basis-proj} = \{\} \implies \text{basis-finmap} = \{Pi' \{\} \text{ undefined}\}$
by (*auto simp: Pi'-iff basis-finmap-def*)

lemma *countable-basis-finmap*: *countable basis-finmap*
by (*cases basis-proj = \{\}*) (*auto simp: basis-finmap-eq basis-finmap-eq-empty*)

lemma *finmap-topological-basis*:
topological-basis basis-finmap

proof (*subst topological-basis-iff, safe*)
fix B' **assume** $B' \in \text{basis-finmap}$
thus *open B'*
by (*auto intro!: open-Pi'I topological-basis-open[OF basis-proj]*)
simp: topological-basis-def basis-finmap-def Let-def

next
fix $O'::('a \Rightarrow_F 'b) \text{ set}$ **and** x
assume $O': \text{open } O' x \in O'$
then obtain a **where** a :
 $x \in Pi' (\text{domain } x) a \wedge Pi' (\text{domain } x) a \subseteq O' \wedge i. i \in \text{domain } x \implies \text{open } (a i)$
unfolding *open-fmap-def*

proof (*atomize-elim, induct rule: generate-topology.induct*)
case (*Int a b*)
let $?p = \lambda a f. x \in Pi' (\text{domain } x) f \wedge Pi' (\text{domain } x) f \subseteq a \wedge (\forall i. i \in \text{domain } x \implies \text{open } (f i))$
from *Int* **obtain** $f g$ **where** $?p a f ?p b g$ **by** *auto*

```

thus ?case by (force intro!: exI[where  $x=\lambda i. f i \cap g i$ ] simp: Pi'-def)
next
  case (UN k)
  then obtain  $kk$   $a$  where  $x \in kk$   $kk \in k$   $x \in Pi' (domain x)$   $a \in Pi' (domain x)$ 
 $a \subseteq kk$ 
   $\bigwedge i. i \in domain x \implies open (a i)$ 
  by force
  thus ?case by blast
qed (auto simp: Pi'-def)
have  $\exists B.$ 
  ( $\forall i \in domain x. x i \in B i \wedge B i \subseteq a i \wedge B i \in basis\text{-}proj$ )
proof (rule bchoice, safe)
  fix  $i$  assume  $i \in domain x$ 
  hence  $open (a i)$   $x i \in a i$  using  $a$  by auto
  from topological-basisE[OF basis-proj this] guess  $b'$ .
  thus  $\exists y. x i \in y \wedge y \subseteq a i \wedge y \in basis\text{-}proj$  by auto
qed
then guess  $B$  .. note  $B = this$ 
define  $B'$  where  $B' = Pi' (domain x) (\lambda i. (B i)::'b\ set)$ 
have  $B' \subseteq Pi' (domain x)$   $a$  using  $B$  by (auto intro!: Pi'-mono simp: B'-def)
also note ( $\dots \subseteq O'$ )
finally show  $\exists B' \in basis\text{-}finmap. x \in B' \wedge B' \subseteq O'$  using  $B$ 
  by (auto intro!: bexI[where  $x=B'$ ] Pi'-mono in-basis-finmapI simp: B'-def)
qed

lemma range-enum-basis-finmap-imp-open:
  assumes  $x \in basis\text{-}finmap$ 
  shows  $open x$ 
  using finmap-topological-basis assms by (auto simp: topological-basis-def)

instance proof qed (blast intro: finmap-topological-basis countable-basis-finmap
topological-basis-imp-subbasis)

end

```

24.8 Polish Space of Finite Maps

```

instance fmap :: (countable, polish-space) polish-space proof qed

```

24.9 Product Measurable Space of Finite Maps

```

definition PiF I M  $\equiv$ 

```

```

  sigma ( $\bigcup J \in I. (\Pi' j \in J. space (M j))$ ) { $(\Pi' j \in J. X j) \mid X J. J \in I \wedge X \in (\Pi$ 
 $j \in J. sets (M j))$ }

```

```

abbreviation

```

```

  PiF I M  $\equiv$  PiF I M

```

```

syntax

```

```

  -PiF :: pptrn  $\Rightarrow$  'i set  $\Rightarrow$  'a measure  $\Rightarrow$  ('i  $\Rightarrow$  'a) measure (( $\exists \Pi_F$  - $\in$ -./ -) 10)

```


translations

$\Pi_F x \in I. M == \text{CONST } \text{PiF } I (\%x. M)$

lemma *PiF-gen-subset*: $\{(\Pi' j \in J. X j) \mid X J. J \in I \wedge X \in (\Pi j \in J. \text{sets } (M j))\}$
 \subseteq

$\text{Pow } (\bigcup J \in I. (\Pi' j \in J. \text{space } (M j)))$

by (*auto simp*: *Pi'-def*) (*blast dest*: *sets.sets-into-space*)

lemma *space-PiF*: $\text{space } (\text{PiF } I M) = (\bigcup J \in I. (\Pi' j \in J. \text{space } (M j)))$

unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule space-measure-of*)

lemma *sets-PiF*:

$\text{sets } (\text{PiF } I M) = \text{sigma-sets } (\bigcup J \in I. (\Pi' j \in J. \text{space } (M j)))$

$\{(\Pi' j \in J. X j) \mid X J. J \in I \wedge X \in (\Pi j \in J. \text{sets } (M j))\}$

unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule sets-measure-of*)

lemma *sets-PiF-singleton*:

$\text{sets } (\text{PiF } \{I\} M) = \text{sigma-sets } (\Pi' j \in I. \text{space } (M j))$

$\{(\Pi' j \in I. X j) \mid X. X \in (\Pi j \in I. \text{sets } (M j))\}$

unfolding *sets-PiF* **by** *simp*

lemma *in-sets-PiFI*:

assumes $X = (\text{Pi}' J S) J \in I \wedge i. i \in J \implies S i \in \text{sets } (M i)$

shows $X \in \text{sets } (\text{PiF } I M)$

unfolding *sets-PiF*

using *assms* **by** *blast*

lemma *product-in-sets-PiFI*:

assumes $J \in I \wedge i. i \in J \implies S i \in \text{sets } (M i)$

shows $(\text{Pi}' J S) \in \text{sets } (\text{PiF } I M)$

unfolding *sets-PiF*

using *assms* **by** *blast*

lemma *singleton-space-subset-in-sets*:

fixes J

assumes $J \in I$

assumes *finite* J

shows $\text{space } (\text{PiF } \{J\} M) \in \text{sets } (\text{PiF } I M)$

using *assms*

by (*intro in-sets-PiFI*[**where** $J=J$ **and** $S=\lambda i. \text{space } (M i)$])

(*auto simp*: *product-def space-PiF*)

lemma *singleton-subspace-set-in-sets*:

assumes $A: A \in \text{sets } (\text{PiF } \{J\} M)$

assumes *finite* J

assumes $J \in I$

shows $A \in \text{sets } (\text{PiF } I M)$

using A [*unfolded sets-PiF*]

apply (*induct* A)

unfolding *sets-PiF[symmetric]* **unfolding** *space-PiF[symmetric]*
using *assms*
by (*auto intro: in-sets-PiFI intro!: singleton-space-subset-in-sets*)

lemma *finite-measurable-singletonI*:

assumes *finite I*
assumes $\bigwedge J. J \in I \implies \text{finite } J$
assumes *MN*: $\bigwedge J. J \in I \implies A \in \text{measurable } (PiF \{J\} M) N$
shows $A \in \text{measurable } (PiF I M) N$
unfolding *measurable-def*
proof *safe*
fix *y* **assume** $y \in \text{sets } N$
have $A -' y \cap \text{space } (PiF I M) = (\bigcup J \in I. A -' y \cap \text{space } (PiF \{J\} M))$
by (*auto simp: space-PiF*)
also have $\dots \in \text{sets } (PiF I M)$
proof (*rule sets.finite-UN*)
show *finite I* **by fact**
fix *J* **assume** $J \in I$
with *assms* **have** *finite J* **by simp**
show $A -' y \cap \text{space } (PiF \{J\} M) \in \text{sets } (PiF I M)$
by (*rule singleton-subspace-set-in-sets[OF measurable-sets[OF assms(3)]]*)

fact+

qed

finally show $A -' y \cap \text{space } (PiF I M) \in \text{sets } (PiF I M)$.

next

fix *x* **assume** $x \in \text{space } (PiF I M)$ **thus** $A x \in \text{space } N$
using *MN[of domain x]*
by (*auto simp: space-PiF measurable-space Pi'-def*)

qed

lemma *countable-finite-comprehension*:

fixes $f :: 'a :: \text{countable set} \Rightarrow -$
assumes $\bigwedge s. P s \implies \text{finite } s$
assumes $\bigwedge s. P s \implies f s \in \text{sets } M$
shows $\bigcup \{f s \mid s. P s\} \in \text{sets } M$

proof $-$

have $\bigcup \{f s \mid s. P s\} = (\bigcup n :: \text{nat. let } s = \text{set } (from\text{-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$

proof *safe*

fix $x \in X$ **assume** $*$: $x \in f s P s$

with *assms* **obtain** *l* **where** $s = \text{set } l$ **using** *finite-list* **by blast**

with $*$ **show** $x \in (\bigcup n. \text{let } s = \text{set } (from\text{-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$

using $\langle P s \rangle$

by (*auto intro!: exI[where x=to-nat l]*)

next

fix $x \in X$ **assume** $x \in (\text{let } s = \text{set } (from\text{-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$

thus $x \in \bigcup \{f s \mid s. P s\}$ **using** *assms* **by** (*auto simp: Let-def split: if-split-asm*)

qed

hence $\bigcup \{f s \mid s. P s\} = (\bigcup n. \text{let } s = \text{set } (from\text{-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$

by *simp*
 also have $\dots \in \text{sets } M$ using *assms* by (*auto simp: Let-def*)
 finally show *?thesis* .
 qed

lemma *space-subset-in-sets*:
 fixes $J::'a::\text{countable set set}$
 assumes $J \subseteq I$
 assumes $\bigwedge j. j \in J \implies \text{finite } j$
 shows $\text{space } (PiF J M) \in \text{sets } (PiF I M)$
proof –
 have $\text{space } (PiF J M) = \bigcup \{\text{space } (PiF \{j\} M) \mid j. j \in J\}$
 unfolding *space-PiF* by *blast*
 also have $\dots \in \text{sets } (PiF I M)$ using *assms*
 by (*intro countable-finite-comprehension*) (*auto simp: singleton-space-subset-in-sets*)
 finally show *?thesis* .
 qed

lemma *subspace-set-in-sets*:
 fixes $J::'a::\text{countable set set}$
 assumes $A: A \in \text{sets } (PiF J M)$
 assumes $J \subseteq I$
 assumes $\bigwedge j. j \in J \implies \text{finite } j$
 shows $A \in \text{sets } (PiF I M)$
 using *A[unfolded sets-PiF]*
 apply (*induct A*)
 unfolding *sets-PiF[symmetric]* *space-PiF[symmetric]*
 using *assms*
 by (*auto intro: in-sets-PiFI intro!: space-subset-in-sets*)

lemma *countable-measurable-PiFI*:
 fixes $I::'a::\text{countable set set}$
 assumes $MN: \bigwedge J. J \in I \implies \text{finite } J \implies A \in \text{measurable } (PiF \{J\} M) N$
 shows $A \in \text{measurable } (PiF I M) N$
 unfolding *measurable-def*
proof *safe*
 fix y assume $y \in \text{sets } N$
 have $A -' y = (\bigcup \{A -' y \cap \{x. \text{domain } x = J\} \mid J. \text{finite } J\})$ by *auto*
 { fix $x::'a \Rightarrow_F 'b$
 from *finite-list[of domain x]* **obtain** xs **where** $\text{set } xs = \text{domain } x$ by *auto*
 hence $\exists n. \text{domain } x = \text{set } (\text{from-nat } n)$
 by (*intro exI[where x=to-nat xs]*) *auto* }
 hence $A -' y \cap \text{space } (PiF I M) = (\bigcup n. A -' y \cap \text{space } (PiF (\{\text{set } (\text{from-nat } n)\} \cap I) M))$
 by (*auto simp: space-PiF Pi'-def*)
 also have $\dots \in \text{sets } (PiF I M)$
 apply (*intro sets.Int sets.countable-nat-UN subsetI, safe*)
 apply (*case-tac set (from-nat i) \in I*)
 apply *simp-all*

```

apply (rule singleton-subspace-set-in-sets[OF measurable-sets[OF MN]])
using assms ⟨y ∈ sets N⟩
apply (auto simp: space-PiF)
done
finally show A - ‘ y ∩ space (PiF I M) ∈ sets (PiF I M) .
next
fix x assume x ∈ space (PiF I M) thus A x ∈ space N
using MN[of domain x] by (auto simp: space-PiF measurable-space Pi'-def)
qed

lemma measurable-PiF:
  assumes f:  $\bigwedge x. x \in \text{space } N \implies \text{domain } (f x) \in I \wedge (\forall i \in \text{domain } (f x). (f x) i \in \text{space } (M i))$ 
  assumes S:  $\bigwedge J S. J \in I \implies (\bigwedge i. i \in J \implies S i \in \text{sets } (M i)) \implies$ 
    f - ‘ (Pi' J S) ∩ space N ∈ sets N
  shows f ∈ measurable N (PiF I M)
  unfolding PiF-def
  using PiF-gen-subset
  apply (rule measurable-measure-of)
  using f apply force
  apply (insert S, auto)
  done

lemma restrict-sets-measurable:
  assumes A: A ∈ sets (PiF I M) and J ⊆ I
  shows A ∩ {m. domain m ∈ J} ∈ sets (PiF J M)
  using A[unfolded sets-PiF]
proof (induct A)
  case (Basic a)
  then obtain K S where S: a = Pi' K S K ∈ I (∀ i ∈ K. S i ∈ sets (M i))
    by auto
  show ?case
  proof cases
    assume K ∈ J
    hence a ∩ {m. domain m ∈ J} ∈ {Pi' K X | X K. K ∈ J ∧ X ∈ (Π j ∈ K. sets (M j))} using S
    by (auto intro!: exI[where x=K] exI[where x=S] simp: Pi'-def)
    also have ... ⊆ sets (PiF J M) unfolding sets-PiF by auto
    finally show ?thesis .
  next
    assume K ∉ J
    hence a ∩ {m. domain m ∈ J} = {} using S by (auto simp: Pi'-def)
    also have ... ∈ sets (PiF J M) by simp
    finally show ?thesis .
  qed
next
  case (Union a)
  have UNION UNIV a ∩ {m. domain m ∈ J} = (∪ i. (a i ∩ {m. domain m ∈ J}))

```

```

    by simp
    also have ... ∈ sets (PiF J M) using Union by (intro sets.countable-nat-UN)
  auto
  finally show ?case .
next
  case (Compl a)
  have (space (PiF I M) - a) ∩ {m. domain m ∈ J} = (space (PiF J M) - (a
  ∩ {m. domain m ∈ J}))
    using ⟨J ⊆ I⟩ by (auto simp: space-PiF Pi'-def)
  also have ... ∈ sets (PiF J M) using Compl by auto
  finally show ?case by (simp add: space-PiF)
qed simp

```

lemma measurable-finmap-of:

```

  assumes f: ∧i. (∃ x ∈ space N. i ∈ J x) ⇒ (λx. f x i) ∈ measurable N (M i)
  assumes J: ∧x. x ∈ space N ⇒ J x ∈ I ∧x. x ∈ space N ⇒ finite (J x)
  assumes JN: ∧S. {x. J x = S} ∩ space N ∈ sets N
  shows (λx. finmap-of (J x) (f x)) ∈ measurable N (PiF I M)
proof (rule measurable-PiF)
  fix x assume x ∈ space N
  with J[of x] measurable-space[OF f]
  show domain (finmap-of (J x) (f x)) ∈ I ∧
    (∀ i ∈ domain (finmap-of (J x) (f x)). (finmap-of (J x) (f x)) i ∈ space (M
  i))
    by auto
next
  fix K S assume K ∈ I and *: ∧i. i ∈ K ⇒ S i ∈ sets (M i)
  with J have eq: (λx. finmap-of (J x) (f x)) -‘ Pi' K S ∩ space N =
    (if ∃ x ∈ space N. K = J x ∧ finite K then if K = {} then {x ∈ space N. J x
  = K}
    else (∩ i ∈ K. (λx. f x i) -‘ S i ∩ {x ∈ space N. J x = K}) else {})
    by (auto simp: Pi'-def)
  have r: {x ∈ space N. J x = K} = space N ∩ ({x. J x = K} ∩ space N) by
  auto
  show (λx. finmap-of (J x) (f x)) -‘ Pi' K S ∩ space N ∈ sets N
    unfolding eq r
    apply (simp del: INT-simps add: )
    apply (intro conjI impI sets.finite-INT JN sets.Int[OF sets.top])
    apply simp apply assumption
    apply (subst Int-assoc[symmetric])
    apply (rule sets.Int)
    apply (intro measurable-sets[OF f] *) apply force apply assumption
    apply (intro JN)
  done
qed

```

lemma measurable-PiM-finmap-of:

```

  assumes finite J
  shows finmap-of J ∈ measurable (Pi_M J M) (PiF {J} M)

```

```

apply (rule measurable-finmap-of)
apply (rule measurable-component-singleton)
apply simp
apply rule
apply (rule ⟨finite J⟩)
apply simp
done

```

lemma *proj-measurable-singleton*:

```

assumes  $A \in \text{sets } (M \ i)$ 
shows  $(\lambda x. (x)_F \ i) -' A \cap \text{space } (PiF \ \{I\} \ M) \in \text{sets } (PiF \ \{I\} \ M)$ 
proof cases
  assume  $i \in I$ 
  hence  $(\lambda x. (x)_F \ i) -' A \cap \text{space } (PiF \ \{I\} \ M) =$ 
     $Pi' \ I \ (\lambda x. \text{if } x = i \text{ then } A \text{ else } \text{space } (M \ x))$ 
    using sets.sets-into-space[OF ]  $\langle A \in \text{sets } (M \ i) \rangle$  assms
    by (auto simp: space-PiF Pi'-def)
  thus ?thesis using assms  $\langle A \in \text{sets } (M \ i) \rangle$ 
    by (intro in-sets-PiFI auto)
next
  assume  $i \notin I$ 
  hence  $(\lambda x. (x)_F \ i) -' A \cap \text{space } (PiF \ \{I\} \ M) =$ 
     $(\text{if } \text{undefined} \in A \text{ then } \text{space } (PiF \ \{I\} \ M) \text{ else } \{\})$  by (auto simp: space-PiF
Pi'-def)
  thus ?thesis by simp
qed

```

lemma *measurable-proj-singleton*:

```

assumes  $i \in I$ 
shows  $(\lambda x. (x)_F \ i) \in \text{measurable } (PiF \ \{I\} \ M) \ (M \ i)$ 
by (unfold measurable-def, intro CollectI conjI ballI proj-measurable-singleton
assms)
  (insert  $\langle i \in I \rangle$ , auto simp: space-PiF)

```

lemma *measurable-proj-countable*:

```

fixes  $I::'a::\text{countable set set}$ 
assumes  $y \in \text{space } (M \ i)$ 
shows  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } (x)_F \ i \text{ else } y) \in \text{measurable } (PiF \ I \ M) \ (M \ i)$ 
proof (rule countable-measurable-PiFI)
  fix  $J$  assume  $J \in I$  finite J
  show  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) \in \text{measurable } (PiF \ \{J\} \ M) \ (M \ i)$ 
    unfolding measurable-def
proof safe
  fix  $z$  assume  $z \in \text{sets } (M \ i)$ 
  have  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) -' z \cap \text{space } (PiF \ \{J\} \ M) =$ 
     $(\lambda x. \text{if } i \in J \text{ then } (x)_F \ i \text{ else } y) -' z \cap \text{space } (PiF \ \{J\} \ M)$ 
    by (auto simp: space-PiF Pi'-def)
  also have  $\dots \in \text{sets } (PiF \ \{J\} \ M)$  using  $\langle z \in \text{sets } (M \ i) \rangle$  ⟨finite J⟩
    by (cases  $i \in J$ ) (auto intro!: measurable-sets[OF measurable-proj-singleton])

```

finally show $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \ \text{else } y) - ' z \cap \text{space } (PiF \ \{J\} \ M)$
 \in
 $\text{sets } (PiF \ \{J\} \ M) .$
qed $(\text{insert } (y \in \text{space } (M \ i)), \ \text{auto simp: space-PiF Pi'-def})$
qed

lemma measurable-restrict-proj:

assumes $J \in II \ \text{finite } J$
shows $\text{finmap-of } J \in \text{measurable } (PiM \ J \ M) \ (PiF \ II \ M)$
using assms
by $(\text{intro measurable-finmap-of measurable-component-singleton}) \ \text{auto}$

lemma measurable-proj-PiM:

fixes $J \ K \ :: 'a :: \text{countable set}$ **and** $I :: 'a \ \text{set set}$
assumes $\text{finite } J \ J \in I$
assumes $x \in \text{space } (PiM \ J \ M)$
shows $\text{proj} \in \text{measurable } (PiF \ \{J\} \ M) \ (PiM \ J \ M)$
proof $(\text{rule measurable-PiM-single})$
show $\text{proj} \in \text{space } (PiF \ \{J\} \ M) \rightarrow (\Pi_E \ i \in J. \ \text{space } (M \ i))$
using $\text{assms by } (\text{auto simp add: space-PiM space-PiF extensional-def sets-PiF Pi'-def})$
next
fix $A \ i$ **assume** $A: i \in J \ A \in \text{sets } (M \ i)$
show $\{\omega \in \text{space } (PiF \ \{J\} \ M). (\omega)_F \ i \in A\} \in \text{sets } (PiF \ \{J\} \ M)$
proof
have $\{\omega \in \text{space } (PiF \ \{J\} \ M). (\omega)_F \ i \in A\} =$
 $(\lambda \omega. (\omega)_F \ i) - ' A \cap \text{space } (PiF \ \{J\} \ M) \ \text{by } \text{auto}$
also have $\dots \in \text{sets } (PiF \ \{J\} \ M)$
using $\text{assms } A \ \text{by } (\text{auto intro: measurable-sets[OF measurable-proj-singleton]})$
 simp: space-PiM
finally show $?thesis .$
qed simp
qed

lemma space-PiF-singleton-eq-product:

assumes $\text{finite } I$
shows $\text{space } (PiF \ \{I\} \ M) = (\Pi' \ i \in I. \ \text{space } (M \ i))$
by $(\text{auto simp: product-def space-PiF assms})$

adapted from $\text{sets } (PiM \ ?I \ ?M) = \text{sigma-sets } (\Pi_E \ i \in ?I. \ \text{space } (?M \ i)) \ \{\{f \in \Pi_E \ i \in ?I. \ \text{space } (?M \ i). \ f \ i \in A\} \mid i \ A. \ i \in ?I \wedge A \in \text{sets } (?M \ i)\}$

lemma sets-PiF-single:

assumes $\text{finite } I \ I \neq \{\}$
shows $\text{sets } (PiF \ \{I\} \ M) =$
 $\text{sigma-sets } (\Pi' \ i \in I. \ \text{space } (M \ i))$
 $\{\{f \in \Pi' \ i \in I. \ \text{space } (M \ i). \ f \ i \in A\} \mid i \ A. \ i \in I \wedge A \in \text{sets } (M \ i)\}$
(is $- = \text{sigma-sets } ?\Omega \ ?R)$
unfolding $\text{sets-PiF-singleton}$
proof $(\text{rule sigma-sets-eqI})$

interpret R : *sigma-algebra* ? Ω *sigma-sets* ? Ω ? R **by** (*rule sigma-algebra-sigma-sets*)
auto
fix A **assume** $A \in \{Pi' I X \mid X. X \in (\Pi j \in I. sets (M j))\}$
then obtain X **where** $X: A = Pi' I X X \in (\Pi j \in I. sets (M j))$ **by** *auto*
show $A \in sigma-sets ?\Omega ?R$
proof –
from $\langle I \neq \{\} \rangle X$ **have** $A = (\bigcap j \in I. \{f \in space (PiF \{I\} M). f j \in X j\})$
using *sets.sets-into-space*
by (*auto simp: space-PiF product-def*) *blast*
also have $\dots \in sigma-sets ?\Omega ?R$
using $X \langle I \neq \{\} \rangle$ *assms* **by** (*intro R.finite-INT*) (*auto simp: space-PiF*)
finally show $A \in sigma-sets ?\Omega ?R$.
qed
next
fix A **assume** $A \in ?R$
then obtain $i B$ **where** $A: A = \{f \in \Pi' i \in I. space (M i). f i \in B\} i \in I B \in sets (M i)$
by *auto*
then have $A = (\Pi' j \in I. if j = i then B else space (M j))$
using *sets.sets-into-space[OF A(3)]*
apply (*auto simp: Pi'-iff split: if-split-asm*)
apply *blast*
done
also have $\dots \in sigma-sets ?\Omega \{Pi' I X \mid X. X \in (\Pi j \in I. sets (M j))\}$
using A
by (*intro sigma-sets.Basic*)
*(auto intro: exI[**where** $x = \lambda j. if j = i then B else space (M j)$])*
finally show $A \in sigma-sets ?\Omega \{Pi' I X \mid X. X \in (\Pi j \in I. sets (M j))\}$.
qed

adapted from $(\bigwedge i. i \in ?I \implies ?A i = ?B i) \implies Pi_E ?I ?A = Pi_E ?I ?B$

lemma *Pi'-cong*:

assumes *finite I*

assumes $\bigwedge i. i \in I \implies f i = g i$

shows $Pi' I f = Pi' I g$

using *assms* **by** (*auto simp: Pi'-def*)

adapted from $\llbracket finite ?I; \bigwedge i n m. \llbracket i \in ?I; n \leq m \rrbracket \implies ?A n i \subseteq ?A m i \rrbracket$
 $\implies (\bigcup_n Pi ?I (?A n)) = (\Pi i \in ?I. \bigcup_n ?A n i)$

lemma *Pi'-UN*:

fixes $A :: nat \Rightarrow 'i \Rightarrow 'a set$

assumes *finite I*

assumes *mono*: $\bigwedge i n m. i \in I \implies n \leq m \implies A n i \subseteq A m i$

shows $(\bigcup_n Pi' I (A n)) = Pi' I (\lambda i. \bigcup_n. A n i)$

proof (*intro set-eqI iffI*)

fix f **assume** $f \in Pi' I (\lambda i. \bigcup_n. A n i)$

then have $\forall i \in I. \exists n. f i \in A n i$ *domain* $f = I$ **by** (*auto simp: (finite I) Pi'-def*)

from *bchoice[OF this(1)]* **obtain** n **where** $n: \bigwedge i. i \in I \implies f i \in (A (n i) i)$

by *auto*

obtain k **where** $k: \bigwedge i. i \in I \implies n\ i \leq k$
using $\langle \text{finite } I \rangle$ *finite-nat-set-iff-bounded-le*[of $n'I$] **by** *auto*
have $f \in \text{Pi}'\ I\ (\lambda i. A\ k\ i)$
proof
fix i **assume** $i \in I$
from *mono*[*OF this*, of $n\ i\ k$] k [*OF this*] n [*OF this*] $\langle \text{domain } f = I \rangle\ \langle i \in I \rangle$
show $f\ i \in A\ k\ i$ **by** (*auto simp*: $\langle \text{finite } I \rangle$)
qed (*simp add*: $\langle \text{domain } f = I \rangle\ \langle \text{finite } I \rangle$)
then show $f \in (\bigcup n. \text{Pi}'\ I\ (A\ n))$ **by** *auto*
qed (*auto simp*: $\text{Pi}'\text{-def}$ $\langle \text{finite } I \rangle$)

adapted from $\llbracket \bigwedge i. i \in ?I \implies \exists S \subseteq ?E\ i. \text{countable } S \wedge ?\Omega\ i = \bigcup S; \bigwedge i. i \in ?I \implies ?E\ i \subseteq \text{Pow}\ (?\Omega\ i); \bigwedge j. j \in ?J \implies \text{finite } j; \bigcup ?J = ?I \rrbracket \implies$
 $\text{sets}\ (\text{Pi}_M\ ?I\ (\lambda i. \text{sigma}\ (?\Omega\ i)\ (?\E\ i))) = \text{sets}\ (\text{sigma}\ (\text{Pi}_E\ ?I\ ?\Omega)\ \{\{f \in \text{Pi}_E\ ?I\ ?\Omega. \forall i \in j. f\ i \in A\ i\} \mid A\ j. j \in ?J \wedge A \in \text{Pi}\ j\ ?E\})$

lemma *sigma-fprod-algebra-sigma-eq*:

fixes $E :: 'i \Rightarrow 'a\ \text{set}\ \text{set}$ **and** $S :: 'i \Rightarrow \text{nat} \Rightarrow 'a\ \text{set}$
assumes [*simp*]: *finite* $I\ I \neq \{\}$
and *S-union*: $\bigwedge i. i \in I \implies (\bigcup j. S\ i\ j) = \text{space}\ (M\ i)$
and *S-in-E*: $\bigwedge i. i \in I \implies \text{range}\ (S\ i) \subseteq E\ i$
assumes *E-closed*: $\bigwedge i. i \in I \implies E\ i \subseteq \text{Pow}\ (\text{space}\ (M\ i))$
and *E-generates*: $\bigwedge i. i \in I \implies \text{sets}\ (M\ i) = \text{sigma-sets}\ (\text{space}\ (M\ i))\ (E\ i)$
defines $P == \{ \text{Pi}'\ I\ F \mid F. \forall i \in I. F\ i \in E\ i \}$
shows $\text{sets}\ (\text{Pi}F\ \{I\}\ M) = \text{sigma-sets}\ (\text{space}\ (\text{Pi}F\ \{I\}\ M))\ P$

proof

let $?P = \text{sigma}\ (\text{space}\ (\text{Pi}_F\ \{I\}\ M))\ P$
from $\langle \text{finite } I \rangle$ [*THEN ex-bij-betw-finite-nat*] **guess** $T ..$
then have $T: \bigwedge i. i \in I \implies T\ i < \text{card } I \wedge i. i \in I \implies \text{the-inv-into } I\ T\ (T\ i)$
 $= i$
by (*auto simp add*: *bij-betw-def set-eq-iff image-iff the-inv-into-f-f simp del*: $\langle \text{finite } I \rangle$)
have *P-closed*: $P \subseteq \text{Pow}\ (\text{space}\ (\text{Pi}_F\ \{I\}\ M))$
using *E-closed* **by** (*auto simp*: *space-PiF P-def Pi'-iff subset-eq*)
then have *space-P*: $\text{space}\ ?P = (\prod' i \in I. \text{space}\ (M\ i))$
by (*simp add*: *space-PiF*)
have $\text{sets}\ (\text{Pi}F\ \{I\}\ M) =$
 $\text{sigma-sets}\ (\text{space}\ ?P)\ \{\{f \in \prod' i \in I. \text{space}\ (M\ i). f\ i \in A\} \mid i\ A. i \in I \wedge A \in \text{sets}\ (M\ i)\}$
using *sets-PiF-single*[of $I\ M$] **by** (*simp add*: *space-P*)
also have $\dots \subseteq \text{sets}\ (\text{sigma}\ (\text{space}\ (\text{Pi}_F\ \{I\}\ M))\ P)$
proof (*safe intro!*: *sets.sigma-sets-subset*)
fix $i\ A$ **assume** $i \in I$ **and** $A: A \in \text{sets}\ (M\ i)$
have $(\lambda x. (x)_F\ i) \in \text{measurable}\ ?P\ (\text{sigma}\ (\text{space}\ (M\ i))\ (E\ i))$
proof (*subst measurable-iff-measure-of*)
show $E\ i \subseteq \text{Pow}\ (\text{space}\ (M\ i))$ **using** $\langle i \in I \rangle$ **by** *fact*
from *space-P* $\langle i \in I \rangle$ **show** $(\lambda x. (x)_F\ i) \in \text{space}\ ?P \rightarrow \text{space}\ (M\ i)$
by *auto*
show $\forall A \in E\ i. (\lambda x. (x)_F\ i) - 'A \cap \text{space}\ ?P \in \text{sets}\ ?P$
proof

```

fix A assume A: A ∈ E i
then have (λx. (x)F i) -‘ A ∩ space ?P = (Π' j∈I. if i = j then A else
space (M j))
using E-closed ⟨i ∈ I⟩ by (auto simp: space-P Pi-iff subset-eq split:
if-split-asm)
also have ... = (Π' j∈I. ∪ n. if i = j then A else S j n)
by (intro Pi'-cong) (simp-all add: S-union)
also have ... = (∪ xs∈{xs. length xs = card I}. Π' j∈I. if i = j then A
else S j (xs ! T j))
using T
apply (auto simp del: Union-iff)
apply (simp-all add: Pi'-iff bchoice-iff del: Union-iff)
apply (erule conjE exE)+
apply (rule-tac x=map (λn. f (the-inv-into I T n)) [0.. $\text{card } I$ ] in exI)
apply (auto simp: bij-betw-def)
done
also have ... ∈ sets ?P
proof (safe intro!: sets.countable-UN)
fix xs show (Π' j∈I. if i = j then A else S j (xs ! T j)) ∈ sets ?P
using A S-in-E
by (simp add: P-closed)
(auto simp: P-def subset-eq intro!: exI[of - λj. if i = j then A else S j
(xs ! T j)])
qed
finally show (λx. (x)F i) -‘ A ∩ space ?P ∈ sets ?P
using P-closed by simp
qed
qed
from measurable-sets[OF this, of A] A ⟨i ∈ I⟩ E-closed
have (λx. (x)F i) -‘ A ∩ space ?P ∈ sets ?P
by (simp add: E-generates)
also have (λx. (x)F i) -‘ A ∩ space ?P = {f ∈ Π' i∈I. space (M i). f i ∈ A}
using P-closed by (auto simp: space-PiF)
finally show ... ∈ sets ?P .
qed
finally show sets (PiF {I} M) ⊆ sigma-sets (space (PiF {I} M)) P
by (simp add: P-closed)
show sigma-sets (space (PiF {I} M)) P ⊆ sets (PiF {I} M)
using ⟨finite I⟩ ⟨I ≠ {}⟩
by (auto intro!: sets.sigma-sets-subset product-in-sets-PiFI simp: E-generates
P-def)
qed

```

lemma product-open-generates-sets-PiF-single:

```

assumes I ≠ {}
assumes [simp]: finite I
shows sets (PiF {I} (λ-. borel::'b::second-countable-topology measure)) =
sigma-sets (space (PiF {I} (λ-. borel))) {Pi' I F |F. (∀ i∈I. F i ∈ Collect
open)}

```

```

proof –
  from open-countable-basisE[OF open-UNIV] guess S::'b set set . note S = this
  show ?thesis
  proof (rule sigma-fprod-algebra-sigma-eq)
    show finite I by simp
    show I ≠ {} by fact
    define S' where S' = from-nat-into S
    show (∪j. S' j) = space borel
      using S
    apply (auto simp add: from-nat-into countable-basis-proj S'-def basis-proj-def)
    apply (metis (lifting, mono-tags) UNIV-I UnionE basis-proj-def countable-basis-proj
countable-subset from-nat-into-surj)
    done
    show range S' ⊆ Collect open
      using S
    apply (auto simp add: from-nat-into countable-basis-proj S'-def)
    apply (metis UNIV-not-empty Union-empty from-nat-into set-mp topological-basis-open[OF
basis-proj] basis-proj-def)
    done
    show Collect open ⊆ Pow (space borel) by simp
    show sets borel = sigma-sets (space borel) (Collect open)
      by (simp add: borel-def)
  qed
qed

```

```

lemma finmap-UNIV[simp]: (∪ J∈Collect finite. Π' j∈J. UNIV) = UNIV by
auto

```

```

lemma borel-eq-PiF-borel:

```

```

  shows (borel :: ('i::countable ⇒F 'a::polish-space) measure) =
    PiF (Collect finite) (λ-. borel :: 'a measure)
  unfolding borel-def PiF-def
proof (rule measure-eqI, clarsimp, rule sigma-sets-eqI)
  fix a::('i ⇒F 'a) set assume a ∈ Collect open hence open a by simp
  then obtain B' where B': B' ⊆ basis-finmap a = ∪ B'
    using finmap-topological-basis by (force simp add: topological-basis-def)
  have a ∈ sigma UNIV {Pi' J X |X J. finite J ∧ X ∈ J → sigma-sets UNIV
(Collect open)}
    unfolding ⟨a = ∪ B'⟩
  proof (rule sets.countable-Union)
    from B' countable-basis-finmap show countable B' by (metis countable-subset)
  next
    show B' ⊆ sets (sigma UNIV
{Pi' J X |X J. finite J ∧ X ∈ J → sigma-sets UNIV (Collect open)}) (is -
⊆ sets ?s)
  proof
    fix x assume x ∈ B' with B' have x ∈ basis-finmap by auto
    then obtain J X where x = Pi' J X finite J X ∈ J → sigma-sets UNIV
(Collect open)

```

```

    by (auto simp: basis-finmap-def topological-basis-open[OF basis-proj])
  thus  $x \in \text{sets } ?s$  by auto
qed
qed
thus  $a \in \text{sigma-sets UNIV } \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV } (\text{Collect open})\}$ 
  by simp
next
fix  $b::('i \Rightarrow_F 'a)$  set
  assume  $b \in \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV } (\text{Collect open})\}$ 
  hence  $b': b \in \text{sets } (Pi_F (\text{Collect finite}) (\lambda-. \text{borel}))$  by (auto simp: sets-PiF borel-def)
  let  $?b = \lambda J. b \cap \{x. \text{domain } x = J\}$ 
  have  $b = \bigcup ((\lambda J. ?b J) ' \text{Collect finite})$  by auto
  also have  $\dots \in \text{sets borel}$ 
  proof (rule sets.countable-Union, safe)
    fix  $J::'i$  set assume finite J
    { assume  $ef: J = \{\}$ 
      have  $?b J \in \text{sets borel}$ 
    }
    proof cases
      assume  $?b J \neq \{\}$ 
      then obtain  $f$  where  $f \in b$  domain  $f = \{\}$  using  $ef$  by auto
      hence  $?b J = \{f\}$  using  $\langle J = \{\} \rangle$ 
        by (auto simp: finmap-eq-iff)
      also have  $\{f\} \in \text{sets borel}$  by simp
      finally show  $?thesis$  .
    qed simp
  } moreover {
    assume  $J \neq (\{\}::'i \text{ set})$ 
    have  $(?b J) = b \cap \{m. \text{domain } m \in \{J\}\}$  by auto
    also have  $\dots \in \text{sets } (PiF \{J\}) (\lambda-. \text{borel})$ 
      using  $b'$  by (rule restrict-sets-measurable) (auto simp:  $\langle \text{finite } J \rangle$ )
    also have  $\dots = \text{sigma-sets } (\text{space } (PiF \{J\}) (\lambda-. \text{borel}))$ 
       $\{Pi' (J) F \mid F. (\forall j \in J. F j \in \text{Collect open})\}$ 
      (is - = sigma-sets - ?P)
    by (rule product-open-generates-sets-PiF-single[OF  $\langle J \neq \{\} \rangle \langle \text{finite } J \rangle$ ])
    also have  $\dots \subseteq \text{sigma-sets UNIV } (\text{Collect open})$ 
      by (intro sigma-sets-mono'') (auto intro!: open-Pi'I simp: space-PiF)
    finally have  $(?b J) \in \text{sets borel}$  by (simp add: borel-def)
  } ultimately show  $(?b J) \in \text{sets borel}$  by blast
qed (simp add: countable-Collect-finite)
finally show  $b \in \text{sigma-sets UNIV } (\text{Collect open})$  by (simp add: borel-def)
qed (simp add: emeasure-sigma borel-def PiF-def)

```

24.10 Isomorphism between Functions and Finite Maps

lemma *measurable-finmap-compose*:

shows $(\lambda m. \text{compose } J m f) \in \text{measurable } (PiM (f ' J) (\lambda-. M)) (PiM J (\lambda-. M))$

M)

unfolding *compose-def by measurable*

lemma *measurable-compose-inv*:

assumes *inj*: $\bigwedge j. j \in J \implies f' (f j) = j$

shows $(\lambda m. \text{compose } (f' J) m f') \in \text{measurable } (PiM J (\lambda-. M)) (PiM (f' J) (\lambda-. M))$

unfolding *compose-def by (rule measurable-restrict) (auto simp: inj)*

locale *function-to-finmap* =

fixes $J::'a \text{ set}$ **and** $f :: 'a \Rightarrow 'b::\text{countable}$ **and** f'

assumes [*simp*]: *finite J*

assumes *inv*: $i \in J \implies f' (f i) = i$

begin

to measure finmaps

definition $fm = (\text{finmap-of } (f' J)) \circ (\lambda g. \text{compose } (f' J) g f')$

lemma *domain-fm[simp]*: $\text{domain } (fm x) = f' J$

unfolding *fm-def by simp*

lemma *fm-restrict[simp]*: $fm (\text{restrict } y J) = fm y$

unfolding *fm-def by (auto simp: compose-def inv intro: restrict-ext)*

lemma *fm-product*:

assumes $\bigwedge i. \text{space } (M i) = UNIV$

shows $fm -' Pi' (f' J) S \cap \text{space } (PiM J M) = (PiE j \in J. S (f j))$

using *assms*

by (*auto simp: inv fm-def compose-def space-PiM Pi'-def*)

lemma *fm-measurable*:

assumes $f' J \in N$

shows $fm \in \text{measurable } (PiM J (\lambda-. M)) (PiF N (\lambda-. M))$

unfolding *fm-def*

proof (*rule measurable-comp, rule measurable-compose-inv*)

show $\text{finmap-of } (f' J) \in \text{measurable } (PiM (f' J) (\lambda-. M)) (PiF N (\lambda-. M))$

using *assms by (intro measurable-finmap-of measurable-component-singleton)*

auto

qed (*simp-all add: inv*)

lemma *proj-fm*:

assumes $x \in J$

shows $fm m (f x) = m x$

using *assms by (auto simp: fm-def compose-def o-def inv)*

lemma *inj-on-compose-f'*: *inj-on* $(\lambda g. \text{compose } (f' J) g f')$ (*extensional J*)

proof (*rule inj-on-inverseI*)

fix $x::'a \Rightarrow 'c$ **assume** $x \in \text{extensional } J$

thus $(\lambda x. \text{compose } J x f) (\text{compose } (f' J) x f') = x$

by (auto simp: compose-def inv extensional-def)
qed

lemma inj-on-fm:

assumes $\bigwedge i. \text{space } (M \ i) = \text{UNIV}$
shows inj-on fm (space (Pi_M J M))
using assms
apply (auto simp: fm-def space-PiM PiE-def)
apply (rule comp-inj-on)
apply (rule inj-on-compose-f')
apply (rule finmap-of-inj-on-extensional-finite)
apply simp
apply (auto)
done

to measure functions

definition mf = (λg. compose J g f) o proj

lemma mf-fm:

assumes $x \in \text{space } (Pi_M \ J \ (\lambda-. \ M))$
shows mf (fm x) = x
proof –
have mf (fm x) ∈ extensional J
by (auto simp: mf-def extensional-def compose-def)
moreover
have $x \in \text{extensional } J$ using assms sets.sets-into-space
by (force simp: space-PiM PiE-def)
moreover
{ fix i assume $i \in J$
hence mf (fm x) i = x i
by (auto simp: inv mf-def compose-def fm-def)
}
ultimately
show ?thesis by (rule extensionalityI)
qed

lemma mf-measurable:

assumes $\text{space } M = \text{UNIV}$
shows mf ∈ measurable (PiF {f ‘ J} (λ-. M)) (PiM J (λ-. M))
unfolding mf-def
proof (rule measurable-comp, rule measurable-proj-PiM)
show (λg. compose J g f) ∈ measurable (PiM (f ‘ J) (λx. M)) (PiM J (λ-. M))
by (rule measurable-finmap-compose)
qed (auto simp add: space-PiM extensional-def assms)

lemma fm-image-measurable:

assumes $\text{space } M = \text{UNIV}$
assumes $X \in \text{sets } (Pi_M \ J \ (\lambda-. \ M))$
shows fm ‘ X ∈ sets (PiF {f ‘ J} (λ-. M))

proof –
have $fm \text{ ' } X = (mf) \text{ - ' } X \cap \text{space } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$
proof safe
fix x **assume** $x \in X$
with $mf\text{-}fm[of \ x] \text{ sets.sets-into-space}[OF \ \text{assms}(2)]$ **show** $fm \ x \in mf \text{ - ' } X$ **by**
auto
show $fm \ x \in \text{space } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$ **by** (*simp add: space-PiF assms*)
next
fix $y \ x$
assume $x: mf \ y \in X$
assume $y: y \in \text{space } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$
thus $y \in fm \text{ ' } X$
by (*intro image-eqI[OF - x], unfold finmap-eq-iff*)
(auto simp: space-PiF fm-def mf-def compose-def inv Pi'-def)
qed
also have $\dots \in \text{sets } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$
using *assms*
by (*intro measurable-sets[OF mf-measurable]*) *auto*
finally show *?thesis .*
qed

lemma *fm-image-measurable-finite*:
assumes $\text{space } M = UNIV$
assumes $X \in \text{sets } (Pi_M \ J \ (\lambda\text{-. } M::'c \ \text{measure}))$
shows $fm \text{ ' } X \in \text{sets } (PiF \ (\text{Collect } \text{finite}) \ (\lambda\text{-. } M::'c \ \text{measure}))$
using *fm-image-measurable[OF assms]*
by (*rule subspace-set-in-sets*) (*auto simp: finite-subset*)

measure on finmaps

definition $\text{mapmeasure } M \ N = \text{distr } M \ (PiF \ (\text{Collect } \text{finite}) \ N) \ (fm)$

lemma *sets-mapmeasure[simp]*: $\text{sets } (\text{mapmeasure } M \ N) = \text{sets } (PiF \ (\text{Collect } \text{finite}) \ N)$
unfolding *mapmeasure-def* **by** *simp*

lemma *space-mapmeasure[simp]*: $\text{space } (\text{mapmeasure } M \ N) = \text{space } (PiF \ (\text{Collect } \text{finite}) \ N)$
unfolding *mapmeasure-def* **by** *simp*

lemma *mapmeasure-PiF*:
assumes $s1: \text{space } M = \text{space } (Pi_M \ J \ (\lambda\text{-. } N))$
assumes $s2: \text{sets } M = \text{sets } (Pi_M \ J \ (\lambda\text{-. } N))$
assumes $\text{space } N = UNIV$
assumes $X \in \text{sets } (PiF \ (\text{Collect } \text{finite}) \ (\lambda\text{-. } N))$
shows $\text{emeasure } (\text{mapmeasure } M \ (\lambda\text{-. } N)) \ X = \text{emeasure } M \ ((fm \text{ - ' } X \cap \text{extensional } J))$
using *assms*
by (*auto simp: measurable-cong-sets[OF s2 refl] mapmeasure-def emeasure-distr fm-measurable space-PiM PiE-def*)

```

lemma mapmeasure-PiM:
  fixes  $N::'c$  measure
  assumes  $s1$ : space  $M = \text{space } (Pi_M J (\lambda-. N))$ 
  assumes  $s2$ : sets  $M = (Pi_M J (\lambda-. N))$ 
  assumes  $N$ : space  $N = UNIV$ 
  assumes  $X$ :  $X \in \text{sets } M$ 
  shows  $\text{emeasure } M X = \text{emeasure } (\text{mapmeasure } M (\lambda-. N)) (fm \text{ ` } X)$ 
  unfolding mapmeasure-def
proof (subst emeasure-distr, subst measurable-cong-sets[OF s2 refl], rule fm-measurable)
  have  $X \subseteq \text{space } (Pi_M J (\lambda-. N))$  using assms by (simp add: sets.sets-into-space)
  from assms inj-on-fm[of  $\lambda-. N$ ] set-mp[OF this] have  $fm \text{ -' } fm \text{ ` } X \cap \text{space}$ 
  ( $Pi_M J (\lambda-. N) = X$ )
    by (auto simp: vimage-image-eq inj-on-def)
  thus  $\text{emeasure } M X = \text{emeasure } M (fm \text{ -' } fm \text{ ` } X \cap \text{space } M)$  using  $s1$ 
    by simp
  show  $fm \text{ ` } X \in \text{sets } (PiF (Collect finite) (\lambda-. N))$ 
    by (rule fm-image-measurable-finite[OF N X[simplified s2]])
qed simp

end

end

```

25 Projective Limit

```

theory Projective-Limit
imports
  Fin-Map
  Infinite-Product-Measure
  HOL-Library.Diagonal-Subsequence
begin

```

25.1 Sequences of Finite Maps in Compact Sets

```

locale finmap-seqs-into-compact =
  fixes  $K::nat \Rightarrow (nat \Rightarrow_F 'a::\text{metric-space}) \text{ set}$  and  $f::nat \Rightarrow (nat \Rightarrow_F 'a)$  and
   $M$ 
  assumes compact:  $\bigwedge n. \text{compact } (K n)$ 
  assumes f-in-K:  $\bigwedge n. K n \neq \{\}$ 
  assumes domain-K:  $\bigwedge n. k \in K n \implies \text{domain } k = \text{domain } (f n)$ 
  assumes proj-in-K:
     $\bigwedge t n m. m \geq n \implies t \in \text{domain } (f n) \implies (f m)_F t \in (\lambda k. (k)_F t) \text{ ` } K n$ 
begin

lemma proj-in-K':  $(\exists n. \forall m \geq n. (f m)_F t \in (\lambda k. (k)_F t) \text{ ` } K n)$ 
  using proj-in-K f-in-K
proof cases
  obtain  $k$  where  $k \in K (Suc 0)$  using f-in-K by auto

```


assume $\forall n. t \notin \text{domain } (f\ n)$
thus *?thesis*
by (*auto intro!*: *exI*[**where** $x=1$] *image-eqI*[*OF* - $\langle k \in K\ (Suc\ 0) \rangle$]
simp: *domain-K*[*OF* $\langle k \in K\ (Suc\ 0) \rangle$])
qed *blast*

lemma *proj-in-KE*:
obtains n **where** $\bigwedge m. m \geq n \implies (f\ m)_F\ t \in (\lambda k. (k)_F\ t) \text{ ' } K\ n$
using *proj-in-K'* **by** *blast*

lemma *compact-projset*:
shows *compact* $((\lambda k. (k)_F\ i) \text{ ' } K\ n)$
using *continuous-proj compact* **by** (*rule compact-continuous-image*)

end

lemma *compactE'*:
fixes $S :: 'a :: \text{metric-space set}$
assumes *compact* $S\ \forall n \geq m. f\ n \in S$
obtains $l\ r$ **where** $l \in S$ *strict-mono* $(r :: \text{nat} \Rightarrow \text{nat})\ ((f \circ r) \longrightarrow l)$ *sequentially*
proof *atomize-elim*
have *strict-mono* $(op + m)$ **by** (*simp add: strict-mono-def*)
have $\forall n. (f \circ (\lambda i. m + i))\ n \in S$ **using** *assms* **by** *auto*
from *seq-compactE*[*OF* $\langle \text{compact } S \rangle$][*unfolded compact-eq-seq-compact-metric*] *this*
guess $l\ r$.
hence $l \in S$ *strict-mono* $((\lambda i. m + i) \circ r) \wedge (f \circ ((\lambda i. m + i) \circ r)) \longrightarrow l$
using *strict-mono-o*[*OF* $\langle \text{strict-mono } (op + m) \rangle \langle \text{strict-mono } r \rangle$] **by** (*auto simp:*
o-def)
thus $\exists l\ r. l \in S \wedge \text{strict-mono } r \wedge (f \circ r) \longrightarrow l$ **by** *blast*
qed

sublocale *finmap-seqs-into-compact* \subseteq *subseqs* $\lambda n\ s. (\exists l. (\lambda i. ((f \circ s)\ i)_F\ n) \longrightarrow l)$

proof

fix n **and** $s :: \text{nat} \Rightarrow \text{nat}$
assume *strict-mono* s
from *proj-in-KE*[*of* n] **guess** $n0$. **note** $n0 = \text{this}$
have $\forall i \geq n0. ((f \circ s)\ i)_F\ n \in (\lambda k. (k)_F\ n) \text{ ' } K\ n0$
proof *safe*
fix i **assume** $n0 \leq i$
also **have** $\dots \leq s\ i$ **by** (*rule seq-suble*) *fact*
finally **have** $n0 \leq s\ i$.
with $n0$ **show** $((f \circ s)\ i)_F\ n \in (\lambda k. (k)_F\ n) \text{ ' } K\ n0$
by *auto*
qed
from *compactE'*[*OF* *compact-projset* *this*] **guess** $l\ r\ s$.
thus $\exists r'. \text{strict-mono } r' \wedge (\exists l. (\lambda i. ((f \circ (s \circ r'))\ i)_F\ n) \longrightarrow l)$ **by** (*auto*
simp: o-def)
qed

lemma (in *finmap-seqs-into-compact*) *diagonal-tendsto*: $\exists l. (\lambda i. (f (diagseq i))_F n) \longrightarrow l$

proof –

obtain l **where** $(\lambda i. ((f \circ (diagseq \circ op + (Suc n))) i)_F n) \longrightarrow l$

proof (*atomize-elim*, *rule diagseq-holds*)

fix $r s n$

assume *strict-mono* ($r :: nat \Rightarrow nat$)

assume $\exists l. (\lambda i. ((f \circ s) i)_F n) \longrightarrow l$

then obtain l **where** $((\lambda i. (f i)_F n) \circ s) \longrightarrow l$

by (*auto simp: o-def*)

hence $((\lambda i. (f i)_F n) \circ s \circ r) \longrightarrow l$ **using** (*strict-mono r*)

by (*rule LIMSEQ-subseq-LIMSEQ*)

thus $\exists l. (\lambda i. ((f \circ (s \circ r)) i)_F n) \longrightarrow l$ **by** (*auto simp add: o-def*)

qed

hence $(\lambda i. ((f (diagseq (i + Suc n))))_F n) \longrightarrow l$ **by** (*simp add: ac-simps*)

hence $(\lambda i. (f (diagseq i))_F n) \longrightarrow l$ **by** (*rule LIMSEQ-offset*)

thus *?thesis* ..

qed

25.2 Daniell-Kolmogorov Theorem

Existence of Projective Limit

locale *polish-projective* = *projective-family I P* λ -. *borel::'a::polish-space measure*
for $I::'i$ *set* **and** P
begin

lemma *emeasure-lim-emb*:

assumes $X: J \subseteq I$ *finite* $J X \in$ *sets* $(\Pi_M i \in J. \text{borel})$

shows $\lim (emb I J X) = P J X$

proof (*rule emeasure-lim*)

write μ - G (μG)

interpret *generator: algebra space* $(\Pi_M I (\lambda i. \text{borel}))$ *generator*

by (*rule algebra-generator*)

fix J **and** $B :: nat \Rightarrow ('i \Rightarrow 'a)$ *set*

assume $J: \bigwedge n. \text{finite } (J n) \bigwedge n. J n \subseteq I \bigwedge n. B n \in$ *sets* $(\Pi_M i \in J n. \text{borel})$

incseq J

and $B: \text{decseq } (\lambda n. emb I (J n) (B n))$

and $0 < (INF i. P (J i) (B i))$ (**is** $0 < ?a$)

moreover have $?a \leq 1$

using J **by** (*auto intro!: INF-lower2[of 0] prob-space-P[THEN prob-space.measure-le-1]*)

ultimately obtain r **where** $r: ?a = \text{ennreal } r$ $0 < r \leq 1$

by (*cases ?a*) (*auto simp: top-unique*)

define Z **where** $Z n = emb I (J n) (B n)$ **for** n

have Z -*mono*: $n \leq m \implies Z m \subseteq Z n$ **for** $n m$

unfolding Z -*def* **using** B [*THEN antimonoD*, *of n m*].

have J -*mono*: $\bigwedge n m. n \leq m \implies J n \subseteq J m$

using (*incseq J*) **by** (*force simp: incseq-def*)

```

note [simp] = ⟨ $\bigwedge n. \text{finite } (J\ n)$ ⟩
interpret prob-space  $P\ (J\ i)$  for  $i$  using  $J$  prob-space- $P$  by simp

have  $P$ -eq[simp]:
  sets  $(P\ (J\ i)) = \text{sets } (\prod_M\ i \in J\ i. \text{borel})\ \text{space } (P\ (J\ i)) = \text{space } (\prod_M\ i \in J\ i. \text{borel})$ 
for  $i$ 
using  $J$  by (auto simp: sets- $P$  space- $P$ )

have  $Z\ i \in \text{generator}$  for  $i$ 
unfolding  $Z$ -def by (auto intro!: generator.intros  $J$ )

have countable-UN- $J$ : countable  $(\bigcup n. J\ n)$  by (simp add: countable-finite)
define  $Utn$  where  $Utn = \text{to-nat-on } (\bigcup n. J\ n)$ 
interpret function-to-finmap  $J\ n\ Utn$  from-nat-into  $(\bigcup n. J\ n)$  for  $n$ 
by unfold-locale (auto simp:  $Utn$ -def intro: from-nat-into-to-nat-on[OF countable-UN- $J$ ])
have inj-on- $Utn$ : inj-on  $Utn\ (\bigcup n. J\ n)$ 
unfolding  $Utn$ -def using countable-UN- $J$  by (rule inj-on-to-nat-on)
hence inj-on- $Utn$ - $J$ :  $\bigwedge n. \text{inj-on } Utn\ (J\ n)$  by (rule subset-inj-on) auto
define  $P'$  where  $P'\ n = \text{mapmeasure } n\ (P\ (J\ n))\ (\lambda \cdot. \text{borel})$  for  $n$ 
interpret  $P'$ : prob-space  $P'\ n$  for  $n$ 
unfolding  $P'$ -def mapmeasure-def using  $J$ 
by (auto intro!: prob-space-distr fm-measurable simp: measurable-cong-sets[OF sets- $P$ ])

let ? $SUP = \lambda n. \text{SUP } K : \{K. K \subseteq \text{fm } n\ ' (B\ n) \wedge \text{compact } K\}. \text{emeasure } (P'\ n)\ K$ 
{ fix  $n$ 
have  $\text{emeasure } (P\ (J\ n))\ (B\ n) = \text{emeasure } (P'\ n)\ (\text{fm } n\ ' (B\ n))$ 
using  $J$  by (auto simp:  $P'$ -def mapmeasure-PiM space- $P$  sets- $P$ )
also
have ... = ? $SUP\ n$ 
proof (rule inner-regular)
show sets  $(P'\ n) = \text{sets borel}$  by (simp add: borel-eq-PiF-borel  $P'$ -def)
next
show  $\text{fm } n\ ' B\ n \in \text{sets borel}$ 
unfolding borel-eq-PiF-borel by (auto simp:  $P'$ -def fm-image-measurable-finite sets- $P\ J(\beta)$ )
qed simp
finally have *:  $\text{emeasure } (P\ (J\ n))\ (B\ n) = ?\mathit{SUP}\ n$  .
have ? $SUP\ n \neq \infty$ 
unfolding *[symmetric] by simp
note * this
} note  $R = \text{this}$ 
have  $\forall n. \exists K. \text{emeasure } (P\ (J\ n))\ (B\ n) - \text{emeasure } (P'\ n)\ K \leq 2\ \text{powr } (-n)$ 
* ? $a \wedge \text{compact } K \wedge K \subseteq \text{fm } n\ ' B\ n$ 
proof
fix  $n$  show  $\exists K. \text{emeasure } (P\ (J\ n))\ (B\ n) - \text{emeasure } (P'\ n)\ K \leq \text{ennreal } (2\ \text{powr } -\ \text{real } n) * ?a \wedge$ 
compact } K \wedge K \subseteq \text{fm } n\ ' B\ n

```

```

unfolding R[of n]
proof (rule ccontr)
  assume H:  $\nexists K'. ?SUP\ n - \text{emeasure } (P'\ n)\ K' \leq \text{ennreal } (2\ \text{powr} - \text{real } n) * ?a \wedge$ 
     $\text{compact } K' \wedge K' \subseteq \text{fm } n\ ' B\ n$ 
  have  $?SUP\ n + 0 < ?SUP\ n + 2\ \text{powr } (-n) * ?a$ 
  using R[of n] unfolding ennreal-add-left-cancel-less ennreal-zero-less-mult-iff
    by (auto intro:  $0 < ?a$ )
  also have  $\dots = (SUP\ K:\{K. K \subseteq \text{fm } n\ ' B\ n \wedge \text{compact } K\}. \text{emeasure } (P'\ n)\ K + 2\ \text{powr } (-n) * ?a)$ 
    by (rule ennreal-SUP-add-left[symmetric]) auto
  also have  $\dots \leq ?SUP\ n$ 
  proof (intro SUP-least)
    fix K assume  $K \in \{K. K \subseteq \text{fm } n\ ' B\ n \wedge \text{compact } K\}$ 
    with H have  $2\ \text{powr } (-n) * ?a < ?SUP\ n - \text{emeasure } (P'\ n)\ K$ 
    by auto
    then show  $\text{emeasure } (P'\ n)\ K + (2\ \text{powr } (-n)) * ?a \leq ?SUP\ n$ 
      by (subst (asm) less-diff-eq-ennreal) (auto simp: less-top[symmetric])
  R(1)[symmetric] ac-simps)
  qed
  finally show False by simp
qed
qed
then obtain K' where K':
   $\bigwedge n. \text{emeasure } (P\ (J\ n))\ (B\ n) - \text{emeasure } (P'\ n)\ (K'\ n) \leq \text{ennreal } (2\ \text{powr} - \text{real } n) * ?a$ 
   $\bigwedge n. \text{compact } (K'\ n) \wedge n. K'\ n \subseteq \text{fm } n\ ' B\ n$ 
  unfolding choice-iff by blast
define K where  $K\ n = \text{fm } n\ -' K'\ n \cap \text{space } (Pi_M\ (J\ n)\ (\lambda-. \text{borel}))$  for n
have K-sets:  $\bigwedge n. K\ n \in \text{sets } (Pi_M\ (J\ n)\ (\lambda-. \text{borel}))$ 
  unfolding K-def
  using compact-imp-closed[OF  $\langle \text{compact } (K'\ -) \rangle$ ]
  by (intro measurable-sets[OF fm-measurable, of - Collect finite])
    (auto simp: borel-eq-PiF-borel[symmetric])
have K-B:  $\bigwedge n. K\ n \subseteq B\ n$ 
proof
  fix x n assume  $x \in K\ n$ 
  then have fm-in:  $\text{fm } n\ x \in \text{fm } n\ ' B\ n$ 
  using K' by (force simp: K-def)
  show  $x \in B\ n$ 
  using  $\langle x \in K\ n \rangle$  K-sets sets.sets-into-space J(1,2,3)[of n] inj-on-image-mem-iff[OF inj-on-fm]
    by (metis (no-types) Int-iff K-def fm-in space-borel)
qed
define Z' where  $Z'\ n = \text{emb } I\ (J\ n)\ (K\ n)$  for n
have Z':  $\bigwedge n. Z'\ n \subseteq Z\ n$ 
  unfolding Z'-def Z-def
proof (rule prod-emb-mono, safe)
  fix n x assume  $x \in K\ n$ 

```

hence $fm\ n\ x \in K'\ n\ x \in space\ (Pi_M\ (J\ n)\ (\lambda.\ borel))$
by $(simp\text{-}all\ add:\ K\text{-}def\ space\text{-}P)$
note $this(1)$
also have $K'\ n \subseteq fm\ n\ 'B\ n$ **by** $(simp\ add:\ K')$
finally have $fm\ n\ x \in fm\ n\ 'B\ n$.
thus $x \in B\ n$
proof safe
fix y **assume** $y: y \in B\ n$
hence $y \in space\ (Pi_M\ (J\ n)\ (\lambda.\ borel))$ **using** $J\ sets.\ sets\text{-}into\ space[of\ B\ n\ P\ (J\ n)]$
by $(auto\ simp\ add:\ space\text{-}P\ sets\text{-}P)$
assume $fm\ n\ x = fm\ n\ y$
note $inj\text{-}onD[OF\ inj\text{-}on\text{-}fm[OF\ space\text{-}borel],$
 $OF\ \langle fm\ n\ x = fm\ n\ y \rangle\ \langle x \in space \rightarrow \langle y \in space \rightarrow \rangle]$
with y **show** $x \in B\ n$ **by** $simp$
qed
qed
have $\bigwedge n. Z'\ n \in generator$ **using** $J\ K'(\varnothing)$ **unfolding** $Z'\text{-}def$
by $(auto\ intro!\!: generator.\ intros\ measurable\text{-}sets[OF\ fm\text{-}measurable[of\ -\ Collect\ finite]])$
 $simp:\ K\text{-}def\ borel\text{-}eq\text{-}PiF\text{-}borel[symmetric]\ compact\text{-}imp\text{-}closed)$
define Y **where** $Y\ n = (\bigcap i \in \{1..n\}. Z'\ i)$ **for** n
hence $\bigwedge n\ k. Y\ (n + k) \subseteq Y\ n$ **by** $(induct\text{-}tac\ k)$ $(auto\ simp:\ Y\text{-}def)$
hence $Y\text{-}mono:$ $\bigwedge n\ m. n \leq m \implies Y\ m \subseteq Y\ n$ **by** $(auto\ simp:\ le\text{-}iff\text{-}add)$
have $Y\text{-}Z':$ $\bigwedge n. n \geq 1 \implies Y\ n \subseteq Z'\ n$ **by** $(auto\ simp:\ Y\text{-}def)$
hence $Y\text{-}Z:$ $\bigwedge n. n \geq 1 \implies Y\ n \subseteq Z\ n$ **using** Z' **by** $auto$

have $Y\text{-}notempty:$ $\bigwedge n. n \geq 1 \implies (Y\ n) \neq \{\}$
proof –
fix $n::nat$ **assume** $n \geq 1$ **hence** $Y\ n \subseteq Z\ n$ **by** $fact$
have $Y\ n = (\bigcap i \in \{1..n\}. emb\ I\ (J\ n)\ (emb\ (J\ n)\ (J\ i)\ (K\ i)))$ **using** $J\ J\text{-}mono$
by $(auto\ simp:\ Y\text{-}def\ Z'\text{-}def)$
also have $\dots = prod\text{-}emb\ I\ (\lambda.\ borel)\ (J\ n)\ (\bigcap i \in \{1..n\}. emb\ (J\ n)\ (J\ i)\ (K\ i))$
using $\langle n \geq 1 \rangle$
by $(subst\ prod\text{-}emb\text{-}INT)\ auto$
finally
have $Y\text{-}emb:$
 $Y\ n = prod\text{-}emb\ I\ (\lambda.\ borel)\ (J\ n)\ (\bigcap i \in \{1..n\}. prod\text{-}emb\ (J\ n)\ (\lambda.\ borel)\ (J\ i)\ (K\ i))$.
hence $Y\ n \in generator$ **using** $J\ J\text{-}mono\ K\text{-}sets\ \langle n \geq 1 \rangle$
by $(auto\ simp\ del:\ prod\text{-}emb\text{-}INT\ intro!\!: generator.\ intros)$
have $*$: $\mu G\ (Z\ n) = P\ (J\ n)\ (B\ n)$
unfolding $Z\text{-}def$ **using** J **by** $(intro\ mu\text{-}G\text{-}spec)\ auto$
then have $\mu G\ (Z\ n) \neq \infty$ **by** $auto$
note $*$
moreover have $*$: $\mu G\ (Y\ n) = P\ (J\ n)\ (\bigcap i \in \{Suc\ 0..n\}. prod\text{-}emb\ (J\ n)\ (\lambda.\ borel)\ (J\ i)\ (K\ i))$
unfolding $Y\text{-}emb$ **using** $J\ J\text{-}mono\ K\text{-}sets\ \langle n \geq 1 \rangle$ **by** $(subst\ mu\text{-}G\text{-}spec)\ auto$

then have $\mu G (Y n) \neq \infty$ by auto
 note *
 moreover have $\mu G (Z n - Y n) =$
 $P (J n) (B n - (\bigcap_{i \in \{1..n\}} \{Suc\ 0..n\}. prod-emb (J n) (\lambda-. borel) (J i) (K i)))$
 unfolding Z-def Y-emb prod-emb-Diff[symmetric] using J J-mono K-sets $\langle n$
 $\geq 1 \rangle$
 by (subst mu-G-spec) (auto intro!: sets.Diff)
 ultimately
 have $\mu G (Z n) - \mu G (Y n) = \mu G (Z n - Y n)$
 using J J-mono K-sets $\langle n \geq 1 \rangle$
 by (simp only: emeasure-eq-measure Z-def)
 (auto dest!: bspec[where x=n] intro!: measure-Diff[symmetric] set-mp[OF
 K-B]
 intro!: arg-cong[where f=enreal]
 simp: extensional-restrict emeasure-eq-measure prod-emb-iff sets-P
 space-P
 enreal-minus measure-nonneg)
 also have subs: $Z n - Y n \subseteq (\bigcup_{i \in \{1..n\}} (Z i - Z' i))$
 using $\langle n \geq 1 \rangle$ unfolding Y-def UN-extend-simps(7) by (intro UN-mono
 Diff-mono Z-mono order-refl) auto
 have $Z n - Y n \in generator (\bigcup_{i \in \{1..n\}} (Z i - Z' i)) \in generator$
 using $\langle Z' - \in generator \rangle \langle Z - \in generator \rangle \langle Y - \in generator \rangle$ by auto
 hence $\mu G (Z n - Y n) \leq \mu G (\bigcup_{i \in \{1..n\}} (Z i - Z' i))$
 using subs generator.additive-increasing[OF positive-mu-G additive-mu-G]
 unfolding increasing-def by auto
 also have $\dots \leq (\sum_{i \in \{1..n\}} \mu G (Z i - Z' i))$ using $\langle Z - \in generator \rangle \langle Z' - \in generator \rangle$
 by (intro generator.subadditive[OF positive-mu-G additive-mu-G]) auto
 also have $\dots \leq (\sum_{i \in \{1..n\}} 2^{powr - real i} * ?a)$
 proof (rule sum-mono)
 fix i assume $i \in \{1..n\}$ hence $i \leq n$ by simp
 have $\mu G (Z i - Z' i) = \mu G (prod-emb I (\lambda-. borel) (J i) (B i - K i))$
 unfolding Z'-def Z-def by simp
 also have $\dots = P (J i) (B i - K i)$
 using J K-sets by (subst mu-G-spec) auto
 also have $\dots = P (J i) (B i) - P (J i) (K i)$
 using K-sets J $\langle K - \subseteq B \rangle$ by (simp add: emeasure-Diff)
 also have $\dots = P (J i) (B i) - P' i (K' i)$
 unfolding K-def P'-def
 by (auto simp: mapmeasure-PiF borel-eq-PiF-borel[symmetric]
 compact-imp-closed[OF compact (K' -)] space-PiM PiE-def)
 also have $\dots \leq ennreal (2^{powr - real i}) * ?a$ using $K'(1)[of i]$.
 finally show $\mu G (Z i - Z' i) \leq (2^{powr - real i}) * ?a$.
 qed
 also have $\dots = ennreal ((\sum_{i \in \{1..n\}} (2^{powr - enn2real i})) * enn2real ?a)$
 using r by (simp add: sum-distrib-right ennreal-mult[symmetric])
 also have $\dots < ennreal (1 * enn2real ?a)$
 proof (intro ennreal-lessI mult-strict-right-mono)
 have $(\sum_{i = 1..n} 2^{powr - real i}) = (\sum_{i = 1..<Suc n. (1/2) ^ i}$

by (*rule sum.cong*) (*auto simp: powr-realpow powr-divide power-divide powr-minus-divide*)
also have $\{1..<Suc\ n\} = \{..<Suc\ n\} - \{0\}$ **by** *auto*
also have $sum\ (op\ \wedge\ (1\ /\ 2::real))\ (\{..<Suc\ n\} - \{0\}) =$
 $sum\ (op\ \wedge\ (1\ /\ 2))\ (\{..<Suc\ n\}) - 1$ **by** (*auto simp: sum-diff1*)
also have $\dots < 1$ **by** (*subst geometric-sum*) *auto*
finally show $(\sum\ i = 1..n.\ 2\ powr - enn2real\ i) < 1$ **by** *simp*
qed (*auto simp: r enn2real-positive-iff*)
also have $\dots = ?a$ **by** (*auto simp: r*)
also have $\dots \leq \mu G\ (Z\ n)$
using *J* **by** (*auto intro: INF-lower simp: Z-def mu-G-spec*)
finally have $\mu G\ (Z\ n) - \mu G\ (Y\ n) < \mu G\ (Z\ n)$.
hence $R: \mu G\ (Z\ n) < \mu G\ (Z\ n) + \mu G\ (Y\ n)$
using $\langle \mu G\ (Y\ n) \neq \infty \rangle$ **by** (*auto simp: zero-less-iff-neq-zero*)
then have $\mu G\ (Y\ n) > 0$
by *simp*
thus $Y\ n \neq \{\}$ **using** *positive-mu-G* **by** (*auto simp add: positive-def*)
qed
hence $\forall n \in \{1..\}. \exists y. y \in Y\ n$ **by** *auto*
then obtain y **where** $y: \bigwedge n. n \geq 1 \implies y\ n \in Y\ n$ **unfolding** *bchoice-iff* **by**
force
{
fix t **and** $n\ m::nat$
assume $1 \leq n\ n \leq m$ **hence** $1 \leq m$ **by** *simp*
from *Y-mono*[*OF* $\langle m \geq n \rangle$] *y*[*OF* $\langle 1 \leq m \rangle$] **have** $y\ m \in Y\ n$ **by** *auto*
also have $\dots \subseteq Z'\ n$ **using** *Y-Z'*[*OF* $\langle 1 \leq n \rangle$] .
finally
have $fm\ n\ (restrict\ (y\ m)\ (J\ n)) \in K'\ n$
unfolding *Z'-def K-def prod-emb-iff* **by** (*simp add: Z'-def K-def prod-emb-iff*)
moreover have $finmap-of\ (J\ n)\ (restrict\ (y\ m)\ (J\ n)) = finmap-of\ (J\ n)\ (y$
 $m)$
using *J* **by** (*simp add: fm-def*)
ultimately have $fm\ n\ (y\ m) \in K'\ n$ **by** *simp*
} note *fm-in-K'* = *this*
interpret *finmap-seqs-into-compact* $\lambda n. K'\ (Suc\ n)\ \lambda k. fm\ (Suc\ k)\ (y\ (Suc\ k))$
borel
proof
fix n **show** *compact* $(K'\ n)$ **by** *fact*
next
fix n
from *Y-mono*[*of* $n\ Suc\ n$] *y*[*of* $Suc\ n$] **have** $y\ (Suc\ n) \in Y\ (Suc\ n)$ **by** *auto*
also have $\dots \subseteq Z'\ (Suc\ n)$ **using** *Y-Z'* **by** *auto*
finally
have $fm\ (Suc\ n)\ (restrict\ (y\ (Suc\ n))\ (J\ (Suc\ n))) \in K'\ (Suc\ n)$
unfolding *Z'-def K-def prod-emb-iff* **by** (*simp add: Z'-def K-def prod-emb-iff*)
thus $K'\ (Suc\ n) \neq \{\}$ **by** *auto*
fix k
assume $k \in K'\ (Suc\ n)$
with *K'*[*of* $Suc\ n$] *sets.sets-into-space* **have** $k \in fm\ (Suc\ n)\ 'B\ (Suc\ n)$ **by**

auto

then obtain b **where** $k = \text{fm } (\text{Suc } n) b$ **by** *auto*
thus $\text{domain } k = \text{domain } (\text{fm } (\text{Suc } n) (y (\text{Suc } n)))$
by (*simp-all add: fm-def*)

next

fix t **and** $n m :: \text{nat}$

assume $n \leq m$ **hence** $\text{Suc } n \leq \text{Suc } m$ **by** *simp*

assume $t \in \text{domain } (\text{fm } (\text{Suc } n) (y (\text{Suc } n)))$

then obtain j **where** $j: t = \text{Utn } j j \in J (\text{Suc } n)$ **by** *auto*

hence $j \in J (\text{Suc } m)$ **using** $J\text{-mono}[OF \langle \text{Suc } n \leq \text{Suc } m \rangle]$ **by** *auto*

have $\text{img}: \text{fm } (\text{Suc } n) (y (\text{Suc } m)) \in K' (\text{Suc } n)$ **using** $\langle n \leq m \rangle$

by (*intro fm-in-K'*) *simp-all*

show $(\text{fm } (\text{Suc } m) (y (\text{Suc } m)))_F t \in (\lambda k. (k)_F t) ' K' (\text{Suc } n)$

apply (*rule image-eqI[OF - img]*)

using $\langle j \in J (\text{Suc } n) \rangle \langle j \in J (\text{Suc } m) \rangle$

unfolding j **by** (*subst proj-fm, auto*)**+**

qed

have $\forall t. \exists z. (\lambda i. (\text{fm } (\text{Suc } (\text{diagseq } i)) (y (\text{Suc } (\text{diagseq } i))))_F t \longrightarrow z$

using *diagonal-tendsto ..*

then obtain z **where** $z:$

$\bigwedge t. (\lambda i. (\text{fm } (\text{Suc } (\text{diagseq } i)) (y (\text{Suc } (\text{diagseq } i))))_F t \longrightarrow z t$

unfolding *choice-iff* **by** *blast*

{

fix $n :: \text{nat}$ **assume** $n \geq 1$

have $\bigwedge i. \text{domain } (\text{fm } n (y (\text{Suc } (\text{diagseq } i)))) = \text{domain } (\text{finmap-of } (\text{Utn } ' J n) z)$

by *simp*

moreover

{

fix t

assume $t: t \in \text{domain } (\text{finmap-of } (\text{Utn } ' J n) z)$

hence $t \in \text{Utn } ' J n$ **by** *simp*

then obtain j **where** $j: t = \text{Utn } j j \in J n$ **by** *auto*

have $(\lambda i. (\text{fm } n (y (\text{Suc } (\text{diagseq } i))))_F t \longrightarrow z t$

apply (*subst (2) tendsto-iff, subst eventually-sequentially*)

proof *safe*

fix $e :: \text{real}$ **assume** $0 < e$

{ **fix** i **and** $x :: 'i \Rightarrow 'a$ **assume** $i: i \geq n$

assume $t \in \text{domain } (\text{fm } n x)$

hence $t \in \text{domain } (\text{fm } i x)$ **using** $J\text{-mono}[OF \langle i \geq n \rangle]$ **by** *auto*

with i **have** $(\text{fm } i x)_F t = (\text{fm } n x)_F t$

using j **by** (*auto simp: proj-fm dest!: inj-onD[OF inj-on-Utn]*)

} **note** *index-shift = this*

have $I: \bigwedge i. i \geq n \implies \text{Suc } (\text{diagseq } i) \geq n$

apply (*rule le-SucI*)

apply (*rule order-trans*) **apply** *simp*

apply (*rule seq-suble[OF subseq-diagseq]*)

done

from z

have $\exists N. \forall i \geq N. \text{dist} ((\text{fm} (\text{Suc} (\text{diagseq } i)) (\text{y} (\text{Suc} (\text{diagseq } i))))_F t) (z t) < e$
unfolding *tendsto-iff eventually-sequentially using* $\langle 0 < e \rangle$ **by** *auto*
then obtain N **where** $N: \bigwedge i. i \geq N \implies$
 $\text{dist} ((\text{fm} (\text{Suc} (\text{diagseq } i)) (\text{y} (\text{Suc} (\text{diagseq } i))))_F t) (z t) < e$ **by** *auto*
show $\exists N. \forall na \geq N. \text{dist} ((\text{fm } n (\text{y} (\text{Suc} (\text{diagseq } na))))_F t) (z t) < e$
proof (*rule exI[where x=max N n], safe*)
fix na **assume** $\max N n \leq na$
hence $\text{dist} ((\text{fm } n (\text{y} (\text{Suc} (\text{diagseq } na))))_F t) (z t) =$
 $\text{dist} ((\text{fm} (\text{Suc} (\text{diagseq } na)) (\text{y} (\text{Suc} (\text{diagseq } na))))_F t) (z t)$ **using**
 t
by (*subst index-shift[OF I]*) *auto*
also have $\dots < e$ **using** $\langle \max N n \leq na \rangle$ **by** (*intro N*) *simp*
finally show $\text{dist} ((\text{fm } n (\text{y} (\text{Suc} (\text{diagseq } na))))_F t) (z t) < e$.
qed
qed
hence $(\lambda i. (\text{fm } n (\text{y} (\text{Suc} (\text{diagseq } i))))_F t) \longrightarrow (\text{finmap-of } (Utn \text{ ` } J n)$
 $z)_F t$
by (*simp add: tendsto-intros*)
} ultimately
have $(\lambda i. \text{fm } n (\text{y} (\text{Suc} (\text{diagseq } i)))) \longrightarrow \text{finmap-of } (Utn \text{ ` } J n) z$
by (*rule tendsto-finmap*)
hence $((\lambda i. \text{fm } n (\text{y} (\text{Suc} (\text{diagseq } i)))) o (\lambda i. i + n)) \longrightarrow \text{finmap-of } (Utn$
 $\text{ ` } J n) z$
by (*rule LIMSEQ-subseq-LIMSEQ*) (*simp add: strict-mono-def*)
moreover
have $(\forall i. ((\lambda i. \text{fm } n (\text{y} (\text{Suc} (\text{diagseq } i)))) o (\lambda i. i + n)) i \in K' n)$
apply (*auto simp add: o-def intro!: fm-in-K' $\langle 1 \leq n \rangle$ le-SucI*)
apply (*rule le-trans*)
apply (*rule le-add2*)
using *seq-suble[OF subseq-diagseq]*
apply *auto*
done
moreover
from $\langle \text{compact } (K' n) \rangle$ **have** $\text{closed } (K' n)$ **by** (*rule compact-imp-closed*)
ultimately
have $\text{finmap-of } (Utn \text{ ` } J n) z \in K' n$
unfolding *closed-sequential-limits* **by** *blast*
also have $\text{finmap-of } (Utn \text{ ` } J n) z = \text{fm } n (\lambda i. z (Utn i))$
unfolding *finmap-eq-iff*
proof *clarsimp*
fix i **assume** $i: i \in J n$
hence $\text{from-nat-into } (\bigcup n. J n) (Utn i) = i$
unfolding *Utn-def*
by (*subst from-nat-into-to-nat-on[OF countable-UN-J]*) *auto*
with i **show** $z (Utn i) = (\text{fm } n (\lambda i. z (Utn i)))_F (Utn i)$
by (*simp add: finmap-eq-iff fm-def compose-def*)
qed
finally have $\text{fm } n (\lambda i. z (Utn i)) \in K' n$.

```

moreover
let ?J =  $\bigcup n. J n$ 
have (?J  $\cap$  J n) = J n by auto
ultimately have restrict ( $\lambda i. z (Utn i)$ ) (?J  $\cap$  J n)  $\in$  K n
  unfolding K-def by (auto simp: space-P space-PiM)
hence restrict ( $\lambda i. z (Utn i)$ ) ?J  $\in$  Z' n unfolding Z'-def
  using J by (auto simp: prod-emb-def PiE-def extensional-def)
also have ...  $\subseteq$  Z n using Z' by simp
finally have restrict ( $\lambda i. z (Utn i)$ ) ?J  $\in$  Z n .
} note in-Z = this
hence ( $\bigcap i \in \{1.. \}. Z i$ )  $\neq$  {} by auto
thus ( $\bigcap i. Z i$ )  $\neq$  {}
  using INT-decseq-offset[OF antimonoI[OF Z-mono]] by simp
qed fact+

```

lemma *measure-lim-emb*:

```

J  $\subseteq$  I  $\implies$  finite J  $\implies$  X  $\in$  sets ( $\prod_M i \in J. \text{borel}$ )  $\implies$  measure lim (emb I J X)
= measure (P J) X
unfolding measure-def by (subst emeasure-lim-emb) auto

```

end

```

hide-const (open) PiF
hide-const (open) PiF
hide-const (open) Pi'
hide-const (open) finmap-of
hide-const (open) proj
hide-const (open) domain
hide-const (open) basis-finmap

```

sublocale *polish-projective* \subseteq P: prob-space lim

proof

```

have *: emb I {} { $\lambda x. \text{undefined}$ } = space ( $\prod_M i \in I. \text{borel}$ )
  by (auto simp: prod-emb-def space-PiM)
interpret prob-space P {}
  using prob-space-P by simp
show emeasure lim (space lim) = 1
  using emeasure-lim-emb[of {} { $\lambda x. \text{undefined}$ }] emeasure-space-1
  by (simp add: * PiM-empty space-P)

```

qed

locale *polish-product-prob-space* =

```

product-prob-space  $\lambda-. \text{borel}::('a::\text{polish-space})$  measure I for I::'i set

```

sublocale *polish-product-prob-space* \subseteq P: *polish-projective* I $\lambda J. \text{PiM } J (\lambda-. \text{borel}::('a)$
measure)

..

lemma (in *polish-product-prob-space*) *limP-eq-PiM*: lim = PiM I ($\lambda-. \text{borel}$)

by (rule PiM-eq) (auto simp: emeasure-PiM emeasure-lim-emb)

end

26 Random Permutations

theory *Random-Permutations*

imports

~~/src/HOL/Probability/Probability-Mass-Function

HOL-Library.Multiset-Permutations

begin

Choosing a set permutation (i.e. a distinct list with the same elements as the set) uniformly at random is the same as first choosing the first element of the list and then choosing the rest of the list as a permutation of the remaining set.

lemma *random-permutation-of-set*:

assumes *finite A A ≠ {}*

shows *pmf-of-set (permutations-of-set A) =*

do {

x ← pmf-of-set A;

xs ← pmf-of-set (permutations-of-set (A - {x}));

return-pmf (x#xs)

} (is ?lhs = ?rhs)

proof –

from *assms* **have** *permutations-of-set A = (∪ x∈A. op # x ‘ permutations-of-set (A - {x}))*

by (*simp add: permutations-of-set-nonempty*)

also from *assms* **have** *pmf-of-set ... = ?rhs*

by (*subst pmf-of-set-UN [where n = fact (card A - 1)]*)

(*auto simp: card-image disjoint-family-on-def map-pmf-def [symmetric] map-pmf-of-set-inj*)

finally show *?thesis .*

qed

A generic fold function that takes a function, an initial state, and a set and chooses a random order in which it then traverses the set in the same fashion as a left fold over a list. We first give a recursive definition.

function *fold-random-permutation* :: (*'a ⇒ 'b ⇒ 'b*) ⇒ *'b ⇒ 'a set ⇒ 'b pmf*

where

fold-random-permutation f x {} = return-pmf x

| *¬finite A ⇒ fold-random-permutation f x A = return-pmf x*

| *finite A ⇒ A ≠ {} ⇒*

fold-random-permutation f x A =

pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A - {a}))

by (*force, simp-all*)

termination proof (*relation Wellfounded.measure (λ(-,-,A). card A)*)

fix *A :: 'a set and f :: 'a ⇒ 'b ⇒ 'b and x :: 'b and y :: 'a*

```

assume  $A$ : finite  $A$   $A \neq \{\}$   $y \in \text{set-pmf}$  (pmf-of-set  $A$ )
then have  $\text{card } A > 0$  by (simp add: card-gt-0-iff)
with  $A$  show  $((f, f y x, A - \{y\}), f, x, A) \in \text{Wellfounded.measure } (\lambda(-, -, A). \text{card } A)$ 
by simp
qed simp-all

```

We can now show that the above recursive definition is equivalent to choosing a random set permutation and folding over it (in any direction).

```

lemma fold-random-permutation-foldl:
  assumes finite  $A$ 
  shows  $\text{fold-random-permutation } f x A =$ 
     $\text{map-pmf } (\text{foldl } (\lambda x y. f y x) x) (\text{pmf-of-set } (\text{permutations-of-set } A))$ 
using assms
proof (induction  $f x A$  rule: fold-random-permutation.induct [case-names empty infinite remove])
  case (remove  $A f x$ )
  from remove
  have  $\text{fold-random-permutation } f x A =$ 
     $\text{pmf-of-set } A \gg (\lambda a. \text{fold-random-permutation } f (f a x) (A - \{a\}))$  by
simp
  also from remove
  have  $\dots = \text{pmf-of-set } A \gg (\lambda a. \text{map-pmf } (\text{foldl } (\lambda x y. f y x) x)$ 
     $(\text{map-pmf } (\text{op } \# a) (\text{pmf-of-set } (\text{permutations-of-set } (A - \{a\}))))$ 
    by (intro bind-pmf-cong) (simp-all add: pmf.map-comp o-def)
  also from remove have  $\dots = \text{map-pmf } (\text{foldl } (\lambda x y. f y x) x) (\text{pmf-of-set } (\text{permutations-of-set } A))$ 
    by (simp-all add: random-permutation-of-set map-bind-pmf map-pmf-def [symmetric])
  finally show ?case .
qed (simp-all add: pmf-of-set-singleton)

```

```

lemma fold-random-permutation-foldr:
  assumes finite  $A$ 
  shows  $\text{fold-random-permutation } f x A =$ 
     $\text{map-pmf } (\lambda xs. \text{foldr } f xs x) (\text{pmf-of-set } (\text{permutations-of-set } A))$ 
proof –
  have  $\text{fold-random-permutation } f x A =$ 
     $\text{map-pmf } (\text{foldl } (\lambda x y. f y x) x \circ \text{rev}) (\text{pmf-of-set } (\text{permutations-of-set } A))$ 
    using assms by (subst fold-random-permutation-foldl [OF assms])
    (simp-all add: pmf.map-comp [symmetric] map-pmf-of-set-inj)
  also have  $\text{foldl } (\lambda x y. f y x) x \circ \text{rev} = (\lambda xs. \text{foldr } f xs x)$ 
    by (intro ext) (simp add: foldl-conv-foldr)
  finally show ?thesis .
qed

```

```

lemma fold-random-permutation-fold:
  assumes finite  $A$ 
  shows  $\text{fold-random-permutation } f x A =$ 
     $\text{map-pmf } (\lambda xs. \text{fold } f xs x) (\text{pmf-of-set } (\text{permutations-of-set } A))$ 

```

by (*subst fold-random-permutation-foldl [OF assms], intro map-pmf-cong*)
(simp-all add: foldl-conv-fold)

lemma *fold-random-permutation-code* [code]:

fold-random-permutation f x (set xs) =
map-pmf (foldl (λx y. f y x) x) (pmf-of-set (permutations-of-set (set xs)))
 by (*simp add: fold-random-permutation-foldl*)

We now introduce a slightly generalised version of the above fold operation that does not simply return the result in the end, but applies a monadic bind to it. This may seem somewhat arbitrary, but it is a common use case, e.g. in the Social Decision Scheme of Random Serial Dictatorship, where voters narrow down a set of possible winners in a random order and the winner is chosen from the remaining set uniformly at random.

function *fold-bind-random-permutation*

:: ('a ⇒ 'b ⇒ 'c) ⇒ ('b ⇒ 'c pmf) ⇒ 'b ⇒ 'a set ⇒ 'c pmf **where**

fold-bind-random-permutation f g x {} = g x

| *¬finite A ⇒ fold-bind-random-permutation f g x A = g x*

| *finite A ⇒ A ≠ {} ⇒*

fold-bind-random-permutation f g x A =

pmf-of-set A ≫ (λa. fold-bind-random-permutation f g (f a x) (A - {a}))

by (*force, simp-all*)

termination proof (*relation Wellfounded.measure (λ(-,-,-,A). card A)*)

fix *A :: 'a set and f :: 'a ⇒ 'b ⇒ 'c and x :: 'b*

and *y :: 'a and g :: 'b ⇒ 'c pmf*

assume *A: finite A A ≠ {} y ∈ set-pmf (pmf-of-set A)*

then have *card A > 0* by (*simp add: card-gt-0-iff*)

with *A show* (*(f, g, f y x, A - {y}), f, g, x, A*) ∈ *Wellfounded.measure (λ(-, -, -, A). card A)*

by *simp*

qed *simp-all*

We now show that the recursive definition is equivalent to a random fold followed by a monadic bind.

lemma *fold-bind-random-permutation-altdef* [code]:

fold-bind-random-permutation f g x A = fold-random-permutation f x A ≫ g

proof (*induction f x A rule: fold-random-permutation.induct [case-names empty infinite remove]*)

case (*remove A f x*)

from *remove have* *pmf-of-set A ≫ (λa. fold-bind-random-permutation f g (f a x) (A - {a})) =*

pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A - {a})) ≫ g

by (*intro bind-pmf-cong*) *simp-all*

with *remove show ?case* by (*simp add: bind-return-pmf bind-assoc-pmf*)

qed (*simp-all add: bind-return-pmf*)

We can now derive the following nice monadic representations of the combined fold-and-bind:

lemma *fold-bind-random-permutation-foldl*:
assumes *finite A*
shows $\text{fold-bind-random-permutation } f \ g \ x \ A =$
 $\text{do } \{xs \leftarrow \text{pmf-of-set } (\text{permutations-of-set } A); g \ (\text{foldl } (\lambda x \ y. \ f \ y \ x) \ x$
 $xs)\}$
using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-foldl *bind-return-pmf* *map-pmf-def*)

lemma *fold-bind-random-permutation-foldr*:
assumes *finite A*
shows $\text{fold-bind-random-permutation } f \ g \ x \ A =$
 $\text{do } \{xs \leftarrow \text{pmf-of-set } (\text{permutations-of-set } A); g \ (\text{foldr } f \ xs \ x)\}$
using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-foldr *bind-return-pmf* *map-pmf-def*)

lemma *fold-bind-random-permutation-fold*:
assumes *finite A*
shows $\text{fold-bind-random-permutation } f \ g \ x \ A =$
 $\text{do } \{xs \leftarrow \text{pmf-of-set } (\text{permutations-of-set } A); g \ (\text{fold } f \ xs \ x)\}$
using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-fold *bind-return-pmf* *map-pmf-def*)

The following useful lemma allows us to swap partitioning a set w. r. t. a predicate and drawing a random permutation of that set.

lemma *partition-random-permutations*:
assumes *finite A*
shows $\text{map-pmf } (\text{partition } P) \ (\text{pmf-of-set } (\text{permutations-of-set } A)) =$
 $\text{pair-pmf } (\text{pmf-of-set } (\text{permutations-of-set } \{x \in A. P \ x\}))$
 $(\text{pmf-of-set } (\text{permutations-of-set } \{x \in A. \neg P \ x\})) \ (\text{is } ?lhs = ?rhs)$
proof (*rule* *pmf-eqI*, *clarify*, *goal-cases*)
case (*1* *xs* *ys*)
show *?case*
proof (*cases* $xs \in \text{permutations-of-set } \{x \in A. P \ x\} \wedge ys \in \text{permutations-of-set}$
 $\{x \in A. \neg P \ x\}$)
case *True*
let *?n1* = *card* $\{x \in A. P \ x\}$ **and** *?n2* = *card* $\{x \in A. \neg P \ x\}$
have *card-eq*: *card* *A* = *?n1* + *?n2*
proof –
have *?n1* + *?n2* = *card* $(\{x \in A. P \ x\} \cup \{x \in A. \neg P \ x\})$
using *assms* **by** (*intro* *card-Un-disjoint* [*symmetric*]) *auto*
also **have** $\{x \in A. P \ x\} \cup \{x \in A. \neg P \ x\} = A$ **by** *blast*
finally **show** *?thesis* ..
qed
from *True* **have** *lengths* [*simp*]: *length* *xs* = *?n1* *length* *ys* = *?n2*
by (*auto* *intro!*: *length-finite-permutations-of-set*)
have *pmf* *?lhs* (*xs*, *ys*) =
 $\text{real } (\text{card } (\text{permutations-of-set } A \cap \text{partition } P - \{ \{xs, ys\} \})) / \text{fact}$
 $(\text{card } A)$

```

    using assms by (auto simp: pmf-map measure-pmf-of-set)
  also have partition  $P - \{ (xs, ys) \} = \text{shuffle } xs \ ys$ 
  using True by (intro inv-image-partition) (auto simp: permutations-of-set-def)
  also have permutations-of-set  $A \cap \text{shuffle } xs \ ys = \text{shuffle } xs \ ys$ 
    using True distinct-disjoint-shuffle[of xs ys]
    by (auto simp: permutations-of-set-def dest: set-shuffle)
  also have card  $(\text{shuffle } xs \ ys) = \text{length } xs + \text{length } ys$  choose length xs
  using True by (intro card-disjoint-shuffle) (auto simp: permutations-of-set-def)
  also have length xs + length ys = card A by (simp add: card-eq)
  also have real  $(\text{card } A \text{ choose } \text{length } xs) = \text{fact } (\text{card } A) / (\text{fact } ?n1 * \text{fact } (\text{card } A - ?n1))$ 
    by (subst binomial-fact) (auto intro!: card-mono assms)
  also have  $\dots / \text{fact } (\text{card } A) = 1 / (\text{fact } ?n1 * \text{fact } ?n2)$ 
    by (simp add: divide-simps card-eq)
  also have  $\dots = \text{pmf } ?rhs \ (xs, ys)$  using True assms by (simp add: pmf-pair)
  finally show ?thesis .
next
case False
  hence  $*: xs \notin \text{permutations-of-set } \{x \in A. P \ x\} \vee ys \notin \text{permutations-of-set } \{x \in A. \neg P \ x\}$  by blast
  hence eq: permutations-of-set  $A \cap (\text{partition } P - \{ (xs, ys) \}) = \{ \}$ 
    by (auto simp: o-def permutations-of-set-def)
  from  $*$  show ?thesis
  by (elim disjE) (insert assms eq, simp-all add: pmf-pair pmf-map measure-pmf-of-set)
qed
qed
end

```

27 Discrete subprobability distribution

theory *SPMF* imports

Probability-Mass-Function

HOL-Library.Complete-Partial-Order2

HOL-Library.Rewrite

begin

27.1 Auxiliary material

lemma *cSUP-singleton* [*simp*]: $(\text{SUP } x:\{x\}. f \ x :: - :: \text{conditionally-complete-lattice}) = f \ x$
 by (*metis cSup-singleton image-empty image-insert*)

27.1.1 More about extended reals

lemma [*simp*]:
 shows *ennreal-max-0*: $\text{ennreal } (\max \ 0 \ x) = \text{ennreal } x$
 and *ennreal-max-0'*: $\text{ennreal } (\max \ x \ 0) = \text{ennreal } x$
 by (*simp-all add: max-def ennreal-eq-0-iff*)

lemma *ennreal-enn2real-if*: $\text{ennreal } (\text{enn2real } r) = (\text{if } r = \top \text{ then } 0 \text{ else } r)$
by(*auto intro!*: *ennreal-enn2real simp add: less-top*)

lemma *e2ennreal-0* [*simp*]: $e2\text{ennreal } 0 = 0$
by(*simp add: zero-ennreal-def*)

lemma *enn2real-bot* [*simp*]: $\text{enn2real } \perp = 0$
by(*simp add: bot-ennreal-def*)

lemma *continuous-at-ennreal*[*continuous-intros*]: $\text{continuous } F f \implies \text{continuous } F (\lambda x. \text{ennreal } (f x))$
unfolding *continuous-def* **by** *auto*

lemma *ennreal-Sup*:
assumes *: $(\text{SUP } a:A. \text{ennreal } a) \neq \top$
and $A \neq \{\}$
shows $\text{ennreal } (\text{Sup } A) = (\text{SUP } a:A. \text{ennreal } a)$
proof (*rule continuous-at-Sup-mono*)
obtain r **where** $r: \text{ennreal } r = (\text{SUP } a:A. \text{ennreal } a) \text{ } r \geq 0$
using * **by**(*cases (SUP a:A. ennreal a) simp-all*)
then show *bdd-above* A
by(*auto intro!*: *SUP-upper bdd-aboveI[of - r] simp add: ennreal-le-iff[symmetric]*)
qed (*auto simp: mono-def continuous-at-imp-continuous-at-within continuous-at-ennreal ennreal-leI assms*)

lemma *ennreal-SUP*:
 $\llbracket (\text{SUP } a:A. \text{ennreal } (f a)) \neq \top; A \neq \{\} \rrbracket \implies \text{ennreal } (\text{SUP } a:A. f a) = (\text{SUP } a:A. \text{ennreal } (f a))$
using *ennreal-Sup*[*of f ' A*] **by** *auto*

lemma *ennreal-lt-0*: $x < 0 \implies \text{ennreal } x = 0$
by(*simp add: ennreal-eq-0-iff*)

27.1.2 More about 'a option

lemma *None-in-map-option-image* [*simp*]: $\text{None} \in \text{map-option } f \text{ ' } A \longleftrightarrow \text{None} \in A$
by *auto*

lemma *Some-in-map-option-image* [*simp*]: $\text{Some } x \in \text{map-option } f \text{ ' } A \longleftrightarrow (\exists y. x = f y \wedge \text{Some } y \in A)$
by(*auto intro: rev-image-eqI dest: sym*)

lemma *case-option-collapse*: $\text{case-option } x (\lambda \cdot. x) = (\lambda \cdot. x)$
by(*simp add: fun-eq-iff split: option.split*)

lemma *case-option-id*: $\text{case-option } \text{None } \text{Some} = \text{id}$
by(*rule ext*)(*simp split: option.split*)

inductive *ord-option* :: ('a ⇒ 'b ⇒ bool) ⇒ 'a option ⇒ 'b option ⇒ bool
for *ord* :: 'a ⇒ 'b ⇒ bool

where

None: *ord-option ord None x*
| *Some*: *ord x y* ⇒ *ord-option ord (Some x) (Some y)*

inductive-simps *ord-option-simps* [*simp*]:

ord-option ord None x
ord-option ord x None
ord-option ord (Some x) (Some y)
ord-option ord (Some x) None

inductive-simps *ord-option-eq-simps* [*simp*]:

ord-option op = None y
ord-option op = (Some x) y

lemma *ord-option-reflI*: (∧y. y ∈ *set-option x* ⇒ *ord y y*) ⇒ *ord-option ord x*
x

by(*cases x*) *simp-all*

lemma *reflp-ord-option*: *reflp ord* ⇒ *reflp (ord-option ord)*

by(*simp add: reflp-def ord-option-reflI*)

lemma *ord-option-trans*:

[[*ord-option ord x y*; *ord-option ord y z*;
∧ a b c. [[a ∈ *set-option x*; b ∈ *set-option y*; c ∈ *set-option z*; *ord a b*; *ord b c*
]] ⇒ *ord a c*]]
⇒ *ord-option ord x z*

by(*auto elim!: ord-option.cases*)

lemma *transp-ord-option*: *transp ord* ⇒ *transp (ord-option ord)*

unfolding *transp-def* **by**(*blast intro: ord-option-trans*)

lemma *antisymp-ord-option*: *antisymp ord* ⇒ *antisymp (ord-option ord)*

by(*auto intro!: antisympI elim!: ord-option.cases dest: antisympD*)

lemma *ord-option-chainD*:

Complete-Partial-Order.chain (ord-option ord) Y
⇒ *Complete-Partial-Order.chain ord {x. Some x ∈ Y}*

by(*rule chainI*)(*auto dest: chainD*)

definition *lub-option* :: ('a set ⇒ 'b) ⇒ 'a option set ⇒ 'b option

where *lub-option lub Y* = (*if Y* ⊆ {*None*} *then None* *else Some (lub {x. Some x*
∈ *Y*)}))

lemma *map-lub-option*: *map-option f (lub-option lub Y)* = *lub-option (f ∘ lub) Y*

by(*simp add: lub-option-def*)

lemma *lub-option-upper*:
assumes *Complete-Partial-Order.chain* (*ord-option ord*) Y $x \in Y$
and *lub-upper*: $\bigwedge Y x. \llbracket \text{Complete-Partial-Order.chain } \text{ord } Y; x \in Y \rrbracket \implies \text{ord } x (\text{lub } Y)$
shows *ord-option ord* $x (\text{lub-option lub } Y)$
using *assms(1-2)*
by(*cases x*)(*auto simp add: lub-option-def intro: lub-upper[OF ord-option-chainD]*)

lemma *lub-option-least*:
assumes $Y: \text{Complete-Partial-Order.chain } (\text{ord-option ord}) Y$
and *upper*: $\bigwedge x. x \in Y \implies \text{ord-option ord } x y$
assumes *lub-least*: $\bigwedge Y y. \llbracket \text{Complete-Partial-Order.chain } \text{ord } Y; \bigwedge x. x \in Y \implies \text{ord } x y \rrbracket \implies \text{ord } (\text{lub } Y) y$
shows *ord-option ord* (*lub-option lub* Y) y
using Y
by(*cases y*)(*auto 4 3 simp add: lub-option-def intro: lub-least[OF ord-option-chainD] dest: upper*)

lemma *lub-map-option*: *lub-option lub* (*map-option f* ‘ Y) = *lub-option* (*lub* \circ *op* ‘ f) Y
apply(*auto simp add: lub-option-def*)
apply(*erule notE*)
apply(*rule arg-cong[where f=lub]*)
apply(*auto intro: rev-image-eqI dest: sym*)
done

lemma *ord-option-mono*: $\llbracket \text{ord-option } A \ x \ y; \bigwedge x \ y. A \ x \ y \implies B \ x \ y \rrbracket \implies \text{ord-option } B \ x \ y$
by(*auto elim: ord-option.cases*)

lemma *ord-option-mono'* [*mono*]:
 $(\bigwedge x \ y. A \ x \ y \longrightarrow B \ x \ y) \implies \text{ord-option } A \ x \ y \longrightarrow \text{ord-option } B \ x \ y$
by(*blast intro: ord-option-mono*)

lemma *ord-option-compp*: *ord-option* ($A \ \text{OO} \ B$) = *ord-option* $A \ \text{OO} \ \text{ord-option } B$
by(*auto simp add: fun-eq-iff elim!: ord-option.cases intro: ord-option.intros*)

lemma *ord-option-inf*: *inf* (*ord-option* A) (*ord-option* B) = *ord-option* (*inf* $A \ B$)
(is ?lhs = ?rhs)
proof(*rule antisym*)
show $?lhs \leq ?rhs$ **by**(*auto elim!: ord-option.cases*)
qed(*auto elim: ord-option-mono*)

lemma *ord-option-map2*: *ord-option ord* x (*map-option f* y) = *ord-option* ($\lambda x \ y. \text{ord } x (f \ y)$) $x \ y$
by(*auto elim: ord-option.cases*)

lemma *ord-option-map1*: *ord-option ord* (*map-option f* x) y = *ord-option* ($\lambda x \ y. \text{ord } (f \ x) \ y$) $x \ y$

by(*auto elim: ord-option.cases*)

lemma *option-ord-Some1-iff*: *option-ord* (*Some* x) $y \longleftrightarrow y = \text{Some } x$
by(*auto simp add: flat-ord-def*)

27.1.3 A relator for sets that treats sets like predicates

context *includes lifting-syntax*
begin

definition *rel-pred* :: ($'a \Rightarrow 'b \Rightarrow \text{bool}$) \Rightarrow $'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow \text{bool}$
where *rel-pred* $R A B = (R \text{====>} \text{op} =) (\lambda x. x \in A) (\lambda y. y \in B)$

lemma *rel-predI*: $(R \text{====>} \text{op} =) (\lambda x. x \in A) (\lambda y. y \in B) \Longrightarrow \text{rel-pred } R A B$
by(*simp add: rel-pred-def*)

lemma *rel-predD*: $\llbracket \text{rel-pred } R A B; R x y \rrbracket \Longrightarrow x \in A \longleftrightarrow y \in B$
by(*simp add: rel-pred-def rel-fun-def*)

lemma *Collect-parametric*: $((A \text{====>} \text{op} =) \text{====>} \text{rel-pred } A) \text{Collect Collect}$
 — Declare this rule as *transfer-rule* only locally because it blows up the search space for *transfer* (in combination with *Collect-transfer*)
by(*simp add: rel-funI rel-predI*)

end

27.1.4 Monotonicity rules

lemma *monotone-gfp-eadd1*: *monotone* $\text{op} \geq \text{op} \geq (\lambda x. x + y :: \text{enat})$
by(*auto intro!: monotoneI*)

lemma *monotone-gfp-eadd2*: *monotone* $\text{op} \geq \text{op} \geq (\lambda y. x + y :: \text{enat})$
by(*auto intro!: monotoneI*)

lemma *mono2mono-gfp-eadd*[*THEN* *gfp.mono2mono2*, *cont-intro*, *simp*]:
shows *monotone-eadd*: *monotone* $(\text{rel-prod } \text{op} \geq \text{op} \geq) \text{op} \geq (\lambda(x, y). x + y :: \text{enat})$
by(*simp add: monotone-gfp-eadd1 monotone-gfp-eadd2*)

lemma *eadd-gfp-partial-function-mono* [*partial-function-mono*]:
 $\llbracket \text{monotone } (\text{fun-ord } \text{op} \geq) \text{op} \geq f; \text{monotone } (\text{fun-ord } \text{op} \geq) \text{op} \geq g \rrbracket$
 $\Longrightarrow \text{monotone } (\text{fun-ord } \text{op} \geq) \text{op} \geq (\lambda x. f x + g x :: \text{enat})$
by(*rule mono2mono-gfp-eadd*)

lemma *mono2mono-ereal*[*THEN* *lfp.mono2mono*]:
shows *monotone-ereal*: *monotone* $\text{op} \leq \text{op} \leq \text{ereal}$
by(*rule monotoneI*) *simp*

lemma *mono2mono-ennreal*[*THEN* *lfp.mono2mono*]:
shows *monotone-ennreal*: *monotone* $\text{op} \leq \text{op} \leq \text{ennreal}$

by(rule monotoneI)(simp add: ennreal-leI)

27.1.5 Bijections

lemma *bi-unique-rel-set-bij-betw*:

assumes *unique*: *bi-unique* *R*

and *rel*: *rel-set* *R* *A* *B*

shows $\exists f. \text{bij-betw } f \ A \ B \wedge (\forall x \in A. R \ x \ (f \ x))$

proof –

from *assms* **obtain** *f* **where** $f: \bigwedge x. x \in A \implies R \ x \ (f \ x)$ **and** $B: \bigwedge x. x \in A \implies f \ x \in B$

apply(*atomize-elim*)

apply(*fold all-conj-distrib*)

apply(*subst choice-iff*[*symmetric*])

apply(*auto dest: rel-setD1*)

done

have *inj-on* *f* *A* **by**(rule *inj-onI*)(*auto dest!*: *f dest: bi-uniqueDl*[*OF unique*])

moreover **have** $f \ ' \ A = B$ **using** *rel*

by(*auto* $\&$ *intro: B dest: rel-setD2 f bi-uniqueDr*[*OF unique*])

ultimately **have** *bij-betw* *f* *A* *B* **unfolding** *bij-betw-def* ..

thus *?thesis* **using** *f* **by** *blast*

qed

lemma *bij-betw-rel-setD*: *bij-betw* *f* *A* *B* $\implies \text{rel-set } (\lambda x \ y. y = f \ x) \ A \ B$

by(rule *rel-setI*)(*auto dest: bij-betwE bij-betw-imp-surj-on*[*symmetric*])

27.2 Subprobability mass function

type-synonym *'a spmf* = *'a option pmf*

translations (*type*) *'a spmf* \leftarrow (*type*) *'a option pmf*

definition *measure-spmf* :: *'a spmf* \Rightarrow *'a measure*

where *measure-spmf* *p* = *distr* (*restrict-space* (*measure-pmf* *p*) (*range Some*))
(*count-space UNIV*) *the*

abbreviation *spmf* :: *'a spmf* \Rightarrow *'a* \Rightarrow *real*

where *spmf* *p* *x* \equiv *pmf* *p* (*Some* *x*)

lemma *space-measure-spmf*: *space* (*measure-spmf* *p*) = *UNIV*

by(*simp add: measure-spmf-def*)

lemma *sets-measure-spmf* [*simp*, *measurable-cong*]: *sets* (*measure-spmf* *p*) = *sets*
(*count-space UNIV*)

by(*simp add: measure-spmf-def*)

lemma *measure-spmf-not-bot* [*simp*]: *measure-spmf* *p* $\neq \perp$

proof

assume *measure-spmf* *p* = \perp

hence *space* (*measure-spmf* *p*) = *space* \perp **by** *simp*

thus *False* **by**(*simp add: space-measure-spmf*)

qed

lemma *measurable-the-measure-pmf-Some* [*measurable, simp*]:

the \in *measurable* (*restrict-space* (*measure-pmf* *p*) (*range* *Some*)) (*count-space* *UNIV*)

by(*auto simp add: measurable-def sets-restrict-space space-restrict-space integral-restrict-space*)

lemma *measurable-spmf-measure1*[*simp*]: *measurable* (*measure-spmf* *M*) *N* = *UNIV*
 \rightarrow *space* *N*

by(*auto simp: measurable-def space-measure-spmf*)

lemma *measurable-spmf-measure2*[*simp*]: *measurable* *N* (*measure-spmf* *M*) = *measurable* *N* (*count-space* *UNIV*)

by(*intro measurable-cong-sets simp-all*)

lemma *subprob-space-measure-spmf* [*simp, intro!*]: *subprob-space* (*measure-spmf* *p*)

proof

show *emeasure* (*measure-spmf* *p*) (*space* (*measure-spmf* *p*)) ≤ 1

by(*simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space measure-pmf.measure-le-1*)

qed(*simp add: space-measure-spmf*)

interpretation *measure-spmf*: *subprob-space* *measure-spmf* *p* **for** *p*

by(*rule subprob-space-measure-spmf*)

lemma *finite-measure-spmf* [*simp*]: *finite-measure* (*measure-spmf* *p*)

by *unfold-locales*

lemma *spmf-conv-measure-spmf*: *spmf* *p* *x* = *measure* (*measure-spmf* *p*) {*x*}

by(*auto simp add: measure-spmf-def measure-distr measure-restrict-space pmf.rep-eq space-restrict-space intro: arg-cong2[where f=measure]*)

lemma *emeasure-measure-spmf-conv-measure-pmf*:

emeasure (*measure-spmf* *p*) *A* = *emeasure* (*measure-pmf* *p*) (*Some* ‘*A*)

by(*auto simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *measure-measure-spmf-conv-measure-pmf*:

measure (*measure-spmf* *p*) *A* = *measure* (*measure-pmf* *p*) (*Some* ‘*A*)

using *emeasure-measure-spmf-conv-measure-pmf*[*of* *p* *A*]

by(*simp add: measure-spmf.emeasure-eq-measure measure-pmf.emeasure-eq-measure*)

lemma *emeasure-spmf-map-pmf-Some* [*simp*]:

emeasure (*measure-spmf* (*map-pmf* *Some* *p*)) *A* = *emeasure* (*measure-pmf* *p*) *A*

by(*auto simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *measure-spmf-map-pmf-Some* [*simp*]:

$\text{measure } (\text{measure-spmf } (\text{map-pmf } \text{Some } p)) A = \text{measure } (\text{measure-pmf } p) A$
using *emeasure-spmf-map-pmf-Some*[of *p A*] **by**(*simp add: measure-spmf.emeasure-eq-measure*
measure-pmf.emeasure-eq-measure)

lemma *nn-integral-measure-spmf*: $(\int^+ x. f x \partial \text{measure-spmf } p) = \int^+ x. \text{ennreal}$
 $(\text{spmfmf } p x) * f x \partial \text{count-space UNIV}$
(is ?lhs = ?rhs)

proof –

have $?lhs = \int^+ x. \text{pmf } p x * f (\text{the } x) \partial \text{count-space } (\text{range } \text{Some})$
by(*simp add: measure-spmf-def nn-integral-distr nn-integral-restrict-space nn-integral-measure-pmf*
nn-integral-count-space-indicator ac-simps times-ereal.simps(1)[symmetric] del: times-ereal.simps(1))
also have $\dots = \int^+ x. \text{ennreal } (\text{spmfmf } p (\text{the } x)) * f (\text{the } x) \partial \text{count-space } (\text{range}$
Some)
by(*rule nn-integral-cong*) *auto*
also have $\dots = \int^+ x. \text{spmfmf } p (\text{the } (\text{Some } x)) * f (\text{the } (\text{Some } x)) \partial \text{count-space}$
UNIV
by(*rule nn-integral-bij-count-space[symmetric]*)(*simp add: bij-betw-def*)
also have $\dots = ?rhs$ **by** *simp*
finally show *?thesis* .

qed

lemma *integral-measure-spmf*:

assumes *integrable (measure-spmf p) f*
shows $(\int x. f x \partial \text{measure-spmf } p) = \int x. \text{spmfmf } p x * f x \partial \text{count-space UNIV}$
proof –
have *integrable (count-space UNIV) ($\lambda x. \text{spmfmf } p x * f x$)*
using *assms* **by**(*simp add: integrable-iff-bounded nn-integral-measure-spmf abs-mult*
ennreal-mult'')
then show *?thesis* **using** *assms*
by(*simp add: real-lebesgue-integral-def nn-integral-measure-spmf ennreal-mult'[symmetric]*)

qed

lemma *emeasure-spmf-single*: $\text{emeasure } (\text{measure-spmf } p) \{x\} = \text{spmfmf } p x$
by(*simp add: measure-spmf.emeasure-eq-measure spmfmf-conv-measure-spmf*)

lemma *measurable-measure-spmf[measurable]*:

$(\lambda x. \text{measure-spmf } (M x)) \in \text{measurable } (\text{count-space UNIV}) (\text{subprob-algebra}$
 $(\text{count-space UNIV}))$
by (*auto simp: space-subprob-algebra*)

lemma *nn-integral-measure-spmf-conv-measure-pmf*:

assumes [*measurable*]: $f \in \text{borel-measurable } (\text{count-space UNIV})$
shows $\text{nn-integral } (\text{measure-spmf } p) f = \text{nn-integral } (\text{restrict-space } (\text{measure-pmf}$
 $p) (\text{range } \text{Some})) (f \circ \text{the})$
by(*simp add: measure-spmf-def nn-integral-distr o-def*)

lemma *measure-spmf-in-space-subprob-algebra [simp]*:

$\text{measure-spmf } p \in \text{space } (\text{subprob-algebra } (\text{count-space UNIV}))$
by(*simp add: space-subprob-algebra*)

lemma *nn-integral-spmf-neq-top*: $(\int^+ x. \text{spmf } p \ x \ \partial\text{count-space } UNIV) \neq \top$
using *nn-integral-measure-spmf* [where $f=\lambda\cdot. 1$, of p , symmetric] **by** *simp*

lemma *SUP-spmf-neq-top'*: $(SUP \ p:Y. \ \text{ennreal } (\text{spmf } p \ x)) \neq \top$
proof(*rule neq-top-trans*)
show $(SUP \ p:Y. \ \text{ennreal } (\text{spmf } p \ x)) \leq 1$ **by**(*rule SUP-least*)(*simp add: pmf-le-1*)
qed *simp*

lemma *SUP-spmf-neq-top*: $(SUP \ i. \ \text{ennreal } (\text{spmf } (Y \ i) \ x)) \neq \top$
proof(*rule neq-top-trans*)
show $(SUP \ i. \ \text{ennreal } (\text{spmf } (Y \ i) \ x)) \leq 1$ **by**(*rule SUP-least*)(*simp add: pmf-le-1*)
qed *simp*

lemma *SUP-emeasure-spmf-neq-top*: $(SUP \ p:Y. \ \text{emeasure } (\text{measure-spmf } p) \ A) \neq \top$
proof(*rule neq-top-trans*)
show $(SUP \ p:Y. \ \text{emeasure } (\text{measure-spmf } p) \ A) \leq 1$
by(*rule SUP-least*)(*simp add: measure-spmf.subprob-emeasure-le-1*)
qed *simp*

27.3 Support

definition *set-spmf* :: $'a \ \text{spmf} \Rightarrow 'a \ \text{set}$
where *set-spmf* $p = \text{set-pmf } p \ggset \text{set-option}$

lemma *set-spmf-rep-eq*: $\text{set-spmf } p = \{x. \ \text{measure } (\text{measure-spmf } p) \ \{x\} \neq 0\}$
proof –
have $\bigwedge x :: 'a. \ \text{the } -' \ \{x\} \cap \text{range } \text{Some} = \{\text{Some } x\}$ **by** *auto*
then show *?thesis*
by(*auto simp add: set-spmf-def set-pmf.rep-eq measure-spmf-def measure-distr measure-restrict-space space-restrict-space intro: rev-image-eqI*)
qed

lemma *in-set-spmf*: $x \in \text{set-spmf } p \iff \text{Some } x \in \text{set-pmf } p$
by(*simp add: set-spmf-def*)

lemma *AE-measure-spmf-iff* [*simp*]: $(AE \ x \ \text{in } \text{measure-spmf } p. \ P \ x) \iff (\forall x \in \text{set-spmf } p. \ P \ x)$
by(*auto 4 3 simp add: measure-spmf-def AE-distr-iff AE-restrict-space-iff AE-measure-pmf-iff set-spmf-def cong del: AE-cong*)

lemma *spmf-eq-0-set-spmf*: $\text{spmf } p \ x = 0 \iff x \notin \text{set-spmf } p$
by(*auto simp add: pmf-eq-0-set-pmf set-spmf-def intro: rev-image-eqI*)

lemma *in-set-spmf-iff-spmf*: $x \in \text{set-spmf } p \iff \text{spmf } p \ x \neq 0$
by(*auto simp add: set-spmf-def set-pmf-iff intro: rev-image-eqI*)

lemma *set-spmf-return-pmf-None* [*simp*]: *set-spmf* (return-pmf None) = {}
by(auto simp add: set-spmf-def)

lemma *countable-set-spmf* [*simp*]: countable (set-spmf p)
by(simp add: set-spmf-def bind-UNION)

lemma *spmf-eqI*:

assumes $\bigwedge i. \text{spmf } p \ i = \text{spmf } q \ i$
shows $p = q$
proof(rule pmf-eqI)
fix i
show $\text{pmf } p \ i = \text{pmf } q \ i$
proof(cases i)
case (Some i')
thus ?thesis **by**(simp add: assms)
next
case None
have $\text{ennreal } (\text{pmf } p \ i) = \text{measure } (\text{measure-pmf } p) \ \{i\}$ **by**(simp add: pmf-def)
also have $\{i\} = \text{space } (\text{measure-pmf } p) - \text{range } \text{Some}$
by(auto simp add: None intro: ccontr)
also have $\text{measure } (\text{measure-pmf } p) \ \dots = \text{ennreal } 1 - \text{measure } (\text{measure-pmf } p) \ (\text{range } \text{Some})$
by(simp add: measure-pmf.prob-compl ennreal-minus[symmetric] del: space-measure-pmf)
also have $\text{range } \text{Some} = (\bigcup x \in \text{set-spmf } p. \ \{\text{Some } x\}) \cup \text{Some } '(- \text{set-spmf } p)$
by auto
also have $\text{measure } (\text{measure-pmf } p) \ \dots = \text{measure } (\text{measure-pmf } p) \ (\bigcup x \in \text{set-spmf } p. \ \{\text{Some } x\})$
by(rule measure-pmf.measure-zero-union)(auto simp add: measure-pmf.prob-eq-0 AE-measure-pmf-iff in-set-spmf-iff-spmf set-pmf-iff)
also have $\text{ennreal } \dots = \int^+ x. \text{measure } (\text{measure-pmf } p) \ \{\text{Some } x\} \ \partial \text{count-space } (\text{set-spmf } p)$
unfolding measure-pmf.emmeasure-eq-measure[symmetric]
by(simp-all add: emmeasure-UN-countable disjoint-family-on-def)
also have $\dots = \int^+ x. \text{spmf } p \ x \ \partial \text{count-space } (\text{set-spmf } p)$ **by**(simp add: pmf-def)
also have $\dots = \int^+ x. \text{spmf } q \ x \ \partial \text{count-space } (\text{set-spmf } p)$ **by**(simp add: assms)
also have $\text{set-spmf } p = \text{set-spmf } q$ **by**(auto simp add: in-set-spmf-iff-spmf assms)
also have $(\int^+ x. \text{spmf } q \ x \ \partial \text{count-space } (\text{set-spmf } q)) = \int^+ x. \text{measure } (\text{measure-pmf } q) \ \{\text{Some } x\} \ \partial \text{count-space } (\text{set-spmf } q)$
by(simp add: pmf-def)
also have $\dots = \text{measure } (\text{measure-pmf } q) \ (\bigcup x \in \text{set-spmf } q. \ \{\text{Some } x\})$
unfolding measure-pmf.emmeasure-eq-measure[symmetric]
by(simp-all add: emmeasure-UN-countable disjoint-family-on-def)
also have $\dots = \text{measure } (\text{measure-pmf } q) \ ((\bigcup x \in \text{set-spmf } q. \ \{\text{Some } x\}) \cup \text{Some } '(- \text{set-spmf } q))$
by(rule ennreal-cong measure-pmf.measure-zero-union[symmetric])+(auto simp add: measure-pmf.prob-eq-0 AE-measure-pmf-iff in-set-spmf-iff-spmf set-pmf-iff)

also have $((\bigcup x \in \text{set-spmf } q. \{ \text{Some } x \}) \cup \text{Some } (- \text{ set-spmf } q)) = \text{range } \text{Some}$
by *auto*
also have $\text{ennreal } 1 - \text{measure } (\text{measure-pmf } q) \dots = \text{measure } (\text{measure-pmf } q) (\text{space } (\text{measure-pmf } q) - \text{range } \text{Some})$
by (*simp add: one-ereal-def measure-pmf.prob-compl ennreal-minus[symmetric]*)
del: space-measure-pmf)
also have $\text{space } (\text{measure-pmf } q) - \text{range } \text{Some} = \{i\}$
by (*auto simp add: None intro: ccontr*)
also have $\text{measure } (\text{measure-pmf } q) \dots = \text{pmf } q \ i$ **by** (*simp add: pmf-def*)
finally show *?thesis* **by** *simp*
qed
qed

lemma *integral-measure-spmf-restrict*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$ **shows**
 $(\int x. f \ x \ \partial \text{measure-spmf } M) = (\int x. f \ x \ \partial \text{restrict-space } (\text{measure-spmf } M) (\text{set-spmf } M))$
by (*auto intro!: integral-cong-AE simp add: integral-restrict-space*)

lemma *nn-integral-measure-spmf'*:

$(\int^+ x. f \ x \ \partial \text{measure-spmf } p) = \int^+ x. \text{ennreal } (\text{spmfs } p \ x) * f \ x \ \partial \text{count-space } (\text{set-spmf } p)$
by (*auto simp add: nn-integral-measure-spmf nn-integral-count-space-indicator in-set-spmf-iff-spmf intro!: nn-integral-cong split: split-indicator*)

27.4 Functorial structure

abbreviation $\text{map-spmf} :: ('a \Rightarrow 'b) \Rightarrow 'a \ \text{spmfs} \Rightarrow 'b \ \text{spmfs}$
where $\text{map-spmf } f \equiv \text{map-pmf } (\text{map-option } f)$

context begin

local-setup $\langle \text{Local-Theory.map-background-naming } (\text{Name-Space.mandatory-path } \text{spmfs}) \rangle$

lemma *map-comp*: $\text{map-spmf } f (\text{map-spmf } g \ p) = \text{map-spmf } (f \circ g) \ p$
by (*simp add: pmf.map-comp o-def option.map-comp*)

lemma *map-id0*: $\text{map-spmf } \text{id} = \text{id}$
by (*simp add: pmf.map-id option.map-id0*)

lemma *map-id [simp]*: $\text{map-spmf } \text{id} \ p = p$
by (*simp add: map-id0*)

lemma *map-ident [simp]*: $\text{map-spmf } (\lambda x. x) \ p = p$
by (*simp add: id-def[symmetric]*)

end

lemma *set-map-spmf [simp]*: $\text{set-spmf } (\text{map-spmf } f \ p) = f \ \text{set-spmf } p$

by(*simp add: set-spmf-def image-bind bind-image o-def Option.option.set-map*)

lemma *map-spmf-cong*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \implies f x = g x \rrbracket$
 $\implies \text{map-spmf } f p = \text{map-spmf } g q$

by(*auto intro: pmf.map-cong option.map-cong simp add: in-set-spmf*)

lemma *map-spmf-cong-simp*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q =_{\text{simp}} \implies f x = g x \rrbracket$
 $\implies \text{map-spmf } f p = \text{map-spmf } g q$

unfolding *simp-implies-def* **by**(*rule map-spmf-cong*)

lemma *map-spmf-idI*: $(\bigwedge x. x \in \text{set-spmf } p \implies f x = x) \implies \text{map-spmf } f p = p$

by(*rule map-pmf-idI map-option-idI*)+(*simp add: in-set-spmf*)

lemma *emeasure-map-spmf*:

$\text{emeasure } (\text{measure-spmf } (\text{map-spmf } f p)) A = \text{emeasure } (\text{measure-spmf } p) (f \text{ - ' } A)$

by(*auto simp add: measure-spmf-def emeasure-distr measurable-restrict-space1 space-restrict-space emeasure-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *measure-map-spmf*: $\text{measure } (\text{measure-spmf } (\text{map-spmf } f p)) A = \text{measure } (\text{measure-spmf } p) (f \text{ - ' } A)$

using *emeasure-map-spmf[of f p A]* **by**(*simp add: measure-spmf.emeasure-eq-measure*)

lemma *measure-map-spmf-conv-distr*:

$\text{measure-spmf } (\text{map-spmf } f p) = \text{distr } (\text{measure-spmf } p) (\text{count-space UNIV}) f$
by(*rule measure-eqI*)(*simp-all add: emeasure-map-spmf emeasure-distr*)

lemma *spmf-map-pmf-Some* [*simp*]: $\text{spmf } (\text{map-pmf } \text{Some } p) i = \text{pmf } p i$

by(*simp add: pmf-map-inj'*)

lemma *spmf-map-inj*: $\llbracket \text{inj-on } f (\text{set-spmf } M); x \in \text{set-spmf } M \rrbracket \implies \text{spmf } (\text{map-spmf } f M) (f x) = \text{spmf } M x$

by(*subst option.map(2)[symmetric, where f=f]*)(*rule pmf-map-inj, auto simp add: in-set-spmf inj-on-def elim!: option.inj-map-strong[rotated]*)

lemma *spmf-map-inj'*: $\text{inj } f \implies \text{spmf } (\text{map-spmf } f M) (f x) = \text{spmf } M x$

by(*subst option.map(2)[symmetric, where f=f]*)(*rule pmf-map-inj'[OF option.inj-map]*)

lemma *spmf-map-outside*: $x \notin f \text{ - ' } \text{set-spmf } M \implies \text{spmf } (\text{map-spmf } f M) x = 0$

unfolding *spmf-eq-0-set-spmf* **by** *simp*

lemma *ennreal-spmf-map*: $\text{ennreal } (\text{spmf } (\text{map-spmf } f p) x) = \text{emeasure } (\text{measure-spmf } p) (f \text{ - ' } \{x\})$

by(*auto simp add: ennreal-pmf-map measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *spmf-map*: $\text{spmf } (\text{map-spmf } f p) x = \text{measure } (\text{measure-spmf } p) (f \text{ - ' } \{x\})$

using *ennreal-spmf-map*[*off p x*] **by**(*simp add: measure-spmf.emmeasure-eq-measure*)

lemma *ennreal-spmf-map-conv-nn-integral*:

ennreal (spm (map-spmf f p) x) = integral^N (measure-spmf p) (indicator (f - ‘{x}))

by(*auto simp add: ennreal-pmf-map measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space intro: arg-cong2[where f=emeasure]*)

27.5 Monad operations

27.5.1 Return

abbreviation *return-spmf* :: ‘*a* ⇒ ‘*a* *spm*

where *return-spmf* *x* ≡ *return-pmf* (Some *x*)

lemma *pmf-return-spmf*: *pmf (return-spmf x) y = indicator {y} (Some x)*

by(*fact pmf-return*)

lemma *measure-spmf-return-spmf*: *measure-spmf (return-spmf x) = Giry-Monad.return (count-space UNIV) x*

by(*rule measure-eqI*)(*simp-all add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space indicator-def*)

lemma *measure-spmf-return-pmf-None* [*simp*]: *measure-spmf (return-pmf None) = null-measure (count-space UNIV)*

by(*rule measure-eqI*)(*auto simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space indicator-eq-0-iff*)

lemma *set-return-spmf* [*simp*]: *set-spmf (return-spmf x) = {x}*

by(*auto simp add: set-spmf-def*)

27.5.2 Bind

definition *bind-spmf* :: ‘*a* *spm* ⇒ (‘*a* ⇒ ‘*b* *spm*) ⇒ ‘*b* *spm*

where *bind-spmf* *x f* = *bind-pmf* *x* (λ*a*. case *a* of None ⇒ *return-pmf* None | Some *a'* ⇒ *f a'*)

adhoc-overloading *Monad-Syntax.bind* *bind-spmf*

lemma *return-None-bind-spmf* [*simp*]: *return-pmf* None ≫ (f :: ‘*a* ⇒ -) = *return-pmf* None

by(*simp add: bind-spmf-def bind-return-pmf*)

lemma *return-bind-spmf* [*simp*]: *return-spmf* *x* ≫ f = f *x*

by(*simp add: bind-spmf-def bind-return-pmf*)

lemma *bind-return-spmf* [*simp*]: *x* ≫ *return-spmf* = *x*

proof –

have ∧*a* :: ‘*a* option. (case *a* of None ⇒ *return-pmf* None | Some *a'* ⇒ *return-spmf a'*) = *return-pmf a*

by(*simp split: option.split*)
 then show *?thesis*
 by(*simp add: bind-spmf-def bind-return-pmf'*)
 qed

lemma *bind-spmf-assoc* [*simp*]:

fixes $x :: 'a \text{ pmf}$ and $f :: 'a \Rightarrow 'b \text{ pmf}$ and $g :: 'b \Rightarrow 'c \text{ pmf}$
 shows $(x \ggg f) \ggg g = x \ggg (\lambda y. f y \ggg g)$
 by(*auto simp add: bind-spmf-def bind-assoc-pmf fun-eq-iff bind-return-pmf split: option.split intro: arg-cong[where f=bind-pmf x]*)

lemma *pmf-bind-spmf-None*: $\text{pmf } (p \ggg f) \text{ None} = \text{pmf } p \text{ None} + \int x. \text{pmf } (f x) \text{ None} \partial \text{measure-spmf } p$
 (is *?lhs = ?rhs*)

proof –

let $?f = \lambda x. \text{pmf } (\text{case } x \text{ of None} \Rightarrow \text{return-pmf None} \mid \text{Some } x \Rightarrow f x) \text{ None}$
 have $?lhs = \int x. ?f x \partial \text{measure-pmf } p$
 by(*simp add: bind-spmf-def pmf-bind*)
 also have $\dots = \int x. ?f \text{ None} * \text{indicator } \{\text{None}\} x + ?f x * \text{indicator } (\text{range } \text{Some}) x \partial \text{measure-pmf } p$
 by(*rule Bochner-Integration.integral-cong*)(*auto simp add: indicator-def*)
 also have $\dots = (\int x. ?f \text{ None} * \text{indicator } \{\text{None}\} x \partial \text{measure-pmf } p) + (\int x. ?f x * \text{indicator } (\text{range } \text{Some}) x \partial \text{measure-pmf } p)$
 by(*rule Bochner-Integration.integral-add*)(*auto 4 3 intro: integrable-real-mult-indicator measure-pmf.integrable-const-bound[where B=1] simp add: AE-measure-pmf-iff pmf-le-1*)
 also have $\dots = \text{pmf } p \text{ None} + \int x. \text{indicator } (\text{range } \text{Some}) x * \text{pmf } (f (the x)) \text{ None} \partial \text{measure-pmf } p$
 by(*auto simp add: measure-measure-pmf-finite indicator-eq-0-iff intro!: Bochner-Integration.integral-cong*)
 also have $\dots = ?rhs$ **unfolding** *measure-spmf-def*
 by(*subst integral-distr*)(*auto simp add: integral-restrict-space*)
 finally show *?thesis* .
 qed

lemma *spmf-bind*: $\text{spmf } (p \ggg f) y = \int x. \text{spmf } (f x) y \partial \text{measure-spmf } p$
unfolding *measure-spmf-def*

by(*subst integral-distr*)(*auto simp add: bind-spmf-def pmf-bind integral-restrict-space indicator-eq-0-iff intro!: Bochner-Integration.integral-cong split: option.split*)

lemma *ennreal-spmf-bind*: $\text{ennreal } (\text{spmf } (p \ggg f) x) = \int^+ y. \text{spmf } (f y) x \partial \text{measure-spmf } p$

by(*auto simp add: bind-spmf-def ennreal-pmf-bind nn-integral-measure-spmf-conv-measure-pmf nn-integral-restrict-space intro: nn-integral-cong split: split-indicator option.split*)

lemma *measure-spmf-bind-pmf*: $\text{measure-spmf } (p \ggg f) = \text{measure-pmf } p \ggg \text{measure-spmf } \circ f$

(is *?lhs = ?rhs*)

proof(*rule measure-eqI*)

show *sets ?lhs = sets ?rhs*

```

  by(simp add: sets-bind[where N=count-space UNIV] space-measure-spmf)
next
  fix A :: 'a set
  have emeasure ?lhs A =  $\int^+ x. \text{emeasure } (\text{measure-spmf } (f x)) A \partial \text{measure-pmf}$ 
  p
  by(simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space
  bind-spmf-def)
  also have ... = emeasure ?rhs A
  by(simp add: emeasure-bind[where N=count-space UNIV] space-measure-spmf
  space-subprob-algebra)
  finally show emeasure ?lhs A = emeasure ?rhs A .
qed

```

```

lemma measure-spmf-bind:  $\text{measure-spmf } (p \ggg f) = \text{measure-spmf } p \ggg \text{measure-spmf } f \circ f$ 
  (is ?lhs = ?rhs)
proof(rule measure-eqI)
  show sets ?lhs = sets ?rhs
  by(simp add: sets-bind[where N=count-space UNIV] space-measure-spmf)
next
  fix A :: 'a set
  let ?A = the -' A  $\cap$  range Some
  have emeasure ?lhs A =  $\int^+ x. \text{emeasure } (\text{measure-pmf } (\text{case } x \text{ of None } \Rightarrow \text{return-pmf None} \mid \text{Some } x \Rightarrow f x)) ?A \partial \text{measure-pmf } p$ 
  by(simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space
  bind-spmf-def)
  also have ... =  $\int^+ x. \text{emeasure } (\text{measure-pmf } (f (\text{the } x))) ?A * \text{indicator } (\text{range } \text{Some}) x \partial \text{measure-pmf } p$ 
  by(rule nn-integral-cong)(auto split: option.split simp add: indicator-def)
  also have ... =  $\int^+ x. \text{emeasure } (\text{measure-spmf } (f x)) A \partial \text{measure-spmf } p$ 
  by(simp add: measure-spmf-def nn-integral-distr nn-integral-restrict-space emeasure-distr
  space-restrict-space emeasure-restrict-space)
  also have ... = emeasure ?rhs A
  by(simp add: emeasure-bind[where N=count-space UNIV] space-measure-spmf
  space-subprob-algebra)
  finally show emeasure ?lhs A = emeasure ?rhs A .
qed

```

```

lemma map-spmf-bind-spmf:  $\text{map-spmf } f (\text{bind-spmf } p g) = \text{bind-spmf } p (\text{map-spmf } f \circ g)$ 
  by(auto simp add: bind-spmf-def map-bind-pmf fun-eq-iff split: option.split intro:
  arg-cong2[where f=bind-pmf])

```

```

lemma bind-map-spmf:  $\text{map-spmf } f p \ggg g = p \ggg g \circ f$ 
  by(simp add: bind-spmf-def bind-map-pmf o-def cong del: option.case-cong-weak)

```

```

lemma spmf-bind-leI:
  assumes  $\bigwedge y. y \in \text{set-spmf } p \implies \text{spmfs } (f y) x \leq r$ 
  and  $0 \leq r$ 

```

shows $\text{spmf } (\text{bind-spmf } p \ f) \ x \leq r$
proof –
 have $\text{ennreal } (\text{spmf } (\text{bind-spmf } p \ f) \ x) = \int^+ y. \text{spmf } (f \ y) \ x \ \partial \text{measure-spmf } p$
by(rule *ennreal-spmf-bind*)
 also have $\dots \leq \int^+ y. r \ \partial \text{measure-spmf } p$ **by**(rule *nn-integral-mono-AE*)(*simp add: assms*)
 also have $\dots \leq r$ **using** *assms measure-spmf.emmeasure-space-le-1*
by(*auto simp add: measure-spmf.emmeasure-eq-measure intro!: mult-left-le*)
finally show *?thesis* **using** *assms(2)* **by**(*simp*)
qed

lemma *map-spmf-conv-bind-spmf*: $\text{map-spmf } f \ p = (p \ggg (\lambda x. \text{return-spmf } (f \ x)))$
by(*simp add: map-pmf-def bind-spmf-def*)(rule *bind-pmf-cong, simp-all split: option.split*)

lemma *bind-spmf-cong*:
 $\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \implies f \ x = g \ x \rrbracket$
 $\implies \text{bind-spmf } p \ f = \text{bind-spmf } q \ g$
by(*auto simp add: bind-spmf-def in-set-spmf intro: bind-pmf-cong option.case-cong*)

lemma *bind-spmf-cong-simp*:
 $\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q = \text{simp} \implies f \ x = g \ x \rrbracket$
 $\implies \text{bind-spmf } p \ f = \text{bind-spmf } q \ g$
by(*simp add: simp-implies-def cong: bind-spmf-cong*)

lemma *set-bind-spmf*: $\text{set-spmf } (M \ggg f) = \text{set-spmf } M \ggg (\text{set-spmf } \circ f)$
by(*auto simp add: set-spmf-def bind-spmf-def bind-UNION split: option.splits*)

lemma *bind-spmf-const-return-None* [*simp*]: $\text{bind-spmf } p \ (\lambda _. \text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$
by(*simp add: bind-spmf-def case-option-collapse*)

lemma *bind-commute-spmf*:
 $\text{bind-spmf } p \ (\lambda x. \text{bind-spmf } q \ (f \ x)) = \text{bind-spmf } q \ (\lambda y. \text{bind-spmf } p \ (\lambda x. f \ x \ y))$
(is ?lhs = ?rhs)
proof –
 let $?f = \lambda x \ y. \text{case } x \ \text{of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } a \Rightarrow (\text{case } y \ \text{of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } b \Rightarrow f \ a \ b)$
 have $?lhs = p \ggg (\lambda x. q \ggg ?f \ x)$
 unfolding *bind-spmf-def* **by**(rule *bind-pmf-cong[OF refl]*)(*simp split: option.split*)
 also have $\dots = q \ggg (\lambda y. p \ggg (\lambda x. ?f \ x \ y))$ **by**(rule *bind-commute-pmf*)
 also have $\dots = ?rhs$ unfolding *bind-spmf-def*
by(rule *bind-pmf-cong[OF refl]*)(*auto split: option.split, metis bind-spmf-const-return-None bind-spmf-def*)
finally show *?thesis* .
qed

27.6 Relator

abbreviation $rel\text{-}spmf :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ spmf \Rightarrow 'b\ spmf \Rightarrow bool$
where $rel\text{-}spmf\ R \equiv rel\text{-}pmf\ (rel\text{-}option\ R)$

lemma $rel\text{-}pmf\text{-}mono$:

$\llbracket rel\text{-}pmf\ A\ f\ g; \bigwedge x\ y. A\ x\ y \implies B\ x\ y \rrbracket \implies rel\text{-}pmf\ B\ f\ g$
using $pmf.rel\text{-}mono[of\ A\ B]$ **by** $(simp\ add: le\text{-}fun\text{-}def)$

lemma $rel\text{-}spmf\text{-}mono$:

$\llbracket rel\text{-}spmf\ A\ f\ g; \bigwedge x\ y. A\ x\ y \implies B\ x\ y \rrbracket \implies rel\text{-}spmf\ B\ f\ g$
apply $(erule\ rel\text{-}pmf\text{-}mono)$
using $option.rel\text{-}mono[of\ A\ B]$ **by** $(simp\ add: le\text{-}fun\text{-}def)$

lemma $rel\text{-}spmf\text{-}mono\text{-}strong$:

$\llbracket rel\text{-}spmf\ A\ f\ g; \bigwedge x\ y. \llbracket A\ x\ y; x \in set\text{-}spmf\ f; y \in set\text{-}spmf\ g \rrbracket \implies B\ x\ y \rrbracket \implies rel\text{-}spmf\ B\ f\ g$
apply $(erule\ pmf.rel\text{-}mono\text{-}strong)$
apply $(erule\ option.rel\text{-}mono\text{-}strong)$
apply $(auto\ simp\ add: in\text{-}set\text{-}spmf)$
done

lemma $rel\text{-}spmf\text{-}reflI$: $(\bigwedge x. x \in set\text{-}spmf\ p \implies P\ x\ x) \implies rel\text{-}spmf\ P\ p\ p$
by $(rule\ rel\text{-}pmf\text{-}reflI)(auto\ simp\ add: set\text{-}spmf\text{-}def\ intro: rel\text{-}option\text{-}reflI)$

lemma $rel\text{-}spmfI$ $[intro?]$:

$\llbracket \bigwedge x\ y. (x, y) \in set\text{-}spmf\ pq \implies P\ x\ y; map\text{-}spmf\ fst\ pq = p; map\text{-}spmf\ snd\ pq = q \rrbracket$
 $\implies rel\text{-}spmf\ P\ p\ q$
by $(rule\ rel\text{-}pmf.intros[\mathbf{where}\ pq=map\text{-}pmf\ (\lambda x. case\ x\ of\ None \Rightarrow (None, None) \mid Some\ (a, b) \Rightarrow (Some\ a, Some\ b))\ pq])$
 $(auto\ simp\ add: pmf.map\text{-}comp\ o\text{-}def\ in\text{-}set\text{-}spmf\ split: option.splits\ intro: pmf.map\text{-}cong)$

lemma $rel\text{-}spmfE$ $[elim?, consumes\ 1, case\text{-}names\ rel\text{-}spmf]$:

assumes $rel\text{-}spmf\ P\ p\ q$
obtains pq **where**
 $\bigwedge x\ y. (x, y) \in set\text{-}spmf\ pq \implies P\ x\ y$
 $p = map\text{-}spmf\ fst\ pq$
 $q = map\text{-}spmf\ snd\ pq$
using $assms$
proof $(cases\ rule: rel\text{-}pmf.cases[consumes\ 1, case\text{-}names\ rel\text{-}pmf])$
case $(rel\text{-}pmf\ pq)$
let $?pq = map\text{-}pmf\ (\lambda(a, b). case\ (a, b)\ of\ (Some\ x, Some\ y) \Rightarrow Some\ (x, y) \mid - \Rightarrow None)\ pq$
have $\bigwedge x\ y. (x, y) \in set\text{-}spmf\ ?pq \implies P\ x\ y$
by $(auto\ simp\ add: in\text{-}set\text{-}spmf\ split: option.split\text{-}asm\ dest: rel\text{-}pmf(1))$
moreover
have $\bigwedge x. (x, None) \in set\text{-}pmf\ pq \implies x = None$ **by** $(auto\ dest!: rel\text{-}pmf(1))$
then have $p = map\text{-}spmf\ fst\ ?pq$ **using** $rel\text{-}pmf(2)$
by $(auto\ simp\ add: pmf.map\text{-}comp\ split\text{-}beta\ intro!: pmf.map\text{-}cong\ split: op$

tion.split)

moreover

have $\bigwedge y. (None, y) \in \text{set-pmf } pq \implies y = None$ **by** (*auto dest!*: *rel-pmf*(1))

then have $q = \text{map-spmf } \text{snd } ?pq$ **using** *rel-pmf*(3)

by (*auto simp add: pmf.map-comp split-beta intro!*: *pmf.map-cong split: option.split*)

ultimately show thesis ..

qed

lemma *rel-spmf-simps*:

$\text{rel-spmf } R \ p \ q \longleftrightarrow (\exists pq. (\forall (x, y) \in \text{set-spmf } pq. R \ x \ y) \wedge \text{map-spmf } \text{fst } pq = p \wedge \text{map-spmf } \text{snd } pq = q)$

by (*auto intro: rel-spmfI elim!*: *rel-spmfE*)

lemma *spmfm-rel-map*:

shows *spmfm-rel-map1*: $\bigwedge R \ f \ x. \text{rel-spmf } R \ (\text{map-spmf } f \ x) = \text{rel-spmf } (\lambda x. R \ (f \ x)) \ x$

and *spmfm-rel-map2*: $\bigwedge R \ x \ g \ y. \text{rel-spmf } R \ x \ (\text{map-spmf } g \ y) = \text{rel-spmf } (\lambda x \ y. R \ x \ (g \ y)) \ x \ y$

by (*simp-all add: fun-eq-iff pmf.rel-map option.rel-map[abs-def]*)

lemma *spmfm-rel-conversep*: $\text{rel-spmf } R^{-1-1} = (\text{rel-spmf } R)^{-1-1}$

by (*simp add: option.rel-conversep pmf.rel-conversep*)

lemma *spmfm-rel-eq*: $\text{rel-spmf } op = op =$

by (*simp add: pmf.rel-eq option.rel-eq*)

context includes *lifting-syntax*

begin

lemma *bind-spmf-parametric* [*transfer-rule*]:

$(\text{rel-spmf } A \ ===> (A \ ===> \text{rel-spmf } B) \ ===> \text{rel-spmf } B)$ *bind-spmf bind-spmf*
unfolding *bind-spmf-def[abs-def]* **by** *transfer-prover*

lemma *return-spmf-parametric*: $(A \ ===> \text{rel-spmf } A)$ *return-spmf return-spmf*

by *transfer-prover*

lemma *map-spmf-parametric*: $((A \ ===> B) \ ===> \text{rel-spmf } A \ ===> \text{rel-spmf } B)$ *map-spmf map-spmf*

by *transfer-prover*

lemma *rel-spmf-parametric*:

$((A \ ===> B \ ===> op =) \ ===> \text{rel-spmf } A \ ===> \text{rel-spmf } B \ ===> op =)$
rel-spmf rel-spmf

by *transfer-prover*

lemma *set-spmf-parametric* [*transfer-rule*]:

$(\text{rel-spmf } A \ ===> \text{rel-set } A)$ *set-spmf set-spmf*

unfolding *set-spmf-def[abs-def]* **by** *transfer-prover*

lemma *return-spmf-None-parametric*:

(*rel-spmf A*) (*return-pmf None*) (*return-pmf None*)
by *simp*

end

lemma *rel-spmf-bindI*:

$\llbracket \text{rel-spmf } R \text{ } p \text{ } q; \bigwedge x \ y. R \ x \ y \implies \text{rel-spmf } P \ (f \ x) \ (g \ y) \rrbracket$
 $\implies \text{rel-spmf } P \ (p \ggg f) \ (q \ggg g)$
by(*fact bind-spmf-parametric*[*THEN rel-funD, THEN rel-funD, OF - rel-funI*])

lemma *rel-spmf-bind-reflI*:

$(\bigwedge x. x \in \text{set-spmf } p \implies \text{rel-spmf } P \ (f \ x) \ (g \ x)) \implies \text{rel-spmf } P \ (p \ggg f) \ (p \ggg g)$
by(*rule rel-spmf-bindI*[**where** $R = \lambda x \ y. x = y \wedge x \in \text{set-spmf } p$])(*auto intro: rel-spmf-reflI*)

lemma *rel-pmf-return-pmfI*: $P \ x \ y \implies \text{rel-pmf } P \ (\text{return-pmf } x) \ (\text{return-pmf } y)$

by(*rule rel-pmf.intros*[**where** $pq = \text{return-pmf } (x, y)$])(*simp-all*)

context includes *lifting-syntax*

begin

We do not yet have a relator for 'a *measure*, so we combine *Sigma-Algebra.measure* and *measure-pmf*

lemma *measure-pmf-parametric*:

$(\text{rel-pmf } A \implies \text{rel-pred } A \implies \text{op } =) \ (\lambda p. \text{measure } (\text{measure-pmf } p)) \ (\lambda q. \text{measure } (\text{measure-pmf } q))$

proof(*rule rel-funI*)+

fix $p \ q \ X \ Y$

assume $\text{rel-pmf } A \ p \ q$ **and** $\text{rel-pred } A \ X \ Y$

from *this*(1) **obtain** pq **where** $A: \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies A \ x \ y$

and $p: p = \text{map-pmf } \text{fst } pq$ **and** $q: q = \text{map-pmf } \text{snd } pq$ **by** *cases auto*

show $\text{measure } p \ X = \text{measure } q \ Y$ **unfolding** $p \ q$ *measure-map-pmf*

by(*rule measure-pmf.finite-measure-eq-AE*)(*auto simp add: AE-measure-pmf-iff dest!: A rel-predD*[*OF rel-pred - -*])

qed

lemma *measure-spmf-parametric*:

$(\text{rel-spmf } A \implies \text{rel-pred } A \implies \text{op } =) \ (\lambda p. \text{measure } (\text{measure-spmf } p)) \ (\lambda q. \text{measure } (\text{measure-spmf } q))$

unfolding *measure-measure-spmf-conv-measure-pmf*[*abs-def*]

apply(*rule rel-funI*)+

apply(*erule measure-pmf-parametric*[*THEN rel-funD, THEN rel-funD*])

apply(*auto simp add: rel-pred-def rel-fun-def elim: option.rel-cases*)

done

end

27.7 From 'a pmf to 'a spmf**definition** *spmf-of-pmf* :: 'a pmf \Rightarrow 'a spmf**where** *spmf-of-pmf* = *map-pmf Some***lemma** *set-spmf-spmf-of-pmf* [*simp*]: *set-spmf* (*spmf-of-pmf* p) = *set-pmf* p**by**(*auto simp add: spmf-of-pmf-def set-spmf-def bind-image o-def*)**lemma** *spmf-spmf-of-pmf* [*simp*]: *spmf* (*spmf-of-pmf* p) x = *pmf* p x**by**(*simp add: spmf-of-pmf-def*)**lemma** *pmf-spmf-of-pmf-None* [*simp*]: *pmf* (*spmf-of-pmf* p) None = 0**using** *ennreal-pmf-map[of Some p None]* **by**(*simp add: spmf-of-pmf-def*)**lemma** *emeasure-spmf-of-pmf* [*simp*]: *emeasure* (*measure-spmf* (*spmf-of-pmf* p))A = *emeasure* (*measure-pmf* p) A**by**(*simp add: emeasure-measure-spmf-conv-measure-pmf spmf-of-pmf-def inj-vimage-image-eq*)**lemma** *measure-spmf-spmf-of-pmf* [*simp*]: *measure-spmf* (*spmf-of-pmf* p) = *measure-pmf* p**by**(*rule measure-eqI*) *simp-all***lemma** *map-spmf-of-pmf* [*simp*]: *map-spmf* f (*spmf-of-pmf* p) = *spmf-of-pmf* (*map-pmf* f p)**by**(*simp add: spmf-of-pmf-def pmf.map-comp o-def*)**lemma** *rel-spmf-spmf-of-pmf* [*simp*]: *rel-spmf* R (*spmf-of-pmf* p) (*spmf-of-pmf* q)= *rel-pmf* R p q**by**(*simp add: spmf-of-pmf-def pmf.rel-map*)**lemma** *spmf-of-pmf-return-pmf* [*simp*]: *spmf-of-pmf* (*return-pmf* x) = *return-spmf* x**by**(*simp add: spmf-of-pmf-def*)**lemma** *bind-spmf-of-pmf* [*simp*]: *bind-spmf* (*spmf-of-pmf* p) f = *bind-pmf* p f**by**(*simp add: spmf-of-pmf-def bind-spmf-def bind-map-pmf*)**lemma** *set-spmf-bind-pmf*: *set-spmf* (*bind-pmf* p f) = *Set.bind* (*set-pmf* p) (*set-spmf* o f)**unfolding** *bind-spmf-of-pmf[symmetric]* **by**(*subst set-bind-spmf*) *simp***lemma** *spmf-of-pmf-bind*: *spmf-of-pmf* (*bind-pmf* p f) = *bind-pmf* p ($\lambda x. \text{spmf-of-pmf } (f x)$)**by**(*simp add: spmf-of-pmf-def map-bind-pmf*)**lemma** *bind-pmf-return-spmf*: p \gg ($\lambda x. \text{return-spmf } (f x)$) = *spmf-of-pmf* (*map-pmf* f p)**by**(*simp add: map-pmf-def spmf-of-pmf-bind*)

27.8 Weight of a subprobability

abbreviation *weight-spmf* :: 'a spmf \Rightarrow real

where *weight-spmf* $p \equiv \text{measure } (\text{measure-spmf } p) (\text{space } (\text{measure-spmf } p))$

lemma *weight-spmf-def*: *weight-spmf* $p = \text{measure } (\text{measure-spmf } p) \text{ UNIV}$

by(*simp* *add*: *space-measure-spmf*)

lemma *weight-spmf-le-1*: *weight-spmf* $p \leq 1$

by(*simp* *add*: *measure-spmf.subprob-measure-le-1*)

lemma *weight-return-spmf* [*simp*]: *weight-spmf* (*return-spmf* x) = 1

by(*simp* *add*: *measure-spmf-return-spmf* *measure-return*)

lemma *weight-return-pmf-None* [*simp*]: *weight-spmf* (*return-pmf* *None*) = 0

by(*simp*)

lemma *weight-map-spmf* [*simp*]: *weight-spmf* (*map-spmf* f p) = *weight-spmf* p

by(*simp* *add*: *weight-spmf-def* *measure-map-spmf*)

lemma *weight-spmf-of-pmf* [*simp*]: *weight-spmf* (*spmf-of-pmf* p) = 1

using *measure-pmf.prob-space[of p]* **by**(*simp* *add*: *spmf-of-pmf-def* *weight-spmf-def*)

lemma *weight-spmf-nonneg*: *weight-spmf* $p \geq 0$

by(*fact* *measure-nonneg*)

lemma (**in** *finite-measure*) *integrable-weight-spmf* [*simp*]:

$(\lambda x. \text{weight-spmf } (f x)) \in \text{borel-measurable } M \implies \text{integrable } M (\lambda x. \text{weight-spmf } (f x))$

by(*rule* *integrable-const-bound[where B=1]*)(*simp-all* *add*: *weight-spmf-nonneg* *weight-spmf-le-1*)

lemma *weight-spmf-eq-nn-integral-spmf*: *weight-spmf* $p = \int^+ x. \text{spmf } p x \partial \text{count-space}$

UNIV

by(*simp* *add*: *measure-measure-spmf-conv-measure-pmf* *space-measure-spmf* *measure-pmf.emeasure-eq-measure-nn-integral-pmf[symmetric]* *embed-measure-count-space[symmetric]* *inj-on-def* *nn-integral-embed-measure* *measurable-embed-measure1*)

lemma *weight-spmf-eq-nn-integral-support*:

weight-spmf $p = \int^+ x. \text{spmf } p x \partial \text{count-space}$ (*set-spmf* p)

unfolding *weight-spmf-eq-nn-integral-spmf*

by(*auto* *simp* *add*: *nn-integral-count-space-indicator* *in-set-spmf-iff-spmf* *intro!*: *nn-integral-cong* *split*: *split-indicator*)

lemma *pmf-None-eq-weight-spmf*: *pmf* $p \text{ None} = 1 - \text{weight-spmf } p$

proof –

have *weight-spmf* $p = \int^+ x. \text{spmf } p x \partial \text{count-space}$ UNIV **by**(*rule* *weight-spmf-eq-nn-integral-spmf*)

also have $\dots = \int^+ x. \text{ennreal } (\text{pmf } p x) * \text{indicator } (\text{range } \text{Some}) x \partial \text{count-space}$

UNIV

by(*simp* *add*: *nn-integral-count-space-indicator[symmetric]* *embed-measure-count-space[symmetric]* *nn-integral-embed-measure* *measurable-embed-measure1*)

also have $\dots + \text{pmf } p \text{ None} = \int^+ x. \text{ennreal } (\text{pmf } p \ x) * \text{indicator } (\text{range } \text{Some}) \ x + \text{ennreal } (\text{pmf } p \ \text{None}) * \text{indicator } \{\text{None}\} \ x \ \partial \text{count-space } \text{UNIV}$
by(subst nn-integral-add)(simp-all add: max-def)
also have $\dots = \int^+ x. \text{pmf } p \ x \ \partial \text{count-space } \text{UNIV}$
by(rule nn-integral-cong)(auto split: split-indicator)
also have $\dots = 1$ **by** (simp add: nn-integral-pmf)
finally show ?thesis **by**(simp add: ennreal-plus[symmetric] del: ennreal-plus)
qed

lemma *weight-spmf-conv-pmf-None*: $\text{weight-spmf } p = 1 - \text{pmf } p \ \text{None}$
by(simp add: pmf-None-eq-weight-spmf)

lemma *weight-spmf-le-0*: $\text{weight-spmf } p \leq 0 \iff \text{weight-spmf } p = 0$
by(rule measure-le-0-iff)

lemma *weight-spmf-lt-0*: $\neg \text{weight-spmf } p < 0$
by(simp add: not-less weight-spmf-nonneg)

lemma *spm-f-le-weight*: $\text{spm-f } p \ x \leq \text{weight-spmf } p$
proof –

have $\text{ennreal } (\text{spm-f } p \ x) \leq \text{weight-spmf } p$
unfolding *weight-spmf-eq-nn-integral-spmf* **by**(rule nn-integral-ge-point) simp
then show ?thesis **by** simp
qed

lemma *weight-spmf-eq-0*: $\text{weight-spmf } p = 0 \iff p = \text{return-pmf } \text{None}$
by(auto intro!: pmf-eqI simp add: pmf-None-eq-weight-spmf split: split-indicator)(metis not-Some-eq pmf-le-0-iff spm-f-le-weight)

lemma *weight-bind-spmf*: $\text{weight-spmf } (x \ggg f) = \text{lebesgue-integral } (\text{measure-spmf } x) \ (\text{weight-spmf } \circ f)$
unfolding *weight-spmf-def*
by(simp add: measure-spmf-bind o-def measure-spmf.measure-bind[**where** $N = \text{count-space } \text{UNIV}$])

lemma *rel-spmf-weightD*: $\text{rel-spmf } A \ p \ q \implies \text{weight-spmf } p = \text{weight-spmf } q$
by(erule rel-spmfE) simp

lemma *rel-spmf-bij-betw*:

assumes f : *bij-betw* f (*set-spmf* p) (*set-spmf* q)
and eq : $\bigwedge x. x \in \text{set-spmf } p \implies \text{spm-f } p \ x = \text{spm-f } q \ (f \ x)$
shows $\text{rel-spmf } (\lambda x \ y. f \ x = y) \ p \ q$

proof –

let ? f = *map-option* f

have weq : $\text{ennreal } (\text{weight-spmf } p) = \text{ennreal } (\text{weight-spmf } q)$

unfolding *weight-spmf-eq-nn-integral-support*

by(subst nn-integral-bij-count-space[OF f , symmetric])(rule nn-integral-cong-AE, simp add: eq AE-count-space)

```

then have None ∈ set-pmf p ↔ None ∈ set-pmf q
  by(simp add: pmf-None-eq-weight-spmf set-pmf-iff)
with f have bij-betw (map-option f) (set-pmf p) (set-pmf q)
  apply(auto simp add: bij-betw-def in-set-spmf inj-on-def intro: option.expand)
  apply(rename-tac [!] x)
  apply(case-tac [!] x)
  apply(auto iff: in-set-spmf)
  done
then have rel-pmf (λx y. ?f x = y) p q
  by(rule rel-pmf-bij-betw)(case-tac x, simp-all add: weq[simplified] eq in-set-spmf
  pmf-None-eq-weight-spmf)
  thus ?thesis by(rule pmf.rel-mono-strong)(auto intro!: rel-optionI simp add:
  Option.is-none-def)
qed

```

27.9 From density to spmfs

context fixes $f :: 'a \Rightarrow \text{real}$ **begin**

definition *embed-spmf* :: $'a$ spmf

where *embed-spmf* = *embed-pmf* (λx. case x of None ⇒ 1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV) | Some x' ⇒ max 0 (f x'))

context

assumes *prob*: (∫⁺ x. ennreal (f x) ∂count-space UNIV) ≤ 1

begin

lemma *nn-integral-embed-spmf-eq-1*:

(∫⁺ x. ennreal (case x of None ⇒ 1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV) | Some x' ⇒ max 0 (f x')) ∂count-space UNIV) = 1

(**is** ?lhs = - **is** (∫⁺ x. ?f x ∂?M) = -)

proof -

have ?lhs = ∫⁺ x. ?f x * indicator {None} x + ?f x * indicator (range Some) x ∂?M

by(rule nn-integral-cong)(auto split: split-indicator)

also have ... = (1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV)) + ∫⁺ x. ?f x * indicator (range Some) x ∂?M

(**is** - = ?None + ?Some)

by(subst nn-integral-add)(simp-all add: AE-count-space max-def le-diff-eq real-le-ereal-iff one-ereal-def[symmetric] prob split: option.split)

also have ?Some = ∫⁺ x. ?f x ∂count-space (range Some)

by(simp add: nn-integral-count-space-indicator)

also have count-space (range Some) = embed-measure (count-space UNIV) Some

by(simp add: embed-measure-count-space)

also have (∫⁺ x. ?f x ∂...) = ∫⁺ x. ennreal (f x) ∂count-space UNIV

by(subst nn-integral-embed-measure)(simp-all add: measurable-embed-measure1)

also have ?None + ... = 1 **using** *prob*

by(auto simp add: ennreal-minus[symmetric] ennreal-1[symmetric] ennreal-enn2real-if top-unique simp del: ennreal-1)(simp add: diff-add-self-ennreal)

```

finally show ?thesis .
qed

```

```

lemma pmf-embed-spmf-None: pmf embed-spmf None = 1 - enn2real (f+ x.
ennreal (f x) ∂count-space UNIV)
unfolding embed-spmf-def
apply(subst pmf-embed-pmf)
  subgoal using prob by(simp add: field-simps enn2real-leI split: option.split)
  apply(rule nn-integral-embed-spmf-eq-1)
apply simp
done

```

```

lemma spmf-embed-spmf [simp]: spmf embed-spmf x = max 0 (f x)
unfolding embed-spmf-def
apply(subst pmf-embed-pmf)
  subgoal using prob by(simp add: field-simps enn2real-leI split: option.split)
  apply(rule nn-integral-embed-spmf-eq-1)
apply simp
done

```

```

end

```

```

end

```

```

lemma embed-spmf-K-0[simp]: embed-spmf (λ-. 0) = return-pmf None (is ?lhs =
?rhs)
by(rule spmf-eqI)(simp add: zero-ereal-def[symmetric])

```

27.10 Ordering on spmfs

rel-pmf does not preserve a ccpo structure. Counterexample by Saheb-Djahromi: Take prefix order over *bool llist* and the set *range* ($\lambda n :: nat. uniform (llist-n n)$) where *llist-n* is the set of all *llists* of length *n* and *uniform* returns a uniform distribution over the given set. The set forms a chain in *ord-pmf lprefix*, but it has not an upper bound. Any upper bound may contain only infinite lists in its support because otherwise it is not greater than the *n+1*-st element in the chain where *n* is the length of the finite list. Moreover its support must contain all infinite lists, because otherwise there is a finite list all of whose finite extensions are not in the support - a contradiction to the upper bound property. Hence, the support is uncountable, but pmf’s only have countable support.

However, if all chains in the ccpo are finite, then it should preserve the ccpo structure.

```

abbreviation ord-spmf :: ('a ⇒ 'a ⇒ bool) ⇒ 'a spmf ⇒ 'a spmf ⇒ bool
where ord-spmf ord ≡ rel-pmf (ord-option ord)

```

```

locale ord-spmf-syntax begin

```

notation *ord-spmf* (**infix** \sqsubseteq_1 60)
end

lemma *ord-spmf-map-spmf1*: *ord-spmf* *R* (*map-spmf* *f* *p*) = *ord-spmf* ($\lambda x. R$ (*f* *x*)) *p*
by(*simp* *add*: *pmf.rel-map*[*abs-def*] *ord-option-map1*[*abs-def*])

lemma *ord-spmf-map-spmf2*: *ord-spmf* *R* *p* (*map-spmf* *f* *q*) = *ord-spmf* ($\lambda x y. R$ *x* (*f* *y*)) *p* *q*
by(*simp* *add*: *pmf.rel-map* *ord-option-map2*)

lemma *ord-spmf-map-spmf12*: *ord-spmf* *R* (*map-spmf* *f* *p*) (*map-spmf* *f* *q*) = *ord-spmf* ($\lambda x y. R$ (*f* *x*) (*f* *y*)) *p* *q*
by(*simp* *add*: *pmf.rel-map* *ord-option-map1*[*abs-def*] *ord-option-map2*)

lemmas *ord-spmf-map-spmf* = *ord-spmf-map-spmf1* *ord-spmf-map-spmf2* *ord-spmf-map-spmf12*

context **fixes** *ord* :: 'a \Rightarrow 'a \Rightarrow *bool* (**structure**) **begin**
interpretation *ord-spmf-syntax* .

lemma *ord-spmfI*:

$\llbracket \bigwedge x y. (x, y) \in \text{set-spmf } pq \implies \text{ord } x y; \text{map-spmf } \text{fst } pq = p; \text{map-spmf } \text{snd } pq = q \rrbracket$
 $\implies p \sqsubseteq q$

by(*rule* *rel-pmf.intros*[**where** *pq=map-pmf* ($\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow (\text{None}, \text{None}) \mid \text{Some } (a, b) \Rightarrow (\text{Some } a, \text{Some } b)$) *pq*])
(*auto* *simp* *add*: *pmf.map-comp* *o-def* *in-set-spmf* *split*: *option.splits* *intro*: *pmf.map-cong*)

lemma *ord-spmf-None* [*simp*]: *return-pmf* *None* \sqsubseteq *x*

by(*rule* *rel-pmf.intros*[**where** *pq=map-pmf* (*Pair* *None*) *x*])(*auto* *simp* *add*: *pmf.map-comp* *o-def*)

lemma *ord-spmf-reflI*: ($\bigwedge x. x \in \text{set-spmf } p \implies \text{ord } x x$) $\implies p \sqsubseteq p$

by(*rule* *rel-pmf-reflI* *ord-option-reflI*)+(*auto* *simp* *add*: *in-set-spmf*)

lemma *rel-spmf-inf*:

assumes $p \sqsubseteq q$

and $q \sqsubseteq p$

and *refl*: *reflp* *ord*

and *trans*: *transp* *ord*

shows *rel-spmf* (*inf* *ord* *ord*⁻¹⁻¹) *p* *q*

proof –

from $\langle p \sqsubseteq q \rangle \langle q \sqsubseteq p \rangle$

have *rel-pmf* (*inf* (*ord-option* *ord*) (*ord-option* *ord*)⁻¹⁻¹) *p* *q*

by(*rule* *rel-pmf-inf*)(*blast* *intro*: *reflp-ord-option* *transp-ord-option* *refl* *trans*)
also **have** *inf* (*ord-option* *ord*) (*ord-option* *ord*)⁻¹⁻¹ = *rel-option* (*inf* *ord* *ord*⁻¹⁻¹)

by(*auto* *simp* *add*: *fun-eq-iff* *elim*: *ord-option.cases* *option.rel-cases*)

finally **show** *?thesis* .

qed

end

lemma *ord-spmf-return-spmf2*: $\text{ord-spmf } R \ p \ (\text{return-spmf } y) \longleftrightarrow (\forall x \in \text{set-spmf } p. R \ x \ y)$
by(*auto simp add: rel-pmf-return-pmf2 in-set-spmf ord-option.simps intro: ccontr*)

lemma *ord-spmf-mono*: $\llbracket \text{ord-spmf } A \ p \ q; \bigwedge x \ y. A \ x \ y \implies B \ x \ y \rrbracket \implies \text{ord-spmf } B \ p \ q$
by(*erule rel-pmf-mono*)(*erule ord-option-mono*)

lemma *ord-spmf-compp*: $\text{ord-spmf } (A \ OO \ B) = \text{ord-spmf } A \ OO \ \text{ord-spmf } B$
by(*simp add: ord-option-compp pmf.rel-compp*)

lemma *ord-spmf-bindI*:
assumes *pq*: $\text{ord-spmf } R \ p \ q$
and *fg*: $\bigwedge x \ y. R \ x \ y \implies \text{ord-spmf } P \ (f \ x) \ (g \ y)$
shows $\text{ord-spmf } P \ (p \ggg f) \ (q \ggg g)$
unfolding *bind-spmf-def* **using** *pq*
by(*rule rel-pmf-bindI*)(*auto split: option.split intro: fg*)

lemma *ord-spmf-bind-refI*:
 $(\bigwedge x. x \in \text{set-spmf } p \implies \text{ord-spmf } R \ (f \ x) \ (g \ x))$
 $\implies \text{ord-spmf } R \ (p \ggg f) \ (p \ggg g)$
by(*rule ord-spmf-bindI*[**where** $R = \lambda x \ y. x = y \wedge x \in \text{set-spmf } p$])(*auto intro: ord-spmf-refI*)

lemma *ord-pmf-increaseI*:
assumes *le*: $\bigwedge x. \text{spm}f \ p \ x \leq \text{spm}f \ q \ x$
and *refl*: $\bigwedge x. x \in \text{set-spm}f \ p \implies R \ x \ x$
shows $\text{ord-spm}f \ R \ p \ q$
proof(*rule rel-pmf.intros*)
define *pq* **where** $pq = \text{embed-pmf}$
 $(\lambda(x, y). \text{case } x \ \text{of } \text{Some } x' \Rightarrow (\text{case } y \ \text{of } \text{Some } y' \Rightarrow \text{if } x' = y' \ \text{then } \text{spm}f \ p \ x' \ \text{else } 0 \mid \text{None} \Rightarrow 0)$
 $\mid \text{None} \Rightarrow (\text{case } y \ \text{of } \text{None} \Rightarrow \text{pmf } q \ \text{None} \mid \text{Some } y' \Rightarrow \text{spm}f \ q \ y' - \text{spm}f \ p \ y'))$
(is $- = \text{embed-pmf } ?f$)
have *nonneg*: $\bigwedge xy. ?f \ xy \geq 0$
by(*clarsimp simp add: le field-simps split: option.split*)
have *integral*: $(\int^+ xy. ?f \ xy \ \partial \text{count-space UNIV}) = 1$ **(is** *nn-integral ?M - = -*)
proof $-$
have $(\int^+ xy. ?f \ xy \ \partial \text{count-space UNIV}) =$
 $\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } \{(None, None)\} \ xy +$
 $\text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (None, \text{Some } x))) \ xy +$
 $\text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (\text{Some } x, \text{Some } x))) \ xy \ \partial ?M$
by(*rule nn-integral-cong*)(*auto split: split-indicator option.splits if-split-asm*)
also have $\dots = (\int^+ xy. ?f \ xy * \text{indicator } \{(None, None)\} \ xy \ \partial ?M) +$
 $(\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (None, \text{Some } x))) \ xy \ \partial ?M)$


```

+
  (∫+ xy. ennreal (?f xy) * indicator (range (λx. (Some x, Some x))) xy
∂?M)
  (is - = ?None + ?Some2 + ?Some)
  by(subst nn-integral-add)(simp-all add: nn-integral-add AE-count-space le-diff-eq
le split: option.split)
  also have ?None = pmf q None by simp
  also have ?Some2 = ∫+ x. ennreal (spmf q x) - spmf p x ∂count-space UNIV
  by(simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric]
inj-on-def nn-integral-embed-measure measurable-embed-measure1 ennreal-minus)
  also have ... = (∫+ x. spmf q x ∂count-space UNIV) - (∫+ x. spmf p x
∂count-space UNIV)
  (is - = ?Some2' - ?Some2'')
  by(subst nn-integral-diff)(simp-all add: le nn-integral-spmf-neq-top)
  also have ?Some = ∫+ x. spmf p x ∂count-space UNIV
  by(simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric]
inj-on-def nn-integral-embed-measure measurable-embed-measure1)
  also have pmf q None + (?Some2' - ?Some2'') + ... = pmf q None +
?Some2'
  by(auto simp add: diff-add-self-ennreal le intro!: nn-integral-mono)
  also have ... = ∫+ x. ennreal (pmf q x) * indicator {None} x + ennreal (pmf
q x) * indicator (range Some) x ∂count-space UNIV
  by(subst nn-integral-add)(simp-all add: nn-integral-count-space-indicator[symmetric]
embed-measure-count-space[symmetric] nn-integral-embed-measure measurable-embed-measure1)
  also have ... = ∫+ x. pmf q x ∂count-space UNIV
  by(rule nn-integral-cong)(auto split: split-indicator)
  also have ... = 1 by(simp add: nn-integral-pmf)
  finally show ?thesis .
qed
note f = nonneg integral

{ fix x y
  assume (x, y) ∈ set-pmf pq
  hence ?f (x, y) ≠ 0 unfolding pq-def by(simp add: set-embed-pmf[OF f])
  then show ord-option R x y
    by(simp add: spmf-eq-0-set-spmf refl split: option.split-asm if-split-asm) }

have weight-le: weight-spmf p ≤ weight-spmf q
  by(subst ennreal-le-iff[symmetric])(auto simp add: weight-spmf-eq-nn-integral-spmf
intro!: nn-integral-mono le)

show map-pmf fst pq = p
proof(rule pmf-eqI)
  fix i
  have ennreal (pmf (map-pmf fst pq) i) = (∫+ y. pmf pq (i, y) ∂count-space
UNIV)
  unfolding pq-def ennreal-pmf-map
  apply(simp add: embed-pmf.rep-eq[OF f] o-def emeasure-density nn-integral-count-space-indicator[symmet
  apply(subst pmf-embed-pmf[OF f])

```

```

apply(rule nn-integral-bij-count-space[symmetric])
apply(auto simp add: bij-betw-def inj-on-def)
done
also have ... = pmf p i
proof(cases i)
  case (Some x)
    have ( $\int^+ y. \text{pmf } pq \text{ (Some } x, y) \partial \text{count-space UNIV}$ ) =  $\int^+ y. \text{pmf } p \text{ (Some } x) * \text{indicator } \{\text{Some } x\} y \partial \text{count-space UNIV}$ 
      by(rule nn-integral-cong)(simp add: pq-def pmf-embed-pmf[OF f] split:
option.split)
      then show ?thesis using Some by simp
    next
      case None
        have ( $\int^+ y. \text{pmf } pq \text{ (None, } y) \partial \text{count-space UNIV}$ ) =
          ( $\int^+ y. \text{ennreal (pmf } pq \text{ (None, Some (the } y)) * \text{indicator (range Some)}$ 
 $y +$ 
           $\text{ennreal (pmf } pq \text{ (None, None)) * indicator } \{\text{None}\} y \partial \text{count-space}$ 
UNIV)
          by(rule nn-integral-cong)(auto split: split-indicator)
        also have ... = ( $\int^+ y. \text{ennreal (pmf } pq \text{ (None, Some (the } y)) \partial \text{count-space}$ 
(range Some)) + pmf pq (None, None)
          by(subst nn-integral-add)(simp-all add: nn-integral-count-space-indicator)
        also have ... = ( $\int^+ y. \text{ennreal (spm f } q \text{ } y) - \text{ennreal (spm f } p \text{ } y) \partial \text{count-space}$ 
UNIV) + pmf q None
          by(simp add: pq-def pmf-embed-pmf[OF f] embed-measure-count-space[symmetric]
nn-integral-embed-measure measurable-embed-measure1 ennreal-minus)
        also have ( $\int^+ y. \text{ennreal (spm f } q \text{ } y) - \text{ennreal (spm f } p \text{ } y) \partial \text{count-space}$ 
UNIV) =
          ( $\int^+ y. \text{spm f } q \text{ } y \partial \text{count-space UNIV}$ ) - ( $\int^+ y. \text{spm f } p \text{ } y \partial \text{count-space}$ 
UNIV)
          by(subst nn-integral-diff)(simp-all add: AE-count-space le nn-integral-spmf-neq-top
split: split-indicator)
        also have ... = pmf p None - pmf q None
          by(simp add: pmf-None-eq-weight-spmf weight-spmf-eq-nn-integral-spmf[symmetric]
ennreal-minus)
        also have ... = ennreal (pmf p None) - ennreal (pmf q None) by(simp add:
ennreal-minus)
      finally show ?thesis using None weight-le
        by(auto simp add: diff-add-self-ennreal pmf-None-eq-weight-spmf intro:
ennreal-leI)
    qed
  finally show pmf (map-pmf fst pq) i = pmf p i by simp
qed

show map-pmf snd pq = q
proof(rule pmf-eqI)
  fix i
  have ennreal (pmf (map-pmf snd pq) i) = ( $\int^+ x. \text{pmf } pq \text{ (} x, i) \partial \text{count-space}$ 
UNIV)

```

```

unfolding pq-def ennreal-pmf-map
apply(simp add: embed-pmf.rep-eq[OF f] o-def emeasure-density nn-integral-count-space-indicator[symmetric])
apply(subst pmf-embed-pmf[OF f])
apply(rule nn-integral-bij-count-space[symmetric])
apply(auto simp add: bij-betw-def inj-on-def)
done
also have ... = ennreal (pmf q i)
proof(cases i)
  case None
    have  $(\int^+ x. \text{pmf } pq (x, \text{None}) \partial \text{count-space UNIV}) = \int^+ x. \text{pmf } q \text{ None} * \text{indicator } \{\text{None} :: 'a \text{ option}\} x \partial \text{count-space UNIV}$ 
      by(rule nn-integral-cong)(simp add: pq-def pmf-embed-pmf[OF f] split: option.split)
    then show ?thesis using None by simp
  next
    case (Some y)
    have  $(\int^+ x. \text{pmf } pq (x, \text{Some } y) \partial \text{count-space UNIV}) =$ 
       $(\int^+ x. \text{ennreal } (\text{pmf } pq (x, \text{Some } y)) * \text{indicator } (\text{range } \text{Some}) x +$ 
       $\text{ennreal } (\text{pmf } pq (\text{None}, \text{Some } y)) * \text{indicator } \{\text{None}\} x \partial \text{count-space UNIV})$ 
      by(rule nn-integral-cong)(auto split: split-indicator)
    also have ... =  $(\int^+ x. \text{ennreal } (\text{pmf } pq (x, \text{Some } y)) * \text{indicator } (\text{range } \text{Some}) x \partial \text{count-space UNIV}) + \text{pmf } pq (\text{None}, \text{Some } y)$ 
      by(subst nn-integral-add)(simp-all)
    also have ... =  $(\int^+ x. \text{ennreal } (\text{spmf } p y) * \text{indicator } \{\text{Some } y\} x \partial \text{count-space UNIV}) + (\text{spmf } q y - \text{spmf } p y)$ 
      by(auto simp add: pq-def pmf-embed-pmf[OF f] one-ereal-def[symmetric] simp del: nn-integral-indicator-singleton intro!: arg-cong2[where f=op +] nn-integral-cong split: option.split)
    also have ... =  $\text{spmf } q y$  by(simp add: ennreal-minus[symmetric] le)
    finally show ?thesis using Some by simp
  qed
finally show  $\text{pmf } (\text{map-pmf snd } pq) i = \text{pmf } q i$  by simp
qed
qed

```

lemma ord-spmf-eq-leD:

```

assumes ord-spmf op = p q
shows  $\text{spmf } p x \leq \text{spmf } q x$ 
proof(cases x  $\in$  set-spmf p)
  case False
    thus ?thesis by(simp add: in-set-spmf-iff-spmf)
  next
    case True
    from assms obtain pq
      where pq:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies \text{ord-option } op = x y$ 
      and p:  $p = \text{map-pmf fst } pq$ 
      and q:  $q = \text{map-pmf snd } pq$  by cases auto
    have  $\text{ennreal } (\text{spmf } p x) = \text{integral}^N pq (\text{indicator } (\text{fst} - \{ \text{Some } x \}))$ 

```

```

using  $p$  by(simp add: ennreal-pmf-map)
also have  $\dots = \text{integral}^N pq$  (indicator  $\{(Some\ x, Some\ x)\}$ )
by(rule nn-integral-cong-AE)(auto simp add: AE-measure-pmf-iff split: split-indicator
dest: pq)
also have  $\dots \leq \text{integral}^N pq$  (indicator  $(snd - \{Some\ x\})$ )
by(rule nn-integral-mono) simp
also have  $\dots = \text{ennreal} (spmf\ q\ x)$  using  $q$  by(simp add: ennreal-pmf-map)
finally show ?thesis by simp
qed

```

lemma *ord-spmf-eqD-set-spmf*: $ord\text{-}spmf\ op = p\ q \implies set\text{-}spmf\ p \subseteq set\text{-}spmf\ q$
by(*rule subsetI*)(*drule-tac x=x in ord-spmf-eq-leD, auto simp add: in-set-spmf-iff-spmf*)

lemma *ord-spmf-eqD-emeasure*:

$ord\text{-}spmf\ op = p\ q \implies \text{emeasure} (measure\text{-}spmf\ p)\ A \leq \text{emeasure} (measure\text{-}spmf\ q)\ A$

by(*auto intro!: nn-integral-mono split: split-indicator dest: ord-spmf-eq-leD simp*
add: nn-integral-measure-spmf nn-integral-indicator[symmetric])

lemma *ord-spmf-eqD-measure-spmf*: $ord\text{-}spmf\ op = p\ q \implies measure\text{-}spmf\ p \leq measure\text{-}spmf\ q$

by (*subst le-measure*) (*auto simp: ord-spmf-eqD-emeasure*)

27.11 CCPO structure for the flat ccpo $ord\text{-}option\ op =$

context *fixes* $Y :: 'a\ spmf\ set$ **begin**

definition *lub-spmf* :: $'a\ spmf$

where $lub\text{-}spmf = \text{embed-spmf} (\lambda x. \text{enn2real} (\text{SUP}\ p : Y. \text{ennreal} (spmf\ p\ x)))$

— We go through *ennreal* to have a sensible definition even if Y is empty.

lemma *lub-spmf-empty* [*simp*]: $SPMF.lub\text{-}spmf\ \{\} = \text{return-pmf}\ None$

by(*simp add: SPMF.lub-spmf-def bot-ereal-def*)

context *assumes* *chain*: *Complete-Partial-Order.chain* ($ord\text{-}spmf\ op =$) Y **begin**

lemma *chain-ord-spmf-eqD*: *Complete-Partial-Order.chain* ($op \leq$) $((\lambda p\ x. \text{ennreal} (spmf\ p\ x)) \text{ ' } Y)$

(*is Complete-Partial-Order.chain* - ($?f \text{ ' } -$))

proof(*rule chainI*)

fix $f\ g$

assume $f \in ?f \text{ ' } Y\ g \in ?f \text{ ' } Y$

then obtain $p\ q$ **where** $f: f = ?f\ p\ p \in Y$ **and** $g: g = ?f\ q\ q \in Y$ **by** *blast*

from *chain* $\langle p \in Y \rangle \langle q \in Y \rangle$ **have** $ord\text{-}spmf\ op = p\ q \vee ord\text{-}spmf\ op = q\ p$

by(*rule chainD*)

thus $f \leq g \vee g \leq f$

proof

assume $ord\text{-}spmf\ op = p\ q$

hence $\bigwedge x. spmf\ p\ x \leq spmf\ q\ x$ **by**(*rule ord-spmf-eq-leD*)

```

    hence  $f \leq g$  unfolding  $f g$  by(auto intro: le-funI)
    thus ?thesis ..
  next
    assume ord-spmf op = q p
    hence  $\bigwedge x. \text{spm}f\ q\ x \leq \text{spm}f\ p\ x$  by(rule ord-spmf-eq-leD)
    hence  $g \leq f$  unfolding  $f g$  by(auto intro: le-funI)
    thus ?thesis ..
  qed
qed

lemma ord-spmf-eq-pmf-None-eq:
  assumes le: ord-spmf op = p q
  and None: pmf p None = pmf q None
  shows  $p = q$ 
proof(rule spmf-eqI)
  fix  $i$ 
  from le have  $le'$ :  $\bigwedge x. \text{spm}f\ p\ x \leq \text{spm}f\ q\ x$  by(rule ord-spmf-eq-leD)
  have  $(\int^+ x. \text{ennreal}(\text{spm}f\ q\ x) - \text{spm}f\ p\ x\ \partial\text{count-space UNIV}) =$ 
     $(\int^+ x. \text{spm}f\ q\ x\ \partial\text{count-space UNIV}) - (\int^+ x. \text{spm}f\ p\ x\ \partial\text{count-space UNIV})$ 
    by(subst nn-integral-diff)(simp-all add: AE-count-space le' nn-integral-spmf-neq-top)
  also have  $\dots = (1 - \text{pmf}\ q\ \text{None}) - (1 - \text{pmf}\ p\ \text{None})$  unfolding pmf-None-eq-weight-spmf
    by(simp add: weight-spmf-eq-nn-integral-spmf[symmetric] ennreal-minus)
  also have  $\dots = 0$  using None by simp
  finally have  $\bigwedge x. \text{spm}f\ q\ x \leq \text{spm}f\ p\ x$ 
    by(simp add: nn-integral-0-iff-AE AE-count-space ennreal-minus ennreal-eq-0-iff)
  with  $le'$  show  $\text{spm}f\ p\ i = \text{spm}f\ q\ i$  by(rule antisym)
qed

lemma ord-spmf-eqD-pmf-None:
  assumes ord-spmf op = x y
  shows  $\text{pmf}\ x\ \text{None} \geq \text{pmf}\ y\ \text{None}$ 
using assms
apply cases
apply(clarsimp simp only: ennreal-le-iff[symmetric, OF pmf-nonneg] ennreal-pmf-map)
apply(fastforce simp add: AE-measure-pmf-iff intro!: nn-integral-mono-AE)
done

Chains on 'a spmf maintain countable support. Thanks to Johannes Hölzl
for the proof idea.

lemma spmf-chain-countable: countable  $(\bigcup_{p \in Y. \text{set-spm}f\ p}$ )
proof(cases Y = {})
  case Y: False
  show ?thesis
  proof(cases  $\exists x \in Y. \forall y \in Y. \text{ord-spm}f\ op = y\ x$ )
    case True
    then obtain  $x$  where  $x: x \in Y$  and upper:  $\bigwedge y. y \in Y \implies \text{ord-spm}f\ op = y$ 
    by blast
    hence  $(\bigcup_{x \in Y. \text{set-spm}f\ x} \subseteq \text{set-spm}f\ x$  by(auto dest: ord-spmf-eqD-set-spmf)
    thus ?thesis by(rule countable-subset) simp
  
```

```

next
  case False
  define N :: 'a option pmf  $\Rightarrow$  real where N p = pmf p None for p

  have N-less-imp-le-spmf:  $\llbracket x \in Y; y \in Y; N y < N x \rrbracket \Longrightarrow \text{ord-spmf } op = x$ 
y for x y
  using chainD[OF chain, of x y] ord-spmf-eqD-pmf-None[of x y] ord-spmf-eqD-pmf-None[of
y x]
  by (auto simp: N-def)
  have N-eq-imp-eq:  $\llbracket x \in Y; y \in Y; N y = N x \rrbracket \Longrightarrow x = y$  for x y
  using chainD[OF chain, of x y] by (auto simp add: N-def dest: ord-spmf-eq-pmf-None-eq)

  have NC:  $N \text{ ' } Y \neq \{\}$  bdd-below (N ' Y)
  using  $\langle Y \neq \{\} \rangle$  by (auto intro!: bdd-belowI[of - 0] simp: N-def)
  have NC-less:  $\text{Inf } (N \text{ ' } Y) < N x$  if  $x \in Y$  for x unfolding cInf-less-iff[OF
NC]
  proof (rule ccontr)
    assume **:  $\neg (\exists y \in N \text{ ' } Y. y < N x)$ 
    { fix y
      assume  $y \in Y$ 
      with ** consider  $N x < N y \mid N x = N y$  by (auto simp add: not-less
le-less)
      hence  $\text{ord-spmf } op = y x$  using  $\langle y \in Y \rangle \langle x \in Y \rangle$ 
      by cases (auto dest: N-less-imp-le-spmf N-eq-imp-eq intro: ord-spmf-refl)
    }
    with False  $\langle x \in Y \rangle$  show False by blast
  qed

  from NC have  $\text{Inf } (N \text{ ' } Y) \in \text{closure } (N \text{ ' } Y)$  by (intro closure-contains-Inf)
  then obtain X' where  $\bigwedge n. X' n \in N \text{ ' } Y$  and  $X': X' \longrightarrow \text{Inf } (N \text{ ' } Y)$ 
  unfolding closure-sequential by auto
  then obtain X where  $X: \bigwedge n. X n \in Y$  and  $X' = (\lambda n. N (X n))$  unfolding
image-iff Bex-def by metis

  with X' have seq:  $(\lambda n. N (X n)) \longrightarrow \text{Inf } (N \text{ ' } Y)$  by simp
  have  $(\bigcup x \in Y. \text{set-spmf } x) \subseteq (\bigcup n. \text{set-spmf } (X n))$ 
  proof (rule UN-least)
    fix x
    assume  $x \in Y$ 
    from order-tendstoD(2)[OF seq NC-less[OF  $\langle x \in Y \rangle$ ]]
    obtain i where  $N (X i) < N x$  by (auto simp: eventually-sequentially)
    thus  $\text{set-spmf } x \subseteq (\bigcup n. \text{set-spmf } (X n))$  using X  $\langle x \in Y \rangle$ 
    by (blast dest: N-less-imp-le-spmf ord-spmf-eqD-set-spmf)
  qed
  qed
  thus ?thesis by (rule countable-subset) simp
  qed
qed simp

```

lemma lub-spmf-subprob: $(\int^+ x. (\text{SUP } p : Y. \text{ennreal } (\text{spmfmf } p x))) \partial \text{count-space}$

```

UNIV) ≤ 1
proof(cases Y = {})
  case True
  thus ?thesis by(simp add: bot-ennreal)
next
  case False
  let ?B =  $\bigcup_{p \in Y} \text{set-spmf } p$ 
  have countable: countable ?B by(rule spmf-chain-countable)

  have ( $\int^+ x. (\text{SUP } p:Y. \text{ennreal } (\text{spmf } p \ x)) \ \partial \text{count-space UNIV}$ ) =
    ( $\int^+ x. (\text{SUP } p:Y. \text{ennreal } (\text{spmf } p \ x) * \text{indicator } ?B \ x) \ \partial \text{count-space UNIV}$ )
  by(intro nn-integral-cong SUP-cong)(auto split: split-indicator simp add: spmf-eq-0-set-spmf)
  also have ... = ( $\int^+ x. (\text{SUP } p:Y. \text{ennreal } (\text{spmf } p \ x)) \ \partial \text{count-space } ?B$ )
    unfolding ennreal-indicator[symmetric] using False
  by(subst SUP-mult-right-ennreal[symmetric])(simp add: ennreal-indicator nn-integral-count-space-indicator)
  also have ... = (SUP p:Y.  $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } ?B$ ) using False -
countable
  by(rule nn-integral-monotone-convergence-SUP-countable)(rule chain-ord-spmf-eqD)
  also have ... ≤ 1
  proof(rule SUP-least)
    fix p
    assume p ∈ Y
    have ( $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } ?B$ ) =  $\int^+ x. \text{ennreal } (\text{spmf } p \ x) * \text{indicator}$ 
    ?B x  $\partial \text{count-space UNIV}$ 
      by(simp add: nn-integral-count-space-indicator)
    also have ... =  $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space UNIV}$ 
      by(rule nn-integral-cong)(auto split: split-indicator simp add: spmf-eq-0-set-spmf
    (p ∈ Y))
    also have ... ≤ 1
      by(simp add: weight-spmf-eq-nn-integral-spmf[symmetric] weight-spmf-le-1)
    finally show ( $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } ?B$ ) ≤ 1 .
  qed
  finally show ?thesis .
qed

```

lemma *spmf-lub-spmf*:

```

assumes Y ≠ {}
shows spmf lub-spmf x = (SUP p : Y. spmf p x)
proof –
  from assms obtain p where p ∈ Y by auto
  have spmf lub-spmf x = max 0 (enn2real (SUP p:Y. ennreal (spmf p x))) un-
folding lub-spmf-def
  by(rule spmf-embed-spmf)(simp del: SUP-eq-top-iff Sup-eq-top-iff add: ennreal-enn2real-if
  SUP-spmf-neq-top' lub-spmf-subprob)
  also have ... = enn2real (SUP p:Y. ennreal (spmf p x))
    by(rule max-absorb2)(simp)
  also have ... = enn2real (ennreal (SUP p : Y. spmf p x)) using assms
  by(subst ennreal-SUP[symmetric])(simp-all add: SUP-spmf-neq-top' del: SUP-eq-top-iff
  Sup-eq-top-iff)

```

also have $0 \leq (\bigsqcup_{p \in Y}. \text{spm}f\ p\ x)$ **using** *assms*
by(*auto intro!*: *cSUP-upper2 bdd-aboveI*[**where** $M=1$] *simp add: pmf-le-1*)
then have $\text{enn2real}(\text{ennreal}(\text{SUP } p : Y. \text{spm}f\ p\ x)) = (\text{SUP } p : Y. \text{spm}f\ p\ x)$
by(*rule enn2real-ennreal*)
finally show *?thesis* .
qed

lemma *ennreal-spmf-lub-spmf*: $Y \neq \{\}$ $\implies \text{ennreal}(\text{spm}f\ \text{lub-spm}f\ x) = (\text{SUP } p : Y. \text{ennreal}(\text{spm}f\ p\ x))$
unfolding *spm-f-lub-spm-f* **by**(*subst ennreal-SUP*)(*simp-all add: SUP-spmf-neq-top'*
del: SUP-eq-top-iff Sup-eq-top-iff)

lemma *lub-spmf-upper*:
assumes $p : p \in Y$
shows $\text{ord-spm}f\ op = p\ \text{lub-spm}f$
proof(*rule ord-pmf-increaseI*)
fix x
from p **have** [*simp*]: $Y \neq \{\}$ **by** *auto*
from p **have** $\text{ennreal}(\text{spm}f\ p\ x) \leq (\text{SUP } p : Y. \text{ennreal}(\text{spm}f\ p\ x))$ **by**(*rule SUP-upper*)
also have $\dots = \text{ennreal}(\text{spm}f\ \text{lub-spm}f\ x)$ **using** p
by(*subst spmf-lub-spmf*)(*auto simp add: ennreal-SUP SUP-spmf-neq-top' simp del: SUP-eq-top-iff Sup-eq-top-iff*)
finally show $\text{spm}f\ p\ x \leq \text{spm}f\ \text{lub-spm}f\ x$ **by** *simp*
qed *simp*

lemma *lub-spmf-least*:
assumes $z : \bigwedge x. x \in Y \implies \text{ord-spm}f\ op = x\ z$
shows $\text{ord-spm}f\ op = \text{lub-spm}f\ z$
proof(*cases Y = \{\}*)
case *nonempty*: *False*
show *?thesis*
proof(*rule ord-pmf-increaseI*)
fix x
from *nonempty* **obtain** p **where** $p : p \in Y$ **by** *auto*
have $\text{ennreal}(\text{spm}f\ \text{lub-spm}f\ x) = (\text{SUP } p : Y. \text{ennreal}(\text{spm}f\ p\ x))$
by(*subst spmf-lub-spmf*)(*auto simp add: ennreal-SUP SUP-spmf-neq-top'*
nonempty simp del: SUP-eq-top-iff Sup-eq-top-iff)
also have $\dots \leq \text{ennreal}(\text{spm}f\ z\ x)$ **by**(*rule SUP-least*)(*simp add: ord-spmf-eq-leD z*)
finally show $\text{spm}f\ \text{lub-spm}f\ x \leq \text{spm}f\ z\ x$ **by** *simp*
qed *simp*
qed *simp*

lemma *set-lub-spmf*: $\text{set-spm}f\ \text{lub-spm}f = (\bigcup_{p \in Y}. \text{set-spm}f\ p)$ (**is** *?lhs = ?rhs*)
proof(*cases Y = \{\}*)
case [*simp*]: *False*
show *?thesis*
proof(*rule set-eqI*)


```

fix  $x$ 
have  $x \in ?lhs \longleftrightarrow \text{ennreal } (\text{spmf } \text{lub-spmf } x) > 0$ 
  by (simp-all add: in-set-spmf-iff-spmf less-le)
also have  $\dots \longleftrightarrow (\exists p \in Y. \text{ennreal } (\text{spmf } p \ x) > 0)$ 
  by (simp add: ennreal-spmf-lub-spmf less-SUP-iff)
also have  $\dots \longleftrightarrow x \in ?rhs$ 
  by (auto simp add: in-set-spmf-iff-spmf less-le)
finally show  $x \in ?lhs \longleftrightarrow x \in ?rhs$  .
qed
qed simp

```

lemma *emeasure-lub-spmf*:

```

assumes  $Y: Y \neq \{\}$ 
shows  $\text{emeasure } (\text{measure-spmf } \text{lub-spmf}) A = (\text{SUP } y:Y. \text{emeasure } (\text{measure-spmf } y) A)$ 
  (is  $?lhs = ?rhs$ )
proof –
  let  $?M = \text{count-space } (\text{set-spmf } \text{lub-spmf})$ 
  have  $?lhs = \int^+ x. \text{ennreal } (\text{spmf } \text{lub-spmf } x) * \text{indicator } A \ x \ \partial ?M$ 
    by (auto simp add: nn-integral-indicator[symmetric] nn-integral-measure-spmf')
  also have  $\dots = \int^+ x. (\text{SUP } y:Y. \text{ennreal } (\text{spmf } y \ x) * \text{indicator } A \ x) \ \partial ?M$ 
    unfolding ennreal-indicator[symmetric]
    by (simp add: spmf-lub-spmf assms ennreal-SUP[OF SUP-spmf-neq-top'] SUP-mult-right-ennreal)
  also from assms have  $\dots = (\text{SUP } y:Y. \int^+ x. \text{ennreal } (\text{spmf } y \ x) * \text{indicator } A \ x \ \partial ?M)$ 
    proof (rule nn-integral-monotone-convergence-SUP-countable)
      have  $(\lambda i \ x. \text{ennreal } (\text{spmf } i \ x) * \text{indicator } A \ x) \ ' Y = (\lambda f \ x. f \ x * \text{indicator } A \ x) \ ' (\lambda p \ x. \text{ennreal } (\text{spmf } p \ x)) \ ' Y$ 
        by (simp add: image-image)
      also have Complete-Partial-Order.chain op  $\leq \dots$  using chain-ord-spmf-eqD
        by (rule chain-imageI) (auto simp add: le-fun-def split: split-indicator)
      finally show Complete-Partial-Order.chain op  $\leq ((\lambda i \ x. \text{ennreal } (\text{spmf } i \ x) * \text{indicator } A \ x) \ ' Y)$  .
    qed simp
  also have  $\dots = (\text{SUP } y:Y. \int^+ x. \text{ennreal } (\text{spmf } y \ x) * \text{indicator } A \ x \ \partial \text{count-space } \text{UNIV})$ 
    by (auto simp add: nn-integral-count-space-indicator set-lub-spmf spmf-eq-0-set-spmf split: split-indicator intro!: SUP-cong nn-integral-cong)
  also have  $\dots = ?rhs$ 
    by (auto simp add: nn-integral-indicator[symmetric] nn-integral-measure-spmf)
  finally show ?thesis .
qed

```

lemma *measure-lub-spmf*:

```

assumes  $Y: Y \neq \{\}$ 
shows  $\text{measure } (\text{measure-spmf } \text{lub-spmf}) A = (\text{SUP } y:Y. \text{measure } (\text{measure-spmf } y) A)$  (is  $?lhs = ?rhs$ )
proof –
  have  $\text{ennreal } ?lhs = \text{ennreal } ?rhs$ 

```

```

    using emeasure-lub-spmf[OF assms] SUP-emeasure-spmf-neg-top[of A Y] Y
    unfolding measure-spmf.emeasure-eq-measure by(subst ennreal-SUP)
    moreover have  $0 \leq ?rhs$  using Y
    by(auto intro!: cSUP-upper2 bdd-aboveI[where M=1] measure-spmf.subprob-measure-le-1)
    ultimately show ?thesis by(simp)
qed

```

```

lemma weight-lub-spmf:
  assumes Y:  $Y \neq \{\}$ 
  shows weight-spmf lub-spmf = (SUP y:Y. weight-spmf y)
unfolding weight-spmf-def by(rule measure-lub-spmf) fact

```

```

lemma measure-spmf-lub-spmf:
  assumes Y:  $Y \neq \{\}$ 
  shows measure-spmf lub-spmf = (SUP p:Y. measure-spmf p) (is ?lhs = ?rhs)
proof(rule measure-eqI)
  from assms obtain p where  $p \in Y$  by auto
  from chain have chain': Complete-Partial-Order.chain op  $\leq$  (measure-spmf ‘
Y)
  by(rule chain-imageI)(rule ord-spmf-eqD-measure-spmf)
  show sets ?lhs = sets ?rhs
  using Y by (subst sets-SUP) auto
  show emeasure ?lhs A = emeasure ?rhs A for A
  using chain' Y p by (subst emeasure-SUP-chain) (auto simp: emeasure-lub-spmf)
qed

```

end

end

```

lemma partial-function-definitions-spmf: partial-function-definitions (ord-spmf op
=) lub-spmf
(is partial-function-definitions ?R -)
proof
  fix x show ?R x x by(simp add: ord-spmf-refl)
next
  fix x y z
  assume ?R x y ?R y z
  with transp-ord-option[OF transp-equality] show ?R x z by(rule transp-rel-pmf[THEN
transpD])
next
  fix x y
  assume ?R x y ?R y x
  thus  $x = y$ 
  by(rule rel-pmf-antisym)(simp-all add: reflp-ord-option transp-ord-option antisymp-ord-option)
next
  fix Y x
  assume Complete-Partial-Order.chain ?R Y  $x \in Y$ 
  then show ?R x (lub-spmf Y)

```

```

  by(rule lub-spmf-upper)
next
  fix Y z
  assume Complete-Partial-Order.chain ?R Y  $\wedge x. x \in Y \implies ?R x z$ 
  then show ?R (lub-spmf Y) z
    by(cases Y = {})(simp-all add: lub-spmf-least)
qed

lemma ccpo-spmf: class.ccpo lub-spmf (ord-spmf op =) (mk-less (ord-spmf op =))
by(rule ccpo partial-function-definitions-spmf)+

interpretation spmf: partial-function-definitions ord-spmf op = lub-spmf
  rewrites lub-spmf {}  $\equiv$  return-pmf None
by(rule partial-function-definitions-spmf) simp

declaration  $\langle$ Partial-Function.init spmf @{term spmf.fixp-fun}
  @{term spmf.mono-body} @{thm spmf.fixp-rule-uc} @{thm spmf.fixp-induct-uc}
  NONE $\rangle$ 

declare spmf.leq-refl[simp]
declare admissible-leI[OF ccpo-spmf, cont-intro]

abbreviation mono-spmf  $\equiv$  monotone (fun-ord (ord-spmf op =)) (ord-spmf op =)

lemma lub-spmf-const [simp]: lub-spmf {p} = p
by(rule spmf-eqI)(simp add: spmf-lub-spmf[OF ccpo.chain-singleton[OF ccpo-spmf]])

lemma bind-spmf-mono':
  assumes fg: ord-spmf op = f g
  and hk:  $\wedge x :: 'a. ord-spmf op = (h x) (k x)$ 
  shows ord-spmf op = (f  $\ggg$  h) (g  $\ggg$  k)
unfolding bind-spmf-def using assms(1)
by(rule rel-pmf-bindI)(auto split: option.split simp add: hk)

lemma bind-spmf-mono [partial-function-mono]:
  assumes mf: mono-spmf B and mg:  $\wedge y. mono-spmf (\lambda f. C y f)$ 
  shows mono-spmf ( $\lambda f. bind-spmf (B f) (\lambda y. C y f)$ )
proof (rule monotoneI)
  fix f g :: 'a  $\Rightarrow$  'b spmf
  assume fg: fun-ord (ord-spmf op =) f g
  with mf have ord-spmf op = (B f) (B g) by (rule monotoneD[of - - f g])
  moreover from mg have  $\wedge y'. ord-spmf op = (C y' f) (C y' g)$ 
  by (rule monotoneD) (rule fg)
  ultimately show ord-spmf op = (bind-spmf (B f) ( $\lambda y. C y f$ )) (bind-spmf (B
g) ( $\lambda y'. C y' g$ ))
  by(rule bind-spmf-mono')
qed

```

lemma *monotone-bind-spmf1*: *monotone* (*ord-spmf op =*) (*ord-spmf op =*) ($\lambda y.$
bind-spmf y g)

by(*rule monotoneI*)(*simp add: bind-spmf-mono' ord-spmf-reflI*)

lemma *monotone-bind-spmf2*:

assumes *g*: $\bigwedge x.$ *monotone ord* (*ord-spmf op =*) ($\lambda y.$ *g y x*)

shows *monotone ord* (*ord-spmf op =*) ($\lambda y.$ *bind-spmf p (g y)*)

by(*rule monotoneI*)(*auto intro: bind-spmf-mono' monotoneD[OF g] ord-spmf-reflI*)

lemma *bind-lub-spmf*:

assumes *chain*: *Complete-Partial-Order.chain* (*ord-spmf op =*) *Y*

shows *bind-spmf (lub-spmf Y) f = lub-spmf (($\lambda p.$ *bind-spmf p f*) ‘ *Y*)* (**is** *?lhs = ?rhs*)

proof(*cases Y = {}*)

case *Y: False*

show *?thesis*

proof(*rule spmf-eqI*)

fix *i*

have *chain'*: *Complete-Partial-Order.chain* *op* \leq ($(\lambda p x.$ *ennreal (spm f p x * spmf (f x) i)*) ‘ *Y*)

using *chain* **by**(*rule chain-imageI*)(*auto simp add: le-fun-def dest: ord-spmf-eq-leD intro: mult-right-mono*)

have *chain''*: *Complete-Partial-Order.chain* (*ord-spmf op =*) ($(\lambda p.$ *p \gg f*) ‘ *Y*)

using *chain* **by**(*rule chain-imageI*)(*auto intro!: monotoneI bind-spmf-mono' ord-spmf-reflI*)

let *?M* = *count-space (set-spmf (lub-spmf Y))*

have *ennreal (spm f ?lhs i) = $\int^+ x.$ *ennreal (spm f (lub-spmf Y) x) * *ennreal (spm f (f x) i) ∂ ?M***

by(*auto simp add: ennreal-spmf-lub-spmf ennreal-spmf-bind nn-integral-measure-spmf'*)

also have $\dots = \int^+ x.$ (*SUP p:Y. *ennreal (spm f p x * spmf (f x) i)* ∂ ?M*)

by(*subst ennreal-spmf-lub-spmf[OF chain Y]*)(*subst SUP-mult-right-ennreal, simp-all add: ennreal-mult Y*)

also have $\dots =$ (*SUP p:Y. $\int^+ x.$ *ennreal (spm f p x * spmf (f x) i) ∂ ?M**)

using *Y chain'* **by**(*rule nn-integral-monotone-convergence-SUP-countable*)

simp

also have $\dots =$ (*SUP p:Y. *ennreal (spm f (bind-spmf p f) i)**)

by(*auto simp add: ennreal-spmf-bind nn-integral-measure-spmf nn-integral-count-space-indicator set-lub-spmf[OF chain] in-set-spmf-iff-spmf ennreal-mult intro!: SUP-cong nn-integral-cong split: split-indicator*)

also have $\dots =$ *ennreal (spm f ?rhs i)* **using** *chain''* **by**(*simp add: ennreal-spmf-lub-spmf Y*)

finally show *spm f ?lhs i = spmf ?rhs i* **by** *simp*

qed

qed *simp*

lemma *map-lub-spmf*:

Complete-Partial-Order.chain (*ord-spmf op =*) *Y*

\implies *map-spmf f (lub-spmf Y) = lub-spmf (map-spmf f ‘ Y)*

unfolding *map-spmf-conv-bind-spmf*[*abs-def*] **by**(*simp add: bind-lub-spmf o-def*)

lemma *mcont-bind-spmf1*: *mcont lub-spmf (ord-spmf op =) lub-spmf (ord-spmf op =) (λy. bind-spmf y f)*

using *monotone-bind-spmf1* **by**(*rule mcontI*)(*rule contI, simp add: bind-lub-spmf*)

lemma *bind-lub-spmf2*:

assumes *chain*: *Complete-Partial-Order.chain ord Y*

and *g*: $\bigwedge y. \text{monotone } ord \text{ (ord-spmf op =) } (g y)$

shows *bind-spmf x (λy. lub-spmf (g y ‘ Y)) = lub-spmf ((λp. bind-spmf x (λy. g y p)) ‘ Y)*

(**is** *?lhs = ?rhs*)

proof(*cases Y = {}*)

case *Y: False*

show *?thesis*

proof(*rule spmf-eqI*)

fix *i*

have *chain'*: $\bigwedge y. \text{Complete-Partial-Order.chain (ord-spmf op =) } (g y ‘ Y)$

using *chain g [THEN monotoneD] by (rule chain-imageI)*

have *chain''*: *Complete-Partial-Order.chain op ≤ ((λp y. ennreal (spm f x y * spmf (g y p) i)) ‘ Y)*

using *chain by (rule chain-imageI)(auto simp add: le-fun-def dest: ord-spmf-eq-leD monotoneD[OF g] intro!: mult-left-mono)*

have *chain'''*: *Complete-Partial-Order.chain (ord-spmf op =) ((λp. bind-spmf x (λy. g y p)) ‘ Y)*

using *chain by (rule chain-imageI)(rule monotone-bind-spmf2[OF g, THEN monotoneD])*

have *ennreal (spm f ?lhs i) = $\int^+ y. (\text{SUP } p:Y. \text{ennreal (spm f x y * spmf (g y p) i)}) \partial \text{count-space (set-spmf x)}$*

by(*simp add: ennreal-spmf-bind ennreal-spmf-lub-spmf[OF chain'] Y nn-integral-measure-spmf' SUP-mult-left-ennreal ennreal-mult*)

also have $\dots = (\text{SUP } p:Y. \int^+ y. \text{ennreal (spm f x y * spmf (g y p) i)}) \partial \text{count-space (set-spmf x)}$

unfolding *nn-integral-measure-spmf'* **using** *Y chain''*

by(*rule nn-integral-monotone-convergence-SUP-countable*) *simp*

also have $\dots = (\text{SUP } p:Y. \text{ennreal (spm f (bind-spmf x (λy. g y p)) i)})$

by(*simp add: ennreal-spmf-bind nn-integral-measure-spmf' ennreal-mult*)

also have $\dots = \text{ennreal (spm f ?rhs i)}$ **using** *chain'''*

by(*auto simp add: ennreal-spmf-lub-spmf Y*)

finally show *spm f ?lhs i = spmf ?rhs i* **by** *simp*

qed

qed *simp*

lemma *mcont-bind-spmf [cont-intro]*:

assumes *f*: *mcont lub-spmf (ord-spmf op =) f*

and *g*: $\bigwedge y. \text{mcont lub-spmf (ord-spmf op =) } (g y)$

shows *mcont lub-spmf (ord-spmf op =) (λx. bind-spmf (f x) (λy. g y x))*

```

proof(rule spmf.mcont2mcont [OF - - f])
  fix z
  show mcont lub-spmf (ord-spmf op =) lub-spmf (ord-spmf op =) ( $\lambda x. \textit{bind-spmf}$ 
x ( $\lambda y. g y z$ ))
    by(rule mcont-bind-spmf1)
next
  fix x
  let ?f =  $\lambda z. \textit{bind-spmf}$  x ( $\lambda y. g y z$ )
  have monotone orda (ord-spmf op =) ?f using mcont-mono[OF g] by(rule
monotone-bind-spmf2)
  moreover have cont luba orda lub-spmf (ord-spmf op =) ?f
  proof(rule contI)
    fix Y
    assume chain: Complete-Partial-Order.chain orda Y and Y: Y  $\neq$  {}
    have bind-spmf x ( $\lambda y. g y (\textit{luba Y})$ ) = bind-spmf x ( $\lambda y. \textit{lub-spmf}$  (g y ‘ Y))
      by(rule bind-spmf-cong)(simp-all add: mcont-contD[OF g chain Y])
    also have ... = lub-spmf (( $\lambda p. x \gg= (\lambda y. g y p)$ ) ‘ Y) using chain
      by(rule bind-lub-spmf2)(rule mcont-mono[OF g])
    finally show bind-spmf x ( $\lambda y. g y (\textit{luba Y})$ ) = ... .
  qed
  ultimately show mcont luba orda lub-spmf (ord-spmf op =) ?f by(rule mcontI)
qed

```

```

lemma bind-pmf-mono [partial-function-mono]:
  ( $\bigwedge y. \textit{mono-spmf}$  ( $\lambda f. C y f$ ))  $\implies$  mono-spmf ( $\lambda f. \textit{bind-pmf}$  p ( $\lambda x. C x f$ ))
using bind-spmf-mono[of  $\lambda-. \textit{spmf-of-pmf}$  p C] by simp

```

```

lemma map-spmf-mono [partial-function-mono]: mono-spmf B  $\implies$  mono-spmf
( $\lambda g. \textit{map-spmf}$  f (B g))
unfolding map-spmf-conv-bind-spmf by(rule bind-spmf-mono) simp-all

```

```

lemma mcont-map-spmf [cont-intro]:
  mcont luba orda lub-spmf (ord-spmf op =) g
   $\implies$  mcont luba orda lub-spmf (ord-spmf op =) ( $\lambda x. \textit{map-spmf}$  f (g x))
unfolding map-spmf-conv-bind-spmf by(rule mcont-bind-spmf) simp-all

```

```

lemma monotone-set-spmf: monotone (ord-spmf op =) op  $\subseteq$  set-spmf
by(rule monotoneI)(rule ord-spmf-eqD-set-spmf)

```

```

lemma cont-set-spmf: cont lub-spmf (ord-spmf op =) Union op  $\subseteq$  set-spmf
by(rule contI)(subst set-lub-spmf; simp)

```

```

lemma mcont2mcont-set-spmf [THEN mcont2mcont, cont-intro]:
  shows mcont-set-spmf: mcont lub-spmf (ord-spmf op =) Union op  $\subseteq$  set-spmf
by(rule mcontI monotone-set-spmf cont-set-spmf)+

```

```

lemma monotone-spmf: monotone (ord-spmf op =) op  $\leq$  ( $\lambda p. \textit{spmf}$  p x)
by(rule monotoneI)(simp add: ord-spmf-eq-leD)

```

lemma *cont-spmf*: *cont lub-spmf (ord-spmf op =) Sup op ≤ (λp. spmf p x)*
by(rule *contI*)(*simp add: spmf-lub-spmf*)

lemma *mcont-spmf*: *mcont lub-spmf (ord-spmf op =) Sup op ≤ (λp. spmf p x)*
by(rule *mcontI monotone-spmf cont-spmf*)**+**

lemma *cont-ennreal-spmf*: *cont lub-spmf (ord-spmf op =) Sup op ≤ (λp. ennreal (spm f p x))*
by(rule *contI*)(*simp add: ennreal-spmf-lub-spmf*)

lemma *mcont2mcont-ennreal-spmf* [*THEN mcont2mcont, cont-intro*]:
shows *mcont-ennreal-spmf: mcont lub-spmf (ord-spmf op =) Sup op ≤ (λp. ennreal (spm f p x))*
by(rule *mcontI mono2mono-ennreal monotone-spmf cont-ennreal-spmf*)**+**

lemma *nn-integral-map-spmf* [*simp*]: *nn-integral (measure-spmf (map-spmf f p)) g = nn-integral (measure-spmf p) (g ∘ f)*
by(*auto 4 3 simp add: measure-spmf-def nn-integral-distr nn-integral-restrict-space intro: nn-integral-cong split: split-indicator*)

27.11.1 Admissibility of *rel-spmf*

lemma *rel-spmf-measureD*:

assumes *rel-spmf R p q*

shows *measure (measure-spmf p) A ≤ measure (measure-spmf q) {y. ∃ x ∈ A. R x y}* (**is** *?lhs ≤ ?rhs*)

proof –

have *?lhs = measure (measure-pmf p) (Some ‘A)* **by**(*simp add: measure-measure-spmf-conv-measure-pmf*)

also have *... ≤ measure (measure-pmf q) {y. ∃ x ∈ Some ‘A. rel-option R x y}*

using *assms* **by**(rule *rel-pmf-measureD*)

also have *... = ?rhs* **unfolding** *measure-measure-spmf-conv-measure-pmf*

by(rule *arg-cong2[where f=measure]*)(*auto simp add: option-rel-Some1*)

finally show *?thesis* .

qed

locale *rel-spmf-characterisation* =

assumes *rel-pmf-measureI*:

$\bigwedge(R :: 'a \text{ option} \Rightarrow 'b \text{ option} \Rightarrow \text{bool}) p q.$

$(\bigwedge A. \text{measure (measure-pmf p) } A \leq \text{measure (measure-pmf q) } \{y. \exists x \in A. R x y\})$

$\implies \text{rel-pmf } R p q$

— This assumption is shown to hold in general in the AFP entry *MFMC-Countable*.

begin

context *fixes* *R :: 'a ⇒ 'b ⇒ bool* **begin**

lemma *rel-spmf-measureI*:

assumes *eq1: $\bigwedge A. \text{measure (measure-spmf p) } A \leq \text{measure (measure-spmf q) }$*

```

{y.  $\exists x \in A. R x y$ }
  assumes eq2: weight-spmf q  $\leq$  weight-spmf p
  shows rel-spmf R p q
proof(rule rel-pmf-measureI)
  fix A :: 'a option set
  define A' where A' = the ' (A  $\cap$  range Some)
  define A'' where A'' = A  $\cap$  {None}
  have A: A = Some ' A'  $\cup$  A'' Some ' A'  $\cap$  A'' = {}
    unfolding A'-def A''-def by(auto  $\&$   $\exists$  intro: rev-image-eqI)
  have measure (measure-pmf p) A = measure (measure-pmf p) (Some ' A') +
measure (measure-pmf p) A''
    by(simp add: A measure-pmf.finite-measure-Union)
  also have measure (measure-pmf p) (Some ' A') = measure (measure-spmf p)
A'
    by(simp add: measure-measure-spmf-conv-measure-pmf)
  also have ...  $\leq$  measure (measure-spmf q) {y.  $\exists x \in A'. R x y$ } by(rule eq1)
  also (ord-eq-le-trans[OF - add-right-mono])
  have ... = measure (measure-pmf q) {y.  $\exists x \in A'. \text{rel-option } R \text{ (Some } x \text{) } y$ }
    unfolding measure-measure-spmf-conv-measure-pmf
    by(rule arg-cong2[where f=measure])(auto simp add: A'-def option-rel-Some1)
  also
  { have weight-spmf p  $\leq$  measure (measure-spmf q) {y.  $\exists x. R x y$ }
    using eq1[of UNIV] unfolding weight-spmf-def by simp
    also have ...  $\leq$  weight-spmf q unfolding weight-spmf-def
      by(rule measure-spmf.finite-measure-mono) simp-all
    finally have weight-spmf p = weight-spmf q using eq2 by simp }
  then have measure (measure-pmf p) A'' = measure (measure-pmf q) (if None
 $\in$  A then {None} else {})
    unfolding A''-def by(simp add: pmf-None-eq-weight-spmf measure-pmf-single)
  also have measure (measure-pmf q) {y.  $\exists x \in A'. \text{rel-option } R \text{ (Some } x \text{) } y$ } + ...
= measure (measure-pmf q) {y.  $\exists x \in A. \text{rel-option } R x y$ }
    by(subst measure-pmf.finite-measure-Union[symmetric])
    (auto  $\&$   $\exists$  intro!: arg-cong2[where f=measure] simp add: option-rel-Some1
option-rel-Some2 A'-def intro: rev-bexI elim: option.rel-cases)
    finally show measure (measure-pmf p) A  $\leq$  ... .
qed

```

lemma *admissible-rel-spmf*:

```

ccpo.admissible (prod-lub lub-spmf lub-spmf) (rel-prod (ord-spmf op =) (ord-spmf
op =)) (case-prod (rel-spmf R))
  (is ccpo.admissible ?lub ?ord ?P)

```

proof(*rule ccpo.admissibleI*)

fix Y

assume *chain: Complete-Partial-Order.chain ?ord Y*

and Y: Y \neq {}

and R: $\forall (p, q) \in Y. \text{rel-spmf } R p q$

from R have R: $\bigwedge p q. (p, q) \in Y \implies \text{rel-spmf } R p q$ **by** *auto*

have *chain1: Complete-Partial-Order.chain* (*ord-spmf op =*) (*fst* ' Y)

and *chain2: Complete-Partial-Order.chain* (*ord-spmf op =*) (*snd* ' Y)


```

using chain by(rule chain-imageI; clarsimp)+
from Y have Y1: fst ‘ Y ≠ {} and Y2: snd ‘ Y ≠ {} by auto

have rel-spmf R (lub-spmf (fst ‘ Y)) (lub-spmf (snd ‘ Y))
proof(rule rel-spmf-measureI)
  show weight-spmf (lub-spmf (snd ‘ Y)) ≤ weight-spmf (lub-spmf (fst ‘ Y))
  by(auto simp add: weight-lub-spmf chain1 chain2 Y rel-spmf-weightD[OF R,
symmetric] intro!: cSUP-least intro: cSUP-upper2[OF bdd-aboveI2[OF weight-spmf-le-1]])

  fix A
  have measure (measure-spmf (lub-spmf (fst ‘ Y))) A = (SUP y:fst ‘ Y. measure
(measure-spmf y) A)
  using chain1 Y1 by(rule measure-lub-spmf)
  also have ... ≤ (SUP y:snd ‘ Y. measure (measure-spmf y) {y. ∃ x∈A. R x
y}) using Y1
  by(rule cSUP-least)(auto intro!: cSUP-upper2[OF bdd-aboveI2[OF measure-spmf.subprob-measure-le-1]]
rel-spmf-measureD R)
  also have ... = measure (measure-spmf (lub-spmf (snd ‘ Y))) {y. ∃ x∈A. R
x y}
  using chain2 Y2 by(rule measure-lub-spmf[symmetric])
  finally show measure (measure-spmf (lub-spmf (fst ‘ Y))) A ≤ ... .
qed
then show ?P (?lub Y) by(simp add: prod-lub-def)
qed

```

```

lemma admissible-rel-spmf-mcont [cont-intro]:
  [[ mcont lub ord lub-spmf (ord-spmf op =) f; mcont lub ord lub-spmf (ord-spmf
op =) g ]
  ⇒ ccpo.admissible lub ord (λx. rel-spmf R (f x) (g x))
by(rule admissible-subst[OF admissible-rel-spmf, where f=λx. (f x, g x), simpli-
fied])(rule mcont-Pair)

```

```

context includes lifting-syntax
begin

```

```

lemma fixp-spmf-parametric':
  assumes f: λx. monotone (ord-spmf op =) (ord-spmf op =) F
  and g: λx. monotone (ord-spmf op =) (ord-spmf op =) G
  and param: (rel-spmf R ===> rel-spmf R) F G
  shows (rel-spmf R) (ccpo.fixp lub-spmf (ord-spmf op =) F) (ccpo.fixp lub-spmf
(ord-spmf op =) G)
by(rule parallel-fixp-induct[OF ccpo-spmf ccpo-spmf - f g])(auto intro: param[THEN
rel-funD])

```

```

lemma fixp-spmf-parametric:
  assumes f: λx. mono-spmf (λf. F f x)
  and g: λx. mono-spmf (λf. G f x)
  and param: ((A ===> rel-spmf R) ===> A ===> rel-spmf R) F G
  shows (A ===> rel-spmf R) (spmfixp-fun F) (spmfixp-fun G)

```

```

using f g
proof(rule parallel-fixp-induct-1-1[OF partial-function-definitions-spmf partial-function-definitions-spmf
- - reflexive reflexive, where P=(A ===> rel-spmf R)])
  show ccpo.admissible (prod-lub (fun-lub lub-spmf) (fun-lub lub-spmf)) (rel-prod
(fun-ord (ord-spmf op =)) (fun-ord (ord-spmf op =))) (λx. (A ===> rel-spmf R)
(fst x) (snd x))
    unfolding rel-fun-def
    apply(rule admissible-all admissible-imp admissible-rel-spmf-mcont)+
    apply(rule spmf.mcont2mcont[OF mcont-call])
    apply(rule mcont-fst)
    apply(rule spmf.mcont2mcont[OF mcont-call])
    apply(rule mcont-snd)
  done
show (A ===> rel-spmf R) (λ-. lub-spmf {}) (λ-. lub-spmf {}) by auto
show (A ===> rel-spmf R) (F f) (G g) if (A ===> rel-spmf R) f g for f g
  using that by(rule rel-funD[OF param])
qed

end

end

end

end

```

27.12 Restrictions on spmfs

definition *restrict-spmf* :: 'a spmf ⇒ 'a set ⇒ 'a spmf (**infixl** 1 110)
where $p \upharpoonright A = \text{map-pmf } (\lambda x. x \gg= (\lambda y. \text{if } y \in A \text{ then Some } y \text{ else None})) p$

lemma *set-restrict-spmf* [simp]: $\text{set-spmf } (p \upharpoonright A) = \text{set-spmf } p \cap A$
by(fastforce simp add: restrict-spmf-def set-spmf-def split: bind-splits if-split-asm)

lemma *restrict-map-spmf*: $\text{map-spmf } f p \upharpoonright A = \text{map-spmf } f (p \upharpoonright (f^{-1} A))$
by(simp add: restrict-spmf-def pmf.map-comp o-def map-option-bind bind-map-option
if-distrib cong del: if-weak-cong)

lemma *restrict-restrict-spmf* [simp]: $p \upharpoonright A \upharpoonright B = p \upharpoonright (A \cap B)$
by(auto simp add: restrict-spmf-def pmf.map-comp o-def intro!: pmf.map-comp
bind-option-cong)

lemma *restrict-spmf-empty* [simp]: $p \upharpoonright \{\} = \text{return-pmf None}$
by(simp add: restrict-spmf-def)

lemma *restrict-spmf-UNIV* [simp]: $p \upharpoonright \text{UNIV} = p$
by(simp add: restrict-spmf-def)

lemma *spmfs-restrict-spmf-outside* [simp]: $x \notin A \implies \text{spmfs } (p \upharpoonright A) x = 0$
by(simp add: spmf-eq-0-set-spmf)

lemma *emeasure-restrict-spmf* [simp]:
 $\text{emeasure } (\text{measure-spmf } (p \upharpoonright A)) X = \text{emeasure } (\text{measure-spmf } p) (X \cap A)$
by(*auto simp add: restrict-spmf-def measure-spmf-def emeasure-distr measurable-restrict-space1*
emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]
split: bind-splits if-split-asm)

lemma *measure-restrict-spmf* [simp]:
 $\text{measure } (\text{measure-spmf } (p \upharpoonright A)) X = \text{measure } (\text{measure-spmf } p) (X \cap A)$
using *emeasure-restrict-spmf[of p A X]*
by(*simp only: measure-spmf.emeasure-eq-measure ennreal-inj measure-nonneg*)

lemma *spmf-restrict-spmf*: $\text{spmf } (p \upharpoonright A) x = (\text{if } x \in A \text{ then } \text{spmf } p x \text{ else } 0)$
by(*simp add: spmf-conv-measure-spmf*)

lemma *spmf-restrict-spmf-inside* [simp]: $x \in A \implies \text{spmf } (p \upharpoonright A) x = \text{spmf } p x$
by(*simp add: spmf-restrict-spmf*)

lemma *pmf-restrict-spmf-None*: $\text{pmf } (p \upharpoonright A) \text{None} = \text{pmf } p \text{None} + \text{measure}$
 $(\text{measure-spmf } p) (-A)$

proof –

have [simp]: $\text{None} \notin \text{Some } (-A)$ **by** *auto*

have $(\lambda x. x \gg (\lambda y. \text{if } y \in A \text{ then } \text{Some } y \text{ else } \text{None})) - \{ \text{None} \} = \{ \text{None} \} \cup$
 $(\text{Some } (-A))$

by(*auto split: bind-splits if-split-asm*)

then show *?thesis unfolding ereal.inject[symmetric]*

by(*simp add: restrict-spmf-def ennreal-pmf-map emeasure-pmf-single del: ereal.inject*)

(*simp add: pmf.rep-eq measure-pmf.finite-measure-Union[symmetric] measure-measure-spmf-conv-measure*
measure-pmf.emeasure-eq-measure)

qed

lemma *restrict-spmf-trivial*: $(\bigwedge x. x \in \text{set-spmf } p \implies x \in A) \implies p \upharpoonright A = p$
by(*rule spmf-eqI*)(*auto simp add: spmf-restrict-spmf spmf-eq-0-set-spmf*)

lemma *restrict-spmf-trivial'*: $\text{set-spmf } p \subseteq A \implies p \upharpoonright A = p$
by(*rule restrict-spmf-trivial*) *blast*

lemma *restrict-return-spmf*: $\text{return-spmf } x \upharpoonright A = (\text{if } x \in A \text{ then } \text{return-spmf } x$
 $\text{else } \text{return-pmf } \text{None})$
by(*simp add: restrict-spmf-def*)

lemma *restrict-return-spmf-inside* [simp]: $x \in A \implies \text{return-spmf } x \upharpoonright A = \text{return-spmf}$
 x
by(*simp add: restrict-return-spmf*)

lemma *restrict-return-spmf-outside* [simp]: $x \notin A \implies \text{return-spmf } x \upharpoonright A = \text{return-pmf}$
 None
by(*simp add: restrict-return-spmf*)

lemma *restrict-spmf-return-pmf-None* [simp]: $\text{return-pmf } \text{None} \upharpoonright A = \text{return-pmf}$

None

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

lemma *restrict-bind-pmf*: $\text{bind-pmf } p \text{ } g \upharpoonright A = p \ggg (\lambda x. g \ x \upharpoonright A)$

by(*simp add: restrict-spmf-def map-bind-pmf o-def*)

lemma *restrict-bind-spmf*: $\text{bind-spmf } p \text{ } g \upharpoonright A = p \ggg (\lambda x. g \ x \upharpoonright A)$

by(*auto simp add: bind-spmf-def restrict-bind-pmf cong del: option.case-cong-weak cong: option.case-cong intro!: bind-pmf-cong split: option.split*)

lemma *bind-restrict-pmf*: $\text{bind-pmf } (p \upharpoonright A) \text{ } g = p \ggg (\lambda x. \text{if } x \in \text{Some } \text{' } A \text{ then } g \ x \text{ else } g \ \text{None})$

by(*auto simp add: restrict-spmf-def bind-map-pmf fun-eq-iff split: bind-split intro: arg-cong2[where f=bind-pmf]*)

lemma *bind-restrict-spmf*: $\text{bind-spmf } (p \upharpoonright A) \text{ } g = p \ggg (\lambda x. \text{if } x \in A \text{ then } g \ x \text{ else return-pmf } \text{None})$

by(*auto simp add: bind-spmf-def bind-restrict-pmf fun-eq-iff intro: arg-cong2[where f=bind-pmf] split: option.split*)

lemma *spmfm-map-restrict*: $\text{spmfm } (\text{map-spmfm } \text{fst } (p \upharpoonright (\text{snd } - \text{' } \{y\}))) \text{ } x = \text{spmfm } p \text{ } (x, y)$

by(*subst spmf-map*)(*auto intro: arg-cong2[where f=measure] simp add: spmf-conv-measure-spmfm*)

lemma *measure-eqI-restrict-spmf*:

assumes *rel-spmfm R* (*restrict-spmfm p A*) (*restrict-spmfm q B*)

shows $\text{measure } (\text{measure-spmfm } p) \text{ } A = \text{measure } (\text{measure-spmfm } q) \text{ } B$

proof –

from *assms* **have** $\text{weight-spmfm } (\text{restrict-spmfm } p \text{ } A) = \text{weight-spmfm } (\text{restrict-spmfm } q \text{ } B)$ **by**(*rule rel-spmfm-weightD*)

thus *?thesis* **by**(*simp add: weight-spmfm-def*)

qed

27.13 Subprobability distributions of sets

definition *spmfm-of-set* :: $\text{' } a \text{ set} \Rightarrow \text{' } a \text{ spmf}$

where

$\text{spmfm-of-set } A = (\text{if } \text{finite } A \wedge A \neq \{\} \text{ then } \text{spmfm-of-pmf } (\text{pmfm-of-set } A) \text{ else return-pmf } \text{None})$

lemma *spmfm-of-set*: $\text{spmfm } (\text{spmfm-of-set } A) \text{ } x = \text{indicator } A \text{ } x / \text{card } A$

by(*auto simp add: spmf-of-set-def*)

lemma *pmfm-spmfm-of-set-None* [*simp*]: $\text{pmfm } (\text{spmfm-of-set } A) \text{ } \text{None} = \text{indicator } \{A. \text{infinite } A \vee A = \{\}\} \text{ } A$

by(*simp add: spmf-of-set-def*)

lemma *set-spmfm-of-set*: $\text{set-spmfm } (\text{spmfm-of-set } A) = (\text{if } \text{finite } A \text{ then } A \text{ else } \{\})$

by(*simp add: spmf-of-set-def*)

lemma *set-spmf-of-set-finite* [simp]: $\text{finite } A \implies \text{set-spmf } (\text{spmf-of-set } A) = A$
by(simp add: set-spmf-of-set)

lemma *spmf-of-set-singleton*: $\text{spmf-of-set } \{x\} = \text{return-spmf } x$
by(simp add: spmf-of-set-def pmf-of-set-singleton)

lemma *map-spmf-of-set-inj-on* [simp]:
 $\text{inj-on } f \ A \implies \text{map-spmf } f \ (\text{spmf-of-set } A) = \text{spmf-of-set } (f \ ' \ A)$
by(auto simp add: spmf-of-set-def map-pmf-of-set-inj dest: finite-imageD)

lemma *spmf-of-pmf-pmf-of-set* [simp]:
 $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{spmf-of-pmf } (\text{pmf-of-set } A) = \text{spmf-of-set } A$
by(simp add: spmf-of-set-def)

lemma *weight-spmf-of-set*:
 $\text{weight-spmf } (\text{spmf-of-set } A) = (\text{if } \text{finite } A \wedge A \neq \{\} \text{ then } 1 \text{ else } 0)$
by(auto simp only: spmf-of-set-def weight-spmf-of-pmf weight-return-pmf-None split: if-split)

lemma *weight-spmf-of-set-finite* [simp]: $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{weight-spmf } (\text{spmf-of-set } A) = 1$
by(simp add: weight-spmf-of-set)

lemma *weight-spmf-of-set-infinite* [simp]: $\text{infinite } A \implies \text{weight-spmf } (\text{spmf-of-set } A) = 0$
by(simp add: weight-spmf-of-set)

lemma *measure-spmf-spmf-of-set*:
 $\text{measure-spmf } (\text{spmf-of-set } A) = (\text{if } \text{finite } A \wedge A \neq \{\} \text{ then } \text{measure-pmf } (\text{pmf-of-set } A) \text{ else } \text{null-measure } (\text{count-space } \text{UNIV}))$
by(simp add: spmf-of-set-def del: spmf-of-pmf-pmf-of-set)

lemma *emeasure-spmf-of-set*:
 $\text{emeasure } (\text{measure-spmf } (\text{spmf-of-set } S)) \ A = \text{card } (S \cap A) / \text{card } S$
by(auto simp add: measure-spmf-spmf-of-set emeasure-pmf-of-set)

lemma *measure-spmf-of-set*:
 $\text{measure } (\text{measure-spmf } (\text{spmf-of-set } S)) \ A = \text{card } (S \cap A) / \text{card } S$
by(auto simp add: measure-spmf-spmf-of-set measure-pmf-of-set)

lemma *nn-integral-spmf-of-set*: $\text{nn-integral } (\text{measure-spmf } (\text{spmf-of-set } A)) \ f = \text{sum } f \ A / \text{card } A$
by(cases finite A)(auto simp add: spmf-of-set-def nn-integral-pmf-of-set card-gt-0-iff simp del: spmf-of-pmf-pmf-of-set)

lemma *integral-spmf-of-set*: $\text{integral}^L (\text{measure-spmf } (\text{spmf-of-set } A)) \ f = \text{sum } f \ A / \text{card } A$
by(clarsimp simp add: spmf-of-set-def integral-pmf-of-set card-gt-0-iff simp del:

spmf-of-pmf-pmf-of-set)

notepad begin — *pmf-of-set* is not fully parametric.

```

define R :: nat ⇒ nat ⇒ bool where R x y ↔ (x ≠ 0 → y = 0) for x y
define A :: nat set where A = {0, 1}
define B :: nat set where B = {0, 1, 2}
have rel-set R A B unfolding R-def[abs-def] A-def B-def rel-set-def by auto
have ¬ rel-pmf R (pmf-of-set A) (pmf-of-set B)
proof
  assume rel-pmf R (pmf-of-set A) (pmf-of-set B)
  then obtain pq where pq: ∧x y. (x, y) ∈ set-pmf pq ⇒ R x y
    and 1: map-pmf fst pq = pmf-of-set A
    and 2: map-pmf snd pq = pmf-of-set B
    by cases auto
  have pmf (pmf-of-set B) 1 = 1 / 3 by (simp add: B-def)
  have pmf (pmf-of-set B) 2 = 1 / 3 by (simp add: B-def)

  have 2 / 3 = pmf (pmf-of-set B) 1 + pmf (pmf-of-set B) 2 by (simp add:
  B-def)
  also have ... = measure (measure-pmf (pmf-of-set B)) ({1} ∪ {2})
  by (subst measure-pmf.finite-measure-Union) (simp-all add: measure-pmf-single)
  also have ... = emeasure (measure-pmf pq) (snd - ' {2, 1})
  unfolding 2[symmetric] measure-pmf.emeasure-eq-measure[symmetric] by (simp)
  also have ... = emeasure (measure-pmf pq) {(0, 2), (0, 1)}
    by (rule emeasure-eq-AE) (auto simp add: AE-measure-pmf-iff R-def dest!: pq)
  also have ... ≤ emeasure (measure-pmf pq) (fst - ' {0})
    by (rule emeasure-mono) auto
  also have ... = emeasure (measure-pmf (pmf-of-set A)) {0}
    unfolding 1[symmetric] by simp
  also have ... = pmf (pmf-of-set A) 0
    by (simp add: measure-pmf-single measure-pmf.emeasure-eq-measure)
  also have pmf (pmf-of-set A) 0 = 1 / 2 by (simp add: A-def)
  finally show False by (subst (asm) ennreal-le-iff; simp)
qed
end

```

lemma *rel-pmf-of-set-bij*:

```

assumes f: bij-betw f A B
and A: A ≠ {} finite A
and B: B ≠ {} finite B
and R: ∧x. x ∈ A ⇒ R x (f x)
shows rel-pmf R (pmf-of-set A) (pmf-of-set B)
proof (rule pmf.rel-mono-strong)
  define AB where AB = (λx. (x, f x)) ' A
  define R' where R' x y ↔ (x, y) ∈ AB for x y
  have (x, y) ∈ AB if (x, y) ∈ set-pmf (pmf-of-set AB) for x y
    using that by (auto simp add: AB-def A)
  moreover have map-pmf fst (pmf-of-set AB) = pmf-of-set A
    by (simp add: AB-def map-pmf-of-set-inj[symmetric] inj-on-def A pmf.map-comp

```

o-def)
moreover
from f **have** $[simp]: inj\text{-}on\ f\ A$ **by**(rule *bij-betw-imp-inj-on*)
from f **have** $[simp]: f\ 'A = B$ **by**(rule *bij-betw-imp-surj-on*)
have $map\text{-}pmf\ snd\ (pmf\text{-}of\text{-}set\ AB) = pmf\text{-}of\text{-}set\ B$
by(*simp add: AB-def map-pmf-of-set-inj[symmetric] inj-on-def A pmf.map-comp*
o-def)
(*simp add: map-pmf-of-set-inj A*)
ultimately show $rel\text{-}pmf\ (\lambda x\ y. (x, y) \in AB)\ (pmf\text{-}of\text{-}set\ A)\ (pmf\text{-}of\text{-}set\ B) ..$
qed(*auto intro: R*)

lemma *rel-spmf-of-set-bij*:
assumes $f: bij\text{-}betw\ f\ A\ B$
and $R: \bigwedge x. x \in A \implies R\ x\ (f\ x)$
shows $rel\text{-}spm\ f\ R\ (spm\text{-}of\text{-}set\ A)\ (spm\text{-}of\text{-}set\ B)$
proof –
have $finite\ A \longleftrightarrow finite\ B$ **using** f **by**(rule *bij-betw-finite*)
moreover have $A = \{\}\longleftrightarrow B = \{\}$ **using** f **by**(*auto dest: bij-betw-empty2*
bij-betw-empty1)
ultimately show *?thesis* **using** *assms*
by(*auto simp add: spmf-of-set-def simp del: spmf-of-pmf-pmf-of-set intro: rel-pmf-of-set-bij*)
qed

context includes *lifting-syntax*
begin

lemma *rel-spmf-of-set*:
assumes *bi-unique R*
shows $(rel\text{-}set\ R \implies rel\text{-}spm\ f\ R)\ spm\text{-}of\text{-}set\ spm\text{-}of\text{-}set$
proof
fix $A\ B$
assume $R: rel\text{-}set\ R\ A\ B$
with *assms* **obtain** f **where** $bij\text{-}betw\ f\ A\ B$ **and** $f: \bigwedge x. x \in A \implies R\ x\ (f\ x)$
by(*auto dest: bi-unique-rel-set-bij-betw*)
then show $rel\text{-}spm\ f\ R\ (spm\text{-}of\text{-}set\ A)\ (spm\text{-}of\text{-}set\ B)$ **by**(rule *rel-spmf-of-set-bij*)
qed

end

lemma *map-mem-spmf-of-set*:
assumes $finite\ B\ B \neq \{\}$
shows $map\text{-}spm\ f\ (\lambda x. x \in A)\ (spm\text{-}of\text{-}set\ B) = spm\text{-}of\text{-}pmf\ (bernoulli\text{-}pmf\ (card\ (A \cap B) / card\ B))$
(*is ?lhs = ?rhs*)
proof(rule *spm-f-eqI*)
fix i
have $ennreal\ (spm\ f\ ?lhs\ i) = card\ (B \cap (\lambda x. x \in A) - \{i\}) / (card\ B)$
by(*subst ennreal-spmf-map*)(*simp add: measure-spmf-spmf-of-set assms emeasure-pmf-of-set*)
also have $\dots = (if\ i\ then\ card\ (B \cap A) / card\ B\ else\ card\ (B - A) / card\ B)$

by(*auto intro: arg-cong*[**where** $f = \text{card}$])
also have $\dots = (\text{if } i \text{ then } \text{card } (B \cap A) / \text{card } B \text{ else } (\text{card } B - \text{card } (B \cap A)) / \text{card } B)$
by(*auto simp add: card-Diff-subset-Int assms*)
also have $\dots = \text{ennreal } (\text{spmf } ?\text{rhs } i)$
by(*simp add: assms card-gt-0-iff field-simps card-mono Int-commute-of-nat-diff*)
finally show $\text{spmf } ?\text{lhs } i = \text{spmf } ?\text{rhs } i$ **by** *simp*
qed

abbreviation $\text{coin-spmf} :: \text{bool } \text{spmf}$
where $\text{coin-spmf} \equiv \text{spmf-of-set } \text{UNIV}$

lemma $\text{map-eq-const-coin-spmf}: \text{map-spmf } (op = c) \text{ coin-spmf} = \text{coin-spmf}$
proof –
have $\text{inj } (op \longleftrightarrow c) \text{ range } (op \longleftrightarrow c) = \text{UNIV}$ **by**(*auto intro: inj-onI*)
then show *?thesis* **by** *simp*
qed

lemma $\text{bind-coin-spmf-eq-const}: \text{coin-spmf} \ggg (\lambda x :: \text{bool}. \text{return-spmf } (b = x)) = \text{coin-spmf}$
using $\text{map-eq-const-coin-spmf}$ **unfolding** $\text{map-spmf-conv-bind-spmf}$ **by** *simp*

lemma $\text{bind-coin-spmf-eq-const}' : \text{coin-spmf} \ggg (\lambda x :: \text{bool}. \text{return-spmf } (x = b)) = \text{coin-spmf}$
by(*rewrite in - =* $\sqsupset \text{bind-coin-spmf-eq-const}[\text{symmetric, of } b]$)(*auto intro: bind-spmf-cong*)

27.14 Losslessness

definition $\text{lossless-spmf} :: 'a \text{ spmf} \Rightarrow \text{bool}$
where $\text{lossless-spmf } p \longleftrightarrow \text{weight-spmf } p = 1$

lemma $\text{lossless-iff-pmf-None}: \text{lossless-spmf } p \longleftrightarrow \text{pmf } p \text{ None} = 0$
by(*simp add: lossless-spmf-def pmf-None-eq-weight-spmf*)

lemma $\text{lossless-return-spmf [iff]}: \text{lossless-spmf } (\text{return-spmf } x)$
by(*simp add: lossless-iff-pmf-None*)

lemma $\text{lossless-return-pmf-None [iff]}: \neg \text{lossless-spmf } (\text{return-pmf } \text{None})$
by(*simp add: lossless-iff-pmf-None*)

lemma $\text{lossless-map-spmf [simp]}: \text{lossless-spmf } (\text{map-spmf } f p) \longleftrightarrow \text{lossless-spmf } p$
by(*auto simp add: lossless-iff-pmf-None pmf-eq-0-set-pmf*)

lemma $\text{lossless-bind-spmf [simp]}:$
 $\text{lossless-spmf } (p \ggg f) \longleftrightarrow \text{lossless-spmf } p \wedge (\forall x \in \text{set-spmf } p. \text{lossless-spmf } (f x))$
by(*simp add: lossless-iff-pmf-None pmf-bind-spmf-None add-nonneg-eq-0-iff integral-nonneg-AE integral-nonneg-eq-0-iff-AE measure-spmf.integrable-const-bound*[**where** $B=1$] *pmf-le-1*)

lemma *lossless-weight-spmfD*: $\text{lossless-spmf } p \implies \text{weight-spmf } p = 1$
by(*simp add: lossless-spmf-def*)

lemma *lossless-iff-set-pmf-None*:
 $\text{lossless-spmf } p \iff \text{None} \notin \text{set-pmf } p$
by (*simp add: lossless-iff-pmf-None pmf-eq-0-set-pmf*)

lemma *lossless-spmf-of-set* [*simp*]: $\text{lossless-spmf } (\text{spmof-of-set } A) \iff \text{finite } A \wedge A \neq \{\}$
by(*auto simp add: lossless-spmf-def weight-spmf-of-set*)

lemma *lossless-spmf-spmof-of-spmf* [*simp*]: $\text{lossless-spmf } (\text{spmof-of-pmf } p)$
by(*simp add: lossless-spmf-def*)

lemma *lossless-spmf-bind-pmf* [*simp*]:
 $\text{lossless-spmf } (\text{bind-pmf } p f) \iff (\forall x \in \text{set-pmf } p. \text{lossless-spmf } (f x))$
by(*simp add: lossless-iff-pmf-None pmf-bind integral-nonneg-AE integral-nonneg-eq-0-iff-AE measure-pmf.integrable-const-bound[where B=1] AE-measure-pmf-iff pmf-le-1*)

lemma *lossless-spmf-conv-spmof-of-pmf*: $\text{lossless-spmf } p \iff (\exists p'. p = \text{spmof-of-pmf } p')$

proof

assume *lossless-spmf p*
hence *: $\bigwedge y. y \in \text{set-pmf } p \implies \exists x. y = \text{Some } x$
by(*case-tac y*)(*simp-all add: lossless-iff-set-pmf-None*)

let *?p = map-pmf the p*
have *p = spmof-of-pmf ?p*
proof(*rule spmf-eqI*)
fix *i*

have $\text{ennreal } (\text{pmf } (\text{map-pmf } \text{the } p) i) = \int^+ x. \text{indicator } (\text{the } -' \{i\}) x \partial p$
by(*simp add: ennreal-pmf-map*)
also have $\dots = \int^+ x. \text{indicator } \{i\} x \partial \text{measure-spmf } p$ **unfolding** *measure-spmf-def*
by(*subst nn-integral-distr*)(*auto simp add: nn-integral-restrict-space AE-measure-pmf-iff simp del: nn-integral-indicator intro!: nn-integral-cong-AE split: split-indicator dest!: **)

also have $\dots = \text{spmof } p i$ **by**(*simp add: emeasure-spmf-single*)

finally show $\text{spmof } p i = \text{spmof } (\text{spmof-of-pmf } ?p) i$ **by** *simp*

qed

thus $\exists p'. p = \text{spmof-of-pmf } p' ..$

qed *auto*

lemma *spmof-False-conv-True*: $\text{lossless-spmf } p \implies \text{spmof } p \text{ False} = 1 - \text{spmof } p \text{ True}$

by(*clarsimp simp add: lossless-spmf-conv-spmof-of-pmf pmf-False-conv-True*)

lemma *spmof-True-conv-False*: $\text{lossless-spmf } p \implies \text{spmof } p \text{ True} = 1 - \text{spmof } p \text{ False}$
by(*simp add: spmf-False-conv-True*)

lemma *bind-eq-return-spmf*:

$bind\text{-}spmf\ p\ f = return\text{-}spmf\ x \iff (\forall y \in set\text{-}spmf\ p. f\ y = return\text{-}spmf\ x) \wedge lossless\text{-}spmf\ p$

by(*auto simp add: bind-spmf-def bind-eq-return-pmf in-set-spmf lossless-iff-pmf-None pmf-eq-0-set-pmf iff del: not-None-eq split: option.split*)

lemma *rel-spmf-return-spmf2*:

$rel\text{-}spmf\ R\ p\ (return\text{-}spmf\ x) \iff lossless\text{-}spmf\ p \wedge (\forall a \in set\text{-}spmf\ p. R\ a\ x)$

by(*auto simp add: lossless-iff-set-pmf-None rel-pmf-return-pmf2 option-rel-Some2 in-set-spmf, metis in-set-spmf not-None-eq*)

lemma *rel-spmf-return-spmf1*:

$rel\text{-}spmf\ R\ (return\text{-}spmf\ x)\ p \iff lossless\text{-}spmf\ p \wedge (\forall a \in set\text{-}spmf\ p. R\ x\ a)$

using *rel-spmf-return-spmf2*[of R^{-1-1}] **by**(*simp add: spmf-rel-conversep*)

lemma *rel-spmf-bindI1*:

assumes $f: \bigwedge x. x \in set\text{-}spmf\ p \implies rel\text{-}spmf\ R\ (f\ x)\ q$

and $p: lossless\text{-}spmf\ p$

shows $rel\text{-}spmf\ R\ (bind\text{-}spmf\ p\ f)\ q$

proof –

fix $x :: 'a$

have $rel\text{-}spmf\ R\ (bind\text{-}spmf\ p\ f)\ (bind\text{-}spmf\ (return\text{-}spmf\ x)\ (\lambda-. q))$

by(*rule rel-spmf-bindI*[**where** $R = \lambda x -. x \in set\text{-}spmf\ p$])(*simp-all add: rel-spmf-return-spmf2 p f*)

then show *?thesis* **by** *simp*

qed

lemma *rel-spmf-bindI2*:

$\llbracket \bigwedge x. x \in set\text{-}spmf\ q \implies rel\text{-}spmf\ R\ p\ (f\ x); lossless\text{-}spmf\ q \rrbracket$

$\implies rel\text{-}spmf\ R\ p\ (bind\text{-}spmf\ q\ f)$

using *rel-spmf-bindI1*[of $q\ conversep\ R\ f\ p$] **by**(*simp add: spmf-rel-conversep*)

27.15 Scaling

definition *scale-spmf* $:: real \Rightarrow 'a\ spmf \Rightarrow 'a\ spmf$

where

$scale\text{-}spmf\ r\ p = embed\text{-}spmf\ (\lambda x. min\ (inverse\ (weight\text{-}spmf\ p))\ (max\ 0\ r)) * spmf\ p\ x$

lemma *scale-spmf-le-1*:

$(\int^+ x. min\ (inverse\ (weight\text{-}spmf\ p))\ (max\ 0\ r)) * spmf\ p\ x \partial count\text{-}space\ UNIV \leq 1$ (**is** *?lhs* \leq -)

proof –

have *?lhs* $= min\ (inverse\ (weight\text{-}spmf\ p))\ (max\ 0\ r) * \int^+ x. spmf\ p\ x \partial count\text{-}space\ UNIV$

by(*subst nn-integral-cmult*[*symmetric*])(*simp-all add: weight-spmf-nonneg max-def min-def ennreal-mult*)

also have $\dots \leq 1$ **unfolding** *weight-spmf-eq-nn-integral-spmf*[*symmetric*]

by(*simp add: min-def max-def weight-spmf-nonneg order.strict-iff-order field-simps ennreal-mult[symmetric]*)
finally show *?thesis* .
qed

lemma *spmf-scale-spmf*: *spmf (scale-spmf r p) x = max 0 (min (inverse (weight-spmf p)) r) * spmf p x (is ?lhs = ?rhs)*
unfolding *scale-spmf-def*
apply(*subst spmf-embed-spmf[OF scale-spmf-le-1]*)
apply(*simp add: max-def min-def weight-spmf-le-0 field-simps weight-spmf-nonneg not-le order.strict-iff-order*)
apply(*metis antisym-conv order-trans weight-spmf-nonneg zero-le-mult-iff zero-le-one*)
done

lemma *real-inverse-le-1-iff*: **fixes** *x :: real*
shows $\llbracket 0 \leq x; x \leq 1 \rrbracket \implies 1 / x \leq 1 \longleftrightarrow x = 1 \vee x = 0$
by *auto*

lemma *spmf-scale-spmf'*: $r \leq 1 \implies \text{spmf (scale-spmf r p) x} = \text{max 0 r} * \text{spmf p x}$
using *real-inverse-le-1-iff[OF weight-spmf-nonneg weight-spmf-le-1, of p]*
by(*auto simp add: spmf-scale-spmf max-def min-def field-simps*)(*metis pmf-le-0-iff spmf-le-weight*)

lemma *scale-spmf-neg*: $r \leq 0 \implies \text{scale-spmf r p} = \text{return-pmf None}$
by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf' max-def*)

lemma *scale-spmf-return-None* [*simp*]: $\text{scale-spmf r (return-pmf None)} = \text{return-pmf None}$
by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf*)

lemma *scale-spmf-conv-bind-bernoulli*:
assumes $r \leq 1$
shows $\text{scale-spmf r p} = \text{bind-pmf (bernoulli-pmf r) } (\lambda b. \text{if } b \text{ then } p \text{ else } \text{return-pmf None})$ (**is** *?lhs = ?rhs*)
proof(*rule spmf-eqI*)
fix *x*
have $\text{ennreal (spmf ?lhs x)} = \text{ennreal (spmf ?rhs x)}$ **using** *assms*
unfolding *spmf-scale-spmf ennreal-pmf-bind nn-integral-measure-pmf UNIV-bool bernoulli-pmf.rep-eq*
apply(*auto simp add: nn-integral-count-space-finite max-def min-def field-simps real-inverse-le-1-iff[OF weight-spmf-nonneg weight-spmf-le-1] weight-spmf-lt-0 not-le ennreal-mult[symmetric]*)
apply (*metis pmf-le-0-iff spmf-le-weight*)
apply (*metis pmf-le-0-iff spmf-le-weight*)
apply (*meson le-divide-eq-1-pos measure-spmf.subprob-measure-le-1 not-less order-trans weight-spmf-le-0*)
by (*meson divide-le-0-1-iff less-imp-le order-trans weight-spmf-le-0*)
thus $\text{spmf ?lhs x} = \text{spmf ?rhs x}$ **by** *simp*

qed

lemma *nn-integral-spmf*: $(\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } A) = \text{emeasure } (\text{measure-spmf } p) \ A$
apply(*simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space nn-integral-pmf[symmetric]*)
apply(*rule nn-integral-bij-count-space[where g=Some]*)
apply(*auto simp add: bij-betw-def*)
done

lemma *measure-spmf-scale-spmf*: $\text{measure-spmf } (\text{scale-spmf } r \ p) = \text{scale-measure } (\text{min } (\text{inverse } (\text{weight-spmf } p)) \ r) \ (\text{measure-spmf } p)$
apply(*rule measure-eqI*)
apply *simp*
apply(*simp add: nn-integral-spmf[symmetric] spmf-scale-spmf*)
apply(*subst nn-integral-cmult[symmetric]*)
apply(*auto simp add: max-def min-def ennreal-mult[symmetric] not-le ennreal-lt-0*)
done

lemma *measure-spmf-scale-spmf'*:
 $r \leq 1 \implies \text{measure-spmf } (\text{scale-spmf } r \ p) = \text{scale-measure } r \ (\text{measure-spmf } p)$
unfolding *measure-spmf-scale-spmf*
apply(*cases weight-spmf p > 0*)
apply(*simp add: min.absorb2 field-simps weight-spmf-le-1 mult-le-one*)
apply(*clarsimp simp add: weight-spmf-le-0 min-def scale-spmf-neg weight-spmf-eq-0 not-less*)
done

lemma *scale-spmf-1 [simp]*: $\text{scale-spmf } 1 \ p = p$
apply(*rule spmf-eqI*)
apply(*simp add: spmf-scale-spmf max-def min-def order.strict-iff-order field-simps weight-spmf-nonneg*)
apply(*metis antisym-conv divide-le-eq-1 less-imp-le pmf-nonneg spmf-le-weight weight-spmf-nonneg weight-spmf-le-1*)
done

lemma *scale-spmf-0 [simp]*: $\text{scale-spmf } 0 \ p = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf min-def max-def weight-spmf-le-0*)

lemma *bind-scale-spmf*:
assumes $r: r \leq 1$
shows $\text{bind-spmf } (\text{scale-spmf } r \ p) \ f = \text{bind-spmf } p \ (\lambda x. \text{scale-spmf } r \ (f \ x))$
(is ?lhs = ?rhs)
proof(*rule spmf-eqI*)
fix x
have $\text{ennreal } (\text{spmf } ?lhs \ x) = \text{ennreal } (\text{spmf } ?rhs \ x)$ **using** r
by(*simp add: ennreal-spmf-bind measure-spmf-scale-spmf' nn-integral-scale-measure spmf-scale-spmf'*)
(simp add: ennreal-mult ennreal-lt-0 nn-integral-cmult max-def min-def)

thus $\text{spmf } ?lhs\ x = \text{spmf } ?rhs\ x$ **by** *simp*
qed

lemma *scale-bind-spmf*:

assumes $r \leq 1$

shows $\text{scale-spmf } r\ (\text{bind-spmf } p\ f) = \text{bind-spmf } p\ (\lambda x. \text{scale-spmf } r\ (f\ x))$

(is $?lhs = ?rhs$)

proof(*rule* *spmf-eqI*)

fix x

have $\text{ennreal } (\text{spmf } ?lhs\ x) = \text{ennreal } (\text{spmf } ?rhs\ x)$ **using** *assms*

unfolding *spmf-scale-spmf'*[*OF* *assms*]

by(*simp* *add*: *ennreal-mult* *ennreal-spmf-bind* *spmf-scale-spmf'* *nn-integral-cmult* *max-def* *min-def*)

thus $\text{spmf } ?lhs\ x = \text{spmf } ?rhs\ x$ **by** *simp*

qed

lemma *bind-spmf-const*: $\text{bind-spmf } p\ (\lambda x. q) = \text{scale-spmf } (\text{weight-spmf } p)\ q$ **(is** $?lhs = ?rhs$)

proof(*rule* *spmf-eqI*)

fix x

have $\text{ennreal } (\text{spmf } ?lhs\ x) = \text{ennreal } (\text{spmf } ?rhs\ x)$

using *measure-spmf.subprob-measure-le-1*[*of* p *space* (*measure-spmf* p)]

by(*subst* *ennreal-spmf-bind*)(*simp* *add*: *spmf-scale-spmf'* *weight-spmf-le-1* *ennreal-mult* *mult commute* *max-def* *min-def* *measure-spmf.emmeasure-eq-measure*)

thus $\text{spmf } ?lhs\ x = \text{spmf } ?rhs\ x$ **by** *simp*

qed

lemma *map-scale-spmf*: $\text{map-spmf } f\ (\text{scale-spmf } r\ p) = \text{scale-spmf } r\ (\text{map-spmf } f\ p)$ **(is** $?lhs = ?rhs$)

proof(*rule* *spmf-eqI*)

fix i

show $\text{spmf } ?lhs\ i = \text{spmf } ?rhs\ i$ **unfolding** *spmf-scale-spmf*

by(*subst* ($1\ 2$) *spmf-map*)(*auto* *simp* *add*: *measure-spmf-scale-spmf* *max-def* *min-def* *ennreal-lt-0*)

qed

lemma *set-scale-spmf*: $\text{set-spmf } (\text{scale-spmf } r\ p) = (\text{if } r > 0 \text{ then } \text{set-spmf } p \text{ else } \{\})$

apply(*auto* *simp* *add*: *in-set-spmf-iff-spmf* *spmf-scale-spmf*)

apply(*simp* *add*: *max-def* *min-def* *not-le* *weight-spmf-lt-0* *weight-spmf-eq-0* *split*: *if-split-asm*)

done

lemma *set-scale-spmf'* [*simp*]: $0 < r \implies \text{set-spmf } (\text{scale-spmf } r\ p) = \text{set-spmf } p$
by(*simp* *add*: *set-scale-spmf*)

lemma *rel-spmf-scaleI*:

assumes $r > 0 \implies \text{rel-spmf } A\ p\ q$

shows $\text{rel-spmf } A\ (\text{scale-spmf } r\ p)\ (\text{scale-spmf } r\ q)$

proof(cases $r > 0$)
 case *True*
 from *assms*[*OF this*] **show** *?thesis*
 by(*rule rel-spmfE*)(*auto simp add: map-scale-spmf[symmetric] spmf-rel-map*
True intro: rel-spmf-reflI)
qed(*simp add: not-less scale-spmf-neg*)

lemma *weight-scale-spmf*: $\text{weight-spmf } (\text{scale-spmf } r \ p) = \min \ 1 \ (\max \ 0 \ r \ * \ \text{weight-spmf } p)$

proof –
 have $\text{ennreal } (\text{weight-spmf } (\text{scale-spmf } r \ p)) = \min \ 1 \ (\max \ 0 \ r \ * \ \text{ennreal } (\text{weight-spmf } p))$
 unfolding *weight-spmf-eq-nn-integral-spmf*
 apply(*simp add: spmf-scale-spmf ennreal-mult zero-ereal-def[symmetric] nn-integral-cmult*)
 apply(*auto simp add: weight-spmf-eq-nn-integral-spmf[symmetric] field-simps*
min-def max-def not-le weight-spmf-lt-0 ennreal-mult[symmetric])
 subgoal by(*subst (asm) ennreal-mult[symmetric], meson divide-less-0-1-iff*
le-less-trans not-le weight-spmf-lt-0, simp+, meson not-le pos-divide-le-eq weight-spmf-le-0)
 subgoal by(*cases r ≥ 0*)(*simp-all add: ennreal-mult[symmetric] weight-spmf-nonneg*
ennreal-lt-0, meson le-less-trans not-le pos-divide-le-eq zero-less-divide-1-iff)
 done
 thus *?thesis* by(*auto simp add: min-def max-def ennreal-mult[symmetric] split:*
if-split-asm)
qed

lemma *weight-scale-spmf'* [*simp*]:

$\llbracket 0 \leq r; r \leq 1 \rrbracket \implies \text{weight-spmf } (\text{scale-spmf } r \ p) = r \ * \ \text{weight-spmf } p$
 by(*simp add: weight-scale-spmf max-def min-def*)(*metis antisym-conv mult-left-le*
order-trans weight-spmf-le-1)

lemma *pmf-scale-spmf-None*:

$\text{pmf } (\text{scale-spmf } k \ p) \ \text{None} = 1 - \min \ 1 \ (\max \ 0 \ k \ * \ (1 - \text{pmf } p \ \text{None}))$
 unfolding *pmf-None-eq-weight-spmf* by(*simp add: weight-scale-spmf*)

lemma *scale-scale-spmf*:

$\text{scale-spmf } r \ (\text{scale-spmf } r' \ p) = \text{scale-spmf } (r \ * \ \max \ 0 \ (\min \ (\text{inverse } (\text{weight-spmf } p)) \ r')) \ p$

(*is ?lhs = ?rhs*)

proof(*rule spmf-eqI*)

fix *i*

have $\max \ 0 \ (\min \ (1 \ / \ \text{weight-spmf } p) \ r') \ * \ \max \ 0 \ (\min \ (1 \ / \ \min \ 1 \ (\text{weight-spmf } p \ * \ \max \ 0 \ r')) \ r) = \max \ 0 \ (\min \ (1 \ / \ \text{weight-spmf } p) \ (r \ * \ \max \ 0 \ (\min \ (1 \ / \ \text{weight-spmf } p) \ r')))$

proof(cases *weight-spmf p > 0*)

case *False*

thus *?thesis* by(*simp add: not-less weight-spmf-le-0*)

next

case *True*

thus *?thesis* by(*simp add: field-simps max-def min.absorb-iff2[symmetric]*)(*auto*)

simp add: min-def field-simps zero-le-mult-iff)

qed

then show *spmf ?lhs i = spmf ?rhs i*

by(*simp add: spmf-scale-spmf field-simps weight-scale-spmf*)

qed

lemma *scale-scale-spmf' [simp]*:

$\llbracket 0 \leq r; r \leq 1; 0 \leq r'; r' \leq 1 \rrbracket$

$\implies \text{scale-spmf } r (\text{scale-spmf } r' p) = \text{scale-spmf } (r * r') p$

apply(*cases weight-spmf p > 0*)

apply(*auto simp add: scale-scale-spmf min-def max-def field-simps not-le weight-spmf-lt-0 weight-spmf-eq-0 not-less weight-spmf-le-0*)

apply(*subgoal-tac 1 = r'*)

apply (*metis (no-types) div-by-1 eq-iff measure-spmf.subprob-measure-le-1 mult.commute mult-cancel-right1*)

apply(*meson eq-iff le-divide-eq-1-pos measure-spmf.subprob-measure-le-1 mult-imp-div-pos-le order.trans*)

done

lemma *scale-spmf-eq-same: scale-spmf r p = p \longleftrightarrow weight-spmf p = 0 \vee r = 1*

$\vee r \geq 1 \wedge \text{weight-spmf } p = 1$

(*is ?lhs \longleftrightarrow ?rhs*)

proof

assume *?lhs*

hence *weight-spmf (scale-spmf r p) = weight-spmf p* **by** *simp*

hence ***: $\min 1 (\max 0 r * \text{weight-spmf } p) = \text{weight-spmf } p$ **by**(*simp add: weight-scale-spmf*)

hence ****: $\text{weight-spmf } p = 0 \vee r \geq 1$ **by**(*auto simp add: min-def max-def split: if-split-asm*)

show *?rhs*

proof(*cases weight-spmf p = 0*)

case *False*

with **** **have** $r \geq 1$ **by** *simp*

with *** **False** **have** $r = 1 \vee \text{weight-spmf } p = 1$ **by**(*simp add: max-def min-def not-le split: if-split-asm*)

with ($r \geq 1$) **show** *?thesis* **by** *simp*

qed *simp*

qed(*auto intro!: spmf-eqI simp add: spmf-scale-spmf, metis pmf-le-0-iff spmf-le-weight*)

lemma *map-const-spmf-of-set*:

$\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{map-spmf } (\lambda-. c) (\text{spmof-of-set } A) = \text{return-spmf } c$

by(*simp add: map-spmf-conv-bind-spmf bind-spmf-const*)

27.16 Conditional spmfs

lemma *set-pmf-Int-Some: set-pmf p \cap Some 'A = {} \longleftrightarrow set-spmf p \cap A = {}*

by(*auto simp add: in-set-spmf*)

lemma *measure-spmf-zero-iff: measure (measure-spmf p) A = 0 \longleftrightarrow set-spmf p*

$\cap A = \{\}$

unfolding *measure-measure-spmf-conv-measure-pmf* **by** (*simp add: measure-pmf-zero-iff set-pmf-Int-Some*)

definition *cond-spmf* :: 'a spmf \Rightarrow 'a set \Rightarrow 'a spmf

where *cond-spmf* p A = (if *set-spmf* p \cap A = $\{\}$ then *return-pmf* None else *cond-pmf* p (Some ' A))

lemma *set-cond-spmf* [*simp*]: *set-spmf* (*cond-spmf* p A) = *set-spmf* p \cap A

by (*auto* $\&$ *simp add: cond-spmf-def in-set-spmf iff: set-cond-pmf [THEN set-eq-iff [THEN iffD1], THEN spec, rotated]*)

lemma *cond-map-spmf* [*simp*]: *cond-spmf* (*map-spmf* f p) A = *map-spmf* f (*cond-spmf* p (f -' A))

proof -

have *map-option* f -' Some ' A = Some ' f -' A **by** *auto*

moreover have *set-pmf* p \cap *map-option* f -' Some ' A \neq $\{\}$ **if** Some x \in *set-pmf* p f x \in A **for** x

using *that* **by** *auto*

ultimately show *?thesis* **by** (*auto simp add: cond-spmf-def in-set-spmf cond-map-pmf*)

qed

lemma *spmf-cond-spmf* [*simp*]:

spmf (*cond-spmf* p A) x = (if x \in A then *spmf* p x / *measure* (*measure-spmf* p) A else 0)

by (*auto simp add: cond-spmf-def pmf-cond set-pmf-Int-Some [symmetric] measure-measure-spmf-conv-measure-measure-pmf-zero-iff*)

lemma *bind-eq-return-pmf-None*:

bind-spmf p f = *return-pmf* None \iff ($\forall x \in \text{set-spmf } p. f x = \text{return-pmf } \text{None}$)

by (*auto simp add: bind-spmf-def bind-eq-return-pmf in-set-spmf split: option.splits*)

lemma *return-pmf-None-eq-bind*:

return-pmf None = *bind-spmf* p f \iff ($\forall x \in \text{set-spmf } p. f x = \text{return-pmf } \text{None}$)

using *bind-eq-return-pmf-None* [of p f] **by** *auto*

27.17 Product spmf

definition *pair-spmf* :: 'a spmf \Rightarrow 'b spmf \Rightarrow ('a \times 'b) spmf

where *pair-spmf* p q = *bind-pmf* (*pair-pmf* p q) ($\lambda xy. \text{case } xy \text{ of } (\text{Some } x, \text{Some } y) \Rightarrow \text{return-spmf } (x, y) \mid - \Rightarrow \text{return-pmf } \text{None}$)

lemma *map-fst-pair-spmf* [*simp*]: *map-spmf* fst (*pair-spmf* p q) = *scale-spmf* (*weight-spmf* q) p

unfolding *bind-spmf-const* [*symmetric*]

apply (*simp add: pair-spmf-def map-bind-pmf pair-pmf-def bind-assoc-pmf option.case-distrib*)

apply (*subst bind-commute-pmf*)

apply (*auto intro!: bind-pmf-cong [OF refl] simp add: bind-return-pmf bind-spmf-def bind-return-pmf' case-option-collapse option.case-distrib [where h = map-spmf -] op-*

tion.case-distrib[symmetric] case-option-id split: option.split cong del: option.case-cong-weak)
done

lemma *map-snd-pair-spmf [simp]: map-spmf snd (pair-spmf p q) = scale-spmf (weight-spmf p) q*

unfolding *bind-spmf-const[symmetric]*

apply(*simp add: pair-spmf-def map-bind-pmf pair-pmf-def bind-assoc-pmf option.case-distrib*

cong del: option.case-cong-weak)

apply(*auto intro!: bind-pmf-cong[OF refl] simp add: bind-return-pmf bind-spmf-def bind-return-pmf' case-option-collapse option.case-distrib[where h=map-spmf -] option.case-distrib[symmetric] case-option-id split: option.split cong del: option.case-cong-weak*)

done

lemma *set-pair-spmf [simp]: set-spmf (pair-spmf p q) = set-spmf p × set-spmf q*

by(*auto 4 3 simp add: pair-spmf-def set-spmf-bind-pmf bind-UNION in-set-spmf intro: rev-bezI split: option.splits*)

lemma *spmf-pair [simp]: spmf (pair-spmf p q) (x, y) = spmf p x * spmf q y (is ?lhs = ?rhs)*

proof –

have *ennreal ?lhs = ∫⁺ a. ∫⁺ b. indicator {(x, y)} (a, b) ∂measure-spmf q*
∂measure-spmf p

unfolding *measure-spmf-def pair-spmf-def ennreal-pmf-bind nn-integral-pair-pmf'*

by(*auto simp add: zero-ereal-def[symmetric] nn-integral-distr nn-integral-restrict-space nn-integral-multc[symmetric] intro!: nn-integral-cong split: option.split split-indicator*)

also have *... = ∫⁺ a. (∫⁺ b. indicator {y} b ∂measure-spmf q) * indicator {x} a ∂measure-spmf p*

by(*subst nn-integral-multc[symmetric](auto intro!: nn-integral-cong split: split-indicator)*)

also have *... = ennreal ?rhs* **by**(*simp add: emeasure-spmf-single max-def ennreal-mult mult commute*)

finally show *?thesis* **by** *simp*

qed

lemma *pair-map-spmf2: pair-spmf p (map-spmf f q) = map-spmf (apsnd f) (pair-spmf p q)*

by(*auto simp add: pair-spmf-def pair-map-pmf2 bind-map-pmf map-bind-pmf intro: bind-pmf-cong split: option.split*)

lemma *pair-map-spmf1: pair-spmf (map-spmf f p) q = map-spmf (apfst f) (pair-spmf p q)*

by(*auto simp add: pair-spmf-def pair-map-pmf1 bind-map-pmf map-bind-pmf intro: bind-pmf-cong split: option.split*)

lemma *pair-map-spmf: pair-spmf (map-spmf f p) (map-spmf g q) = map-spmf (map-prod f g) (pair-spmf p q)*

unfolding *pair-map-spmf2 pair-map-spmf1 spmf.map-comp* **by**(*simp add: apfst-def apsnd-def o-def prod.map-comp*)

lemma *pair-spmf-alt-def*: $\text{pair-spmf } p \ q = \text{bind-spmf } p \ (\lambda x. \text{bind-spmf } q \ (\lambda y. \text{return-spmf } (x, y)))$
by(*auto simp add: pair-spmf-def pair-pmf-def bind-spmf-def bind-assoc-pmf bind-return-pmf split: option.split intro: bind-pmf-cong*)

lemma *weight-pair-spmf* [*simp*]: $\text{weight-spmf } (\text{pair-spmf } p \ q) = \text{weight-spmf } p \ * \ \text{weight-spmf } q$
unfolding *pair-spmf-alt-def* **by**(*simp add: weight-bind-spmf o-def*)

lemma *pair-scale-spmf1*:
 $r \leq 1 \implies \text{pair-spmf } (\text{scale-spmf } r \ p) \ q = \text{scale-spmf } r \ (\text{pair-spmf } p \ q)$
by(*simp add: pair-spmf-alt-def scale-bind-spmf bind-scale-spmf*)

lemma *pair-scale-spmf2*:
 $r \leq 1 \implies \text{pair-spmf } p \ (\text{scale-spmf } r \ q) = \text{scale-spmf } r \ (\text{pair-spmf } p \ q)$
by(*simp add: pair-spmf-alt-def scale-bind-spmf bind-scale-spmf*)

lemma *pair-spmf-return-None1* [*simp*]: $\text{pair-spmf } (\text{return-pmf } \text{None}) \ p = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*clarsimp*)

lemma *pair-spmf-return-None2* [*simp*]: $\text{pair-spmf } p \ (\text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*clarsimp*)

lemma *pair-spmf-return-spmf1*: $\text{pair-spmf } (\text{return-spmf } x) \ q = \text{map-spmf } (\text{Pair } x) \ q$
by(*rule spmf-eqI*)(*auto split: split-indicator simp add: spmf-map-inj' inj-on-def intro: spmf-map-outside*)

lemma *pair-spmf-return-spmf2*: $\text{pair-spmf } p \ (\text{return-spmf } y) = \text{map-spmf } (\lambda x. (x, y)) \ p$
by(*rule spmf-eqI*)(*auto split: split-indicator simp add: inj-on-def intro!: spmf-map-outside spmf-map-inj'[symmetric]*)

lemma *pair-spmf-return-spmf* [*simp*]: $\text{pair-spmf } (\text{return-spmf } x) \ (\text{return-spmf } y) = \text{return-spmf } (x, y)$
by(*simp add: pair-spmf-return-spmf1*)

lemma *rel-pair-spmf-prod*:
 $\text{rel-spmf } (\text{rel-prod } A \ B) \ (\text{pair-spmf } p \ q) \ (\text{pair-spmf } p' \ q') \longleftrightarrow$
 $\text{rel-spmf } A \ (\text{scale-spmf } (\text{weight-spmf } q) \ p) \ (\text{scale-spmf } (\text{weight-spmf } q') \ p') \wedge$
 $\text{rel-spmf } B \ (\text{scale-spmf } (\text{weight-spmf } p) \ q) \ (\text{scale-spmf } (\text{weight-spmf } p') \ q')$
(*is ?lhs \longleftrightarrow ?rhs is - \longleftrightarrow ?A \wedge ?B is - \longleftrightarrow rel-spmf - ?p ?p' \wedge rel-spmf - ?q ?q'*)
proof(*intro iffI conjI*)
assume *?rhs*
then obtain *pq pq'* **where** *p*: $\text{map-spmf } \text{fst } pq = ?p$ **and** *p'*: $\text{map-spmf } \text{snd } pq = ?p'$

```

and  $q$ :  $\text{map-spmf fst } pq' = ?q$  and  $q'$ :  $\text{map-spmf snd } pq' = ?q'$ 
and  $*$ :  $\bigwedge x x'. (x, x') \in \text{set-spmf } pq \implies A x x'$ 
and  $**$ :  $\bigwedge y y'. (y, y') \in \text{set-spmf } pq' \implies B y y'$  by( $\text{auto elim!}:$   $\text{rel-spmfE}$ )
let  $?f = \lambda((x, x'), (y, y')). ((x, y), (x', y'))$ 
let  $?r = 1 / (\text{weight-spmf } p * \text{weight-spmf } q)$ 
let  $?pq = \text{scale-spmf } ?r (\text{map-spmf } ?f (\text{pair-spmf } pq \text{ } pq'))$ 

{ fix  $p :: 'x \text{ spmf}$  and  $q :: 'y \text{ spmf}$ 
  assume  $\text{weight-spmf } q \neq 0$ 
  and  $\text{weight-spmf } p \neq 0$ 
  and  $1 / (\text{weight-spmf } p * \text{weight-spmf } q) \leq \text{weight-spmf } p * \text{weight-spmf } q$ 
  hence  $1 \leq (\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q)$ 
  by( $\text{simp add: pos-divide-le-eq order.strict-iff-order weight-spmf-nonneg}$ )
  moreover have  $(\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q) \leq (1 * 1) * (1 * 1)$ 
  by( $\text{intro mult-mono}$ )( $\text{simp-all add: weight-spmf-nonneg weight-spmf-le-1}$ )
  ultimately have  $(\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q) = 1$  by  $\text{simp}$ 
  hence  $*$ :  $\text{weight-spmf } p * \text{weight-spmf } q = 1$ 
  by( $\text{metis antisym-conv less-le mult-less-cancel-left1 weight-pair-spmf weight-spmf-le-1 weight-spmf-nonneg}$ )
  hence  $**$ :  $\text{weight-spmf } p = 1$  by( $\text{metis antisym-conv mult-left-le weight-spmf-le-1 weight-spmf-nonneg}$ )
  moreover from  $**$  have  $\text{weight-spmf } q = 1$  by  $\text{simp}$ 
  moreover note  $\text{calculation}$  }
note  $\text{full} = \text{this}$ 

show  $?lhs$ 
proof
  have  $[\text{simp}]$ :  $\text{fst} \circ ?f = \text{map-prod fst fst}$  by( $\text{simp add: fun-eq-iff}$ )
  have  $\text{map-spmf fst } ?pq = \text{scale-spmf } ?r (\text{pair-spmf } ?p \text{ } ?q)$ 
  by( $\text{simp add: pair-map-spmf[symmetric]} p q \text{ map-scale-spmf spmf.map-comp}$ )
  also have  $\dots = \text{pair-spmf } p \text{ } q$  using  $\text{full[of } p \text{ } q]$ 
  by( $\text{simp add: pair-scale-spmf1 pair-scale-spmf2 weight-spmf-le-1 weight-spmf-nonneg}$ )
  ( $\text{auto simp add: scale-scale-spmf max-def min-def field-simps weight-spmf-nonneg weight-spmf-eq-0}$ )
  finally show  $\text{map-spmf fst } ?pq = \dots$ 

  have  $[\text{simp}]$ :  $\text{snd} \circ ?f = \text{map-prod snd snd}$  by( $\text{simp add: fun-eq-iff}$ )
  from  $\langle ?rhs \rangle$  have  $\text{eq: weight-spmf } p * \text{weight-spmf } q = \text{weight-spmf } p' * \text{weight-spmf } q'$ 
  by( $\text{auto dest!}:$   $\text{rel-spmf-weightD simp add: weight-spmf-le-1 weight-spmf-nonneg}$ )

  have  $\text{map-spmf snd } ?pq = \text{scale-spmf } ?r (\text{pair-spmf } ?p' \text{ } ?q')$ 
  by( $\text{simp add: pair-map-spmf[symmetric]} p' q' \text{ map-scale-spmf spmf.map-comp}$ )
  also have  $\dots = \text{pair-spmf } p' \text{ } q'$  using  $\text{full[of } p' \text{ } q'] \text{ eq}$ 
  by( $\text{simp add: pair-scale-spmf1 pair-scale-spmf2 weight-spmf-le-1 weight-spmf-nonneg}$ )
  ( $\text{auto simp add: scale-scale-spmf max-def min-def field-simps weight-spmf-nonneg weight-spmf-eq-0}$ )

```

```

finally show map-spmf snd ?pq = ... .
qed(auto simp add: set-scale-spmf split: if-split-asm dest: * ** )
next
assume ?lhs
then obtain pq where pq: map-spmf fst pq = pair-spmf p q
  and pq': map-spmf snd pq = pair-spmf p' q'
  and *:  $\bigwedge x y x' y'. ((x, y), (x', y')) \in \text{set-spmf } pq \implies A x x' \wedge B y y'$ 
  by(auto elim: rel-spmfE)

show ?A
proof
  let ?f =  $(\lambda((x, y), (x', y')). (x, x'))$ 
  let ?pq = map-spmf ?f pq
  have [simp]:  $\text{fst} \circ ?f = \text{fst} \circ \text{fst}$  by(simp add: split-def o-def)
  show map-spmf fst ?pq = scale-spmf (weight-spmf q) p using pq
    by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

  have [simp]:  $\text{snd} \circ ?f = \text{fst} \circ \text{snd}$  by(simp add: split-def o-def)
  show map-spmf snd ?pq = scale-spmf (weight-spmf q') p' using pq'
    by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])
qed(auto dest: * )

show ?B
proof
  let ?f =  $(\lambda((x, y), (x', y')). (y, y'))$ 
  let ?pq = map-spmf ?f pq
  have [simp]:  $\text{fst} \circ ?f = \text{snd} \circ \text{fst}$  by(simp add: split-def o-def)
  show map-spmf fst ?pq = scale-spmf (weight-spmf p) q using pq
    by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

  have [simp]:  $\text{snd} \circ ?f = \text{snd} \circ \text{snd}$  by(simp add: split-def o-def)
  show map-spmf snd ?pq = scale-spmf (weight-spmf p') q' using pq'
    by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])
qed(auto dest: * )
qed

```

lemma pair-pair-spmf:

```

pair-spmf (pair-spmf p q) r = map-spmf ( $\lambda(x, (y, z)). ((x, y), z)$ ) (pair-spmf p
(pair-spmf q r))
by(simp add: pair-spmf-alt-def map-spmf-conv-bind-spmf)

```

lemma pair-commute-spmf:

```

pair-spmf p q = map-spmf ( $\lambda(y, x). (x, y)$ ) (pair-spmf q p)
unfolding pair-spmf-alt-def by(subst bind-commute-spmf)(simp add: map-spmf-conv-bind-spmf)

```

27.18 Assertions

definition assert-spmf :: $\text{bool} \Rightarrow \text{unit spmf}$

where assert-spmf b = (if b then return-spmf () else return-pmf None)

lemma *assert-spmf-simps* [*simp*]:
assert-spmf True = return-spmf ()
assert-spmf False = return-pmf None
by(*simp-all add: assert-spmf-def*)

lemma *in-set-assert-spmf* [*simp*]: $x \in \text{set-spmf } (\text{assert-spmf } p) \longleftrightarrow p$
by(*cases p simp-all*)

lemma *set-spmf-assert-spmf-eq-empty* [*simp*]: $\text{set-spmf } (\text{assert-spmf } b) = \{\} \longleftrightarrow \neg b$
by(*cases b simp-all*)

lemma *lossless-assert-spmf* [*iff*]: $\text{lossless-spmf } (\text{assert-spmf } b) \longleftrightarrow b$
by(*cases b simp-all*)

27.19 Try

definition *try-spmf* :: $'a \text{ spmf} \Rightarrow 'a \text{ spmf} \Rightarrow 'a \text{ spmf}$ (*TRY - ELSE - [0,60] 59*)
where *try-spmf* $p \ q = \text{bind-pmf } p \ (\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow q \mid \text{Some } y \Rightarrow \text{return-spmf } y)$

lemma *try-spmf-lossless* [*simp*]:
assumes *lossless-spmf p*
shows *TRY p ELSE q = p*
proof –
have *TRY p ELSE q = bind-pmf p return-pmf* **unfolding** *try-spmf-def* **using** *assms*
by(*auto simp add: lossless-iff-set-pmf-None split: option.split intro: bind-pmf-cong*)
thesis *?thesis* **by**(*simp add: bind-return-pmf'*)
qed

lemma *try-spmf-return-spmf1*: *TRY return-spmf x ELSE q = return-spmf x*
by(*simp add: try-spmf-def bind-return-pmf*)

lemma *try-spmf-return-None* [*simp*]: *TRY return-pmf None ELSE q = q*
by(*simp add: try-spmf-def bind-return-pmf*)

lemma *try-spmf-return-pmf-None2* [*simp*]: *TRY p ELSE return-pmf None = p*
by(*simp add: try-spmf-def option.case-distrib[symmetric] bind-return-pmf' case-option-id*)

lemma *map-try-spmf*: $\text{map-spmf } f \ (\text{try-spmf } p \ q) = \text{try-spmf } (\text{map-spmf } f \ p) \ (\text{map-spmf } f \ q)$
by(*simp add: try-spmf-def map-bind-pmf bind-map-pmf option.case-distrib[where h=map-spmf f] o-def cong del: option.case-cong-weak*)

lemma *try-spmf-bind-pmf*: *TRY (bind-pmf p f) ELSE q = bind-pmf p (\lambda x. TRY (f x) ELSE q)*
by(*simp add: try-spmf-def bind-assoc-pmf*)

lemma *try-spmf-bind-spmf-lossless*:

$lossless\text{-}spmf\ p \implies TRY\ (bind\text{-}spmf\ p\ f)\ ELSE\ q = bind\text{-}spmf\ p\ (\lambda x.\ TRY\ (f\ x)\ ELSE\ q)$

by(*auto simp add: try-spmf-def bind-spmf-def bind-assoc-pmf bind-return-pmf lossless-iff-set-pmf-None intro!: bind-pmf-cong split: option.split*)

lemma *try-spmf-bind-out*:

$lossless\text{-}spmf\ p \implies bind\text{-}spmf\ p\ (\lambda x.\ TRY\ (f\ x)\ ELSE\ q) = TRY\ (bind\text{-}spmf\ p\ f)\ ELSE\ q$

by(*simp add: try-spmf-bind-spmf-lossless*)

lemma *lossless-try-spmf [simp]*:

$lossless\text{-}spmf\ (TRY\ p\ ELSE\ q) \iff lossless\text{-}spmf\ p \vee lossless\text{-}spmf\ q$

by(*auto simp add: try-spmf-def in-set-spmf lossless-iff-set-pmf-None split: option.splits*)

context includes *lifting-syntax*

begin

lemma *try-spmf-parametric [transfer-rule]*:

$(rel\text{-}spmf\ A\ ==>\ rel\text{-}spmf\ A\ ==>\ rel\text{-}spmf\ A)\ try\text{-}spmf\ try\text{-}spmf$

unfolding *try-spmf-def [abs-def]* **by** *transfer-prover*

end

lemma *try-spmf-cong*:

$\llbracket p = p'; \neg lossless\text{-}spmf\ p' \implies q = q' \rrbracket \implies TRY\ p\ ELSE\ q = TRY\ p'\ ELSE\ q'$

unfolding *try-spmf-def*

by(*rule bind-pmf-cong*)(*auto split: option.split simp add: lossless-iff-set-pmf-None*)

lemma *rel-spmf-try-spmf*:

$\llbracket rel\text{-}spmf\ R\ p\ p'; \neg lossless\text{-}spmf\ p' \implies rel\text{-}spmf\ R\ q\ q' \rrbracket$

$\implies rel\text{-}spmf\ R\ (TRY\ p\ ELSE\ q)\ (TRY\ p'\ ELSE\ q')$

unfolding *try-spmf-def*

apply(*rule rel-pmf-bindI [where R= $\lambda x\ y.\ rel\text{-}option\ R\ x\ y \wedge x \in set\text{-}pmf\ p \wedge y \in set\text{-}pmf\ p'$]*)

apply(*erule pmf.rel-mono-strong; simp*)

apply(*auto split: option.split simp add: lossless-iff-set-pmf-None*)

done

lemma *spmf-try-spmf*:

$spmf\ (TRY\ p\ ELSE\ q)\ x = spmf\ p\ x + pmf\ p\ None * spmf\ q\ x$

proof –

have $ennreal\ (spmf\ (TRY\ p\ ELSE\ q)\ x) = \int^+ y.\ ennreal\ (spmf\ q\ x) * indicator\ \{None\}\ y + indicator\ \{Some\ x\}\ y\ \partial measure\text{-}pmf\ p$

unfolding *try-spmf-def ennreal-pmf-bind* **by**(*rule nn-integral-cong*)(*simp split: option.split split-indicator*)

also have $\dots = (\int^+ y.\ ennreal\ (spmf\ q\ x) * indicator\ \{None\}\ y\ \partial measure\text{-}pmf\ p) + \int^+ y.\ indicator\ \{Some\ x\}\ y\ \partial measure\text{-}pmf\ p$

by(*simp add: nn-integral-add*)
also have $\dots = \text{ennreal } (\text{spmf } q \ x) * \text{pmf } p \ \text{None} + \text{spmf } p \ x$ **by**(*simp add: emeasure-pmf-single*)
finally show *?thesis* **by**(*simp add: ennreal-mult[symmetric] ennreal-plus[symmetric] del: ennreal-plus*)
qed

lemma *try-scale-spmf-same* [*simp*]: $\text{lossless-spmf } p \implies \text{TRY } \text{scale-spmf } k \ p \ \text{ELSE } p = p$
by(*rule spmf-eqI*)(*auto simp add: spmf-try-spmf spmf-scale-spmf pmf-scale-spmf-None lossless-iff-pmf-None weight-spmf-conv-pmf-None min-def max-def field-simps*)

lemma *pmf-try-spmf-None* [*simp*]: $\text{pmf } (\text{TRY } p \ \text{ELSE } q) \ \text{None} = \text{pmf } p \ \text{None} * \text{pmf } q \ \text{None}$ (**is** *?lhs = ?rhs*)
proof –
have $?lhs = \int x. \text{pmf } q \ \text{None} * \text{indicator } \{\text{None}\} \ x \ \partial \text{measure-pmf } p$
unfolding *try-spmf-def pmf-bind* **by**(*rule Bochner-Integration.integral-cong*)(*simp-all split: option.split*)
also have $\dots = ?rhs$ **by**(*simp add: measure-pmf-single*)
finally show *?thesis* .
qed

lemma *try-bind-spmf-lossless2*:
 $\text{lossless-spmf } q \implies \text{TRY } (\text{bind-spmf } p \ f) \ \text{ELSE } q = \text{TRY } (p \gg (\lambda x. \text{TRY } (f \ x) \ \text{ELSE } q)) \ \text{ELSE } q$
by(*rule spmf-eqI*)(*simp add: spmf-try-spmf pmf-bind-spmf-None spmf-bind field-simps measure-spmf.integrable-const-bound[where B=1] pmf-le-1 lossless-iff-pmf-None*)

lemma *try-bind-spmf-lossless2'*:
fixes $f :: 'a \Rightarrow 'b$ **spmfs** **shows**
 $\llbracket \text{NO-MATCH } (\lambda x :: 'a. \text{try-spmf } (g \ x :: 'b \ \text{spmfs}) \ (h \ x)) \ f; \text{lossless-spmf } q \rrbracket$
 $\implies \text{TRY } (\text{bind-spmf } p \ f) \ \text{ELSE } q = \text{TRY } (p \gg (\lambda x :: 'a. \text{TRY } (f \ x) \ \text{ELSE } q)) \ \text{ELSE } q$
by(*rule try-bind-spmf-lossless2*)

lemma *try-bind-assert-spmf*:
 $\text{TRY } (\text{assert-spmf } b \gg f) \ \text{ELSE } q = (\text{if } b \ \text{then } \text{TRY } (f \ ()) \ \text{ELSE } q \ \text{else } q)$
by *simp*

27.20 Miscellaneous

lemma **assumes** $\text{rel-spmf } (\lambda x \ y. \text{bad1 } x = \text{bad2 } y \wedge (\neg \text{bad2 } y \longrightarrow A \ x \longleftrightarrow B \ y)) \ p \ q$ (**is** *rel-spmf ?A - -*)
shows *fundamental-lemma-bad*: $\text{measure } (\text{measure-spmf } p) \ \{x. \text{bad1 } x\} = \text{measure } (\text{measure-spmf } q) \ \{y. \text{bad2 } y\}$ (**is** *?bad*)
and *fundamental-lemma*: $|\text{measure } (\text{measure-spmf } p) \ \{x. A \ x\} - \text{measure } (\text{measure-spmf } q) \ \{y. B \ y\}| \leq \text{measure } (\text{measure-spmf } p) \ \{x. \text{bad1 } x\}$ (**is** *?fundamental*)
proof –

have *good*: *rel-fun* $?A$ *op* = $(\lambda x. A\ x \wedge \neg\ bad1\ x)\ (\lambda y. B\ y \wedge \neg\ bad2\ y)$ **by** (*auto simp add: rel-fun-def*)
from *assms* **have** *1*: *measure* (*measure-spmf* *p*) $\{x. A\ x \wedge \neg\ bad1\ x\} = \text{measure}$
(*measure-spmf* *q*) $\{y. B\ y \wedge \neg\ bad2\ y\}$
by (*rule measure-spmf-parametric* [*THEN rel-funD, THEN rel-funD*]) (*rule Collect-parametric* [*THEN rel-funD, OF good*])

have *bad*: *rel-fun* $?A$ *op* = *bad1 bad2* **by** (*simp add: rel-fun-def*)
show *2*: $?bad$ **using** *assms*
by (*rule measure-spmf-parametric* [*THEN rel-funD, THEN rel-funD*]) (*rule Collect-parametric* [*THEN rel-funD, OF bad*])

let $?\mu p = \text{measure}\ (\text{measure-spmf}\ p)$ **and** $?\mu q = \text{measure}\ (\text{measure-spmf}\ q)$
have $\{x. A\ x \wedge\ bad1\ x\} \cup \{x. A\ x \wedge \neg\ bad1\ x\} = \{x. A\ x\}$
and $\{y. B\ y \wedge\ bad2\ y\} \cup \{y. B\ y \wedge \neg\ bad2\ y\} = \{y. B\ y\}$ **by** *auto*
then **have** $|\ ?\mu p\ \{x. A\ x\} - ?\mu q\ \{x. B\ x\} | = |\ ?\mu p\ (\{x. A\ x \wedge\ bad1\ x\} \cup \{x. A\ x \wedge \neg\ bad1\ x\}) - ?\mu q\ (\{y. B\ y \wedge\ bad2\ y\} \cup \{y. B\ y \wedge \neg\ bad2\ y\}) |$
by *simp*
also **have** $\dots = |\ ?\mu p\ \{x. A\ x \wedge\ bad1\ x\} + ?\mu p\ \{x. A\ x \wedge \neg\ bad1\ x\} - ?\mu q\ \{y. B\ y \wedge\ bad2\ y\} - ?\mu q\ \{y. B\ y \wedge \neg\ bad2\ y\} |$
by (*subst* (*1 2*) *measure-Union*) (*auto*)
also **have** $\dots = |\ ?\mu p\ \{x. A\ x \wedge\ bad1\ x\} - ?\mu q\ \{y. B\ y \wedge\ bad2\ y\} |$ **using** *1* **by** *simp*
also **have** $\dots \leq \max\ (\ ?\mu p\ \{x. A\ x \wedge\ bad1\ x\})\ (\ ?\mu q\ \{y. B\ y \wedge\ bad2\ y\})$
by (*rule abs-leI*) (*auto simp add: max-def not-le, simp-all only: add-increasing measure-nonneg mult-2*)
also **have** $\dots \leq \max\ (\ ?\mu p\ \{x. bad1\ x\})\ (\ ?\mu q\ \{y. bad2\ y\})$
by (*rule max.mono; rule measure-spmf.finite-measure-mono; auto*)
also **note** *2* [*symmetric*]
finally **show** $?fundamental$ **by** *simp*
qed

end

theory *Stream-Space*

imports

Infinite-Product-Measure

HOL-Library.Stream

HOL-Library.Linear-Temporal-Logic-on-Streams

begin

lemma *stream-eq-Stream-iff*: $s = x\ \#\#\ t \longleftrightarrow (\text{shd}\ s = x \wedge \text{stl}\ s = t)$

by (*cases s*) *simp*

lemma *Stream-snth*: $(x\ \#\#\ s)\ \#\#\ n = (\text{case}\ n\ \text{of}\ 0 \Rightarrow x \mid \text{Suc}\ n \Rightarrow s\ \#\#\ n)$

by (*cases n*) *simp-all*

definition *to-stream* :: $(\text{nat} \Rightarrow 'a) \Rightarrow 'a\ \text{stream}$ **where**

to-stream $X = \text{smap } X \text{ nats}$

lemma *to-stream-nat-case*: *to-stream* (case-nat x X) = $x \#\#$ *to-stream* X
unfolding *to-stream-def*
by (*subst siterate.ctr*) (*simp add: smap-siterate[symmetric] stream.map-comp comp-def*)

lemma *to-stream-in-streams*: *to-stream* $X \in \text{streams } S \iff (\forall n. X \ n \in S)$
by (*simp add: to-stream-def streams-iff-snth*)

definition *stream-space* :: 'a *measure* \Rightarrow 'a *stream measure* **where**
stream-space $M =$
distr ($\prod_M i \in \text{UNIV}. M$) (*vimage-algebra* (*streams* (*space* M)) *snth* ($\prod_M i \in \text{UNIV}. M$)) *to-stream*

lemma *space-stream-space*: *space* (*stream-space* M) = *streams* (*space* M)
by (*simp add: stream-space-def*)

lemma *streams-stream-space[intro]*: *streams* (*space* M) \in *sets* (*stream-space* M)
using *sets.top[of stream-space M]* **by** (*simp add: space-stream-space*)

lemma *stream-space-Stream*:
 $x \#\# \omega \in \text{space} (\text{stream-space } M) \iff x \in \text{space } M \wedge \omega \in \text{space} (\text{stream-space } M)$
by (*simp add: space-stream-space streams-Stream*)

lemma *stream-space-eq-distr*: *stream-space* $M = \text{distr} (\prod_M i \in \text{UNIV}. M) (\text{stream-space } M) \text{ to-stream}$
unfolding *stream-space-def* **by** (*rule distr-cong*) *auto*

lemma *sets-stream-space-cong[measurable-cong]*:
 $\text{sets } M = \text{sets } N \implies \text{sets} (\text{stream-space } M) = \text{sets} (\text{stream-space } N)$
using *sets-eq-imp-space-eq[of M N]* **by** (*simp add: stream-space-def vimage-algebra-def cong: sets-PiM-cong*)

lemma *measurable-snth-PiM*: $(\lambda \omega. \omega \ !\! n) \in \text{measurable} (\text{stream-space } M) (\prod_M i \in \text{UNIV}. M)$
by (*auto intro!: measurable-vimage-algebra1*
simp: space-PiM streams-iff-sset sset-range image-subset-iff stream-space-def)

lemma *measurable-snth[measurable]*: $(\lambda \omega. \omega \ !\! n) \in \text{measurable} (\text{stream-space } M) M$
using *measurable-snth-PiM measurable-component-singleton* **by** (*rule measurable-compose*)
simp

lemma *measurable-shd[measurable]*: *shd* \in *measurable* (*stream-space* M) M
using *measurable-snth[of 0]* **by** *simp*

lemma *measurable-stream-space2*:

assumes $f\text{-snth}$: $\bigwedge n. (\lambda x. f\ x \ !!\ n) \in \text{measurable } N\ M$
shows $f \in \text{measurable } N\ (\text{stream-space } M)$
unfolding $\text{stream-space-def measurable-distr-eq2}$
proof (rule $\text{measurable-vimage-algebra2}$)
show $f \in \text{space } N \rightarrow \text{streams } (\text{space } M)$
using $f\text{-snth}[\text{THEN measurable-space}]$ **by** (auto simp add: $\text{streams-iff-sset sset-range}$)
show $(\lambda x. \text{op } !!\ (f\ x)) \in \text{measurable } N\ (Pi_M\ UNIV\ (\lambda i. M))$
proof (rule $\text{measurable-PiM-single'}$)
show $(\lambda x. \text{op } !!\ (f\ x)) \in \text{space } N \rightarrow UNIV \rightarrow_E\ \text{space } M$
using $f\text{-snth}[\text{THEN measurable-space}]$ **by** auto
qed (rule $f\text{-snth}$)
qed

lemma $\text{measurable-stream-coinduct}[\text{consumes } 1, \text{case-names shd stl, coinduct set: measurable}]$:
assumes $F\ f$
assumes h : $\bigwedge f. F\ f \implies (\lambda x. \text{shd } (f\ x)) \in \text{measurable } N\ M$
assumes t : $\bigwedge f. F\ f \implies F\ (\lambda x. \text{stl } (f\ x))$
shows $f \in \text{measurable } N\ (\text{stream-space } M)$
proof (rule $\text{measurable-stream-space2}$)
fix n **show** $(\lambda x. f\ x \ !!\ n) \in \text{measurable } N\ M$
using $\langle F\ f \rangle$ **by** (induction n arbitrary: f) (auto intro: $h\ t$)
qed

lemma $\text{measurable-sdrop}[\text{measurable}]$: $\text{sdrop } n \in \text{measurable } (\text{stream-space } M)$
 $(\text{stream-space } M)$
by (rule $\text{measurable-stream-space2}$) (simp add: sdrop-snth)

lemma $\text{measurable-stl}[\text{measurable}]$: $(\lambda \omega. \text{stl } \omega) \in \text{measurable } (\text{stream-space } M)$
 $(\text{stream-space } M)$
by (rule $\text{measurable-stream-space2}$) (simp del: snth.simps add: $\text{snth.simps}[\text{symmetric}]$)

lemma $\text{measurable-to-stream}[\text{measurable}]$: $\text{to-stream} \in \text{measurable } (\prod_M\ i \in UNIV. M)$
 $(\text{stream-space } M)$
by (rule $\text{measurable-stream-space2}$) (simp add: to-stream-def)

lemma $\text{measurable-Stream}[\text{measurable } (\text{raw})]$:
assumes $f[\text{measurable}]$: $f \in \text{measurable } N\ M$
assumes $g[\text{measurable}]$: $g \in \text{measurable } N\ (\text{stream-space } M)$
shows $(\lambda x. f\ x \ ##\ g\ x) \in \text{measurable } N\ (\text{stream-space } M)$
by (rule $\text{measurable-stream-space2}$) (simp add: Stream-snth)

lemma $\text{measurable-smap}[\text{measurable}]$:
assumes $X[\text{measurable}]$: $X \in \text{measurable } N\ M$
shows $\text{smap } X \in \text{measurable } (\text{stream-space } N)$ $(\text{stream-space } M)$
by (rule $\text{measurable-stream-space2}$) simp

lemma $\text{measurable-stake}[\text{measurable}]$:

stake $i \in \text{measurable } (\text{stream-space } (\text{count-space } \text{UNIV})) (\text{count-space } (\text{UNIV} :: 'a::\text{countable list set}))$

by (*induct* i) *auto*

lemma *measurable-shift*[*measurable*]:

assumes $f: f \in \text{measurable } N (\text{stream-space } M)$

assumes [*measurable*]: $g \in \text{measurable } N (\text{stream-space } M)$

shows $(\lambda x. \text{stake } n (f x) @- g x) \in \text{measurable } N (\text{stream-space } M)$

using f **by** (*induction* n *arbitrary*: f) *simp-all*

lemma *measurable-case-stream-replace*[*measurable* (*raw*)]:

$(\lambda x. f x (\text{shd } (g x)) (\text{stl } (g x))) \in \text{measurable } M N \implies (\lambda x. \text{case-stream } (f x) (g x)) \in \text{measurable } M N$

unfolding *stream.case-eq-if* .

lemma *measurable-ev-at*[*measurable*]:

assumes [*measurable*]: $\text{Measurable.pred } (\text{stream-space } M) P$

shows $\text{Measurable.pred } (\text{stream-space } M) (\text{ev-at } P n)$

by (*induction* n) *auto*

lemma *measurable-alw*[*measurable*]:

$\text{Measurable.pred } (\text{stream-space } M) P \implies \text{Measurable.pred } (\text{stream-space } M) (\text{alw } P)$

unfolding *alw-def*

by (*coinduction rule*: *measurable-gfp-coinduct*) (*auto simp*: *inf-continuous-def*)

lemma *measurable-ev*[*measurable*]:

$\text{Measurable.pred } (\text{stream-space } M) P \implies \text{Measurable.pred } (\text{stream-space } M) (\text{ev } P)$

unfolding *ev-def*

by (*coinduction rule*: *measurable-lfp-coinduct*) (*auto simp*: *sup-continuous-def*)

lemma *measurable-until*:

assumes [*measurable*]: $\text{Measurable.pred } (\text{stream-space } M) \varphi \text{ Measurable.pred } (\text{stream-space } M) \psi$

shows $\text{Measurable.pred } (\text{stream-space } M) (\varphi \text{ until } \psi)$

unfolding *UNTIL-def*

by (*coinduction rule*: *measurable-gfp-coinduct*) (*simp-all add*: *inf-continuous-def fun-eq-iff*)

lemma *measurable-holds* [*measurable*]: $\text{Measurable.pred } M P \implies \text{Measurable.pred } (\text{stream-space } M) (\text{holds } P)$

unfolding *holds.simps*[*abs-def*]

by (*rule* *measurable-compose*[*OF measurable-shd*]) *simp*

lemma *measurable-hld*[*measurable*]: **assumes** [*measurable*]: $t \in \text{sets } M$ **shows** $\text{Measurable.pred } (\text{stream-space } M) (\text{HLD } t)$

unfolding *HLD-def* **by** *measurable*

lemma *measurable-nxt*[*measurable (raw)*]:
Measurable.pred (stream-space M) P \implies *Measurable.pred (stream-space M) (nxt P)*
unfolding *nxt.simps*[*abs-def*] **by** *simp*

lemma *measurable-suntil*[*measurable*]:
assumes [*measurable*]: *Measurable.pred (stream-space M) Q Measurable.pred (stream-space M) P*
shows *Measurable.pred (stream-space M) (Q suntil P)*
unfolding *suntil-def* **by** (*coinduction rule: measurable-lfp-coinduct*) (*auto simp: sup-continuous-def*)

lemma *measurable-szip*:
 $(\lambda(\omega 1, \omega 2). \text{ szip } \omega 1 \ \omega 2) \in \text{ measurable } (\text{ stream-space } M \otimes_M \text{ stream-space } N)$
 $(\text{ stream-space } (M \otimes_M N))$
proof (*rule measurable-stream-space2*)
fix *n*
have $(\lambda x. (\text{ case } x \text{ of } (\omega 1, \omega 2) \Rightarrow \text{ szip } \omega 1 \ \omega 2) !! n) = (\lambda(\omega 1, \omega 2). (\omega 1 !! n, \omega 2 !! n))$
by *auto*
also have $\dots \in \text{ measurable } (\text{ stream-space } M \otimes_M \text{ stream-space } N) (M \otimes_M N)$
by *measurable*
finally show $(\lambda x. (\text{ case } x \text{ of } (\omega 1, \omega 2) \Rightarrow \text{ szip } \omega 1 \ \omega 2) !! n) \in \text{ measurable } (\text{ stream-space } M \otimes_M \text{ stream-space } N) (M \otimes_M N)$
qed

lemma (*in prob-space*) *prob-space-stream-space*: *prob-space (stream-space M)*
proof –
interpret *product-prob-space* $\lambda\cdot. M \text{ UNIV } ..$
show *?thesis*
by (*subst stream-space-eq-distr*) (*auto intro!: P.prob-space-distr*)
qed

lemma (*in prob-space*) *nn-integral-stream-space*:
assumes [*measurable*]: $f \in \text{ borel-measurable } (\text{ stream-space } M)$
shows $(\int^+ X. f X \ \partial \text{ stream-space } M) = (\int^+ x. (\int^+ X. f (x \#\# X) \ \partial \text{ stream-space } M) \ \partial M)$
proof –
interpret *S*: *sequence-space M* ..
interpret *P*: *pair-sigma-finite M* $\Pi_M i::\text{ nat} \in \text{ UNIV}. M ..$
have $(\int^+ X. f X \ \partial \text{ stream-space } M) = (\int^+ X. f (\text{ to-stream } X) \ \partial S.S)$
by (*subst stream-space-eq-distr*) (*simp add: nn-integral-distr*)
also have $\dots = (\int^+ X. f (\text{ to-stream } ((\lambda(s, \omega). \text{ case-nat } s \ \omega) X)) \ \partial(M \otimes_M S.S))$
by (*subst S.PiM-iter*[*symmetric*]) (*simp add: nn-integral-distr*)
also have $\dots = (\int^+ x. \int^+ X. f (\text{ to-stream } ((\lambda(s, \omega). \text{ case-nat } s \ \omega) (x, X)))$

$\partial S.S \partial M$)

by (*subst S.nn-integral-fst simp-all*)

also have $\dots = (\int^+ x. \int^+ X. f (x \#\# \text{to-stream } X) \partial S.S \partial M)$

by (*auto intro!: nn-integral-cong simp: to-stream-nat-case*)

also have $\dots = (\int^+ x. \int^+ X. f (x \#\# X) \partial \text{stream-space } M \partial M)$

by (*subst stream-space-eq-distr*)

(*simp add: nn-integral-distr cong: nn-integral-cong*)

finally show ?thesis .

qed

lemma (in *prob-space*) *emeasure-stream-space*:

assumes $X[\text{measurable}]$: $X \in \text{sets } (\text{stream-space } M)$

shows $\text{emeasure } (\text{stream-space } M) X = (\int^+ t. \text{emeasure } (\text{stream-space } M) \{x \in \text{space } (\text{stream-space } M). t \#\# x \in X\} \partial M)$

proof –

have *eq*: $\bigwedge x \text{ xs}. \text{xs} \in \text{space } (\text{stream-space } M) \implies x \in \text{space } M \implies$

$\text{indicator } X (x \#\# \text{xs}) = \text{indicator } \{x \in \text{space } (\text{stream-space } M). x \#\# \text{xs} \in X\} \text{xs}$

by (*auto split: split-indicator*)

show ?thesis

using *nn-integral-stream-space[of indicator X]*

apply (*auto intro!: nn-integral-cong*)

apply (*subst nn-integral-cong*)

apply (*rule eq*)

apply *simp-all*

done

qed

lemma (in *prob-space*) *prob-stream-space*:

assumes $P[\text{measurable}]$: $\{x \in \text{space } (\text{stream-space } M). P x\} \in \text{sets } (\text{stream-space } M)$

shows $\mathcal{P}(x \text{ in } \text{stream-space } M. P x) = (\int^+ t. \mathcal{P}(x \text{ in } \text{stream-space } M. P (t \#\# x)) \partial M)$

proof –

interpret *S*: *prob-space stream-space M*

by (*rule prob-space-stream-space*)

show ?thesis

unfolding *S.emeasure-eq-measure[symmetric]*

by (*subst emeasure-stream-space*) (*auto simp: stream-space-Stream intro!: nn-integral-cong*)

qed

lemma (in *prob-space*) *AE-stream-space*:

assumes $[\text{measurable}]$: $\text{Measurable.pred } (\text{stream-space } M) P$

shows $(\text{AE } X \text{ in } \text{stream-space } M. P X) = (\text{AE } x \text{ in } M. \text{AE } X \text{ in } \text{stream-space } M. P (x \#\# X))$

proof –

interpret *stream*: *prob-space stream-space M*

by (*rule prob-space-stream-space*)

```

have eq:  $\bigwedge x X. \text{indicator } \{x. \neg P x\} (x \#\# X) = \text{indicator } \{X. \neg P (x \#\# X)\} X$ 
  by (auto split: split-indicator)
show ?thesis
  apply (subst AE-iff-nn-integral, simp)
  apply (subst nn-integral-stream-space, simp)
  apply (subst eq)
  apply (subst nn-integral-0-iff-AE, simp)
  apply (simp add: AE-iff-nn-integral[symmetric])
done
qed

```

```

lemma (in prob-space) AE-stream-all:
  assumes [measurable]: Measurable.pred M P and P: AE x in M. P x
  shows AE x in stream-space M. stream-all P x
proof –
  { fix n have AE x in stream-space M. P (x !! n)
    proof (induct n)
      case 0 with P show ?case
        by (subst AE-stream-space) (auto elim!: eventually-mono)
      next
        case (Suc n) then show ?case
          by (subst AE-stream-space) auto
    }
  qed }
then show ?thesis
  unfolding stream-all-def by (simp add: AE-all-countable)
qed

```

```

lemma streams-sets:
  assumes X[measurable]: X ∈ sets M shows streams X ∈ sets (stream-space M)
proof –
  have streams X = {x ∈ space (stream-space M). x ∈ streams X}
  using streams-mono[OF - sets.sets-into-space[OF X]] by (auto simp: space-stream-space)
  also have ... = {x ∈ space (stream-space M). gfp (λp x. shd x ∈ X ∧ p (stl x))
x}
  apply (simp add: set-eq-iff streams-def streamsp-def)
  apply (intro allI conj-cong refl arg-cong2[where f=gfp] ext)
  apply (case-tac xa)
  apply auto
  done
  also have ... ∈ sets (stream-space M)
  apply (intro predE)
  apply (coinduction rule: measurable-gfp-coinduct)
  apply (auto simp: inf-continuous-def)
  done
  finally show ?thesis .
qed

```

```

lemma sets-stream-space-in-sets:

```

assumes *space*: $\text{space } N = \text{streams } (\text{space } M)$
assumes *sets*: $\bigwedge i. (\lambda x. x !! i) \in \text{measurable } N M$
shows *sets* $(\text{stream-space } M) \subseteq \text{sets } N$
unfolding *stream-space-def sets-distr*
by (*auto intro!*: *sets-image-in-sets measurable-Sup2 measurable-vimage-algebra2*
del: *subsetI equalityI*
simp add: *sets-PiM-eq-proj snth-in space sets cong*: *measurable-cong-sets*)

lemma *sets-stream-space-eq*: $\text{sets } (\text{stream-space } M) =$
 $\text{sets } (\text{SUP } i:\text{UNIV}. \text{vimage-algebra } (\text{streams } (\text{space } M)) (\lambda s. s !! i) M)$
by (*auto intro!*: *sets-stream-space-in-sets sets-Sup-in-sets sets-image-in-sets*
measurable-Sup1 snth-in measurable-vimage-algebra1 del: *subsetI*
simp: *space-Sup-eq-UN space-stream-space*)

lemma *sets-restrict-stream-space*:
assumes $S[\text{measurable}]$: $S \in \text{sets } M$
shows $\text{sets } (\text{restrict-space } (\text{stream-space } M) (\text{streams } S)) = \text{sets } (\text{stream-space } (\text{restrict-space } M S))$
using $S[\text{THEN sets.sets-into-space}]$
apply (*subst restrict-space-eq-vimage-algebra*)
apply (*simp add*: *space-stream-space streams-mono2*)
apply (*subst vimage-algebra-cong[OF refl refl sets-stream-space-eq]*)
apply (*subst sets-stream-space-eq*)
apply (*subst sets-vimage-Sup-eq[where Y=streams (space M)]*)
apply *simp*
apply *auto* []
apply (*auto intro*: *streams-mono*) []
apply *auto* []
apply (*simp add*: *image-image space-restrict-space*)
apply (*simp add*: *vimage-algebra-cong[OF refl refl restrict-space-eq-vimage-algebra]*)
apply (*subst (1 2) vimage-algebra-vimage-algebra-eq*)
apply (*auto simp*: *streams-mono snth-in*)
done

primrec *sstart* :: $'a \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ stream set}$ **where**
 $\text{sstart } S [] = \text{streams } S$
| [*simp del*]: $\text{sstart } S (x \# xs) = \text{op} \#\# x \text{ ' sstart } S xs$

lemma *in-sstart[simp]*: $s \in \text{sstart } S (x \# xs) \iff \text{shd } s = x \wedge \text{stl } s \in \text{sstart } S xs$
by (*cases s*) (*auto simp*: *sstart.simps(2)*)

lemma *sstart-in-streams*: $xs \in \text{lists } S \implies \text{sstart } S xs \subseteq \text{streams } S$
by (*induction xs*) (*auto simp*: *sstart.simps(2)*)

lemma *sstart-eq*: $x \in \text{streams } S \implies x \in \text{sstart } S xs = (\forall i < \text{length } xs. x !! i = xs ! i)$
by (*induction xs arbitrary: x*) (*auto simp*: *nth-Cons streams-stl split: nat.splits*)

lemma *sstart-sets*: $\text{sstart } S xs \in \text{sets } (\text{stream-space } (\text{count-space } \text{UNIV}))$

```

proof (induction xs)
  case (Cons x xs)
  note Cons[measurable]
  have sstart S (x # xs) =
    {s∈space (stream-space (count-space UNIV)). shd s = x ∧ stl s ∈ sstart S xs}
  by (simp add: set-eq-iff space-stream-space)
  also have ... ∈ sets (stream-space (count-space UNIV))
    by measurable
  finally show ?case .
qed (simp add: streams-sets)

```

```

lemma sigma-sets-singletons:
  assumes countable S
  shows sigma-sets S ((λs. {s})'S) = Pow S
proof safe
  interpret sigma-algebra S sigma-sets S ((λs. {s})'S)
  by (rule sigma-algebra-sigma-sets) auto
  fix A assume A ⊆ S
  with assms have (⋃ a∈A. {a}) ∈ sigma-sets S ((λs. {s})'S)
  by (intro countable-UN') (auto dest: countable-subset)
  then show A ∈ sigma-sets S ((λs. {s})'S)
  by simp
qed (auto dest: sigma-sets-into-sp[rotated])

```

```

lemma sets-count-space-eq-sigma:
  countable S ⇒ sets (count-space S) = sets (sigma S ((λs. {s})'S))
  by (subst sets-measure-of) (auto simp: sigma-sets-singletons)

```

```

lemma sets-stream-space-sstart:
  assumes S[simp]: countable S
  shows sets (stream-space (count-space S)) = sets (sigma (streams S) (sstart
S'lists S ∪ {{})))
proof
  have [simp]: sstart S ' lists S ⊆ Pow (streams S)
  by (simp add: image-subset-iff sstart-in-streams)

```

```

let ?S = sigma (streams S) (sstart S ' lists S ∪ {{}))
  { fix i a assume a ∈ S
    { fix x have (x !! i = a ∧ x ∈ streams S) ↔ (∃ xs∈lists S. length xs = i ∧
x ∈ sstart S (xs @ [a]))
      proof (induction i arbitrary: x)
        case (Suc i) from this[of stl x] show ?case
          by (simp add: length-Suc-conv Bex-def ex-simps[symmetric] del: ex-simps)
            (metis stream.collapse streams-Stream)
          qed (insert ⟨a ∈ S⟩, auto intro: streams-stl in-streams) }
    then have (λx. x !! i) -' {a} ∩ streams S = (⋃ xs∈{xs∈lists S. length xs =
i}. sstart S (xs @ [a]))
      by (auto simp add: set-eq-iff)
      also have ... ∈ sets ?S

```


using $\langle a \in S \rangle$ **by** (*intro sets.countable-UN'*) (*auto intro!*: *sigma-sets.Basic image-eqI*)
finally have $(\lambda x. x !! i) - \{a\} \cap \text{streams } S \in \text{sets } ?S . \}$
then show $\text{sets } (\text{stream-space } (\text{count-space } S)) \subseteq \text{sets } (\text{sigma } (\text{streams } S))$ (*sstart S'lists S \cup {{{}}*)
by (*intro sets-stream-space-in-sets*) (*auto simp*: *measurable-count-space-eq-countable snth-in*)

have $\text{sigma-sets } (\text{space } (\text{stream-space } (\text{count-space } S)))$ (*sstart S'lists S \cup {{{}}*)
 $\subseteq \text{sets } (\text{stream-space } (\text{count-space } S))$
proof (*safe intro!*: *sets.sigma-sets-subset*)
fix xs **assume** $\forall x \in \text{set } xs. x \in S$
then have $\text{sstart } S \text{ } xs = \{x \in \text{space } (\text{stream-space } (\text{count-space } S)). \forall i < \text{length } xs. x !! i = xs ! i\}$
by (*induction xs*)
(auto simp: *space-stream-space nth-Cons split*: *nat.split intro*: *in-streams streams-stl*)
also have $\dots \in \text{sets } (\text{stream-space } (\text{count-space } S))$
by *measurable*
finally show $\text{sstart } S \text{ } xs \in \text{sets } (\text{stream-space } (\text{count-space } S)) .$
qed
then show $\text{sets } (\text{sigma } (\text{streams } S))$ (*sstart S'lists S \cup {{{}}*) $\subseteq \text{sets } (\text{stream-space } (\text{count-space } S))$
by (*simp add*: *space-stream-space*)
qed

lemma *Int-stable-sstart*: *Int-stable* (*sstart S'lists S \cup {{{}}*)

proof –

{ fix $xs \ ys$ **assume** $xs \in \text{lists } S \ ys \in \text{lists } S$
then have $\text{sstart } S \text{ } xs \cap \text{sstart } S \text{ } ys \in \text{sstart } S \text{ } \langle \text{lists } S \cup \{\{\}\} \rangle$
proof (*induction xs ys rule*: *list-induct2'*)
case $(_4 \ x \ xs \ y \ ys)$
show *?case*
proof *cases*
assume $x = y$
then have $\text{sstart } S \text{ } (x \# \text{ } xs) \cap \text{sstart } S \text{ } (y \# \text{ } ys) = \text{op } \# \# \ x \text{ } \langle \text{sstart } S \text{ } xs \cap \text{sstart } S \text{ } ys \rangle$
by (*auto simp*: *image-iff intro!*: *stream.collapse[symmetric]*)
also have $\dots \in \text{sstart } S \text{ } \langle \text{lists } S \cup \{\{\}\} \rangle$
using $_4$ **by** (*auto simp*: *sstart.simps(2)[symmetric] del*: *in-listsD*)
finally show *?case* .
qed *auto*
qed (*simp-all add*: *sstart-in-streams inf.absorb1 inf.absorb2 image-eqI* [**where** $x = []$]) }
then show *?thesis*
by (*auto simp*: *Int-stable-def*)
qed

lemma *stream-space-eq-sstart*:

```

assumes  $S[simp]$ : countable  $S$ 
assumes  $P$ : prob-space  $M$  prob-space  $N$ 
assumes  $ae$ :  $AE\ x\ in\ M.\ x \in\ streams\ S\ AE\ x\ in\ N.\ x \in\ streams\ S$ 
assumes  $sets-M$ : sets  $M = sets\ (stream-space\ (count-space\ UNIV))$ 
assumes  $sets-N$ : sets  $N = sets\ (stream-space\ (count-space\ UNIV))$ 
assumes  $*$ :  $\bigwedge xs.\ xs \neq [] \implies xs \in lists\ S \implies emeasure\ M\ (sstart\ S\ xs) =$ 
 $emeasure\ N\ (sstart\ S\ xs)$ 
shows  $M = N$ 
proof (rule  $measure-eqI-restrict-generator[OF\ Int-stable-sstart]$ )
  have  $[simp]$ :  $sstart\ S \text{ ‘ } lists\ S \subseteq Pow\ (streams\ S)$ 
    by ( $simp\ add: image-subset-iff\ sstart-in-streams$ )

  interpret  $M$ : prob-space  $M$  by fact

  show  $sstart\ S \text{ ‘ } lists\ S \cup \{\{\}\} \subseteq Pow\ (streams\ S)$ 
    by ( $auto\ dest: sstart-in-streams\ del: in-listsD$ )

  { fix  $M :: \text{‘}a\ stream\ measure$  assume  $M$ : sets  $M = sets\ (stream-space\ (count-space\ UNIV))$ 
    have sets  $restrict-space\ M\ (streams\ S) = sigma-sets\ (streams\ S)\ (sstart\ S \text{ ‘ } lists\ S \cup \{\{\}\})$ 
      by ( $subst\ sets-restrict-space-cong[OF\ M]$ )
      ( $simp\ add: sets-restrict-stream-space\ restrict-count-space\ sets-stream-space-sstart$ )
    }

  from  $this[OF\ sets-M]\ this[OF\ sets-N]$ 
  show sets  $restrict-space\ M\ (streams\ S) = sigma-sets\ (streams\ S)\ (sstart\ S \text{ ‘ } lists\ S \cup \{\{\}\})$ 
    sets  $restrict-space\ N\ (streams\ S) = sigma-sets\ (streams\ S)\ (sstart\ S \text{ ‘ } lists\ S \cup \{\{\}\})$ 
    by  $auto$ 
  show  $\{streams\ S\} \subseteq sstart\ S \text{ ‘ } lists\ S \cup \{\{\}\}$ 
     $\bigcup \{streams\ S\} = streams\ S \bigwedge s.\ s \in \{streams\ S\} \implies emeasure\ M\ s \neq \infty$ 
    using  $M.emeasure-space-1\ space-stream-space[of\ count-space\ S]\ sets-eq-imp-space-eq[OF\ sets-M]$ 
    by ( $auto\ simp\ add: image-eqI[where\ x=[]]$ )
  show sets  $M = sets\ N$ 
    by ( $simp\ add: sets-M\ sets-N$ )
  next
  fix  $X$  assume  $X \in sstart\ S \text{ ‘ } lists\ S \cup \{\{\}\}$ 
  then obtain  $xs$  where  $X = \{\} \vee (xs \in lists\ S \wedge X = sstart\ S\ xs)$ 
    by  $auto$ 
  moreover have  $emeasure\ M\ (streams\ S) = 1$ 
    using  $ae$  by ( $intro\ prob-space.emeasure-eq-1-AE[OF\ P(1)]$ ) ( $auto\ simp: sets-M\ streams-sets$ )
  moreover have  $emeasure\ N\ (streams\ S) = 1$ 
    using  $ae$  by ( $intro\ prob-space.emeasure-eq-1-AE[OF\ P(2)]$ ) ( $auto\ simp: sets-N\ streams-sets$ )
  ultimately show  $emeasure\ M\ X = emeasure\ N\ X$ 
    using  $P[THEN\ prob-space.emeasure-space-1]$ 

```

by (*cases* $xs = []$) (*auto simp: * space-stream-space del: in-listsD*)
qed (*auto simp: * ae sets-M del: in-listsD intro!: streams-sets*)

lemma *sets-sstart[measurable]*: $sstart \ \Omega \ xs \in \text{sets} \ (\text{stream-space} \ (\text{count-space} \ UNIV))$

proof (*induction xs*)

case (*Cons x xs*)

note *this[measurable]*

have $sstart \ \Omega \ (x \ \# \ xs) = \{\omega \in \text{space} \ (\text{stream-space} \ (\text{count-space} \ UNIV)). \ \omega \in sstart \ \Omega \ (x \ \# \ xs)\}$

by (*auto simp: space-stream-space*)

also have $\dots \in \text{sets} \ (\text{stream-space} \ (\text{count-space} \ UNIV))$

unfolding *in-sstart by measurable*

finally show *?case .*

qed (*auto intro!: streams-sets*)

primrec *scylinder* :: $'a \ \text{set} \Rightarrow 'a \ \text{set list} \Rightarrow 'a \ \text{stream set}$

where

$scylinder \ S \ [] = \text{streams} \ S$

$| \ scylinder \ S \ (A \ \# \ As) = \{\omega \in \text{streams} \ S. \ shd \ \omega \in A \wedge \ stl \ \omega \in scylinder \ S \ As\}$

lemma *scylinder-streams*: $scylinder \ S \ xs \subseteq \text{streams} \ S$

by (*induction xs auto*)

lemma *sets-scylinder*: $(\forall x \in \text{set} \ xs. \ x \in \text{sets} \ S) \Longrightarrow scylinder \ (\text{space} \ S) \ xs \in \text{sets} \ (\text{stream-space} \ S)$

by (*induction xs (auto simp: space-stream-space[symmetric])*)

lemma *stream-space-eq-scylinder*:

assumes *P: prob-space M prob-space N*

assumes *Int-stable G and S: sets S = sets (sigma (space S) G)*

and *C: countable C C ⊆ G ∪ C = space S and G: G ⊆ Pow (space S)*

assumes *sets-M: sets M = sets (stream-space S)*

assumes *sets-N: sets N = sets (stream-space S)*

assumes $*$: $\bigwedge xs. \ xs \neq [] \Longrightarrow xs \in \text{lists} \ G \Longrightarrow \text{emeasure} \ M \ (scylinder \ (\text{space} \ S) \ xs) = \text{emeasure} \ N \ (scylinder \ (\text{space} \ S) \ xs)$

shows $M = N$

proof (*rule measure-eqI-generator-eq*)

interpret *M: prob-space M by fact*

interpret *N: prob-space N by fact*

let $?G = scylinder \ (\text{space} \ S) \ \text{'lists} \ G$

show *sc-Pow: ?G ⊆ Pow (streams (space S))*

using *scylinder-streams by auto*

have $\text{sets} \ (\text{stream-space} \ S) = \text{sets} \ (\text{sigma} \ (\text{streams} \ (\text{space} \ S)) \ ?G)$

(*is ?S = sets ?R*)

proof (*rule antisym*)

let $?V = \lambda i. \ \text{vimage-algebra} \ (\text{streams} \ (\text{space} \ S)) \ (\lambda s. \ s \ !! \ i) \ S$

show $?S \subseteq \text{sets} \ ?R$

```

unfolding sets-stream-space-eq
proof (safe intro!: sets-Sup-in-sets del: subsetI equalityI)
  fix i :: nat
  show space (?V i) = space ?R
    using scylinder-streams by (subst space-measure-of) (auto simp: )
    { fix A assume A ∈ G
      then have scylinder (space S) (replicate i (space S) @ [A]) = (λs. s !! i)
    - ‘ A ∩ streams (space S)
      by (induction i) (auto simp add: streams-shd streams-stl cong: conj-cong)
      also have scylinder (space S) (replicate i (space S) @ [A]) = (∪ xs∈{xs∈lists C.
    C. length xs = i}. scylinder (space S) (xs @ [A]))
      apply (induction i)
      apply auto []
      apply (simp add: length-Suc-conv set-eq-iff ex-simps(1,2)[symmetric] cong:
    conj-cong del: ex-simps(1,2))
      apply rule
      subgoal for i x
        apply (cases x)
        apply (subst (2) C(3)[symmetric])
        apply (simp del: ex-simps(1,2) add: ex-simps(1,2)[symmetric] ac-simps
    Bex-def)
        apply auto
        done
      done
      finally have (λs. s !! i) - ‘ A ∩ streams (space S) = (∪ xs∈{xs∈lists C.
    length xs = i}. scylinder (space S) (xs @ [A]))
      ..
      also have ... ∈ ?R
        using C(2) ⟨A∈G⟩
        by (intro sets.countable-UN' countable-Collect countable-lists C)
          (auto intro!: in-measure-of[OF sc-Pow] imageI)
      finally have (λs. s !! i) - ‘ A ∩ streams (space S) ∈ ?R . }
    then show sets (?V i) ⊆ ?R
      apply (subst vimage-algebra-cong[OF refl refl S])
      apply (subst vimage-algebra-sigma[OF G])
      apply (simp add: streams-iff-snth) []
      apply (subst sigma-le-sets)
      apply auto
      done
    qed
  have G ⊆ sets S
    unfolding S using G by auto
  with C(2) show sets ?R ⊆ ?S
    unfolding sigma-le-sets[OF sc-Pow] by (auto intro!: sets-scylinder)
  qed
  then show sets M = sigma-sets (streams (space S)) (scylinder (space S) ‘ lists
    G)
    sets N = sigma-sets (streams (space S)) (scylinder (space S) ‘ lists G)
  unfolding sets-M sets-N by (simp-all add: sc-Pow)

```

```

show Int-stable ?G
proof (rule Int-stableI-image)
  fix xs ys assume xs ∈ lists G ys ∈ lists G
  then show ∃ zs ∈ lists G. scylinder (space S) xs ∩ scylinder (space S) ys =
scylinder (space S) zs
  proof (induction xs arbitrary: ys)
    case Nil then show ?case
      by (auto simp add: Int-absorb1 scylinder-streams)
  next
  case xs: (Cons x xs)
  show ?case
  proof (cases ys)
    case Nil with xs.hyps show ?thesis
      by (auto simp add: Int-absorb2 scylinder-streams intro!: bexI[of - x#xs])
  next
  case ys: (Cons y ys')
  with xs.IH[of ys'] xs.prems obtain zs where
    zs ∈ lists G and eq: scylinder (space S) xs ∩ scylinder (space S) ys' =
scylinder (space S) zs
    by auto
  show ?thesis
  proof (intro bexI[of - (x ∩ y)#zs])
    show x ∩ y # zs ∈ lists G
    using ⟨zs ∈ lists G⟩ ⟨x ∈ G⟩ ⟨ys ∈ lists G⟩ ys (Int-stable G) [THEN Int-stableD,
of x y] by auto
    show scylinder (space S) (x # xs) ∩ scylinder (space S) ys = scylinder
(space S) (x ∩ y # zs)
    by (auto simp add: eq[symmetric] ys)
  qed
qed
qed
qed

```

```

show range (λ-::nat. streams (space S)) ⊆ scylinder (space S) ‘ lists G
  (∪ i. streams (space S)) = streams (space S)
  emeasure M (streams (space S)) ≠ ∞
  by (auto intro!: image-eqI[of - - []])

```

```

fix X assume X ∈ scylinder (space S) ‘ lists G
then obtain xs where xs: xs ∈ lists G and eq: X = scylinder (space S) xs
  by auto
then show emeasure M X = emeasure N X
proof (cases xs = [])
  assume xs = [] then show ?thesis
    unfolding eq
    using sets-M [THEN sets-eq-imp-space-eq] sets-N [THEN sets-eq-imp-space-eq]
      M.emeasure-space-1 N.emeasure-space-1
    by (simp add: space-stream-space[symmetric])

```

```

next
  assume  $xs \neq []$  with  $xs$  show ?thesis
  unfolding eq by (intro *)
qed
qed

lemma stream-space-coinduct:
  fixes  $R :: 'a \text{ stream measure} \Rightarrow 'a \text{ stream measure} \Rightarrow \text{bool}$ 
  assumes  $R A B$ 
  assumes  $R: \bigwedge A B. R A B \implies \exists K \in \text{space} (\text{prob-algebra } M).$ 
   $\exists A' \in M \rightarrow_M \text{prob-algebra} (\text{stream-space } M). \exists B' \in M \rightarrow_M \text{prob-algebra} (\text{stream-space } M).$ 
  ( $\exists y \text{ in } K. R (A' y) (B' y) \vee A' y = B' y$ )  $\wedge$ 
   $A = \text{do } \{ y \leftarrow K; \omega \leftarrow A' y; \text{return } (\text{stream-space } M) (y \#\#\omega) \}$   $\wedge$ 
   $B = \text{do } \{ y \leftarrow K; \omega \leftarrow B' y; \text{return } (\text{stream-space } M) (y \#\#\omega) \}$ 
  shows  $A = B$ 
proof (rule stream-space-eq-cylinder)
  let ?step =  $\lambda K L. \text{do } \{ y \leftarrow K; \omega \leftarrow L y; \text{return } (\text{stream-space } M) (y \#\#\omega) \}$ 
  { fix  $K A A'$  assume  $K: K \in \text{space} (\text{prob-algebra } M)$ 
    and  $A'[\text{measurable}]: A' \in M \rightarrow_M \text{prob-algebra} (\text{stream-space } M)$  and  $A\text{-eq}: A = ?step K A'$ 
    have  $ps: \text{prob-space } A$ 
    unfolding  $A\text{-eq}$  by (rule prob-space-bind'[OF K]) measurable
    have  $sets A = \text{sets} (\text{stream-space } M)$ 
    unfolding  $A\text{-eq}$  by (rule sets-bind'[OF K]) measurable
    note  $ps \text{ this}$  }
  note  $** = \text{this}$ 

  { fix  $A B$  assume  $R A B$ 
    obtain  $K A' B'$  where  $K: K \in \text{space} (\text{prob-algebra } M)$ 
    and  $A': A' \in M \rightarrow_M \text{prob-algebra} (\text{stream-space } M) A = ?step K A'$ 
    and  $B': B' \in M \rightarrow_M \text{prob-algebra} (\text{stream-space } M) B = ?step K B'$ 
    using  $R[\text{OF } \langle R A B \rangle]$  by blast
    have  $\text{prob-space } A \text{ prob-space } B \text{ sets } A = \text{sets} (\text{stream-space } M) \text{ sets } B = \text{sets} (\text{stream-space } M)$ 
    using  $**[\text{OF } K A'] **[\text{OF } K B']$  by auto }
  note  $R\text{-D} = \text{this}$ 

  show  $\text{prob-space } A \text{ prob-space } B \text{ sets } A = \text{sets} (\text{stream-space } M) \text{ sets } B = \text{sets} (\text{stream-space } M)$ 
  using  $R\text{-D}[\text{OF } \langle R A B \rangle]$  by auto

  show  $\text{Int-stable} (\text{sets } M) \text{ sets } M = \text{sets} (\text{sigma} (\text{space } M) (\text{sets } M)) \text{ countable } \{\text{space } M\}$ 
   $\{\text{space } M\} \subseteq \text{sets } M \cup \{\text{space } M\} = \text{space } M \text{ sets } M \subseteq \text{Pow} (\text{space } M)$ 
  using  $\text{sets.space-closed}[\text{of } M]$  by (auto simp:  $\text{Int-stable-def}$ )

  { fix  $A As L K$  assume  $K[\text{measurable}]: K \in \text{space} (\text{prob-algebra } M)$  and  $A: A \in \text{sets } M As \in \text{lists} (\text{sets } M)$ 

```

```

    and L[measurable]: L ∈ M →M prob-algebra (stream-space M)
  from A have [measurable]: ∀ x ∈ set (A # As). x ∈ sets M ∨ x ∈ set As. x ∈ sets
M
  by auto
  have [simp]: space K = space M sets K = sets M
  using K by (auto simp: space-prob-algebra intro!: sets-eq-imp-space-eq)
  have [simp]: x ∈ space M ⇒ sets (L x) = sets (stream-space M) for x
  using measurable-space[OF L] by (auto simp: space-prob-algebra)
  note sets-scylinder[measurable]
  have *: indicator (scylinder (space M) (A # As)) (x ## ω) =
    (indicator A x * indicator (scylinder (space M) As) ω) :: ennreal for ω x
  using scylinder-streams[of space M As] ⟨A ∈ sets M⟩ [THEN sets-sets-into-space]
  by (auto split: split-indicator)
  have emeasure (?step K L) (scylinder (space M) (A # As)) = (∫+ y. L y
(scylinder (space M) As) * indicator A y ∂K)
  apply (subst emeasure-bind-prob-algebra[OF K])
  apply measurable
  apply (rule nn-integral-cong)
  apply (subst emeasure-bind-prob-algebra[OF L [THEN measurable-space]])
  apply (simp-all add: ac-simps * nn-integral-cmult-indicator del: scylin-
der.simps)
  apply measurable
  done }
  note emeasure-step = this

  fix Xs assume Xs ∈ lists (sets M)
  from this ⟨R A B⟩ show emeasure A (scylinder (space M) Xs) = emeasure B
(scylinder (space M) Xs)
  proof (induction Xs arbitrary: A B)
  case (Cons X Xs)
  obtain K A' B' where K: K ∈ space (prob-algebra M)
  and A'[measurable]: A' ∈ M →M prob-algebra (stream-space M) and A: A
= ?step K A'
  and B'[measurable]: B' ∈ M →M prob-algebra (stream-space M) and B: B
= ?step K B'
  and AE-R: AE x in K. R (A' x) (B' x) ∨ A' x = B' x
  using R[OF ⟨R A B⟩] by blast

  show ?case
  unfolding A B emeasure-step[OF K Cons.hyps A] emeasure-step[OF K
Cons.hyps B]
  apply (rule nn-integral-cong-AE)
  using AE-R by eventually-elim (auto simp add: Cons.IH)
next
case Nil
note R-D[OF this]
from this(1,2) [THEN prob-space.emeasure-space-1] this(3,4) [THEN sets-eq-imp-space-eq]
show ?case
by (simp add: space-stream-space)

```

qed
 qed
 end

theory *Tree-Space*

imports *HOL-Analysis.Analysis HOL-Library.Tree*
begin

lemma *countable-lfp*:

assumes *step*: $\bigwedge Y. \text{countable } Y \implies \text{countable } (F Y)$
and *cont*: *Order-Continuity.sup-continuous* *F*
shows *countable* (*lfp F*)
by(*subst sup-continuous-lfp[OF cont]*)(*simp add: countable-funpow[OF step]*)

lemma *countable-lfp-apply*:

assumes *step*: $\bigwedge Y x. (\bigwedge x. \text{countable } (Y x)) \implies \text{countable } (F Y x)$
and *cont*: *Order-Continuity.sup-continuous* *F*
shows *countable* (*lfp F x*)

proof –

{ **fix** *n*
have $\bigwedge x. \text{countable } ((F \hat{\ } n) \text{ bot } x)$
by(*induct n*)(*auto intro: step*) }
thus *?thesis using cont by*(*simp add: sup-continuous-lfp*)
 qed

primrec *left* :: 'a tree \Rightarrow 'a tree

where

left (*Node l v r*) = *l*
 | *left Leaf* = *Leaf*

primrec *right* :: 'a tree \Rightarrow 'a tree

where

right (*Node l v r*) = *r*
 | *right Leaf* = *Leaf*

inductive-set *trees* :: 'a set \Rightarrow 'a tree set **for** *S* :: 'a set **where**

[*intro!*]: *Leaf* \in *trees S*
 | $l \in \text{trees } S \implies r \in \text{trees } S \implies v \in S \implies \text{Node } l v r \in \text{trees } S$

lemma *Node-in-trees-iff*[*simp*]: $\text{Node } l v r \in \text{trees } S \iff (l \in \text{trees } S \wedge v \in S \wedge r \in \text{trees } S)$

by (*subst trees.simps*) *auto*

lemma *trees-sub-lfp*: $\text{trees } S \subseteq \text{lfp } (\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{\text{Node } l v r\}))))$

proof

have *mono*: *mono* $(\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{\text{Node } l v r\}))))$


```

r}}))
  by (auto simp: mono-def)
  fix t assume t ∈ trees S then show t ∈ lfp (λT. T ∪ {Leaf} ∪ (∪ l ∈ T. (∪ v ∈ S.
(∪ r ∈ T. {Node l v r})))
  proof induction
    case 1 then show ?case
      by (subst lfp-unfold[OF mono]) auto
    next
      case 2 then show ?case
        by (subst lfp-unfold[OF mono]) auto
  qed
qed

```

lemma *countable-trees*: $\text{countable } A \implies \text{countable } (\text{trees } A)$

```

proof (intro countable-subset[OF trees-sub-lfp] countable-lfp
  sup-continuous-sup sup-continuous-const sup-continuous-id)
  show sup-continuous (λT. (∪ l ∈ T. ∪ v ∈ A. ∪ r ∈ T. {\langle l, v, r \rangle}))
    unfolding sup-continuous-def
  proof (intro allI impI equalityI subsetI, goal-cases)
    case (1 M t)
    then obtain i j :: nat and l x r where t = Node l x r x ∈ A l ∈ M i r ∈ M j
      by auto
    hence l ∈ M (max i j) r ∈ M (max i j)
      using incseqD[OF \langle incseq M \rangle, of i max i j] incseqD[OF \langle incseq M \rangle, of j max
i j] by auto
    with \langle t = Node l x r \rangle and \langle x ∈ A \rangle show ?case by auto
  qed auto
qed auto

```

lemma *trees-UNIV*[simp]: $\text{trees UNIV} = \text{UNIV}$

```

proof –
  have t ∈ trees UNIV for t :: 'a tree
    by (induction t) (auto intro: trees.intros(2))
  then show ?thesis by auto
qed

```

instance *tree* :: (countable) countable

```

proof
  have countable (UNIV :: 'a tree set)
    by (subst trees-UNIV[symmetric]) (intro countable-trees[OF countableI-type])
  then show ∃ to-nat :: 'a tree ⇒ nat. inj to-nat
    by (auto simp: countable-def)
qed

```

lemma *map-in-trees*[intro]: $(\bigwedge x. x \in \text{set-tree } t \implies f x \in S) \implies \text{map-tree } f t \in \text{trees } S$

```

  by (induction t) (auto intro: trees.intros(2))

```

primrec *trees-cyl* :: 'a set tree ⇒ 'a tree set **where**

$trees-cyl\ Leaf = \{Leaf\}$
 $| trees-cyl (Node\ l\ v\ r) = (\bigcup_{l' \in trees-cyl\ l}. (\bigcup_{v' \in v}. (\bigcup_{r' \in trees-cyl\ r}. \{Node\ l'\ v'\ r'\})))$

definition $tree-sigma :: 'a\ measure \Rightarrow 'a\ tree\ measure$

where

$tree-sigma\ M = sigma\ (trees\ (space\ M))\ (trees-cyl\ ' trees\ (sets\ M))$

lemma $Node-in-trees-cyl: Node\ l'\ v'\ r' \in trees-cyl\ t \longleftrightarrow$

$(\exists l\ v\ r. t = Node\ l\ v\ r \wedge l' \in trees-cyl\ l \wedge r' \in trees-cyl\ r \wedge v' \in v)$

by $(cases\ t)\ auto$

lemma $trees-cyl-sub-trees:$

assumes $t \in trees\ A\ A \subseteq Pow\ B$ **shows** $trees-cyl\ t \subseteq trees\ B$

using $assms(1)$

proof $induction$

case $(2\ l\ v\ r)$ **with** $\langle A \subseteq Pow\ B \rangle$ **show** $?case$

by $(auto\ intro!: trees.intros(2))$

qed $auto$

lemma $trees-cyl-sets-in-space: trees-cyl\ ' trees\ (sets\ M) \subseteq Pow\ (trees\ (space\ M))$

using $trees-cyl-sub-trees[OF\ -\ sets.space-closed, of\ -\ M]$ **by** $auto$

lemma $space-tree-sigma: space\ (tree-sigma\ M) = trees\ (space\ M)$

unfolding $tree-sigma-def$ **by** $(rule\ space-measure-of-conv)$

lemma $sets-tree-sigma-eq: sets\ (tree-sigma\ M) = sigma-sets\ (trees\ (space\ M))$
 $(trees-cyl\ ' trees\ (sets\ M))$

unfolding $tree-sigma-def$ **by** $(rule\ sets-measure-of)\ (rule\ trees-cyl-sets-in-space)$

lemma $Leaf-in-space-tree-sigma\ [measurable, simp, intro]: Leaf \in space\ (tree-sigma\ M)$

by $(auto\ simp: space-tree-sigma)$

lemma $Leaf-in-tree-sigma\ [measurable, simp, intro]: \{Leaf\} \in sets\ (tree-sigma\ M)$

unfolding $sets-tree-sigma-eq$

by $(rule\ sigma-sets.Basic)\ (auto\ intro: trees.intros(2)\ image-eqI[\mathbf{where}\ x=Leaf])$

lemma $trees-cyl-map-treeI: t \in trees-cyl\ (map-tree\ (\lambda x. A)\ t)$ **if** $*$: $t \in trees\ A$

using $*$ **by** $induction\ auto$

lemma $trees-cyl-map-in-sets:$

$(\bigwedge x. x \in set-tree\ t \implies f\ x \in sets\ M) \implies trees-cyl\ (map-tree\ f\ t) \in sets\ (tree-sigma\ M)$

by $(subst\ sets-tree-sigma-eq)\ auto$

lemma $Node-in-tree-sigma:$

assumes $L: X \in sets\ (M \otimes_M (tree-sigma\ M) \otimes_M tree-sigma\ M)$

shows $\{Node\ l\ v\ r \mid l\ v\ r. (v, l, r) \in X\} \in sets\ (tree-sigma\ M)$

proof –

let $?E = \lambda s::\text{unit tree. trees-cyl } (\text{map-tree } (\lambda-. \text{ space } M) s)$
have $1: \text{countable } (\text{range } ?E)$
by (*intro countable-image countableI-type*)
have $2: \text{trees-cyl } \text{‘ trees } (\text{sets } M) \subseteq \text{Pow } (\text{space } (\text{tree-sigma } M))$
using *trees-cyl-sets-in-space[of M]* **by** (*simp add: space-tree-sigma*)
have $3: \text{sets } (\text{tree-sigma } M) = \text{sigma-sets } (\text{space } (\text{tree-sigma } M))$ (*trees-cyl* ‘
trees (sets M))
unfolding *sets-tree-sigma-eq* **by** (*simp add: space-tree-sigma*)
have $4: (\bigcup s. ?E s) = \text{space } (\text{tree-sigma } M)$
proof (*safe; clarsimp simp: space-tree-sigma*)
fix $t s$ **assume** $t \in \text{trees-cyl } (\text{map-tree } (\lambda::\text{unit. space } M) s)$
then show $t \in \text{trees } (\text{space } M)$
by (*induction s arbitrary: t*) *auto*
next
fix t **assume** $t \in \text{trees } (\text{space } M)$
then show $\exists t'. t \in ?E t'$
by (*intro exI[of - map-tree (\lambda-. ()) t]*)
(auto simp: tree.map-comp comp-def intro: trees-cyl-map-treeI)
qed
have $5: \text{range } ?E \subseteq \text{trees-cyl } \text{‘ trees } (\text{sets } M)$ **by** *auto*
let $?P = \{A \times B \mid A B. A \in \text{trees-cyl } \text{‘ trees } (\text{sets } M) \wedge B \in \text{trees-cyl } \text{‘ trees } (\text{sets } M)\}$
have $P: \text{sets } (\text{tree-sigma } M \otimes_M \text{tree-sigma } M) = \text{sets } (\text{sigma } (\text{space } (\text{tree-sigma } M) \times \text{space } (\text{tree-sigma } M))) ?P$
by (*rule sets-pair-eq[OF 2 3 1 5 4 2 3 1 5 4]*)

have $\text{sets } (M \otimes_M (\text{tree-sigma } M \otimes_M \text{tree-sigma } M)) =$
 $\text{sets } (\text{sigma } (\text{space } M \times \text{space } (\text{tree-sigma } M \otimes_M \text{tree-sigma } M))) \{A \times BC \mid$
 $A BC. A \in \text{sets } M \wedge BC \in ?P\}$

proof (*rule sets-pair-eq*)

show $\text{sets } M \subseteq \text{Pow } (\text{space } M)$ $\text{sets } M = \text{sigma-sets } (\text{space } M)$ (*sets M*)
by (*auto simp: sets.sigma-sets-eq sets.space-closed*)

show $\text{countable } \{\text{space } M\} \{\text{space } M\} \subseteq \text{sets } M \cup \{\text{space } M\} = \text{space } M$
by *auto*

show $?P \subseteq \text{Pow } (\text{space } (\text{tree-sigma } M \otimes_M \text{tree-sigma } M))$

using *trees-cyl-sets-in-space[of M]*

by (*auto simp: space-pair-measure space-tree-sigma subset-eq*)

then show $\text{sets } (\text{tree-sigma } M \otimes_M \text{tree-sigma } M) =$

$\text{sigma-sets } (\text{space } (\text{tree-sigma } M \otimes_M \text{tree-sigma } M)) ?P$

by (*subst P, subst sets-measure-of*) (*auto simp: space-tree-sigma space-pair-measure*)

show $\text{countable } ((\lambda(a, b). a \times b) \text{ ‘ } (\text{range } ?E \times \text{range } ?E))$

by (*intro countable-image countable-SIGMA countableI-type*)

show $(\lambda(a, b). a \times b) \text{ ‘ } (\text{range } ?E \times \text{range } ?E) \subseteq ?P$

by *auto*

qed (*insert 4, auto simp: space-pair-measure space-tree-sigma set-eq-iff*)

also have $\dots = \text{sigma-sets } (\text{space } M \times \text{trees } (\text{space } M) \times \text{trees } (\text{space } M))$

$\{A \times BC \mid A BC. A \in \text{sets } M \wedge BC \in \{A \times B \mid A B.$

$A \in \text{trees-cyl } \text{‘ trees } (\text{sets } M) \wedge B \in \text{trees-cyl } \text{‘ trees } (\text{sets } M)\}$

```

(is - = sigma-sets ?X ?Y) using sets.space-closed[of M] trees-cyl-sub-trees[of -
sets M space M]
  by (subst sets-measure-of)
    (auto simp: space-pair-measure space-tree-sigma)
  also have ?Y = {A × trees-cyl B × trees-cyl C | A B C. A ∈ sets M ∧
    B ∈ trees (sets M) ∧ C ∈ trees (sets M)} by blast
  finally have X ∈ sigma-sets (space M × trees (space M) × trees (space M))
    {A × trees-cyl B × trees-cyl C | A B C. A ∈ sets M ∧ B ∈ trees (sets M) ∧
    C ∈ trees (sets M) }
    using assms by blast
  then show ?thesis
  proof induction
    case (Basic A')
      then obtain A B C where A' = A × trees-cyl B × trees-cyl C
        and *: A ∈ sets M B ∈ trees (sets M) C ∈ trees (sets M)
        by auto
      then have {Node l v r | l v r. (v, l, r) ∈ A'} = trees-cyl (Node B A C)
        by auto
      then show ?case
        by (auto simp del: trees-cyl.simps simp: sets-tree-sigma-eq intro!: sigma-sets.Basic
*)
    next
      case Empty show ?case by auto
    next
      case (Compl A)
        have {Node l v r | l v r. (v, l, r) ∈ space M × trees (space M) × trees (space
M) - A} =
          (space (tree-sigma M) - {Node l v r | l v r. (v, l, r) ∈ A}) - {Leaf}
          by (auto simp: space-tree-sigma elim: trees.cases)
        also have ... ∈ sets (tree-sigma M)
          by (intro sets.Diff Compl) auto
        finally show ?case .
    next
      case (Union I)
        have *: {Node l v r | l v r. (v, l, r) ∈ UNION UNIV I} =
          (⋃ i. {Node l v r | l v r. (v, l, r) ∈ I i}) by auto
        show ?case unfolding * using Union(2) by (intro sets.countable-UN) auto
  qed
qed

```

lemma *measurable-left[measurable]*: $left \in tree\text{-}\sigma M \rightarrow_M tree\text{-}\sigma M$

proof (*rule measurableI*)

show $t \in space (tree\text{-}\sigma M) \implies left t \in space (tree\text{-}\sigma M)$ **for** t

by (*cases t*) (*auto simp: space-tree-sigma*)

fix A **assume** $A: A \in sets (tree\text{-}\sigma M)$

from *sets.sets-into-space[OF this]*

have *: $left - 'A \cap space (tree\text{-}\sigma M) =$

(*if Leaf* $\in A$ *then* {Leaf} *else* {}) \cup

{Node $a v r$ | $a v r. (v, a, r) \in space M \times A \times space (tree\text{-}\sigma M)$ }

by (auto simp: space-tree-sigma elim: trees.cases)
 show left - ‘ $A \cap \text{space } (\text{tree-sigma } M) \in \text{sets } (\text{tree-sigma } M)$
 unfolding * using A by (intro sets.Un Node-in-tree-sigma pair-measureI) auto
 qed

lemma measurable-right[measurable]: $\text{right} \in \text{tree-sigma } M \rightarrow_M \text{tree-sigma } M$
proof (rule measurableI)
 show $t \in \text{space } (\text{tree-sigma } M) \implies \text{right } t \in \text{space } (\text{tree-sigma } M)$ for t
 by (cases t) (auto simp: space-tree-sigma)
 fix A assume A: $A \in \text{sets } (\text{tree-sigma } M)$
 from sets.sets-into-space[OF this]
 have *: $\text{right} - ‘ A \cap \text{space } (\text{tree-sigma } M) =$
 (if Leaf $\in A$ then {Leaf} else {}) \cup
 {Node l v a | l v a. $(v, l, a) \in \text{space } M \times \text{space } (\text{tree-sigma } M) \times A$ }
 by (auto simp: space-tree-sigma elim: trees.cases)
 show right - ‘ $A \cap \text{space } (\text{tree-sigma } M) \in \text{sets } (\text{tree-sigma } M)$
 unfolding * using A by (intro sets.Un Node-in-tree-sigma pair-measureI) auto
 qed

lemma measurable-root-val': $\text{root-val} \in \text{restrict-space } (\text{tree-sigma } M) (-\{\text{Leaf}\})$
 $\rightarrow_M M$
proof (rule measurableI)
 show $t \in \text{space } (\text{restrict-space } (\text{tree-sigma } M) (-\{\text{Leaf}\})) \implies \text{root-val } t \in \text{space } M$ for t
 by (cases t) (auto simp: space-restrict-space space-tree-sigma)
 fix A assume A: $A \in \text{sets } M$
 from sets.sets-into-space[OF this]
 have root-val - ‘ $A \cap \text{space } (\text{restrict-space } (\text{tree-sigma } M) (-\{\text{Leaf}\})) =$
 {Node l a r | l a r. $(a, l, r) \in A \times \text{space } (\text{tree-sigma } M) \times \text{space } (\text{tree-sigma } M)$ }
 by (auto simp: space-tree-sigma space-restrict-space elim: trees.cases)
 also have ... $\in \text{sets } (\text{tree-sigma } M)$
 using A by (intro sets.Un Node-in-tree-sigma pair-measureI) auto
 finally show root-val - ‘ $A \cap \text{space } (\text{restrict-space } (\text{tree-sigma } M) (-\{\text{Leaf}\}))$
 \in
 sets (restrict-space (tree-sigma M) (- {Leaf}))
 by (auto simp: sets-restrict-space-iff space-restrict-space)
 qed

lemma measurable-restrict-mono:
 assumes $f: f \in \text{restrict-space } M A \rightarrow_M N$ and $B \subseteq A$
 shows $f \in \text{restrict-space } M B \rightarrow_M N$
by (rule measurable-compose[OF measurable-restrict-space3 f])
 (insert $\langle B \subseteq A \rangle$, auto)

lemma measurable-root-val[measurable (raw)]:
 assumes $f \in X \rightarrow_M \text{tree-sigma } M$
 and $\bigwedge x. x \in \text{space } X \implies f x \neq \text{Leaf}$

shows $(\lambda\omega. \text{root-val } (f \ \omega)) \in X \rightarrow_M M$
proof –
 from *assms* have $f \in X \rightarrow_M \text{restrict-space } (\text{tree-sigma } M) \ (- \ \{\text{Leaf}\})$
 by (*intro measurable-restrict-space2*) *auto*
 from *this* and *measurable-root-val'* **show** *?thesis* by (*rule measurable-compose*)
qed

lemma *measurable-Node* [*measurable*]:

$(\lambda(l,x,r). \text{Node } l \ x \ r) \in \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \rightarrow_M \text{tree-sigma } M$

proof (*rule measurable-sigma-sets*)

show $\text{sets } (\text{tree-sigma } M) = \text{sigma-sets } (\text{trees } (\text{space } M)) \ (\text{trees-cyl } \text{' trees } (\text{sets } M))$

by (*simp add: sets-tree-sigma-eq*)

show $\text{trees-cyl } \text{' trees } (\text{sets } M) \subseteq \text{Pow } (\text{trees } (\text{space } M))$

by (*rule trees-cyl-sets-in-space*)

show $(\lambda(l, x, r). \langle l, x, r \rangle) \in \text{space } (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M) \rightarrow \text{trees } (\text{space } M)$

by (*auto simp: space-pair-measure space-tree-sigma*)

fix *A* **assume** *t*: $A \in \text{trees-cyl } \text{' trees } (\text{sets } M)$

then obtain *t* **where** $t: t \in \text{trees } (\text{sets } M) \ A = \text{trees-cyl } t$ **by** *auto*

show $(\lambda(l, x, r). \langle l, x, r \rangle) - \text{' } A \cap$

$\text{space } (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M)$

$\in \text{sets } (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M)$

proof (*cases t*)

case *Leaf*

have $(\lambda(l, x, r). \langle l, x, r \rangle) - \text{' } \{\text{Leaf} :: \text{'a tree}\} = \{\}$ **by** *auto*

with *Leaf* **show** *?thesis* **using** *t* **by** *simp*

next

case $(\text{Node } l \ B \ r)$

hence $(\lambda(l, x, r). \langle l, x, r \rangle) - \text{' } A \cap \text{space } (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M) =$

$\text{trees-cyl } l \times B \times \text{trees-cyl } r$

using *t* **and** *Node* **and** *trees-cyl-sub-trees*[*of - sets M space M*]

by (*auto simp: space-pair-measure space-tree-sigma*)

dest: sets.sets-into-space[*of - M*])

thus *?thesis* **using** *t* **and** *Node*

by (*auto intro!: pair-measureI simp: sets-tree-sigma-eq*)

qed

qed

lemma *measurable-Node'* [*measurable (raw)*]:

assumes [*measurable*]: $l \in B \rightarrow_M \text{tree-sigma } A$

assumes [*measurable*]: $x \in B \rightarrow_M A$

assumes [*measurable*]: $r \in B \rightarrow_M \text{tree-sigma } A$

shows $(\lambda y. \text{Node } (l \ y) \ (x \ y) \ (r \ y)) \in B \rightarrow_M \text{tree-sigma } A$

proof –

have $(\lambda y. \text{Node } (l \ y) \ (x \ y) \ (r \ y)) = (\lambda(a,b,c). \text{Node } a \ b \ c) \circ (\lambda y. (l \ y, x \ y, r \ y))$

by (*simp add: o-def*)
 also have $\dots \in B \rightarrow_M \text{tree-sigma } A$
 by (*intro measurable-comp[OF - measurable-Node]*) *simp-all*
 finally show *?thesis* .
 qed

lemma *measurable-rec-tree[measurable (raw)]*:

assumes $t: t \in B \rightarrow_M \text{tree-sigma } M$
 assumes $l: l \in B \rightarrow_M A$
 assumes $n: (\lambda(x, l, v, r, al, ar). n\ x\ l\ v\ r\ al\ ar) \in$
 $(B \otimes_M \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \otimes_M A \otimes_M A) \rightarrow_M A$ (*is*
 $?N \in ?M \rightarrow_M A$)
 shows $(\lambda x. \text{rec-tree } (l\ x) (n\ x) (t\ x)) \in B \rightarrow_M A$
proof (*rule measurable-piecewise-restrict*)
 let $?C = \lambda t. \lambda s::\text{unit tree}. t - ' \text{trees-cyl } (\text{map-tree } (\lambda-. \text{space } M) s)$
 show *countable* (*range* ($?C\ t$)) **by** (*intro countable-image countableI-type*)
 show *space* $B \subseteq (\bigcup s. ?C\ t\ s)$
proof (*safe; clarsimp*)
 fix x **assume** $x: x \in \text{space } B$ **have** $t\ x \in \text{trees } (\text{space } M)$
 using t [*THEN measurable-space, OF x*] **by** (*simp add: space-tree-sigma*)
 then show $\exists xa::\text{unit tree}. t\ x \in \text{trees-cyl } (\text{map-tree } (\lambda-. \text{space } M) xa)$
 by (*intro exI[of - map-tree (\lambda-. ()) (t x)]*)
 (*simp add: tree.map-comp comp-def trees-cyl-map-treeI*)

qed

fix Ω **assume** $\Omega \in \text{range } (?C\ t)$

then **obtain** $s::\text{unit tree}$ **where** $\Omega: \Omega = ?C\ t\ s$ **by** *auto*

then **show** $\Omega \cap \text{space } B \in \text{sets } B$

by (*safe intro!: measurable-sets[OF t] trees-cyl-map-in-sets*)

show $(\lambda x. \text{rec-tree } (l\ x) (n\ x) (t\ x)) \in \text{restrict-space } B\ \Omega \rightarrow_M A$

unfolding Ω **using** t

proof (*induction s arbitrary: t*)

case *Leaf*

show *?case*

proof (*rule measurable-cong[THEN iffD2]*)

fix ω **assume** $\omega \in \text{space } (\text{restrict-space } B\ (?C\ t\ \text{Leaf}))$

then **show** $\text{rec-tree } (l\ \omega) (n\ \omega) (t\ \omega) = l\ \omega$

by (*auto simp: space-restrict-space*)

next

show $l \in \text{restrict-space } B\ (?C\ t\ \text{Leaf}) \rightarrow_M A$

using l **by** (*rule measurable-restrict-space1*)

qed

next

case (*Node ls u rs*)

let $?F = \lambda\omega. ?N\ (\omega, \text{left } (t\ \omega), \text{root-val } (t\ \omega), \text{right } (t\ \omega),$

$\text{rec-tree } (l\ \omega) (n\ \omega) (\text{left } (t\ \omega)), \text{rec-tree } (l\ \omega) (n\ \omega) (\text{right } (t\ \omega)))$

show *?case*

proof (*rule measurable-cong[THEN iffD2]*)

fix ω **assume** $\omega \in \text{space } (\text{restrict-space } B\ (?C\ t\ (\text{Node } ls\ u\ rs)))$

then **show** $\text{rec-tree } (l\ \omega) (n\ \omega) (t\ \omega) = ?F\ \omega$

```

    by (auto simp: space-restrict-space)
  next
  show ?F ∈ (restrict-space B (?C t (Node l s u r s))) →M A
  apply (intro measurable-compose[OF - n] measurable-Pair[rotated])
  subgoal
    apply (rule measurable-restrict-mono[OF Node(2)])
    apply (rule measurable-compose[OF Node(3) measurable-right])
    by auto
  subgoal
    apply (rule measurable-restrict-mono[OF Node(1)])
    apply (rule measurable-compose[OF Node(3) measurable-left])
    by auto
  subgoal
    by (rule measurable-restrict-space1)
      (rule measurable-compose[OF Node(3) measurable-right])
  subgoal
    apply (rule measurable-compose[OF - measurable-root-val])
    apply (rule measurable-restrict-space3[OF Node(3)])
    by auto
  subgoal
    by (rule measurable-restrict-space1)
      (rule measurable-compose[OF Node(3) measurable-left])
  by (rule measurable-restrict-space1) auto
qed
qed
qed

lemma measurable-case-tree [measurable (raw)]:
  assumes t ∈ B →M tree-sigma M
  assumes l ∈ B →M A
  assumes (λ(x, l, v, r). n x l v r)
    ∈ B ⊗M tree-sigma M ⊗M M ⊗M tree-sigma M →M A
  shows (λx. case-tree (l x) (n x) (t x)) ∈ B →M (A :: 'a measure)
proof -
  define n' where n' = (λx l v r (-::'a) (-::'a). n x l v r)
  have (λx. case-tree (l x) (n x) (t x)) = (λx. rec-tree (l x) (n' x) (t x))
    (is - = (λx. rec-tree - (?n' x -))) by (rule ext) (auto split: tree.splits simp:
n'-def)
  also have ... ∈ B →M A
  proof (rule measurable-rec-tree)
    have (λ(x, l, v, r, al, ar). n' x l v r al ar) =
      (λ(x,l,v,r). n x l v r) ∘ (λ(x,l,v,r,al,ar). (x,l,v,r))
    by (simp add: n'-def o-def case-prod-unfold)
    also have ... ∈ B ⊗M tree-sigma M ⊗M M ⊗M tree-sigma M ⊗M A
    ⊗M A →M A
    using assms(3) by measurable
    finally show (λ(x, l, v, r, al, ar). n' x l v r al ar) ∈ ... .
  qed (insert assms, simp-all)
  finally show ?thesis .

```


qed

hide-const (open) left
hide-const (open) right

end

28 Conditional Expectation

theory *Conditional-Expectation*
imports *Probability-Measure*
begin

28.1 Restricting a measure to a sub-sigma-algebra

definition *subalgebra*::'a measure \Rightarrow 'a measure \Rightarrow bool **where**
subalgebra $M F = ((space\ F = space\ M) \wedge (sets\ F \subseteq sets\ M))$

lemma *sub-measure-space*:

assumes *i*: *subalgebra* $M F$

shows *measure-space* (*space* M) (*sets* F) (*emeasure* M)

proof –

have *sigma-algebra* (*space* M) (*sets* F)

by (*metis* *i* *measure-space* *measure-space-def* *subalgebra-def*)

moreover have *positive* (*sets* F) (*emeasure* M)

using *Sigma-Algebra.positive-def* **by** *auto*

moreover have *countably-additive* (*sets* F) (*emeasure* M)

by (*meson* *countably-additive-def* *emeasure-countably-additive* *i* *order-trans* *subalgebra-def* *subsetCE*)

ultimately show *?thesis* **unfolding** *measure-space-def* **by** *simp*

qed

definition *restr-to-subalg*::'a measure \Rightarrow 'a measure \Rightarrow 'a measure **where**
restr-to-subalg $M F = measure-of\ (space\ M)\ (sets\ F)\ (emeasure\ M)$

lemma *space-restr-to-subalg*:

space (*restr-to-subalg* $M F$) = *space* M

unfolding *restr-to-subalg-def* **by** (*simp* *add*: *space-measure-of-conv*)

lemma *sets-restr-to-subalg* [*measurable-cong*]:

assumes *subalgebra* $M F$

shows *sets* (*restr-to-subalg* $M F$) = *sets* F

unfolding *restr-to-subalg-def* **by** (*metis* *sets.sets-measure-of-eq* *assms* *subalgebra-def*)

lemma *emeasure-restr-to-subalg*:

assumes *subalgebra* $M F$

$A \in sets\ F$

shows *emeasure* (*restr-to-subalg* $M F$) $A = emeasure\ M\ A$

unfolding *restr-to-subalg-def*

by (metis assms subalgebra-def emeasure-measure-of-conv sub-measure-space sets.sigma-sets-eq)

lemma *null-sets-restr-to-subalg*:

assumes *subalgebra M F*

shows *null-sets (restr-to-subalg M F) = null-sets M ∩ sets F*

proof

have *x ∈ null-sets M ∩ sets F* **if** *x ∈ null-sets (restr-to-subalg M F)* **for** *x*

by (metis that Int-iff assms emeasure-restr-to-subalg null-setsD1 null-setsD2 null-setsI

sets-restr-to-subalg subalgebra-def subsetD)

then show *null-sets (restr-to-subalg M F) ⊆ null-sets M ∩ sets F* **by** *auto*

next

have *x ∈ null-sets (restr-to-subalg M F)* **if** *x ∈ null-sets M ∩ sets F* **for** *x*

by (metis that Int-iff assms null-setsD1 null-setsI sets-restr-to-subalg emeasure-restr-to-subalg[OF assms])

then show *null-sets M ∩ sets F ⊆ null-sets (restr-to-subalg M F)* **by** *auto*

qed

lemma *AE-restr-to-subalg*:

assumes *subalgebra M F*

AE x in (restr-to-subalg M F). P x

shows *AE x in M. P x*

proof –

obtain *A* **where** *A: ∧x. x ∈ space (restr-to-subalg M F) – A ⇒ P x* *A ∈ null-sets (restr-to-subalg M F)*

using *AE-E3[OF assms(2)]* **by** *auto*

then have *A ∈ null-sets M* **using** *null-sets-restr-to-subalg[OF assms(1)]* **by** *auto*

moreover have *∧x. x ∈ space M – A ⇒ P x*

using *space-restr-to-subalg A(1)* **by** *fastforce*

ultimately show *?thesis*

unfolding *eventually-ae-filter* **by** *auto*

qed

lemma *AE-restr-to-subalg2*:

assumes *subalgebra M F*

AE x in M. P x **and** *[measurable]: P ∈ measurable F (count-space UNIV)*

shows *AE x in (restr-to-subalg M F). P x*

proof –

define *U* **where** *U = {x ∈ space M. ¬(P x)}*

then have ***: *U = {x ∈ space F. ¬(P x)}* **using** *assms(1)* **by** (simp add: *subalgebra-def*)

then have *U ∈ sets F* **by** *simp*

then have *U ∈ sets M* **using** *assms(1)* **by** (meson *subalgebra-def subsetD*)

then have *U ∈ null-sets M* **unfolding** *U-def* **using** *assms(2)* **using** *AE-iff-measurable* **by** *blast*

then have *U ∈ null-sets (restr-to-subalg M F)* **using** *null-sets-restr-to-subalg[OF assms(1)]* *⟨U ∈ sets F⟩* **by** *auto*

then show *?thesis* **using** *** **by** (metis (no-types, lifting) *Collect-mono U-def eventually-ae-filter space-restr-to-subalg*)

qed

lemma *prob-space-restr-to-subalg*:

assumes *subalgebra* $M F$
prob-space M

shows *prob-space* (*restr-to-subalg* $M F$)

by (*metis* (*no-types*, *lifting*) *assms*(1) *assms*(2) *emeasure-restr-to-subalg* *prob-space.emeasure-space-1* *prob-spaceI*
sets.top space-restr-to-subalg subalgebra-def)

lemma *finite-measure-restr-to-subalg*:

assumes *subalgebra* $M F$
finite-measure M

shows *finite-measure* (*restr-to-subalg* $M F$)

by (*metis* (*no-types*, *lifting*) *assms* *emeasure-restr-to-subalg* *finite-measure.finite-emeasure-space* *finite-measureI* *sets.top space-restr-to-subalg subalgebra-def infinity-enreal-def*)

lemma *measurable-in-subalg*:

assumes *subalgebra* $M F$
 $f \in$ *measurable* $F N$

shows $f \in$ *measurable* (*restr-to-subalg* $M F$) N

by (*metis* *measurable-cong-sets* *assms*(2) *sets-restr-to-subalg[OF assms(1)]*)

lemma *measurable-in-subalg'*:

assumes *subalgebra* $M F$
 $f \in$ *measurable* (*restr-to-subalg* $M F$) N

shows $f \in$ *measurable* $F N$

by (*metis* *measurable-cong-sets* *assms*(2) *sets-restr-to-subalg[OF assms(1)]*)

lemma *measurable-from-subalg*:

assumes *subalgebra* $M F$
 $f \in$ *measurable* $F N$

shows $f \in$ *measurable* $M N$

using *assms* **unfolding** *measurable-def subalgebra-def* **by** *auto*

The following is the direct transposition of *nn_integral_subalgebra* (from *Nonnegative_Lebesgue_Integration*) in the current notations, with the removal of the useless assumption $f \geq 0$.

lemma *nn-integral-subalgebra2*:

assumes *subalgebra* $M F$ **and** [*measurable*]: $f \in$ *borel-measurable* F

shows $(\int^+ x. f x \partial(\text{restr-to-subalg } M F)) = (\int^+ x. f x \partial M)$

proof (*rule nn-integral-subalgebra*)

show $f \in$ *borel-measurable* (*restr-to-subalg* $M F$)

by (*rule measurable-in-subalg[OF assms(1)]*) *simp*

show *sets* (*restr-to-subalg* $M F$) \subseteq *sets* M **by** (*metis* *sets-restr-to-subalg[OF assms(1)]* *assms(1)* *subalgebra-def*)

fix A **assume** $A \in$ *sets* (*restr-to-subalg* $M F$)

then show *emeasure* (*restr-to-subalg* $M F$) $A =$ *emeasure* $M A$

by (*metis* *sets-restr-to-subalg[OF assms(1)]* *emeasure-restr-to-subalg[OF assms(1)]*)

qed (*auto simp add: assms space-restr-to-subalg sets-restr-to-subalg[OF assms(1)]*)

The following is the direct transposition of `integral_subalgebra` (from `Bochner_Integration`) in the current notations.

lemma *integral-subalgebra2*:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

assumes *subalgebra M F and*

[measurable]: f \in borel-measurable F

shows $(\int x. f x \partial(\text{restr-to-subalg } M F)) = (\int x. f x \partial M)$

by (*rule integral-subalgebra,*

metis measurable-in-subalg[OF assms(1)] assms(2),

auto simp add: assms space-restr-to-subalg sets-restr-to-subalg emeasure-restr-to-subalg,

meson assms(1) subalgebra-def subset-eq)

lemma *integrable-from-subalg*:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

assumes *subalgebra M F*

integrable (restr-to-subalg M F) f

shows *integrable M f*

proof (*rule integrableI-bounded*)

have *[measurable]: f \in borel-measurable F using assms by auto*

then show $f \in \text{borel-measurable } M$ **using** *assms(1) measurable-from-subalg by*

blast

have $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) = (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial(\text{restr-to-subalg } M F))$

by (*rule nn-integral-subalgebra2[symmetric], auto simp add: assms*)

also have $\dots < \infty$ **using** *integrable-iff-bounded assms by auto*

finally show $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) < \infty$ **by** *simp*

qed

lemma *integrable-in-subalg*:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

assumes *[measurable]: subalgebra M F*

f \in borel-measurable F

integrable M f

shows *integrable (restr-to-subalg M F) f*

proof (*rule integrableI-bounded*)

show $f \in \text{borel-measurable } (\text{restr-to-subalg } M F)$ **using** *assms(2) assms(1) by auto*

have $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial(\text{restr-to-subalg } M F)) = (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M)$

by (*rule nn-integral-subalgebra2, auto simp add: assms*)

also have $\dots < \infty$ **using** *integrable-iff-bounded assms by auto*

finally show $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial(\text{restr-to-subalg } M F)) < \infty$ **by** *simp*

qed

28.2 Nonnegative conditional expectation

The conditional expectation of a function f , on a measure space M , with respect to a sub sigma algebra F , should be a function g which is F -measurable whose integral on any F -set coincides with the integral of f . Such a function is uniquely defined almost everywhere. The most direct construction is to use the measure $f dM$, restrict it to the sigma-algebra F , and apply the Radon-Nikodym theorem to write it as $g dM|_F$ for some F -measurable function g . Another classical construction for L^2 functions is done by orthogonal projection on F -measurable functions, and then extending by density to L^1 . The Radon-Nikodym point of view avoids the L^2 machinery, and works for all positive functions.

In this paragraph, we develop the definition and basic properties for nonnegative functions, as the basics of the general case. As in the definition of integrals, the nonnegative case is done with ennreal-valued functions, without any integrability assumption.

definition *nn-cond-exp* :: 'a measure \Rightarrow 'a measure \Rightarrow ('a \Rightarrow ennreal) \Rightarrow ('a \Rightarrow ennreal)

where

nn-cond-exp $M F f =$
 (if $f \in \text{borel-measurable } M \wedge \text{subalgebra } M F$
 then $\text{RN-deriv } (\text{restr-to-subalg } M F) (\text{restr-to-subalg } (\text{density } M f) F)$
 else $(\lambda-. 0)$)

lemma

shows *borel-measurable-nn-cond-exp* [*measurable*]: *nn-cond-exp* $M F f \in \text{borel-measurable } F$

and *borel-measurable-nn-cond-exp2* [*measurable*]: *nn-cond-exp* $M F f \in \text{borel-measurable } M$

by (*simp-all add: nn-cond-exp-def*)

(*metis borel-measurable-RN-deriv borel-measurable-subalgebra sets-restr-to-subalg space-restr-to-subalg subalgebra-def*)

The good setting for conditional expectations is the situation where the subalgebra F gives rise to a sigma-finite measure space. To see what goes wrong if it is not sigma-finite, think of \mathbb{R} with the trivial sigma-algebra $\{\emptyset, \mathbb{R}\}$. In this case, conditional expectations have to be constant functions, so they have integral 0 or ∞ . This means that a positive integrable function can have no meaningful conditional expectation.

locale *sigma-finite-subalgebra* =

fixes $M F :: 'a \text{ measure}$

assumes *subalg*: *subalgebra* $M F$

and *sigma-fin-subalg*: *sigma-finite-measure* (*restr-to-subalg* $M F$)

lemma *sigma-finite-subalgebra-is-sigma-finite*:

assumes *sigma-finite-subalgebra* $M F$

shows *sigma-finite-measure* M
proof
have *subalg*: *subalgebra* $M F$
and *sigma-fin-subalg*: *sigma-finite-measure* (*restr-to-subalg* $M F$)
using *assms* **unfolding** *sigma-finite-subalgebra-def* **by** *auto*
obtain A **where** *Ap*: *countable* $A \wedge A \subseteq \text{sets}$ (*restr-to-subalg* $M F$) $\wedge \bigcup A =$
space (*restr-to-subalg* $M F$) $\wedge (\forall a \in A. \text{emeasure}$ (*restr-to-subalg* $M F$) $a \neq \infty)$
using *sigma-finite-measure.sigma-finite-countable*[*OF sigma-fin-subalg*] **by** *fastforce*
have $A \subseteq \text{sets}$ F **using** *Ap sets-restr-to-subalg*[*OF subalg*] **by** *fastforce*
then have $A \subseteq \text{sets}$ M **using** *subalg subalgebra-def* **by** *force*
moreover have $\bigcup A = \text{space}$ M **using** *Ap space-restr-to-subalg* **by** *simp*
moreover have $\forall a \in A. \text{emeasure}$ $M a \neq \infty$ **by** (*metis subsetD emeasure-restr-to-subalg*[*OF*
subalg] $\langle A \subseteq \text{sets}$ $F \rangle$ *Ap*)
ultimately show $\exists A. \text{countable}$ $A \wedge A \subseteq \text{sets}$ $M \wedge \bigcup A = \text{space}$ $M \wedge (\forall a \in A.$
 emeasure $M a \neq \infty)$ **using** *Ap* **by** *auto*
qed

sublocale *sigma-finite-subalgebra* \subseteq *sigma-finite-measure*
using *sigma-finite-subalgebra-is-sigma-finite* *sigma-finite-subalgebra-axioms* **by** *blast*

Conditional expectations are very often used in probability spaces. This is a special case of the previous one, as we prove now.

locale *finite-measure-subalgebra* = *finite-measure* +
fixes $F::'a \text{ measure}$
assumes *subalg*: *subalgebra* $M F$

lemma *finite-measure-subalgebra-is-sigma-finite*:
assumes *finite-measure-subalgebra* $M F$
shows *sigma-finite-subalgebra* $M F$
proof –
interpret *finite-measure-subalgebra* $M F$ **using** *assms* **by** *simp*
have *finite-measure* (*restr-to-subalg* $M F$)
using *finite-measure-restr-to-subalg* *subalg* *finite-emeasure-space* *finite-measureI*
unfolding *infinity-ennreal-def* **by** *blast*
then have *sigma-finite-measure* (*restr-to-subalg* $M F$)
unfolding *finite-measure-def* **by** *simp*
then show *sigma-finite-subalgebra* $M F$ **unfolding** *sigma-finite-subalgebra-def*
using *subalg* **by** *simp*
qed

sublocale *finite-measure-subalgebra* \subseteq *sigma-finite-subalgebra*
proof –
have *finite-measure* (*restr-to-subalg* $M F$)
using *finite-measure-restr-to-subalg* *subalg* *finite-emeasure-space* *finite-measureI*
unfolding *infinity-ennreal-def* **by** *blast*
then have *sigma-finite-measure* (*restr-to-subalg* $M F$)
unfolding *finite-measure-def* **by** *simp*
then show *sigma-finite-subalgebra* $M F$ **unfolding** *sigma-finite-subalgebra-def*
using *subalg* **by** *simp*

qed

context *sigma-finite-subalgebra*
begin

The next lemma is arguably the most fundamental property of conditional expectation: when computing an expectation against an F -measurable function, it is equivalent to work with a function or with its F -conditional expectation.

This property (even for bounded test functions) characterizes conditional expectations, as the second lemma below shows. From this point on, we will only work with it, and forget completely about the definition using Radon-Nikodym derivatives.

lemma *nn-cond-exp-intg*:

assumes [*measurable*]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$
shows $(\int^+ x. f x * \text{nn-cond-exp } M F g x \partial M) = (\int^+ x. f x * g x \partial M)$

proof –

have [*measurable*]: $f \in \text{borel-measurable } M$

by (*meson assms subalg borel-measurable-subalgebra subalgebra-def*)

have *ac*: *absolutely-continuous* (*restr-to-subalg* $M F$) (*restr-to-subalg* (*density* $M g$) F)

unfolding *absolutely-continuous-def*

proof –

have *null-sets* (*restr-to-subalg* $M F$) = *null-sets* $M \cap \text{sets } F$ **by** (*rule null-sets-restr-to-subalg*[*OF subalg*])

moreover have *null-sets* $M \subseteq \text{null-sets}$ (*density* $M g$)

by (*rule absolutely-continuousI-density*[*unfolded absolutely-continuous-def*])

auto

ultimately have *null-sets* (*restr-to-subalg* $M F$) $\subseteq \text{null-sets}$ (*density* $M g$) $\cap \text{sets } F$ **by** *auto*

moreover have *null-sets* (*density* $M g$) $\cap \text{sets } F = \text{null-sets}$ (*restr-to-subalg* (*density* $M g$) F)

by (*rule null-sets-restr-to-subalg*[*symmetric*]) (*metis subalg sets-density space-density subalgebra-def*)

ultimately show *null-sets* (*restr-to-subalg* $M F$) $\subseteq \text{null-sets}$ (*restr-to-subalg* (*density* $M g$) F) **by** *auto*

qed

have $(\int^+ x. f x * \text{nn-cond-exp } M F g x \partial M) = (\int^+ x. f x * \text{nn-cond-exp } M F g x \partial(\text{restr-to-subalg } M F))$

by (*rule nn-integral-subalgebra2*[*symmetric*]) (*simp-all add: assms subalg*)

also have ... = $(\int^+ x. f x * \text{RN-deriv } (\text{restr-to-subalg } M F) (\text{restr-to-subalg } (\text{density } M g) F) x \partial(\text{restr-to-subalg } M F))$

unfolding *nn-cond-exp-def* **using** *assms subalg* **by** *simp*

also have ... = $(\int^+ x. \text{RN-deriv } (\text{restr-to-subalg } M F) (\text{restr-to-subalg } (\text{density } M g) F) x * f x \partial(\text{restr-to-subalg } M F))$

by (*simp add: mult commute*)

also have ... = $(\int^+ x. f x \partial(\text{restr-to-subalg } (\text{density } M g) F))$

proof (rule *sigma-finite-measure.RN-deriv-nn-integral[symmetric]*)
show $\text{sets (restr-to-subalg (density M g) F) = sets (restr-to-subalg M F)}$
by (*metis subalg restr-to-subalg-def sets.sets-measure-of-eq space-density subalgebra-def*)
qed (*auto simp add: assms measurable-restrict ac measurable-in-subalg subalg sigma-fin-subalg*)
also have $\dots = (\int^+ x. f x \partial(\text{density M g}))$
by (*metis nn-integral-subalgebra2 subalg assms(1) sets-density space-density subalgebra-def*)
also have $\dots = (\int^+ x. g x * f x \partial M)$
by (*rule nn-integral-density*) (*simp-all add: assms*)
also have $\dots = (\int^+ x. f x * g x \partial M)$
by (*simp add: mult commute*)
finally show *?thesis* **by** *simp*
qed

lemma *nn-cond-exp-charact*:

assumes $\bigwedge A. A \in \text{sets } F \implies (\int^+ x \in A. f x \partial M) = (\int^+ x \in A. g x \partial M)$ **and**
 $[measurable]; f \in \text{borel-measurable } M g \in \text{borel-measurable } F$
shows $AE x \text{ in } M. g x = \text{nn-cond-exp } M F f x$
proof –
let $?MF = \text{restr-to-subalg } M F$
{
fix A **assume** $A \in \text{sets } ?MF$
then have $[measurable]: A \in \text{sets } F$ **using** *sets-restr-to-subalg[OF subalg]* **by**
simp
have $(\int^+ x \in A. g x \partial ?MF) = (\int^+ x \in A. g x \partial M)$
by (*simp add: nn-integral-subalgebra2 subalg*)
also have $\dots = (\int^+ x \in A. f x \partial M)$ **using** *assms(1)* **by** *simp*
also have $\dots = (\int^+ x. \text{indicator } A x * f x \partial M)$ **by** (*simp add: mult commute*)
also have $\dots = (\int^+ x. \text{indicator } A x * \text{nn-cond-exp } M F f x \partial M)$
by (*rule nn-cond-exp-intg[symmetric]*) (*auto simp add: assms*)
also have $\dots = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial M)$ **by** (*simp add: mult commute*)
also have $\dots = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial ?MF)$
by (*simp add: nn-integral-subalgebra2 subalg*)
finally have $(\int^+ x \in A. g x \partial ?MF) = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial ?MF)$ **by** *simp*
} note $*$ = *this*
have $AE x \text{ in } ?MF. g x = \text{nn-cond-exp } M F f x$
by (*rule sigma-finite-measure.density-unique2*)
*(auto simp add: assms subalg sigma-fin-subalg AE-restr-to-subalg2 *)*
then show *?thesis* **using** *AE-restr-to-subalg[OF subalg]* **by** *simp*
qed

lemma *nn-cond-exp-F-meas*:

assumes $f \in \text{borel-measurable } F$
shows $AE x \text{ in } M. f x = \text{nn-cond-exp } M F f x$
by (*rule nn-cond-exp-charact*) (*auto simp add: assms measurable-from-subalg[OF subalg]*)

lemma *nn-cond-exp-prod*:

assumes [*measurable*]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$
shows $\text{AE } x \text{ in } M. f x * \text{nn-cond-exp } M F g x = \text{nn-cond-exp } M F (\lambda x. f x * g x) x$

proof (*rule nn-cond-exp-charact*)

have [*measurable*]: $f \in \text{borel-measurable } M$ **by** (*rule measurable-from-subalg[OF subalg assms(1)]*)

show $(\lambda x. f x * g x) \in \text{borel-measurable } M$ **by** *measurable*

fix A **assume** $A \in \text{sets } F$

then have [*measurable*]: $(\lambda x. f x * \text{indicator } A x) \in \text{borel-measurable } F$ **by** *measurable*

have $\int^{+x \in A}. (f x * g x) \partial M = \int^{+x}. (f x * \text{indicator } A x) * g x \partial M$

by (*simp add: mult.commute mult.left-commute*)

also have $\dots = \int^{+x}. (f x * \text{indicator } A x) * \text{nn-cond-exp } M F g x \partial M$

by (*rule nn-cond-exp-intg[symmetric]*) (*auto simp add: assms*)

also have $\dots = \int^{+x \in A}. (f x * \text{nn-cond-exp } M F g x) \partial M$

by (*simp add: mult.commute mult.left-commute*)

finally show $\int^{+x \in A}. (f x * g x) \partial M = \int^{+x \in A}. (f x * \text{nn-cond-exp } M F g x) \partial M$ **by** *simp*

qed (*auto simp add: assms*)

lemma *nn-cond-exp-sum*:

assumes [*measurable*]: $f \in \text{borel-measurable } M$ $g \in \text{borel-measurable } M$

shows $\text{AE } x \text{ in } M. \text{nn-cond-exp } M F f x + \text{nn-cond-exp } M F g x = \text{nn-cond-exp } M F (\lambda x. f x + g x) x$

proof (*rule nn-cond-exp-charact*)

fix A **assume** [*measurable*]: $A \in \text{sets } F$

then have $A \in \text{sets } M$ **by** (*meson subalg subalgebra-def subsetD*)

have $\int^{+x \in A}. (\text{nn-cond-exp } M F f x + \text{nn-cond-exp } M F g x) \partial M = (\int^{+x \in A}. \text{nn-cond-exp } M F f x \partial M) + (\int^{+x \in A}. \text{nn-cond-exp } M F g x \partial M)$

by (*rule nn-set-integral-add*) (*auto simp add: assms ⟨A ∈ sets M⟩*)

also have $\dots = (\int^{+x}. \text{indicator } A x * \text{nn-cond-exp } M F f x \partial M) + (\int^{+x}. \text{indicator } A x * \text{nn-cond-exp } M F g x \partial M)$

by (*metis (no-types, lifting) mult.commute nn-integral-cong*)

also have $\dots = (\int^{+x}. \text{indicator } A x * f x \partial M) + (\int^{+x}. \text{indicator } A x * g x \partial M)$

by (*simp add: nn-cond-exp-intg*)

also have $\dots = (\int^{+x \in A}. f x \partial M) + (\int^{+x \in A}. g x \partial M)$

by (*metis (no-types, lifting) mult.commute nn-integral-cong*)

also have $\dots = \int^{+x \in A}. (f x + g x) \partial M$

by (*rule nn-set-integral-add[symmetric]*) (*auto simp add: assms ⟨A ∈ sets M⟩*)

finally show $\int^{+x \in A}. (f x + g x) \partial M = \int^{+x \in A}. (\text{nn-cond-exp } M F f x + \text{nn-cond-exp } M F g x) \partial M$

by *simp*

qed (*auto simp add: assms*)

lemma *nn-cond-exp-cong*:

assumes $AE\ x\ in\ M. f\ x = g\ x$
and $[measurable]: f \in \text{borel-measurable } M\ g \in \text{borel-measurable } M$
shows $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ f\ x = nn\text{-cond-exp } M\ F\ g\ x$
proof (rule $nn\text{-cond-exp-charact}$)
fix A **assume** $[measurable]: A \in \text{sets } F$
have $\int^+ x \in A. nn\text{-cond-exp } M\ F\ f\ x\ \partial M = \int^+ x. \text{indicator } A\ x * nn\text{-cond-exp } M\ F\ f\ x\ \partial M$
by (simp add: mult commute)
also have $\dots = \int^+ x. \text{indicator } A\ x * f\ x\ \partial M$ **by** (simp add: nn-cond-exp-intg assms)
also have $\dots = \int^+ x \in A. f\ x\ \partial M$ **by** (simp add: mult commute)
also have $\dots = \int^+ x \in A. g\ x\ \partial M$ **by** (rule nn-set-integral-cong[OF assms(1)])
finally show $\int^+ x \in A. g\ x\ \partial M = \int^+ x \in A. nn\text{-cond-exp } M\ F\ f\ x\ \partial M$ **by** simp
qed (auto simp add: assms)

lemma $nn\text{-cond-exp-mono}$:

assumes $AE\ x\ in\ M. f\ x \leq g\ x$
and $[measurable]: f \in \text{borel-measurable } M\ g \in \text{borel-measurable } M$
shows $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ f\ x \leq nn\text{-cond-exp } M\ F\ g\ x$
proof –
define h **where** $h = (\lambda x. g\ x - f\ x)$
have $[measurable]: h \in \text{borel-measurable } M$ **unfolding** $h\text{-def}$ **by** simp
have $*$: $AE\ x\ in\ M. g\ x = f\ x + h\ x$ **unfolding** $h\text{-def}$ **using** $assms(1)$ **by** (auto simp: ennreal-ineq-diff-add)
have $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ g\ x = nn\text{-cond-exp } M\ F\ (\lambda x. f\ x + h\ x)\ x$
by (rule nn-cond-exp-cong) (auto simp add: * assms)
moreover have $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ f\ x + nn\text{-cond-exp } M\ F\ h\ x = nn\text{-cond-exp } M\ F\ (\lambda x. f\ x + h\ x)\ x$
by (rule nn-cond-exp-sum) (auto simp add: assms)
ultimately have $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ g\ x = nn\text{-cond-exp } M\ F\ f\ x + nn\text{-cond-exp } M\ F\ h\ x$ **by** auto
then show $?thesis$ **by** force
qed

lemma $nested\text{-subalg-is-sigma-finite}$:

assumes $subalgebra\ M\ G\ subalgebra\ G\ F$
shows $sigma\text{-finite-subalgebra } M\ G$
unfolding $sigma\text{-finite-subalgebra-def}$
proof (auto simp add: assms)
have $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (restr\text{-to-subalg } M\ F) \wedge \bigcup A = \text{space } (restr\text{-to-subalg } M\ F) \wedge (\forall a \in A. \text{emeasure } (restr\text{-to-subalg } M\ F)\ a \neq \infty)$
using $sigma\text{-fin-subalg } sigma\text{-finite-measure-def}$ **by** auto
then obtain A **where** $A: \text{countable } A \wedge A \subseteq \text{sets } (restr\text{-to-subalg } M\ F) \wedge \bigcup A = \text{space } (restr\text{-to-subalg } M\ F) \wedge (\forall a \in A. \text{emeasure } (restr\text{-to-subalg } M\ F)\ a \neq \infty)$
by auto
have $\text{sets } F \subseteq \text{sets } M$
by (meson assms order-trans subalgebra-def)
then have $\text{countable } A \wedge A \subseteq \text{sets } (restr\text{-to-subalg } M\ G) \wedge \bigcup A = \text{space } (restr\text{-to-subalg } M\ G) \wedge (\forall a \in A. \text{emeasure } (restr\text{-to-subalg } M\ G)\ a \neq \infty)$

by (metis (no-types) A assms basic-trans-rules(31) emeasure-restr-to-subalg order-trans sets-restr-to-subalg subalgebra-def)
 then show sigma-finite-measure (restr-to-subalg M G)
 by (metis sigma-finite-measure.intro space-restr-to-subalg)
 qed

lemma nn-cond-exp-nested-subalg:

assumes subalgebra M G subalgebra G F
 and [measurable]: f ∈ borel-measurable M
 shows AE x in M. nn-cond-exp M F f x = nn-cond-exp M F (nn-cond-exp M G f) x
 proof (rule nn-cond-exp-charact, auto)
 interpret G: sigma-finite-subalgebra M G by (rule nested-subalg-is-sigma-finite[OF assms(1) assms(2)])
 fix A assume [measurable]: A ∈ sets F
 then have [measurable]: A ∈ sets G using assms(2) by (meson set-mp subalgebra-def)

 have set-nn-integral M A (nn-cond-exp M G f) = (∫⁺ x. indicator A x * nn-cond-exp M G f x ∂M)
 by (metis (no-types) mult.commute)
 also have ... = (∫⁺ x. indicator A x * f x ∂M) by (rule G.nn-cond-exp-intg, auto simp add: assms)
 also have ... = (∫⁺ x. indicator A x * nn-cond-exp M F f x ∂M) by (rule nn-cond-exp-intg[symmetric], auto simp add: assms)
 also have ... = set-nn-integral M A (nn-cond-exp M F f) by (metis (no-types) mult.commute)
 finally show set-nn-integral M A (nn-cond-exp M G f) = set-nn-integral M A (nn-cond-exp M F f).
 qed

end

28.3 Real conditional expectation

Once conditional expectations of positive functions are defined, the definition for real-valued functions follows readily, by taking the difference of positive and negative parts. One could also define a conditional expectation of vector-space valued functions, as in `Bochner_Integral`, but since the real-valued case is the most important, and quicker to formalize, I concentrate on it. (It is also essential for the case of the most general Pettis integral.)

definition *real-cond-exp* :: 'a measure ⇒ 'a measure ⇒ ('a ⇒ real) ⇒ ('a ⇒ real) where

real-cond-exp M F f =
 (λx. enn2real(nn-cond-exp M F (λx. ennreal (f x)) x) - enn2real(nn-cond-exp M F (λx. ennreal (-f x)) x))

lemma

shows *borel-measurable-cond-exp* [*measurable*]: *real-cond-exp* $M F f \in$ *borel-measurable* F
and *borel-measurable-cond-exp2* [*measurable*]: *real-cond-exp* $M F f \in$ *borel-measurable* M
unfolding *real-cond-exp-def* **by** *auto*

context *sigma-finite-subalgebra*
begin

lemma *real-cond-exp-abs*:

assumes [*measurable*]: $f \in$ *borel-measurable* M
shows *AE* x *in* M . $\text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F (\lambda x. \text{ennreal}(\text{abs}(f x))) x$

proof –

define *fp* **where** $fp = (\lambda x. \text{ennreal } (f x))$
define *fm* **where** $fm = (\lambda x. \text{ennreal } (- f x))$
have [*measurable*]: $fp \in$ *borel-measurable* M $fm \in$ *borel-measurable* M **unfolding** *fp-def fm-def* **by** *auto*
have *eq*: $\bigwedge x. \text{ennreal } |f x| = fp x + fm x$ **unfolding** *fp-def fm-def* **by** (*simp add: abs-real-def ennreal-neg*)

{
fix x **assume** H : *nn-cond-exp* $M F fp x + \text{nn-cond-exp } M F fm x = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$
have $|\text{real-cond-exp } M F f x| \leq |\text{enn2real}(\text{nn-cond-exp } M F fp x)| + |\text{enn2real}(\text{nn-cond-exp } M F fm x)|$
unfolding *real-cond-exp-def fp-def fm-def* **by** (*auto intro: abs-triangle-ineq4 simp del: enn2real-nonneg*)
from *ennreal-leI*[*OF this*]
have $\text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F fp x + \text{nn-cond-exp } M F fm x$
by *simp (metis add commute ennreal-enn2real le-iff-add not-le top-unique)*
also have $\dots = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$ **using** H **by** *simp*
finally have $\text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$ **by** *simp*
}
moreover have *AE* x *in* M . $\text{nn-cond-exp } M F fp x + \text{nn-cond-exp } M F fm x = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$
by (*rule nn-cond-exp-sum*) (*auto simp add: fp-def fm-def*)
ultimately have *AE* x *in* M . $\text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$
by *auto*
then show *?thesis* **using** *eq* **by** *simp*
qed

The next lemma shows that the conditional expectation is an F -measurable function whose average against an F -measurable function f coincides with the average of the original function against f . It is obtained as a consequence of the same property for the positive conditional expectation, taking the

difference of the positive and the negative part. The proof is given first assuming $f \geq 0$ for simplicity, and then extended to the general case in the subsequent lemma. The idea of the proof is essentially trivial, but the implementation is slightly tedious as one should check all the integrability properties of the different parts, and go back and forth between positive integral and signed integrals, and between real-valued functions and ennreal-valued functions.

Once this lemma is available, we will use it to characterize the conditional expectation, and never come back to the original technical definition, as we did in the case of the nonnegative conditional expectation.

lemma *real-cond-exp-intg-fpos*:

assumes *integrable* M $(\lambda x. f x * g x)$ **and** *f-pos[simp]*: $\bigwedge x. f x \geq 0$ **and**
[measurable]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$
shows *integrable* M $(\lambda x. f x * \text{real-cond-exp } M F g x)$
 $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$

proof –

have *[measurable]*: $f \in \text{borel-measurable } M$ **by** (*rule measurable-from-subalg[OF subalg assms(3)]*)

define *gp* **where** $gp = (\lambda x. \text{ennreal } (g x))$

define *gm* **where** $gm = (\lambda x. \text{ennreal } (-g x))$

have *[measurable]*: $gp \in \text{borel-measurable } M$ $gm \in \text{borel-measurable } M$ **unfolding**
gp-def gm-def **by** *auto*

define *h* **where** $h = (\lambda x. \text{ennreal}(\text{abs}(g x)))$

have *hgpgm*: $\bigwedge x. h x = gp x + gm x$ **unfolding** *gp-def gm-def h-def* **by** (*simp add: abs-real-def ennreal-neg*)

have *[measurable]*: $h \in \text{borel-measurable } M$ **unfolding** *h-def* **by** *simp*

have *pos[simp]*: $\bigwedge x. h x \geq 0 \bigwedge x. gp x \geq 0 \bigwedge x. gm x \geq 0$ **unfolding** *h-def*
gp-def gm-def **by** *simp-all*

have *gp-real*: $\bigwedge x. \text{enn2real}(gp x) = \max (g x) 0$

unfolding *gp-def* **by** (*simp add: max-def ennreal-neg*)

have *gm-real*: $\bigwedge x. \text{enn2real}(gm x) = \max (-g x) 0$

unfolding *gm-def* **by** (*simp add: max-def ennreal-neg*)

have $(\int^+ x. \text{norm}(f x * \max (g x) 0) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (*simp add: nn-integral-mono*)

also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)

finally have $(\int^+ x. \text{norm}(f x * \max (g x) 0) \partial M) < \infty$ **by** *simp*

then have *int1*: *integrable* M $(\lambda x. f x * \max (g x) 0)$ **by** (*simp add: integrableI-bounded*)

have $(\int^+ x. \text{norm}(f x * \max (-g x) 0) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (*simp add: nn-integral-mono*)

also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)

finally have $(\int^+ x. \text{norm}(f x * \max (-g x) 0) \partial M) < \infty$ **by** *simp*

then have *int2*: *integrable* M $(\lambda x. f x * \max (-g x) 0)$ **by** (*simp add: integrableI-bounded*)

have $(\int^+ x. f x * \text{nn-cond-exp } M F h x \partial M) = (\int^+ x. f x * h x \partial M)$

by (*rule nn-cond-exp-intg*) *auto*

also have $\dots = \int^+ x. \text{ennreal } (f x * \max (g x) 0 + f x * \max (-g x) 0) \partial M$

unfolding *h-def*
by (*intro nn-integral-cong*)(*auto simp: ennreal-mult[symmetric] abs-mult split: split-max*)
also have $\dots < \infty$
using *Bochner-Integration.integrable-add[OF int1 int2, THEN integrableD(2)]*
by (*auto simp add: less-top*)
finally have $\ast: (\int^+ x. f x \ast nn\text{-cond-exp } M F h x \partial M) < \infty$ **by** *simp*

have $(\int^+ x. norm(f x \ast real\text{-cond-exp } M F g x) \partial M) = (\int^+ x. f x \ast abs(real\text{-cond-exp } M F g x) \partial M)$
by (*simp add: abs-mult*)
also have $\dots \leq (\int^+ x. f x \ast nn\text{-cond-exp } M F h x \partial M)$
proof (*rule nn-integral-mono-AE*)
 $\{$
fix *x* **assume** $\ast: abs(real\text{-cond-exp } M F g x) \leq nn\text{-cond-exp } M F h x$
have $ennreal(f x \ast |real\text{-cond-exp } M F g x|) = f x \ast ennreal(|real\text{-cond-exp } M F g x|)$
by (*simp add: ennreal-mult*)
also have $\dots \leq f x \ast nn\text{-cond-exp } M F h x$
using \ast **by** (*auto intro!: mult-left-mono*)
finally have $ennreal(f x \ast |real\text{-cond-exp } M F g x|) \leq f x \ast nn\text{-cond-exp } M F h x$
by *simp*
 $\}$
then show *AE x in M. ennreal(f x \ast |real-cond-exp M F g x|) ≤ f x \ast nn-cond-exp M F h x*
using *real-cond-exp-abs[OF assms(4)] h-def* **by** *auto*
qed
finally have $\ast\ast: (\int^+ x. norm(f x \ast real\text{-cond-exp } M F g x) \partial M) < \infty$ **using** \ast **by** *auto*
show *integrable M (λx. f x \ast real-cond-exp M F g x)*
using $\ast\ast$ **by** (*intro integrableI-bounded*) *auto*

have $(\int^+ x. f x \ast nn\text{-cond-exp } M F gp x \partial M) \leq (\int^+ x. f x \ast nn\text{-cond-exp } M F h x \partial M)$
proof (*rule nn-integral-mono-AE*)
have *AE x in M. nn-cond-exp M F gp x ≤ nn-cond-exp M F h x*
by (*rule nn-cond-exp-mono*) (*auto simp add: hgpgm*)
then show *AE x in M. f x \ast nn-cond-exp M F gp x ≤ f x \ast nn-cond-exp M F h x*
by (*auto simp: mult-left-mono*)
qed
then have *a: (∫⁺x. f x \ast nn-cond-exp M F gp x ∂M) < ∞*
using \ast **by** *auto*
have $ennreal(norm(f x \ast enn2real(nn\text{-cond-exp } M F gp x))) \leq f x \ast nn\text{-cond-exp } M F gp x$ **for** *x*
by (*auto simp add: ennreal-mult intro!: mult-left-mono*)
(metis enn2real-ennreal enn2real-nonneg le-cases le-ennreal-iff)

then have $(\int^+ x. \text{norm}(f x * \text{enn2real}(\text{nn-cond-exp } M F gp x)) \partial M) \leq (\int^+ x. f x * \text{nn-cond-exp } M F gp x \partial M)$
by (*simp add: nn-integral-mono*)
then have $(\int^+ x. \text{norm}(f x * \text{enn2real}(\text{nn-cond-exp } M F gp x)) \partial M) < \infty$ **using** *a* **by** *auto*
then have *gp-int: integrable* $M (\lambda x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x))$ **by** (*simp add: integrableI-bounded*)
have *gp-fin: AE* x *in* $M. f x * \text{nn-cond-exp } M F gp x \neq \infty$
apply (*rule nn-integral-PInf-AE*) **using** *a* **by** *auto*

have $(\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) \partial M) = \text{enn2real} (\int^+ x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) \partial M)$
by (*rule integral-eq-nn-integral*) *auto*
also have $\dots = \text{enn2real}(\int^+ x. \text{ennreal}(f x * \text{enn2real}(gp x)) \partial M)$
proof –

- {
- fix** x **assume** $f x * \text{nn-cond-exp } M F gp x \neq \infty$
- then have** $\text{ennreal} (f x * \text{enn2real} (\text{nn-cond-exp } M F gp x)) = \text{ennreal} (f x) * \text{nn-cond-exp } M F gp x$
- by** (*auto simp add: ennreal-mult ennreal-mult-eq-top-iff less-top intro!: ennreal-mult-left-cong*)
- }
- then have** *AE* x *in* $M. \text{ennreal} (f x * \text{enn2real} (\text{nn-cond-exp } M F gp x)) = \text{ennreal} (f x) * \text{nn-cond-exp } M F gp x$
- using** *gp-fin* **by** *auto*
- then have** $(\int^+ x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) \partial M) = (\int^+ x. f x * \text{nn-cond-exp } M F gp x \partial M)$
- by** (*rule nn-integral-cong-AE*)
- also have** $\dots = (\int^+ x. f x * gp x \partial M)$
- by** (*rule nn-cond-exp-intg*) (*auto simp add: gp-def*)
- also have** $\dots = (\int^+ x. \text{ennreal}(f x * \text{enn2real}(gp x)) \partial M)$
- by** (*rule nn-integral-cong-AE*) (*auto simp: ennreal-mult gp-def*)
- finally have** $(\int^+ x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) \partial M) = (\int^+ x. \text{ennreal}(f x * \text{enn2real}(gp x)) \partial M)$ **by** *simp*
- then show** *?thesis* **by** *simp*

qed

also have $\dots = (\int x. f x * \text{enn2real}(gp x) \partial M)$
by (*rule integral-eq-nn-integral[symmetric]*) (*auto simp add: gp-def*)
finally have *gp-expr:* $(\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) \partial M) = (\int x. f x * \text{enn2real}(gp x) \partial M)$ **by** *simp*

have $(\int^+ x. f x * \text{nn-cond-exp } M F gm x \partial M) \leq (\int^+ x. f x * \text{nn-cond-exp } M F h x \partial M)$
proof (*rule nn-integral-mono-AE*)
have *AE* x *in* $M. \text{nn-cond-exp } M F gm x \leq \text{nn-cond-exp } M F h x$
by (*rule nn-cond-exp-mono*) (*auto simp add: hgpgm*)
then show *AE* x *in* $M. f x * \text{nn-cond-exp } M F gm x \leq f x * \text{nn-cond-exp } M F h x$

by (auto simp: mult-left-mono)
 qed
 then have $a: (\int^+ x. f x * nn\text{-cond-exp } M F gm x \partial M) < \infty$
 using * by auto
 have $\bigwedge x. ennreal(norm(f x * enn2real(nn\text{-cond-exp } M F gm x))) \leq f x * nn\text{-cond-exp } M F gm x$
 by (auto simp add: ennreal-mult intro!: mult-left-mono)
 (metis enn2real-ennreal enn2real-nonneg le-cases le-ennreal-iff)
 then have $(\int^+ x. norm(f x * enn2real(nn\text{-cond-exp } M F gm x)) \partial M) \leq (\int^+ x. f x * nn\text{-cond-exp } M F gm x \partial M)$
 by (simp add: nn-integral-mono)
 then have $(\int^+ x. norm(f x * enn2real(nn\text{-cond-exp } M F gm x)) \partial M) < \infty$ using a by auto
 then have $gm\text{-int}: integrable\ M (\lambda x. f x * enn2real(nn\text{-cond-exp } M F gm x))$
 by (simp add: integrableI-bounded)
 have $gm\text{-fin}: AE\ x\ in\ M. f x * nn\text{-cond-exp } M F gm x \neq \infty$
 apply (rule nn-integral-PInf-AE) using a by auto

 have $(\int x. f x * enn2real(nn\text{-cond-exp } M F gm x) \partial M) = enn2real(\int^+ x. f x * enn2real(nn\text{-cond-exp } M F gm x) \partial M)$
 by (rule integral-eq-nn-integral) auto
 also have $\dots = enn2real(\int^+ x. ennreal(f x * enn2real(gm x)) \partial M)$
 proof -
 {
 fix x assume $f x * nn\text{-cond-exp } M F gm x \neq \infty$
 then have $ennreal(f x * enn2real(nn\text{-cond-exp } M F gm x)) = ennreal(f x * nn\text{-cond-exp } M F gm x)$
 by (auto simp add: ennreal-mult ennreal-mult-eq-top-iff less-top intro!: ennreal-mult-left-cong)
 }
 then have $AE\ x\ in\ M. ennreal(f x * enn2real(nn\text{-cond-exp } M F gm x)) = ennreal(f x * nn\text{-cond-exp } M F gm x)$
 using $gm\text{-fin}$ by auto
 then have $(\int^+ x. f x * enn2real(nn\text{-cond-exp } M F gm x) \partial M) = (\int^+ x. f x * nn\text{-cond-exp } M F gm x \partial M)$
 by (rule nn-integral-cong-AE)
 also have $\dots = (\int^+ x. f x * gm x \partial M)$
 by (rule nn-cond-exp-intg) (auto simp add: gm-def)
 also have $\dots = (\int^+ x. ennreal(f x * enn2real(gm x)) \partial M)$
 by (rule nn-integral-cong-AE) (auto simp: ennreal-mult gm-def)
 finally have $(\int^+ x. f x * enn2real(nn\text{-cond-exp } M F gm x) \partial M) = (\int^+ x. ennreal(f x * enn2real(gm x)) \partial M)$ by simp
 then show ?thesis by simp
 qed
 also have $\dots = (\int x. f x * enn2real(gm x) \partial M)$
 by (rule integral-eq-nn-integral[symmetric]) (auto simp add: gm-def)
 finally have $gm\text{-expr}: (\int x. f x * enn2real(nn\text{-cond-exp } M F gm x) \partial M) = (\int x. f x * enn2real(gm x) \partial M)$ by simp

have $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) - f x * \text{enn2real}(\text{nn-cond-exp } M F gm x) \partial M)$
unfolding *real-cond-exp-def gp-def gm-def* **by** (*simp add: right-diff-distrib*)
also have $\dots = (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F gp x) \partial M) - (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F gm x) \partial M)$
by (*rule Bochner-Integration.integral-diff*) (*simp-all add: gp-int gm-int*)
also have $\dots = (\int x. f x * \text{enn2real}(gp x) \partial M) - (\int x. f x * \text{enn2real}(gm x) \partial M)$
using *gp-expr gm-expr* **by** *simp*
also have $\dots = (\int x. f x * \text{max } (g x) 0 \partial M) - (\int x. f x * \text{max } (-g x) 0 \partial M)$
using *gp-real gm-real* **by** *simp*
also have $\dots = (\int x. f x * \text{max } (g x) 0 - f x * \text{max } (-g x) 0 \partial M)$
by (*rule Bochner-Integration.integral-diff[symmetric]*) (*simp-all add: int1 int2*)
also have $\dots = (\int x. f x * g x \partial M)$
by (*metis (mono-tags, hide-lams) diff-0 diff-zero eq-iff max.cobounded2 max-def minus-minus neg-le-0-iff-le right-diff-distrib*)
finally show $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$
by *simp*
qed

lemma *real-cond-exp-intg*:

assumes *integrable M* $(\lambda x. f x * g x)$ **and**

[measurable]: f \in *borel-measurable F* $g \in$ *borel-measurable M*

shows *integrable M* $(\lambda x. f x * \text{real-cond-exp } M F g x)$

$(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$

proof –

have *[measurable]: f* \in *borel-measurable M* **by** (*rule measurable-from-subalg[OF subalg assms(2)]*)

define *fp* **where** $fp = (\lambda x. \text{max } (f x) 0)$

define *fm* **where** $fm = (\lambda x. \text{max } (-f x) 0)$

have *[measurable]: fp* \in *borel-measurable M* *fm* \in *borel-measurable M*

unfolding *fp-def fm-def* **by** *simp-all*

have *[measurable]: fp* \in *borel-measurable F* *fm* \in *borel-measurable F*

unfolding *fp-def fm-def* **by** *simp-all*

have $(\int^+ x. \text{norm}(fp x * g x) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (*simp add: fp-def nn-integral-mono*)

also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)

finally have $(\int^+ x. \text{norm}(fp x * g x) \partial M) < \infty$ **by** *simp*

then have *intp: integrable M* $(\lambda x. fp x * g x)$ **by** (*simp add: integrableI-bounded*)

moreover have $\bigwedge x. fp x \geq 0$ **unfolding** *fp-def* **by** *simp*

ultimately have *Rp: integrable M* $(\lambda x. fp x * \text{real-cond-exp } M F g x)$

$(\int x. fp x * \text{real-cond-exp } M F g x \partial M) = (\int x. fp x * g x \partial M)$

using *real-cond-exp-intg-fpos* **by** *auto*

have $(\int^+ x. \text{norm}(fm x * g x) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (*simp add: fm-def nn-integral-mono*)

also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)

finally have $(\int^+ x. \text{norm}(f m x * g x) \partial M) < \infty$ **by** *simp*
then have *intm*: *integrable* $M (\lambda x. f m x * g x)$ **by** (*simp add*: *integrableI-bounded*)
moreover have $\bigwedge x. f m x \geq 0$ **unfolding** *fm-def* **by** *simp*
ultimately have *Rm*: *integrable* $M (\lambda x. f m x * \text{real-cond-exp } M F g x)$
 $(\int x. f m x * \text{real-cond-exp } M F g x \partial M) = (\int x. f m x * g x \partial M)$
using *real-cond-exp-intg-fpos* **by** *auto*

have *integrable* $M (\lambda x. f p x * \text{real-cond-exp } M F g x - f m x * \text{real-cond-exp } M F g x)$
using *Rp(1)* *Rm(1)* *integrable-diff* **by** *simp*
moreover have $*$: $\bigwedge x. f x * \text{real-cond-exp } M F g x = f p x * \text{real-cond-exp } M F g x - f m x * \text{real-cond-exp } M F g x$
unfolding *fp-def fm-def* **by** (*simp add*: *max-def*)
ultimately show *integrable* $M (\lambda x. f x * \text{real-cond-exp } M F g x)$
by *simp*

have $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f p x * \text{real-cond-exp } M F g x - f m x * \text{real-cond-exp } M F g x \partial M)$
using $*$ **by** *simp*
also have $\dots = (\int x. f p x * \text{real-cond-exp } M F g x \partial M) - (\int x. f m x * \text{real-cond-exp } M F g x \partial M)$
using *Rp(1)* *Rm(1)* **by** *simp*
also have $\dots = (\int x. f p x * g x \partial M) - (\int x. f m x * g x \partial M)$
using *Rp(2)* *Rm(2)* **by** *simp*
also have $\dots = (\int x. f p x * g x - f m x * g x \partial M)$
using *intm intp* **by** *simp*
also have $\dots = (\int x. f x * g x \partial M)$
unfolding *fp-def fm-def* **by** (*metis* (*no-types*, *hide-lams*) *diff-0 diff-zero max commute max-def minus-minus mult commute neg-le-iff-le right-diff-distrib*)
finally show $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$ **by** *simp*
qed

lemma *real-cond-exp-intA*:

assumes [*measurable*]: *integrable* $M f A \in \text{sets } F$
shows $(\int x \in A. f x \partial M) = (\int x \in A. \text{real-cond-exp } M F f x \partial M)$
proof –
have $A \in \text{sets } M$ **by** (*meson assms(2) subalg subalgebra-def subsetD*)
have *integrable* $M (\lambda x. \text{indicator } A x * f x)$ **using** *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *assms(1)*] **by** *auto*
then show *thesis* **using** *real-cond-exp-intg(2)*[**where** $?f = \text{indicator } A$ **and** $?g = f$, *symmetric*] **by** *auto*
qed

lemma *real-cond-exp-int* [*intro*]:

assumes *integrable* $M f$
shows *integrable* $M (\text{real-cond-exp } M F f)$ $(\int x. \text{real-cond-exp } M F f x \partial M) = (\int x. f x \partial M)$
using *real-cond-exp-intg*[**where** $?f = \lambda x. 1$ **and** $?g = f$] *assms* **by** *auto*

lemma *real-cond-exp-charact*:
assumes $\bigwedge A. A \in \text{sets } F \implies (\int x \in A. f x \partial M) = (\int x \in A. g x \partial M)$
and *[measurable]: integrable M f integrable M g*
 $g \in \text{borel-measurable } F$
shows *AE x in M. real-cond-exp M F f x = g x*
proof –
let $?MF = \text{restr-to-subalg } M F$
have *AE x in ?MF. real-cond-exp M F f x = g x*
proof (*rule AE-symmetric[OF density-unique-real]*)
fix A **assume** $A \in \text{sets } ?MF$
then have *[measurable]: A ∈ sets F using sets-restr-to-subalg[OF subalg] by simp*
then have *a [measurable]: A ∈ sets M by (meson subalg subalgebra-def subsetD)*
have $(\int x \in A. g x \partial ?MF) = (\int x \in A. g x \partial M)$
by (*simp add: integral-subalgebra2 subalg*)
also have $\dots = (\int x \in A. f x \partial M)$ **using** *assms(1) by simp*
also have $\dots = (\int x. \text{indicator } A x * f x \partial M)$ **by** (*simp add: mult.commute*)
also have $\dots = (\int x. \text{indicator } A x * \text{real-cond-exp } M F f x \partial M)$
apply (*rule real-cond-exp-intg(2)[symmetric]*) **using** *integrable-mult-indicator[OF a assms(2)] by (auto simp add: assms)*
also have $\dots = (\int x \in A. \text{real-cond-exp } M F f x \partial M)$ **by** (*simp add: mult.commute*)
also have $\dots = (\int x \in A. \text{real-cond-exp } M F f x \partial ?MF)$
by (*simp add: integral-subalgebra2 subalg*)
finally show $(\int x \in A. g x \partial ?MF) = (\int x \in A. \text{real-cond-exp } M F f x \partial ?MF)$
by *simp*
next
have *integrable M (real-cond-exp M F f) by (rule real-cond-exp-int(1)[OF assms(2)])*
then show *integrable ?MF (real-cond-exp M F f) by (metis borel-measurable-cond-exp integrable-in-subalg[OF subalg])*
show *integrable (restr-to-subalg M F) g by (simp add: assms(3) integrable-in-subalg[OF subalg])*
qed
then show *?thesis using AE-restr-to-subalg[OF subalg] by auto*
qed

lemma *real-cond-exp-F-meas [intro, simp]*:
assumes *integrable M f*
 $f \in \text{borel-measurable } F$
shows *AE x in M. real-cond-exp M F f x = f x*
by (*rule real-cond-exp-charact, auto simp add: assms measurable-from-subalg[OF subalg]*)

lemma *real-cond-exp-mult*:
assumes *[measurable]: f ∈ borel-measurable F g ∈ borel-measurable M integrable M (λx. f x * g x)*
shows *AE x in M. real-cond-exp M F (λx. f x * g x) x = f x * real-cond-exp M F g x*

proof (*rule real-cond-exp-charact*)
fix A **assume** $A \in \text{sets } F$
then have [*measurable*]: $(\lambda x. f x * \text{indicator } A x) \in \text{borel-measurable } F$ **by** *measurable*
have [*measurable*]: $A \in \text{sets } M$ **using** *subalg* **by** (*meson* $\langle A \in \text{sets } F \rangle$ *subalgebra-def subsetD*)
have $\int x \in A. (f x * g x) \partial M = \int x. (f x * \text{indicator } A x) * g x \partial M$
by (*simp add: mult.commute mult.left-commute*)
also have $\dots = \int x. (f x * \text{indicator } A x) * \text{real-cond-exp } M F g x \partial M$
apply (*rule real-cond-exp-intg(2)[symmetric]*, *auto simp add: assms*)
using *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *assms(3)*] **by** (*simp add: mult.commute mult.left-commute*)
also have $\dots = \int x \in A. (f x * \text{real-cond-exp } M F g x) \partial M$
by (*simp add: mult.commute mult.left-commute*)
finally show $\int x \in A. (f x * g x) \partial M = \int x \in A. (f x * \text{real-cond-exp } M F g x) \partial M$
by *simp*
qed (*auto simp add: real-cond-exp-intg(1) assms*)

lemma *real-cond-exp-add* [*intro*]:
assumes [*measurable*]: *integrable* $M f$ *integrable* $M g$
shows $A E x$ *in* M . *real-cond-exp* $M F (\lambda x. f x + g x) x = \text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x$
proof (*rule real-cond-exp-charact*)
have *integrable* $M (\text{real-cond-exp } M F f)$ *integrable* $M (\text{real-cond-exp } M F g)$
using *real-cond-exp-int(1) assms* **by** *auto*
then show *integrable* $M (\lambda x. \text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x)$
by *auto*

fix A **assume** [*measurable*]: $A \in \text{sets } F$
then have $A \in \text{sets } M$ **by** (*meson* *subalg subalgebra-def subsetD*)
have *intAf*: *integrable* $M (\lambda x. \text{indicator } A x * f x)$
using *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *assms(1)*] **by** *auto*
have *intAg*: *integrable* $M (\lambda x. \text{indicator } A x * g x)$
using *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *assms(2)*] **by** *auto*

have $\int x \in A. (\text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x) \partial M = (\int x \in A. \text{real-cond-exp } M F f x \partial M) + (\int x \in A. \text{real-cond-exp } M F g x \partial M)$
apply (*rule set-integral-add, auto simp add: assms*)
using *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *real-cond-exp-int(1)* [*OF* *assms(1)*]]
integrable-mult-indicator[*OF* $\langle A \in \text{sets } M \rangle$ *real-cond-exp-int(1)* [*OF* *assms(2)*]] **by** *simp-all*
also have $\dots = (\int x. \text{indicator } A x * \text{real-cond-exp } M F f x \partial M) + (\int x. \text{indicator } A x * \text{real-cond-exp } M F g x \partial M)$
by *auto*
also have $\dots = (\int x. \text{indicator } A x * f x \partial M) + (\int x. \text{indicator } A x * g x \partial M)$
using *real-cond-exp-intg(2) assms* $\langle A \in \text{sets } F \rangle$ *intAf intAg* **by** *auto*
also have $\dots = (\int x \in A. f x \partial M) + (\int x \in A. g x \partial M)$
by *auto*
also have $\dots = \int x \in A. (f x + g x) \partial M$

by (rule *set-integral-add*(2)[*symmetric*]) (auto simp add: *assms* $\langle A \in \text{sets } M \rangle$ *intAf intAg*)

finally show $\int x \in A. (f x + g x) \partial M = \int x \in A. (\text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x) \partial M$

by *simp*

qed (auto simp add: *assms*)

lemma *real-cond-exp-cong*:

assumes *ae*: $AE x \text{ in } M. f x = g x$ **and** [*measurable*]: $f \in \text{borel-measurable } M g \in \text{borel-measurable } M$

shows $AE x \text{ in } M. \text{real-cond-exp } M F f x = \text{real-cond-exp } M F g x$

proof –

have $AE x \text{ in } M. \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (f x)) x = \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (g x)) x$

apply (rule *nn-cond-exp-cong*) **using** *assms* **by** *auto*

moreover have $AE x \text{ in } M. \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-f x)) x = \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-g x)) x$

apply (rule *nn-cond-exp-cong*) **using** *assms* **by** *auto*

ultimately show $AE x \text{ in } M. \text{real-cond-exp } M F f x = \text{real-cond-exp } M F g x$

unfolding *real-cond-exp-def* **by** *auto*

qed

lemma *real-cond-exp-cmult* [*intro*, *simp*]:

fixes *c*::*real*

assumes *integrable* $M f$

shows $AE x \text{ in } M. \text{real-cond-exp } M F (\lambda x. c * f x) x = c * \text{real-cond-exp } M F f x$

by (rule *real-cond-exp-mult*[**where** $?f = \lambda x. c$ **and** $?g = f$], auto simp add: *assms* *borel-measurable-integrable*)

lemma *real-cond-exp-cdiv* [*intro*, *simp*]:

fixes *c*::*real*

assumes *integrable* $M f$

shows $AE x \text{ in } M. \text{real-cond-exp } M F (\lambda x. f x / c) x = \text{real-cond-exp } M F f x / c$

using *real-cond-exp-cmult*[*of* $- 1/c$, *OF* *assms*] **by** (auto simp add: *divide-simps*)

lemma *real-cond-exp-diff* [*intro*, *simp*]:

assumes [*measurable*]: *integrable* $M f$ *integrable* $M g$

shows $AE x \text{ in } M. \text{real-cond-exp } M F (\lambda x. f x - g x) x = \text{real-cond-exp } M F f x - \text{real-cond-exp } M F g x$

proof –

have $AE x \text{ in } M. \text{real-cond-exp } M F (\lambda x. f x + (- g x)) x = \text{real-cond-exp } M F f x + \text{real-cond-exp } M F (\lambda x. -g x) x$

using *real-cond-exp-add*[**where** $?f = f$ **and** $?g = \lambda x. -g x$] *assms* **by** *auto*

moreover have $AE x \text{ in } M. \text{real-cond-exp } M F (\lambda x. -g x) x = - \text{real-cond-exp } M F g x$

using *real-cond-exp-cmult*[**where** $?f = g$ **and** $?c = -1$] *assms*(2) **by** *auto*

ultimately show *?thesis* **by** *auto*

qed

lemma *real-cond-exp-pos* [intro]:

assumes *AE x in M. f x ≥ 0* **and** [measurable]: *f ∈ borel-measurable M*
shows *AE x in M. real-cond-exp M F f x ≥ 0*

proof –

define *g* **where** *g = (λx. max (f x) 0)*

have *AE x in M. f x = g x* **using** *assms g-def* **by** *auto*

then have *: *AE x in M. real-cond-exp M F f x = real-cond-exp M F g x* **using**
real-cond-exp-cong g-def **by** *auto*

have $\bigwedge x. g x \geq 0$ **unfolding** *g-def* **by** *simp*

then have $(\lambda x. \text{ennreal}(-g x)) = (\lambda x. 0)$

by (*simp add: ennreal-neg*)

moreover have *AE x in M. 0 = nn-cond-exp M F (λx. 0) x*

by (*rule nn-cond-exp-F-meas, auto*)

ultimately have *AE x in M. nn-cond-exp M F (λx. ennreal(-g x)) x = 0*

by *simp*

then have *AE x in M. real-cond-exp M F g x = enn2real(nn-cond-exp M F (λx. ennreal (g x)) x)*

unfolding *real-cond-exp-def* **by** *auto*

then have *AE x in M. real-cond-exp M F g x ≥ 0* **by** *auto*

then show ?thesis **using** * **by** *auto*

qed

lemma *real-cond-exp-mono*:

assumes *AE x in M. f x ≤ g x* **and** [measurable]: *integrable M f integrable M g*
shows *AE x in M. real-cond-exp M F f x ≤ real-cond-exp M F g x*

proof –

have *AE x in M. real-cond-exp M F g x – real-cond-exp M F f x = real-cond-exp M F (λx. g x – f x) x*

by (*rule AE-symmetric[OF real-cond-exp-diff], auto simp add: assms*)

moreover have *AE x in M. real-cond-exp M F (λx. g x – f x) x ≥ 0*

by (*rule real-cond-exp-pos, auto simp add: assms(1)*)

ultimately have *AE x in M. real-cond-exp M F g x – real-cond-exp M F f x ≥ 0* **by** *auto*

then show ?thesis **by** *auto*

qed

lemma (in –) *measurable-P-restriction* [measurable (raw)]:

assumes [measurable]: *Measurable.pred M P A ∈ sets M*

shows $\{x \in A. P x\} \in \text{sets } M$

proof –

have $A \subseteq \text{space } M$ **using** *sets.sets-into-space[OF assms(2)]*.

then have $\{x \in A. P x\} = A \cap \{x \in \text{space } M. P x\}$ **by** *blast*

then show ?thesis **by** *auto*

qed

lemma *real-cond-exp-gr-c*:

```

assumes [measurable]: integrable M f
and AE x in M. f x > c
shows AE x in M. real-cond-exp M F f x > c
proof –
define X where X = {x ∈ space M. real-cond-exp M F f x ≤ c}
have [measurable]: X ∈ sets F
unfolding X-def apply measurable by (metis sets.top subalg subalgebra-def)
then have [measurable]: X ∈ sets M using sets-restr-to-subalg subalg subalgebra-def
by blast
have emeasure M X = 0
proof (rule ccontr)
assume ¬(emeasure M X) = 0
have emeasure (restr-to-subalg M F) X = emeasure M X
by (simp add: emeasure-restr-to-subalg subalg)
then have emeasure (restr-to-subalg M F) X > 0
using ⟨¬(emeasure M X) = 0⟩ gr-zeroI by auto
then obtain A where A ∈ sets (restr-to-subalg M F) A ⊆ X emeasure
(restr-to-subalg M F) A > 0 emeasure (restr-to-subalg M F) A < ∞
using sigma-fin-subalg by (metis emeasure-notin-sets ennreal-0 infinity-ennreal-def
le-less-linear neq-top-trans
not-gr-zero order-refl sigma-finite-measure.approx-PInf-emeasure-with-finite)
then have [measurable]: A ∈ sets F using subalg sets-restr-to-subalg by blast
then have [measurable]: A ∈ sets M using sets-restr-to-subalg subalg subalgebra-def
by blast
have Ic: set-integrable M A (λx. c)
using ⟨emeasure (restr-to-subalg M F) A < ∞⟩ emeasure-restr-to-subalg subalg
by fastforce
have If: set-integrable M A f
by (rule integrable-mult-indicator, auto simp add: ⟨integrable M f⟩)
have *: (∫ x∈A. c ∂M) = (∫ x∈A. f x ∂M)
proof (rule antisym)
show (∫ x∈A. c ∂M) ≤ (∫ x∈A. f x ∂M)
apply (rule set-integral-mono-AE) using Ic If assms(2) by auto
have (∫ x∈A. f x ∂M) = (∫ x∈A. real-cond-exp M F f x ∂M)
by (rule real-cond-exp-intA, auto simp add: ⟨integrable M f⟩)
also have ... ≤ (∫ x∈A. c ∂M)
apply (rule set-integral-mono)
apply (rule integrable-mult-indicator, simp, simp add: real-cond-exp-int(1)[OF
⟨integrable M f⟩])
using Ic X-def ⟨A ⊆ X⟩ by auto
finally show (∫ x∈A. f x ∂M) ≤ (∫ x∈A. c ∂M) by simp
qed
have AE x in M. indicator A x * c = indicator A x * f x
apply (rule integral-ineq-eq-0-then-AE) using Ic If * apply auto
using assms(2) unfolding indicator-def by auto
then have AE x∈A in M. c = f x by auto
then have AE x∈A in M. False using assms(2) by auto
have A ∈ null-sets M unfolding ae-filter-def by (meson AE-iff-null-sets ⟨A
∈ sets M⟩ ⟨AE x∈A in M. False⟩)

```

then show *False* **using** $\langle \text{emeasure } (\text{restr-to-subalg } M F) A > 0 \rangle$
by $(\text{simp add: emeasure-restr-to-subalg null-setsD1 subalg})$
qed
then show *?thesis* **using** $\text{AE-iff-null-sets}[OF \langle X \in \text{sets } M \rangle]$ **unfolding** $X\text{-def}$
by *auto*
qed

lemma *real-cond-exp-less-c*:
assumes $[\text{measurable}]$: $\text{integrable } M f$
and $\text{AE } x \text{ in } M. f x < c$
shows $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x < c$
proof –
have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x = -\text{real-cond-exp } M F (\lambda x. -f x) x$
using $\text{real-cond-exp-cmult}[OF \langle \text{integrable } M f \rangle, \text{of } -1]$ **by** *auto*
moreover **have** $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. -f x) x > -c$
apply $(\text{rule } \text{real-cond-exp-gr-c})$ **using** *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-ge-c*:
assumes $[\text{measurable}]$: $\text{integrable } M f$
and $\text{AE } x \text{ in } M. f x \geq c$
shows $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x \geq c$
proof –
obtain $u::\text{nat} \Rightarrow \text{real}$ **where** $u: \bigwedge n. u n < c \ u \longrightarrow c$
using $\text{approx-from-below-dense-linorder}[of \ c-1 \ c]$ **by** *auto*
have $*$: $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x > u n$ **for** $n::\text{nat}$
apply $(\text{rule } \text{real-cond-exp-gr-c})$ **using** *assms* $\langle u n < c \rangle$ **by** *auto*
have $\text{AE } x \text{ in } M. \forall n. \text{real-cond-exp } M F f x > u n$
by $(\text{subst } \text{AE-all-countable}, \text{auto } \text{simp add: } *)$
moreover **have** $\text{real-cond-exp } M F f x \geq c$ **if** $\forall n. \text{real-cond-exp } M F f x > u n$
for x
proof –
have $\text{real-cond-exp } M F f x \geq u n$ **for** n **using** *that less-imp-le* **by** *auto*
then show *?thesis* **using** $u(2)$ LIMSEQ-le-const2 **by** *blast*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-le-c*:
assumes $[\text{measurable}]$: $\text{integrable } M f$
and $\text{AE } x \text{ in } M. f x \leq c$
shows $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x \leq c$
proof –
have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x = -\text{real-cond-exp } M F (\lambda x. -f x) x$
using $\text{real-cond-exp-cmult}[OF \langle \text{integrable } M f \rangle, \text{of } -1]$ **by** *auto*
moreover **have** $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. -f x) x \geq -c$
apply $(\text{rule } \text{real-cond-exp-ge-c})$ **using** *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*

qed

lemma *real-cond-exp-mono-strict*:

assumes *AE x in M. f x < g x* **and** [*measurable*]: *integrable M f integrable M g*
shows *AE x in M. real-cond-exp M F f x < real-cond-exp M F g x*

proof –

have *AE x in M. real-cond-exp M F g x – real-cond-exp M F f x = real-cond-exp M F (λx. g x – f x) x*

by (*rule AE-symmetric[OF real-cond-exp-diff], auto simp add: assms*)

moreover have *AE x in M. real-cond-exp M F (λx. g x – f x) x > 0*

by (*rule real-cond-exp-gr-c, auto simp add: assms*)

ultimately have *AE x in M. real-cond-exp M F g x – real-cond-exp M F f x > 0* **by** *auto*

then show *?thesis* **by** *auto*

qed

lemma *real-cond-exp-nested-subalg* [*intro, simp*]:

assumes *subalgebra M G subalgebra G F*

and [*measurable*]: *integrable M f*

shows *AE x in M. real-cond-exp M F (real-cond-exp M G f) x = real-cond-exp M F f x*

proof (*rule real-cond-exp-charact*)

interpret *G: sigma-finite-subalgebra M G* **by** (*rule nested-subalg-is-sigma-finite[OF assms(1) assms(2)]*)

show *integrable M (real-cond-exp M G f)* **by** (*auto simp add: assms G.real-cond-exp-int(1)*)

fix *A* **assume** [*measurable*]: *A ∈ sets F*

then have [*measurable*]: *A ∈ sets G* **using** *assms(2)* **by** (*meson set-mp subalgebra-def*)

have *set-lebesgue-integral M A (real-cond-exp M G f) = set-lebesgue-integral M A f*

by (*rule G.real-cond-exp-intA[symmetric], auto simp add: assms(3)*)

also have *... = set-lebesgue-integral M A (real-cond-exp M F f)*

by (*rule real-cond-exp-intA, auto simp add: assms(3)*)

finally show *set-lebesgue-integral M A (real-cond-exp M G f) = set-lebesgue-integral M A (real-cond-exp M F f)* **by** *auto*

qed (*auto simp add: assms real-cond-exp-int(1)*)

lemma *real-cond-exp-sum* [*intro, simp*]:

fixes *f::'b ⇒ 'a ⇒ real*

assumes [*measurable*]: $\bigwedge i. \text{integrable } M (f i)$

shows *AE x in M. real-cond-exp M F (λx. $\sum i \in I. f i x$) x = ($\sum i \in I. \text{real-cond-exp M F (f i) x}$)*

proof (*rule real-cond-exp-charact*)

fix *A* **assume** [*measurable*]: *A ∈ sets F*

then have *A-meas* [*measurable*]: *A ∈ sets M* **by** (*meson set-mp subalg subalgebra-def*)

have ***: *integrable M (λx. indicator A x * f i x)* **for** *i*

using *integrable-mult-indicator[OF (A ∈ sets M) assms(1)]* **by** *auto*

have ****: *integrable M (λx. indicator A x * real-cond-exp M F (f i) x)* **for** *i*

using *integrable-mult-indicator[OF (A ∈ sets M) real-cond-exp-int(1)[OF assms(1)]]*

by auto

have inti: $(\int x. \text{indicator } A \ x * f \ i \ x \ \partial M) = (\int x. \text{indicator } A \ x * \text{real-cond-exp } M \ F \ (f \ i) \ x \ \partial M)$ for i
by (rule real-cond-exp-intg(2)[symmetric], auto simp add: *)

have $(\int x \in A. (\sum i \in I. f \ i \ x) \ \partial M) = (\int x. (\sum i \in I. \text{indicator } A \ x * f \ i \ x) \ \partial M)$
by (simp add: sum-distrib-left)

also have ... = $(\sum i \in I. (\int x. \text{indicator } A \ x * f \ i \ x \ \partial M))$
by (rule Bochner-Integration.integral-sum, simp add: *)

also have ... = $(\sum i \in I. (\int x. \text{indicator } A \ x * \text{real-cond-exp } M \ F \ (f \ i) \ x \ \partial M))$
using inti by auto

also have ... = $(\int x. (\sum i \in I. \text{indicator } A \ x * \text{real-cond-exp } M \ F \ (f \ i) \ x) \ \partial M)$
by (rule Bochner-Integration.integral-sum[symmetric], simp add: **)

also have ... = $(\int x \in A. (\sum i \in I. \text{real-cond-exp } M \ F \ (f \ i) \ x) \ \partial M)$
by (simp add: sum-distrib-left)

finally show $(\int x \in A. (\sum i \in I. f \ i \ x) \ \partial M) = (\int x \in A. (\sum i \in I. \text{real-cond-exp } M \ F \ (f \ i) \ x) \ \partial M)$ by auto

qed (auto simp add: assms real-cond-exp-int(1)[OF assms(1)])

Jensen’s inequality, describing the behavior of the integral under a convex function, admits a version for the conditional expectation, as follows.

theorem real-cond-exp-jensens-inequality:

fixes q :: real \Rightarrow real

assumes X: integrable M X AE x in M. X x \in I

assumes I: I = {a <..< b} \vee I = {a <..} \vee I = {..< b} \vee I = UNIV

assumes q: integrable M $(\lambda x. q \ (X \ x))$ convex-on I q q \in borel-measurable borel
shows AE x in M. real-cond-exp M F X x \in I

AE x in M. q (real-cond-exp M F X x) \leq real-cond-exp M F $(\lambda x. q \ (X \ x)) \ x$

proof –

have open I using I by auto

then have interior I = I by (simp add: interior-eq)

have [measurable]: I \in sets borel using I by auto

define phi where phi = $(\lambda x. \text{Inf } ((\lambda t. (q \ x - q \ t) / (x - t)) \ ‘ (\{x <..\} \cap I)))$

have **: q (X x) \geq q (real-cond-exp M F X x) + phi (real-cond-exp M F X x)
* (X x - real-cond-exp M F X x)

if X x \in I real-cond-exp M F X x \in I for x

unfolding phi-def apply (rule convex-le-Inf-differential)

using <convex-on I q> that <interior I = I> by auto

It is not clear that the function ϕ is measurable. We replace it by a version which is better behaved.

define psi where psi = $(\lambda x. \text{phi } x * \text{indicator } I \ x)$

have A: psi y = phi y if y \in I for y unfolding psi-def indicator-def using that
by auto

have *: q (X x) \geq q (real-cond-exp M F X x) + psi (real-cond-exp M F X x) *
(X x - real-cond-exp M F X x)

if X x \in I real-cond-exp M F X x \in I for x

unfolding A[OF <real-cond-exp M F X x \in I>] using ** that by auto

note I
moreover have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ X\ x\ >\ a\ if\ I\ \subseteq\ \{a\ <..\}$ **for** a
apply (rule *real-cond-exp-gr-c*) **using** X **that by** *auto*
moreover have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ X\ x\ <\ b\ if\ I\ \subseteq\ \{..<b\}$ **for** b
apply (rule *real-cond-exp-less-c*) **using** X **that by** *auto*
ultimately show $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ X\ x\ \in\ I$
by (*elim disjE*) (*auto simp: subset-eq*)
then have *main-ineq*: $AE\ x\ in\ M.\ q\ (X\ x)\ \geq\ q\ (real\ cond\ exp\ M\ F\ X\ x)\ +\ psi$
 $(real\ cond\ exp\ M\ F\ X\ x)\ * (X\ x\ -\ real\ cond\ exp\ M\ F\ X\ x)$
using $*\ X(2)$ **by** *auto*

Then, one wants to take the conditional expectation of this inequality. On the left, one gets the conditional expectation of $q \circ X$. On the right, the last term vanishes, and one is left with q of the conditional expectation, as desired. Unfortunately, this argument only works if $\psi \cdot X$ and $q(E(X|F))$ are integrable, and there is no reason why this should be true. The trick is to multiply by a F -measurable function which is small enough to make everything integrable.

obtain $f::'a\ \Rightarrow\ real$ **where** [*measurable*]: $f\ \in\ borel\ measurable\ (restr\ to\ subalg\ M\ F)$
 $integrable\ (restr\ to\ subalg\ M\ F)\ f$
and $f: \bigwedge x.\ f\ x\ >\ 0\ \bigwedge x.\ f\ x\ \leq\ 1$
using *sigma-finite-measure.obtain-positive-integrable-function*[*OF sigma-fin-subalg*]
by *metis*
then have [*measurable*]: $f\ \in\ borel\ measurable\ F$ **by** (*simp add: subalg*)
then have [*measurable*]: $f\ \in\ borel\ measurable\ M$ **using** *measurable-from-subalg*[*OF subalg*] **by** *blast*
define g **where** $g = (\lambda x.\ f\ x / (1 + |psi\ (real\ cond\ exp\ M\ F\ X\ x)| + |q\ (real\ cond\ exp\ M\ F\ X\ x)|))$
define G **where** $G = (\lambda x.\ g\ x * psi\ (real\ cond\ exp\ M\ F\ X\ x))$
have $g: g\ x\ >\ 0\ g\ x\ \leq\ 1$ **for** x **unfolding** *G-def g-def* **using** $f[of\ x]$ **by** (*auto simp add: abs-mult*)
have $G: |G\ x| \leq 1$ **for** x **unfolding** *G-def g-def* **using** $f[of\ x]$
proof (*auto simp add: abs-mult*)
have $f\ x * |psi\ (real\ cond\ exp\ M\ F\ X\ x)| \leq 1 * |psi\ (real\ cond\ exp\ M\ F\ X\ x)|$
apply (*rule mult-mono*) **using** $f[of\ x]$ **by** *auto*
also have $... \leq 1 + |psi\ (real\ cond\ exp\ M\ F\ X\ x)| + |q\ (real\ cond\ exp\ M\ F\ X\ x)|$ **by** *auto*
finally show $f\ x * |psi\ (real\ cond\ exp\ M\ F\ X\ x)| \leq 1 + |psi\ (real\ cond\ exp\ M\ F\ X\ x)| + |q\ (real\ cond\ exp\ M\ F\ X\ x)|$
by *simp*
qed
have $AE\ x\ in\ M.\ g\ x * q\ (X\ x)\ \geq\ g\ x * (q\ (real\ cond\ exp\ M\ F\ X\ x)\ +\ psi$
 $(real\ cond\ exp\ M\ F\ X\ x)\ * (X\ x\ -\ real\ cond\ exp\ M\ F\ X\ x))$
using *main-ineq g* **by** (*auto simp add: divide-simps*)
then have *main-G*: $AE\ x\ in\ M.\ g\ x * q\ (X\ x)\ \geq\ g\ x * q\ (real\ cond\ exp\ M\ F\ X\ x)$
 $+ G\ x * (X\ x\ -\ real\ cond\ exp\ M\ F\ X\ x)$
unfolding *G-def* **by** (*auto simp add: algebra-simps*)

To proceed, we need to know that ψ is measurable.

```

have phi-mono:  $\phi x \leq \phi y$  if  $x \leq y$   $x \in I$   $y \in I$  for  $x$   $y$ 
proof (cases  $x < y$ )
  case True
    have  $q x + \phi x * (y-x) \leq q y$ 
      unfolding phi-def apply (rule convex-le-Inf-differential) using  $\langle$ convex-on I
q $\rangle$  that  $\langle$ interior I = I $\rangle$  by auto
    then have  $\phi x \leq (q x - q y) / (x - y)$ 
      using that  $\langle$  $x < y$  $\rangle$  by (auto simp add: divide-simps algebra-simps)
    moreover have  $(q x - q y) / (x - y) \leq \phi y$ 
      unfolding phi-def proof (rule cInf-greatest, auto)
      fix  $t$  assume  $t \in I$   $y < t$ 
      have  $(q x - q y) / (x - y) \leq (q x - q t) / (x - t)$ 
        apply (rule convex-on-diff[OF q(2)]) using  $\langle$  $y < t$  $\rangle$   $\langle$  $x < y$  $\rangle$   $\langle$  $t \in I$  $\rangle$   $\langle$  $x \in I$  $\rangle$ 
by auto
      also have  $\dots \leq (q y - q t) / (y - t)$ 
        apply (rule convex-on-diff[OF q(2)]) using  $\langle$  $y < t$  $\rangle$   $\langle$  $x < y$  $\rangle$   $\langle$  $t \in I$  $\rangle$   $\langle$  $x \in I$  $\rangle$ 
by auto
      finally show  $(q x - q y) / (x - y) \leq (q y - q t) / (y - t)$  by simp
    next
      obtain  $e$  where  $0 < e$  ball  $y e \subseteq I$  using  $\langle$ open I $\rangle$   $\langle$  $y \in I$  $\rangle$  openE by blast
      then have  $y + e/2 \in \{y <..\} \cap I$  by (auto simp: dist-real-def)
      then show  $\{y <..\} \cap I = \{\}$   $\implies$  False by auto
    qed
    ultimately show  $\phi x \leq \phi y$  by auto
  next
    case False
      then have  $x = y$  using  $\langle$  $x \leq y$  $\rangle$  by auto
      then show ?thesis by auto
    qed
have [measurable]:  $\psi \in$  borel-measurable borel
  by (rule borel-measurable-piecewise-mono[of {I, -I}])
  (auto simp add: psi-def indicator-def phi-mono intro: mono-onI)
have [measurable]:  $q \in$  borel-measurable borel using  $q$  by simp

have [measurable]:  $X \in$  borel-measurable M
  real-cond-exp M F X  $\in$  borel-measurable F
   $g \in$  borel-measurable F  $g \in$  borel-measurable M
   $G \in$  borel-measurable F  $G \in$  borel-measurable M
  using  $X$  measurable-from-subalg[OF subalg] unfolding  $G$ -def  $g$ -def by auto
have int1: integrable (restr-to-subalg M F)  $(\lambda x. g x * q$  (real-cond-exp M F X
 $x$ ))
  apply (rule Bochner-Integration.integrable-bound[of - f], auto simp add: subalg
integrable (restr-to-subalg M F) f)
  unfolding  $g$ -def by (auto simp add: divide-simps abs-mult algebra-simps)
have int2: integrable M  $(\lambda x. G x * (X x -$  real-cond-exp M F X
 $x$ ))
  apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. |X x| + |$ real-cond-exp
M F X x])
apply (auto intro!: Bochner-Integration.integrable-add integrable-abs real-cond-exp-int

```

$\langle \text{integrable } M X \rangle \text{ AE-I2}$
using G **unfolding** abs-mult **by** $(\text{meson } \text{abs-ge-zero } \text{abs-triangle-ineq4 } \text{dual-order.trans } \text{mult-left-le-one-le})$
have int3 : $\text{integrable } M (\lambda x. g x * q (X x))$
apply $(\text{rule } \text{Bochner-Integration.integrable-bound}[\text{of } - \lambda x. q(X x)], \text{auto simp add: } q(1) \text{ abs-mult})$
using g **by** $(\text{simp add: less-imp-le mult-left-le-one-le})$

Taking the conditional expectation of the main convexity inequality main_G , we get the following.

have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. g x * q (X x)) x \geq \text{real-cond-exp } M F (\lambda x. g x * q (\text{real-cond-exp } M F X x) + G x * (X x - \text{real-cond-exp } M F X x)) x$
apply $(\text{rule } \text{real-cond-exp-mono}[\text{OF } \text{main-G}])$
apply $(\text{rule } \text{Bochner-Integration.integrable-add}[\text{OF } \text{integrable-from-subalg}[\text{OF } \text{subalg } \text{int1}]])$
using $\text{int2 } \text{int3}$ **by** auto

This reduces to the desired inequality thanks to the properties of conditional expectation, i.e., the conditional expectation of an F -measurable function is this function, and one can multiply an F -measurable function outside of conditional expectations. Since all these equalities only hold almost everywhere, we formulate them separately, and then combine all of them to simplify the above equation, again almost everywhere.

moreover have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. g x * q (X x)) x = g x * \text{real-cond-exp } M F (\lambda x. q (X x)) x$
by $(\text{rule } \text{real-cond-exp-mult}, \text{auto simp add: } \text{int3})$
moreover have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. g x * q (\text{real-cond-exp } M F X x) + G x * (X x - \text{real-cond-exp } M F X x)) x$
 $= \text{real-cond-exp } M F (\lambda x. g x * q (\text{real-cond-exp } M F X x)) x + \text{real-cond-exp } M F (\lambda x. G x * (X x - \text{real-cond-exp } M F X x)) x$
by $(\text{rule } \text{real-cond-exp-add}, \text{auto simp add: } \text{integrable-from-subalg}[\text{OF } \text{subalg } \text{int1}] \text{int2})$
moreover have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. g x * q (\text{real-cond-exp } M F X x)) x = g x * q (\text{real-cond-exp } M F X x)$
by $(\text{rule } \text{real-cond-exp-F-meas}, \text{auto simp add: } \text{integrable-from-subalg}[\text{OF } \text{subalg } \text{int1}])$
moreover have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. G x * (X x - \text{real-cond-exp } M F X x)) x = G x * \text{real-cond-exp } M F (\lambda x. (X x - \text{real-cond-exp } M F X x)) x$
by $(\text{rule } \text{real-cond-exp-mult}, \text{auto simp add: } \text{int2})$
moreover have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. (X x - \text{real-cond-exp } M F X x)) x = \text{real-cond-exp } M F X x - \text{real-cond-exp } M F (\lambda x. \text{real-cond-exp } M F X x) x$
by $(\text{rule } \text{real-cond-exp-diff}, \text{auto intro!: } \text{real-cond-exp-int } \langle \text{integrable } M X \rangle)$
moreover have $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. \text{real-cond-exp } M F X x) x = \text{real-cond-exp } M F X x$
by $(\text{rule } \text{real-cond-exp-F-meas}, \text{auto intro!: } \text{real-cond-exp-int } \langle \text{integrable } M X \rangle)$
ultimately have $\text{AE } x \text{ in } M. g x * \text{real-cond-exp } M F (\lambda x. q (X x)) x \geq g x * q (\text{real-cond-exp } M F X x)$

by auto
 then show $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ q\ (X\ x))\ x\ \geq\ q\ (real\ cond\ exp\ M\ F\ X\ x)$
 using $g(1)$ by (auto simp add: divide-simps)
 qed

Jensen’s inequality does not imply that $q(E(X|F))$ is integrable, as it only proves an upper bound for it. Indeed, this is not true in general, as the following counterexample shows:

on $[1, \infty)$ with Lebesgue measure, let F be the sigma-algebra generated by the intervals $[n, n + 1)$ for integer n . Let $q(x) = -\sqrt{x}$ for $x \geq 0$. Define X which is equal to $1/n$ over $[n, n + 1/n)$ and 2^{-n} on $[n + 1/n, n + 1)$. Then X is integrable as $\sum 1/n^2 < \infty$, and $q(X)$ is integrable as $\sum 1/n^{3/2} < \infty$. On the other hand, $E(X|F)$ is essentially equal to $1/n^2$ on $[n, n + 1)$ (we neglect the term 2^{-n} , we only put it there because X should take its values in $I = (0, \infty)$). Hence, $q(E(X|F))$ is equal to $-1/n$ on $[n, n + 1)$, hence it is not integrable.

However, this counterexample is essentially the only situation where this function is not integrable, as shown by the next lemma.

lemma *integrable-convex-cond-exp:*

fixes $q :: real \Rightarrow real$

assumes $X: integrable\ M\ X\ AE\ x\ in\ M.\ X\ x \in I$

assumes $I: I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$

assumes $q: integrable\ M\ (\lambda x.\ q\ (X\ x))\ convex\ on\ I\ q\ q \in borel\ measurable\ borel$

assumes $H: emeasure\ M\ (space\ M) = \infty \implies 0 \in I$

shows $integrable\ M\ (\lambda x.\ q\ (real\ cond\ exp\ M\ F\ X\ x))$

proof –

have $[measurable]: (\lambda x.\ q\ (real\ cond\ exp\ M\ F\ X\ x)) \in borel\ measurable\ M$

$q \in borel\ measurable\ borel$

$X \in borel\ measurable\ M$

using $X(1)\ q(3)$ by auto

have open I using I by auto

then have interior $I = I$ by (simp add: interior-eq)

consider $emeasure\ M\ (space\ M) = 0 \mid emeasure\ M\ (space\ M) > 0 \wedge emeasure\ M\ (space\ M) < \infty \mid emeasure\ M\ (space\ M) = \infty$

by (metis infinity-enreal-def not-gr-zero top.not-eq-extremum)

then show ?thesis

proof (cases)

case 1

show ?thesis by (subst integrable-cong-AE[of - - $\lambda x.\ 0$], auto intro: emeasure-0-AE[OF 1])

next

case 2

interpret finite-measure M using 2 by (auto intro!: finite-measureI)

have $I \neq \{\}$

using $\langle AE\ x\ in\ M.\ X\ x\ \in\ I \rangle$ 2 eventually-mono integral-less-AE-space **by**
fastforce
then obtain z **where** $z \in I$ **by** *auto*

define A **where** $A = \text{Inf } ((\lambda t. (q\ z - q\ t) / (z - t)) \text{ ‘ } (\{z < ..\} \cap I))$
have $q\ y \geq q\ z + A * (y - z)$ **if** $y \in I$ **for** y **unfolding** $A\text{-def}$ **apply** (*rule*
convex-le-Inf-differential)
using $\langle z \in I \rangle \langle y \in I \rangle \langle \text{interior } I = I \rangle$ $q(2)$ **by** *auto*
then have $AE\ x\ in\ M.\ q\ (\text{real-cond-exp } M\ F\ X\ x) \geq q\ z + A * (\text{real-cond-exp } M\ F\ X\ x - z)$
using *real-cond-exp-jensens-inequality(1)[OF X I q]* **by** *auto*
moreover have $AE\ x\ in\ M.\ q\ (\text{real-cond-exp } M\ F\ X\ x) \leq \text{real-cond-exp } M\ F$
 $(\lambda x. q\ (X\ x))\ x$
using *real-cond-exp-jensens-inequality(2)[OF X I q]* **by** *auto*
moreover have $|a| \leq |b| + |c|$ **if** $b \leq a \wedge a \leq c$ **for** $a\ b\ c :: \text{real}$
using *that* **by** *auto*
ultimately have $*$: $AE\ x\ in\ M.\ |q\ (\text{real-cond-exp } M\ F\ X\ x)|$
 $\leq |\text{real-cond-exp } M\ F\ (\lambda x. q\ (X\ x))\ x| + |q\ z + A * (\text{real-cond-exp } M\ F\ X\ x - z)|$
by *auto*

show *integrable* $M\ (\lambda x. q\ (\text{real-cond-exp } M\ F\ X\ x))$
apply (*rule* *Bochner-Integration.integrable-bound[of - \lambda x. |real-cond-exp M F*
 $(\lambda x. q\ (X\ x))\ x| + |q\ z + A * (\text{real-cond-exp } M\ F\ X\ x - z)|]$)
apply (*auto intro!*: *Bochner-Integration.integrable-add integrable-abs integrable-mult-right*
Bochner-Integration.integrable-diff real-cond-exp-int(1))
using $X(1)\ q(1)\ *$ **by** *auto*
next
case 3
then have $0 \in I$ **using** $H\ \text{finite-measure.finite-emeasure-space}$ **by** *auto*
have $q(0) = 0$
proof (*rule* *ccontr*)
assume $*$: $\neg(q(0) = 0)$
define e **where** $e = |q(0)| / 2$
then have $e > 0$ **using** $*$ **by** *auto*
have *continuous* (at 0) q
using $q(2)\ \langle 0 \in I \rangle \langle \text{open } I \rangle \langle \text{interior } I = I \rangle$ *continuous-on-interior*
convex-on-continuous **by** *blast*
then obtain d **where** $d: d > 0 \wedge y. |y - 0| < d \implies |q\ y - q\ 0| < e$ **using**
 $\langle e > 0 \rangle$
by (*metis* *continuous-at-real-range real-norm-def*)
then have $*$: $|q(y)| > e$ **if** $|y| < d$ **for** y
proof –
have $|q\ 0| \leq |q\ 0 - q\ y| + |q\ y|$ **by** *auto*
also have $\dots < e + |q\ y|$ **using** $d(2)$ *that* **by** *force*
finally have $|q\ y| > |q\ 0| - e$ **by** *auto*
then show *?thesis* **unfolding** $e\text{-def}$ **by** *simp*
qed
have *emeasure* $M\ \{x \in \text{space } M.\ |X\ x| < d\} \leq \text{emeasure } M\ (\{x \in \text{space } M.$

$1 \leq \text{ennreal}(1/e) * |q(X x)| \} \cap \text{space } M$
by (rule *emeasure-mono*, *auto simp add: * <e>0 >* *less-imp-le ennreal-mult''[symmetric]*)
also have $\dots \leq (1/e) * (\int^+ x. \text{ennreal}(|q(X x)|) * \text{indicator } (\text{space } M) x \partial M)$
by (rule *nn-integral-Markov-inequality*, *auto*)
also have $\dots = (1/e) * (\int^+ x. \text{ennreal}(|q(X x)|) \partial M)$ **by** *auto*
also have $\dots = (1/e) * \text{ennreal}(\int x. |q(X x)| \partial M)$
using *nn-integral-eq-integral[OF integrable-abs[OF q(1)]]* **by** *auto*
also have $\dots < \infty$
by (*simp add: ennreal-mult-less-top*)
finally have $A: \text{emeasure } M \{x \in \text{space } M. |X x| < d\} < \infty$ **by** *simp*

have $\{x \in \text{space } M. |X x| \geq d\} = \{x \in \text{space } M. 1 \leq \text{ennreal}(1/d) * |X x|\}$
 $\cap \text{space } M$
by (*auto simp add: <d>0 >* *ennreal-mult''[symmetric]*)
then have $\text{emeasure } M \{x \in \text{space } M. |X x| \geq d\} = \text{emeasure } M (\{x \in \text{space } M. 1 \leq \text{ennreal}(1/d) * |X x|\} \cap \text{space } M)$
by *auto*
also have $\dots \leq (1/d) * (\int^+ x. \text{ennreal}(|X x|) * \text{indicator } (\text{space } M) x \partial M)$
by (rule *nn-integral-Markov-inequality*, *auto*)
also have $\dots = (1/d) * (\int^+ x. \text{ennreal}(|X x|) \partial M)$ **by** *auto*
also have $\dots = (1/d) * \text{ennreal}(\int x. |X x| \partial M)$
using *nn-integral-eq-integral[OF integrable-abs[OF X(1)]]* **by** *auto*
also have $\dots < \infty$
by (*simp add: ennreal-mult-less-top*)
finally have $B: \text{emeasure } M \{x \in \text{space } M. |X x| \geq d\} < \infty$ **by** *simp*

have $\text{space } M = \{x \in \text{space } M. |X x| < d\} \cup \{x \in \text{space } M. |X x| \geq d\}$ **by** *auto*
then have $\text{emeasure } M (\text{space } M) = \text{emeasure } M (\{x \in \text{space } M. |X x| < d\} \cup \{x \in \text{space } M. |X x| \geq d\})$
by *simp*
also have $\dots \leq \text{emeasure } M \{x \in \text{space } M. |X x| < d\} + \text{emeasure } M \{x \in \text{space } M. |X x| \geq d\}$
by (*auto intro!: emeasure-subadditive*)
also have $\dots < \infty$ **using** $A B$ **by** *auto*
finally show *False using <emeasure M (space M) = ∞ >* **by** *auto*
qed

define A **where** $A = \text{Inf } ((\lambda t. (q \ 0 - q \ t) / (0 - t)) ' (\{0 <.. \} \cap I))$
have $q \ y \geq q \ 0 + A * (y - 0)$ **if** $y \in I$ **for** y **unfolding** $A\text{-def}$ **apply** (rule *convex-le-Inf-differential*)
using $\langle 0 \in I \rangle \langle y \in I \rangle \langle \text{interior } I = I \rangle q(2)$ **by** *auto*
then have $q \ y \geq A * y$ **if** $y \in I$ **for** y **using** $\langle q \ 0 = 0 \rangle$ **that** **by** *auto*
then have $A E \ x \text{ in } M. q \ (\text{real-cond-exp } M \ F \ X \ x) \geq A * \text{real-cond-exp } M \ F \ X \ x$
using *real-cond-exp-jensens-inequality(1)[OF X I q]* **by** *auto*
moreover have $A E \ x \text{ in } M. q \ (\text{real-cond-exp } M \ F \ X \ x) \leq \text{real-cond-exp } M \ F \ X \ x$
 $(\lambda x. q \ (X \ x)) \ x$
using *real-cond-exp-jensens-inequality(2)[OF X I q]* **by** *auto*


```

moreover have  $|a| \leq |b| + |c|$  if  $b \leq a \wedge a \leq c$  for  $a\ b\ c::real$ 
using that by auto
ultimately have  $*$ :  $AE\ x\ in\ M.$   $|q\ (real-cond-exp\ M\ F\ X\ x)|$ 
 $\leq |real-cond-exp\ M\ F\ (\lambda x. q\ (X\ x))\ x| + |A * real-cond-exp\ M\ F\ X\ x|$ 
by auto

show integrable  $M\ (\lambda x. q\ (real-cond-exp\ M\ F\ X\ x))$ 
apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. |real-cond-exp\ M\ F$ 
 $(\lambda x. q\ (X\ x))\ x| + |A * real-cond-exp\ M\ F\ X\ x|$  ]])
apply (auto intro!: Bochner-Integration.integrable-add integrable-abs integrable-mult-right
Bochner-Integration.integrable-diff real-cond-exp-int(1))
using  $X(1)\ q(1)\ *$  by auto
qed
qed

end

end

```

```

theory Essential-Supremum
imports HOL-Analysis.Analysis
begin

```

```

lemma ae-filter-eq-bot-iff:  $ae-filter\ M = bot \longleftrightarrow emeasure\ M\ (space\ M) = 0$ 
by (simp add: AE-iff-measurable trivial-limit-def)

```

29 The essential supremum

In this paragraph, we define the essential supremum and give its basic properties. The essential supremum of a function is its maximum value if one is allowed to throw away a set of measure 0. It is convenient to define it to be infinity for non-measurable functions, as it allows for neater statements in general. This is a prerequisite to define the space L^∞ .

```

definition esssup:: $'a\ measure \Rightarrow ('a \Rightarrow 'b::\{second-countable-topology, dense-linorder,$ 
 $linorder-topology, complete-linorder\}) \Rightarrow 'b$ 

```

```

where esssup  $M\ f = (if\ f \in borel-measurable\ M\ then\ Limsup\ (ae-filter\ M)\ f\ else\ top)$ 

```

```

lemma esssup-non-measurable:  $f \notin M \rightarrow_M\ borel \implies esssup\ M\ f = top$ 
by (simp add: esssup-def)

```

```

lemma esssup-eq-AE:

```

```

assumes  $f: f \in M \rightarrow_M\ borel$  shows  $esssup\ M\ f = Inf\ \{z. AE\ x\ in\ M. f\ x \leq z\}$ 

```

```

unfolding esssup-def if-P[OF f] Limsup-def

```

```

proof (intro antisym INF-greatest Inf-greatest; clarsimp)

```

```

fix  $y$  assume  $AE\ x\ in\ M. f\ x \leq y$ 

```

```

then have  $(\lambda x. f\ x \leq y) \in \{P. AE\ x\ in\ M. P\ x\}$ 

```

```

  by simp
  then show (INF P:{P. AE x in M. P x}. SUP x:Collect P. f x) ≤ y
    by (rule INF-lower2) (auto intro: SUP-least)
next
  fix P assume P: AE x in M. P x
  show Inf {z. AE x in M. f x ≤ z} ≤ (SUP x:Collect P. f x)
  proof (rule Inf-lower; clarsimp)
    show AE x in M. f x ≤ (SUP x:Collect P. f x)
      using P by (auto elim: eventually-mono simp: SUP-upper)
  qed
qed

lemma esssup-eq: f ∈ M →M borel ⇒ esssup M f = Inf {z. emeasure M {x ∈
space M. f x > z} = 0}
  by (auto simp add: esssup-eq-AE not-less[symmetric] AE-iff-measurable[OF -
refl] intro!: arg-cong[where f=Inf])

lemma esssup-zero-measure:
  emeasure M {x ∈ space M. f x > esssup M f} = 0
proof (cases esssup M f = top)
  case True
  then show ?thesis by auto
next
  case False
  then have f[measurable]: f ∈ M →M borel unfolding esssup-def by meson
  have esssup M f < top using False by (auto simp: less-top)
  have *: {x ∈ space M. f x > z} ∈ null-sets M if z > esssup M f for z
  proof -
    have ∃ w. w < z ∧ emeasure M {x ∈ space M. f x > w} = 0
      using ⟨z > esssup M f⟩ f by (auto simp: esssup-eq Inf-less-iff)
    then obtain w where w < z emeasure M {x ∈ space M. f x > w} = 0 by
auto
    then have a: {x ∈ space M. f x > w} ∈ null-sets M by auto
    have b: {x ∈ space M. f x > z} ⊆ {x ∈ space M. f x > w} using ⟨w < z⟩ by
auto
    show ?thesis using null-sets-subset[OF a - b] by simp
  qed
  obtain u::nat ⇒ 'b where u: ∧n. u n > esssup M f u ⟶ esssup M f
    using approx-from-above-dense-linorder[OF ⟨esssup M f < top⟩] by auto
  have {x ∈ space M. f x > esssup M f} = (⋃ n. {x ∈ space M. f x > u n})
    using u apply auto
  apply (metis (mono-tags, lifting) order-tendsto-iff eventually-mono LIMSEQ-unique)
    using less-imp-le less-le-trans by blast
  also have ... ∈ null-sets M
    using *[OF u(1)] by auto
  finally show ?thesis by auto
qed

lemma esssup-AE: AE x in M. f x ≤ esssup M f

```

proof (cases $f \in M \rightarrow_M \text{borel}$)
case *True* **then show** *?thesis*
by (intro *AE-I*[*OF* - *esssup-zero-measure*[*of* - *f*]]) *auto*
qed (*simp add: esssup-non-measurable*)

lemma *esssup-pos-measure*:
 $f \in \text{borel-measurable } M \implies z < \text{esssup } M f \implies \text{emeasure } M \{x \in \text{space } M. f x > z\} > 0$
using *Inf-less-iff mem-Collect-eq not-gr-zero* **by** (*force simp: esssup-eq*)

lemma *esssup-I* [*intro*]: $f \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x \leq c \implies \text{esssup } M f \leq c$
unfolding *esssup-def* **by** (*simp add: Limsup-bounded*)

lemma *esssup-AE-mono*: $f \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x \leq g x \implies \text{esssup } M f \leq \text{esssup } M g$
by (*auto simp: esssup-def Limsup-mono*)

lemma *esssup-mono*: $f \in \text{borel-measurable } M \implies (\bigwedge x. f x \leq g x) \implies \text{esssup } M f \leq \text{esssup } M g$
by (*rule esssup-AE-mono*) *auto*

lemma *esssup-AE-cong*:
 $f \in \text{borel-measurable } M \implies g \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x = g x \implies \text{esssup } M f = \text{esssup } M g$
by (*auto simp: esssup-def intro!: Limsup-eq*)

lemma *esssup-const*: $\text{emeasure } M (\text{space } M) \neq 0 \implies \text{esssup } M (\lambda x. c) = c$
by (*simp add: esssup-def Limsup-const ae-filter-eq-bot-iff*)

lemma *esssup-cmult*: **assumes** $c > (0::\text{real})$ **shows** $\text{esssup } M (\lambda x. c * f x::\text{ereal}) = c * \text{esssup } M f$

proof –
have $(\lambda x. \text{ereal } c * f x) \in M \rightarrow_M \text{borel} \implies f \in M \rightarrow_M \text{borel}$
proof (*subst measurable-cong*)
fix ω **show** $f \omega = \text{ereal } (1/c) * (\text{ereal } c * f \omega)$
using $\langle 0 < c \rangle$ **by** (*cases f* ω) *auto*
qed *auto*
then have $(\lambda x. \text{ereal } c * f x) \in M \rightarrow_M \text{borel} \iff f \in M \rightarrow_M \text{borel}$
by(*safe intro!: borel-measurable-ereal-times borel-measurable-const*)
with $\langle 0 < c \rangle$ **show** *?thesis*
by (*cases ae-filter M = bot*)
(*auto simp: esssup-def bot-ereal-def top-ereal-def Limsup-ereal-mult-left*)
qed

lemma *esssup-add*:
 $\text{esssup } M (\lambda x. f x + g x::\text{ereal}) \leq \text{esssup } M f + \text{esssup } M g$
proof (cases $f \in \text{borel-measurable } M \wedge g \in \text{borel-measurable } M$)
case *True*

then have $[\text{measurable}]$: $(\lambda x. f x + g x) \in \text{borel-measurable } M$ **by** *auto*
have $f x + g x \leq \text{esssup } M f + \text{esssup } M g$ **if** $f x \leq \text{esssup } M f$ $g x \leq \text{esssup } M g$
for x
using *that ereal-add-mono* **by** *auto*
then have $\text{AE } x \text{ in } M. f x + g x \leq \text{esssup } M f + \text{esssup } M g$
using $\text{esssup-AE}[of f M]$ $\text{esssup-AE}[of g M]$ **by** *auto*
then show *?thesis* **using** *esssup-I* **by** *auto*
next
case *False*
then have $\text{esssup } M f + \text{esssup } M g = \infty$ **unfolding** *esssup-def top-ereal-def*
by *auto*
then show *?thesis* **by** *auto*
qed

lemma *esssup-zero-space*:

$\text{emeasure } M (\text{space } M) = 0 \implies f \in \text{borel-measurable } M \implies \text{esssup } M f = (-\infty :: \text{ereal})$

by (*simp add: esssup-def ae-filter-eq-bot-iff[symmetric] bot-ereal-def*)

end

30 Stopping times

theory *Stopping-Time*

imports *HOL-Analysis.Analysis*

begin

30.1 Stopping Time

This is also called strong stopping time. Then stopping time is T with alternative is $T x < t$ measurable.

definition *stopping-time* :: $(t :: \text{linorder} \implies 'a \text{ measure}) \implies ('a \implies 't) \implies \text{bool}$

where

$\text{stopping-time } F T = (\forall t. \text{Measurable.pred } (F t) (\lambda x. T x \leq t))$

lemma *stopping-time-cong*: $(\bigwedge t x. x \in \text{space } (F t) \implies T x = S x) \implies \text{stopping-time } F T = \text{stopping-time } F S$

unfolding *stopping-time-def* **by** (*intro arg-cong[where f=All] ext measurable-cong*) *simp*

lemma *stopping-timeD*: $\text{stopping-time } F T \implies \text{Measurable.pred } (F t) (\lambda x. T x \leq t)$

by (*auto simp: stopping-time-def*)

lemma *stopping-timeD2*: $\text{stopping-time } F T \implies \text{Measurable.pred } (F t) (\lambda x. t < T x)$

unfolding *not-le[symmetric]* **by** (*auto intro: stopping-timeD Measurable.pred-intros-logic*)

lemma *stopping-timeI*[intro?]: $(\bigwedge t. \text{Measurable.pred } (F t) (\lambda x. T x \leq t)) \implies \text{stopping-time } F T$

by (*auto simp: stopping-time-def*)

lemma *measurable-stopping-time*:

fixes $T :: 'a \Rightarrow 't::\{\text{linorder-topology, second-countable-topology}\}$

assumes $T: \text{stopping-time } F T$

and $M: \bigwedge t. \text{sets } (F t) \subseteq \text{sets } M \bigwedge t. \text{space } (F t) = \text{space } M$

shows $T \in M \rightarrow_M \text{borel}$

proof (*rule borel-measurableI-le*)

show $\{x \in \text{space } M. T x \leq t\} \in \text{sets } M$ **for** t

using *stopping-timeD[OF T] M* **by** (*auto simp: Measurable.pred-def*)

qed

lemma *stopping-time-const*: *stopping-time* $F (\lambda x. c)$

by (*auto simp: stopping-time-def*)

lemma *stopping-time-min*:

stopping-time $F T \implies \text{stopping-time } F S \implies \text{stopping-time } F (\lambda x. \min (T x) (S x))$

by (*auto simp: stopping-time-def min-le-iff-disj intro!: pred-intros-logic*)

lemma *stopping-time-max*:

stopping-time $F T \implies \text{stopping-time } F S \implies \text{stopping-time } F (\lambda x. \max (T x) (S x))$

by (*auto simp: stopping-time-def intro!: pred-intros-logic*)

31 Filtration

locale *filtration* =

fixes $\Omega :: 'a \text{ set}$ **and** $F :: 't::\{\text{linorder-topology, second-countable-topology}\} \Rightarrow 'a \text{ measure}$

assumes *space-F*: $\bigwedge i. \text{space } (F i) = \Omega$

assumes *sets-F-mono*: $\bigwedge i j. i \leq j \implies \text{sets } (F i) \subseteq \text{sets } (F j)$

begin

31.1 σ -algebra of a Stopping Time

definition *pre-sigma* :: $('a \Rightarrow 't) \Rightarrow 'a \text{ measure}$

where

pre-sigma $T = \text{sigma } \Omega \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$

lemma *space-pre-sigma*: $\text{space } (\text{pre-sigma } T) = \Omega$

unfolding *pre-sigma-def* **using** *sets.space-closed[of F -]*

by (*intro space-measure-of*) (*auto simp: space-F*)

lemma *measure-pre-sigma[simp]*: $\text{emeasure } (\text{pre-sigma } T) = (\lambda. 0)$

by (*simp add: pre-sigma-def emeasure-sigma*)

lemma *sigma-algebra-pre-sigma*:
assumes T : *stopping-time* F T
shows *sigma-algebra* Ω $\{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
unfolding *sigma-algebra-iff2*
proof (*intro sigma-algebra-iff2*[*THEN iffD2*] *conjI ballI allI impI CollectI*)
show $\{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\} \subseteq \text{Pow } \Omega$
using *sets.space-closed*[*of F -*] **by** (*auto simp: space-F*)
next
fix A t **assume** $A \in \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
then have $\{\omega \in \text{space } (F t). T \omega \leq t\} - \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)$
using T *stopping-timeD*[*measurable*] **by** *auto*
also have $\{\omega \in \text{space } (F t). T \omega \leq t\} - \{\omega \in A. T \omega \leq t\} = \{\omega \in \Omega - A. T \omega \leq t\}$
by (*auto simp: space-F*)
finally show $\{\omega \in \Omega - A. T \omega \leq t\} \in \text{sets } (F t)$.
next
fix $AA :: \text{nat} \Rightarrow 'a \text{ set}$ **and** t **assume** $\text{range } AA \subseteq \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
then have $(\bigcup i. \{\omega \in AA i. T \omega \leq t\}) \in \text{sets } (F t)$ **for** t
by *auto*
also have $(\bigcup i. \{\omega \in AA i. T \omega \leq t\}) = \{\omega \in \text{UNION UNIV } AA. T \omega \leq t\}$
by *auto*
finally show $\{\omega \in \text{UNION UNIV } AA. T \omega \leq t\} \in \text{sets } (F t)$.
qed *auto*

lemma *sets-pre-sigma*: *stopping-time* F $T \implies \text{sets } (\text{pre-sigma } T) = \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
unfolding *pre-sigma-def* **by** (*rule sigma-algebra.sets-measure-of-eq*[*OF sigma-algebra-pre-sigma*])

lemma *sets-pre-sigmaI*: *stopping-time* F $T \implies (\bigwedge t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)) \implies A \in \text{sets } (\text{pre-sigma } T)$
unfolding *sets-pre-sigma* **by** *auto*

lemma *pred-pre-sigmaI*:
assumes T : *stopping-time* F T
shows $(\bigwedge t. \text{Measurable.pred } (F t) (\lambda \omega. P \omega \wedge T \omega \leq t)) \implies \text{Measurable.pred } (\text{pre-sigma } T) P$
unfolding *pred-def space-F space-pre-sigma* **by** (*intro sets-pre-sigmaI*[*OF T*])
simp

lemma *sets-pre-sigmaD*: *stopping-time* F $T \implies A \in \text{sets } (\text{pre-sigma } T) \implies \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)$
unfolding *sets-pre-sigma* **by** *auto*

lemma *stopping-time-le-const*: *stopping-time* F $T \implies s \leq t \implies \text{Measurable.pred } (F t) (\lambda \omega. T \omega \leq s)$
using *stopping-timeD*[*of F T*] *sets-F-mono*[*of - t*] **by** (*auto simp: pred-def space-F*)

lemma *measurable-stopping-time-pre-sigma*:

assumes T : *stopping-time* F T **shows** $T \in \text{pre-sigma } T \rightarrow_M \text{borel}$
proof (*intro borel-measurableI-le sets-pre-sigmaI[OF T]*)
fix t t'
have $\{\omega \in \text{space } (F (\text{min } t' t)), T \omega \leq \text{min } t' t\} \in \text{sets } (F (\text{min } t' t))$
using T **unfolding** *pred-def[symmetric]* **by** (*rule stopping-timeD*)
also have $\dots \subseteq \text{sets } (F t)$
by (*rule sets-F-mono simp*)
finally show $\{\omega \in \{x \in \text{space } (\text{pre-sigma } T), T x \leq t'\}, T \omega \leq t\} \in \text{sets } (F t)$
by (*simp add: space-pre-sigma space-F*)
qed

lemma *mono-pre-sigma*:

assumes T : *stopping-time* F T **and** S : *stopping-time* F S
and le : $\bigwedge \omega. \omega \in \Omega \implies T \omega \leq S \omega$
shows $\text{sets } (\text{pre-sigma } T) \subseteq \text{sets } (\text{pre-sigma } S)$
unfolding *sets-pre-sigma[OF S]* *sets-pre-sigma[OF T]*
proof *safe*
interpret *sigma-algebra* Ω $\{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
using T **by** (*rule sigma-algebra-pre-sigma*)
fix A t **assume** $A: \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)$
then have $A \subseteq \Omega$
using *sets-into-space* **by** *auto*
from A **have** $\{\omega \in A. T \omega \leq t\} \cap \{\omega \in \text{space } (F t), S \omega \leq t\} \in \text{sets } (F t)$
using *stopping-timeD[OF S]* **by** (*auto simp: pred-def*)
also have $\{\omega \in A. T \omega \leq t\} \cap \{\omega \in \text{space } (F t), S \omega \leq t\} = \{\omega \in A. S \omega \leq t\}$
using $\langle A \subseteq \Omega \rangle$ *sets-into-space[of A]* le **by** (*auto simp: space-F intro: order-trans*)
finally show $\{\omega \in A. S \omega \leq t\} \in \text{sets } (F t)$
by *auto*
qed

lemma *stopping-time-less-const*:

assumes T : *stopping-time* F T **shows** *Measurable.pred* $(F t)$ $(\lambda \omega. T \omega < t)$
proof –
guess $D :: 't$ **set** **by** (*rule countable-dense-setE*)
note $D = \text{this}$
show *?thesis*
proof *cases*
assume $*$: $\forall t' < t. \exists t''. t' < t'' \wedge t'' < t$
{ **fix** t' **assume** $t' < t$
with $*$ **have** $\{t' < .. < t\} \neq \{\}$
by *fastforce*
with $D(2)[OF - \text{this}]$
have $\exists d \in D. t' < d \wedge d < t$
by *auto* **}**
note $** = \text{this}$

show *?thesis*
proof (*rule measurable-cong[THEN iffD2]*)
show $T \omega < t \iff (\exists r \in \{r \in D. r < t\}, T \omega \leq r)$ **for** ω

```

    by (auto dest: ** intro: less-imp-le)
    show Measurable.pred (F t) ( $\lambda w. \exists r \in \{r \in D. r < t\}. T w \leq r$ )
    by (intro measurable-pred-countable stopping-time-le-const[OF T] countable-Collect
D) auto
  qed
next
  assume  $\neg (\forall t' < t. \exists t''. t' < t'' \wedge t'' < t)$ 
  then obtain t' where t':  $t' < t \wedge t''. t'' < t \implies t'' \leq t'$ 
    by (auto simp: not-less[symmetric])
  show ?thesis
  proof (rule measurable-cong[THEN iffD2])
    show  $T \omega < t \longleftrightarrow T \omega \leq t'$  for  $\omega$ 
      using t' by auto
    show Measurable.pred (F t) ( $\lambda w. T w \leq t'$ )
      using  $t' < t$  by (intro stopping-time-le-const[OF T]) auto
  qed
qed
qed
qed

```

lemma *stopping-time-eq-const*: $stopping-time\ F\ T \implies Measurable.pred\ (F\ t)\ (\lambda\omega. T\ \omega = t)$

```

  unfolding eq-iff using stopping-time-less-const[of T t]
  by (intro pred-intros-logic stopping-time-le-const) (auto simp: not-less[symmetric]
)

```

lemma *stopping-time-less*:

```

  assumes T: stopping-time F T and S: stopping-time F S
  shows Measurable.pred (pre-sigma T) ( $\lambda\omega. T \omega < S \omega$ )
  proof (rule pred-pre-sigmaI[OF T])
    fix t
    obtain D :: 't set
      where [simp]: countable D and semidense-D:  $\bigwedge x\ y. x < y \implies (\exists b \in D. x \leq b \wedge b < y)$ 
      using countable-separating-set-linorder2 by auto
    show Measurable.pred (F t) ( $\lambda\omega. T \omega < S \omega \wedge T \omega \leq t$ )
    proof (rule measurable-cong[THEN iffD2])
      let ?f =  $\lambda\omega. if\ T\ \omega = t\ then\ \neg\ S\ \omega \leq t\ else\ \exists\ s \in \{s \in D. s \leq t\}. T\ \omega \leq s \wedge \neg (S\ \omega \leq s)$ 
      { fix  $\omega$  assume  $T \omega \leq t \wedge T \omega \neq t \wedge T \omega < S \omega$ 
        then have  $T \omega < \min\ t\ (S \omega)$ 
          by auto
        then obtain r where  $r \in D \wedge T \omega \leq r \wedge r < \min\ t\ (S \omega)$ 
          by (metis semidense-D)
        then have  $\exists s \in \{s \in D. s \leq t\}. T \omega \leq s \wedge s < S \omega$ 
          by auto }
      then show  $(T \omega < S \omega \wedge T \omega \leq t) = ?f\ \omega$  for  $\omega$ 
        by (auto simp: not-le)
    show Measurable.pred (F t) ?f
    by (intro pred-intros-logic measurable-If measurable-pred-countable countable-Collect

```



```

      stopping-time-le-const predE stopping-time-eq-const T S)
    auto
  qed
qed

end

lemma stopping-time-SUP-enat:
  fixes T :: nat  $\Rightarrow$  ('a  $\Rightarrow$  enat)
  shows  $(\bigwedge i. \text{stopping-time } F (T i)) \implies \text{stopping-time } F (SUP i. T i)$ 
  unfolding stopping-time-def SUP-apply SUP-le-iff by (auto intro!: pred-intros-countable)

lemma less-eSuc-iff:  $a < eSuc b \longleftrightarrow (a \leq b \wedge a \neq \infty)$ 
  by (cases a) auto

lemma stopping-time-Inf-enat:
  fixes F :: enat  $\Rightarrow$  'a measure
  assumes F: filtration  $\Omega$  F
  assumes P:  $\bigwedge i. \text{Measurable.pred } (F i) (P i)$ 
  shows stopping-time F  $(\lambda \omega. \text{Inf } \{i. P i \omega\})$ 
proof (rule stopping-timeI, cases)
  fix t :: enat assume t =  $\infty$  then show Measurable.pred (F t)  $(\lambda \omega. \text{Inf } \{i. P i \omega\} \leq t)$ 
    by auto
next
  fix t :: enat assume t  $\neq \infty$ 
  moreover
  { fix i  $\omega$  assume  $\text{Inf } \{i. P i \omega\} \leq t$ 
    with  $\langle t \neq \infty \rangle$  have  $(\exists i \leq t. P i \omega)$ 
    unfolding Inf-le-iff by (cases t) (auto elim!: allE[of - eSuc t] simp: less-eSuc-iff)
  }
  ultimately have *:  $\bigwedge \omega. \text{Inf } \{i. P i \omega\} \leq t \longleftrightarrow (\exists i \in \{..t\}. P i \omega)$ 
    by (auto intro!: Inf-lower2)
  show Measurable.pred (F t)  $(\lambda \omega. \text{Inf } \{i. P i \omega\} \leq t)$ 
    unfolding * using filtration.sets-F-mono[OF F, of - t] P
    by (intro pred-intros-countable-bounded) (auto simp: pred-def filtration.space-F[OF F])
qed

lemma stopping-time-Inf-nat:
  fixes F :: nat  $\Rightarrow$  'a measure
  assumes F: filtration  $\Omega$  F
  assumes P:  $\bigwedge i. \text{Measurable.pred } (F i) (P i)$  and wf:  $\bigwedge i \omega. \omega \in \Omega \implies \exists n. P n \omega$ 
  shows stopping-time F  $(\lambda \omega. \text{Inf } \{i. P i \omega\})$ 
  unfolding stopping-time-def
proof (intro allI, subst measurable-cong)
  fix t  $\omega$  assume  $\omega \in \text{space } (F t)$ 
  then have  $\omega \in \Omega$ 

```

```

using filtration.space-F[OF F] by auto
from wf[OF this] have  $((LEAST\ n.\ P\ n\ \omega) \leq t) = (\exists\ i \leq t.\ P\ i\ \omega)$ 
by (rule LeastI2-wellorder-ex) auto
then show  $(Inf\ \{i.\ P\ i\ \omega\} \leq t) = (\exists\ i \in \{..t\}.\ P\ i\ \omega)$ 
by (simp add: Inf-nat-def Bex-def)
next
fix t from P show Measurable.pred (F t)  $(\lambda w.\ \exists\ i \in \{..t\}.\ P\ i\ w)$ 
using filtration.sets-F-mono[OF F, of - t]
by (intro pred-intros-countable-bounded) (auto simp: pred-def filtration.space-F[OF
F])
qed

end

```

```

theory Probability
imports
  Central-Limit-Theorem
  Discrete-Topology
  PMF-Impl
  Projective-Limit
  Random-Permutations
  SPMF
  Stream-Space
  Tree-Space
  Conditional-Expectation
  Essential-Supremum
  Stopping-Time
begin

end

```