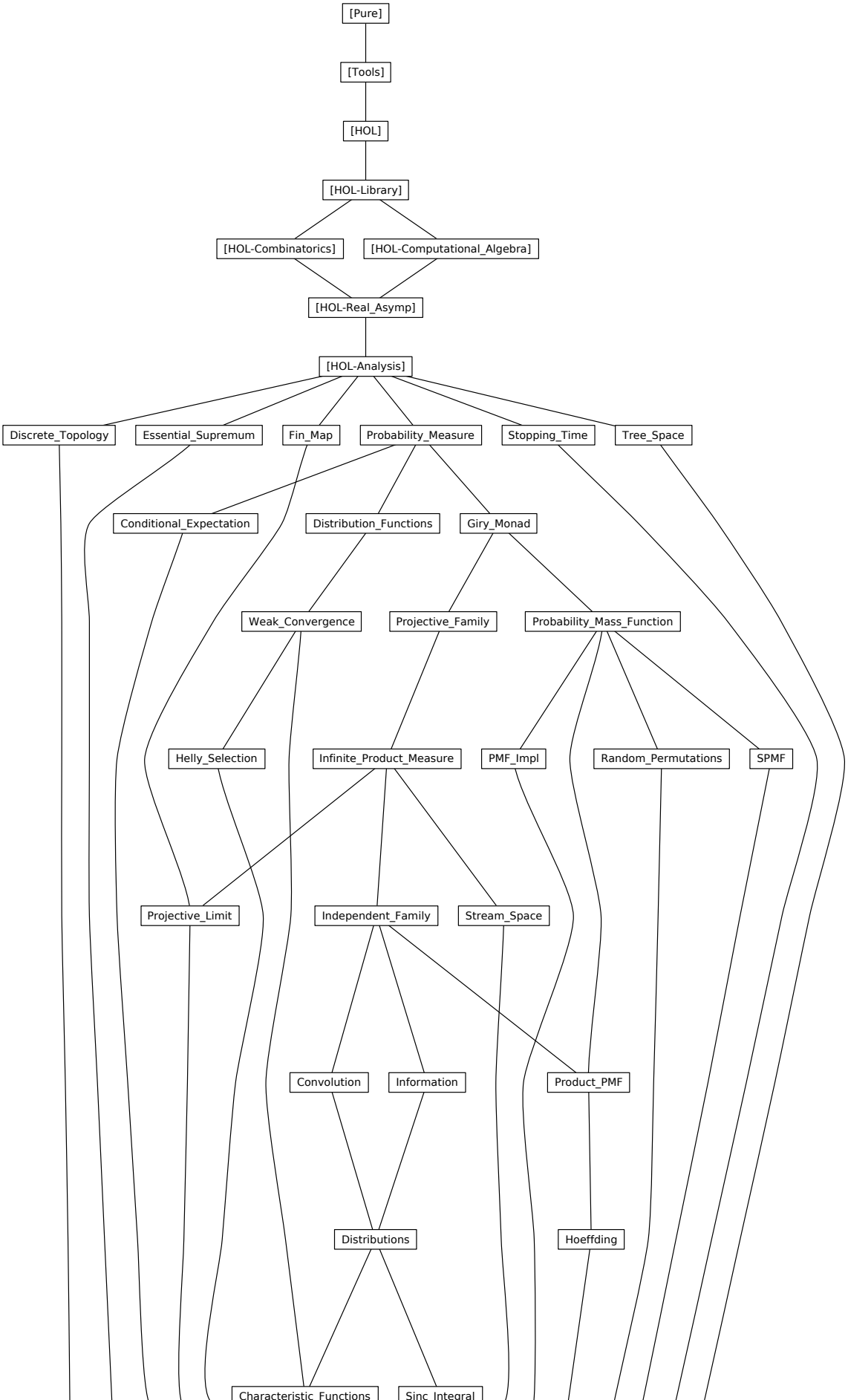
Measure and Probability Theory

March 13, 2025

Contents


[Pure]
[Tools]
[HOL]
[HOL-Library]
[HOL-Combinatorics]
[HOL-Computational_Algebra]
[HOL-Real_Asymp]
[HOL-Analysis]
Discrete_Topology
Essential_Supremum
Fin_Map
Probability_Measure
Stopping_Time
Tree_Space
Conditional_Expectation
Distribution_Functions
Giry_Monad
Weak_Convergence
Projective_Family
Probability_Mass_Function
Helly_Selection
Infinite_Product_Measure
PMF_Impl
Random_Permutations
SPMF
Projective_Limit
Independent_Family
Stream_Space
Convolution
Information
Product_PMF
Distributions
Hoeffding
Characteristic_Functions
Sinc_Integral

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*⟩

lemma *prob-space-imp-sigma-finite*: *prob-space M* ⟹ *sigma-finite-measure M*
⟨*proof*⟩

abbreviation (**in** *prob-space*) *events* ≡ *sets M*
abbreviation (**in** *prob-space*) *prob* ≡ *measure M*
abbreviation (**in** *prob-space*) *random-variable M′ X* ≡ *X* ∈ *measurable M M′*
abbreviation (**in** *prob-space*) *expectation* ≡ *integral*^{L} *M*
abbreviation (**in** *prob-space*) *variance X* ≡ *integral*^{L} *M* (λ*x*. (*X x* − *expectation X*)^2)

lemma (**in** *prob-space*) *finite-measure* [*simp*]: *finite-measure M*
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-space-distr*:
assumes *f*: *f* ∈ *measurable M M′* **shows** *prob-space* (*distr M M′ f*)
⟨*proof*⟩

lemma *prob-space-distrD*:
assumes *f*: *f* ∈ *measurable M N* **and** *M*: *prob-space* (*distr M N f*) **shows** *prob-space M*
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-space*: *prob* (*space M*) = *1*
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-le-1*[*simp*, *intro*]: *prob A* ≤ *1*
⟨*proof*⟩

lemma (**in** *prob-space*) *not-empty*: *space M* ≠ {}
⟨*proof*⟩

lemma (**in** *prob-space*) *emeasure-eq-1-AE*:
S ∈ *sets M* ⟹ *AE x in M*. *x* ∈ *S* ⟹ *emeasure M S* = *1*
⟨*proof*⟩

lemma (**in** *prob-space*) *emeasure-le-1*: $emeasure\ M\ S \leq 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *emeasure-ge-1-iff*: $emeasure\ M\ A \geq 1 \longleftrightarrow emeasure\ M\ A = 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-iff-emeasure-eq-1*:
assumes [*measurable*]: *Measurable.pred M P*
shows $(AE\ x\ in\ M.\ P\ x) \longleftrightarrow emeasure\ M\ \{x \in space\ M.\ P\ x\} = 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *measure-le-1*: $emeasure\ M\ X \leq 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *measure-ge-1-iff*: $measure\ M\ A \geq 1 \longleftrightarrow measure\ M\ A = 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-I-eq-1*:
assumes $emeasure\ M\ \{x \in space\ M.\ P\ x\} = 1$ $\{x \in space\ M.\ P\ x\} \in sets\ M$
shows $AE\ x\ in\ M.\ P\ x$
⟨*proof*⟩

lemma *prob-space-restrict-space*:
$S \in sets\ M \Longrightarrow emeasure\ M\ S = 1 \Longrightarrow prob\text{-}space\ (restrict\text{-}space\ M\ S)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-compl*:
assumes *A*: $A \in events$
shows $prob\ (space\ M - A) = 1 - prob\ A$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-in-set-eq-1*:
assumes *A*[*measurable*]: $A \in events$ **shows** $(AE\ x\ in\ M.\ x \in A) \longleftrightarrow prob\ A = 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-False*: $(AE\ x\ in\ M.\ False) \longleftrightarrow False$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-prob-1*:
assumes $prob\ A = 1$ **shows** $AE\ x\ in\ M.\ x \in A$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-const*[*simp*]: $(AE\ x\ in\ M.\ P) \longleftrightarrow P$
⟨*proof*⟩

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

⟨*proof*⟩

lemma (**in** *prob-space*) *AE-contr*:
assumes *ae*: *AE* ω *in* *M*. *P* ω *AE* ω *in* *M*. ¬ *P* ω
shows *False*
⟨*proof*⟩

lemma (**in** *prob-space*) *integral-ge-const*:
fixes *c* :: *real*
shows *integrable* *M* *f* ⟹ (*AE* *x* *in* *M*. *c* ≤ *f* *x*) ⟹ *c* ≤ (∫ *x*. *f* *x* ∂*M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *integral-le-const*:
fixes *c* :: *real*
shows *integrable* *M* *f* ⟹ (*AE* *x* *in* *M*. *f* *x* ≤ *c*) ⟹ (∫ *x*. *f* *x* ∂*M*) ≤ *c*
⟨*proof*⟩

lemma (**in** *prob-space*) *nn-integral-ge-const*:
(*AE* *x* *in* *M*. *c* ≤ *f* *x*) ⟹ *c* ≤ (∫⁺*x*. *f* *x* ∂*M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *expectation-less*:
fixes *X* :: - ⇒ *real*
assumes [*simp*]: *integrable* *M* *X*
assumes *gt*: *AE* *x* *in* *M*. *X* *x* < *b*
shows *expectation* *X* < *b*
⟨*proof*⟩

lemma (**in** *prob-space*) *expectation-greater*:
fixes *X* :: - ⇒ *real*
assumes [*simp*]: *integrable* *M* *X*
assumes *gt*: *AE* *x* *in* *M*. *a* < *X* *x*
shows *a* < *expectation* *X*
⟨*proof*⟩

lemma (**in** *prob-space*) *jensens-inequality*:
fixes *q* :: *real* ⇒ *real*
assumes *X*: *integrable* *M* *X* *AE* *x* *in* *M*. *X* *x* ∈ *I*
assumes *I*: *I* = {*a* <..< *b*} ∨ *I* = {*a* <..} ∨ *I* = {..< *b*} ∨ *I* = *UNIV*
assumes *q*: *integrable* *M* (λ*x*. *q* (*X* *x*)) *convex-on* *I* *q*
shows *q* (*expectation* *X*) ≤ *expectation* (λ*x*. *q* (*X* *x*))
⟨*proof*⟩

lemma (**in** *prob-space*) *finite-borel-measurable-integrable*:
assumes *f*∈ *borel-measurable* *M*
and *finite* (*f*‘(*space* *M*))
shows *integrable* *M* *f*
⟨*proof*⟩

1.1 Introduce binder for probability

syntax
-prob :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* (‹('𝒫'((/- *in* -./ -)'))›)
syntax-consts
-prob == *measure*
translations
$\mathcal{P}(x\ in\ M.\ P)$ => *CONST measure M* $\{x \in$ *CONST space M*. $P\}$

⟨*ML*⟩

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

syntax
-conditional-prob :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* (‹('𝒫'(- *in* -. - |/ -'))›)
syntax-consts
-conditional-prob == *cond-prob*
translations
$\mathcal{P}(x\ in\ M.\ P \mid Q)$ => *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** $S \subseteq \{x \in$ *space M*. *P x*$\}$ *S* ∈ *events prob S* = *1*
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-neg*: $\{x \in$ *space M*. *P x*$\}$ ∈ *events* ⟹ $\mathcal{P}(x\ in\ M.\ \neg\ P\ x) = 1 - \mathcal{P}(x\ in\ M.\ P\ x)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-eq-AE*:
(*AE x in M*. *P x* ⟷ *Q x*) ⟹ $\{x \in$ *space M*. *P x*$\}$ ∈ *events* ⟹ $\{x \in$ *space M*. *Q x*$\}$ ∈ *events* ⟹ $\mathcal{P}(x\ in\ M.\ P\ x) = \mathcal{P}(x\ in\ M.\ Q\ x)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-eq-0-AE*:
assumes *not*: *AE x in M*. ¬ *P x* **shows** $\mathcal{P}(x\ in\ M.\ P\ x) = 0$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-Collect-eq-0*:
$\{x \in$ *space M*. *P x*$\}$ ∈ *sets M* ⟹ $\mathcal{P}(x\ in\ M.\ P\ x) = 0$ ⟷ (*AE x in M*. ¬ *P x*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-Collect-eq-1*:
$\{x \in$ *space M*. *P x*$\}$ ∈ *sets M* ⟹ $\mathcal{P}(x\ in\ M.\ P\ x) = 1$ ⟷ (*AE x in M*. *P x*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-eq-0*:
A ∈ *sets M* ⟹ *prob A* = *0* ⟷ (*AE x in M*. *x* ∉ *A*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-eq-1*:
A ∈ *sets M* ⟹ *prob A* = *1* ⟷ (*AE x in M*. *x* ∈ *A*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-sums*:
assumes *P*: ⋀*n*. {*x*∈*space M*. *P n x*} ∈ *events*
assumes *Q*: {*x*∈*space M*. *Q x*} ∈ *events*
assumes *ae*: *AE x in M*. (∀ *n*. *P n x* ⟶ *Q x*) ∧ (*Q x* ⟶ (∃!*n*. *P n x*))
shows (λ*n*. *P*(*x in M*. *P n x*)) *sums P*(*x in M*. *Q x*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-sum*:
assumes [*simp*, *intro*]: *finite I*
assumes *P*: ⋀*n*. *n* ∈ *I* ⟹ {*x*∈*space M*. *P n x*} ∈ *events*
assumes *Q*: {*x*∈*space M*. *Q x*} ∈ *events*
assumes *ae*: *AE x in M*. (∀ *n*∈*I*. *P n x* ⟶ *Q x*) ∧ (*Q x* ⟶ (∃!*n*∈*I*. *P n x*))
shows *P*(*x in M*. *Q x*) = (∑ *n*∈*I*. *P*(*x in M*. *P n x*))
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-EX-countable*:
assumes *sets*: ⋀*i*. *i* ∈ *I* ⟹ {*x*∈*space M*. *P i x*} ∈ *sets M* **and** *I*: *countable I*
assumes *disj*: *AE x in M*. ∀ *i*∈*I*. ∀ *j*∈*I*. *P i x* ⟶ *P j x* ⟶ *i* = *j*
shows *P*(*x in M*. ∃ *i*∈*I*. *P i x*) = (∫ ⁺*i*. *P*(*x in M*. *P i x*) ∂*count-space I*)
⟨*proof*⟩

lemma (**in** *prob-space*) *cond-prob-eq-AE*:
assumes *P*: *AE x in M*. *Q x* ⟶ *P x* ⟷ *P*′ *x* {*x*∈*space M*. *P x*} ∈ *events* {*x*∈*space M*. *P*′ *x*} ∈ *events*
assumes *Q*: *AE x in M*. *Q x* ⟷ *Q*′ *x* {*x*∈*space M*. *Q x*} ∈ *events* {*x*∈*space M*. *Q*′ *x*} ∈ *events*
shows *cond-prob M P Q* = *cond-prob M P*′ *Q*′
⟨*proof*⟩

lemma (**in** *prob-space*) *joint-distribution-Times-le-fst*:
random-variable MX X ⟹ *random-variable MY Y* ⟹ *A* ∈ *sets MX* ⟹ *B* ∈ *sets MY*
⟹ *emeasure* (*distr M* (*MX* ⨂$_M$ *MY*) (λ*x*. (*X x*, *Y x*))) (*A* × *B*) ≤ *emeasure* (*distr M MX X*) *A*
⟨*proof*⟩

lemma (**in** *prob-space*) *joint-distribution-Times-le-snd*:
random-variable MX X ⟹ *random-variable MY Y* ⟹ *A* ∈ *sets MX* ⟹ *B* ∈ *sets MY*
⟹ *emeasure* (*distr M* (*MX* ⨂$_M$ *MY*) (λ*x*. (*X x*, *Y x*))) (*A* × *B*) ≤ *emeasure* (*distr M MY Y*) *B*
⟨*proof*⟩

lemma (**in** *prob-space*) *variance-eq*:
fixes $X :: 'a \Rightarrow real$
assumes [*simp*]: *integrable M X*
assumes [*simp*]: *integrable M* $(\lambda x.\ (X\ x)^2)$
shows *variance X* = *expectation* $(\lambda x.\ (X\ x)^2)$ − (*expectation X*)^2
⟨*proof*⟩

lemma (**in** *prob-space*) *variance-positive*: *0* ≤ *variance* $(X::'a \Rightarrow real)$
⟨*proof*⟩

lemma (**in** *prob-space*) *variance-mean-zero*:
expectation X = *0* ⟹ *variance X* = *expectation* $(\lambda x.\ (X\ x)\hat{}2)$
⟨*proof*⟩

theorem (**in** *prob-space*) *Chebyshev-inequality*:
assumes [*measurable*]: *random-variable borel f*
assumes *integrable M* $(\lambda x.\ f\ x\ \hat{}\ 2)$
defines $\mu \equiv$ *expectation f*
assumes *a* > *0*
shows *prob* $\{x \in$ *space M*. $|f\ x - \mu| \geq a\} \leq$ *variance f* / a^2
unfolding μ-*def*
proof (*rule second-moment-method*)
have *integrable*: *integrable M f*
using *assms* **by** (*blast dest*: *square-integrable-imp-integrable*)
show *integrable M* $(\lambda x.\ (f\ x - expectation\ f)^2)$
using *assms integrable* **unfolding** *power2-eq-square ring-distribs*
by (*intro Bochner-Integration.integrable-diff*) *auto*
qed (*use assms* **in** *auto*)

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

sublocale *pair-prob-space* ⊆ *P?*: *prob-space M1* $\bigotimes_M$ *M2*
⟨*proof*⟩

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

sublocale *product-prob-space* ⊆ *M?*: *prob-space M i* **for** *i*
⟨*proof*⟩

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

sublocale *finite-product-prob-space* ⊆ *prob-space* $\Pi_M\ i \in I.\ M\ i$
⟨*proof*⟩

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

assumes X: $\bigwedge i.\ i \in I \Longrightarrow X\ i \in sets\ (M\ i)$
shows $prob\ (\Pi_E\ i{\in}I.\ X\ i) = (\prod i{\in}I.\ M.prob\ i\ (X\ i))$
⟨*proof*⟩

lemma *product-prob-spaceI*:
assumes $\bigwedge i.$ *prob-space* $(M\ i)$
shows *product-prob-space M*
⟨*proof*⟩

1.2 Distributions

definition *distributed* :: *'a measure* ⇒ *'b measure* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *ennreal*) ⇒ *bool*
where
distributed M N X f ⟷
distr M N X = *density N f* ∧ *f* ∈ *borel-measurable N* ∧ *X* ∈ *measurable M N*

lemma
assumes *distributed M N X f*
shows *distributed-distr-eq-density*: *distr M N X* = *density N f*
and *distributed-measurable*: *X* ∈ *measurable M N*
and *distributed-borel-measurable*: *f* ∈ *borel-measurable N*
⟨*proof*⟩

lemma
assumes *D*: *distributed M N X f*
shows *distributed-measurable'*[*measurable-dest*]:
g ∈ *measurable L M* ⟹ (λ*x*. *X* (*g x*)) ∈ *measurable L N*
and *distributed-borel-measurable'*[*measurable-dest*]:
h ∈ *measurable L N* ⟹ (λ*x*. *f* (*h x*)) ∈ *borel-measurable L*
⟨*proof*⟩

lemma *distributed-real-measurable*:
(⋀*x*. *x* ∈ *space N* ⟹ 0 ≤ *f x*) ⟹ *distributed M N X* (λ*x*. *ennreal* (*f x*)) ⟹ *f* ∈ *borel-measurable N*
⟨*proof*⟩

lemma *distributed-real-measurable'*:
(⋀*x*. *x* ∈ *space N* ⟹ 0 ≤ *f x*) ⟹ *distributed M N X* (λ*x*. *ennreal* (*f x*)) ⟹
h ∈ *measurable L N* ⟹ (λ*x*. *f* (*h x*)) ∈ *borel-measurable L*
⟨*proof*⟩

lemma *joint-distributed-measurable1*:
distributed M (*S* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*)) *f* ⟹ *h1* ∈ *measurable N M* ⟹ (λ*x*. *X* (*h1 x*)) ∈ *measurable N S*
⟨*proof*⟩

lemma *joint-distributed-measurable2*:
distributed M (*S* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*)) *f* ⟹ *h2* ∈ *measurable N M* ⟹ (λ*x*.

$Y\ (h2\ x)) \in$ *measurable* $N\ T$
⟨*proof*⟩

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

lemma *distributed-cong-density*:
$(AE\ x$ *in* $N.\ f\ x = g\ x) \Longrightarrow g \in$ *borel-measurable* $N \Longrightarrow f \in$ *borel-measurable* N
$\Longrightarrow$
distributed $M\ N\ X\ f \longleftrightarrow$ *distributed* $M\ N\ X\ g$
⟨*proof*⟩

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*⟩

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*⟩

lemma *subdensity-real*:
fixes $g :: 'a \Rightarrow$ *real* **and** $f :: 'b \Rightarrow$ *real*
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* $P.\ 0 \leq g\ (T\ x)) \Longrightarrow (AE\ x$ *in* $P.\ 0 \leq f\ x) \Longrightarrow AE\ x$ *in* $P.\ g\ (T\ x) = 0 \longrightarrow f\ x = 0$
⟨*proof*⟩

lemma *distributed-emeasure*:
distributed $M\ N\ X\ f \Longrightarrow A \in$ *sets* $N \Longrightarrow$ *emeasure* $M\ (X -` A \cap$ *space* $M) = (\int^+ x.\ f\ x * indicator\ A\ x\ \partial N)$
⟨*proof*⟩

lemma *distributed-nn-integral*:
distributed $M\ N\ X\ f \Longrightarrow g \in$ *borel-measurable* $N \Longrightarrow (\int^+ x.\ f\ x * g\ x\ \partial N) = (\int^+ x.\ g\ (X\ x)\ \partial M)$
⟨*proof*⟩

lemma *distributed-integral*:
distributed $M\ N\ X\ f \Longrightarrow g \in$ *borel-measurable* $N \Longrightarrow (\bigwedge x.\ x \in$ *space* $N \Longrightarrow 0 \leq$

$f\ x) \Longrightarrow$
$(\int x.\ f\ x * g\ x\ \partial N) = (\int x.\ g\ (X\ x)\ \partial M)$
⟨*proof*⟩

lemma *distributed-transform-integral*:
assumes *Px*: *distributed M N X Px* $\bigwedge x.\ x \in$ *space N* $\Longrightarrow 0 \leq Px\ x$
assumes *distributed M P Y Py* $\bigwedge x.\ x \in$ *space P* $\Longrightarrow 0 \leq Py\ x$
assumes *Y*: $Y = T \circ X$ **and** *T*: $T \in$ *measurable N P* **and** *f*: $f \in$ *borel-measurable P*
shows $(\int x.\ Py\ x * f\ x\ \partial P) = (\int x.\ Px\ x * f\ (T\ x)\ \partial N)$
⟨*proof*⟩

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*⟩

lemma (**in** *prob-space*) *distributed-jointI*:
assumes *sigma-finite-measure S sigma-finite-measure T*
assumes *X*[*measurable*]: $X \in$ *measurable M S* **and** *Y*[*measurable*]: $Y \in$ *measurable M T*
assumes [*measurable*]: $f \in$ *borel-measurable* $(S \bigotimes_M T)$ **and** *f*: *AE x in* $S \bigotimes_M T$. $0 \leq f\ x$
assumes *eq*: $\bigwedge A\ B.\ A \in$ *sets S* $\Longrightarrow B \in$ *sets T* $\Longrightarrow$
emeasure M $\{x \in$ *space M*. $X\ x \in A \wedge Y\ x \in B\} = (\int^+ x.\ (\int^+ y.\ f\ (x,\ y) *$ *indicator B y* $\partial T) *$ *indicator A x* $\partial S)$
shows *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))\ f$
⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-swap*:
assumes *sigma-finite-measure S sigma-finite-measure T*
assumes *Pxy*: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
shows *distributed M* $(T \bigotimes_M S)$ $(\lambda x.\ (Y\ x,\ X\ x))$ $(\lambda(x,\ y).\ Pxy\ (y,\ x))$
⟨*proof*⟩

lemma (**in** *prob-space*) *distr-marginal1*:
assumes *sigma-finite-measure S sigma-finite-measure T*
assumes *Pxy*: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
defines $Px \equiv \lambda x.\ (\int^+ z.\ Pxy\ (x,\ z)\ \partial T)$
shows *distributed M S X Px*
⟨*proof*⟩

lemma (**in** *prob-space*) *distr-marginal2*:
assumes *S*: *sigma-finite-measure S* **and** *T*: *sigma-finite-measure T*
assumes *Pxy*: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
shows *distributed M T Y* $(\lambda y.\ (\int^+ x.\ Pxy\ (x,\ y)\ \partial S))$
⟨*proof*⟩

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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
shows *AE x in S. Px x* $= (\int^+ y.\ Pxy\ (x,\ y)\ \partial T)$
⟨*proof*⟩

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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
shows *AE y in T. Py y* $= (\int^+ x.\ Pxy\ (x,\ y)\ \partial S)$
⟨*proof*⟩

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*: *distr M S X* $\bigotimes_M$ *distr M T Y* $=$ *distr M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$
shows *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ $(\lambda(x,\ y).\ Px\ x * Py\ y)$
⟨*proof*⟩

lemma *distributed-integrable*:
distributed M N X f $\Longrightarrow$ *g* $\in$ *borel-measurable N* $\Longrightarrow$ $(\bigwedge x.\ x \in space\ N \Longrightarrow 0 \leq f\ x) \Longrightarrow$
integrable N $(\lambda x.\ f\ x * g\ x) \longleftrightarrow$ *integrable M* $(\lambda x.\ g\ (X\ x))$
⟨*proof*⟩

lemma *distributed-transform-integrable*:
assumes Px: *distributed M N X Px* $\bigwedge x.\ x \in space\ N \Longrightarrow 0 \leq Px\ x$
assumes *distributed M P Y Py* $\bigwedge x.\ x \in space\ P \Longrightarrow 0 \leq Py\ x$
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*⟩

lemma *distributed-integrable-var*:
fixes $X :: 'a \Rightarrow real$
shows *distributed M lborel X* $(\lambda x.\ ennreal\ (f\ x)) \Longrightarrow (\bigwedge x.\ 0 \leq f\ x) \Longrightarrow$
integrable lborel $(\lambda x.\ f\ x * x) \Longrightarrow$ *integrable M X*
⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-variance*:
fixes f::*real* $\Rightarrow$ *real*
assumes D: *distributed M lborel X f* **and** [*simp*]: $\bigwedge x.\ 0 \leq f\ x$
shows *variance X* $= (\int x.\ x^2 * f\ (x + expectation\ X)\ \partial lborel)$

⟨*proof*⟩

lemma (**in** *prob-space*) *variance-affine*:
fixes f::*real* ⇒ *real*
assumes [*arith*]: $b \neq 0$
assumes D[*intro*]: *distributed M lborel X f*
assumes [*simp*]: *prob-space* (*density lborel f*)
assumes I[*simp*]: *integrable M X*
assumes *I2*[*simp*]: *integrable M* ($\lambda x.$ $(X\ x)^2$)
shows *variance* ($\lambda x.$ $a + b * X\ x$) = b^2 * *variance X*
⟨*proof*⟩

definition
simple-distributed M X f ⟷
($\forall x.$ $0 \leq f\ x$) ∧
distributed M (*count-space* (*X*‘*space M*)) *X* ($\lambda x.$ *ennreal* (*f x*)) ∧
finite (*X*‘*space M*)

lemma *simple-distributed-nonneg*[*dest*]: *simple-distributed M X f* ⟹ $0 \leq f\ x$
⟨*proof*⟩

lemma *simple-distributed*:
simple-distributed M X Px ⟹ *distributed M* (*count-space* (*X*‘*space M*)) *X Px*
⟨*proof*⟩

lemma *simple-distributed-finite*[*dest*]: *simple-distributed M X P* ⟹ *finite* (*X*‘*space M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-simple-function-superset*:
assumes *X*: *simple-function M X* ⋀$x.$ $x \in X$ ‘ *space M* ⟹ *P x* = *measure M* (X −‘ {x} ∩ *space M*)
assumes *A*: *X*‘*space M* ⊆ *A finite A*
defines *S* ≡ *count-space A* **and** *P′* ≡ ($\lambda x.$ *if x* ∈ *X*‘*space M then P x else* 0)
shows *distributed M S X P′*
⟨*proof*⟩

lemma (**in** *prob-space*) *simple-distributedI*:
assumes *X*: *simple-function M X*
⋀$x.$ $0 \leq P\ x$
⋀$x.$ $x \in X$ ‘ *space M* ⟹ *P x* = *measure M* (X −‘ {x} ∩ *space M*)
shows *simple-distributed M X P*
⟨*proof*⟩

lemma *simple-distributed-joint-finite*:
assumes *X*: *simple-distributed M* ($\lambda x.$ ($X\ x$, $Y\ x$)) *Px*
shows *finite* (*X* ‘ *space M*) *finite* (*Y* ‘ *space M*)
⟨*proof*⟩

lemma *simple-distributed-joint2-finite*:
assumes *X*: *simple-distributed M* (λ*x*. (*X x*, *Y x*, *Z x*)) *Px*
shows *finite* (*X* ' *space M*) *finite* (*Y* ' *space M*) *finite* (*Z* ' *space M*)
⟨*proof*⟩

lemma *simple-distributed-simple-function*:
simple-distributed M X Px ⟹ *simple-function M X*
⟨*proof*⟩

lemma *simple-distributed-measure*:
simple-distributed M X P ⟹ *a* ∈ *X*'*space M* ⟹ *P a* = *measure M* (*X* −' {*a*} ∩ *space M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *simple-distributed-joint*:
assumes *X*: *simple-distributed M* (λ*x*. (*X x*, *Y x*)) *Px*
defines *S* ≡ *count-space* (*X*'*space M*) ⨂$_M$ *count-space* (*Y*'*space M*)
defines *P* ≡ (λ*x*. *if x* ∈ (λ*x*. (*X x*, *Y x*))'*space M then Px x else 0*)
shows *distributed M S* (λ*x*. (*X x*, *Y x*)) *P*
⟨*proof*⟩

lemma (**in** *prob-space*) *simple-distributed-joint2*:
assumes *X*: *simple-distributed M* (λ*x*. (*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* ≡ (λ*x*. *if x* ∈ (λ*x*. (*X x*, *Y x*, *Z x*))'*space M then Px x else 0*)
shows *distributed M S* (λ*x*. (*X x*, *Y x*, *Z x*)) *P*
⟨*proof*⟩

lemma (**in** *prob-space*) *simple-distributed-sum-space*:
assumes *X*: *simple-distributed M X f*
shows *sum f* (*X*'*space M*) = *1*
⟨*proof*⟩

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* (λ*x*. (*X x*, *Y x*)) *Pxy*
assumes *y*: *y* ∈ *Y*'*space M*
shows *Py y* = (∑ *x*∈*X*'*space M*. *if* (*x*, *y*) ∈ (λ*x*. (*X x*, *Y x*)) ' *space M then Pxy* (*x*, *y*) *else 0*)
⟨*proof*⟩

lemma *distributedI-real*:
fixes *f* :: '*a* ⇒ *real*
assumes *gen*: *sets M1* = *sigma-sets* (*space M1*) *E* **and** *Int-stable E*
and *A*: *range A* ⊆ *E* (⋃ *i*::*nat*. *A i*) = *space M1* ⋀*i*. *emeasure* (*distr M M1 X*) (*A i*) ≠ ∞
and *X*: *X* ∈ *measurable M M1*

and *f*: $f \in$ *borel-measurable M1 AE x in M1*. $0 \leq f\ x$
and *eq*: $\bigwedge A$. $A \in E \Longrightarrow$ *emeasure M* $(X -` A \cap$ *space M*$) = (\int^+ x.\ f\ x * indicator\ A\ x\ \partial M1)$
shows *distributed M M1 X f*
⟨*proof*⟩

lemma *distributedI-borel-atMost*:
fixes $f :: real \Rightarrow real$
assumes [*measurable*]: $X \in$ *borel-measurable M*
and [*measurable*]: $f \in$ *borel-measurable borel* **and** *f*[*simp*]: *AE x in lborel*. $0 \leq f\ x$
and *g-eq*: $\bigwedge a$. $(\int^+ x.\ f\ x * indicator\ \{..a\}\ x\ \partial lborel) = ennreal\ (g\ a)$
and *M-eq*: $\bigwedge a$. *emeasure M* $\{x \in$ *space M*. $X\ x \leq a\} = ennreal\ (g\ a)$
shows *distributed M lborel X f*
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distributed-params*:
assumes *X*: *distributed M MX X* $(\lambda x.$ *indicator A x / measure MX A*$)$
shows $A \in$ *sets MX measure MX A* $\neq 0$
⟨*proof*⟩

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*⟩

lemma *prob-space-uniform-count-measure*: *finite* $A \Longrightarrow A \neq \{\} \Longrightarrow$ *prob-space* (*uniform-count-measure A*)
⟨*proof*⟩

lemma (**in** *prob-space*) *measure-uniform-measure-eq-cond-prob*:
assumes [*measurable*]: *Measurable.pred M P Measurable.pred M Q*
shows $\mathcal{P}(x$ *in uniform-measure M* $\{x \in$ *space M*. *Q x*$\}$. *P x*$) = \mathcal{P}(x$ *in M*. *P x* | *Q x*$)$
⟨*proof*⟩

lemma *prob-space-point-measure*:
finite $S \Longrightarrow (\bigwedge s.\ s \in S \Longrightarrow 0 \leq p\ s) \Longrightarrow (\sum s \in S.\ p\ s) = 1 \Longrightarrow$ *prob-space* (*point-measure S p*)
⟨*proof*⟩

lemma (**in** *prob-space*) *distr-pair-fst*: *distr* $(N \bigotimes_M M)\ N\ fst = N$
⟨*proof*⟩

lemma (**in** *product-prob-space*) *distr-reorder*:
assumes *inj-on t J* $t \in J \to K$ *finite K*
shows *distr* $(PiM\ K\ M)\ (Pi_M\ J\ (\lambda x.\ M\ (t\ x)))\ (\lambda \omega.\ \lambda n \in J.\ \omega\ (t\ n)) = PiM\ J\ (\lambda x.\ M\ (t\ x))$
⟨*proof*⟩

lemma (**in** *product-prob-space*) *distr-restrict*:
$J \subseteq K \Longrightarrow$ *finite* $K \Longrightarrow (\Pi_M\ i{\in}J.\ M\ i) = distr\ (\Pi_M\ i{\in}K.\ M\ i)\ (\Pi_M\ i{\in}J.\ M\ i)\ (\lambda f.\ restrict\ f\ J)$
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-prod-emb*[*simp*]:
assumes *L*: $J \subseteq L$ *finite* L **and** *X*: $X \in sets\ (Pi_M\ J\ M)$
shows $emeasure\ (Pi_M\ L\ M)\ (prod\text{-}emb\ L\ M\ J\ X) = emeasure\ (Pi_M\ J\ M)\ X$
⟨*proof*⟩

lemma *emeasure-distr-restrict*:
assumes $I \subseteq K$ **and** Q[*measurable-cong*]: $sets\ Q = sets\ (PiM\ K\ M)$ **and** A[*measurable*]: $A \in sets\ (PiM\ I\ M)$
shows $emeasure\ (distr\ Q\ (PiM\ I\ M)\ (\lambda\omega.\ restrict\ \omega\ I))\ A = emeasure\ Q\ (prod\text{-}emb\ K\ M\ I\ A)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-space-completion*: *prob-space* (*completion* M)
⟨*proof*⟩

lemma *distr-PiM-finite-prob-space*:
assumes *fin*: *finite* I
assumes *product-prob-space* M
assumes *product-prob-space* M'
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow f \in measurable\ (M\ i)\ (M'\ i)$
shows $distr\ (PiM\ I\ M)\ (PiM\ I\ M')\ (compose\ I\ f) = PiM\ I\ (\lambda i.\ distr\ (M\ i)\ (M'\ i)\ f)$
⟨*proof*⟩

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*: $(\bigcup n.\ \{-real\ n <..\ real\ n\}) = UNIV$
⟨*proof*⟩

2.1 Properties of cdf's

definition
cdf :: *real measure* ⇒ *real* ⇒ *real*
where
cdf M ≡ λ*x*. *measure M* {..*x*}

lemma *cdf-def2*: *cdf M x* = *measure M* {..*x*}
⟨*proof*⟩

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* ∈ *sets borel* ⟹ *a* ∈ *sets M*
⟨*proof*⟩

lemma *cdf-diff-eq*:
assumes *x* < *y*
shows *cdf M y* − *cdf M x* = *measure M* {*x*<..*y*}
⟨*proof*⟩

lemma *cdf-nondecreasing*: *x* ≤ *y* ⟹ *cdf M x* ≤ *cdf M y*
⟨*proof*⟩

lemma *borel-UNIV*: *space M* = *UNIV*
⟨*proof*⟩

lemma *cdf-nonneg*: *cdf M x* ≥ *0*
⟨*proof*⟩

lemma *cdf-bounded*: *cdf M x* ≤ *measure M* (*space M*)
⟨*proof*⟩

lemma *cdf-lim-infty*:
(λ*i*. *cdf M* (*real i*)) ⟶ *measure M* (*space M*))
⟨*proof*⟩

lemma *cdf-lim-at-top*: (*cdf M* ⟶ *measure M* (*space M*)) *at-top*
⟨*proof*⟩

lemma *cdf-lim-neg-infty*: ((λ*i*. *cdf M* (− *real i*)) ⟶ *0*)
⟨*proof*⟩

lemma *cdf-lim-at-bot*: (*cdf M* ⟶ *0*) *at-bot*
⟨*proof*⟩

lemma *cdf-is-right-cont*: *continuous* (*at-right a*) (*cdf M*)
⟨*proof*⟩

lemma *cdf-at-left*: $(cdf\ M \longrightarrow measure\ M\ \{..<a\})\ (at\text{-}left\ a)$
⟨*proof*⟩

lemma *isCont-cdf*: $isCont\ (cdf\ M)\ x \longleftrightarrow measure\ M\ \{x\} = 0$
⟨*proof*⟩

lemma *countable-atoms*: $countable\ \{x.\ measure\ M\ \{x\} > 0\}$
⟨*proof*⟩

end

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

lemma *finite-borel-measure-M*: *finite-borel-measure M*
⟨*proof*⟩

sublocale *finite-borel-measure M*
⟨*proof*⟩

lemma *space-eq-univ* [*simp*]: *space M* = *UNIV*
⟨*proof*⟩

lemma *cdf-bounded-prob*: $\bigwedge x.\ cdf\ M\ x \leq 1$
⟨*proof*⟩

lemma *cdf-lim-infty-prob*: $(\lambda i.\ cdf\ M\ (real\ i)) \longrightarrow 1$
⟨*proof*⟩

lemma *cdf-lim-at-top-prob*: $(cdf\ M \longrightarrow 1)\ at\text{-}top$
⟨*proof*⟩

lemma *measurable-finite-borel* [*simp*]:
$f \in borel\text{-}measurable\ borel \Longrightarrow f \in borel\text{-}measurable\ M$
⟨*proof*⟩

end

lemma (**in** *prob-space*) *real-distribution-distr* [*intro*, *simp*]:
$random\text{-}variable\ borel\ X \Longrightarrow real\text{-}distribution\ (distr\ M\ borel\ X)$
⟨*proof*⟩

2.2 Uniqueness

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

lemma *cdf-unique'*:
fixes *M1 M2*
assumes *finite-borel-measure M1* **and** *finite-borel-measure M2*
assumes *cdf M1 = cdf M2*
shows *M1 = M2*
⟨*proof*⟩

lemma *cdf-unique*:
real-distribution M1 ⟹ real-distribution M2 ⟹ cdf M1 = cdf M2 ⟹ M1 = M2
⟨*proof*⟩

lemma
fixes *F* :: *real ⇒ real*
assumes *nondecF* : ⋀ *x y*. *x ≤ y ⟹ F x ≤ F y*
and *right-cont-F* : ⋀*a*. *continuous* (*at-right a*) *F*
and *lim-F-at-bot* : (*F* ⟶ *0*) *at-bot*
and *lim-F-at-top* : (*F* ⟶ *m*) *at-top*
and *m*: *0 ≤ m*
shows *interval-measure-UNIV*: *emeasure* (*interval-measure F*) *UNIV = m*
and *finite-borel-measure-interval-measure*: *finite-borel-measure* (*interval-measure F*)
⟨*proof*⟩

lemma *real-distribution-interval-measure*:
fixes *F* :: *real ⇒ real*
assumes *nondecF* : ⋀ *x y*. *x ≤ y ⟹ F x ≤ F y* **and**
right-cont-F : ⋀*a*. *continuous* (*at-right a*) *F* **and**
lim-F-at-bot : (*F* ⟶ *0*) *at-bot* **and**
lim-F-at-top : (*F* ⟶ *1*) *at-top*
shows *real-distribution* (*interval-measure F*)
⟨*proof*⟩

lemma
fixes *F* :: *real ⇒ real*
assumes *nondecF* : ⋀ *x y*. *x ≤ y ⟹ F x ≤ F y* **and**
right-cont-F : ⋀*a*. *continuous* (*at-right a*) *F* **and**
lim-F-at-bot : (*F* ⟶ *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*
⟨*proof*⟩

lemma *cdf-interval-measure*:
(⋀ *x y*. *x ≤ y ⟹ F x ≤ F y*) ⟹ (⋀*a*. *continuous* (*at-right a*) *F*) ⟹ (*F* ⟶ *0*) *at-bot ⟹ cdf* (*interval-measure F*) = *F*
⟨*proof*⟩

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

5 Weak Convergence of Distributions

definition
weak-conv-m :: $(nat \Rightarrow real\ measure) \Rightarrow real\ measure \Rightarrow bool$
where
$weak\text{-}conv\text{-}m\ M\text{-}seq\ M \equiv weak\text{-}conv\ (\lambda n.\ cdf\ (M\text{-}seq\ n))\ (cdf\ M)$

6 Skorohod's theorem

locale *right-continuous-mono* =
fixes $f :: real \Rightarrow real$ **and** $a\ b :: real$
assumes *cont*: $\bigwedge x.\ continuous\ (at\text{-}right\ x)\ f$
assumes *mono*: $mono\ f$
assumes *bot*: $(f \longrightarrow a)\ at\text{-}bot$
assumes *top*: $(f \longrightarrow b)\ at\text{-}top$
begin

abbreviation $I :: real \Rightarrow real$ **where**
$I\ \omega \equiv Inf\ \{x.\ \omega \leq f\ x\}$

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

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

lemma *mono-I*: $mono\text{-}on\ \{a <..< b\}\ I$
⟨*proof*⟩

end

locale *cdf-distribution = real-distribution*
begin

abbreviation $C \equiv cdf\ M$

sublocale *right-continuous-mono C 0 1*
⟨*proof*⟩

lemma *measurable-C*[*measurable*]: $C \in borel\text{-}measurable\ borel$
⟨*proof*⟩

lemma *measurable-CI*[*measurable*]: $I \in borel\text{-}measurable\ (restrict\text{-}space\ borel\ \{0<..<1\})$
⟨*proof*⟩

lemma *emeasure-distr-I*: $emeasure\ (distr\ (restrict\text{-}space\ lborel\ \{0<..<1::real\})\ borel\ I)\ UNIV = 1$
⟨*proof*⟩

lemma *distr-I-eq-M*: $distr\ (restrict\text{-}space\ lborel\ \{0<..<1::real\})\ borel\ I = M$ (**is** *?I = -*)
⟨*proof*⟩

end

context
fixes $\mu :: nat \Rightarrow real\ measure$
and $M :: real\ measure$
assumes μ: $\bigwedge n.\ real\text{-}distribution\ (\mu\ n)$
assumes M: $real\text{-}distribution\ M$
assumes $\mu\text{-}to\text{-}M$: $weak\text{-}conv\text{-}m\ \mu\ M$
begin

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

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

theorem *weak-conv-imp-bdd-ae-continuous-conv*:
fixes
$f :: real \Rightarrow 'a::\{banach, second\text{-}countable\text{-}topology\}$

assumes
discont-null: $M\ (\{x.\ \neg\ isCont\ f\ x\}) = 0$ **and**
f-bdd: $\bigwedge x.\ norm\ (f\ x) \leq B$ **and**
[*measurable*]: $f \in borel\text{-}measurable\ borel$
shows
$(\lambda\ n.\ integral^L\ (\mu\ n)\ f) \longrightarrow integral^L\ M\ f$
⟨*proof*⟩

theorem *weak-conv-imp-integral-bdd-continuous-conv*:
fixes f :: *real* ⇒ $'a$::{*banach*, *second-countable-topology*}
assumes
$\bigwedge x.\ isCont\ f\ x$ **and**
$\bigwedge x.\ norm\ (f\ x) \leq B$
shows
$(\lambda\ n.\ integral^L\ (\mu\ n)\ f) \longrightarrow integral^L\ M\ f$
⟨*proof*⟩

theorem *weak-conv-imp-continuity-set-conv*:
fixes f :: *real* ⇒ *real*
assumes [*measurable*]: $A \in sets\ borel$ **and** $M\ (frontier\ A) = 0$
shows $(\lambda n.\ measure\ (\mu\ n)\ A) \longrightarrow measure\ M\ A$
⟨*proof*⟩

end

definition
cts-step :: *real* ⇒ *real* ⇒ *real* ⇒ *real*
where
$cts\text{-}step\ a\ b\ x \equiv if\ x \leq a\ then\ 1\ else\ if\ x \geq b\ then\ 0\ else\ (b - x)\ /\ (b - a)$

lemma *cts-step-uniformly-continuous*:
assumes [*arith*]: $a < b$
shows *uniformly-continuous-on UNIV* (*cts-step a b*)
⟨*proof*⟩

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

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*⟩

context
fixes *M-seq* :: *nat* ⇒ *real measure*
and *M* :: *real measure*
assumes *distr-M-seq* [*simp*]: $\bigwedge n.\ real\text{-}distribution\ (M\text{-}seq\ n)$

assumes *distr-M* [*simp*]: *real-distribution M*
begin

theorem *continuity-set-conv-imp-weak-conv*:
fixes *f* :: *real* ⇒ *real*
assumes ∗: ⋀*A*. *A* ∈ *sets borel* ⟹ *M* (*frontier A*) = *0* ⟹ (λ *n*. (*measure* (*M-seq n*) *A*)) ⟶ *measure M A*
shows *weak-conv-m M-seq M*
⟨*proof*⟩

theorem *integral-cts-step-conv-imp-weak-conv*:
assumes *integral-conv*: ⋀*x y*. *x* < *y* ⟹ (λ*n*. *integral*^{L} (*M-seq n*) (*cts-step x y*)) ⟶ *integral*^{L} *M* (*cts-step x y*)
shows *weak-conv-m M-seq M*
⟨*proof*⟩

theorem *integral-bdd-continuous-conv-imp-weak-conv*:
assumes
⋀*f*. (⋀*x*. *isCont f x*) ⟹ (⋀*x*. *abs* (*f x*) ≤ *1*) ⟹ (λ*n*. *integral*^{L} (*M-seq n*) *f*::*real*) ⟶ *integral*^{L} *M f*
shows
weak-conv-m M-seq M
⟨*proof*⟩

end

end

7 The Giry monad

theory *Giry-Monad*
imports *Probability-Measure HOL−Library.Monad-Syntax*
begin

7.1 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*⟩

lemma (**in** *subprob-space*) *emeasure-subprob-space-less-top*: *emeasure M A* ≠ *top*
⟨*proof*⟩

lemma *prob-space-imp-subprob-space*:
prob-space M $\Longrightarrow$ *subprob-space M*
⟨*proof*⟩

lemma *subprob-space-imp-sigma-finite*: *subprob-space M* $\Longrightarrow$ *sigma-finite-measure M*
⟨*proof*⟩

sublocale *prob-space* $\subseteq$ *subprob-space*
⟨*proof*⟩

lemma *subprob-space-sigma* [*simp*]: $\Omega \neq \{\}$ $\Longrightarrow$ *subprob-space* (*sigma* Ω *X*)
⟨*proof*⟩

lemma *subprob-space-null-measure*: *space M* $\neq \{\}$ $\Longrightarrow$ *subprob-space* (*null-measure M*)
⟨*proof*⟩

lemma (**in** *subprob-space*) *subprob-space-distr*:
assumes *f*: *f* $\in$ *measurable M M'* **and** *space M'* $\neq \{\}$ **shows** *subprob-space* (*distr M M' f*)
⟨*proof*⟩

lemma (**in** *subprob-space*) *subprob-emeasure-le-1*: *emeasure M X* $\leq$ *1*
⟨*proof*⟩

lemma (**in** *subprob-space*) *subprob-measure-le-1*: *measure M X* $\leq$ *1*
⟨*proof*⟩

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*⟩

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

emeasure (*density lborel* (λ*x*. *f* (*g x*) ∗ *g*′ *x*)) {*a*..*b*}
⟨*proof*⟩

locale *pair-subprob-space* =
pair-sigma-finite M1 M2 + *M1*: *subprob-space M1* + *M2*: *subprob-space M2* **for** *M1 M2*

sublocale *pair-subprob-space* ⊆ *P?*: *subprob-space M1* ⨂$_M$ *M2*
⟨*proof*⟩

lemma *subprob-space-null-measure-iff*:
subprob-space (*null-measure M*) ⟷ *space M* ≠ {}
⟨*proof*⟩

lemma *subprob-space-restrict-space*:
assumes *M*: *subprob-space M*
and *A*: *A* ∩ *space M* ∈ *sets M A* ∩ *space M* ≠ {}
shows *subprob-space* (*restrict-space M A*)
⟨*proof*⟩

definition *subprob-algebra* :: *'a measure* ⇒ *'a measure measure* **where**
subprob-algebra K =
(*SUP A* ∈ *sets K*. *vimage-algebra* {*M*. *subprob-space M* ∧ *sets M* = *sets K*}
(λ*M*. *emeasure M A*) *borel*)

lemma *space-subprob-algebra*: *space* (*subprob-algebra A*) = {*M*. *subprob-space M* ∧ *sets M* = *sets A*}
⟨*proof*⟩

lemma *subprob-algebra-cong*: *sets M* = *sets N* ⟹ *subprob-algebra M* = *subprob-algebra N*
⟨*proof*⟩

lemma *measurable-emeasure-subprob-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (λ*M*. *emeasure M a*) ∈ *borel-measurable* (*subprob-algebra A*)
⟨*proof*⟩

lemma *measurable-measure-subprob-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (λ*M*. *measure M a*) ∈ *borel-measurable* (*subprob-algebra A*)
⟨*proof*⟩

lemma *subprob-measurableD*:
assumes *N*: *N* ∈ *measurable M* (*subprob-algebra S*) **and** *x*: *x* ∈ *space M*
shows *space* (*N x*) = *space S*
and *sets* (*N x*) = *sets S*
and *measurable* (*N x*) *K* = *measurable S K*
and *measurable K* (*N x*) = *measurable K S*
⟨*proof*⟩

⟨*ML*⟩

context
fixes *K M N* **assumes** *K*: $K \in$ *measurable M* (*subprob-algebra N*)
begin

lemma *subprob-space-kernel*: $a \in$ *space M* $\Longrightarrow$ *subprob-space* (*K a*)
⟨*proof*⟩

lemma *sets-kernel*: $a \in$ *space M* $\Longrightarrow$ *sets* (*K a*) = *sets N*
⟨*proof*⟩

lemma *measurable-emeasure-kernel*[*measurable*]:
$A \in$ *sets N* $\Longrightarrow$ (λa. *emeasure* (*K a*) *A*) $\in$ *borel-measurable M*
⟨*proof*⟩

end

lemma *measurable-subprob-algebra*:
($\bigwedge a$. $a \in$ *space M* $\Longrightarrow$ *subprob-space* (*K a*)) $\Longrightarrow$
($\bigwedge a$. $a \in$ *space M* $\Longrightarrow$ *sets* (*K a*) = *sets N*) $\Longrightarrow$
($\bigwedge A$. $A \in$ *sets N* $\Longrightarrow$ (λa. *emeasure* (*K a*) *A*) $\in$ *borel-measurable M*) $\Longrightarrow$
K $\in$ *measurable M* (*subprob-algebra N*)
⟨*proof*⟩

lemma *measurable-submarkov*:
K $\in$ *measurable M* (*subprob-algebra M*) $\longleftrightarrow$
($\forall x \in$*space M*. *subprob-space* (*K x*) $\wedge$ *sets* (*K x*) = *sets M*) $\wedge$
($\forall A \in$*sets M*. (λx. *emeasure* (*K x*) *A*) $\in$ *measurable M borel*)
⟨*proof*⟩

lemma *measurable-subprob-algebra-generated*:
assumes *eq*: *sets N* = *sigma-sets* Ω *G* **and** *Int-stable G G* $\subseteq$ *Pow* Ω
assumes *subsp*: $\bigwedge a$. $a \in$ *space M* $\Longrightarrow$ *subprob-space* (*K a*)
assumes *sets*: $\bigwedge a$. $a \in$ *space M* $\Longrightarrow$ *sets* (*K a*) = *sets N*
assumes $\bigwedge A$. $A \in G \Longrightarrow$ (λa. *emeasure* (*K a*) *A*) $\in$ *borel-measurable M*
assumes Ω: (λa. *emeasure* (*K a*) Ω) $\in$ *borel-measurable M*
shows *K* $\in$ *measurable M* (*subprob-algebra N*)
⟨*proof*⟩

lemma *space-subprob-algebra-empty-iff*:
space (*subprob-algebra N*) = {} $\longleftrightarrow$ *space N* = {}
⟨*proof*⟩

lemma *nn-integral-measurable-subprob-algebra*[*measurable*]:
assumes *f*: *f* $\in$ *borel-measurable N*
shows (λM. *integral*^{N} *M f*) $\in$ *borel-measurable* (*subprob-algebra N*) (**is** - $\in$ *?B*)
⟨*proof*⟩

lemma *measurable-distr*:
assumes [*measurable*]: $f \in$ *measurable* M N
shows $(\lambda M'.$ *distr* M' N $f) \in$ *measurable* (*subprob-algebra* M) (*subprob-algebra* N)
⟨*proof*⟩

lemma *emeasure-space-subprob-algebra*[*measurable*]:
$(\lambda a.$ *emeasure* a (*space* a)) $\in$ *borel-measurable* (*subprob-algebra* N)
⟨*proof*⟩

lemma *integrable-measurable-subprob-algebra*[*measurable*]:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes [*measurable*]: $f \in$ *borel-measurable* N
shows *Measurable.pred* (*subprob-algebra* N) $(\lambda M.$ *integrable* M $f)$
⟨*proof*⟩

lemma *integral-measurable-subprob-algebra*[*measurable*]:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes f [*measurable*]: $f \in$ *borel-measurable* N
shows $(\lambda M.$ *integral*^{L} M $f) \in$ *subprob-algebra* $N \rightarrow_M$ *borel*
⟨*proof*⟩

lemma *measurable-pair-measure*:
assumes f: $f \in$ *measurable* M (*subprob-algebra* N)
assumes g: $g \in$ *measurable* M (*subprob-algebra* L)
shows $(\lambda x.$ f $x \bigotimes_M$ g $x) \in$ *measurable* M (*subprob-algebra* $(N \bigotimes_M L)$)
⟨*proof*⟩

lemma *restrict-space-measurable*:
assumes X: $X \neq \{\}$ $X \in$ *sets* K
assumes N: $N \in$ *measurable* M (*subprob-algebra* K)
shows $(\lambda x.$ *restrict-space* $(N\,x)$ $X) \in$ *measurable* M (*subprob-algebra* (*restrict-space* K X))
⟨*proof*⟩

7.2 Properties of "return"

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

lemma *space-return*[*simp*]: *space* (*return* M x) = *space* M
⟨*proof*⟩

lemma *sets-return*[*simp*]: *sets* (*return* M x) = *sets* M
⟨*proof*⟩

lemma *measurable-return1*[*simp*]: *measurable* (*return* N x) L = *measurable* N L
⟨*proof*⟩

lemma *measurable-return2*[*simp*]: *measurable L* (*return N x*) = *measurable L N*
⟨*proof*⟩

lemma *return-sets-cong*: *sets M* = *sets N* ⟹ *return M* = *return N*
⟨*proof*⟩

lemma *return-cong*: *sets A* = *sets B* ⟹ *return A x* = *return B x*
⟨*proof*⟩

lemma *emeasure-return*[*simp*]:
assumes *A* ∈ *sets M*
shows *emeasure* (*return M x*) *A* = *indicator A x*
⟨*proof*⟩

lemma *prob-space-return*: *x* ∈ *space M* ⟹ *prob-space* (*return M x*)
⟨*proof*⟩

lemma *subprob-space-return*: *x* ∈ *space M* ⟹ *subprob-space* (*return M x*)
⟨*proof*⟩

lemma *subprob-space-return-ne*:
assumes *space M* ≠ {} **shows** *subprob-space* (*return M x*)
⟨*proof*⟩

lemma *measure-return*: **assumes** *X*: *X* ∈ *sets M* **shows** *measure* (*return M x*) *X* = *indicator X x*
⟨*proof*⟩

lemma *AE-return*:
assumes [*simp*]: *x* ∈ *space M* **and** [*measurable*]: *Measurable.pred M P*
shows (*AE y in return M x*. *P y*) ⟷ *P x*
⟨*proof*⟩

lemma *nn-integral-return*:
assumes *x* ∈ *space M g* ∈ *borel-measurable M*
shows (∫⁺ *a*. *g a* ∂*return M x*) = *g x*
⟨*proof*⟩

lemma *integral-return*:
fixes *g* :: - ⇒ *'a* :: {*banach*, *second-countable-topology*}
assumes *x* ∈ *space M g* ∈ *borel-measurable M*
shows (∫ *a*. *g a* ∂*return M x*) = *g x*
⟨*proof*⟩

lemma *return-measurable*[*measurable*]: *return N* ∈ *measurable N* (*subprob-algebra N*)
⟨*proof*⟩

lemma *distr-return*:
assumes $f \in$ *measurable* M N **and** $x \in$ *space* M
shows *distr* (*return* M x) N f = *return* N (f x)
⟨*proof*⟩

lemma *return-restrict-space*:
$\Omega \in$ *sets* $M \Longrightarrow$ *return* (*restrict-space* M Ω) x = *restrict-space* (*return* M x) Ω
⟨*proof*⟩

lemma *measurable-distr2*:
assumes f[*measurable*]: *case-prod* $f \in$ *measurable* ($L \bigotimes_M M$) N
assumes g[*measurable*]: $g \in$ *measurable* L (*subprob-algebra* M)
shows (λx. *distr* (g x) N (f x)) $\in$ *measurable* L (*subprob-algebra* N)
⟨*proof*⟩

lemma *nn-integral-measurable-subprob-algebra2*:
assumes f[*measurable*]: ($\lambda(x, y)$. f x y) $\in$ *borel-measurable* ($M \bigotimes_M N$)
assumes N[*measurable*]: $L \in$ *measurable* M (*subprob-algebra* N)
shows (λx. *integral*^{N} (L x) (f x)) $\in$ *borel-measurable* M
⟨*proof*⟩

lemma *emeasure-measurable-subprob-algebra2*:
assumes A[*measurable*]: (*SIGMA* x:*space* M. A x) $\in$ *sets* ($M \bigotimes_M N$)
assumes L[*measurable*]: $L \in$ *measurable* M (*subprob-algebra* N)
shows (λx. *emeasure* (L x) (A x)) $\in$ *borel-measurable* M
⟨*proof*⟩

lemma *measure-measurable-subprob-algebra2*:
assumes A[*measurable*]: (*SIGMA* x:*space* M. A x) $\in$ *sets* ($M \bigotimes_M N$)
assumes L[*measurable*]: $L \in$ *measurable* M (*subprob-algebra* N)
shows (λx. *measure* (L x) (A x)) $\in$ *borel-measurable* M
⟨*proof*⟩

definition *select-sets* M = (*SOME* N. *sets* M = *sets* (*subprob-algebra* N))

lemma *select-sets1*:
sets M = *sets* (*subprob-algebra* N) $\Longrightarrow$ *sets* M = *sets* (*subprob-algebra* (*select-sets* M))
⟨*proof*⟩

lemma *sets-select-sets*[*simp*]:
assumes *sets*: *sets* M = *sets* (*subprob-algebra* N)
shows *sets* (*select-sets* M) = *sets* N
⟨*proof*⟩

lemma *space-select-sets*[*simp*]:
sets M = *sets* (*subprob-algebra* N) $\Longrightarrow$ *space* (*select-sets* M) = *space* N
⟨*proof*⟩

7.3 Join

definition *join* :: *'a measure measure ⇒ 'a measure* **where**
join M = measure-of (space (select-sets M)) (sets (select-sets M)) (λB. ∫⁺ M'. emeasure M' B ∂M)

lemma
shows *space-join*[*simp*]: *space (join M) = space (select-sets M)*
and *sets-join*[*simp*]: *sets (join M) = sets (select-sets M)*
⟨*proof*⟩

lemma *emeasure-join*:
assumes *M*[*simp*, *measurable-cong*]: *sets M = sets (subprob-algebra N)* **and** *A*: *A ∈ sets N*
shows *emeasure (join M) A = (∫⁺ M'. emeasure M' A ∂M)*
⟨*proof*⟩

lemma *measurable-join*:
join ∈ measurable (subprob-algebra (subprob-algebra N)) (subprob-algebra N)
⟨*proof*⟩

lemma *nn-integral-join*:
assumes *f*: *f ∈ borel-measurable N*
and *M*[*measurable-cong*]: *sets M = sets (subprob-algebra N)*
shows *(∫⁺x. f x ∂join M) = (∫⁺M'. ∫⁺x. f x ∂M' ∂M)*
⟨*proof*⟩

lemma *measurable-join1*:
⟦ f ∈ measurable N K; sets M = sets (subprob-algebra N) ⟧
⟹ f ∈ measurable (join M) K
⟨*proof*⟩

lemma
fixes *f* :: *- ⇒ real*
assumes *f-measurable* [*measurable*]: *f ∈ borel-measurable N*
and *f-bounded*: *⋀x. x ∈ space N ⟹ |f x| ≤ 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') ≤ ennreal B'*
shows *integrable-join*: *integrable (join M) f* (**is** *?integrable*)
and *integral-join*: *integral^L (join M) f = ∫ M'. integral^L M' f ∂M* (**is** *?integral*)
⟨*proof*⟩

lemma *join-assoc*:
assumes *M*[*measurable-cong*]: *sets M = sets (subprob-algebra (subprob-algebra N))*
shows *join (distr M (subprob-algebra N) join) = join (join M)*
⟨*proof*⟩

lemma *join-return*:

assumes *sets M* = *sets N* **and** *subprob-space M*
shows *join* (*return* (*subprob-algebra N*) *M*) = *M*
⟨*proof*⟩

lemma *join-return'*:
assumes *sets N* = *sets M*
shows *join* (*distr M* (*subprob-algebra N*) (*return N*)) = *M*
⟨*proof*⟩

lemma *join-distr-distr*:
fixes *f* :: *'a* ⇒ *'b* **and** *M* :: *'a measure measure* **and** *N* :: *'b measure*
assumes *sets M* = *sets* (*subprob-algebra R*) **and** *f* ∈ *measurable R N*
shows *join* (*distr M* (*subprob-algebra N*) (λ*M*. *distr M N f*)) = *distr* (*join M*) *N f* (**is** *?r* = *?l*)
⟨*proof*⟩

definition *bind* :: *'a measure* ⇒ (*'a* ⇒ *'b measure*) ⇒ *'b measure* **where**
bind M f = (*if space M* = {} *then count-space* {} *else*
join (*distr M* (*subprob-algebra* (*f* (*SOME x*. *x* ∈ *space M*))) *f*))

adhoc-overloading *Monad-Syntax.bind* ⇌ *bind*

lemma *bind-empty*:
space M = {} ⟹ *bind M f* = *count-space* {}
⟨*proof*⟩

lemma *bind-nonempty*:
space M ≠ {} ⟹ *bind M f* = *join* (*distr M* (*subprob-algebra* (*f* (*SOME x*. *x* ∈ *space M*))) *f*)
⟨*proof*⟩

lemma *sets-bind-empty*: *sets M* = {} ⟹ *sets* (*bind M f*) = {{}}
⟨*proof*⟩

lemma *space-bind-empty*: *space M* = {} ⟹ *space* (*bind M f*) = {}
⟨*proof*⟩

lemma *sets-bind*[*simp*, *measurable-cong*]:
assumes *f*: ⋀*x*. *x* ∈ *space M* ⟹ *sets* (*f x*) = *sets N* **and** *M*: *space M* ≠ {}
shows *sets* (*bind M f*) = *sets N*
⟨*proof*⟩

lemma *space-bind*[*simp*]:
assumes ⋀*x*. *x* ∈ *space M* ⟹ *sets* (*f x*) = *sets N* **and** *space M* ≠ {}
shows *space* (*bind M f*) = *space N*
⟨*proof*⟩

lemma *bind-cong-All*:
assumes ∀ *x* ∈ *space M*. *f x* = *g x*

shows *bind M f = bind M g*
⟨*proof*⟩

lemma *bind-cong*:
M = N ⟹ (⋀x. x ∈ space M ⟹ f x = g x) ⟹ bind M f = bind N g
⟨*proof*⟩

lemma *bind-nonempty′*:
assumes *f ∈ measurable M (subprob-algebra N) x ∈ space M*
shows *bind M f = join (distr M (subprob-algebra N) f)*
⟨*proof*⟩

lemma *bind-nonempty″*:
assumes *f ∈ measurable M (subprob-algebra N) space M ≠ {}*
shows *bind M f = join (distr M (subprob-algebra N) f)*
⟨*proof*⟩

lemma *emeasure-bind*:
⟦space M ≠ {}; f ∈ measurable M (subprob-algebra N);X ∈ sets N⟧
⟹ emeasure (M ⤜ f) X = ∫⁺x. emeasure (f x) X ∂M
⟨*proof*⟩

lemma *nn-integral-bind*:
assumes *f*: *f ∈ borel-measurable B*
assumes *N*: *N ∈ measurable M (subprob-algebra B)*
shows *(∫⁺x. f x ∂(M ⤜ N)) = (∫⁺x. ∫⁺y. f y ∂N x ∂M)*
⟨*proof*⟩

lemma *AE-bind*:
assumes *N*[*measurable*]: *N ∈ measurable M (subprob-algebra B)*
assumes *P*[*measurable*]: *Measurable.pred B P*
shows *(AE x in M ⤜ N. P x) ⟷ (AE x in M. AE y in N x. P y)*
⟨*proof*⟩

lemma *measurable-bind′*:
assumes *M1*: *f ∈ measurable M (subprob-algebra N)* **and**
M2: *case-prod g ∈ measurable (M ⨂M N) (subprob-algebra R)*
shows *(λx. bind (f x) (g x)) ∈ measurable M (subprob-algebra R)*
⟨*proof*⟩

lemma *measurable-bind*[*measurable (raw)*]:
assumes *M1*: *f ∈ measurable M (subprob-algebra N)* **and**
M2: *(λx. g (fst x) (snd x)) ∈ measurable (M ⨂M N) (subprob-algebra R)*
shows *(λx. bind (f x) (g x)) ∈ measurable M (subprob-algebra R)*
⟨*proof*⟩

lemma *measurable-bind2*:
assumes *f ∈ measurable M (subprob-algebra N)* **and** *g ∈ measurable N (subprob-algebra R)*

shows $(\lambda x.\ bind\ (f\ x)\ g) \in measurable\ M\ (subprob\text{-}algebra\ R)$
⟨*proof*⟩

lemma *subprob-space-bind*:
assumes $subprob\text{-}space\ M\ f \in measurable\ M\ (subprob\text{-}algebra\ N)$
shows $subprob\text{-}space\ (M \gg\!= f)$
⟨*proof*⟩

lemma
fixes $f :: - \Rightarrow real$
assumes *f-measurable* [*measurable*]: $f \in borel\text{-}measurable\ K$
and *f-bounded*: $\bigwedge x.\ x \in space\ K \Longrightarrow |f\ x| \leq B$
and N [*measurable*]: $N \in measurable\ M\ (subprob\text{-}algebra\ K)$
and *fin*: *finite-measure M*
and *M-bounded*: $AE\ x\ in\ M.\ emeasure\ (N\ x)\ (space\ (N\ x)) \leq ennreal\ B'$
shows *integrable-bind*: $integrable\ (bind\ M\ N)\ f$ (**is** *?integrable*)
and *integral-bind*: $integral^L\ (bind\ M\ N)\ f = \int x.\ integral^L\ (N\ x)\ f\ \partial M$ (**is** *?integral*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-space-bind*:
assumes *ae*: $AE\ x\ in\ M.\ prob\text{-}space\ (N\ x)$
and N[*measurable*]: $N \in measurable\ M\ (subprob\text{-}algebra\ S)$
shows $prob\text{-}space\ (M \gg\!= N)$
⟨*proof*⟩

lemma (**in** *subprob-space*) *bind-in-space*:
$A \in measurable\ M\ (subprob\text{-}algebra\ N) \Longrightarrow (M \gg\!= A) \in space\ (subprob\text{-}algebra\ N)$
⟨*proof*⟩

lemma (**in** *subprob-space*) *measure-bind*:
assumes *f*: $f \in measurable\ M\ (subprob\text{-}algebra\ N)$ **and** *X*: $X \in sets\ N$
shows $measure\ (M \gg\!= f)\ X = \int x.\ measure\ (f\ x)\ X\ \partial M$
⟨*proof*⟩

lemma *emeasure-bind-const*:
$space\ M \neq \{\} \Longrightarrow X \in sets\ N \Longrightarrow subprob\text{-}space\ N \Longrightarrow$
$emeasure\ (M \gg\!= (\lambda x.\ N))\ X = emeasure\ N\ X * emeasure\ M\ (space\ M)$
⟨*proof*⟩

lemma *emeasure-bind-const'*:
assumes *subprob-space M subprob-space N*
shows $emeasure\ (M \gg\!= (\lambda x.\ N))\ X = emeasure\ N\ X * emeasure\ M\ (space\ M)$
⟨*proof*⟩

lemma *emeasure-bind-const-prob-space*:
assumes *prob-space M subprob-space N*
shows $emeasure\ (M \gg\!= (\lambda x.\ N))\ X = emeasure\ N\ X$

⟨*proof*⟩

lemma *bind-return*:
assumes $f \in$ *measurable* M (*subprob-algebra* N) **and** $x \in$ *space* M
shows *bind* (*return* M x) $f = f\ x$
⟨*proof*⟩

lemma *bind-return'*:
shows *bind* M (*return* M) $= M$
⟨*proof*⟩

lemma *distr-bind*:
assumes N: $N \in$ *measurable* M (*subprob-algebra* K) *space* $M \neq \{\}$
assumes f: $f \in$ *measurable* K R
shows *distr* ($M \gg\!= N$) R f $= (M \gg\!= (\lambda x.\ \textit{distr}\ (N\ x)\ R\ f))$
⟨*proof*⟩

lemma *bind-distr*:
assumes f[*measurable*]: $f \in$ *measurable* M X
assumes N[*measurable*]: $N \in$ *measurable* X (*subprob-algebra* K) **and** *space* $M \neq \{\}$
shows (*distr* M X $f \gg\!= N$) $= (M \gg\!= (\lambda x.\ N\ (f\ x)))$
⟨*proof*⟩

lemma *bind-count-space-singleton*:
assumes *subprob-space* ($f\ x$)
shows *count-space* $\{x\} \gg\!= f = f\ x$
⟨*proof*⟩

lemma *restrict-space-bind*:
assumes N: $N \in$ *measurable* M (*subprob-algebra* K)
assumes *space* $M \neq \{\}$
assumes X[*simp*]: $X \in$ *sets* K $X \neq \{\}$
shows *restrict-space* (*bind* M N) X $=$ *bind* M ($\lambda x.$ *restrict-space* ($N\ x$) X)
⟨*proof*⟩

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*⟩

lemma *bind-const'*: ⟦*prob-space* M; *subprob-space* N⟧ $\Longrightarrow M \gg\!= (\lambda x.\ N) = N$
⟨*proof*⟩

lemma *bind-return-distr*:
assumes *space* $M \neq \{\}$ $f \in$ *measurable* M N

shows *bind M (return N ∘ f) = distr M N f*
⟨*proof*⟩

lemma *bind-return-distr'*:
space M ≠ {} ⟹ f ∈ measurable M N ⟹ bind M (λx. return N (f x)) = distr M N f
⟨*proof*⟩

lemma *bind-assoc*:
fixes *f :: 'a ⇒ 'b measure* **and** *g :: 'b ⇒ 'c measure*
assumes *M1*: *f ∈ measurable M (subprob-algebra N)* **and** *M2*: *g ∈ measurable N (subprob-algebra R)*
shows *bind (bind M f) g = bind M (λx. bind (f x) g)*
⟨*proof*⟩

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*⟩

lemma (**in** *prob-space*) *M-in-subprob*[*measurable* (*raw*)]: *M ∈ space (subprob-algebra M)*
⟨*proof*⟩

lemma (**in** *pair-prob-space*) *pair-measure-eq-bind*:
(M1 ⨂M M2) = (M1 ⤜ (λx. M2 ⤜ (λy. return (M1 ⨂M M2) (x, y))))
⟨*proof*⟩

lemma (**in** *pair-prob-space*) *bind-rotate*:
assumes *C*[*measurable*]: *(λ(x, y). C x y) ∈ measurable (M1 ⨂M M2) (subprob-algebra N)*
shows *(M1 ⤜ (λx. M2 ⤜ (λy. C x y))) = (M2 ⤜ (λy. M1 ⤜ (λx. C x y)))*
⟨*proof*⟩

lemma *bind-return''*: *sets M = sets N ⟹ M ⤜ return N = M*
⟨*proof*⟩

lemma (**in** *prob-space*) *distr-const*[*simp*]:
c ∈ space N ⟹ distr M N (λx. c) = return N c
⟨*proof*⟩

lemma *return-count-space-eq-density*:
return (count-space M) x = density (count-space M) (indicator {x})
⟨*proof*⟩

lemma *null-measure-in-space-subprob-algebra* [*simp*]:
null-measure M ∈ *space* (*subprob-algebra M*) ⟷ *space M* ≠ {}
⟨*proof*⟩

7.4 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*}
⟨*proof*⟩

lemma *measurable-measure-prob-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (λ*M*. *Sigma-Algebra.measure M a*) ∈ *prob-algebra A* →$_M$ *borel*
⟨*proof*⟩

lemma *measurable-prob-algebraD*:
f ∈ *N* →$_M$ *prob-algebra M* ⟹ *f* ∈ *N* →$_M$ *subprob-algebra M*
⟨*proof*⟩

lemma *measure-measurable-prob-algebra2*:
Sigma (*space M*) *A* ∈ *sets* (*M* ⨂$_M$ *N*) ⟹ *L* ∈ *M* →$_M$ *prob-algebra N* ⟹
(λ*x*. *Sigma-Algebra.measure* (*L x*) (*A x*)) ∈ *borel-measurable M*
⟨*proof*⟩

lemma *measurable-prob-algebraI*:
(⋀*x*. *x* ∈ *space N* ⟹ *prob-space* (*f x*)) ⟹ *f* ∈ *N* →$_M$ *subprob-algebra M* ⟹
f ∈ *N* →$_M$ *prob-algebra M*
⟨*proof*⟩

lemma *measurable-distr-prob-space*:
assumes *f*: *f* ∈ *M* →$_M$ *N*
shows (λ*M'*. *distr M' N f*) ∈ *prob-algebra M* →$_M$ *prob-algebra N*
⟨*proof*⟩

lemma *measurable-return-prob-space*[*measurable*]: *return N* ∈ *N* →$_M$ *prob-algebra N*
⟨*proof*⟩

lemma *measurable-distr-prob-space2*[*measurable* (*raw*)]:
assumes *f*: *g* ∈ *L* →$_M$ *prob-algebra M* (λ(*x*, *y*). *f x y*) ∈ *L* ⨂$_M$ *M* →$_M$ *N*
shows (λ*x*. *distr* (*g x*) *N* (*f x*)) ∈ *L* →$_M$ *prob-algebra N*
⟨*proof*⟩

lemma *measurable-bind-prob-space*:
assumes *f*: *f* ∈ *M* →$_M$ *prob-algebra N* **and** *g*: *g* ∈ *N* →$_M$ *prob-algebra R*
shows (λ*x*. *bind* (*f x*) *g*) ∈ *M* →$_M$ *prob-algebra R*
⟨*proof*⟩

lemma *measurable-bind-prob-space2*[*measurable* (*raw*)]:
assumes *f*: $f \in M \to_M$ *prob-algebra N* **and** *g*: $(\lambda(x, y).\ g\ x\ y) \in (M \bigotimes_M N) \to_M$ *prob-algebra R*
shows $(\lambda x.\ bind\ (f\ x)\ (g\ x)) \in M \to_M$ *prob-algebra R*
⟨*proof*⟩

lemma *measurable-prob-algebra-generated*:
assumes *eq*: *sets N* = *sigma-sets* Ω *G* **and** *Int-stable G* $G \subseteq$ *Pow* Ω
assumes *subsp*: $\bigwedge a.\ a \in space\ M \Longrightarrow prob\text{-}space\ (K\ a)$
assumes *sets*: $\bigwedge a.\ a \in space\ M \Longrightarrow sets\ (K\ a) = sets\ N$
assumes $\bigwedge A.\ A \in G \Longrightarrow (\lambda a.\ emeasure\ (K\ a)\ A) \in borel\text{-}measurable\ M$
shows *K* ∈ *measurable M* (*prob-algebra N*)
⟨*proof*⟩

lemma *in-space-prob-algebra*:
x ∈ *space* (*prob-algebra M*) ⟹ *emeasure x* (*space M*) = *1*
⟨*proof*⟩

lemma *prob-space-pair*:
assumes *prob-space M prob-space N* **shows** *prob-space* $(M \bigotimes_M N)$
⟨*proof*⟩

lemma *measurable-pair-prob*[*measurable*]:
$f \in M \to_M$ *prob-algebra N* ⟹ $g \in M \to_M$ *prob-algebra L* ⟹ $(\lambda x.\ f\ x \bigotimes_M g\ x) \in M \to_M$ *prob-algebra* $(N \bigotimes_M L)$
⟨*proof*⟩

lemma *emeasure-bind-prob-algebra*:
assumes *A*: *A* ∈ *space* (*prob-algebra N*)
assumes *B*: *B* ∈ *N* →*M prob-algebra L*
assumes *X*: *X* ∈ *sets L*
shows *emeasure* (*bind A B*) *X* = $(\int^+ x.\ emeasure\ (B\ x)\ X\ \partial A)$
⟨*proof*⟩

lemma *prob-space-bind′*:
assumes *A*: *A* ∈ *space* (*prob-algebra M*) **and** *B*: *B* ∈ *M* →*M prob-algebra N*
shows *prob-space* (*A* ⤜ *B*)
⟨*proof*⟩

lemma *sets-bind′*:
assumes *A*: *A* ∈ *space* (*prob-algebra M*) **and** *B*: *B* ∈ *M* →*M prob-algebra N*
shows *sets* (*A* ⤜ *B*) = *sets N*
⟨*proof*⟩

lemma *bind-cong-AE′*:
assumes *M*: *M* ∈ *space* (*prob-algebra L*)
and *f*: *f* ∈ *L* →*M prob-algebra N* **and** *g*: *g* ∈ *L* →*M prob-algebra N*

and *ae*: $AE\ x\ in\ M.\ f\ x = g\ x$
shows $bind\ M\ f = bind\ M\ g$
⟨*proof*⟩

lemma *density-discrete*:
$countable\ A \Longrightarrow sets\ N = Set.Pow\ A \Longrightarrow (\bigwedge x.\ f\ x \geq 0) \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\ x = emeasure\ N\ \{x\}) \Longrightarrow$
$density\ (count\text{-}space\ A)\ f = N$
⟨*proof*⟩

lemma *distr-density-discrete*:
fixes f'
assumes $countable\ A$
assumes $f' \in borel\text{-}measurable\ M$
assumes $g \in measurable\ M\ (count\text{-}space\ A)$
defines $f \equiv \lambda x.\ \int^+ t.\ (if\ g\ t = x\ then\ 1\ else\ 0) * f'\ t\ \partial M$
assumes $\bigwedge x.\ \ x \in space\ M \Longrightarrow g\ x \in A$
shows $density\ (count\text{-}space\ A)\ (\lambda x.\ f\ x) = distr\ (density\ M\ f')\ (count\text{-}space\ A)\ g$
⟨*proof*⟩

lemma *bind-cong-AE*:
assumes $M = N$
assumes *f*: $f \in measurable\ N\ (subprob\text{-}algebra\ B)$
assumes *g*: $g \in measurable\ N\ (subprob\text{-}algebra\ B)$
assumes *ae*: $AE\ x\ in\ N.\ f\ x = g\ x$
shows $bind\ M\ f = bind\ N\ g$
⟨*proof*⟩

lemma *bind-cong-simp*: $M = N \Longrightarrow (\bigwedge x.\ x{\in}space\ M\ =simp{=}>\ f\ x = g\ x) \Longrightarrow bind\ M\ f = bind\ N\ g$
⟨*proof*⟩

lemma *sets-bind-measurable*:
assumes *f*: $f \in measurable\ M\ (subprob\text{-}algebra\ B)$
assumes *M*: $space\ M \neq \{\}$
shows $sets\ (M \gg\!= f) = sets\ B$
⟨*proof*⟩

lemma *space-bind-measurable*:
assumes *f*: $f \in measurable\ M\ (subprob\text{-}algebra\ B)$
assumes *M*: $space\ M \neq \{\}$
shows $space\ (M \gg\!= f) = space\ B$
⟨*proof*⟩

lemma *bind-distr-return*:
$f \in M \rightarrow_M N \Longrightarrow g \in N \rightarrow_M L \Longrightarrow space\ M \neq \{\} \Longrightarrow$
$distr\ M\ N\ f \gg\!= (\lambda x.\ return\ L\ (g\ x)) = distr\ M\ L\ (\lambda x.\ g\ (f\ x))$
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-eq-constD*:
assumes *AE x in M*. $x = y$
shows $M = return\ M\ y\ y \in space\ M$
⟨*proof*⟩

end

8 Projective Family

theory *Projective-Family*
imports *Giry-Monad*
begin

lemma *vimage-restrict-preserve-mono*:
assumes *J*: $J \subseteq I$
and *sets*: $A \subseteq (\Pi_E\ i{\in}J.\ S\ i)\ B \subseteq (\Pi_E\ i{\in}J.\ S\ i)$ **and** *ne*: $(\Pi_E\ i{\in}I.\ S\ i) \neq \{\}$
and *eq*: $(\lambda x.\ restrict\ x\ J) -` A \cap (\Pi_E\ i{\in}I.\ S\ i) \subseteq (\lambda x.\ restrict\ x\ J) -` B \cap (\Pi_E\ i{\in}I.\ S\ i)$
shows $A \subseteq B$
⟨*proof*⟩

locale *projective-family* =
fixes $I :: 'i\ set$ **and** $P :: 'i\ set \Rightarrow ('i \Rightarrow 'a)\ measure$ **and** $M :: 'i \Rightarrow 'a\ measure$
assumes *P*: $\bigwedge J\ H.\ J \subseteq H \Longrightarrow finite\ H \Longrightarrow H \subseteq I \Longrightarrow P\ J = distr\ (P\ H)\ (PiM\ J\ M)\ (\lambda f.\ restrict\ f\ J)$
assumes *prob-space-P*: $\bigwedge J.\ finite\ J \Longrightarrow J \subseteq I \Longrightarrow prob\text{-}space\ (P\ J)$
begin

lemma *sets-P*: $finite\ J \Longrightarrow J \subseteq I \Longrightarrow sets\ (P\ J) = sets\ (PiM\ J\ M)$
⟨*proof*⟩

lemma *space-P*: $finite\ J \Longrightarrow J \subseteq I \Longrightarrow space\ (P\ J) = space\ (PiM\ J\ M)$
⟨*proof*⟩

lemma *not-empty-M*: $i \in I \Longrightarrow space\ (M\ i) \neq \{\}$
⟨*proof*⟩

lemma *not-empty*: $space\ (PiM\ I\ M) \neq \{\}$
⟨*proof*⟩

abbreviation
$emb\ L\ K \equiv prod\text{-}emb\ L\ M\ K$

lemma *emb-preserve-mono*:
assumes $J \subseteq L\ L \subseteq I$ **and** *sets*: $X \in sets\ (Pi_M\ J\ M)\ Y \in sets\ (Pi_M\ J\ M)$
assumes $emb\ L\ J\ X \subseteq emb\ L\ J\ Y$
shows $X \subseteq Y$

⟨*proof*⟩

lemma *emb-injective*:
assumes *L*: $J \subseteq L$ $L \subseteq I$ **and** *X*: $X \in sets\ (Pi_M\ J\ M)$ **and** *Y*: $Y \in sets\ (Pi_M\ J\ M)$
shows $emb\ L\ J\ X = emb\ L\ J\ Y \Longrightarrow X = Y$
⟨*proof*⟩

lemma *emeasure-P*: $J \subseteq K \Longrightarrow finite\ K \Longrightarrow K \subseteq I \Longrightarrow X \in sets\ (PiM\ J\ M) \Longrightarrow P\ K\ (emb\ K\ J\ X) = P\ J\ X$
⟨*proof*⟩

inductive-set *generator* :: $('i \Rightarrow 'a)\ set\ set$ **where**
$finite\ J \Longrightarrow J \subseteq I \Longrightarrow X \in sets\ (Pi_M\ J\ M) \Longrightarrow emb\ I\ J\ X \in generator$

lemma *algebra-generator*: $algebra\ (space\ (PiM\ I\ M))\ generator$
⟨*proof*⟩

interpretation *generator*: $algebra\ space\ (PiM\ I\ M)\ generator$
⟨*proof*⟩

lemma *sets-PiM-generator*: $sets\ (PiM\ I\ M) = sigma\text{-}sets\ (space\ (PiM\ I\ M))\ generator$
⟨*proof*⟩

definition *mu-G* (‹μG›) **where**
$\mu G\ A = (THE\ x.\ \forall J{\subseteq}I.\ finite\ J \longrightarrow (\forall X{\in}sets\ (Pi_M\ J\ M).\ A = emb\ I\ J\ X \longrightarrow x = emeasure\ (P\ J)\ X))$

definition *lim* :: $('i \Rightarrow 'a)\ measure$ **where**
$lim = extend\text{-}measure\ (space\ (PiM\ I\ M))\ generator\ (\lambda x.\ x)\ \mu G$

lemma *space-lim*[*simp*]: $space\ lim = space\ (PiM\ I\ M)$
⟨*proof*⟩

lemma *sets-lim*[*simp*, *measurable*]: $sets\ lim = sets\ (PiM\ I\ M)$
⟨*proof*⟩

lemma *mu-G-spec*:
assumes *J*: $finite\ J\ J \subseteq I\ X \in sets\ (Pi_M\ J\ M)$
shows $\mu G\ (emb\ I\ J\ X) = emeasure\ (P\ J)\ X$
⟨*proof*⟩

lemma *positive-mu-G*: $positive\ generator\ \mu G$
⟨*proof*⟩

lemma *additive-mu-G*: $additive\ generator\ \mu G$
⟨*proof*⟩

lemma *emeasure-lim*:
assumes *JX*: *finite J* $J \subseteq I$ $X \in$ *sets* (*PiM J M*)
assumes *cont*: $\bigwedge J\ X.$ $(\bigwedge i.\ J\ i \subseteq I) \Longrightarrow$ *incseq J* $\Longrightarrow$ $(\bigwedge i.$ *finite* (*J i*)$) \Longrightarrow (\bigwedge i.$ *X i* $\in$ *sets* (*PiM* (*J i*) *M*)$) \Longrightarrow$
decseq $(\lambda i.$ *emb I* (*J i*) (*X i*)$) \Longrightarrow 0 <$ (*INF i. P* (*J i*) (*X i*)$) \Longrightarrow (\bigcap i.$ *emb I* (*J i*) (*X i*)$) \neq \{\}$
shows *emeasure lim* (*emb I J X*) = *P J X*
⟨*proof*⟩

end

sublocale *product-prob-space* $\subseteq$ *projective-family I* $\lambda J.$ *PiM J M M*
⟨*proof*⟩

Proof due to Ionescu Tulcea.

locale *Ionescu-Tulcea* =
fixes *P* :: *nat* ⇒ (*nat* ⇒ *'a*) ⇒ *'a measure* **and** *M* :: *nat* ⇒ *'a measure*
assumes *P*[*measurable*]: $\bigwedge i.$ *P i* $\in$ *measurable* (*PiM* {*0..<i*} *M*) (*subprob-algebra* (*M i*))
assumes *prob-space-P*: $\bigwedge i\ x.$ *x* $\in$ *space* (*PiM* {*0..<i*} *M*) $\Longrightarrow$ *prob-space* (*P i x*)
begin

lemma *non-empty*[*simp*]: *space* (*M i*) $\neq \{\}$
⟨*proof*⟩

lemma *space-PiM-not-empty*[*simp*]: *space* (*PiM UNIV M*) $\neq \{\}$
⟨*proof*⟩

lemma *space-P*: *x* $\in$ *space* (*PiM* {*0..<n*} *M*) $\Longrightarrow$ *space* (*P n x*) = *space* (*M n*)
⟨*proof*⟩

lemma *sets-P*[*measurable-cong*]: *x* $\in$ *space* (*PiM* {*0..<n*} *M*) $\Longrightarrow$ *sets* (*P n x*) = *sets* (*M n*)
⟨*proof*⟩

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 $\in$ *measurable* (*PiM* {*0..< n*} *M*) (*subprob-algebra* (*PiM* {*0..<Suc n*} *M*))
⟨*proof*⟩

lemma *space-eP*:
x $\in$ *space* (*PiM* {*0..<n*} *M*) $\Longrightarrow$ *space* (*eP n x*) = *space* (*PiM* {*0..<Suc n*} *M*)
⟨*proof*⟩

lemma *sets-eP*[*measurable*]:
x $\in$ *space* (*PiM* {*0..<n*} *M*) $\Longrightarrow$ *sets* (*eP n x*) = *sets* (*PiM* {*0..<Suc n*} *M*)

⟨*proof*⟩

lemma *prob-space-eP*: $x \in space\ (PiM\ \{0..<n\}\ M) \Longrightarrow prob\text{-}space\ (eP\ n\ x)$
⟨*proof*⟩

lemma *nn-integral-eP*:
$\omega \in space\ (PiM\ \{0..<n\}\ M) \Longrightarrow f \in borel\text{-}measurable\ (PiM\ \{0..<Suc\ n\}\ M)$
$\Longrightarrow$
$(\int^+ x.\ f\ x\ \partial eP\ n\ \omega) = (\int^+ x.\ f\ (fun\text{-}upd\ \omega\ n\ x)\ \partial P\ n\ \omega)$
⟨*proof*⟩

lemma *emeasure-eP*:
assumes ω[*simp*]: $\omega \in space\ (PiM\ \{0..<n\}\ M)$ **and** A[*measurable*]: $A \in sets\ (PiM\ \{0..<Suc\ n\}\ M)$
shows $eP\ n\ \omega\ A = P\ n\ \omega\ ((\lambda x.\ fun\text{-}upd\ \omega\ n\ x) -` A \cap space\ (M\ n))$
⟨*proof*⟩

primrec $C :: nat \Rightarrow nat \Rightarrow (nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a)\ measure$ **where**
$C\ n\ 0\ \omega = return\ (PiM\ \{0..<n\}\ M)\ \omega$
| $C\ n\ (Suc\ m)\ \omega = C\ n\ m\ \omega \gg= eP\ (n + m)$

lemma *measurable-C*[*measurable*]:
$C\ n\ m \in measurable\ (PiM\ \{0..<n\}\ M)\ (subprob\text{-}algebra\ (PiM\ \{0..<n + m\}\ M))$
⟨*proof*⟩

lemma *space-C*:
$x \in space\ (PiM\ \{0..<n\}\ M) \Longrightarrow space\ (C\ n\ m\ x) = space\ (PiM\ \{0..<n + m\}\ M)$
⟨*proof*⟩

lemma *sets-C*[*measurable-cong*]:
$x \in space\ (PiM\ \{0..<n\}\ M) \Longrightarrow sets\ (C\ n\ m\ x) = sets\ (PiM\ \{0..<n + m\}\ M)$
⟨*proof*⟩

lemma *prob-space-C*: $x \in space\ (PiM\ \{0..<n\}\ M) \Longrightarrow prob\text{-}space\ (C\ n\ m\ x)$
⟨*proof*⟩

lemma *split-C*:
assumes ω: $\omega \in space\ (PiM\ \{0..<n\}\ M)$ **shows** $(C\ n\ m\ \omega \gg= C\ (n + m)\ l) = C\ n\ (m + l)\ \omega$
⟨*proof*⟩

lemma *nn-integral-C*:
assumes $m \le m'$ **and** f[*measurable*]: $f \in borel\text{-}measurable\ (PiM\ \{0..<n+m\}\ M)$
and *nonneg*: $\bigwedge x.\ x \in space\ (PiM\ \{0..<n+m\}\ M) \Longrightarrow 0 \le f\ x$
and x: $x \in space\ (PiM\ \{0..<n\}\ M)$
shows $(\int^+ x.\ f\ x\ \partial C\ n\ m\ x) = (\int^+ x.\ f\ (restrict\ x\ \{0..<n+m\})\ \partial C\ n\ m'\ x)$

⟨*proof*⟩

lemma *emeasure-C*:
assumes $m \leq m'$ **and** A[*measurable*]: $A \in sets\ (PiM\ \{0..<n+m\}\ M)$ **and** [*simp*]: $x \in space\ (PiM\ \{0..<n\}\ M)$
shows $emeasure\ (C\ n\ m'\ x)\ (prod\text{-}emb\ \{0..<n+m'\}\ M\ \{0..<n+m\}\ A) = emeasure\ (C\ n\ m\ x)\ A$
⟨*proof*⟩

lemma *distr-C*:
assumes $m \leq m'$ **and** [*simp*]: $x \in space\ (PiM\ \{0..<n\}\ M)$
shows $C\ n\ m\ x = distr\ (C\ n\ m'\ x)\ (PiM\ \{0..<n+m\}\ M)\ (\lambda x.\ restrict\ x\ \{0..<n+m\})$
⟨*proof*⟩

definition *up-to* :: *nat set* ⇒ *nat* **where**
$up\text{-}to\ J = (LEAST\ n.\ \forall i{\geq}n.\ i \notin J)$

lemma *up-to-less*: *finite* $J \Longrightarrow i \in J \Longrightarrow i < up\text{-}to\ J$
⟨*proof*⟩

lemma *up-to-iff*: *finite* $J \Longrightarrow up\text{-}to\ J \leq n \longleftrightarrow (\forall i{\in}J.\ i < n)$
⟨*proof*⟩

lemma *up-to-iff-Ico*: *finite* $J \Longrightarrow up\text{-}to\ J \leq n \longleftrightarrow J \subseteq \{0..<n\}$
⟨*proof*⟩

lemma *up-to*: *finite* $J \Longrightarrow J \subseteq \{0..<up\text{-}to\ J\}$
⟨*proof*⟩

lemma *up-to-mono*: $J \subseteq H \Longrightarrow$ *finite* $H \Longrightarrow up\text{-}to\ J \leq up\text{-}to\ H$
⟨*proof*⟩

definition *CI* :: *nat set* ⇒ (*nat* ⇒ *'a*) *measure* **where**
$CI\ J = distr\ (C\ 0\ (up\text{-}to\ J)\ (\lambda x.\ undefined))\ (PiM\ J\ M)\ (\lambda f.\ restrict\ f\ J)$

sublocale *PF*: *projective-family UNIV CI*
⟨*proof*⟩

lemma *emeasure-CI'*:
finite $J \Longrightarrow X \in sets\ (PiM\ J\ M) \Longrightarrow CI\ J\ X = C\ 0\ (up\text{-}to\ J)\ (\lambda\text{-}.\ undefined)\ (PF.emb\ \{0..<up\text{-}to\ J\}\ J\ X)$
⟨*proof*⟩

lemma *emeasure-CI*:
$J \subseteq \{0..<n\} \Longrightarrow X \in sets\ (PiM\ J\ M) \Longrightarrow CI\ J\ X = C\ 0\ n\ (\lambda\text{-}.\ undefined)\ (PF.emb\ \{0..<n\}\ J\ X)$
⟨*proof*⟩

lemma *lim*:
assumes *J*: *finite J* **and** *X*: $X \in sets\ (PiM\ J\ M)$
shows *emeasure PF.lim* $(PF.emb\ UNIV\ J\ X) = emeasure\ (CI\ J)\ X$
⟨*proof*⟩

lemma *distr-lim*: **assumes** *J*[*simp*]: *finite J* **shows** *distr PF.lim* $(PiM\ J\ M)\ (\lambda x.$ *restrict x J*$) = CI\ J$
⟨*proof*⟩

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*⟩

end

9 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* $J \subseteq I$
shows *distr* $(PiM\ I\ M)\ (PiM\ J\ M)\ (\lambda x.$ *restrict x J*$) = PiM\ J\ M$
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-PiM-emb′*:
$J \subseteq I \Longrightarrow$ *finite J* $\Longrightarrow X \in sets\ (PiM\ J\ M) \Longrightarrow emeasure\ (Pi_M\ I\ M)\ (emb\ I\ J\ X) = PiM\ J\ M\ X$
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-PiM-emb*:
$J \subseteq I \Longrightarrow$ *finite J* $\Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)) \Longrightarrow$
emeasure $(Pi_M\ I\ M)\ (emb\ I\ J\ (Pi_E\ J\ X)) = (\prod\ i \in J.\ emeasure\ (M\ i)\ (X\ i))$
⟨*proof*⟩

sublocale *product-prob-space* $\subseteq$ *P?*: *prob-space* $Pi_M\ I\ M$
⟨*proof*⟩

lemma *prob-space-PiM*:
assumes *M*: $\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* $(M\ i)$ **shows** *prob-space* $(PiM\ I\ M)$
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-PiM-Collect*:
assumes *X*: $J \subseteq I$ *finite J* $\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)$
shows *emeasure* $(Pi_M\ I\ M)\ \{x \in space\ (Pi_M\ I\ M).\ \forall\, i \in J.\ x\ i \in X\ i\} = (\prod\ i \in J.$

emeasure (*M i*) (*X i*))
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-PiM-Collect-single*:
assumes *X*: *i* ∈ *I A* ∈ *sets* (*M i*)
shows *emeasure* (*Pi*$_M$ *I M*) {*x*∈*space* (*Pi*$_M$ *I M*). *x i* ∈ *A*} = *emeasure* (*M i*) *A*
⟨*proof*⟩

lemma (**in** *product-prob-space*) *measure-PiM-emb*:
assumes *J* ⊆ *I finite J* ⋀*i*. *i* ∈ *J* ⟹ *X i* ∈ *sets* (*M i*)
shows *measure* (*PiM I M*) (*emb I J* (*Pi*$_E$ *J X*)) = (∏ *i*∈*J*. *measure* (*M i*) (*X i*))
⟨*proof*⟩

lemma *sets-Collect-single'*:
i ∈ *I* ⟹ {*x*∈*space* (*M i*). *P x*} ∈ *sets* (*M i*) ⟹ {*x*∈*space* (*PiM I M*). *P* (*x i*)} ∈ *sets* (*PiM I M*)
⟨*proof*⟩

lemma (**in** *finite-product-prob-space*) *finite-measure-PiM-emb*:
(⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *measure* (*PiM I M*) (*Pi*$_E$ *I A*) = (∏ *i*∈*I*. *measure* (*M i*) (*A i*))
⟨*proof*⟩

lemma (**in** *product-prob-space*) *PiM-component*:
assumes *i* ∈ *I*
shows *distr* (*PiM I M*) (*M i*) (λω. ω *i*) = *M i*
⟨*proof*⟩

lemma (**in** *product-prob-space*) *PiM-eq*:
assumes *M'*: *sets M'* = *sets* (*PiM I M*)
assumes *eq*: ⋀*J F*. *finite J* ⟹ *J* ⊆ *I* ⟹ (⋀*j*. *j* ∈ *J* ⟹ *F j* ∈ *sets* (*M j*)) ⟹
emeasure M' (*prod-emb I M J* (Π$_E$ *j*∈*J*. *F j*)) = (∏ *j*∈*J*. *emeasure* (*M j*) (*F j*))
shows *M'* = (*PiM I M*)
⟨*proof*⟩

lemma (**in** *product-prob-space*) *AE-component*: *i* ∈ *I* ⟹ *AE x in M i*. *P x* ⟹ *AE x in PiM I M*. *P* (*x i*)
⟨*proof*⟩

lemma *emeasure-PiM-emb*:
assumes *M*: ⋀*i*. *i* ∈ *I* ⟹ *prob-space* (*M i*)
assumes *J*: *J* ⊆ *I finite J* **and** *A*: ⋀*i*. *i* ∈ *J* ⟹ *A i* ∈ *sets* (*M i*)
shows *emeasure* (*Pi*$_M$ *I M*) (*prod-emb I M J* (*Pi*$_E$ *J A*)) = (∏ *i*∈*J*. *emeasure* (*M i*) (*A i*))
⟨*proof*⟩

lemma *distr-pair-PiM-eq-PiM*:

fixes i' :: $'i$ **and** I :: $'i$ *set* **and** M :: $'i \Rightarrow 'a$ *measure*
assumes M: $\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* $(M\ i)$ *prob-space* $(M\ i')$
shows *distr* $(M\ i' \bigotimes_M (\Pi_M\ i{\in}I.\ M\ i))\ (\Pi_M\ i{\in}insert\ i'\ I.\ M\ i)\ (\lambda(x,\ X).\ X(i' := x)) =$
$(\Pi_M\ i{\in}insert\ i'\ I.\ M\ i)$ (**is** *?L* = -)
⟨*proof*⟩

lemma *distr-PiM-reindex*:
assumes M: $\bigwedge i.\ i \in K \Longrightarrow$ *prob-space* $(M\ i)$
assumes f: *inj-on f I* $f \in I \to K$
shows *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*⟩

lemma *distr-PiM-component*:
assumes M: $\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* $(M\ i)$
assumes $i \in I$
shows *distr* $(Pi_M\ I\ M)\ (M\ i)\ (\lambda\omega.\ \omega\ i) = M\ i$
⟨*proof*⟩

lemma *AE-PiM-component*:
$(\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* $(M\ i)) \Longrightarrow i \in I \Longrightarrow$ *AE x in M i. P x* $\Longrightarrow$ *AE x in PiM I M. P* $(x\ i)$
⟨*proof*⟩

lemma *decseq-emb-PiE*:
incseq J $\Longrightarrow$ *decseq* $(\lambda i.$ *prod-emb I M* $(J\ i)\ (\Pi_E\ j{\in}J\ i.\ X\ j))$
⟨*proof*⟩

9.1 Sequence space

definition *comb-seq* :: *nat* $\Rightarrow$ $(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a)$ **where**
comb-seq i $\omega\ \omega'\ j =$ (*if* $j < i$ *then* $\omega\ j$ *else* $\omega'\ (j - i)$)

lemma *split-comb-seq*: P (*comb-seq i* $\omega\ \omega'\ j) \longleftrightarrow (j < i \longrightarrow P\ (\omega\ j)) \wedge (\forall k.\ j = i + k \longrightarrow P\ (\omega'\ k))$
⟨*proof*⟩

lemma *split-comb-seq-asm*: P (*comb-seq i* $\omega\ \omega'\ j) \longleftrightarrow \neg\ ((j < i \wedge \neg\ P\ (\omega\ j)) \vee (\exists k.\ j = i + k \wedge \neg\ P\ (\omega'\ k)))$
⟨*proof*⟩

lemma *measurable-comb-seq*:
$(\lambda(\omega, \omega').$ *comb-seq i* $\omega\ \omega') \in$ *measurable* $((\Pi_M\ i{\in}UNIV.\ M) \bigotimes_M (\Pi_M\ i{\in}UNIV.\ M))\ (\Pi_M\ i{\in}UNIV.\ M)$
⟨*proof*⟩

lemma *measurable-comb-seq'*[*measurable* (*raw*)]:

assumes f: $f \in$ *measurable* N (Π_M $i \in$ *UNIV*. M) **and** g: $g \in$ *measurable* N (Π_M $i \in$ *UNIV*. M)
shows (λx. *comb-seq* i (f x) (g x)) $\in$ *measurable* N (Π_M $i \in$ *UNIV*. M)
⟨*proof*⟩

lemma *comb-seq-0*: *comb-seq* 0 ω $\omega' = \omega'$
⟨*proof*⟩

lemma *comb-seq-Suc*: *comb-seq* (*Suc* n) ω $\omega' =$ *comb-seq* n ω (*case-nat* (ω n) ω')
⟨*proof*⟩

lemma *comb-seq-Suc-0*[*simp*]: *comb-seq* (*Suc* 0) $\omega =$ *case-nat* (ω 0)
⟨*proof*⟩

lemma *comb-seq-less*: $i < n \Longrightarrow$ *comb-seq* n ω ω' $i = \omega$ i
⟨*proof*⟩

lemma *comb-seq-add*: *comb-seq* n ω ω' (i + n) = ω' i
⟨*proof*⟩

lemma *case-nat-comb-seq*: *case-nat* s' (*comb-seq* n ω ω') (i + n) = *case-nat* (*case-nat* s' ω n) ω' i
⟨*proof*⟩

lemma *case-nat-comb-seq'*:
case-nat s (*comb-seq* i ω ω') = *comb-seq* (*Suc* i) (*case-nat* s ω) ω'
⟨*proof*⟩

locale *sequence-space* = *product-prob-space* λi. M *UNIV* :: *nat set* **for** M
begin

abbreviation $S \equiv \Pi_M$ $i \in$ *UNIV*::*nat set*. M

lemma *infprod-in-sets*[*intro*]:
fixes E :: *nat* ⇒ *'a set* **assumes** E: $\bigwedge i$. E $i \in$ *sets* M
shows *Pi UNIV* $E \in$ *sets* S
⟨*proof*⟩

lemma *measure-PiM-countable*:
fixes E :: *nat* ⇒ *'a set* **assumes** E: $\bigwedge i$. E $i \in$ *sets* M
shows (λn. $\prod i \leq n$. *measure* M (E i)) ⟶ *measure* S (*Pi UNIV* E)
⟨*proof*⟩

lemma *nat-eq-diff-eq*:
fixes a b c :: *nat*
shows $c \leq b \Longrightarrow a = b - c \longleftrightarrow a + c = b$
⟨*proof*⟩

lemma *PiM-comb-seq*:

distr $(S \bigotimes_M S)$ S $(\lambda(\omega, \omega').$ *comb-seq* i ω $\omega') = S$ (**is** *?D* = -)
⟨*proof*⟩

lemma *PiM-iter*:
distr $(M \bigotimes_M S)$ S $(\lambda(s, \omega).$ *case-nat* s $\omega) = S$ (**is** *?D* = -)
⟨*proof*⟩

end

lemma *PiM-return*:
assumes *finite I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow \{a\ i\} \in$ *sets* $(M\ i)$
shows *PiM I* $(\lambda i.$ *return* $(M\ i)$ $(a\ i)) =$ *return* $(PiM\ I\ M)$ (*restrict a I*)
⟨*proof*⟩

lemma *distr-PiM-finite-prob-space'*:
assumes *fin*: *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* $(M\ i)$
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* $(M'\ i)$
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow f \in$ *measurable* $(M\ i)$ $(M'\ i)$
shows *distr* $(PiM\ I\ M)$ $(PiM\ I\ M')$ $(compose\ I\ f) = PiM\ I$ $(\lambda i.$ *distr* $(M\ i)$ $(M'\ i)$ $f)$
⟨*proof*⟩

end

10 Independent families of events, event sets, and random variables

theory *Independent-Family*
imports *Infinite-Product-Measure*
begin

definition (**in** *prob-space*)
indep-sets $F\ I \longleftrightarrow (\forall i{\in}I.\ F\ i \subseteq$ *events*$) \wedge$
$(\forall J{\subseteq}I.\ J \neq \{\} \longrightarrow$ *finite* $J \longrightarrow (\forall A{\in}Pi\ J\ F.$ *prob* $(\bigcap j{\in}J.\ A\ j) = (\prod j{\in}J.$ *prob* $(A\ j))))$

definition (**in** *prob-space*)
indep-set $A\ B \longleftrightarrow$ *indep-sets* (*case-bool A B*) *UNIV*

definition (**in** *prob-space*)
indep-events-def-alt: *indep-events* $A\ I \longleftrightarrow$ *indep-sets* $(\lambda i.\ \{A\ i\})\ I$

lemma (**in** *prob-space*) *indep-events-def*:
indep-events $A\ I \longleftrightarrow (A`I \subseteq$ *events*$) \wedge$
$(\forall J{\subseteq}I.\ J \neq \{\} \longrightarrow$ *finite* $J \longrightarrow$ *prob* $(\bigcap j{\in}J.\ A\ j) = (\prod j{\in}J.$ *prob* $(A\ j)))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-eventsI*:
(⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *sets M*) ⟹ (⋀*J*. *J* ⊆ *I* ⟹ *finite J* ⟹ *J* ≠ {} ⟹ *prob* (⋂ *i*∈*J*. *F i*) = (∏ *i*∈*J*. *prob* (*F i*))) ⟹ *indep-events F I*
⟨*proof*⟩

definition (**in** *prob-space*)
indep-event A B ⟷ *indep-events* (*case-bool A B*) *UNIV*

lemma (**in** *prob-space*) *indep-sets-cong*:
I = *J* ⟹ (⋀*i*. *i* ∈ *I* ⟹ *F i* = *G i*) ⟹ *indep-sets F I* ⟷ *indep-sets G J*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-events-finite-index-events*:
indep-events F I ⟷ (∀ *J*⊆*I*. *J* ≠ {} ⟶ *finite J* ⟶ *indep-events F J*)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-finite-index-sets*:
indep-sets F I ⟷ (∀ *J*⊆*I*. *J* ≠ {} ⟶ *finite J* ⟶ *indep-sets F J*)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-mono-index*:
J ⊆ *I* ⟹ *indep-sets F I* ⟹ *indep-sets F J*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-mono-sets*:
assumes *indep*: *indep-sets F I*
assumes *mono*: ⋀*i*. *i*∈*I* ⟹ *G i* ⊆ *F i*
shows *indep-sets G I*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-mono*:
assumes *indep*: *indep-sets F I*
assumes *mono*: *J* ⊆ *I* ⋀*i*. *i*∈*J* ⟹ *G i* ⊆ *F i*
shows *indep-sets G J*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-setsI*:
assumes ⋀*i*. *i* ∈ *I* ⟹ *F i* ⊆ *events*
and ⋀*A J*. *J* ≠ {} ⟹ *J* ⊆ *I* ⟹ *finite J* ⟹ (∀ *j*∈*J*. *A j* ∈ *F j*) ⟹ *prob* (⋂ *j*∈*J*. *A j*) = (∏ *j*∈*J*. *prob* (*A j*))
shows *indep-sets F I*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-setsD*:
assumes *indep-sets F I* **and** *J* ⊆ *I J* ≠ {} *finite J* ∀ *j*∈*J*. *A j* ∈ *F j*
shows *prob* (⋂ *j*∈*J*. *A j*) = (∏ *j*∈*J*. *prob* (*A j*))
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-setI*:
assumes *ev*: $A \subseteq$ *events* $B \subseteq$ *events*
and *indep*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow$ *prob* $(a \cap b) =$ *prob* $a *$ *prob* b
shows *indep-set* $A\ B$
⟨*proof*⟩

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
⟨*proof*⟩

lemma (**in** *prob-space*)
assumes *indep*: *indep-set* $A\ B$
shows *indep-setD-ev1*: $A \subseteq$ *events*
and *indep-setD-ev2*: $B \subseteq$ *events*
⟨*proof*⟩

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*⟩

lemma (**in** *prob-space*) *indep-sets-sigma*:
assumes *indep*: *indep-sets* $F\ I$
assumes *stable*: $\bigwedge i.\ i \in I \Longrightarrow$ *Int-stable* $(F\ i)$
shows *indep-sets* $(\lambda i.$ *sigma-sets* $($*space* $M)$ $(F\ i))\ I$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-sigma-sets-iff*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *Int-stable* $(F\ i)$
shows *indep-sets* $(\lambda i.$ *sigma-sets* $($*space* $M)$ $(F\ i))\ I \longleftrightarrow$ *indep-sets* $F\ I$
⟨*proof*⟩

definition (**in** *prob-space*)
indep-vars-def2: *indep-vars* $M'\ X\ I \longleftrightarrow$
$(\forall i \in I.$ *random-variable* $(M'\ i)\ (X\ i)) \wedge$
indep-sets $(\lambda i.\ \{\ X\ i -` A \cap$ *space* $M \mid A.\ A \in$ *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 $M'\ X\ I \longleftrightarrow$
$(\forall i \in I.$ *random-variable* $(M'\ i)\ (X\ i)) \wedge$
indep-sets $(\lambda i.$ *sigma-sets* $($*space* $M)\ \{\ X\ i -` A \cap$ *space* $M \mid A.\ A \in$ *sets* $(M'$
$i)\})\ I$
⟨*proof*⟩

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*})
⟨*proof*⟩

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*. *prob* (*a* $\cap$ *b*) = *prob a* $*$ *prob b*)
⟨*proof*⟩

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*⟩

lemma (**in** *prob-space*) *indep-eventsI-indep-vars*:
assumes *indep*: *indep-vars N X I*
assumes *P*: $\bigwedge$*i*. *i* $\in$ *I* $\Longrightarrow$ {*x*$\in$*space* (*N i*). *P i x*} $\in$ *sets* (*N i*)
shows *indep-events* (λ*i*. {*x*$\in$*space M*. *P i* (*X i x*)}) *I*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-collect-sigma*:
fixes *I* :: *'j* $\Rightarrow$ *'i set* **and** *J* :: *'j set* **and** *E* :: *'i* $\Rightarrow$ *'a set set*
assumes *indep*: *indep-sets E* ($\bigcup$*j*$\in$*J*. *I j*)
assumes *Int-stable*: $\bigwedge$*i j*. *j* $\in$ *J* $\Longrightarrow$ *i* $\in$ *I j* $\Longrightarrow$ *Int-stable* (*E i*)
assumes *disjoint*: *disjoint-family-on I J*
shows *indep-sets* (λ*j*. *sigma-sets* (*space M*) ($\bigcup$ *i*$\in$*I j*. *E i*)) *J*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-restrict*:
assumes *ind*: *indep-vars M' X I* **and** *K*: $\bigwedge$*j*. *j* $\in$ *L* $\Longrightarrow$ *K j* $\subseteq$ *I* **and** *J*: *disjoint-family-on K L*
shows *indep-vars* (λ*j*. *PiM* (*K j*) *M'*) (λ*j* ω. *restrict* (λ*i*. *X i* ω) (*K j*)) *L*
⟨*proof*⟩

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* (*PiM A M'*) ($\lambda\omega$. *restrict* (λ*i*. *X i* ω) *A*) (*PiM B M'*) ($\lambda\omega$. *restrict* (λ*i*. *X i* ω) *B*)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-subset*:
assumes *indep-vars M' X I J* $\subseteq$ *I*
shows *indep-vars M' X J*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-cong*:
$I = J \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow X\ i = Y\ i) \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow M'\ i = N'\ i) \Longrightarrow$ *indep-vars* $M'\ X\ I \longleftrightarrow$ *indep-vars* $N'\ Y\ J$
⟨*proof*⟩

definition (**in** *prob-space*) *tail-events* **where**
tail-events $A = (\bigcap n.$ *sigma-sets* (*space* M) $(\bigcup\ (A\ `\ \{n..\})))$

lemma (**in** *prob-space*) *tail-events-sets*:
assumes A: $\bigwedge i$::*nat*. $A\ i \subseteq$ *events*
shows *tail-events* $A \subseteq$ *events*
⟨*proof*⟩

lemma (**in** *prob-space*) *sigma-algebra-tail-events*:
assumes $\bigwedge i$::*nat*. *sigma-algebra* (*space* M) $(A\ i)$
shows *sigma-algebra* (*space* M) (*tail-events* A)
⟨*proof*⟩

lemma (**in** *prob-space*) *kolmogorov-0-1-law*:
fixes A :: *nat* ⇒ *'a set set*
assumes $\bigwedge i$::*nat*. *sigma-algebra* (*space* M) $(A\ i)$
assumes *indep*: *indep-sets* A *UNIV*
and X: $X \in$ *tail-events* A
shows *prob* $X = 0 \vee$ *prob* $X = 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *borel-0-1-law*:
fixes F :: *nat* ⇒ *'a set*
assumes *F2*: *indep-events* F *UNIV*
shows *prob* $(\bigcap n.\ \bigcup m{\in}\{n..\}.\ F\ m) = 0 \vee$ *prob* $(\bigcap n.\ \bigcup m{\in}\{n..\}.\ F\ m) = 1$
⟨*proof*⟩

lemma (**in** *prob-space*) *borel-0-1-law-AE*:
fixes P :: *nat* ⇒ *'a* ⇒ *bool*
assumes *indep-events* $(\lambda m.\ \{x{\in}$*space* $M.\ P\ m\ x\})$ *UNIV* (**is** *indep-events ?P* -)
shows (*AE* x *in* M. *infinite* $\{m.\ P\ m\ x\}) \vee$ (*AE* x *in* M. *finite* $\{m.\ P\ m\ x\}$)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-finite*:
assumes I: $I \neq \{\}$ *finite* I
and F: $\bigwedge i.\ i \in I \Longrightarrow F\ i \subseteq$ *events* $\bigwedge i.\ i \in I \Longrightarrow$ *space* $M \in F\ i$
shows *indep-sets* $F\ I \longleftrightarrow (\forall A{\in}Pi\ I\ F.$ *prob* $(\bigcap j{\in}I.\ A\ j) = (\prod j{\in}I.$ *prob* $(A\ j)))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-finite*:
fixes I :: *'i set*
assumes I: $I \neq \{\}$ *finite* I

and M': $\bigwedge i.\ i \in I \Longrightarrow$ *sets* $(M'\ i) =$ *sigma-sets* (*space* $(M'\ i)$) $(E\ i)$
and *rv*: $\bigwedge i.\ i \in I \Longrightarrow$ *random-variable* $(M'\ i)\ (X\ i)$
and *Int-stable*: $\bigwedge i.\ i \in I \Longrightarrow$ *Int-stable* $(E\ i)$
and *space*: $\bigwedge i.\ i \in I \Longrightarrow$ *space* $(M'\ i) \in E\ i$ **and** *closed*: $\bigwedge i.\ i \in I \Longrightarrow E\ i \subseteq$ *Pow* (*space* $(M'\ i)$)
shows *indep-vars* $M'\ X\ I \longleftrightarrow$
$(\forall A \in (\Pi\ i \in I.\ E\ i).$ *prob* $(\bigcap j \in I.\ X\ j\ -`\ A\ j \cap$ *space* $M) = (\prod j \in I.$ *prob* $(X\ j\ -`\ A\ j \cap$ *space* $M)))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-compose*:
assumes *indep-vars* $M'\ X\ I$
assumes *rv*: $\bigwedge i.\ i \in I \Longrightarrow Y\ i \in$ *measurable* $(M'\ i)\ (N\ i)$
shows *indep-vars* $N\ (\lambda i.\ Y\ i \circ X\ i)\ I$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-compose2*:
assumes *indep-vars* $M'\ X\ I$
assumes *rv*: $\bigwedge i.\ i \in I \Longrightarrow Y\ i \in$ *measurable* $(M'\ i)\ (N\ i)$
shows *indep-vars* $N\ (\lambda i\ x.\ Y\ i\ (X\ i\ x))\ I$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-var-compose*:
assumes *indep-var M1 X1 M2 X2 Y1* ∈ *measurable M1 N1 Y2* ∈ *measurable M2 N2*
shows *indep-var N1* (*Y1* ∘ *X1*) *N2* (*Y2* ∘ *X2*)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-Min*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow$ *real*
assumes I: *finite* $I\ i \notin I$ **and** *indep*: *indep-vars* $(\lambda\text{-}.\ borel)\ X\ (insert\ i\ I)$
shows *indep-var borel* $(X\ i)$ *borel* $(\lambda\omega.\ Min\ ((\lambda i.\ X\ i\ \omega)\ `I))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-sum*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow$ *real*
assumes I: *finite* $I\ i \notin I$ **and** *indep*: *indep-vars* $(\lambda\text{-}.\ borel)\ X\ (insert\ i\ I)$
shows *indep-var borel* $(X\ i)$ *borel* $(\lambda\omega.\ \sum i \in I.\ X\ i\ \omega)$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-prod*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow$ *real*
assumes I: *finite* $I\ i \notin I$ **and** *indep*: *indep-vars* $(\lambda\text{-}.\ borel)\ X\ (insert\ i\ I)$
shows *indep-var borel* $(X\ i)$ *borel* $(\lambda\omega.\ \prod i \in I.\ X\ i\ \omega)$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-varsD-finite*:
assumes X: *indep-vars* $M'\ X\ I$
assumes I: $I \neq \{\}$ *finite* $I \bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ (M'\ i)$

shows *prob* $(\bigcap i{\in}I.\ X\ i\ -`\ A\ i \cap \mathit{space}\ M) = (\prod i{\in}I.\ \mathit{prob}\ (X\ i\ -`\ A\ i \cap \mathit{space}\ M))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-varsD*:
assumes *X*: *indep-vars M' X I*
assumes *I*: $J \neq \{\}$ *finite J* $J \subseteq I$ $\bigwedge i.\ i \in J \Longrightarrow A\ i \in \mathit{sets}\ (M'\ i)$
shows *prob* $(\bigcap i{\in}J.\ X\ i\ -`\ A\ i \cap \mathit{space}\ M) = (\prod i{\in}J.\ \mathit{prob}\ (X\ i\ -`\ A\ i \cap \mathit{space}\ M))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-iff-distr-eq-PiM*:
fixes $I :: 'i\ \mathit{set}$ **and** $X :: 'i \Rightarrow 'a \Rightarrow 'b$
assumes $I \neq \{\}$
assumes *rv*: $\bigwedge i.\ \mathit{random\text{-}variable}\ (M'\ i)\ (X\ i)$
shows *indep-vars M' X I* $\longleftrightarrow$
distr M $(\Pi_M\ i{\in}I.\ M'\ i)\ (\lambda x.\ \lambda i{\in}I.\ X\ i\ x) = (\Pi_M\ i{\in}I.\ \mathit{distr}\ M\ (M'\ i)\ (X\ i))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-iff-distr-eq-PiM'*:
fixes $I :: 'i\ \mathit{set}$ **and** $X :: 'i \Rightarrow 'a \Rightarrow 'b$
assumes $I \neq \{\}$
assumes *rv*: $\bigwedge i.\ i \in I \Longrightarrow \mathit{random\text{-}variable}\ (M'\ i)\ (X\ i)$
shows *indep-vars M' X I* $\longleftrightarrow$
distr M $(\Pi_M\ i{\in}I.\ M'\ i)\ (\lambda x.\ \lambda i{\in}I.\ X\ i\ x) = (\Pi_M\ i{\in}I.\ \mathit{distr}\ M\ (M'\ i)\ (X\ i))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-varD*:
assumes *indep*: *indep-var Ma A Mb B*
assumes *sets*: *Xa* $\in$ *sets Ma Xb* $\in$ *sets Mb*
shows *prob* $((\lambda x.\ (A\ x,\ B\ x)) -` (Xa \times Xb) \cap \mathit{space}\ M) =$
prob $(A -` Xa \cap \mathit{space}\ M) * \mathit{prob}\ (B -` Xb \cap \mathit{space}\ M)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-indep-random-variable*:
assumes *ind*[*simp*]: *indep-var N X N Y*
assumes [*simp*]: *A* $\in$ *sets N B* $\in$ *sets N*
shows $\mathcal{P}(x\ \mathit{in}\ M.\ X\ x \in A \wedge Y\ x \in B) = \mathcal{P}(x\ \mathit{in}\ M.\ X\ x \in A) * \mathcal{P}(x\ \mathit{in}\ M.\ Y\ x \in B)$
⟨*proof*⟩

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*⟩

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 $\bigotimes_M$ *distr M T Y* = *distr M* (*S* $\bigotimes_M$ *T*) ($\lambda x.$ (*X x*, *Y x*)) (**is** - $\longleftrightarrow$ - $\wedge$ - $\wedge$ *?S* $\bigotimes_M$ *?T* = *?J*)
⟨*proof*⟩

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* $\bigotimes_M$ *T*) ($\lambda x.$ (*X x*, *Y x*)) ($\lambda(x, y).$ *Px x* $*$ *Py y*)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-nn-integral*:
assumes *I*: *finite I indep-vars* (λ-. *borel*) *X I* $\bigwedge i\ \omega.\ i \in I \Longrightarrow 0 \leq X\ i\ \omega$
shows ($\int^+\omega.$ ($\prod i{\in}I.$ *X i* ω) ∂M) = ($\prod i{\in}I.$ $\int^+\omega.$ *X i* ω ∂M)
⟨*proof*⟩

lemma (**in** *prob-space*)
fixes *X* :: $'i \Rightarrow 'a \Rightarrow 'b$::{*real-normed-field*, *banach*, *second-countable-topology*}
assumes *I*: *finite I indep-vars* (λ-. *borel*) *X I* $\bigwedge i.\ i \in I \Longrightarrow$ *integrable M* (*X i*)
shows *indep-vars-lebesgue-integral*: ($\int \omega.$ ($\prod i{\in}I.$ *X i* ω) ∂M) = ($\prod i{\in}I.$ $\int \omega.$ *X i* ω ∂M) (**is** *?eq*)
and *indep-vars-integrable*: *integrable M* ($\lambda\omega.$ ($\prod i{\in}I.$ *X i* ω)) (**is** *?int*)
⟨*proof*⟩

lemma (**in** *prob-space*)
fixes *X1 X2* :: $'a \Rightarrow 'b$::{*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*: ($\int \omega.$ *X1* ω $*$ *X2* ω ∂M) = ($\int \omega.$ *X1* ω ∂M) $*$ ($\int \omega.$ *X2* ω ∂M) (**is** *?eq*)
and *indep-var-integrable*: *integrable M* ($\lambda\omega.$ *X1* ω $*$ *X2* ω) (**is** *?int*)
⟨*proof*⟩

end

11 Convolution Measure

theory *Convolution*
imports *Independent-Family*
begin

lemma (**in** *finite-measure*) *sigma-finite-measure*: *sigma-finite-measure M*
⟨*proof*⟩

definition *convolution* :: ($'a$ :: *ordered-euclidean-space*) *measure* $\Rightarrow$ $'a$ *measure* $\Rightarrow$ $'a$ *measure* (**infix** $\langle\star\rangle$ *50*) **where**
convolution M N = *distr* (*M* $\bigotimes_M$ *N*) *borel* ($\lambda(x, y).$ *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*
⟨*proof*⟩

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*⟩

lemma *convolution-emeasure*:
assumes *A* ∈ *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* ⋆ *N*) *A* = ∫⁺*x*. (*emeasure N* {*a*. *a* + *x* ∈ *A*}) ∂*M*
⟨*proof*⟩

lemma *convolution-emeasure′*:
assumes [*simp*]:*A* ∈ *sets borel*
assumes [*simp*]: *finite-measure M finite-measure N*
assumes [*simp*]: *sets N* = *sets borel sets M* = *sets borel*
shows *emeasure* (*M* ⋆ *N*) *A* = ∫⁺*x*. ∫⁺*y*. (*indicator A* (*x* + *y*)) ∂*N* ∂*M*
⟨*proof*⟩

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* ⋆ *N*)
⟨*proof*⟩

lemma *convolution-emeasure-3*:
assumes [*simp*, *measurable*]: *A* ∈ *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* ⋆ (*M* ⋆ *N*)) *A* = ∫⁺*x*. ∫⁺*y*. ∫⁺*z*. *indicator A* (*x* + *y* + *z*) ∂*N* ∂*M* ∂*L*
⟨*proof*⟩

lemma *convolution-emeasure-3′*:
assumes [*simp*, *measurable*]:*A* ∈ *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* ⋆ *M*) ⋆ *N*) *A* = ∫⁺*x*. ∫⁺*y*. ∫⁺*z*. *indicator A* (*x* + *y* + *z*)

∂N ∂M ∂L
⟨*proof*⟩

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*⟩

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 $(L \star (M \star N)) = ((L \star M) \star N)$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *convolution-density*:
fixes *f g* :: *real* ⇒ *ennreal*
assumes [*measurable*]: $f \in$ *borel-measurable borel* $g \in$ *borel-measurable borel*
assumes [*simp*]:*finite-measure* (*density lborel f*) *finite-measure* (*density lborel g*)
shows *density lborel f* ⋆ *density lborel g* = *density lborel* $(\lambda x.\ \int^+ y.\ f\ (x - y) * g\ y\ \partial lborel)$
(**is** *?l* = *?r*)
⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-finite-measure-density*:
distributed M N X f ⟹ *finite-measure* (*density N f*)
⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-convolution*:
fixes *f* :: *real* ⇒ -
fixes *g* :: *real* ⇒ -
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* (λ*x*. *X x* + *Y x*) (λ*x*. ∫⁺*y*. *f* (*x* − *y*) ∗ *g y* ∂*lborel*)
⟨*proof*⟩

lemma *prob-space-convolution-density*:
fixes *f*:: *real* ⇒ -
fixes *g*:: *real* ⇒ -
assumes [*measurable*]: *f*∈ *borel-measurable borel*
assumes [*measurable*]: *g*∈ *borel-measurable borel*
assumes *gt-0*[*simp*]: ⋀*x*. *0* ≤ *f x* ⋀*x*. *0* ≤ *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* (λ*x*.∫⁺*y*. *f* (*x* − *y*) ∗ *g y* ∂*lborel*)) (**is** *prob-space ?D*)
⟨*proof*⟩

end

12 Information theory

theory *Information*
imports
Independent-Family
begin

12.1 Information theory

locale *information-space* = *prob-space* +
fixes *b* :: *real* **assumes** *b-gt-1*: *1* < *b*

Introduce some simplification rules for logarithm of base *b*.

lemmas *log-simps* = *log-mult log-inverse log-divide*

12.2 Kullback−Leibler divergence

The Kullback−Leibler divergence is also known as relative entropy or Kullback−Leibler distance.

definition
entropy-density b M N = *log b* ∘ *enn2real* ∘ *RN-deriv M N*

definition
KL-divergence b M N = *integral*^{L} *N* (*entropy-density b M N*)

lemma *measurable-entropy-density*[*measurable*]: *entropy-density b M N* ∈ *borel-measurable M*
⟨*proof*⟩

lemma (**in** *sigma-finite-measure*) *KL-density*:
fixes $f :: 'a \Rightarrow real$
assumes $1 < b$
assumes f[*measurable*]: $f \in$ *borel-measurable* M **and** *nn*: *AE* x *in* M. $0 \leq f\ x$
shows *KL-divergence* $b\ M$ (*density* $M\ f$) $= (\int x.\ f\ x * log\ b\ (f\ x)\ \partial M)$
⟨*proof*⟩

lemma (**in** *sigma-finite-measure*) *KL-density-density*:
fixes $f\ g :: 'a \Rightarrow real$
assumes $1 < b$
assumes f: $f \in$ *borel-measurable* M *AE* x *in* M. $0 \leq f\ x$
assumes g: $g \in$ *borel-measurable* M *AE* x *in* M. $0 \leq g\ x$
assumes *ac*: *AE* x *in* M. $f\ x = 0 \longrightarrow g\ x = 0$
shows *KL-divergence* b (*density* $M\ f$) (*density* $M\ g$) $= (\int x.\ g\ x * log\ b\ (g\ x\ /\ f\ x)\ \partial M)$
⟨*proof*⟩

lemma (**in** *information-space*) *KL-gt-0*:
fixes $D :: 'a \Rightarrow real$
assumes *prob-space* (*density* $M\ D$)
assumes D: $D \in$ *borel-measurable* M *AE* x *in* M. $0 \leq D\ x$
assumes *int*: *integrable* M $(\lambda x.\ D\ x * log\ b\ (D\ x))$
assumes A: *density* $M\ D \neq M$
shows $0 <$ *KL-divergence* $b\ M$ (*density* $M\ D$)
⟨*proof*⟩

lemma (**in** *sigma-finite-measure*) *KL-same-eq-0*: *KL-divergence* $b\ M\ M = 0$
⟨*proof*⟩

lemma (**in** *information-space*) *KL-eq-0-iff-eq*:
fixes $D :: 'a \Rightarrow real$
assumes *prob-space* (*density* $M\ D$)
assumes D: $D \in$ *borel-measurable* M *AE* x *in* M. $0 \leq D\ x$
assumes *int*: *integrable* M $(\lambda x.\ D\ x * log\ b\ (D\ x))$
shows *KL-divergence* $b\ M$ (*density* $M\ D$) $= 0 \longleftrightarrow$ *density* $M\ D = M$
⟨*proof*⟩

lemma (**in** *information-space*) *KL-eq-0-iff-eq-ac*:
fixes $D :: 'a \Rightarrow real$
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* $b\ M\ N = 0 \longleftrightarrow N = M$
⟨*proof*⟩

lemma (**in** *information-space*) *KL-nonneg*:
assumes *prob-space* (*density* $M\ D$)
assumes D: $D \in$ *borel-measurable* M *AE* x *in* M. $0 \leq D\ x$
assumes *int*: *integrable* M $(\lambda x.\ D\ x * log\ b\ (D\ x))$

shows $0 \leq$ *KL-divergence b M* (*density M D*)
⟨*proof*⟩

lemma (**in** *sigma-finite-measure*) *KL-density-density-nonneg*:
fixes $f\ g :: 'a \Rightarrow$ *real*
assumes $1 < b$
assumes *f*: $f \in$ *borel-measurable M AE x in M*. $0 \leq f\ x$ *prob-space* (*density M f*)
assumes *g*: $g \in$ *borel-measurable M AE x in M*. $0 \leq g\ x$ *prob-space* (*density M g*)
assumes *ac*: *AE x in M*. $f\ x = 0 \longrightarrow g\ x = 0$
assumes *int*: *integrable M* (λx. $g\ x *$ *log b* ($g\ x$ / $f\ x$))
shows $0 \leq$ *KL-divergence b* (*density M f*) (*density M g*)
⟨*proof*⟩

12.3 Finite Entropy

definition (**in** *information-space*) *finite-entropy* :: *'b measure* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *real*) ⇒ *bool*
where
finite-entropy S X f ⟷
distributed M S X f ∧
integrable S (λx. $f\ x *$ *log b* ($f\ x$)) ∧
($\forall\, x \in$*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* (λa. *measure M* {$x \in$ *space M*. *X x* = *a*})
⟨*proof*⟩

lemma *ac-fst*:
assumes *sigma-finite-measure T*
shows *absolutely-continuous S* (*distr* ($S \bigotimes_M T$) *S fst*)
⟨*proof*⟩

lemma *ac-snd*:
assumes *sigma-finite-measure T*
shows *absolutely-continuous T* (*distr* ($S \bigotimes_M T$) *T snd*)
⟨*proof*⟩

lemma (**in** *information-space*) *finite-entropy-integrable*:
finite-entropy S X Px ⟹ *integrable S* (λx. *Px x* $*$ *log b* (*Px x*))
⟨*proof*⟩

lemma (**in** *information-space*) *finite-entropy-distributed*:
finite-entropy S X Px ⟹ *distributed M S X Px*
⟨*proof*⟩

lemma (**in** *information-space*) *finite-entropy-nn*:
finite-entropy S X Px $\Longrightarrow$ $x \in$ *space S* $\Longrightarrow$ $0 \leq$ *Px x*
⟨*proof*⟩

lemma (**in** *information-space*) *finite-entropy-measurable*:
finite-entropy S X Px $\Longrightarrow$ *Px* $\in$ *S* $\rightarrow_M$ *borel*
⟨*proof*⟩

lemma (**in** *information-space*) *subdensity-finite-entropy*:
fixes *g* :: *'b* $\Rightarrow$ *real* **and** *f* :: *'c* $\Rightarrow$ *real*
assumes *T*: *T* $\in$ *measurable P Q*
assumes *f*: *finite-entropy P X f*
assumes *g*: *finite-entropy Q Y g*
assumes *Y*: *Y* = *T* $\circ$ *X*
shows *AE x in P. g* (*T x*) = 0 $\longrightarrow$ *f x* = 0
⟨*proof*⟩

lemma (**in** *information-space*) *finite-entropy-integrable-transform*:
finite-entropy S X Px $\Longrightarrow$ *distributed M T Y Py* $\Longrightarrow$ ($\bigwedge$*x. x* $\in$ *space T* $\Longrightarrow$ 0 $\leq$ *Py x*) $\Longrightarrow$
X = (λ*x. f* (*Y x*)) $\Longrightarrow$ *f* $\in$ *measurable T S* $\Longrightarrow$ *integrable T* (λ*x. Py x* $*$ *log b* (*Px* (*f x*)))
⟨*proof*⟩

12.4 Mutual Information

definition (**in** *prob-space*)
mutual-information b S T X Y =
KL-divergence b (*distr M S X* $\bigotimes_M$ *distr M T Y*) (*distr M* (*S* $\bigotimes_M$ *T*) (λ*x.* (*X x*, *Y x*)))

lemma (**in** *information-space*) *mutual-information-indep-vars*:
fixes *S T X Y*
defines *P* $\equiv$ *distr M S X* $\bigotimes_M$ *distr M T Y*
defines *Q* $\equiv$ *distr M* (*S* $\bigotimes_M$ *T*) (λ*x.* (*X x*, *Y x*))
shows *indep-var S X T Y* $\longleftrightarrow$
(*random-variable S X* $\wedge$ *random-variable T Y* $\wedge$
absolutely-continuous P Q $\wedge$ *integrable Q* (*entropy-density b P Q*) $\wedge$
mutual-information b S T X Y = 0)
⟨*proof*⟩

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* $\Rightarrow$ *real* **and** *Px* :: *'b* $\Rightarrow$ *real* **and** *Py* :: *'c* $\Rightarrow$ *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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *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* = $integral^L$ $(S \bigotimes_M T)\ f$ (**is** *?M* = *?R*)
and *mutual-information-nonneg'*: $0 \leq$ *mutual-information b S T X Y*
⟨*proof*⟩

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*: $\bigwedge x.\ x \in space\ S \Longrightarrow 0 \leq Px\ x$
and *Py*: *distributed M T Y Py* **and** *Py-nn*: $\bigwedge y.\ y \in space\ T \Longrightarrow 0 \leq Py\ y$
and *Pxy*: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*: $\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \leq Pxy\ (x,\ y)$
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* = $integral^L$ $(S \bigotimes_M T)\ f$ (**is** *?M* = *?R*)
and *mutual-information-nonneg*: *integrable* $(S \bigotimes_M T)\ f \Longrightarrow 0 \leq$ *mutual-information b S T X Y*
⟨*proof*⟩

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*: $\bigwedge x.\ x \in space\ S \Longrightarrow 0 \leq Px\ x$
and *Py*[*measurable*]: *distributed M T Y Py* **and** *Py-nn*: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
and *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*: $\bigwedge x.\ x \in space\ (S \bigotimes_M T) \Longrightarrow 0 \leq Pxy\ x$
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*⟩

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 $\mathcal{I}(X\ ;\ Y) = (\sum (x,\ y) \in (\lambda x.\ (X\ x,\ Y\ x))\text{`}space\ M.\ Pxy\ (x,\ y) * log\ b\ (Pxy\ (x,\ y)\ /\ (Px\ x * Py\ y)))$
⟨*proof*⟩

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*: $\forall x \in space\ M.\ Pxy\ (X\ x,\ Y\ x) = Px\ (X\ x) * Py\ (Y\ x)$
shows *mutual-information-eq-0-simple*: $\mathcal{I}(X\ ;\ Y) = 0$
⟨*proof*⟩

12.5 Entropy

definition (**in** *prob-space*) *entropy* :: *real* ⇒ *'b measure* ⇒ (*'a* ⇒ *'b*) ⇒ *real* **where**
entropy b S X = − *KL-divergence b S* (*distr M S X*)

abbreviation (**in** *information-space*)
entropy-Pow (‹ℋ'(-')›) **where**
ℋ(*X*) ≡ *entropy b* (*count-space* (*X*‘*space M*)) *X*

lemma (**in** *prob-space*) *distributed-RN-deriv*:
assumes *X*: *distributed M S X Px*
shows *AE x in S*. *RN-deriv S* (*density S Px*) *x* = *Px x*
⟨*proof*⟩

lemma (**in** *information-space*)
fixes *X* :: *'a* ⇒ *'b*
assumes *X*[*measurable*]: *distributed M MX X f* **and** *nn*: ⋀*x*. *x* ∈ *space MX* ⟹ 0 ≤ *f x*
shows *entropy-distr*: *entropy b MX X* = − (∫ *x*. *f x* ∗ *log b* (*f x*) ∂*MX*) (**is** *?eq*)
⟨*proof*⟩

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*]: ⋀*x*. *x* ∈ *space MX* ⟹ 0 ≤ *Px x*
and *fin*: *emeasure MX* {*x* ∈ *space MX*. *Px x* ≠ 0} ≠ *top*
and *int*: *integrable MX* (λ*x*. − *Px x* ∗ *log b* (*Px x*))
shows *entropy b MX X* ≤ *log b* (*measure MX* {*x* ∈ *space MX*. *Px x* ≠ 0})
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-le-space*:
fixes *Px* :: *'b* ⇒ *real* **and** *MX* :: *'b measure*
assumes *X*: *distributed M MX X Px* **and** *Px-nn*[*simp*]: ⋀*x*. *x* ∈ *space MX* ⟹ 0 ≤ *Px x*
and *fin*: *finite-measure MX*
and *int*: *integrable MX* (λ*x*. − *Px x* ∗ *log b* (*Px x*))
shows *entropy b MX X* ≤ *log b* (*measure MX* (*space MX*))
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-uniform*:
assumes *X*: *distributed M MX X* (λ*x*. *indicator A x* / *measure MX A*) (**is** *distributed - - - ?f*)
shows *entropy b MX X* = *log b* (*measure MX A*)
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-simple-distributed*:
simple-distributed M X f ⟹ ℋ(*X*) = − (∑ *x*∈*X*‘*space M*. *f x* ∗ *log b* (*f x*))
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-le-card-not-0*:

assumes X: *simple-distributed M X f*
shows $\mathcal{H}(X) \leq$ *log b* (*card* (X ‘ *space M* $\cap$ $\{x.\ f\ x \neq 0\}$))
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-le-card*:
assumes X: *simple-distributed M X f*
shows $\mathcal{H}(X) \leq$ *log b* (*real* (*card* (X ‘ *space M*)))
⟨*proof*⟩

12.6 Conditional Mutual Information

definition (**in** *prob-space*)
conditional-mutual-information b MX MY MZ X Y Z ≡
mutual-information b MX (*MY* $\bigotimes_M$ *MZ*) *X* (λ*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*) ≡ *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*]: $\bigwedge x.\ x \in$ *space S* $\Longrightarrow$ *0* $\leq$ *Px x*
assumes *Pz*[*measurable*]: *distributed M P Z Pz*
and *Pz-nn*[*simp*]: $\bigwedge z.\ z \in$ *space P* $\Longrightarrow$ *0* $\leq$ *Pz z*
assumes *Pyz*[*measurable*]: *distributed M* (*T* $\bigotimes_M$ *P*) (λ*x*. (*Y x*, *Z x*)) *Pyz*
and *Pyz-nn*[*simp*]: $\bigwedge y\ z.\ y \in$ *space T* $\Longrightarrow$ $z \in$ *space P* $\Longrightarrow$ *0* $\leq$ *Pyz* (*y*, *z*)
assumes *Pxz*[*measurable*]: *distributed M* (*S* $\bigotimes_M$ *P*) (λ*x*. (*X x*, *Z x*)) *Pxz*
and *Pxz-nn*[*simp*]: $\bigwedge x\ z.\ x \in$ *space S* $\Longrightarrow$ $z \in$ *space P* $\Longrightarrow$ *0* $\leq$ *Pxz* (*x*, *z*)
assumes *Pxyz*[*measurable*]: *distributed M* (*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*) (λ*x*. (*X x*, *Y x*, *Z x*)) *Pxyz*
and *Pxyz-nn*[*simp*]: $\bigwedge x\ y\ z.\ x \in$ *space S* $\Longrightarrow$ $y \in$ *space T* $\Longrightarrow$ $z \in$ *space P* $\Longrightarrow$ *0* $\leq$ *Pxyz* (*x*, *y*, *z*)
assumes *I1*: *integrable* (*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*) (λ(*x*, *y*, *z*). *Pxyz* (*x*, *y*, *z*) ∗ *log b* (*Pxyz* (*x*, *y*, *z*) / (*Px x* ∗ *Pyz* (*y*, *z*))))
assumes *I2*: *integrable* (*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*) (λ(*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*))) ∂(*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*)) (**is** *?eq*)
and *conditional-mutual-information-generic-nonneg*: *0* $\leq$ *conditional-mutual-information b S T P X Y Z* (**is** *?nonneg*)
⟨*proof*⟩

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* $(T \bigotimes_M P)$ $(\lambda x.\ (Y\ x,\ Z\ x))$ *Pyz*
assumes *Fxz*: *finite-entropy* $(S \bigotimes_M P)$ $(\lambda x.\ (X\ x,\ Z\ x))$ *Pxz*
assumes *Fxyz*: *finite-entropy* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda x.\ (X\ x,\ Y\ x,\ Z\ x))$ *Pxyz*
shows *conditional-mutual-information-generic-eq'*: *conditional-mutual-information b S T P X Y Z*
$= (\int (x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxyz\ (x,\ y,\ z)\ /\ (Pxz\ (x,\ z) * (Pyz\ (y,z)\ /\ Pz\ z)))\ \partial(S \bigotimes_M T \bigotimes_M P))$ (**is** *?eq*)
and *conditional-mutual-information-generic-nonneg'*: $0 \le$ *conditional-mutual-information b S T P X Y Z* (**is** *?nonneg*)
⟨*proof*⟩

lemma (**in** *information-space*) *conditional-mutual-information-eq*:
assumes *Pz*: *simple-distributed M Z Pz*
assumes *Pyz*: *simple-distributed M* $(\lambda x.\ (Y\ x,\ Z\ x))$ *Pyz*
assumes *Pxz*: *simple-distributed M* $(\lambda x.\ (X\ x,\ Z\ x))$ *Pxz*
assumes *Pxyz*: *simple-distributed M* $(\lambda x.\ (X\ x,\ Y\ x,\ Z\ x))$ *Pxyz*
shows $\mathcal{I}(X\ ;\ Y\ |\ Z) =$
$(\sum (x,\ y,\ z) \in (\lambda x.\ (X\ x,\ Y\ x,\ Z\ x))\text{`space } M.\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxyz\ (x,\ y,\ z)\ /\ (Pxz\ (x,\ z) * (Pyz\ (y,z)\ /\ Pz\ z))))$
⟨*proof*⟩

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 $0 \le \mathcal{I}(X\ ;\ Y\ |\ Z)$
⟨*proof*⟩

12.7 Conditional Entropy

definition (**in** *prob-space*)
conditional-entropy b S T X Y = $-$ $(\int (x,\ y).\ log\ b\ (enn2real\ (RN\text{-}deriv\ (S \bigotimes_M T)\ (distr\ M\ (S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x)))\ (x,\ y))\ /$
enn2real (*RN-deriv T* (*distr M T Y*) *y*)) *∂distr M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x)))$

abbreviation (**in** *information-space*)
conditional-entropy-Pow (‹$\mathcal{H}'$(- | -')›) **where**
$\mathcal{H}(X\ |\ Y) \equiv$ *conditional-entropy b* (*count-space* (*X*‘*space M*)) (*count-space* (*Y*‘*space M*)) *X Y*

lemma (**in** *information-space*) *conditional-entropy-generic-eq*:
fixes *Pxy* :: - ⇒ *real* **and** *Py* :: *'c* ⇒ *real*
assumes *S*: *sigma-finite-measure S* **and** *T*: *sigma-finite-measure T*

assumes *Py*[*measurable*]: *distributed M T Y Py* **and** *Py-nn*[*simp*]: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
assumes *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*[*simp*]: $\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \leq Pxy\ (x,\ y)$
shows *conditional-entropy b S T X Y* $= -\ (\int (x,\ y).\ Pxy\ (x,\ y) * log\ b\ (Pxy\ (x,\ y)\ /\ Py\ y)\ \partial(S \bigotimes_M T))$
⟨*proof*⟩

lemma (**in** *information-space*) *conditional-entropy-eq-entropy*:
fixes $Px :: 'b \Rightarrow real$ **and** $Py :: 'c \Rightarrow real$
assumes *S*: *sigma-finite-measure S* **and** *T*: *sigma-finite-measure T*
assumes *Py*[*measurable*]: *distributed M T Y Py*
and *Py-nn*[*simp*]: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
assumes *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*[*simp*]: $\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \leq Pxy\ (x,\ y)$
assumes *I1*: *integrable* $(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Pxy\ x))$
assumes *I2*: *integrable* $(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Py\ (snd\ x)))$
shows *conditional-entropy b S T X Y* $=$ *entropy b* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ $-$ *entropy b T Y*
⟨*proof*⟩

lemma (**in** *information-space*) *conditional-entropy-eq-entropy-simple*:
assumes *X*: *simple-function M X* **and** *Y*: *simple-function M Y*
shows $\mathcal{H}(X \mid Y) =$ *entropy b* (*count-space* (*X*'*space M*) $\bigotimes_M$ *count-space* (*Y*'*space M*)) $(\lambda x.\ (X\ x,\ Y\ x))$ $-\ \mathcal{H}(Y)$
⟨*proof*⟩

lemma (**in** *information-space*) *conditional-entropy-eq*:
assumes *Y*: *simple-distributed M Y Py*
assumes *XY*: *simple-distributed M* $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
shows $\mathcal{H}(X \mid Y) = -\ (\sum (x,\ y) \in (\lambda x.\ (X\ x,\ Y\ x))\ `\ space\ M.\ Pxy\ (x,\ y) * log\ b\ (Pxy\ (x,\ y)\ /\ Py\ y))$
⟨*proof*⟩

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*⟩

lemma (**in** *information-space*) *conditional-entropy-nonneg*:
assumes *X*: *simple-function M X* **and** *Y*: *simple-function M Y* **shows** $0 \leq \mathcal{H}(X \mid Y)$
⟨*proof*⟩

12.8 Equalities

lemma (**in** *information-space*) *mutual-information-eq-entropy-conditional-entropy-distr*:
fixes $Px :: 'b \Rightarrow real$ **and** $Py :: 'c \Rightarrow real$ **and** $Pxy :: ('b \times 'c) \Rightarrow real$
assumes *S*: *sigma-finite-measure S* **and** *T*: *sigma-finite-measure T*

assumes *Px*[*measurable*]: *distributed M S X Px*
and *Px-nn*[*simp*]: $\bigwedge x.\ x \in space\ S \Longrightarrow 0 \leq Px\ x$
and *Py*[*measurable*]: *distributed M T Y Py*
and *Py-nn*[*simp*]: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
and *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*[*simp*]: $\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \leq Pxy\ (x,\ y)$
assumes *Ix*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Px\ (fst\ x)))$
assumes *Iy*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Py\ (snd\ x)))$
assumes *Ixy*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Pxy\ x))$
shows *mutual-information b S T X Y* = *entropy b S X* + *entropy b T Y* − *entropy b* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$
⟨*proof*⟩

lemma (**in** *information-space*) *mutual-information-eq-entropy-conditional-entropy′*:
fixes *Px* :: $'b \Rightarrow real$ **and** *Py* :: $'c \Rightarrow real$ **and** *Pxy* :: $('b \times 'c) \Rightarrow real$
assumes *S*: *sigma-finite-measure S* **and** *T*: *sigma-finite-measure T*
assumes *Px*: *distributed M S X Px* $\bigwedge x.\ x \in space\ S \Longrightarrow 0 \leq Px\ x$
and *Py*: *distributed M T Y Py* $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
assumes *Pxy*: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
$\bigwedge x.\ x \in space\ (S \bigotimes_M T) \Longrightarrow 0 \leq Pxy\ x$
assumes *Ix*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Px\ (fst\ x)))$
assumes *Iy*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Py\ (snd\ x)))$
assumes *Ixy*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Pxy\ x))$
shows *mutual-information b S T X Y* = *entropy b S X* − *conditional-entropy b S T X Y*
⟨*proof*⟩

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 $\mathcal{I}(X\ ;\ Y) = \mathcal{H}(X) - \mathcal{H}(X \mid Y)$
⟨*proof*⟩

lemma (**in** *information-space*) *mutual-information-nonneg-simple*:
assumes *sf-X*: *simple-function M X* **and** *sf-Y*: *simple-function M Y*
shows $0 \leq \mathcal{I}(X\ ;\ Y)$
⟨*proof*⟩

lemma (**in** *information-space*) *conditional-entropy-less-eq-entropy*:
assumes *X*: *simple-function M X* **and** *Z*: *simple-function M Z*
shows $\mathcal{H}(X \mid Z) \leq \mathcal{H}(X)$
⟨*proof*⟩

lemma (**in** *information-space*)
fixes *Px* :: $'b \Rightarrow real$ **and** *Py* :: $'c \Rightarrow real$ **and** *Pxy* :: $('b \times 'c) \Rightarrow real$
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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
shows *conditional-entropy b S T X Y* ≤ *entropy b S X*
⟨*proof*⟩

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*⟩

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*⟩

corollary (**in** *information-space*) *entropy-data-processing*:
assumes *simple-function M X* **shows** $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$
⟨*proof*⟩

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*⟩

end

13 Properties of Various Distributions

theory *Distributions*
imports *Convolution Information*
begin

lemma (**in** *prob-space*) *distributed-affine*:
fixes f :: *real* ⇒ *ennreal*
assumes f: *distributed M lborel X f*
assumes c: $c \neq 0$
shows *distributed M lborel* $(\lambda x.\ t + c * X\ x)$ $(\lambda x.\ f\ ((x - t)\ /\ c)\ /\ |c|)$
⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-affineI*:
fixes f :: *real* ⇒ *ennreal* **and** c :: *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*⟩

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*⟩

13.1 Erlang

lemma *nn-intergal-power-times-exp-Icc*:
assumes [*arith*]: $0 \leq a$
shows $(\int^+ x.\ ennreal\ (x\widehat{}k * exp\ (-x)) * indicator\ \{0\ ..\ a\}\ x\ \partial lborel) =$
$(1 - (\sum n{\leq}k.\ (a\widehat{}n * exp\ (-a))\ /\ fact\ n)) * fact\ k$ (**is** *?I* = -)
⟨*proof*⟩

lemma *nn-intergal-power-times-exp-Ici*:
shows $(\int^+ x.\ ennreal\ (x\widehat{}k * exp\ (-x)) * indicator\ \{0\ ..\}\ x\ \partial lborel) = real\text{-}of\text{-}nat\ (fact\ k)$
⟨*proof*⟩

definition *erlang-density* :: *nat* ⇒ *real* ⇒ *real* ⇒ *real* **where**
erlang-density k l x = (*if x* < *0 then 0 else* $(l\widehat{}(Suc\ k) * x\widehat{}k * exp\ (-\ l * x))\ /\ fact\ k$)

definition *erlang-CDF* :: *nat* ⇒ *real* ⇒ *real* ⇒ *real* **where**
erlang-CDF k l x = (*if x* < *0 then 0 else 1* − $(\sum n{\leq}k.\ ((l * x)\widehat{}n * exp\ (-\ l * x)\ /\ fact\ n))$)

lemma *erlang-density-nonneg*[*simp*]: $0 \leq l \Longrightarrow 0 \leq erlang\text{-}density\ k\ l\ x$
⟨*proof*⟩

lemma *borel-measurable-erlang-density*[*measurable*]: *erlang-density k l* ∈ *borel-measurable borel*
⟨*proof*⟩

lemma *erlang-CDF-transform*: $0 < l \Longrightarrow erlang\text{-}CDF\ k\ l\ a = erlang\text{-}CDF\ k\ 1\ (l * a)$
⟨*proof*⟩

lemma *erlang-CDF-nonneg*[*simp*]: **assumes** *0* < *l* **shows** *0* ≤ *erlang-CDF k l x*
⟨*proof*⟩

lemma *nn-integral-erlang-density*:
assumes [*arith*]: *0* < *l*
shows $(\int^+ x.\ ennreal\ (erlang\text{-}density\ k\ l\ x) * indicator\ \{..\ a\}\ x\ \partial lborel) = erlang\text{-}CDF\ k\ l\ a$
⟨*proof*⟩

lemma *emeasure-erlang-density*:
0 < *l* ⟹ *emeasure* (*density lborel* (*erlang-density k l*)) {.. *a*} = *erlang-CDF k l a*
⟨*proof*⟩

lemma *nn-integral-erlang-ith-moment*:
fixes *k i* :: *nat* **and** *l* :: *real*
assumes [*arith*]: *0* < *l*
shows $(\int^+ x.\ ennreal\ (erlang\text{-}density\ k\ l\ x * x\ \widehat{}\ i)\ \partial lborel) = fact\ (k + i)\ /$

(*fact k * l ^ i*)
⟨*proof*⟩

lemma *prob-space-erlang-density*:
assumes *l*[*arith*]: *0 < l*
shows *prob-space* (*density lborel* (*erlang-density k l*)) (**is** *prob-space ?D*)
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-distributed-le*:
assumes *D*: *distributed M lborel X* (*erlang-density k l*)
assumes [*simp*, *arith*]: *0 < l 0 ≤ a*
shows $\mathcal{P}$(*x in M. X x ≤ a*) = *erlang-CDF k l a*
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-distributed-gt*:
assumes *D*[*simp*]: *distributed M lborel X* (*erlang-density k l*)
assumes [*arith*]: *0 < l 0 ≤ a*
shows $\mathcal{P}$(*x in M. a < X x*) = *1 −* (*erlang-CDF k l a*)
⟨*proof*⟩

lemma *erlang-CDF-at0*: *erlang-CDF k l 0 = 0*
⟨*proof*⟩

lemma *erlang-distributedI*:
assumes *X*[*measurable*]: *X ∈ borel-measurable M* **and** [*arith*]: *0 < l*
and *X-distr*: ⋀*a. 0 ≤ a ⟹ emeasure M* {*x*∈*space M. X x ≤ a*} = *erlang-CDF k l a*
shows *distributed M lborel X* (*erlang-density k l*)
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-distributed-iff*:
assumes [*arith*]: *0*<*l*
shows *distributed M lborel X* (*erlang-density k l*) ⟷
(*X ∈ borel-measurable M ∧ 0 < l ∧* (∀ *a*≥*0*. $\mathcal{P}$(*x in M. X x ≤ a*) = *erlang-CDF k l a*))
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-distributed-mult-const*:
assumes *erlX*: *distributed M lborel X* (*erlang-density k l*)
assumes *a-pos*[*arith*]: *0 < α 0 < l*
shows *distributed M lborel* (*λx. α * X x*) (*erlang-density k* (*l / α*))
⟨*proof*⟩

lemma (**in** *prob-space*) *has-bochner-integral-erlang-ith-moment*:
fixes *k i* :: *nat* **and** *l* :: *real*
assumes [*arith*]: *0 < l* **and** *D*: *distributed M lborel X* (*erlang-density k l*)
shows *has-bochner-integral M* (*λx. X x ^ i*) (*fact* (*k* + *i*) / (*fact k * l ^ i*))
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-ith-moment-integrable*:
0 < l ⟹ distributed M lborel X (erlang-density k l) ⟹ integrable M (λx. X x ^ i)
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-ith-moment*:
0 < l ⟹ distributed M lborel X (erlang-density k l) ⟹
*expectation (λx. X x ^ i) = fact (k + i) / (fact k * l ^ i)*
⟨*proof*⟩

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*⟩

13.2 Exponential distribution

abbreviation *exponential-density* :: *real ⇒ real ⇒ real* **where**
exponential-density ≡ erlang-density 0

lemma *exponential-density-def*:
*exponential-density l x = (if x < 0 then 0 else l * exp (− x * l))*
⟨*proof*⟩

lemma *erlang-CDF-0*: *erlang-CDF 0 l a = (if 0 ≤ a then 1 − exp (− l * a) else 0)*
⟨*proof*⟩

lemma *prob-space-exponential-density*: *0 < l ⟹ prob-space (density lborel (exponential-density l))*
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributedD-le*:
assumes *D*: *distributed M lborel X (exponential-density l)* **and** *a*: *0 ≤ a* **and** *l*: *0 < l*
shows *P(x in M. X x ≤ a) = 1 − exp (− a * l)*
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributedD-gt*:
assumes *D*: *distributed M lborel X (exponential-density l)* **and** *a*: *0 ≤ a* **and** *l*: *0 < l*
shows *P(x in M. a < X x) = exp (− a * l)*
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributed-memoryless*:
assumes *D*: *distributed M lborel X (exponential-density l)* **and** *a*: *0 ≤ a* **and** *l*: *0 < l* **and** *t*: *0 ≤ t*
shows *P(x in M. a + t < X x | a < X x) = P(x in M. t < X x)*
⟨*proof*⟩

lemma *exponential-distributedI*:
assumes *X*[*measurable*]: $X \in$ *borel-measurable M* **and** [*arith*]: $0 < l$
and *X-distr*: $\bigwedge a.\ 0 \leq a \Longrightarrow$ *emeasure M* $\{x \in$ *space M*. $X\ x \leq a\} = 1 - exp\ (-\ a * l)$
shows *distributed M lborel X* (*exponential-density l*)
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributed-iff*:
assumes $0 < l$
shows *distributed M lborel X* (*exponential-density l*) $\longleftrightarrow$
$(X \in$ *borel-measurable M* $\wedge\ (\forall\, a \geq 0.\ \mathcal{P}(x$ *in M*. $X\ x \leq a) = 1 - exp\ (-\ a * l)))$
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributed-expectation*:
$0 < l \Longrightarrow$ *distributed M lborel X* (*exponential-density l*) $\Longrightarrow$ *expectation X* $= 1\ /\ l$
⟨*proof*⟩

lemma *exponential-density-nonneg*: $0 < l \Longrightarrow 0 \leq$ *exponential-density l x*
⟨*proof*⟩

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* $(\lambda x.$ *min* $(X\ x)\ (Y\ x))$ (*exponential-density* $(l + u)$)
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributed-Min*:
assumes *finI*: *finite I*
assumes *A*: $I \neq \{\}$
assumes *l*: $\bigwedge i.\ i \in I \Longrightarrow 0 < l\ i$
assumes *expX*: $\bigwedge i.\ i \in I \Longrightarrow$ *distributed M lborel* $(X\ i)$ (*exponential-density* $(l\ i)$)
assumes *ind*: *indep-vars* $(\lambda i.$ *borel*$)$ *X I*
shows *distributed M lborel* $(\lambda x.$ *Min* $((\lambda i.\ X\ i\ x)\ `I))$ (*exponential-density* $(\sum i \in I.\ l\ i)$)
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributed-variance*:
$0 < l \Longrightarrow$ *distributed M lborel X* (*exponential-density l*) $\Longrightarrow$ *variance X* $= 1\ /\ l^2$
⟨*proof*⟩

lemma *nn-integral-zero'*: *AE x in M*. $f\ x = 0 \Longrightarrow (\int^+ x.\ f\ x\ \partial M) = 0$
⟨*proof*⟩

lemma *convolution-erlang-density*:
fixes k_1 k_2 :: *nat*
assumes [*simp*, *arith*]: $0 < l$
shows $(\lambda x.\ \int^+ y.\ ennreal\ (erlang\text{-}density\ k_1\ l\ (x - y)) * ennreal\ (erlang\text{-}density\ k_2\ l\ y)\ \partial lborel) =$
$(erlang\text{-}density\ (Suc\ k_1 + Suc\ k_2 - 1)\ l)$
(**is** *?LHS* = *?RHS*)
⟨*proof*⟩

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* k_1 l)
assumes *erlY*: *distributed M lborel Y* (*erlang-density* k_2 l)
shows *distributed M lborel* $(\lambda x.\ X\ x + Y\ x)$ (*erlang-density* (*Suc* k_1 + *Suc* k_2 − 1) l)
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-distributed-sum*:
assumes *finI* : *finite I*
assumes *A*: $I \neq \{\}$
assumes [*simp*, *arith*]: $0 < l$
assumes *expX*: $\bigwedge i.\ i \in I \Longrightarrow$ *distributed M lborel* (*X i*) (*erlang-density* (*k i*) *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* $((\sum i{\in}I.\ Suc\ (k\ i)) - 1)\ l$)
⟨*proof*⟩

lemma (**in** *prob-space*) *exponential-distributed-sum*:
assumes *finI*: *finite I*
assumes *A*: $I \neq \{\}$
assumes *l*: $0 < l$
assumes *expX*: $\bigwedge i.\ i \in I \Longrightarrow$ *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* $((card\ I) - 1)\ l$)
⟨*proof*⟩

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*⟩

13.3 Uniform distribution

lemma *uniform-distrI*:
assumes *X*: $X \in measurable\ M\ M'$
and *A*: $A \in sets\ M'$ *emeasure* $M'\ A \neq \infty$ *emeasure* $M'\ A \neq 0$
assumes *distr*: $\bigwedge B.\ B \in sets\ M' \Longrightarrow emeasure\ M\ (X -` B \cap space\ M) =$

emeasure M' (A ∩ B) / emeasure M' A
shows *distr M M' X = uniform-measure M' A*
⟨*proof*⟩

lemma *uniform-distrI-borel*:
fixes *A* :: *real set*
assumes *X*[*measurable*]: *X* ∈ *borel-measurable M* **and** *A*: *emeasure lborel A = ennreal r 0 < r*
and [*measurable*]: *A* ∈ *sets borel*
assumes *distr*: ⋀*a*. *emeasure M* {*x*∈*space M*. *X x* ≤ *a*} = *emeasure lborel* (*A* ∩ {.. *a*}) / *r*
shows *distributed M lborel X* (λ*x*. *indicator A x / measure lborel A*)
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distrI-borel-atLeastAtMost*:
fixes *a b* :: *real*
assumes *X*: *X* ∈ *borel-measurable M* **and** *a < b*
assumes *distr*: ⋀*t*. *a* ≤ *t* ⟹ *t* ≤ *b* ⟹ *P*(*x in M*. *X x* ≤ *t*) = (*t* − *a*) / (*b* − *a*)
shows *distributed M lborel X* (λ*x*. *indicator* {*a..b*} *x / measure lborel* {*a..b*})
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distributed-measure*:
fixes *a b* :: *real*
assumes *D*: *distributed M lborel X* (λ*x*. *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*⟩

lemma (**in** *prob-space*) *uniform-distributed-bounds*:
fixes *a b* :: *real*
assumes *D*: *distributed M lborel X* (λ*x*. *indicator* {*a .. b*} *x / measure lborel* {*a .. b*})
shows *a < b*
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distributed-iff*:
fixes *a b* :: *real*
shows *distributed M lborel X* (λ*x*. *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*)))
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distributed-expectation*:
fixes *a b* :: *real*
assumes *D*: *distributed M lborel X* (λ*x*. *indicator* {*a .. b*} *x / measure lborel* {*a .. b*})
shows *expectation X* = (*a* + *b*) / *2*

⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distributed-variance*:
fixes *a b* :: *real*
assumes *D*: *distributed M lborel X* (λ*x*. *indicator* {*a* .. *b*} *x* / *measure lborel* {*a* .. *b*})
shows *variance X* = (*b* − *a*)² / *12*
⟨*proof*⟩

13.4 Normal distribution

definition *normal-density* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
normal-density μ σ *x* = *1* / *sqrt* (*2* ∗ *pi* ∗ σ²) ∗ *exp* (−(*x* − μ)²/ (*2* ∗ σ²))

abbreviation *std-normal-density* :: *real* ⇒ *real* **where**
std-normal-density ≡ *normal-density 0 1*

lemma *std-normal-density-def*: *std-normal-density x* = (*1* / *sqrt* (*2* ∗ *pi*)) ∗ *exp* (− *x*² / *2*)
⟨*proof*⟩

lemma *normal-density-nonneg*[*simp*]: *0* ≤ *normal-density* μ σ *x*
⟨*proof*⟩

lemma *normal-density-pos*: *0* < σ ⟹ *0* < *normal-density* μ σ *x*
⟨*proof*⟩

lemma *borel-measurable-normal-density*[*measurable*]: *normal-density* μ σ ∈ *borel-measurable borel*
⟨*proof*⟩

lemma *gaussian-moment-0*:
has-bochner-integral lborel (λ*x*. *indicator* {*0*..} *x* ∗*R* *exp* (− *x*²)) (*sqrt pi* / *2*)
⟨*proof*⟩

lemma *gaussian-moment-1*:
has-bochner-integral lborel (λ*x*::*real*. *indicator* {*0*..} *x* ∗*R* (*exp* (− *x*²) ∗ *x*)) (*1* / *2*)
⟨*proof*⟩

lemma
fixes *k* :: *nat*
shows *gaussian-moment-even-pos*:
has-bochner-integral lborel (λ*x*::*real*. *indicator* {*0*..} *x* ∗*R* (*exp* (−*x*²)∗*x*^(*2* ∗ *k*)))
((*sqrt pi* / *2*) ∗ (*fact* (*2* ∗ *k*) / (*2* ^ (*2* ∗ *k*) ∗ *fact k*)))
(**is** *?even*)
and *gaussian-moment-odd-pos*:
has-bochner-integral lborel (λ*x*::*real*. *indicator* {*0*..} *x* ∗*R* (*exp* (−*x*²)∗*x*^(*2* ∗

$k + 1$))) (*fact* k / 2)
(**is** *?odd*)
⟨*proof*⟩

context
fixes k :: *nat* **and** μ σ :: *real* **assumes** [*arith*]: $0 < \sigma$
begin

lemma *normal-moment-even*:
has-bochner-integral lborel (λx. *normal-density* μ σ x ∗ ($x - \mu$) ^ ($2 * k$)) (*fact* ($2 * k$) / ((2 / σ^2)^k ∗ *fact* k))
⟨*proof*⟩

lemma *normal-moment-abs-odd*:
has-bochner-integral lborel (λx. *normal-density* μ σ x ∗ |$x - \mu$|^($2 * k + 1$)) (2^k ∗ σ^($2 * k + 1$) ∗ *fact* k ∗ *sqrt* (2 / *pi*))
⟨*proof*⟩

lemma *normal-moment-odd*:
has-bochner-integral lborel (λx. *normal-density* μ σ x ∗ ($x - \mu$)^($2 * k + 1$)) 0
⟨*proof*⟩

lemma *integral-normal-moment-even*:
integral^L *lborel* (λx. *normal-density* μ σ x ∗ ($x - \mu$)^($2 * k$)) = *fact* ($2 * k$) / ((2 / σ^2)^k ∗ *fact* k)
⟨*proof*⟩

lemma *integral-normal-moment-abs-odd*:
integral^L *lborel* (λx. *normal-density* μ σ x ∗ |$x - \mu$|^($2 * k + 1$)) = 2 ^ k ∗ σ ^ ($2 * k + 1$) ∗ *fact* k ∗ *sqrt* (2 / *pi*)
⟨*proof*⟩

lemma *integral-normal-moment-odd*:
integral^L *lborel* (λx. *normal-density* μ σ x ∗ ($x - \mu$)^($2 * k + 1$)) = 0
⟨*proof*⟩

end

context
fixes σ :: *real*
assumes σ-*pos*[*arith*]: $0 < \sigma$
begin

lemma *normal-moment-nz-1*: *has-bochner-integral lborel* (λx. *normal-density* μ σ x ∗ x) μ
⟨*proof*⟩

lemma *integral-normal-moment-nz-1*:

integral^L *lborel* (λx. *normal-density* μ σ $x * x$) = μ
⟨*proof*⟩

lemma *integrable-normal-moment-nz-1*: *integrable lborel* (λx. *normal-density* μ σ $x * x$)
⟨*proof*⟩

lemma *integrable-normal-moment*: *integrable lborel* (λx. *normal-density* μ σ $x *$ $(x - \mu)\hat{}k$)
⟨*proof*⟩

lemma *integrable-normal-moment-abs*: *integrable lborel* (λx. *normal-density* μ σ x $* |x - \mu|\hat{}k$)
⟨*proof*⟩

lemma *integrable-normal-density*[*simp*, *intro*]: *integrable lborel* (*normal-density* μ σ)
⟨*proof*⟩

lemma *integral-normal-density*[*simp*]: ($\int$ x. *normal-density* μ σ x ∂*lborel*) = *1*
⟨*proof*⟩

lemma *prob-space-normal-density*:
prob-space (*density lborel* (*normal-density* μ σ))
⟨*proof*⟩

end

context
fixes k :: *nat*
begin

lemma *std-normal-moment-even*:
has-bochner-integral lborel (λx. *std-normal-density* $x * x\hat{}(2 * k)$) (*fact* $(2 * k)$ / $(2\hat{}k * fact\ k)$)
⟨*proof*⟩

lemma *std-normal-moment-abs-odd*:
has-bochner-integral lborel (λx. *std-normal-density* $x * |x|\hat{}(2 * k + 1)$) (*sqrt* $(2/pi) * 2\hat{}k * fact\ k$)
⟨*proof*⟩

lemma *std-normal-moment-odd*:
has-bochner-integral lborel (λx. *std-normal-density* $x * x\hat{}(2 * k + 1)$) *0*
⟨*proof*⟩

lemma *integral-std-normal-moment-even*:

integral^L lborel (*λx. std-normal-density x * x^(2*k)*) = *fact* (*2 * k*) / (*2^k * fact k*)
⟨*proof*⟩

lemma *integral-std-normal-moment-abs-odd*:
integral^L lborel (*λx. std-normal-density x * |x|^(2 * k + 1)*) = *sqrt* (*2 / pi*) * *2^k * fact k*
⟨*proof*⟩

lemma *integral-std-normal-moment-odd*:
integral^L lborel (*λx. std-normal-density x * x^(2 * k + 1)*) = *0*
⟨*proof*⟩

lemma *integrable-std-normal-moment-abs*: *integrable lborel* (*λx. std-normal-density x * |x|^k*)
⟨*proof*⟩

lemma *integrable-std-normal-moment*: *integrable lborel* (*λx. std-normal-density x * x^k*)
⟨*proof*⟩

end

lemma (**in** *prob-space*) *normal-density-affine*:
assumes *X*: *distributed M lborel X* (*normal-density μ σ*)
assumes [*simp*, *arith*]: *0 < σ α ≠ 0*
shows *distributed M lborel* (*λx. β + α * X x*) (*normal-density* (*β + α * μ*) (|*α*| * *σ*))
⟨*proof*⟩

lemma (**in** *prob-space*) *normal-standard-normal-convert*:
assumes *pos-var*[*simp*, *arith*]: *0 < σ*
shows *distributed M lborel X* (*normal-density μ σ*) = *distributed M lborel* (*λx.* (*X x − μ*) / *σ*) *std-normal-density*
⟨*proof*⟩

lemma *conv-normal-density-zero-mean*:
assumes [*simp*, *arith*]: *0 < σ 0 < τ*
shows (*λx.* ∫⁺*y. ennreal* (*normal-density 0 σ* (*x − y*) * *normal-density 0 τ y*) *∂lborel*) =
normal-density 0 (*sqrt* (*σ*² + *τ*²)) (**is** *?LHS = ?RHS*)
⟨*proof*⟩

lemma *conv-std-normal-density*:
(*λx.* ∫⁺*y. ennreal* (*std-normal-density* (*x − y*) * *std-normal-density y*) *∂lborel*) =
(*normal-density 0* (*sqrt 2*))
⟨*proof*⟩

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* (μ + ν) (*sqrt* (σ^2 + τ^2)))
⟨*proof*⟩

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* (μ − ν) (*sqrt* (σ^2 + τ^2)))
⟨*proof*⟩

lemma (**in** *prob-space*) *sum-indep-normal*:
assumes *finite I I* ≠ {} *indep-vars* ($\lambda i.$ *borel*) *X I*
assumes $\bigwedge i.$ *i* ∈ *I* ⟹ $0 < \sigma$ *i*
assumes *normal*: $\bigwedge i.$ *i* ∈ *I* ⟹ *distributed M lborel* (*X i*) (*normal-density* (μ *i*) (σ *i*))
shows *distributed M lborel* ($\lambda x.$ $\sum i \in I.$ *X i x*) (*normal-density* ($\sum i \in I.$ μ *i*) (*sqrt* ($\sum i \in I.$ (σ *i*)^2)))
⟨*proof*⟩

lemma (**in** *prob-space*) *standard-normal-distributed-expectation*:
assumes *D*: *distributed M lborel X std-normal-density*
shows *expectation X* = *0*
⟨*proof*⟩

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* = μ
⟨*proof*⟩

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* = σ^2
⟨*proof*⟩

lemma (**in** *prob-space*) *standard-normal-distributed-variance*:
distributed M lborel X std-normal-density ⟹ *variance X* = *1*
⟨*proof*⟩

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* $*$ σ^2) / *2*
⟨*proof*⟩

end

14 Characteristic Functions

theory *Characteristic-Functions*
imports *Weak-Convergence Independent-Family Distributions*
begin

lemma *mult-min-right*: $a \geq 0 \Longrightarrow$ (*a* :: *real*) $*$ *min b c* = *min* (*a* $*$ *b*) (*a* $*$ *c*)
⟨*proof*⟩

lemma *sequentially-even-odd*:
assumes *E*: *eventually* (λn. *P* (*2* $*$ *n*)) *sequentially* **and** *O*: *eventually* (λn. *P* (*2* $*$ *n* + *1*)) *sequentially*
shows *eventually P sequentially*
⟨*proof*⟩

lemma *limseq-even-odd*:
assumes (λn. *f* (*2* $*$ *n*)) $\longrightarrow$ (*l* :: *'a* :: *topological-space*)
and (λn. *f* (*2* $*$ *n* + *1*)) $\longrightarrow$ *l*
shows *f* $\longrightarrow$ *l*
⟨*proof*⟩

14.1 Application of the FTC: integrating $e^i x$

abbreviation *iexp* :: *real* $\Rightarrow$ *complex* **where**
iexp $\equiv$ (λx. *exp* (i $*$ *complex-of-real x*))

lemma *isCont-iexp* [*simp*]: *isCont iexp x*
⟨*proof*⟩

lemma *has-vector-derivative-iexp*[*derivative-intros*]:
(*iexp has-vector-derivative* i $*$ *iexp x*) (*at x within s*)
⟨*proof*⟩

lemma *interval-integral-iexp*:
fixes *a b* :: *real*
shows (*CLBINT x*=*a*..*b*. *iexp x*) = i $*$ *iexp a* − i $*$ *iexp b*
⟨*proof*⟩

14.2 The Characteristic Function of a Real Measure.

definition

char :: *real measure* ⇒ *real* ⇒ *complex*
where *char M t* ≡ *CLINT x*|*M*. *iexp* (*t* ∗ *x*)

lemma (**in** *real-distribution*) *char-zero*: *char M 0* = *1*
⟨*proof*⟩

lemma (**in** *prob-space*) *integrable-iexp*:
assumes *f*: *f* ∈ *borel-measurable M* ⋀*x*. *Im* (*f x*) = *0*
shows *integrable M* (λ*x*. *exp* (i ∗ (*f x*)))
⟨*proof*⟩

lemma (**in** *real-distribution*) *cmod-char-le-1*: *norm* (*char M t*) ≤ *1*
⟨*proof*⟩

lemma (**in** *real-distribution*) *isCont-char*: *isCont* (*char M*) *t*
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-measurable* [*measurable*]: *char M* ∈ *borel-measurable borel*
⟨*proof*⟩

14.3 Independence

lemma (**in** *prob-space*) *char-distr-add*:
fixes *X1 X2* :: *'a* ⇒ *real* **and** *t* :: *real*
assumes *indep-var borel X1 borel X2*
shows *char* (*distr M borel* (λω. *X1* ω + *X2* ω)) *t* =
char (*distr M borel X1*) *t* ∗ *char* (*distr M borel X2*) *t*
⟨*proof*⟩

lemma (**in** *prob-space*) *char-distr-sum*:
indep-vars (λ*i*. *borel*) *X A* ⟹
char (*distr M borel* (λω. ∑ *i*∈*A*. *X i* ω)) *t* = (∏ *i*∈*A*. *char* (*distr M borel* (*X i*)) *t*)
⟨*proof*⟩

14.4 Approximations to e^{ix}

Proofs from Billingsley, page 343.

lemma *CLBINT-I0c-power-mirror-iexp*:
fixes *x* :: *real* **and** *n* :: *nat*
defines *f s m* ≡ *complex-of-real* ((*x* − *s*) ^ *m*)
shows (*CLBINT s*=*0*..*x*. *f s n* ∗ *iexp s*) =
x^*Suc n* / *Suc n* + (i / *Suc n*) ∗ (*CLBINT s*=*0*..*x*. *f s* (*Suc n*) ∗ *iexp s*)
⟨*proof*⟩

lemma *iexp-eq1*:
fixes *x* :: *real*
defines *f s m* ≡ *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*⟩

lemma *iexp-eq2*:
fixes *x* :: *real*
defines *f s m* ≡ *complex-of-real* ((*x* − *s*) ^ *m*)
shows *iexp x* = ($\sum k \leq Suc\ n$. (i*x)^*k*/*fact k*) + i^*Suc n*/*fact n* * (*CLBINT s=0..x*. *f s n*(*iexp s* − *1*))
⟨*proof*⟩

lemma *abs-LBINT-I0c-abs-power-diff*:
|*LBINT s=0..x*. |(*x* − *s*)^*n*|| = |*x* ^ (*Suc n*) / (*Suc n*)|
⟨*proof*⟩

lemma *iexp-approx1*: *cmod* (*iexp x* − ($\sum k \leq n$. (i * *x*)^*k* / *fact k*)) ≤ |*x*|^(*Suc n*) / *fact* (*Suc n*)
⟨*proof*⟩

lemma *iexp-approx2*: *cmod* (*iexp x* − ($\sum k \leq n$. (i * *x*)^*k* / *fact k*)) ≤ *2* * |*x*|^*n* / *fact n*
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-approx1*:
assumes *integrable-moments*: $\bigwedge k.\ k \leq n \Longrightarrow$ *integrable M* (λ*x*. *x*^*k*)
shows *cmod* (*char M t* − ($\sum k \leq n$. ((i * *t*)^*k* / *fact k*) * *expectation* (λ*x*. *x*^*k*))) ≤
(*2*|*t*|^*n* / *fact n*) * *expectation* (λ*x*. |*x*|^*n*) (**is** *cmod* (*char M t* − *?t1*) ≤ -)
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-approx2*:
assumes *integrable-moments*: $\bigwedge k.\ k \leq n \Longrightarrow$ *integrable M* (λ*x*. *x* ^ *k*)
shows *cmod* (*char M t* − ($\sum k \leq n$. ((i * *t*)^*k* / *fact k*) * *expectation* (λ*x*. *x*^*k*))) ≤
(|*t*|^*n* / *fact* (*Suc n*)) * *expectation* (λ*x*. *min* (*2* * |*x*|^*n* * *Suc n*) (|*t*| * |*x*|^*Suc n*))
(**is** *cmod* (*char M t* − *?t1*) ≤ -)
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-approx3*:
fixes *t*
assumes
integrable-1: *integrable M* (λ*x*. *x*) **and**
integral-1: *expectation* (λ*x*. *x*) = *0* **and**
integrable-2: *integrable M* (λ*x*. *x*^*2*) **and**
integral-2: *variance* (λ*x*. *x*) = σ*2*
shows *cmod* (*char M t* − (*1* − *t*^*2* * σ*2* / *2*)) ≤
(*t*^*2* / *6*) * *expectation* (λ*x*. *min* (*6* * *x*^*2*) (*abs t* * (*abs x*)^*3*))

⟨*proof*⟩

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 μ :: *real measure* **and** X
assumes *rv-X* [*simp*]: *random-variable borel X*
and [*simp*]: *integrable M X integrable M* ($\lambda x.\ (X\ x)\hat{}2$) *expectation X = 0*
and *var-X*: *variance X* $= \sigma 2$
and μ*-def*: μ = *distr M borel X*
shows *cmod* (*char* μ *t* − (*1* − *t*$\hat{}$*2* ∗ $\sigma 2$ / *2*)) ≤
(*t*$\hat{}$*2* / *6*) ∗ *expectation* ($\lambda x.$ *min* (*6* ∗ (*X x*)$\hat{}$*2*) (|*t*| ∗ |*X x*|$\hat{}$*3*))
⟨*proof*⟩

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 μ :: *real measure* **and** X
assumes *integrable-moments* : $\bigwedge k.\ k \le n \Longrightarrow$ *integrable M* ($\lambda x.\ X\ x\ \hat{}\ k$)
and *rv-X*[*measurable*]: *random-variable borel X*
and μ*-distr* : *distr M borel X* = μ
shows *cmod* (*char* μ *t* − ($\sum k \le n.$ ((i ∗ *t*)$\hat{}k$ / *fact k*) ∗ *expectation* ($\lambda x.$ (*X x*)$\hat{}k$))) ≤
(*2* ∗ |*t*|$\hat{}n$ / *fact n*) ∗ *expectation* ($\lambda x.$ |*X x*|$\hat{}n$)
⟨*proof*⟩

14.5 Calculation of the Characteristic Function of the Standard Distribution

abbreviation
std-normal-distribution ≡ *density lborel std-normal-density*

lemma *real-dist-normal-dist*: *real-distribution std-normal-distribution*
⟨*proof*⟩

lemma *std-normal-distribution-even-moments*:
fixes *k* :: *nat*
shows (*LINT x*|*std-normal-distribution*. *x*$\hat{}$(*2* ∗ *k*)) = *fact* (*2* ∗ *k*) / (*2*$\hat{}k$ ∗ *fact k*)
and *integrable std-normal-distribution* ($\lambda x.$ *x*$\hat{}$(*2* ∗ *k*))
⟨*proof*⟩

lemma *integrable-std-normal-distribution-moment*: *integrable std-normal-distribution* ($\lambda x.$ *x*$\hat{}k$)
⟨*proof*⟩

lemma *integral-std-normal-distribution-moment-odd*:

odd k ⟹ integral^L std-normal-distribution (λx. x^k) = 0
⟨*proof*⟩

lemma *std-normal-distribution-even-moments-abs*:
fixes *k* :: *nat*
shows (*LINT x|std-normal-distribution. |x|^(2 * k)) = fact (2 * k) / (2^k * fact k)*
⟨*proof*⟩

lemma *std-normal-distribution-odd-moments-abs*:
fixes *k* :: *nat*
shows (*LINT x|std-normal-distribution. |x|^(2 * k + 1)) = sqrt (2 / pi) * 2 ^ k * fact k*
⟨*proof*⟩

theorem *char-std-normal-distribution*:
char std-normal-distribution = (λt. complex-of-real (exp (− (t^2) / 2)))
⟨*proof*⟩

end

15 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)*
⟨*proof*⟩

theorem *Helly-selection*:
fixes *f* :: *nat ⇒ real ⇒ real*
assumes *rcont*: *⋀n x. continuous (at-right x) (f n)*
assumes *mono*: *⋀n. mono (f n)*
assumes *bdd*: *⋀n x. |f n x| ≤ M*
shows *∃ s. strict-mono (s::nat ⇒ nat) ∧ (∃ F. (∀ x. continuous (at-right x) F) ∧ mono F ∧ (∀ x. |F x| ≤ M) ∧*
(∀ x. continuous (at x) F ⟶ (λn. f (s n) x) ⟶⟶ F x))
⟨*proof*⟩

definition
tight :: *(nat ⇒ real measure) ⇒ bool*
where
tight μ ≡ (∀ n. real-distribution (μ n)) ∧ (∀ (ε::real)>0. ∃ a b::real. a < b ∧ (∀ 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-mono* $(r :: nat \Rightarrow nat) \wedge$ *real-distribution* $M \wedge$ *weak-conv-m* $(\mu \circ s \circ r)\ M$
⟨*proof*⟩

corollary *tight-subseq-weak-converge*:
fixes $\mu ::$ *nat* $\Rightarrow$ *real measure* **and** $M ::$ *real measure*
assumes $\bigwedge n.$ *real-distribution* $(\mu\ n)$ *real-distribution* M **and** *tight*: *tight* μ **and**
subseq: $\bigwedge s\ \nu.$ *strict-mono* $s \Longrightarrow$ *real-distribution* $\nu \Longrightarrow$ *weak-conv-m* $(\mu \circ s)\ \nu \Longrightarrow$ *weak-conv-m* $(\mu \circ s)\ M$
shows *weak-conv-m* $\mu\ M$
⟨*proof*⟩

end

16 Integral of sinc

theory *Sinc-Integral*
imports *Distributions*
begin

16.1 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 $(\lambda x.\ x\hat{}k * exp\ (-x) *$ *indicator* $\{0\ ..\}\ x$::*real*$)$ $(fact\ k)$
⟨*proof*⟩

lemma *has-bochner-integral-I0i-power-exp-m*:
has-bochner-integral lborel $(\lambda x.\ x\hat{}k * exp\ (-x) *$ *indicator* $\{0 <..\}\ x$::*real*$)$ $(fact\ k)$
⟨*proof*⟩

lemma *integrable-I0i-exp-mscale*: $0 <$ (u::*real*) $\Longrightarrow$ *set-integrable lborel* $\{0 <..\}$ $(\lambda x.\ exp\ (-(x * u)))$
⟨*proof*⟩

lemma *LBINT-I0i-exp-mscale*: $0 < (u::real) \Longrightarrow LBINT\ x{=}0..\infty.\ exp\ (-(x * u)) = 1 / u$
⟨*proof*⟩

lemma *LBINT-I0c-exp-mscale-sin*:
$LBINT\ x{=}0..t.\ exp\ (-(u * x)) * sin\ x =$
$(1 / (1 + u\hat{}2)) * (1 - exp\ (-(u * t)) * (u * sin\ t + cos\ t))$ (**is** - = *?F t*)
⟨*proof*⟩

lemma *LBINT-I0i-exp-mscale-sin*:
assumes $0 < x$
shows $LBINT\ u{=}0..\infty.\ |exp\ (-u * x) * sin\ x| = |sin\ x| / x$
⟨*proof*⟩

lemma
shows *integrable-inverse-1-plus-square*:
$set\text{-}integrable\ lborel\ (einterval\ (-\infty)\ \infty)\ (\lambda x.\ inverse\ (1 + x\hat{}2))$
and *LBINT-inverse-1-plus-square*:
$LBINT\ x{=}-\infty..\infty.\ inverse\ (1 + x\hat{}2) = pi$
⟨*proof*⟩

lemma
shows *integrable-I0i-1-div-plus-square*:
$interval\text{-}lebesgue\text{-}integrable\ lborel\ 0\ \infty\ (\lambda x.\ 1 / (1 + x\hat{}2))$
and *LBINT-I0i-1-div-plus-square*:
$LBINT\ x{=}0..\infty.\ 1 / (1 + x\hat{}2) = pi / 2$
⟨*proof*⟩

17 The sinc function, and the sine integral (Si)

abbreviation *sinc* :: *real* ⇒ *real* **where**
$sinc \equiv (\lambda x.\ if\ x = 0\ then\ 1\ else\ sin\ x / x)$

lemma *sinc-at-0*: $((\lambda x.\ sin\ x / x::real) \longrightarrow 1)\ (at\ 0)$
⟨*proof*⟩

lemma *isCont-sinc*: $isCont\ sinc\ x$
⟨*proof*⟩

lemma *continuous-on-sinc*[*continuous-intros*]:
$continuous\text{-}on\ S\ f \Longrightarrow continuous\text{-}on\ S\ (\lambda x.\ sinc\ (f\ x))$
⟨*proof*⟩

lemma *borel-measurable-sinc*[*measurable*]: $sinc \in borel\text{-}measurable\ borel$
⟨*proof*⟩

lemma *sinc-AE*: $AE\ x\ in\ lborel.\ sin\ x / x = sinc\ x$
⟨*proof*⟩

definition *Si* :: *real* ⇒ *real* **where** *Si t* ≡ *LBINT x=0..t. sin x / x*

lemma *sinc-neg* [*simp*]: *sinc* (− *x*) = *sinc x*
⟨*proof*⟩

lemma *Si-alt-def* : *Si t* = *LBINT x=0..t. sinc x*
⟨*proof*⟩

lemma *Si-neg*:
assumes *T* ≥ *0* **shows** *Si* (− *T*) = − *Si T*
⟨*proof*⟩

lemma *integrable-sinc′*:
interval-lebesgue-integrable lborel (*ereal 0*) (*ereal T*) (λ*t. sin* (*t* ∗ *ϑ*) / *t*)
⟨*proof*⟩

lemma *DERIV-Si*: (*Si has-real-derivative sinc x*) (*at x*)
⟨*proof*⟩

lemma *isCont-Si*: *isCont Si x*
⟨*proof*⟩

lemma *borel-measurable-Si*[*measurable*]: *Si* ∈ *borel-measurable borel*
⟨*proof*⟩

lemma *Si-at-top-LBINT*:
((λ*t*. (*LBINT x=0..∞. exp* (−(*x* ∗ *t*)) ∗ (*x* ∗ *sin t* + *cos t*) / (*1* + *x*^*2*))) ⟶ *0*) *at-top*
⟨*proof*⟩

lemma *Si-at-top-integrable*:
assumes *t* ≥ *0*
shows *interval-lebesgue-integrable lborel 0* ∞ (λ*x. exp* (− (*x* ∗ *t*)) ∗ (*x* ∗ *sin t* + *cos t*) / (*1* + *x*^{2}))
⟨*proof*⟩

lemma *Si-at-top*: (*Si* ⟶ *pi* / *2*) *at-top*
⟨*proof*⟩

17.1 The final theorems: boundedness and scalability

lemma *bounded-Si*: ∃ *B*. ∀ *T*. |*Si T*| ≤ *B*
⟨*proof*⟩

lemma *LBINT-I0c-sin-scale-divide*:
assumes *T* ≥ *0*

shows *LBINT t=0..T. sin (t * ϑ) / t = sgn ϑ * Si (T * |ϑ|)*
⟨*proof*⟩

end

18 The Levy inversion theorem, and the Levy continuity theorem.

theory *Levy*
imports *Characteristic-Functions Helly-Selection Sinc-Integral*
begin

18.1 The Levy inversion theorem

lemma *Levy-Inversion-aux1*:
fixes *a b* :: *real*
assumes *a ≤ b*
shows ((λ*t*. (*iexp* (−(*t* * *a*)) − *iexp* (−(*t* * *b*))) / (i * *t*)) ⟶ *b* − *a*) (*at 0*)
(**is** (*?F* ⟶ -) (*at* -))
⟨*proof*⟩

lemma *Levy-Inversion-aux2*:
fixes *a b t* :: *real*
assumes *a ≤ b* **and** *t ≠ 0*
shows *cmod* ((*iexp* (*t* * *b*) − *iexp* (*t* * *a*)) / (i * *t*)) ≤ *b* − *a* (**is** *?F* ≤ -)
⟨*proof*⟩

theorem (**in** *real-distribution*) *Levy-Inversion*:
fixes *a b* :: *real*
assumes *a ≤ b*
defines *μ* ≡ *measure M* **and** *φ* ≡ *char M*
assumes *μ* {*a*} = *0* **and** *μ* {*b*} = *0*
shows (λ*T*. *1* / (*2* * *pi*) * (*CLBINT t*=−*T*..*T*. (*iexp* (−(*t* * *a*)) − *iexp* (−(*t* * *b*))) / (i * *t*) * *φ t*))
⟶ *μ* {*a*<..*b*}
(**is** (λ*T*. *1* / (*2* * *pi*) * (*CLBINT t*=−*T*..*T*. *?F t* * *φ t*)) ⟶ *of-real* (*μ* {*a*<..*b*}))
⟨*proof*⟩

theorem *Levy-uniqueness*:
fixes *M1 M2* :: *real measure*
assumes *real-distribution M1 real-distribution M2* **and**
char M1 = *char M2*
shows *M1* = *M2*
⟨*proof*⟩

18.2 The Levy continuity theorem

theorem *levy-continuity1*:
fixes $M :: nat \Rightarrow real\ measure$ **and** $M' :: real\ measure$
assumes $\bigwedge n.\ real\text{-}distribution\ (M\ n)\ real\text{-}distribution\ M'\ weak\text{-}conv\text{-}m\ M\ M'$
shows $(\lambda n.\ char\ (M\ n)\ t) \longrightarrow char\ M'\ t$
⟨*proof*⟩

theorem *levy-continuity*:
fixes $M :: nat \Rightarrow real\ measure$ **and** $M' :: real\ measure$
assumes *real-distr-M* : $\bigwedge n.\ real\text{-}distribution\ (M\ n)$
and *real-distr-M'*: $real\text{-}distribution\ M'$
and *char-conv*: $\bigwedge t.\ (\lambda n.\ char\ (M\ n)\ t) \longrightarrow char\ M'\ t$
shows $weak\text{-}conv\text{-}m\ M\ M'$
⟨*proof*⟩

end

19 The Central Limit Theorem

theory *Central-Limit-Theorem*
imports *Levy*
begin

theorem (**in** *prob-space*) *central-limit-theorem-zero-mean*:
fixes $X :: nat \Rightarrow 'a \Rightarrow real$
and $\mu :: real\ measure$
and $\sigma :: real$
and $S :: nat \Rightarrow 'a \Rightarrow real$
assumes *X-indep*: $indep\text{-}vars\ (\lambda i.\ borel)\ X\ UNIV$
and *X-mean-0*: $\bigwedge n.\ expectation\ (X\ n) = 0$
and *σ-pos*: $\sigma > 0$
and *X-square-integrable*: $\bigwedge n.\ integrable\ M\ (\lambda x.\ (X\ n\ x)^2)$
and *X-variance*: $\bigwedge n.\ variance\ (X\ n) = \sigma^2$
and *X-distrib*: $\bigwedge n.\ distr\ M\ borel\ (X\ n) = \mu$
defines $S\ n \equiv \lambda x.\ \sum i{<}n.\ X\ i\ x$
shows $weak\text{-}conv\text{-}m\ (\lambda n.\ distr\ M\ borel\ (\lambda x.\ S\ n\ x\ /\ sqrt\ (n * \sigma^2)))\ std\text{-}normal\text{-}distribution$
⟨*proof*⟩

theorem (**in** *prob-space*) *central-limit-theorem*:
fixes $X :: nat \Rightarrow 'a \Rightarrow real$
and $\mu :: real\ measure$
and $\sigma :: real$
and $S :: nat \Rightarrow 'a \Rightarrow real$
assumes *X-indep*: $indep\text{-}vars\ (\lambda i.\ borel)\ X\ UNIV$
and *X-mean*: $\bigwedge n.\ expectation\ (X\ n) = m$
and *σ-pos*: $\sigma > 0$
and *X-square-integrable*: $\bigwedge n.\ integrable\ M\ (\lambda x.\ (X\ n\ x)^2)$
and *X-variance*: $\bigwedge n.\ variance\ (X\ n) = \sigma^2$

and *X-distrib*: $\bigwedge n.$ *distr M borel* $(X\ n) = \mu$
defines $X'\ i\ x \equiv X\ i\ x - m$
shows *weak-conv-m* $(\lambda n.$ *distr M borel* $(\lambda x.\ (\sum i{<}n.\ X'\ i\ x)\ /\ sqrt\ (n{*}\sigma^2)))$
std-normal-distribution
⟨*proof*⟩

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*
⟨*proof*⟩

instantiation *discrete* :: (*type*) *metric-space*
begin

definition *dist-discrete* :: *'a discrete* ⇒ *'a discrete* ⇒ *real*
where *dist-discrete n m* = (*if n* = *m then 0 else 1*)

definition *uniformity-discrete* :: (*'a discrete* × *'a discrete*) *filter* **where**
(*uniformity*::(*'a discrete* × *'a discrete*) *filter*) = (*INF* $e{\in}\{0 <..\}.$ *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-discrete* :: *'a discrete set* ⇒ *bool* **where**
(*open*::*'a discrete set* ⇒ *bool*) *U* ⟷ $(\forall x{\in}U.$ *eventually* $(\lambda(x', y).\ x' = x \longrightarrow y \in U)$ *uniformity*)

instance ⟨*proof*⟩
end

lemma *open-discrete*: *open* (*S* :: *'a discrete set*)
⟨*proof*⟩

instance *discrete* :: (*type*) *complete-space*
⟨*proof*⟩

instance *discrete* :: (*countable*) *countable*
⟨*proof*⟩

instance *discrete* :: (*countable*) *second-countable-topology*
⟨*proof*⟩

instance *discrete* :: (*countable*) *polish-space* ⟨*proof*⟩

end

20 Probability mass function

theory *Probability-Mass-Function*
imports
Giry-Monad
HOL−Library.Multiset
begin

Conflicting notation from *HOL−Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (**infixr** ‹*abs'-summable'-on*› *46*)

lemma *AE-emeasure-singleton*:
assumes *x*: *emeasure M* {*x*} ≠ *0* **and** *ae*: *AE x in M. P x* **shows** *P x*
⟨*proof*⟩

lemma *AE-measure-singleton*: *measure M* {*x*} ≠ *0* ⟹ *AE x in M. P x* ⟹ *P x*
⟨*proof*⟩

lemma (**in** *finite-measure*) *AE-support-countable*:
assumes [*simp*]: *sets M* = *UNIV*
shows (*AE x in M. measure M* {*x*} ≠ *0*) ⟷ (∃ *S. countable S* ∧ (*AE x in M. x* ∈ *S*))
⟨*proof*⟩

20.1 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*
⟨*proof*⟩

declare [[*coercion measure-pmf*]]

lemma *prob-space-measure-pmf*: *prob-space* (*measure-pmf p*)
⟨*proof*⟩

interpretation *measure-pmf*: *prob-space measure-pmf M* **for** *M*
⟨*proof*⟩

interpretation *measure-pmf*: *subprob-space measure-pmf M* **for** *M*
⟨*proof*⟩

lemma *subprob-space-measure-pmf*: *subprob-space* (*measure-pmf x*)
⟨*proof*⟩

locale *pmf-as-measure*
begin

setup-lifting *type-definition-pmf*

end

context
begin

interpretation *pmf-as-measure* ⟨*proof*⟩

lemma *sets-measure-pmf*[*simp*]: *sets* (*measure-pmf p*) = *UNIV*
⟨*proof*⟩

lemma *sets-measure-pmf-count-space*[*measurable-cong*]:
sets (*measure-pmf M*) = *sets* (*count-space UNIV*)
⟨*proof*⟩

lemma *space-measure-pmf*[*simp*]: *space* (*measure-pmf p*) = *UNIV*
⟨*proof*⟩

lemma *measure-pmf-UNIV* [*simp*]: *measure* (*measure-pmf p*) *UNIV* = *1*
⟨*proof*⟩

lemma *measure-pmf-in-subprob-algebra*[*measurable* (*raw*)]: *measure-pmf x* ∈ *space* (*subprob-algebra* (*count-space UNIV*))
⟨*proof*⟩

lemma *measurable-pmf-measure1*[*simp*]: *measurable* (*M* :: *'a pmf*) *N* = *UNIV* → *space N*
⟨*proof*⟩

lemma *measurable-pmf-measure2*[*simp*]: *measurable N* (*M* :: *'a pmf*) = *measurable N* (*count-space UNIV*)
⟨*proof*⟩

lemma *measurable-pair-restrict-pmf2*:
assumes *countable A*
assumes [*measurable*]: ⋀*y*. *y* ∈ *A* ⟹ (λ*x*. *f* (*x*, *y*)) ∈ *measurable M L*
shows *f* ∈ *measurable* (*M* ⨂$_M$ *restrict-space* (*measure-pmf N*) *A*) *L* (**is** *f* ∈ *measurable ?M* -)
⟨*proof*⟩

lemma *measurable-pair-restrict-pmf1*:
assumes *countable A*
assumes [*measurable*]: ⋀*x*. *x* ∈ *A* ⟹ (λ*y*. *f* (*x*, *y*)) ∈ *measurable N L*
shows *f* ∈ *measurable* (*restrict-space* (*measure-pmf M*) *A* ⨂$_M$ *N*) *L*
⟨*proof*⟩

lift-definition *pmf* :: *'a pmf* ⇒ *'a* ⇒ *real* **is** λ*M x*. *measure M* {*x*} ⟨*proof*⟩

lift-definition *set-pmf* :: *'a pmf* ⇒ *'a set* **is** λ*M*. {*x*. *measure M* {*x*} ≠ *0*} ⟨*proof*⟩
declare [[*coercion set-pmf*]]

lemma *AE-measure-pmf*: *AE x in* (*M*::*'a pmf*). *x* ∈ *M*
⟨*proof*⟩

lemma *emeasure-pmf-single-eq-zero-iff*:
fixes *M* :: *'a pmf*
shows *emeasure M* {*y*} = *0* ⟷ *y* ∉ *M*
⟨*proof*⟩

lemma *AE-measure-pmf-iff*: (*AE x in measure-pmf M*. *P x*) ⟷ (∀ *y*∈*M*. *P y*)
⟨*proof*⟩

lemma *AE-pmfI*: (⋀*y*. *y* ∈ *set-pmf M* ⟹ *P y*) ⟹ *almost-everywhere* (*measure-pmf M*) *P*
⟨*proof*⟩

lemma *countable-set-pmf* [*simp*]: *countable* (*set-pmf p*)
⟨*proof*⟩

lemma *pmf-positive*: *x* ∈ *set-pmf p* ⟹ *0* < *pmf p x*
⟨*proof*⟩

lemma *pmf-nonneg*[*simp*]: *0* ≤ *pmf p x*
⟨*proof*⟩

lemma *pmf-not-neg* [*simp*]: ¬*pmf p x* < *0*
⟨*proof*⟩

lemma *pmf-pos* [*simp*]: *pmf p x* ≠ *0* ⟹ *pmf p x* > *0*
⟨*proof*⟩

lemma *pmf-le-1*: *pmf p x* ≤ *1*
⟨*proof*⟩

lemma *set-pmf-not-empty*: *set-pmf M* ≠ {}
⟨*proof*⟩

lemma *set-pmf-iff*: *x* ∈ *set-pmf M* ⟷ *pmf M x* ≠ *0*
⟨*proof*⟩

lemma *pmf-positive-iff*: *0* < *pmf p x* ⟷ *x* ∈ *set-pmf p*
⟨*proof*⟩

lemma *set-pmf-eq*: *set-pmf M* = {*x*. *pmf M x* ≠ *0*}
⟨*proof*⟩

lemma *set-pmf-eq'*: *set-pmf p* = {*x*. *pmf p x* > *0*}
⟨*proof*⟩

lemma *emeasure-pmf-single*:
fixes *M* :: *'a pmf*
shows *emeasure M* {*x*} = *pmf M x*
⟨*proof*⟩

lemma *measure-pmf-single*: *measure* (*measure-pmf M*) {*x*} = *pmf M x*
⟨*proof*⟩

lemma *emeasure-measure-pmf-finite*: *finite S* ⟹ *emeasure* (*measure-pmf M*) *S* = (∑ *s*∈*S*. *pmf M s*)
⟨*proof*⟩

lemma *measure-measure-pmf-finite*: *finite S* ⟹ *measure* (*measure-pmf M*) *S* = *sum* (*pmf M*) *S*
⟨*proof*⟩

lemma *sum-pmf-eq-1*:
assumes *finite A set-pmf p* ⊆ *A*
shows (∑ *x*∈*A*. *pmf p x*) = *1*
⟨*proof*⟩

lemma *nn-integral-measure-pmf-support*:
fixes *f* :: *'a* ⇒ *ennreal*
assumes *f*: *finite A* **and** *nn*: ⋀*x*. *x* ∈ *A* ⟹ *0* ≤ *f x* ⋀*x*. *x* ∈ *set-pmf M* ⟹ *x* ∉ *A* ⟹ *f x* = *0*
shows (∫⁺*x*. *f x* ∂*measure-pmf M*) = (∑ *x*∈*A*. *f x* ∗ *pmf M x*)
⟨*proof*⟩

lemma *nn-integral-measure-pmf-finite*:
fixes *f* :: *'a* ⇒ *ennreal*
assumes *f*: *finite* (*set-pmf M*) **and** *nn*: ⋀*x*. *x* ∈ *set-pmf M* ⟹ *0* ≤ *f x*
shows (∫⁺*x*. *f x* ∂*measure-pmf M*) = (∑ *x*∈*set-pmf M*. *f x* ∗ *pmf M x*)
⟨*proof*⟩

lemma *integrable-measure-pmf-finite*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *finite* (*set-pmf M*) ⟹ *integrable M f*
⟨*proof*⟩

lemma *integral-measure-pmf-real*:
assumes [*simp*]: *finite A* **and** ⋀*a*. *a* ∈ *set-pmf M* ⟹ *f a* ≠ *0* ⟹ *a* ∈ *A*
shows (∫ *x*. *f x* ∂*measure-pmf M*) = (∑ *a*∈*A*. *f a* ∗ *pmf M a*)
⟨*proof*⟩

lemma *integrable-pmf*: *integrable* (*count-space X*) (*pmf M*)
⟨*proof*⟩

lemma *integral-pmf*: $(\int x.\ pmf\ M\ x\ \partial count\text{-}space\ X) = measure\ M\ X$
⟨*proof*⟩

lemma *integral-pmf-restrict*:
$(f::'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}) \in borel\text{-}measurable\ (count\text{-}space\ UNIV) \Longrightarrow$
$(\int x.\ f\ x\ \partial measure\text{-}pmf\ M) = (\int x.\ f\ x\ \partial restrict\text{-}space\ M\ M)$
⟨*proof*⟩

lemma *emeasure-pmf*: $emeasure\ (M::'a\ pmf)\ M = 1$
⟨*proof*⟩

lemma *emeasure-pmf-UNIV* [*simp*]: $emeasure\ (measure\text{-}pmf\ M)\ UNIV = 1$
⟨*proof*⟩

lemma *in-null-sets-measure-pmfI*:
$A \cap set\text{-}pmf\ p = \{\} \Longrightarrow A \in null\text{-}sets\ (measure\text{-}pmf\ p)$
⟨*proof*⟩

lemma *measure-subprob*: $measure\text{-}pmf\ M \in space\ (subprob\text{-}algebra\ (count\text{-}space\ UNIV))$
⟨*proof*⟩

20.2 Monad Interpretation

lemma *measurable-measure-pmf*[*measurable*]:
$(\lambda x.\ measure\text{-}pmf\ (M\ x)) \in measurable\ (count\text{-}space\ UNIV)\ (subprob\text{-}algebra\ (count\text{-}space\ UNIV))$
⟨*proof*⟩

lemma *bind-measure-pmf-cong*:
assumes $\bigwedge x.\ A\ x \in space\ (subprob\text{-}algebra\ N)$ $\bigwedge x.\ B\ x \in space\ (subprob\text{-}algebra\ N)$
assumes $\bigwedge i.\ i \in set\text{-}pmf\ x \Longrightarrow A\ i = B\ i$
shows $bind\ (measure\text{-}pmf\ x)\ A = bind\ (measure\text{-}pmf\ x)\ B$
⟨*proof*⟩

lift-definition *bind-pmf* :: $'a\ pmf \Rightarrow ('a \Rightarrow 'b\ pmf\ ) \Rightarrow 'b\ pmf$ **is** *bind*
⟨*proof*⟩

adhoc-overloading *Monad-Syntax.bind* ⇌ *bind-pmf*

lemma *ennreal-pmf-bind*: $pmf\ (bind\text{-}pmf\ N\ f)\ i = (\int^+ x.\ pmf\ (f\ x)\ i\ \partial measure\text{-}pmf\ N)$
⟨*proof*⟩

lemma *pmf-bind*: $pmf\ (bind\text{-}pmf\ N\ f)\ i = (\int x.\ pmf\ (f\ x)\ i\ \partial measure\text{-}pmf\ N)$
⟨*proof*⟩

lemma *bind-pmf-const*[*simp*]: *bind-pmf M* (λ*x*. *c*) = *c*
⟨*proof*⟩

lemma *set-bind-pmf*[*simp*]: *set-pmf* (*bind-pmf M N*) = (⋃ *M*∈*set-pmf M*. *set-pmf* (*N M*))
⟨*proof*⟩

lemma *bind-pmf-cong* [*fundef-cong*]:
assumes *p* = *q*
shows (⋀*x*. *x* ∈ *set-pmf q* ⟹ *f x* = *g x*) ⟹ *bind-pmf p f* = *bind-pmf q g*
⟨*proof*⟩

lemma *bind-pmf-cong-simp*:
p = *q* ⟹ (⋀*x*. *x* ∈ *set-pmf q* =*simp*=> *f x* = *g x*) ⟹ *bind-pmf p f* = *bind-pmf q g*
⟨*proof*⟩

lemma *measure-pmf-bind*: *measure-pmf* (*bind-pmf M f*) = (*measure-pmf M* ⤜ (λ*x*. *measure-pmf* (*f x*)))
⟨*proof*⟩

lemma *nn-integral-bind-pmf*[*simp*]: (∫⁺*x*. *f x* ∂*bind-pmf M N*) = (∫⁺*x*. ∫⁺*y*. *f y* ∂*N x* ∂*M*)
⟨*proof*⟩

lemma *emeasure-bind-pmf*[*simp*]: *emeasure* (*bind-pmf M N*) *X* = (∫⁺*x*. *emeasure* (*N x*) *X* ∂*M*)
⟨*proof*⟩

lift-definition *return-pmf* :: *'a* ⇒ *'a pmf* **is** *return* (*count-space UNIV*)
⟨*proof*⟩

lemma *bind-return-pmf*: *bind-pmf* (*return-pmf x*) *f* = *f x*
⟨*proof*⟩

lemma *set-return-pmf*[*simp*]: *set-pmf* (*return-pmf x*) = {*x*}
⟨*proof*⟩

lemma *bind-return-pmf'*: *bind-pmf N return-pmf* = *N*
⟨*proof*⟩

lemma *bind-assoc-pmf*: *bind-pmf* (*bind-pmf A B*) *C* = *bind-pmf A* (λ*x*. *bind-pmf* (*B x*) *C*)
⟨*proof*⟩

definition *map-pmf f M* = *bind-pmf M* (λ*x*. *return-pmf* (*f x*))

lemma *map-bind-pmf*: *map-pmf f* (*bind-pmf M g*) = *bind-pmf M* (λ*x*. *map-pmf f*

$(g\ x))$
⟨*proof*⟩

lemma *bind-map-pmf*: $bind\text{-}pmf\ (map\text{-}pmf\ f\ M)\ g = bind\text{-}pmf\ M\ (\lambda x.\ g\ (f\ x))$
⟨*proof*⟩

lemma *map-pmf-transfer*[*transfer-rule*]:
$rel\text{-}fun\ (=)\ (rel\text{-}fun\ cr\text{-}pmf\ cr\text{-}pmf)\ (\lambda f\ M.\ distr\ M\ (count\text{-}space\ UNIV)\ f)\ map\text{-}pmf\ map\text{-}pmf$
⟨*proof*⟩

lemma *map-pmf-rep-eq*:
$measure\text{-}pmf\ (map\text{-}pmf\ f\ M) = distr\ (measure\text{-}pmf\ M)\ (count\text{-}space\ UNIV)\ f$
⟨*proof*⟩

lemma *map-pmf-id*[*simp*]: $map\text{-}pmf\ id = id$
⟨*proof*⟩

lemma *map-pmf-ident*[*simp*]: $map\text{-}pmf\ (\lambda x.\ x) = (\lambda x.\ x)$
⟨*proof*⟩

lemma *map-pmf-compose*: $map\text{-}pmf\ (f \circ g) = map\text{-}pmf\ f \circ map\text{-}pmf\ g$
⟨*proof*⟩

lemma *map-pmf-comp*: $map\text{-}pmf\ f\ (map\text{-}pmf\ g\ M) = map\text{-}pmf\ (\lambda x.\ f\ (g\ x))\ M$
⟨*proof*⟩

lemma *map-pmf-cong*: $p = q \Longrightarrow (\bigwedge x.\ x \in set\text{-}pmf\ q \Longrightarrow f\ x = g\ x) \Longrightarrow map\text{-}pmf\ f\ p = map\text{-}pmf\ g\ q$
⟨*proof*⟩

lemma *pmf-set-map*: $set\text{-}pmf \circ map\text{-}pmf\ f = (`)\ f \circ set\text{-}pmf$
⟨*proof*⟩

lemma *set-map-pmf*[*simp*]: $set\text{-}pmf\ (map\text{-}pmf\ f\ M) = f`set\text{-}pmf\ M$
⟨*proof*⟩

lemma *emeasure-map-pmf*[*simp*]: $emeasure\ (map\text{-}pmf\ f\ M)\ X = emeasure\ M\ (f -` X)$
⟨*proof*⟩

lemma *measure-map-pmf*[*simp*]: $measure\ (map\text{-}pmf\ f\ M)\ X = measure\ M\ (f -` X)$
⟨*proof*⟩

lemma *nn-integral-map-pmf*[*simp*]: $(\int^+ x.\ f\ x\ \partial map\text{-}pmf\ g\ M) = (\int^+ x.\ f\ (g\ x)\ \partial M)$
⟨*proof*⟩

lemma *ennreal-pmf-map*: *pmf* (*map-pmf f p*) *x* = (∫⁺ *y*. *indicator* (*f* −‘ {*x*}) *y* ∂*measure-pmf p*)
⟨*proof*⟩

lemma *pmf-map*: *pmf* (*map-pmf f p*) *x* = *measure p* (*f* −‘ {*x*})
⟨*proof*⟩

lemma *nn-integral-pmf*: (∫⁺ *x*. *pmf p x* ∂*count-space A*) = *emeasure* (*measure-pmf p*) *A*
⟨*proof*⟩

lemma *integral-map-pmf*[*simp*]:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *integral*^{L} (*map-pmf g p*) *f* = *integral*^{L} *p* (λ*x*. *f* (*g x*))
⟨*proof*⟩

lemma *integrable-map-pmf-eq* [*simp*]:
fixes *g* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *integrable* (*map-pmf f p*) *g* ⟷ *integrable* (*measure-pmf p*) (λ*x*. *g* (*f x*))
⟨*proof*⟩

lemma *integrable-map-pmf* [*intro*]:
fixes *g* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *integrable* (*measure-pmf p*) (λ*x*. *g* (*f x*)) ⟹ *integrable* (*map-pmf f p*) *g*
⟨*proof*⟩

lemma *pmf-abs-summable* [*intro*]: *pmf p abs-summable-on A*
⟨*proof*⟩

lemma *measure-pmf-conv-infsetsum*: *measure* (*measure-pmf p*) *A* = *infsetsum* (*pmf p*) *A*
⟨*proof*⟩

lemma *infsetsum-pmf-eq-1*:
assumes *set-pmf p* ⊆ *A*
shows *infsetsum* (*pmf p*) *A* = *1*
⟨*proof*⟩

lemma *map-return-pmf* [*simp*]: *map-pmf f* (*return-pmf x*) = *return-pmf* (*f x*)
⟨*proof*⟩

lemma *map-pmf-const*[*simp*]: *map-pmf* (λ-. *c*) *M* = *return-pmf c*
⟨*proof*⟩

lemma *pmf-return* [*simp*]: *pmf* (*return-pmf x*) *y* = *indicator* {*y*} *x*
⟨*proof*⟩

lemma *nn-integral-return-pmf*[*simp*]: *0* ≤ *f x* ⟹ (∫⁺*x*. *f x* ∂*return-pmf x*) = *f x*
⟨*proof*⟩

lemma *emeasure-return-pmf*[*simp*]: *emeasure* (*return-pmf x*) *X* = *indicator X x*
⟨*proof*⟩

lemma *measure-return-pmf* [*simp*]: *measure-pmf.prob* (*return-pmf x*) *A* = *indicator A x*
⟨*proof*⟩

lemma *return-pmf-inj*[*simp*]: *return-pmf x* = *return-pmf y* ⟷ *x* = *y*
⟨*proof*⟩

lemma *map-pmf-eq-return-pmf-iff*:
map-pmf f p = *return-pmf x* ⟷ (∀ *y* ∈ *set-pmf p*. *f y* = *x*)
⟨*proof*⟩

definition *pair-pmf A B* = *bind-pmf A* (λ*x*. *bind-pmf B* (λ*y*. *return-pmf* (*x*, *y*)))

lemma *pmf-pair*: *pmf* (*pair-pmf M N*) (*a*, *b*) = *pmf M a* ∗ *pmf N b*
⟨*proof*⟩

lemma *set-pair-pmf*[*simp*]: *set-pmf* (*pair-pmf A B*) = *set-pmf A* × *set-pmf B*
⟨*proof*⟩

lemma *measure-pmf-in-subprob-space*[*measurable* (*raw*)]:
measure-pmf M ∈ *space* (*subprob-algebra* (*count-space UNIV*))
⟨*proof*⟩

lemma *nn-integral-pair-pmf′*: (∫⁺*x*. *f x* ∂*pair-pmf A B*) = (∫⁺*a*. ∫⁺*b*. *f* (*a*, *b*) ∂*B* ∂*A*)
⟨*proof*⟩

lemma *bind-pair-pmf*:
assumes *M*[*measurable*]: *M* ∈ *measurable* (*count-space UNIV* ⨂$_M$ *count-space UNIV*) (*subprob-algebra N*)
shows *measure-pmf* (*pair-pmf A B*) ⤜ *M* = (*measure-pmf A* ⤜ (λ*x*. *measure-pmf B* ⤜ (λ*y*. *M* (*x*, *y*))))
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *map-fst-pair-pmf*: *map-pmf fst* (*pair-pmf A B*) = *A*
⟨*proof*⟩

lemma *map-snd-pair-pmf*: *map-pmf snd* (*pair-pmf A B*) = *B*
⟨*proof*⟩

lemma *nn-integral-pmf′*:
inj-on f A ⟹ (∫⁺*x*. *pmf p* (*f x*) ∂*count-space A*) = *emeasure p* (*f* ‘ *A*)
⟨*proof*⟩

lemma *pmf-le-0-iff*[*simp*]: *pmf M p* $\leq$ *0* $\longleftrightarrow$ *pmf M p* = *0*
⟨*proof*⟩

lemma *min-pmf-0*[*simp*]: *min* (*pmf M p*) *0* = *0 min 0* (*pmf M p*) = *0*
⟨*proof*⟩

lemma *pmf-eq-0-set-pmf*: *pmf M p* = *0* $\longleftrightarrow$ *p* $\notin$ *set-pmf M*
⟨*proof*⟩

lemma *pmf-map-inj*: *inj-on f* (*set-pmf M*) $\Longrightarrow$ *x* $\in$ *set-pmf M* $\Longrightarrow$ *pmf* (*map-pmf f M*) (*f x*) = *pmf M x*
⟨*proof*⟩

lemma *pair-return-pmf* [*simp*]: *pair-pmf* (*return-pmf x*) (*return-pmf y*) = *return-pmf* (*x*, *y*)
⟨*proof*⟩

lemma *pmf-map-inj'*: *inj f* $\Longrightarrow$ *pmf* (*map-pmf f M*) (*f x*) = *pmf M x*
⟨*proof*⟩

lemma *expectation-pair-pmf-fst* [*simp*]:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
shows *measure-pmf.expectation* (*pair-pmf p q*) (λ*x*. *f* (*fst x*)) = *measure-pmf.expectation p f*
⟨*proof*⟩

lemma *expectation-pair-pmf-snd* [*simp*]:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
shows *measure-pmf.expectation* (*pair-pmf p q*) (λ*x*. *f* (*snd x*)) = *measure-pmf.expectation q f*
⟨*proof*⟩

lemma *pmf-map-outside*: *x* $\notin$ *f* ' *set-pmf M* $\Longrightarrow$ *pmf* (*map-pmf f M*) *x* = *0*
⟨*proof*⟩

lemma *measurable-set-pmf*[*measurable*]: *Measurable.pred* (*count-space UNIV*) (λ*x*. *x* $\in$ *set-pmf M*)
⟨*proof*⟩

20.3 PMFs as function

context
fixes *f* :: *'a* $\Rightarrow$ *real*
assumes *nonneg*: $\bigwedge$*x*. *0* $\leq$ *f x*
assumes *prob*: ($\int^+$*x*. *f x* ∂*count-space UNIV*) = *1*
begin

lift-definition *embed-pmf* :: *'a pmf* **is** *density* (*count-space UNIV*) (*ennreal* ∘ *f*)
⟨*proof*⟩

lemma *pmf-embed-pmf*: *pmf embed-pmf x = f x*
⟨*proof*⟩

lemma *set-embed-pmf*: *set-pmf embed-pmf = {x. f x ≠ 0}*
⟨*proof*⟩

end

lemma *embed-pmf-transfer*:
rel-fun (*eq-onp* (λ*f*. (∀ *x*. *0* ≤ *f x*) ∧ (∫⁺*x*. *ennreal* (*f x*) ∂*count-space UNIV*) = *1*)) *pmf-as-measure.cr-pmf* (λ*f*. *density* (*count-space UNIV*) (*ennreal* ∘ *f*)) *embed-pmf*
⟨*proof*⟩

lemma *measure-pmf-eq-density*: *measure-pmf p = density* (*count-space UNIV*) (*pmf p*)
⟨*proof*⟩

lemma *td-pmf-embed-pmf*:
type-definition pmf embed-pmf {*f*::'*a* ⇒ *real*. (∀ *x*. *0* ≤ *f x*) ∧ (∫⁺*x*. *ennreal* (*f x*) ∂*count-space UNIV*) = *1*}
⟨*proof*⟩

end

lemma *nn-integral-measure-pmf*: (∫⁺ *x*. *f x* ∂*measure-pmf p*) = ∫⁺ *x*. *ennreal* (*pmf p x*) ∗ *f x* ∂*count-space UNIV*
⟨*proof*⟩

lemma *integral-measure-pmf*:
fixes *f* :: '*a* ⇒ '*b*::{*banach*, *second-countable-topology*}
assumes *A*: *finite A*
shows (⋀*a*. *a* ∈ *set-pmf M* ⟹ *f a* ≠ *0* ⟹ *a* ∈ *A*) ⟹ (*LINT x*|*M*. *f x*) = (∑ *a*∈*A*. *pmf M a* ∗$_R$ *f a*)
⟨*proof*⟩

lemma *expectation-return-pmf* [*simp*]:
fixes *f* :: '*a* ⇒ '*b*::{*banach*, *second-countable-topology*}
shows *measure-pmf.expectation* (*return-pmf x*) *f* = *f x*
⟨*proof*⟩

lemma *pmf-expectation-bind*:
fixes *p* :: '*a pmf* **and** *f* :: '*a* ⇒ '*b pmf*
and *h* :: '*b* ⇒ '*c*::{*banach*, *second-countable-topology*}
assumes *finite A* ⋀*x*. *x* ∈ *A* ⟹ *finite* (*set-pmf* (*f x*)) *set-pmf p* ⊆ *A*
shows *measure-pmf.expectation* (*p* ⤜ *f*) *h* =
(∑ *a*∈*A*. *pmf p a* ∗$_R$ *measure-pmf.expectation* (*f a*) *h*)
⟨*proof*⟩

lemma *continuous-on-LINT-pmf*: — This is dominated convergence!?
fixes $f :: 'i \Rightarrow 'a::topological\text{-}space \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
assumes f: $\bigwedge i.\ i \in set\text{-}pmf\ M \Longrightarrow continuous\text{-}on\ A\ (f\ i)$
and *bnd*: $\bigwedge a\ i.\ a \in A \Longrightarrow i \in set\text{-}pmf\ M \Longrightarrow norm\ (f\ i\ a) \leq B$
shows *continuous-on A* $(\lambda a.\ LINT\ i|M.\ f\ i\ a)$
⟨*proof*⟩

lemma *continuous-on-LBINT*:
fixes $f :: real \Rightarrow real$
assumes f: $\bigwedge b.\ a \leq b \Longrightarrow set\text{-}integrable\ lborel\ \{a..b\}\ f$
shows *continuous-on UNIV* $(\lambda b.\ LBINT\ x{:}\{a..b\}.\ f\ x)$
⟨*proof*⟩

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*
⟨*proof*⟩

end

context
begin

interpretation *pmf-as-function* ⟨*proof*⟩

lemma *pmf-eqI*: $(\bigwedge i.\ pmf\ M\ i = pmf\ N\ i) \Longrightarrow M = N$
⟨*proof*⟩

lemma *pmf-eq-iff*: $M = N \longleftrightarrow (\forall\, i.\ pmf\ M\ i = pmf\ N\ i)$
⟨*proof*⟩

lemma *pmf-neq-exists-less*:
assumes $M \neq N$
shows $\exists\, x.\ pmf\ M\ x < pmf\ N\ x$
⟨*proof*⟩

lemma *bind-commute-pmf*: *bind-pmf A* $(\lambda x.\ bind\text{-}pmf\ B\ (C\ x)) = bind\text{-}pmf\ B\ (\lambda y.$ *bind-pmf A* $(\lambda x.\ C\ x\ y))$
⟨*proof*⟩

lemma *pair-map-pmf1*: *pair-pmf* $(map\text{-}pmf\ f\ A)\ B = map\text{-}pmf\ (apfst\ f)\ (pair\text{-}pmf\ A\ B)$
⟨*proof*⟩

lemma *pair-map-pmf2*: *pair-pmf A* (*map-pmf f B*) = *map-pmf* (*apsnd f*) (*pair-pmf A B*)
⟨*proof*⟩

lemma *map-pair*: *map-pmf* (λ(*a*, *b*). (*f a*, *g b*)) (*pair-pmf A B*) = *pair-pmf* (*map-pmf f A*) (*map-pmf g B*)
⟨*proof*⟩

end

lemma *pair-return-pmf1*: *pair-pmf* (*return-pmf x*) *y* = *map-pmf* (*Pair x*) *y*
⟨*proof*⟩

lemma *pair-return-pmf2*: *pair-pmf x* (*return-pmf y*) = *map-pmf* (λ*x*. (*x*, *y*)) *x*
⟨*proof*⟩

lemma *pair-pair-pmf*: *pair-pmf* (*pair-pmf u v*) *w* = *map-pmf* (λ(*x*, (*y*, *z*)). ((*x*, *y*), *z*)) (*pair-pmf u* (*pair-pmf v w*))
⟨*proof*⟩

lemma *pair-commute-pmf*: *pair-pmf x y* = *map-pmf* (λ(*x*, *y*). (*y*, *x*)) (*pair-pmf y x*)
⟨*proof*⟩

lemma *set-pmf-subset-singleton*: *set-pmf p* ⊆ {*x*} ⟷ *p* = *return-pmf x*
⟨*proof*⟩

lemma *bind-eq-return-pmf*:
bind-pmf p f = *return-pmf x* ⟷ (∀ *y*∈*set-pmf p*. *f y* = *return-pmf x*)
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *pmf-False-conv-True*: *pmf p False* = *1* − *pmf p True*
⟨*proof*⟩

lemma *pmf-True-conv-False*: *pmf p True* = *1* − *pmf p False*
⟨*proof*⟩

20.4 Conditional Probabilities

lemma *measure-pmf-zero-iff*: *measure* (*measure-pmf p*) *s* = *0* ⟷ *set-pmf p* ∩ *s* = {}
⟨*proof*⟩

context
fixes *p* :: *'a pmf* **and** *s* :: *'a set*
assumes *not-empty*: *set-pmf p* ∩ *s* ≠ {}
begin

interpretation *pmf-as-measure* ⟨*proof*⟩

lemma *emeasure-measure-pmf-not-zero*: *emeasure* (*measure-pmf p*) *s* ≠ *0*
⟨*proof*⟩

lemma *measure-measure-pmf-not-zero*: *measure* (*measure-pmf p*) *s* ≠ *0*
⟨*proof*⟩

lift-definition *cond-pmf* :: *'a pmf* **is**
uniform-measure (*measure-pmf p*) *s*
⟨*proof*⟩

lemma *pmf-cond*: *pmf cond-pmf x* = (*if x* ∈ *s then pmf p x* / *measure p s else 0*)
⟨*proof*⟩

lemma *set-cond-pmf*[*simp*]: *set-pmf cond-pmf* = *set-pmf p* ∩ *s*
⟨*proof*⟩

end

lemma *measure-pmf-posI*: *x* ∈ *set-pmf p* ⟹ *x* ∈ *A* ⟹ *measure-pmf.prob p A* >
0
⟨*proof*⟩

lemma *cond-map-pmf*:
assumes *set-pmf p* ∩ *f* −‘ *s* ≠ {}
shows *cond-pmf* (*map-pmf f p*) *s* = *map-pmf f* (*cond-pmf p* (*f* −‘ *s*))
⟨*proof*⟩

lemma *bind-cond-pmf-cancel*:
assumes [*simp*]: ⋀*x*. *x* ∈ *set-pmf p* ⟹ *set-pmf q* ∩ {*y*. *R x y*} ≠ {}
assumes [*simp*]: ⋀*y*. *y* ∈ *set-pmf q* ⟹ *set-pmf p* ∩ {*x*. *R x y*} ≠ {}
assumes [*simp*]: ⋀*x y*. *x* ∈ *set-pmf p* ⟹ *y* ∈ *set-pmf q* ⟹ *R x y* ⟹ *measure*
q {*y*. *R x y*} = *measure p* {*x*. *R x y*}
shows *bind-pmf p* (λ*x*. *cond-pmf q* {*y*. *R x y*}) = *q*
⟨*proof*⟩

20.5 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*: $\bigwedge x\ y.\ x \in$ *set-pmf* $p \Longrightarrow y \in$ *set-pmf* $q \Longrightarrow R\ x\ y \Longrightarrow$ *measure* $p\ \{x.\ R\ x\ y\} =$ *measure* $q\ \{y.\ R\ x\ y\}$
shows *rel-pmf* $R\ p\ q$
⟨*proof*⟩

lemma *rel-pmf-imp-rel-set*: *rel-pmf* $R\ p\ q \Longrightarrow$ *rel-set* R (*set-pmf* p) (*set-pmf* q)
⟨*proof*⟩

lemma *rel-pmfD-measure*:
assumes *rel-R*: *rel-pmf* $R\ p\ q$ **and** R: $\bigwedge a\ b.\ R\ a\ b \Longrightarrow R\ a\ y \longleftrightarrow R\ x\ b$
assumes $x \in$ *set-pmf* $p\ y \in$ *set-pmf* q
shows *measure* $p\ \{x.\ R\ x\ y\} =$ *measure* $q\ \{y.\ R\ x\ y\}$
⟨*proof*⟩

lemma *rel-pmf-measureD*:
assumes *rel-pmf* $R\ p\ q$
shows *measure* (*measure-pmf* p) $A \leq$ *measure* (*measure-pmf* q) $\{y.\ \exists x{\in}A.\ R\ x\ y\}$ (**is** *?lhs* $\leq$ *?rhs*)
⟨*proof*⟩

lemma *rel-pmf-iff-measure*:
assumes *symp* R *transp* R
shows *rel-pmf* $R\ p\ q \longleftrightarrow$
rel-set R (*set-pmf* p) (*set-pmf* q) $\wedge$
$(\forall x{\in}$*set-pmf* $p.\ \forall y{\in}$*set-pmf* $q.\ R\ x\ y \longrightarrow$ *measure* $p\ \{x.\ R\ x\ y\} =$ *measure* $q\ \{y.\ R\ x\ y\})$
⟨*proof*⟩

lemma *quotient-rel-set-disjoint*:
equivp $R \Longrightarrow C \in$ *UNIV* $//\ \{(x,\ y).\ R\ x\ y\} \Longrightarrow$ *rel-set* $R\ A\ B \Longrightarrow A \cap C = \{\} \longleftrightarrow B \cap C = \{\}$
⟨*proof*⟩

lemma *quotientD*: *equiv* $X\ R \Longrightarrow A \in X\ //\ R \Longrightarrow x \in A \Longrightarrow A = R\ ``\ \{x\}$
⟨*proof*⟩

lemma *rel-pmf-iff-equivp*:
assumes *equivp* R
shows *rel-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*⟩

bnf *pmf*: *'a pmf map*: *map-pmf sets*: *set-pmf bd* : *card-suc natLeq rel*: *rel-pmf*
⟨*proof*⟩

lemma *map-pmf-idI*: $(\bigwedge x.\ x \in$ *set-pmf* $p \Longrightarrow f\ x = x) \Longrightarrow$ *map-pmf* $f\ p = p$
⟨*proof*⟩

lemma *rel-pmf-conj*[*simp*]:
rel-pmf (λ*x y*. *P* ∧ *Q x y*) *x y* ⟷ *P* ∧ *rel-pmf Q x y*
rel-pmf (λ*x y*. *Q x y* ∧ *P*) *x y* ⟷ *P* ∧ *rel-pmf Q x y*
⟨*proof*⟩

lemma *rel-pmf-top*[*simp*]: *rel-pmf top* = *top*
⟨*proof*⟩

lemma *rel-pmf-return-pmf1*: *rel-pmf R* (*return-pmf x*) *M* ⟷ (∀ *a*∈*M*. *R x a*)
⟨*proof*⟩

lemma *rel-pmf-return-pmf2*: *rel-pmf R M* (*return-pmf x*) ⟷ (∀ *a*∈*M*. *R a x*)
⟨*proof*⟩

lemma *rel-return-pmf*[*simp*]: *rel-pmf R* (*return-pmf x1*) (*return-pmf x2*) = *R x1 x2*
⟨*proof*⟩

lemma *rel-pmf-False*[*simp*]: *rel-pmf* (λ*x y*. *False*) *x y* = *False*
⟨*proof*⟩

lemma *rel-pmf-rel-prod*:
rel-pmf (*rel-prod R S*) (*pair-pmf A A′*) (*pair-pmf B B′*) ⟷ *rel-pmf R A B* ∧ *rel-pmf S A′ B′*
⟨*proof*⟩

lemma *rel-pmf-reflI*:
assumes ⋀*x*. *x* ∈ *set-pmf p* ⟹ *P x x*
shows *rel-pmf P p p*
⟨*proof*⟩

lemma *rel-pmf-bij-betw*:
assumes *f*: *bij-betw f* (*set-pmf p*) (*set-pmf q*)
and *eq*: ⋀*x*. *x* ∈ *set-pmf p* ⟹ *pmf p x* = *pmf q* (*f x*)
shows *rel-pmf* (λ*x y*. *f x* = *y*) *p q*
⟨*proof*⟩

context
begin

interpretation *pmf-as-measure* ⟨*proof*⟩

definition *join-pmf M* = *bind-pmf M* (λ*x*. *x*)

lemma *bind-eq-join-pmf*: *bind-pmf M f* = *join-pmf* (*map-pmf f M*)
⟨*proof*⟩

lemma *join-eq-bind-pmf*: *join-pmf M* = *bind-pmf M id*
⟨*proof*⟩

lemma *pmf-join*: *pmf* (*join-pmf N*) *i* = (∫ *M*. *pmf M i* ∂*measure-pmf N*)
⟨*proof*⟩

lemma *ennreal-pmf-join*: *ennreal* (*pmf* (*join-pmf N*) *i*) = (∫⁺ *M*. *pmf M i* ∂*measure-pmf N*)
⟨*proof*⟩

lemma *set-pmf-join-pmf*[*simp*]: *set-pmf* (*join-pmf f*) = (⋃ *p*∈*set-pmf f*. *set-pmf p*)
⟨*proof*⟩

lemma *join-return-pmf*: *join-pmf* (*return-pmf M*) = *M*
⟨*proof*⟩

lemma *map-join-pmf*: *map-pmf f* (*join-pmf AA*) = *join-pmf* (*map-pmf* (*map-pmf f*) *AA*)
⟨*proof*⟩

lemma *join-map-return-pmf*: *join-pmf* (*map-pmf return-pmf A*) = *A*
⟨*proof*⟩

end

lemma *rel-pmf-joinI*:
assumes *rel-pmf* (*rel-pmf P*) *p q*
shows *rel-pmf P* (*join-pmf p*) (*join-pmf q*)
⟨*proof*⟩

lemma *rel-pmf-bindI*:
assumes *pq*: *rel-pmf R p q*
and *fg*: ⋀*x y*. *R x y* ⟹ *rel-pmf P* (*f x*) (*g y*)
shows *rel-pmf P* (*bind-pmf p f*) (*bind-pmf q g*)
⟨*proof*⟩

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, https://doi.org/10.1016/0304-3975(80)90003-1

lemma
assumes ∗: *rel-pmf R p q*
and *refl*: *reflp R* **and** *trans*: *transp R*
shows *measure-Ici*: *measure p* {*y*. *R x y*} ≤ *measure q* {*y*. *R x y*} (**is** *?thesis1*)
and *measure-Ioi*: *measure p* {*y*. *R x y* ∧ ¬ *R y x*} ≤ *measure q* {*y*. *R x y* ∧ ¬ *R y x*} (**is** *?thesis2*)
⟨*proof*⟩

lemma *rel-pmf-inf*:
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*
shows *rel-pmf* (*inf R* R^{-1-1}) *p q*
⟨*proof*⟩

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*⟩

lemma *reflp-rel-pmf*: *reflp R* ⟹ *reflp* (*rel-pmf R*)
⟨*proof*⟩

lemma *antisymp-rel-pmf*:
⟦ *reflp R*; *transp R*; *antisymp R* ⟧
⟹ *antisymp* (*rel-pmf R*)
⟨*proof*⟩

lemma *transp-rel-pmf*:
assumes *transp R*
shows *transp* (*rel-pmf R*)
⟨*proof*⟩

20.6 Distributions

context
begin

interpretation *pmf-as-function* ⟨*proof*⟩

20.6.1 Bernoulli Distribution

lift-definition *bernoulli-pmf* :: *real* ⇒ *bool pmf* **is**
λ*p b*. ((λ*p*. *if b then p else 1* − *p*) ∘ *min 1* ∘ *max 0*) *p*
⟨*proof*⟩

lemma *pmf-bernoulli-True*[*simp*]: *0* ≤ *p* ⟹ *p* ≤ *1* ⟹ *pmf* (*bernoulli-pmf p*) *True* = *p*
⟨*proof*⟩

lemma *pmf-bernoulli-False*[*simp*]: *0* ≤ *p* ⟹ *p* ≤ *1* ⟹ *pmf* (*bernoulli-pmf p*) *False* = *1* − *p*
⟨*proof*⟩

lemma *set-pmf-bernoulli*[*simp*]: *0* < *p* ⟹ *p* < *1* ⟹ *set-pmf* (*bernoulli-pmf p*) = *UNIV*

⟨*proof*⟩

lemma *nn-integral-bernoulli-pmf*[*simp*]:
assumes [*simp*]: *0* ≤ *p p* ≤ *1* ⋀*x*. *0* ≤ *f x*
shows (∫⁺*x*. *f x* ∂*bernoulli-pmf p*) = *f True* ∗ *p* + *f False* ∗ (*1* − *p*)
⟨*proof*⟩

lemma *integral-bernoulli-pmf*[*simp*]:
assumes [*simp*]: *0* ≤ *p p* ≤ *1*
shows (∫ *x*. *f x* ∂*bernoulli-pmf p*) = *f True* ∗ *p* + *f False* ∗ (*1* − *p*)
⟨*proof*⟩

lemma *pmf-bernoulli-half* [*simp*]: *pmf* (*bernoulli-pmf* (*1* / *2*)) *x* = *1* / *2*
⟨*proof*⟩

lemma *measure-pmf-bernoulli-half*: *measure-pmf* (*bernoulli-pmf* (*1* / *2*)) = *uniform-count-measure UNIV*
⟨*proof*⟩

20.6.2 Geometric Distribution

context
fixes *p* :: *real* **assumes** *p*[*arith*]: *0* < *p p* ≤ *1*
begin

lift-definition *geometric-pmf* :: *nat pmf* **is** λ*n*. (*1* − *p*)^*n* ∗ *p*
⟨*proof*⟩

lemma *pmf-geometric*[*simp*]: *pmf geometric-pmf n* = (*1* − *p*)^*n* ∗ *p*
⟨*proof*⟩

end

lemma *geometric-pmf-1* [*simp*]: *geometric-pmf 1* = *return-pmf 0*
⟨*proof*⟩

lemma *set-pmf-geometric*: *0* < *p* ⟹ *p* < *1* ⟹ *set-pmf* (*geometric-pmf p*) = *UNIV*
⟨*proof*⟩

lemma *geometric-sums-times-n*:
fixes *c*::'*a*::{*banach*,*real-normed-field*}
assumes *norm c* < *1*
shows (λ*n*. *c*^*n* ∗ *of-nat n*) *sums* (*c* / (*1* − *c*)²)
⟨*proof*⟩

lemma *geometric-sums-times-norm*:
fixes *c*::'*a*::{*banach*,*real-normed-field*}
assumes *norm c* < *1*

shows $(\lambda n.\ norm\ (c \hat{}\ n) * of\text{-}nat\ n))\ sums\ (norm\ c\ /\ (1 - norm\ c)^2)$
⟨*proof*⟩

lemma *integrable-real-geometric-pmf*:
assumes $p \in \{0<..1\}$
shows *integrable* (*geometric-pmf p*) *real*
⟨*proof*⟩

lemma *expectation-geometric-pmf*:
assumes $p \in \{0<..1\}$
shows *measure-pmf.expectation* (*geometric-pmf p*) *real* $= (1 - p)\ /\ p$
⟨*proof*⟩

lemma *geometric-bind-pmf-unfold*:
assumes $p \in \{0<..1\}$
shows *geometric-pmf p* =
do {*b* ← *bernoulli-pmf p*;
if b then return-pmf 0 else map-pmf Suc (*geometric-pmf p*)}
⟨*proof*⟩

20.6.3 Uniform Multiset Distribution

context
fixes $M :: 'a\ multiset$ **assumes** *M-not-empty*: $M \neq \{\#\}$
begin

lift-definition *pmf-of-multiset* :: *'a pmf* **is** $\lambda x.\ count\ M\ x\ /\ size\ M$
⟨*proof*⟩

lemma *pmf-of-multiset*[*simp*]: *pmf pmf-of-multiset x* = *count M x / size M*
⟨*proof*⟩

lemma *set-pmf-of-multiset*[*simp*]: *set-pmf pmf-of-multiset* = *set-mset M*
⟨*proof*⟩

end

20.6.4 Uniform Distribution

context
fixes $S :: 'a\ set$ **assumes** *S-not-empty*: $S \neq \{\}$ **and** *S-finite*: *finite S*
begin

lift-definition *pmf-of-set* :: *'a pmf* **is** $\lambda x.\ indicator\ S\ x\ /\ card\ S$
⟨*proof*⟩

lemma *pmf-of-set*[*simp*]: *pmf pmf-of-set x* = *indicator S x / card S*
⟨*proof*⟩

lemma *set-pmf-of-set*[*simp*]: *set-pmf pmf-of-set* = *S*

⟨*proof*⟩

lemma *emeasure-pmf-of-set-space*[*simp*]: *emeasure pmf-of-set S = 1*
⟨*proof*⟩

lemma *nn-integral-pmf-of-set*: *nn-integral* (*measure-pmf pmf-of-set*) *f* = *sum f S* / *card S*
⟨*proof*⟩

lemma *integral-pmf-of-set*: $integral^L$ (*measure-pmf pmf-of-set*) *f* = *sum f S* / *card S*
⟨*proof*⟩

lemma *emeasure-pmf-of-set*: *emeasure* (*measure-pmf pmf-of-set*) *A* = *card* (*S* ∩ *A*) / *card S*
⟨*proof*⟩

lemma *measure-pmf-of-set*: *measure* (*measure-pmf pmf-of-set*) *A* = *card* (*S* ∩ *A*) / *card S*
⟨*proof*⟩

end

lemma *pmf-expectation-bind-pmf-of-set*:
fixes *A* :: *'a set* **and** *f* :: *'a* ⇒ *'b pmf*
and *h* :: *'b* ⇒ *'c*::{*banach*, *second-countable-topology*}
assumes *A* ≠ {} *finite A* ⋀*x*. *x* ∈ *A* ⟹ *finite* (*set-pmf* (*f x*))
shows *measure-pmf.expectation* (*pmf-of-set A* ⤜ *f*) *h* =
(∑ *a*∈*A*. *measure-pmf.expectation* (*f a*) *h* /$_R$ *real* (*card A*))
⟨*proof*⟩

lemma *map-pmf-of-set*:
assumes *finite A A* ≠ {}
shows *map-pmf f* (*pmf-of-set A*) = *pmf-of-multiset* (*image-mset f* (*mset-set A*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *pmf-bind-pmf-of-set*:
assumes *A* ≠ {} *finite A*
shows *pmf* (*bind-pmf* (*pmf-of-set A*) *f*) *x* =
(∑ *xa*∈*A*. *pmf* (*f xa*) *x*) / *real-of-nat* (*card A*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *pmf-of-set-singleton*: *pmf-of-set* {*x*} = *return-pmf x*
⟨*proof*⟩

lemma *map-pmf-of-set-inj*:
assumes *f*: *inj-on f A*

and [*simp*]: $A \neq \{\}$ *finite A*
shows *map-pmf f* (*pmf-of-set A*) = *pmf-of-set* (*f* ' *A*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *map-pmf-of-set-bij-betw*:
assumes *bij-betw f A B A* ≠ {} *finite A*
shows *map-pmf f* (*pmf-of-set A*) = *pmf-of-set B*
⟨*proof*⟩

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* (⋃(*f* ' *A*)) *A* ≠ {} ⋀*x*. *x* ∈ *A* ⟹ *f x* ≠ {}
⋀*x*. *x* ∈ *A* ⟹ *card* (*f x*) = *n disjoint-family-on f A*
shows *pmf-of-set* (⋃(*f* ' *A*)) = *do* {*x* ← *pmf-of-set A*; *pmf-of-set* (*f x*)}
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *bernoulli-pmf-half-conv-pmf-of-set*: *bernoulli-pmf* (*1* / *2*) = *pmf-of-set UNIV*
⟨*proof*⟩

20.6.5 Poisson Distribution

context
fixes *rate* :: *real* **assumes** *rate-pos*: *0* < *rate*
begin

lift-definition *poisson-pmf* :: *nat pmf* **is** λ*k*. *rate* ^ *k* / *fact k* ∗ *exp* (−*rate*)
⟨*proof*⟩

lemma *pmf-poisson*[*simp*]: *pmf poisson-pmf k* = *rate* ^ *k* / *fact k* ∗ *exp* (−*rate*)
⟨*proof*⟩

lemma *set-pmf-poisson*[*simp*]: *set-pmf poisson-pmf* = *UNIV*
⟨*proof*⟩

end

20.6.6 Binomial Distribution

context
fixes *n* :: *nat* **and** *p* :: *real* **assumes** *p-nonneg*: *0* ≤ *p* **and** *p-le-1*: *p* ≤ *1*
begin

lift-definition *binomial-pmf* :: *nat pmf* **is** λ*k*. (*n choose k*) ∗ *p*^*k* ∗ (*1* − *p*)^(*n* − *k*)
⟨*proof*⟩

lemma *pmf-binomial*[*simp*]: *pmf binomial-pmf k* = (*n choose k*) * *p*^*k* * (*1* − *p*)^(*n* − *k*)
⟨*proof*⟩

lemma *set-pmf-binomial-eq*: *set-pmf binomial-pmf* = (*if p* = *0 then* {*0*} *else if p* = *1 then* {*n*} *else* {.. *n*})
⟨*proof*⟩

end

end

lemma *set-pmf-binomial-0*[*simp*]: *set-pmf* (*binomial-pmf n 0*) = {*0*}
⟨*proof*⟩

lemma *set-pmf-binomial-1*[*simp*]: *set-pmf* (*binomial-pmf n 1*) = {*n*}
⟨*proof*⟩

lemma *set-pmf-binomial*[*simp*]: *0* < *p* ⟹ *p* < *1* ⟹ *set-pmf* (*binomial-pmf n p*) = {..*n*}
⟨*proof*⟩

lemma *finite-set-pmf-binomial-pmf* [*intro*]: *p* ∈ {*0..1*} ⟹ *finite* (*set-pmf* (*binomial-pmf n p*))
⟨*proof*⟩

lemma *expectation-binomial-pmf'*:
fixes *f* :: *nat* ⇒ 'a :: {*banach*, *second-countable-topology*}
assumes *p*: *p* ∈ {*0..1*}
shows *measure-pmf.expectation* (*binomial-pmf n p*) *f* =
(∑ *k*≤*n*. (*real* (*n choose k*) * *p* ^ *k* * (*1* − *p*) ^ (*n* − *k*)) *$_R$ f k*)
⟨*proof*⟩

lemma *integrable-binomial-pmf* [*simp*, *intro*]:
fixes *f* :: *nat* ⇒ 'a :: {*banach*, *second-countable-topology*}
assumes *p*: *p* ∈ {*0..1*}
shows *integrable* (*binomial-pmf n p*) *f*
⟨*proof*⟩

context includes *lifting-syntax*
begin

lemma *bind-pmf-parametric* [*transfer-rule*]:
(*rel-pmf A* ===> (*A* ===> *rel-pmf B*) ===> *rel-pmf B*) *bind-pmf bind-pmf*
⟨*proof*⟩

lemma *return-pmf-parametric* [*transfer-rule*]: (*A* ===> *rel-pmf A*) *return-pmf return-pmf*

⟨*proof*⟩

end

primrec *replicate-pmf* :: *nat* ⇒ *'a pmf* ⇒ *'a list pmf* **where**
  replicate-pmf 0 - = return-pmf []
| *replicate-pmf* (*Suc n*) *p* = *do* {*x* ← *p*; *xs* ← *replicate-pmf n p*; *return-pmf* (*x*#*xs*)}

lemma *replicate-pmf-1*: *replicate-pmf 1 p* = *map-pmf* (λ*x*. [*x*]) *p*
  ⟨*proof*⟩

lemma *set-replicate-pmf*:
  set-pmf (*replicate-pmf n p*) = {*xs*∈*lists* (*set-pmf p*). *length xs* = *n*}
  ⟨*proof*⟩

lemma *replicate-pmf-distrib*:
  replicate-pmf (*m* + *n*) *p* =
    do {*xs* ← *replicate-pmf m p*; *ys* ← *replicate-pmf n p*; *return-pmf* (*xs* @ *ys*)}
  ⟨*proof*⟩

lemma *power-diff'*:
  assumes *b* ≤ *a*
  shows *x* ^ (*a* − *b*) = (*if x* = *0* ∧ *a* = *b then 1 else x* ^ *a* / (*x*::*'a*::*field*) ^ *b*)
⟨*proof*⟩

lemma *binomial-pmf-Suc*:
  assumes *p* ∈ {*0..1*}
  shows *binomial-pmf* (*Suc n*) *p* =
          do {*b* ← *bernoulli-pmf p*;
              k ← *binomial-pmf n p*;
              return-pmf ((*if b then 1 else 0*) + *k*)} (**is** - = *?rhs*)
⟨*proof*⟩

lemma *binomial-pmf-0*: *p* ∈ {*0..1*} ⟹ *binomial-pmf 0 p* = *return-pmf 0*
  ⟨*proof*⟩

lemma *binomial-pmf-altdef*:
  assumes *p* ∈ {*0..1*}
  shows *binomial-pmf n p* = *map-pmf* (*length* ∘ *filter id*) (*replicate-pmf n* (*bernoulli-pmf p*))
  ⟨*proof*⟩

20.7 Negative Binomial distribution

The negative binomial distribution counts the number of times a weighted coin comes up tails before having come up heads n times. In other words: how many failures do we see before seeing the n-th success?

An alternative view is that the negative binomial distribution is the sum of n i.i.d. geometric variables (this is the definition that we use).

Note that there are sometimes different conventions for this distributions in the literature; for instance, sometimes the number of *attempts* is counted instead of the number of failures. This only shifts the entire distribution by a constant number and is thus not a big difference. I think that the convention we use is the most natural one since the support of the distribution starts at 0, whereas for the other convention it starts at n.

primrec *neg-binomial-pmf* :: *nat* ⇒ *real* ⇒ *nat pmf* **where**
  neg-binomial-pmf 0 p = return-pmf 0
| *neg-binomial-pmf (Suc n) p =*
    map-pmf (λ(x,y). (x + y)) (pair-pmf (geometric-pmf p) (neg-binomial-pmf n p))

lemma *neg-binomial-pmf-Suc-0* [*simp*]: *neg-binomial-pmf (Suc 0) p = geometric-pmf p*
  ⟨*proof*⟩

lemmas *neg-binomial-pmf-Suc* [*simp del*] = *neg-binomial-pmf.simps*(*2*)

lemma *neg-binomial-prob-1* [*simp*]: *neg-binomial-pmf n 1 = return-pmf 0*
  ⟨*proof*⟩

We can now show the aforementioned intuition about counting the failures before the n-th success with the following recurrence:

lemma *neg-binomial-pmf-unfold*:
  assumes *p*: *p* ∈ {*0<..1*}
  shows *neg-binomial-pmf (Suc n) p =*
        do {b ← bernoulli-pmf p;
            if b then neg-binomial-pmf n p else map-pmf Suc (neg-binomial-pmf (Suc n) p)}
  (**is** *- = ?rhs*)
  ⟨*proof*⟩

Next, we show an explicit formula for the probability mass function of the negative binomial distribution:

lemma *pmf-neg-binomial*:
  assumes *p*: *p* ∈ {*0<..1*}
  shows *pmf (neg-binomial-pmf n p) k = real ((k + n − 1) choose k) ∗ p ^ n ∗ (1 − p) ^ k*
⟨*proof*⟩

lemma *gbinomial-0-left*: *0 gchoose k = (if k = 0 then 1 else 0)*
  ⟨*proof*⟩

The following alternative formula highlights why it is called 'negative binomial distribution':

lemma *pmf-neg-binomial'*:
assumes *p*: $p \in \{0<..1\}$
shows *pmf* (*neg-binomial-pmf n p*) *k* = (−1) ^ *k* ∗ ((−*real n*) *gchoose k*) ∗ *p* ^ *n* ∗ (1 − *p*) ^ *k*
⟨*proof*⟩

The cumulative distribution function of the negative binomial distribution can be expressed in terms of that of the 'normal' binomial distribution.

lemma *prob-neg-binomial-pmf-atMost*:
assumes *p*: $p \in \{0<..1\}$
shows *measure-pmf.prob* (*neg-binomial-pmf n p*) {..*k*} =
measure-pmf.prob (*binomial-pmf* (*n* + *k*) (1 − *p*)) {..*k*}
⟨*proof*⟩

lemma *prob-neg-binomial-pmf-lessThan*:
assumes *p*: $p \in \{0<..1\}$
shows *measure-pmf.prob* (*neg-binomial-pmf n p*) {..<*k*} =
measure-pmf.prob (*binomial-pmf* (*n* + *k* − 1) (1 − *p*)) {..<*k*}
⟨*proof*⟩

The expected value of the negative binomial distribution is $n(1-p)/p$:

lemma *nn-integral-neg-binomial-pmf-real*:
assumes *p*: $p \in \{0<..1\}$
shows *nn-integral* (*measure-pmf* (*neg-binomial-pmf n p*)) *of-nat* = *ennreal* (*n* ∗ (1 − *p*) / *p*)
⟨*proof*⟩

lemma *integrable-neg-binomial-pmf-real*:
assumes *p*: $p \in \{0<..1\}$
shows *integrable* (*measure-pmf* (*neg-binomial-pmf n p*)) *real*
⟨*proof*⟩

lemma *expectation-neg-binomial-pmf*:
assumes *p*: $p \in \{0<..1\}$
shows *measure-pmf.expectation* (*neg-binomial-pmf n p*) *real* = *n* ∗ (1 − *p*) / *p*
⟨*proof*⟩

20.8 PMFs from association lists

definition *pmf-of-list* :: ('*a* × *real*) *list* ⇒ '*a pmf* **where**
pmf-of-list xs = *embed-pmf* (λ*x*. *sum-list* (*map snd* (*filter* (λ*z*. *fst z* = *x*) *xs*)))

definition *pmf-of-list-wf* **where**
pmf-of-list-wf xs ⟷ (∀ *x*∈*set* (*map snd xs*) . *x* ≥ 0) ∧ *sum-list* (*map snd xs*) = 1

lemma *pmf-of-list-wfI*:
(⋀*x*. *x* ∈ *set* (*map snd xs*) ⟹ *x* ≥ 0) ⟹ *sum-list* (*map snd xs*) = 1 ⟹ *pmf-of-list-wf xs*

⟨*proof*⟩

context
begin

private lemma *pmf-of-list-aux*:
assumes $\bigwedge x.\ x \in set\ (map\ snd\ xs) \Longrightarrow x \geq 0$
assumes $sum\text{-}list\ (map\ snd\ xs) = 1$
shows $(\int^+ x.\ ennreal\ (sum\text{-}list\ (map\ snd\ [z \leftarrow xs\ .\ fst\ z = x]))\ \partial count\text{-}space\ UNIV) = 1$
⟨*proof*⟩

lemma *pmf-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows $pmf\ (pmf\text{-}of\text{-}list\ xs)\ x = sum\text{-}list\ (map\ snd\ (filter\ (\lambda z.\ fst\ z = x)\ xs))$
⟨*proof*⟩

end

lemma *set-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows $set\text{-}pmf\ (pmf\text{-}of\text{-}list\ xs) \subseteq set\ (map\ fst\ xs)$
⟨*proof*⟩

lemma *finite-set-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows $finite\ (set\text{-}pmf\ (pmf\text{-}of\text{-}list\ xs))$
⟨*proof*⟩

lemma *emeasure-Int-set-pmf*:
$emeasure\ (measure\text{-}pmf\ p)\ (A \cap set\text{-}pmf\ p) = emeasure\ (measure\text{-}pmf\ p)\ A$
⟨*proof*⟩

lemma *measure-Int-set-pmf*:
$measure\ (measure\text{-}pmf\ p)\ (A \cap set\text{-}pmf\ p) = measure\ (measure\text{-}pmf\ p)\ A$
⟨*proof*⟩

lemma *measure-prob-cong-0*:
assumes $\bigwedge x.\ x \in A - B \Longrightarrow pmf\ p\ x = 0$
assumes $\bigwedge x.\ x \in B - A \Longrightarrow pmf\ p\ x = 0$
shows $measure\ (measure\text{-}pmf\ p)\ A = measure\ (measure\text{-}pmf\ p)\ B$
⟨*proof*⟩

lemma *emeasure-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows $emeasure\ (pmf\text{-}of\text{-}list\ xs)\ A = ennreal\ (sum\text{-}list\ (map\ snd\ (filter\ (\lambda x.\ fst\ x \in A)\ xs)))$
⟨*proof*⟩

lemma *measure-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows *measure* (*pmf-of-list xs*) *A* = *sum-list* (*map snd* (*filter* (λ*x*. *fst x* ∈ *A*) *xs*))
⟨*proof*⟩

lemma *sum-list-nonneg-eq-zero-iff*:
fixes *xs* :: 'a :: *linordered-ab-group-add list*
shows (⋀*x*. *x* ∈ *set xs* ⟹ *x* ≥ *0*) ⟹ *sum-list xs* = *0* ⟷ *set xs* ⊆ {*0*}
⟨*proof*⟩

lemma *sum-list-filter-nonzero*:
sum-list (*filter* (λ*x*. *x* ≠ *0*) *xs*) = *sum-list xs*
⟨*proof*⟩

lemma *set-pmf-of-list-eq*:
assumes *pmf-of-list-wf xs* ⋀*x*. *x* ∈ *snd* ' *set xs* ⟹ *x* > *0*
shows *set-pmf* (*pmf-of-list xs*) = *fst* ' *set xs*
⟨*proof*⟩

lemma *pmf-of-list-remove-zeros*:
assumes *pmf-of-list-wf xs*
defines *xs'* ≡ *filter* (λ*z*. *snd z* ≠ *0*) *xs*
shows *pmf-of-list-wf xs' pmf-of-list xs'* = *pmf-of-list xs*
⟨*proof*⟩

end

21 Code generation for PMFs

theory *PMF-Impl*
imports *Probability-Mass-Function HOL−Library.AList-Mapping*
begin

21.1 General code generation setup

definition *pmf-of-mapping* :: ('a, *real*) *mapping* ⇒ '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* (λ- *x*. *x* ≥ *0*)
shows *nn-integral* (*count-space UNIV*) (λ*k*. *ennreal* (*Mapping.lookup-default 0 m k*)) =
ennreal (∑ *k*∈*Mapping.keys m*. *Mapping.lookup-default 0 m k*)
⟨*proof*⟩

lemma *pmf-of-mapping*:
assumes *finite* (*Mapping.keys m*) *All-mapping m* (λ- *p*. *p* ≥ *0*)
assumes (∑ *x*∈*Mapping.keys m*. *Mapping.lookup-default 0 m x*) = *1*
shows *pmf* (*pmf-of-mapping m*) *x* = *Mapping.lookup-default 0 m x*
⟨*proof*⟩

lemma *pmf-of-set-pmf-of-mapping*:
assumes *A* ≠ {} *set xs* = *A distinct xs*
shows *pmf-of-set A* = *pmf-of-mapping* (*Mapping.tabulate xs* (λ-. *1* / *real* (*length xs*)))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lift-definition *mapping-of-pmf* :: *'a pmf* ⇒ (*'a*, *real*) *mapping* **is**
λ*p x*. *if pmf p x* = *0 then None else Some* (*pmf p x*) ⟨*proof*⟩

lemma *lookup-default-mapping-of-pmf*:
Mapping.lookup-default 0 (*mapping-of-pmf p*) *x* = *pmf p x*
⟨*proof*⟩

context
begin

interpretation *pmf-as-function* ⟨*proof*⟩

lemma *nn-integral-pmf-eq-1*: (∫⁺ *x*. *ennreal* (*pmf p x*) ∂*count-space UNIV*) = *1*
⟨*proof*⟩
end

lemma *pmf-of-mapping-mapping-of-pmf* [*code abstype*]:
pmf-of-mapping (*mapping-of-pmf p*) = *p*
⟨*proof*⟩

lemma *mapping-of-pmfI*:
assumes ⋀*x*. *x* ∈ *Mapping.keys m* ⟹ *Mapping.lookup m x* = *Some* (*pmf p x*)
assumes *Mapping.keys m* = *set-pmf p*
shows *mapping-of-pmf p* = *m*
⟨*proof*⟩

lemma *mapping-of-pmfI'*:
assumes ⋀*x*. *x* ∈ *Mapping.keys m* ⟹ *Mapping.lookup-default 0 m x* = *pmf p x*

assumes *Mapping.keys m* = *set-pmf p*
shows *mapping-of-pmf p* = *m*
⟨*proof*⟩

lemma *return-pmf-code* [*code abstract*]:
mapping-of-pmf (*return-pmf x*) = *Mapping.update x 1 Mapping.empty*
⟨*proof*⟩

lemma *pmf-of-set-code-aux*:
assumes $A \neq \{\}$ *set xs* = *A distinct xs*
shows *mapping-of-pmf* (*pmf-of-set A*) = *Mapping.tabulate xs* (λ-. *1 / real* (*length xs*))
⟨*proof*⟩

definition *pmf-of-set-impl* **where**
pmf-of-set-impl A = *mapping-of-pmf* (*pmf-of-set A*)

lemma *pmf-of-set-impl-code-alt*:
assumes $A \neq \{\}$ *finite A*
shows *pmf-of-set-impl A* =
(*let p* = *1 / real* (*card A*)
in Finite-Set.fold (λ*x. Mapping.update x p*) *Mapping.empty A*)
⟨*proof*⟩

lemma *pmf-of-set-impl-code* [*code*]:
pmf-of-set-impl (*set xs*) =
(*if xs* = [] *then*
Code.abort (*STR ''pmf-of-set of empty set''*) (λ-. *mapping-of-pmf* (*pmf-of-set* (*set xs*)))
else let xs′ = *remdups xs*; *p* = *1 / real* (*length xs′*) *in*
Mapping.tabulate xs′ (λ-. *p*))
⟨*proof*⟩

lemma *pmf-of-set-code* [*code abstract*]:
mapping-of-pmf (*pmf-of-set A*) = *pmf-of-set-impl A*
⟨*proof*⟩

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* (λ*x. count A x / real* (*size A*))
⟨*proof*⟩

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 (λ*x. Mapping.map-default x 0* ((+) *p*)) *Mapping.empty A*)
⟨*proof*⟩

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' (λ*x*. *real* (*count* (*mset xs*) *x*) ∗ *p*))
⟨*proof*⟩

lemma *pmf-of-multiset-code* [*code abstract*]:
mapping-of-pmf (*pmf-of-multiset A*) = *pmf-of-multiset-impl A*
⟨*proof*⟩

lemma *bernoulli-pmf-code* [*code abstract*]:
mapping-of-pmf (*bernoulli-pmf p*) =
(*if p* ≤ *0 then Mapping.update False 1 Mapping.empty*
else if p ≥ *1 then Mapping.update True 1 Mapping.empty*
else Mapping.update False (*1* − *p*) (*Mapping.update True p Mapping.empty*))
⟨*proof*⟩

lemma *pmf-code* [*code*]: *pmf p x* = *Mapping.lookup-default 0* (*mapping-of-pmf p*) *x*
⟨*proof*⟩

lemma *set-pmf-code* [*code*]: *set-pmf p* = *Mapping.keys* (*mapping-of-pmf p*)
⟨*proof*⟩

lemma *keys-mapping-of-pmf* [*simp*]: *Mapping.keys* (*mapping-of-pmf p*) = *set-pmf p*
⟨*proof*⟩

definition *fold-combine-plus* **where**
fold-combine-plus = *comm-monoid-set.F* (*Mapping.combine* ((+) :: *real* ⇒ -)) *Mapping.empty*

context
begin

interpretation *fold-combine-plus*: *combine-mapping-abel-semigroup* (+) :: *real* ⇒ -
⟨*proof*⟩ **lemma** *lookup-default-fold-combine-plus*:
fixes *A* :: *'b set* **and** *f* :: *'b* ⇒ (*'a*, *real*) *mapping*
assumes *finite A*
shows *Mapping.lookup-default 0* (*fold-combine-plus f A*) *x* =
(∑ *y*∈*A*. *Mapping.lookup-default 0* (*f y*) *x*)
⟨*proof*⟩ **lemma** *keys-fold-combine-plus*:

finite A ⟹ Mapping.keys (fold-combine-plus f A) = (⋃x∈A. Mapping.keys (f x))
⟨*proof*⟩ **lemma** *fold-combine-plus-code* [*code*]:
fold-combine-plus g (set xs) = foldr (λx. Mapping.combine (+) (g x)) (remdups xs) Mapping.empty
⟨*proof*⟩ **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)*
⟨*proof*⟩ **lemma** *mapping-of-bind-pmf*:
assumes *finite (set-pmf p)*
shows *mapping-of-pmf (bind-pmf p f) =*
fold-combine-plus (λx. Mapping.map-values (λ-. (∗) (pmf p x))
(mapping-of-pmf (f x))) (set-pmf p)
⟨*proof*⟩

lift-definition *bind-pmf-aux :: 'a pmf ⇒ ('a ⇒ 'b pmf) ⇒ 'a set ⇒ ('b, real) mapping* **is**
λ(p :: 'a pmf) (f :: 'a ⇒ 'b pmf) (A::'a set) (x::'b).
if x ∈ (⋃y∈A. set-pmf (f y)) then
Some (measure-pmf.expectation p (λy. indicator A y ∗ pmf (f y) x))
else None ⟨*proof*⟩

lemma *keys-bind-pmf-aux* [*simp*]:
Mapping.keys (bind-pmf-aux p f A) = (⋃x∈A. set-pmf (f x))
⟨*proof*⟩

lemma *lookup-default-bind-pmf-aux*:
Mapping.lookup-default 0 (bind-pmf-aux p f A) x =
(if x ∈ (⋃y∈A. set-pmf (f y)) then
measure-pmf.expectation p (λy. indicator A y ∗ pmf (f y) x) else 0)
⟨*proof*⟩

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
⟨*proof*⟩

lemma *bind-pmf-aux-correct*:
mapping-of-pmf (bind-pmf p f) = bind-pmf-aux p f (set-pmf p)
⟨*proof*⟩

lemma *bind-pmf-aux-code-aux*:
assumes *finite A*
shows *bind-pmf-aux p f A =*
fold-combine-plus (λx. Mapping.map-values (λ-. (∗) (pmf p x))
(mapping-of-pmf (f x))) A (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *bind-pmf-aux-code* [*code*]:
bind-pmf-aux p f (set xs) =
fold-combine-plus (λx. Mapping.map-values (λ-. (∗) (pmf p x))
(mapping-of-pmf (f x))) (set xs)
⟨*proof*⟩

lemmas *bind-pmf-code* [*code abstract*] = *bind-pmf-aux-correct*

end

hide-const (**open**) *fold-combine-plus*

lift-definition *cond-pmf-impl* :: *'a pmf ⇒ 'a set ⇒ ('a, real) mapping option* **is**
λp A. if A ∩ set-pmf p = {} then None else
Some (λx. if x ∈ A ∩ set-pmf p then Some (pmf p x / measure-pmf.prob p A)
else None) ⟨*proof*⟩

lemma *cond-pmf-impl-code-alt*:
assumes *finite A*
shows *cond-pmf-impl p A = (*
let C = A ∩ set-pmf p;
prob = (∑ x∈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*⟩

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 (λ- y. y / prob)
(Mapping.filter (λk -. k ∈ C) (mapping-of-pmf p))))
⟨*proof*⟩

lemma *cond-pmf-code* [*code abstract*]:
mapping-of-pmf (cond-pmf p A) =
(case cond-pmf-impl p A of
None ⇒ Code.abort (STR ''cond-pmf with set of probability 0'')
(λ-. mapping-of-pmf (cond-pmf p A))
| Some m ⇒ m)
⟨*proof*⟩

lemma *binomial-pmf-code* [*code abstract*]:
mapping-of-pmf (*binomial-pmf n p*) = (
if p < *0* ∨ *p* > *1 then*
Code.abort (*STR ''binomial-pmf with invalid probability''*)
(λ-. *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..*<*Suc n*] (λ*k*. *real* (*n choose k*) ∗ *p* ^ *k* ∗ (*1* − *p*) ^
(*n* − *k*)))
⟨*proof*⟩

lemma *pred-pmf-code* [*code*]:
pred-pmf P p = (∀ *x*∈*set-pmf p*. *P x*)
⟨*proof*⟩

lemma *mapping-of-pmf-pmf-of-list*:
assumes ⋀*x*. *x* ∈ *snd* ' *set xs* ⟹ *x* > *0 sum-list* (*map snd xs*) = *1*
shows *mapping-of-pmf* (*pmf-of-list xs*) =
Mapping.tabulate (*remdups* (*map fst xs*))
(λ*x*. *sum-list* (*map snd* (*filter* (λ*z*. *fst z* = *x*) *xs*)))
⟨*proof*⟩

lemma *mapping-of-pmf-pmf-of-list'*:
assumes *pmf-of-list-wf xs*
defines *xs'* ≡ *filter* (λ*z*. *snd z* ≠ *0*) *xs*
shows *mapping-of-pmf* (*pmf-of-list xs*) =
Mapping.tabulate (*remdups* (*map fst xs'*))
(λ*x*. *sum-list* (*map snd* (*filter* (λ*z*. *fst z* = *x*) *xs'*))) (**is** - = *?rhs*)
⟨*proof*⟩

lemma *pmf-of-list-wf-code* [*code*]:
pmf-of-list-wf xs ⟷ *list-all* (λ*z*. *snd z* ≥ *0*) *xs* ∧ *sum-list* (*map snd xs*) = *1*
⟨*proof*⟩

lemma *pmf-of-list-code* [*code abstract*]:
mapping-of-pmf (*pmf-of-list xs*) = (
if pmf-of-list-wf xs then
let xs' = *filter* (λ*z*. *snd z* ≠ *0*) *xs*
in Mapping.tabulate (*remdups* (*map fst xs'*))
(λ*x*. *sum-list* (*map snd* (*filter* (λ*z*. *fst z* = *x*) *xs'*)))
else
Code.abort (*STR ''Invalid list for pmf-of-list''*) (λ-. *mapping-of-pmf* (*pmf-of-list*
xs)))
⟨*proof*⟩

lemma *mapping-of-pmf-eq-iff* [*simp*]:

mapping-of-pmf p = mapping-of-pmf q ⟷ p = (q :: 'a pmf)
⟨*proof*⟩

21.2 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 = lebesgue-integral (measure-pmf p) (f :: - ⇒ real)

definition *pmf-set-integral* **where**
pmf-set-integral p f A = lebesgue-integral (measure-pmf p) (λx. indicator A x ∗ f x :: real)

definition *pmf-prob* **where**
pmf-prob p A = measure-pmf.prob p A

lemma *pmf-prob-compl*: *pmf-prob p (−A) = 1 − pmf-prob p A*
⟨*proof*⟩

lemma *pmf-integral-pmf-set-integral* [*code*]:
pmf-integral p f = pmf-set-integral p f (set-pmf p)
⟨*proof*⟩

lemma *pmf-prob-pmf-set-integral*:
pmf-prob p A = pmf-set-integral p (λ-. 1) A
⟨*proof*⟩

lemma *pmf-set-integral-code-alt-finite*:
finite A ⟹ pmf-set-integral p f A = (∑ x∈A. pmf p x ∗ f x)
⟨*proof*⟩

lemma *pmf-set-integral-code* [*code*]:
pmf-set-integral p f (set xs) = (∑ x∈set xs. pmf p x ∗ f x)
⟨*proof*⟩

lemma *pmf-prob-code-alt-finite*:
finite A ⟹ pmf-prob p A = (∑ x∈A. pmf p x)
⟨*proof*⟩

lemma *pmf-prob-code* [*code*]:
pmf-prob p (set xs) = (∑ x∈set xs. pmf p x)
pmf-prob p (List.coset xs) = 1 − (∑ x∈set xs. pmf p x)
⟨*proof*⟩

lemma *pmf-prob-code-unfold* [*code-abbrev*]: *pmf-prob p = measure-pmf.prob p*
⟨*proof*⟩

lemma *pmf-integral-code-unfold* [*code-abbrev*]: *pmf-integral p = measure-pmf.expectation p*
⟨*proof*⟩

definition *pmf-of-alist xs = embed-pmf* (λ*x. case map-of xs x of Some p ⇒ p | None ⇒ 0*)

lemma *pmf-of-mapping-Mapping* [*code-post*]:
pmf-of-mapping (*Mapping xs*) = *pmf-of-alist xs*
⟨*proof*⟩

instantiation *pmf* :: (*equal*) *equal*
begin

definition *equal-pmf p q* = (*mapping-of-pmf p = mapping-of-pmf* (*q* :: *'a pmf*))

instance ⟨*proof*⟩
end

definition *single* :: *'a ⇒ 'a multiset* **where**
single s = {#*s*#}

instantiation *pmf* :: (*random*) *random*
begin

context
includes *state-combinator-syntax* **and** *term-syntax*
begin

definition
pmfify :: (*'b*::*typerep multiset* × (*unit ⇒ Code-Evaluation.term*)) ⇒
'b × (*unit ⇒ Code-Evaluation.term*) ⇒
'b pmf × (*unit ⇒ Code-Evaluation.term*) **where**
[*code-unfold*]: *pmfify A x* =
Code-Evaluation.valtermify pmf-of-multiset {·}
(*Code-Evaluation.valtermify* (+) {·} *A* {·}
(*Code-Evaluation.valtermify single* {·} *x*))

definition
Quickcheck-Random.random i =

Quickcheck-Random.random i $\circ\!\!\rightarrow$ (λ*A*.
Quickcheck-Random.random i $\circ\!\!\rightarrow$ (λ*x*. *Pair* (*pmfify A x*)))

instance ⟨*proof*⟩

end

end

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 (λ*A*.
Quickcheck-Exhaustive.full-exhaustive (λ*x*. *f* (*pmfify A x*)) *i*) *i*

instance ⟨*proof*⟩

end

end

22 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* (‹(‹*notation*=‹*infix fmap*››- ⇒$_F$ /-)› [*22*, *21*] *21*)

unbundle *fmap.lifting*

22.1 Domain and Application

lift-definition *domain*::('*i* ⇒$_F$ '*a*) ⇒ '*i set* **is** *dom* ⟨*proof*⟩

lemma *finite-domain*[*simp*, *intro*]: *finite* (*domain P*)
⟨*proof*⟩

lift-definition *proj* :: ('*i* ⇒$_F$ '*a*) ⇒ '*i* ⇒ '*a*
(‹(‹*indent=1 notation*=‹*mixfix proj*››'(-')$_F$)› [*0*] *1000*) **is**

$\lambda f\ x.$ *if* $x \in$ *dom* f *then the* $(f\ x)$ *else undefined* ⟨*proof*⟩

declare [[*coercion proj*]]

lemma *extensional-proj*[*simp*, *intro*]: $(P)_F \in$ *extensional* (*domain* P)
⟨*proof*⟩

lemma *proj-undefined*[*simp*, *intro*]: $i \notin$ *domain* $P \Longrightarrow P\ i =$ *undefined*
⟨*proof*⟩

lemma *finmap-eq-iff*: $P = Q \longleftrightarrow$ (*domain* $P =$ *domain* $Q \wedge (\forall\, i{\in}$*domain* $P.\ P\ i = Q\ i)$)
⟨*proof*⟩

22.2 Constructor of Finite Maps

lift-definition *finmap-of*::$'i$ *set* $\Rightarrow$ $('i \Rightarrow 'a) \Rightarrow ('i \Rightarrow_F 'a)$ **is**
$\lambda I\ f\ x.$ *if* $x \in I \wedge$ *finite* I *then Some* $(f\ x)$ *else None*
⟨*proof*⟩

lemma *proj-finmap-of*[*simp*]:
assumes *finite inds*
shows $(\mathit{finmap\text{-}of}\ \mathit{inds}\ f)_F = \mathit{restrict}\ f\ \mathit{inds}$
⟨*proof*⟩

lemma *domain-finmap-of*[*simp*]:
assumes *finite inds*
shows *domain* (*finmap-of inds* f) = *inds*
⟨*proof*⟩

lemma *finmap-of-eq-iff*[*simp*]:
assumes *finite* i *finite* j
shows *finmap-of* $i\ m =$ *finmap-of* $j\ n \longleftrightarrow i = j \wedge (\forall\, k{\in}i.\ m\ k{=}\ n\ k)$
⟨*proof*⟩

lemma *finmap-of-inj-on-extensional-finite*:
assumes *finite* K
assumes $S \subseteq$ *extensional* K
shows *inj-on* (*finmap-of* K) S
⟨*proof*⟩

22.3 Product set of Finite Maps

This is *Pi* for Finite Maps, most of this is copied

definition $Pi' :: 'i\ set \Rightarrow ('i \Rightarrow 'a\ set) \Rightarrow ('i \Rightarrow_F 'a)\ set$ **where**
$Pi'\ I\ A = \{\ P.\ \mathit{domain}\ P = I \wedge (\forall\, i.\ i \in I \longrightarrow (P)_F\ i \in A\ i)\ \}$

syntax
-Pi' :: [*pttrn*, $'a$ *set*, $'b$ *set*] => $('a$ => $'b)$ *set*

(‹(‹*indent=3 notation=‹binder* Π′››Π′′ -∈-./ -)› *10*)
syntax-consts
-Pi′ == *Pi′*
translations
Π′ *x*∈*A*. *B* == *CONST Pi′ A* (λ*x*. *B*)

22.3.1 Basic Properties of *Pi′*

lemma *Pi′-I*[*intro*!]: *domain f* = *A* ⟹ (⋀*x*. *x* ∈ *A* ⟹ *f x* ∈ *B x*) ⟹ *f* ∈ *Pi′ A B*
⟨*proof*⟩

lemma *Pi′-I′*[*simp*]: *domain f* = *A* ⟹ (⋀*x*. *x* ∈ *A* ⟶ *f x* ∈ *B x*) ⟹ *f* ∈ *Pi′ A B*
⟨*proof*⟩

lemma *Pi′-mem*: *f*∈ *Pi′ A B* ⟹ *x* ∈ *A* ⟹ *f x* ∈ *B x*
⟨*proof*⟩

lemma *Pi′-iff*: *f* ∈ *Pi′ I X* ⟷ *domain f* = *I* ∧ (∀ *i*∈*I*. *f i* ∈ *X i*)
⟨*proof*⟩

lemma *Pi′E* [*elim*]:
f ∈ *Pi′ A B* ⟹ (*f x* ∈ *B x* ⟹ *domain f* = *A* ⟹ *Q*) ⟹ (*x* ∉ *A* ⟹ *Q*) ⟹ *Q*
⟨*proof*⟩

lemma *in-Pi′-cong*:
domain f = *domain g* ⟹ (⋀ *w*. *w* ∈ *A* ⟹ *f w* = *g w*) ⟹ *f* ∈ *Pi′ A B* ⟷ *g* ∈ *Pi′ A B*
⟨*proof*⟩

lemma *Pi′-eq-empty*[*simp*]:
assumes *finite A* **shows** (*Pi′ A B*) = {} ⟷ (∃ *x*∈*A*. *B x* = {})
⟨*proof*⟩

lemma *Pi′-mono*: (⋀*x*. *x* ∈ *A* ⟹ *B x* ⊆ *C x*) ⟹ *Pi′ A B* ⊆ *Pi′ A C*
⟨*proof*⟩

lemma *Pi-Pi′*: *finite A* ⟹ (*Pi*$_E$ *A B*) = *proj* ‘ *Pi′ A B*
⟨*proof*⟩

22.4 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*: $(\bigwedge i.\ i \in I \Longrightarrow open\ (A\ i)) \Longrightarrow open\ (Pi'\ I\ A)$
⟨*proof*⟩

instance ⟨*proof*⟩

end

lemma *open-restricted-space*:
shows *open* $\{m.\ P\ (domain\ m)\}$
⟨*proof*⟩

lemma *closed-restricted-space*:
shows *closed* $\{m.\ P\ (domain\ m)\}$
⟨*proof*⟩

lemma *tendsto-proj*: $((\lambda x.\ x) \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ (x)_F\ i) \longrightarrow (a)_F\ i)\ F$
⟨*proof*⟩

lemma *continuous-proj*:
shows *continuous-on s* $(\lambda x.\ (x)_F\ i)$
⟨*proof*⟩

instance *fmap* :: (*type*, *first-countable-topology*) *first-countable-topology*
⟨*proof*⟩

22.5 Metric Space of Finite Maps

instantiation *fmap* :: (*type*, *metric-space*) *dist*
begin

definition *dist-fmap* **where**
dist P Q = *Max* (*range* (λ*i*. *dist* ((*P*)$_F$ *i*) ((*Q*)$_F$ *i*))) + (*if domain P* = *domain Q then 0 else 1*)

instance ⟨*proof*⟩
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
⟨*proof*⟩
end

declare *uniformity-Abort*[**where** '*a*=('*a* ⇒$_F$ '*b*::*metric-space*), *code*]

instantiation *fmap* :: (*type*, *metric-space*) *metric-space*
begin

lemma *finite-proj-image'*: $x \notin$ *domain* $P \Longrightarrow$ *finite* $((P)_F$ ' $S)$
⟨*proof*⟩

lemma *finite-proj-image*: *finite* $((P)_F$ ' $S)$
⟨*proof*⟩

lemma *finite-proj-diag*: *finite* $((\lambda i.\ d\ ((P)_F\ i)\ ((Q)_F\ i))$ ' $S)$
⟨*proof*⟩

lemma *dist-le-1-imp-domain-eq*:
shows *dist* $P\ Q < 1 \Longrightarrow$ *domain* $P =$ *domain* Q
⟨*proof*⟩

lemma *dist-proj*:
shows *dist* $((x)_F\ i)\ ((y)_F\ i) \leq$ *dist* $x\ y$
⟨*proof*⟩

lemma *dist-finmap-lessI*:
assumes *domain* $P =$ *domain* Q
assumes $0 < e$
assumes $\bigwedge i.\ i \in$ *domain* $P \Longrightarrow$ *dist* $(P\ i)\ (Q\ i) < e$
shows *dist* $P\ Q < e$
⟨*proof*⟩

instance
⟨*proof*⟩

end

22.6 Complete Space of Finite Maps

lemma *tendsto-finmap*:
fixes f::*nat* $\Rightarrow$ ('*i* $\Rightarrow_F$ ('*a*::*metric-space*))
assumes *ind-f*: $\bigwedge n.$ *domain* $(f\ n) =$ *domain* g
assumes *proj-g*: $\bigwedge i.\ i \in$ *domain* $g \Longrightarrow (\lambda n.\ (f\ n)\ i) \longrightarrow g\ i$
shows $f \longrightarrow g$
⟨*proof*⟩

instance *fmap* :: (*type*, *complete-space*) *complete-space*
⟨*proof*⟩

22.7 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* ∧ *topological-basis B*)

lemma *countable-basis-proj*: *countable basis-proj* **and** *basis-proj*: *topological-basis basis-proj*
⟨*proof*⟩

definition *basis-finmap*::(*'a* ⇒$_F$ *'b*) *set set*
where *basis-finmap* = {*Pi' I S*|*I S. finite I* ∧ (∀ *i* ∈ *I. S i* ∈ *basis-proj*)}

lemma *in-basis-finmapI*:
assumes *finite I* **assumes** ⋀*i. i* ∈ *I* ⟹ *S i* ∈ *basis-proj*
shows *Pi' I S* ∈ *basis-finmap*
⟨*proof*⟩

lemma *basis-finmap-eq*:
assumes *basis-proj* ≠ {}
shows *basis-finmap* = (λ*f. Pi'* (*domain f*) (λ*i. from-nat-into basis-proj* ((*f*)$_F$ *i*))) ‘
(*UNIV*::(*'a* ⇒$_F$ *nat*) *set*) (**is** - = *?f* ‘ -)
⟨*proof*⟩

lemma *basis-finmap-eq-empty*: *basis-proj* = {} ⟹ *basis-finmap* = {*Pi'* {} *undefined*}
⟨*proof*⟩

lemma *countable-basis-finmap*: *countable basis-finmap*
⟨*proof*⟩

lemma *finmap-topological-basis*:
topological-basis basis-finmap
⟨*proof*⟩

lemma *range-enum-basis-finmap-imp-open*:
assumes *x* ∈ *basis-finmap*
shows *open x*
⟨*proof*⟩

instance ⟨*proof*⟩

end

22.8 Polish Space of Finite Maps

instance *fmap* :: (*countable*, *polish-space*) *polish-space* ⟨*proof*⟩

22.9 Product Measurable Space of Finite Maps

definition *PiF I M* ≡

sigma $(\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))\ \{(\Pi'\ j{\in}J.\ X\ j)\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$

abbreviation
$Pi_F\ I\ M \equiv PiF\ I\ M$

syntax
-PiF :: *pttrn* ⇒ *'i set* ⇒ *'a measure* ⇒ (*'i* => *'a*) *measure*
(‹(‹*indent=3 notation=‹binder* Π_F*››*Π_F *-∈-./ -)›* *10*)
syntax-consts
-PiF == *PiF*
translations
$\Pi_F\ x{\in}I.\ M$ == *CONST PiF I* $(\%x.\ M)$

lemma *PiF-gen-subset*: $\{(\Pi'\ j{\in}J.\ X\ j)\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$
$\subseteq$
Pow $(\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$
⟨*proof*⟩

lemma *space-PiF*: *space* $(PiF\ I\ M) = (\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$
⟨*proof*⟩

lemma *sets-PiF*:
sets $(PiF\ I\ M) =$ *sigma-sets* $(\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$
$\{(\Pi'\ j{\in}J.\ X\ j)\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$
⟨*proof*⟩

lemma *sets-PiF-singleton*:
sets $(PiF\ \{I\}\ M) =$ *sigma-sets* $(\Pi'\ j{\in}I.\ space\ (M\ j))$
$\{(\Pi'\ j{\in}I.\ X\ j)\ |X.\ X \in (\Pi\ j{\in}I.\ sets\ (M\ j))\}$
⟨*proof*⟩

lemma *in-sets-PiFI*:
assumes $X = (Pi'\ J\ S)$ $J \in I$ $\bigwedge i.\ i{\in}J \Longrightarrow S\ i \in sets\ (M\ i)$
shows $X \in sets\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *product-in-sets-PiFI*:
assumes $J \in I$ $\bigwedge i.\ i{\in}J \Longrightarrow S\ i \in sets\ (M\ i)$
shows $(Pi'\ J\ S) \in sets\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *singleton-space-subset-in-sets*:
fixes J
assumes $J \in I$
assumes *finite J*
shows *space* $(PiF\ \{J\}\ M) \in sets\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *singleton-subspace-set-in-sets*:
assumes *A*: $A \in sets\ (PiF\ \{J\}\ M)$
assumes *finite J*
assumes $J \in I$
shows $A \in sets\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *finite-measurable-singletonI*:
assumes *finite I*
assumes $\bigwedge J.\ J \in I \Longrightarrow$ *finite J*
assumes *MN*: $\bigwedge J.\ J \in I \Longrightarrow A \in measurable\ (PiF\ \{J\}\ M)\ N$
shows $A \in measurable\ (PiF\ I\ M)\ N$
⟨*proof*⟩

lemma *countable-finite-comprehension*:
fixes *f* :: *'a*::*countable set* ⇒ -
assumes $\bigwedge s.\ P\ s \Longrightarrow$ *finite s*
assumes $\bigwedge s.\ P\ s \Longrightarrow f\ s \in sets\ M$
shows $\bigcup \{f\ s | s.\ P\ s\} \in sets\ M$
⟨*proof*⟩

lemma *space-subset-in-sets*:
fixes *J*::*'a*::*countable set set*
assumes $J \subseteq I$
assumes $\bigwedge j.\ j \in J \Longrightarrow$ *finite j*
shows $space\ (PiF\ J\ M) \in sets\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *subspace-set-in-sets*:
fixes *J*::*'a*::*countable set set*
assumes *A*: $A \in sets\ (PiF\ J\ M)$
assumes $J \subseteq I$
assumes $\bigwedge j.\ j \in J \Longrightarrow$ *finite j*
shows $A \in sets\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *countable-measurable-PiFI*:
fixes *I*::*'a*::*countable set set*
assumes *MN*: $\bigwedge J.\ J \in I \Longrightarrow$ *finite* $J \Longrightarrow A \in measurable\ (PiF\ \{J\}\ M)\ N$
shows $A \in measurable\ (PiF\ I\ M)\ N$
⟨*proof*⟩

lemma *measurable-PiF*:
assumes *f*: $\bigwedge x.\ x \in space\ N \Longrightarrow domain\ (f\ x) \in I \wedge (\forall\, i \in domain\ (f\ x).\ (f\ x)\ i \in space\ (M\ i))$
assumes *S*: $\bigwedge J\ S.\ J \in I \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow S\ i \in sets\ (M\ i)) \Longrightarrow$
$f\ -`\ (Pi'\ J\ S) \cap space\ N \in sets\ N$
shows $f \in measurable\ N\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *restrict-sets-measurable*:
assumes A: $A \in sets\ (PiF\ I\ M)$ **and** $J \subseteq I$
shows $A \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$
⟨*proof*⟩

lemma *measurable-finmap-of*:
assumes f: $\bigwedge i.\ (\exists\, x \in space\ N.\ i \in J\ x) \Longrightarrow (\lambda x.\ f\ x\ i) \in measurable\ N\ (M\ i)$
assumes J: $\bigwedge x.\ x \in space\ N \Longrightarrow J\ x \in I\ \bigwedge x.\ x \in space\ N \Longrightarrow finite\ (J\ x)$
assumes JN: $\bigwedge S.\ \{x.\ J\ x = S\} \cap space\ N \in sets\ N$
shows $(\lambda x.\ finmap\text{-}of\ (J\ x)\ (f\ x)) \in measurable\ N\ (PiF\ I\ M)$
⟨*proof*⟩

lemma *measurable-PiM-finmap-of*:
assumes *finite* J
shows *finmap-of* $J \in measurable\ (Pi_M\ J\ M)\ (PiF\ \{J\}\ M)$
⟨*proof*⟩

lemma *proj-measurable-singleton*:
assumes $A \in sets\ (M\ i)$
shows $(\lambda x.\ (x)_F\ i) -` A \cap space\ (PiF\ \{I\}\ M) \in sets\ (PiF\ \{I\}\ M)$
⟨*proof*⟩

lemma *measurable-proj-singleton*:
assumes $i \in I$
shows $(\lambda x.\ (x)_F\ i) \in measurable\ (PiF\ \{I\}\ M)\ (M\ i)$
⟨*proof*⟩

lemma *measurable-proj-countable*:
fixes I::$'a$::*countable set set*
assumes $y \in space\ (M\ i)$
shows $(\lambda x.\ if\ i \in domain\ x\ then\ (x)_F\ i\ else\ y) \in measurable\ (PiF\ I\ M)\ (M\ i)$
⟨*proof*⟩

lemma *measurable-restrict-proj*:
assumes $J \in II\ finite\ J$
shows *finmap-of* $J \in measurable\ (PiM\ J\ M)\ (PiF\ II\ M)$
⟨*proof*⟩

lemma *measurable-proj-PiM*:
fixes $J\ K$ ::$'a$::*countable set* **and** I::$'a$ *set set*
assumes *finite* $J\ J \in I$
assumes $x \in space\ (PiM\ J\ M)$
shows $proj \in measurable\ (PiF\ \{J\}\ M)\ (PiM\ J\ M)$
⟨*proof*⟩

lemma *space-PiF-singleton-eq-product*:
assumes *finite* I
shows $space\ (PiF\ \{I\}\ M) = (\Pi'\ i \in I.\ space\ (M\ i))$

⟨*proof*⟩

adapted from *sets* (Pi_M *?I ?M*) = *sigma-sets* (Π_E *i*∈*?I*. *space* (*?M i*)) {{*f* ∈ Π_E *i*∈*?I*. *space* (*?M i*). *f i* ∈ *A*} |*i A*. *i* ∈ *?I* ∧ *A* ∈ *sets* (*?M i*)}

lemma *sets-PiF-single*:
assumes *finite I I* ≠ {}
shows *sets* (*PiF* {*I*} *M*) =
sigma-sets (Π' *i*∈*I*. *space* (*M i*))
{{*f*∈Π' *i*∈*I*. *space* (*M i*). *f i* ∈ *A*} | *i A*. *i* ∈ *I* ∧ *A* ∈ *sets* (*M i*)}
(**is** - = *sigma-sets ?*Ω *?R*)
⟨*proof*⟩

adapted from (⋀*i*. *i* ∈ *?I* ⟹ *?A i* = *?B i*) ⟹ Pi_E *?I ?A* = Pi_E *?I ?B*

lemma *Pi'-cong*:
assumes *finite I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* = *g i*
shows *Pi' I f* = *Pi' I g*
⟨*proof*⟩

adapted from ⟦*finite ?I*; ⋀*i n m*. ⟦*i* ∈ *?I*; *n* ≤ *m*⟧ ⟹ *?A n i* ⊆ *?A m i*⟧ ⟹ (⋃$_n$ *Pi ?I* (*?A n*)) = (Π *i*∈*?I*. ⋃$_n$ *?A n i*)

lemma *Pi'-UN*:
fixes *A* :: *nat* ⇒ *'i* ⇒ *'a set*
assumes *finite I*
assumes *mono*: ⋀*i n m*. *i* ∈ *I* ⟹ *n* ≤ *m* ⟹ *A n i* ⊆ *A m i*
shows (⋃*n*. *Pi' I* (*A n*)) = *Pi' I* (λ*i*. ⋃*n*. *A n i*)
⟨*proof*⟩

adapted from ⟦⋀*i*. *i* ∈ *?I* ⟹ ∃ *S*⊆*?E i*. *countable S* ∧ *?*Ω *i* = ⋃ *S*; ⋀*i*. *i* ∈ *?I* ⟹ *?E i* ⊆ *Pow* (*?*Ω *i*); ⋀*j*. *j* ∈ *?J* ⟹ *finite j*; ⋃ *?J* = *?I*⟧ ⟹ *sets* (Pi_M *?I* (λ*i*. *sigma* (*?*Ω *i*) (*?E i*))) = *sets* (*sigma* (Pi_E *?I ?*Ω) {{*f* ∈ Pi_E *?I ?*Ω. ∀ *i*∈*j*. *f i* ∈ *A i*} |*A j*. *j* ∈ *?J* ∧ *A* ∈ *Pi j ?E*})

lemma *sigma-fprod-algebra-sigma-eq*:
fixes *E* :: *'i* ⇒ *'a set set* **and** *S* :: *'i* ⇒ *nat* ⇒ *'a set*
assumes [*simp*]: *finite I I* ≠ {}
and *S-union*: ⋀*i*. *i* ∈ *I* ⟹ (⋃*j*. *S i j*) = *space* (*M i*)
and *S-in-E*: ⋀*i*. *i* ∈ *I* ⟹ *range* (*S i*) ⊆ *E i*
assumes *E-closed*: ⋀*i*. *i* ∈ *I* ⟹ *E i* ⊆ *Pow* (*space* (*M i*))
and *E-generates*: ⋀*i*. *i* ∈ *I* ⟹ *sets* (*M i*) = *sigma-sets* (*space* (*M i*)) (*E i*)
defines *P* == { *Pi' I F* | *F*. ∀ *i*∈*I*. *F i* ∈ *E i* }
shows *sets* (*PiF* {*I*} *M*) = *sigma-sets* (*space* (*PiF* {*I*} *M*)) *P*
⟨*proof*⟩

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*⟩

lemma *finmap-UNIV*[*simp*]: (⋃ *J*∈*Collect finite*. Π' *j*∈*J*. *UNIV*) = *UNIV* ⟨*proof*⟩

lemma *borel-eq-PiF-borel*:
shows (*borel* :: ('*i*::*countable* ⇒$_F$ '*a*::*polish-space*) *measure*) =
PiF (*Collect finite*) (λ-. *borel* :: '*a measure*)
⟨*proof*⟩

22.10 Isomorphism between Functions and Finite Maps

lemma *measurable-finmap-compose*:
shows (λ*m*. *compose* *J* *m* *f*) ∈ *measurable* (*PiM* (*f* ' *J*) (λ-. *M*)) (*PiM* *J* (λ-. *M*))
⟨*proof*⟩

lemma *measurable-compose-inv*:
assumes *inj*: ⋀*j*. *j* ∈ *J* ⟹ *f'* (*f* *j*) = *j*
shows (λ*m*. *compose* (*f* ' *J*) *m* *f'*) ∈ *measurable* (*PiM* *J* (λ-. *M*)) (*PiM* (*f* ' *J*) (λ-. *M*))
⟨*proof*⟩

locale *function-to-finmap* =
fixes *J*::'*a set* **and** *f* :: '*a* ⇒ '*b*::*countable* **and** *f'*
assumes [*simp*]: *finite* *J*
assumes *inv*: *i* ∈ *J* ⟹ *f'* (*f* *i*) = *i*
begin

to measure finmaps

definition *fm* = (*finmap-of* (*f* ' *J*)) *o* (λ*g*. *compose* (*f* ' *J*) *g* *f'*)

lemma *domain-fm*[*simp*]: *domain* (*fm* *x*) = *f* ' *J*
⟨*proof*⟩

lemma *fm-restrict*[*simp*]: *fm* (*restrict* *y* *J*) = *fm* *y*
⟨*proof*⟩

lemma *fm-product*:
assumes ⋀*i*. *space* (*M* *i*) = *UNIV*
shows *fm* −' *Pi'* (*f* ' *J*) *S* ∩ *space* (*Pi*$_M$ *J* *M*) = (Π$_E$ *j* ∈ *J*. *S* (*f* *j*))
⟨*proof*⟩

lemma *fm-measurable*:
assumes *f* ' *J* ∈ *N*
shows *fm* ∈ *measurable* (*Pi*$_M$ *J* (λ-. *M*)) (*Pi*$_F$ *N* (λ-. *M*))
⟨*proof*⟩

lemma *proj-fm*:
assumes $x \in J$
shows *fm m* (*f x*) = *m x*
⟨*proof*⟩

lemma *inj-on-compose-f'*: *inj-on* (λ*g*. *compose* (*f* ‘ *J*) *g f'*) (*extensional J*)
⟨*proof*⟩

lemma *inj-on-fm*:
assumes ⋀*i*. *space* (*M i*) = *UNIV*
shows *inj-on fm* (*space* (Pi_M *J M*))
⟨*proof*⟩

to measure functions

definition *mf* = (λ*g*. *compose J g f*) *o proj*

lemma *mf-fm*:
assumes *x* ∈ *space* (Pi_M *J* (λ-. *M*))
shows *mf* (*fm x*) = *x*
⟨*proof*⟩

lemma *mf-measurable*:
assumes *space M* = *UNIV*
shows *mf* ∈ *measurable* (*PiF* {*f* ‘ *J*} (λ-. *M*)) (*PiM J* (λ-. *M*))
⟨*proof*⟩

lemma *fm-image-measurable*:
assumes *space M* = *UNIV*
assumes *X* ∈ *sets* (Pi_M *J* (λ-. *M*))
shows *fm* ‘ *X* ∈ *sets* (*PiF* {*f* ‘ *J*} (λ-. *M*))
⟨*proof*⟩

lemma *fm-image-measurable-finite*:
assumes *space M* = *UNIV*
assumes *X* ∈ *sets* (Pi_M *J* (λ-. *M*::'*c measure*))
shows *fm* ‘ *X* ∈ *sets* (*PiF* (*Collect finite*) (λ-. *M*::'*c measure*))
⟨*proof*⟩

measure on finmaps

definition *mapmeasure M N* = *distr M* (*PiF* (*Collect finite*) *N*) (*fm*)

lemma *sets-mapmeasure*[*simp*]: *sets* (*mapmeasure M N*) = *sets* (*PiF* (*Collect finite*) *N*)
⟨*proof*⟩

lemma *space-mapmeasure*[*simp*]: *space* (*mapmeasure M N*) = *space* (*PiF* (*Collect finite*) *N*)
⟨*proof*⟩

lemma *mapmeasure-PiF*:
assumes *s1*: *space M* = *space* (*Pi*$_M$ *J* (λ-. *N*))
assumes *s2*: *sets M* = *sets* (*Pi*$_M$ *J* (λ-. *N*))
assumes *space N* = *UNIV*
assumes *X* ∈ *sets* (*PiF* (*Collect finite*) (λ-. *N*))
shows *emeasure* (*mapmeasure M* (λ-. *N*)) *X* = *emeasure M* ((*fm* −‘ *X* ∩ *extensional J*))
⟨*proof*⟩

lemma *mapmeasure-PiM*:
fixes *N*::′*c measure*
assumes *s1*: *space M* = *space* (*Pi*$_M$ *J* (λ-. *N*))
assumes *s2*: *sets M* = (*Pi*$_M$ *J* (λ-. *N*))
assumes *N*: *space N* = *UNIV*
assumes *X*: *X* ∈ *sets M*
shows *emeasure M X* = *emeasure* (*mapmeasure M* (λ-. *N*)) (*fm* ‘ *X*)
⟨*proof*⟩

end

end

23 Projective Limit

theory *Projective-Limit*
imports
Fin-Map
Infinite-Product-Measure
HOL−Library.Diagonal-Subsequence
begin

23.1 Sequences of Finite Maps in Compact Sets

locale *finmap-seqs-into-compact* =
fixes *K*::*nat* ⇒ (*nat* ⇒$_F$ ′*a*::*metric-space*) *set* **and** *f*::*nat* ⇒ (*nat* ⇒$_F$ ′*a*) **and** *M*
assumes *compact*: ⋀*n*. *compact* (*K n*)
assumes *f-in-K*: ⋀*n*. *K n* ≠ {}
assumes *domain-K*: ⋀*n*. *k* ∈ *K n* ⟹ *domain k* = *domain* (*f n*)
assumes *proj-in-K*:
⋀*t n m*. *m* ≥ *n* ⟹ *t* ∈ *domain* (*f n*) ⟹ (*f m*)$_F$ *t* ∈ (λ*k*. (*k*)$_F$ *t*) ‘ *K n*
begin

lemma *proj-in-K′*: (∃ *n*. ∀ *m* ≥ *n*. (*f m*)$_F$ *t* ∈ (λ*k*. (*k*)$_F$ *t*) ‘ *K n*)
⟨*proof*⟩

lemma *proj-in-KE*:
obtains *n* **where** ⋀*m*. *m* ≥ *n* ⟹ (*f m*)$_F$ *t* ∈ (λ*k*. (*k*)$_F$ *t*) ‘ *K n*
⟨*proof*⟩

lemma *compact-projset*:
shows *compact* $((\lambda k.\ (k)_F\ i)\ `\ K\ n)$
⟨*proof*⟩

end

lemma *compactE'*:
fixes $S :: 'a :: metric\text{-}space\ set$
assumes *compact* $S\ \forall\, n{\geq}m.\ f\ n \in S$
obtains $l\ r$ **where** $l \in S$ *strict-mono* $(r{::}nat{\Rightarrow}nat)$ $((f \circ r) \longrightarrow l)$ *sequentially*
⟨*proof*⟩

sublocale *finmap-seqs-into-compact* $\subseteq$ *subseqs* $\lambda n\ s.\ (\exists\, l.\ (\lambda i.\ ((f\ o\ s)\ i)_F\ n) \longrightarrow l)$
⟨*proof*⟩

lemma (**in** *finmap-seqs-into-compact*) *diagonal-tendsto*: $\exists\, l.\ (\lambda i.\ (f\ (diagseq\ i))_F\ n) \longrightarrow l$
⟨*proof*⟩

23.2 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* $J\ X \in sets\ (\Pi_M\ i{\in}J.\ borel)$
shows *lim* $(emb\ I\ J\ X) = P\ J\ X$
⟨*proof*⟩

lemma *measure-lim-emb*:
$J \subseteq I \Longrightarrow finite\ J \Longrightarrow X \in sets\ (\Pi_M\ i{\in}J.\ borel) \Longrightarrow measure\ lim\ (emb\ I\ J\ X) = measure\ (P\ J)\ X$
⟨*proof*⟩

end

hide-const (**open**) *PiF*
hide-const (**open**) Pi_F
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*⟩

locale *polish-product-prob-space* =
  product-prob-space λ-. *borel*::('*a*::*polish-space*) *measure* *I* **for** *I*::'*i set*

sublocale *polish-product-prob-space* ⊆ *P*: *polish-projective* *I* λ*J*. *PiM* *J* (λ-. *borel*::('*a*) *measure*)
  ⟨*proof*⟩

lemma (**in** *polish-product-prob-space*) *limP-eq-PiM*: *lim* = *PiM* *I* (λ-. *borel*)
  ⟨*proof*⟩

end

24 Random Permutations

theory *Random-Permutations*
imports
  HOL−Combinatorics.Multiset-Permutations
  Probability-Mass-Function
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*⟩

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* ⤜ (λ*a*. *fold-random-permutation* *f* (*f* *a* *x*) (*A* − {*a*}))
  ⟨*proof*⟩
termination ⟨*proof*⟩

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 *fold-random-permutation f x A =*
    map-pmf (*foldl* (λ*x y*. *f y x*) *x*) (*pmf-of-set* (*permutations-of-set A*))
⟨*proof*⟩

lemma *fold-random-permutation-foldr*:
  assumes *finite A*
  shows *fold-random-permutation f x A =*
    map-pmf (λ*xs*. *foldr f xs x*) (*pmf-of-set* (*permutations-of-set A*))
⟨*proof*⟩

lemma *fold-random-permutation-fold*:
  assumes *finite A*
  shows *fold-random-permutation f x A =*
    map-pmf (λ*xs*. *fold f xs x*) (*pmf-of-set* (*permutations-of-set A*))
  ⟨*proof*⟩

lemma *fold-random-permutation-code* [*code*]:
  fold-random-permutation f x (*set xs*) =
    map-pmf (*foldl* (λ*x y*. *f y x*) *x*) (*pmf-of-set* (*permutations-of-set* (*set xs*)))
  ⟨*proof*⟩

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* ⇒ '*b*) ⇒ ('*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 ⤜ (λ*a*. *fold-bind-random-permutation f g* (*f a x*) (*A* − {*a*}))
  ⟨*proof*⟩
termination ⟨*proof*⟩

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*⟩

We can now derive the following nice monadic representations of the com-

bined fold-and-bind:

lemma *fold-bind-random-permutation-foldl*:
assumes *finite A*
shows *fold-bind-random-permutation f g x A* =
do {*xs* ← *pmf-of-set* (*permutations-of-set A*); *g* (*foldl* (λ*x y*. *f y x*) *x xs*)}
⟨*proof*⟩

lemma *fold-bind-random-permutation-foldr*:
assumes *finite A*
shows *fold-bind-random-permutation f g x A* =
do {*xs* ← *pmf-of-set* (*permutations-of-set A*); *g* (*foldr f xs x*)}
⟨*proof*⟩

lemma *fold-bind-random-permutation-fold*:
assumes *finite A*
shows *fold-bind-random-permutation f g x A* =
do {*xs* ← *pmf-of-set* (*permutations-of-set A*); *g* (*fold f xs x*)}
⟨*proof*⟩

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 *map-pmf* (*partition P*) (*pmf-of-set* (*permutations-of-set A*)) =
pair-pmf (*pmf-of-set* (*permutations-of-set* {*x*∈*A*. *P x*}))
(*pmf-of-set* (*permutations-of-set* {*x*∈*A*. ¬*P x*})) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

25 Discrete subprobability distribution

theory *SPMF* **imports**
Probability-Mass-Function
HOL−Library.Complete-Partial-Order2
HOL−Library.Rewrite
begin

25.1 Auxiliary material

lemma *cSUP-singleton* [*simp*]: (*SUP x*∈{*x*}. *f x* :: - :: *conditionally-complete-lattice*)
= *f x*
⟨*proof*⟩

25.1.1 More about extended reals

lemma [*simp*]:
shows *ennreal-max-0*: *ennreal* (*max 0 x*) = *ennreal x*

and *ennreal-max-0'*: *ennreal* (*max x 0*) = *ennreal x*
⟨*proof*⟩

lemma *e2ennreal-0* [*simp*]: *e2ennreal 0* = *0*
⟨*proof*⟩

lemma *enn2real-bot* [*simp*]: *enn2real* ⊥ = *0*
⟨*proof*⟩

lemma *continuous-at-ennreal*[*continuous-intros*]: *continuous F f* ⟹ *continuous F* (λ*x*. *ennreal* (*f x*))
⟨*proof*⟩

lemma *ennreal-Sup*:
assumes ∗: (*SUP a*∈*A*. *ennreal a*) ≠ ⊤
and *A* ≠ {}
shows *ennreal* (*Sup A*) = (*SUP a*∈*A*. *ennreal a*)
⟨*proof*⟩

lemma *ennreal-SUP*:
⟦ (*SUP a*∈*A*. *ennreal* (*f a*)) ≠ ⊤; *A* ≠ {} ⟧ ⟹ *ennreal* (*SUP a*∈*A*. *f a*) = (*SUP a*∈*A*. *ennreal* (*f a*))
⟨*proof*⟩

lemma *ennreal-lt-0*: *x* < *0* ⟹ *ennreal x* = *0*
⟨*proof*⟩

25.1.2 More about *'a option*

lemma *None-in-map-option-image* [*simp*]: *None* ∈ *map-option f* ' *A* ⟷ *None* ∈ *A*
⟨*proof*⟩

lemma *Some-in-map-option-image* [*simp*]: *Some x* ∈ *map-option f* ' *A* ⟷ (∃ *y*. *x* = *f y* ∧ *Some y* ∈ *A*)
⟨*proof*⟩

lemma *case-option-collapse*: *case-option x* (λ-. *x*) = (λ-. *x*)
⟨*proof*⟩

lemma *case-option-id*: *case-option None Some* = *id*
⟨*proof*⟩

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 (=) *None y*
ord-option (=) (*Some x*) *y*

lemma *ord-option-reflI*: (⋀*y*. *y* ∈ *set-option x* ⟹ *ord y y*) ⟹ *ord-option ord x x*
⟨*proof*⟩

lemma *reflp-ord-option*: *reflp ord* ⟹ *reflp* (*ord-option ord*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *transp-ord-option*: *transp ord* ⟹ *transp* (*ord-option ord*)
⟨*proof*⟩

lemma *antisymp-ord-option*: *antisymp ord* ⟹ *antisymp* (*ord-option ord*)
⟨*proof*⟩

lemma *ord-option-chainD*:
Complete-Partial-Order.chain (*ord-option ord*) *Y*
⟹ *Complete-Partial-Order.chain ord* {*x*. *Some x* ∈ *Y*}
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *lub-option-upper*:
assumes *Complete-Partial-Order.chain* (*ord-option ord*) *Y x* ∈ *Y*
and *lub-upper*: ⋀*Y x*. ⟦ *Complete-Partial-Order.chain ord Y*; *x* ∈ *Y* ⟧ ⟹ *ord x* (*lub Y*)
shows *ord-option ord x* (*lub-option lub Y*)
⟨*proof*⟩

lemma *lub-option-least*:
assumes *Y*: *Complete-Partial-Order.chain* (*ord-option ord*) *Y*
and *upper*: $\bigwedge x.\ x \in Y \Longrightarrow$ *ord-option ord x y*
assumes *lub-least*: $\bigwedge Y\ y.\ [\![$ *Complete-Partial-Order.chain ord Y*; $\bigwedge x.\ x \in Y \Longrightarrow$ *ord x y* $]\!] \Longrightarrow$ *ord* (*lub Y*) *y*
shows *ord-option ord* (*lub-option lub Y*) *y*
⟨*proof*⟩

lemma *lub-map-option*: *lub-option lub* (*map-option f ' Y*) = *lub-option* (*lub* ∘ (') *f*) *Y*
⟨*proof*⟩

lemma *ord-option-mono*: [[*ord-option A x y*; $\bigwedge x\ y.$ *A x y* $\Longrightarrow$ *B x y*]] $\Longrightarrow$ *ord-option B x y*
⟨*proof*⟩

lemma *ord-option-mono'* [*mono*]:
($\bigwedge x\ y.$ *A x y* $\longrightarrow$ *B x y*) $\Longrightarrow$ *ord-option A x y* $\longrightarrow$ *ord-option B x y*
⟨*proof*⟩

lemma *ord-option-compp*: *ord-option* (*A OO B*) = *ord-option A OO ord-option B*
⟨*proof*⟩

lemma *ord-option-inf*: *inf* (*ord-option A*) (*ord-option B*) = *ord-option* (*inf A B*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *ord-option-map2*: *ord-option ord x* (*map-option f y*) = *ord-option* (λ*x y*. *ord x* (*f y*)) *x y*
⟨*proof*⟩

lemma *ord-option-map1*: *ord-option ord* (*map-option f x*) *y* = *ord-option* (λ*x y*. *ord* (*f x*) *y*) *x y*
⟨*proof*⟩

lemma *option-ord-Some1-iff*: *option-ord* (*Some x*) *y* $\longleftrightarrow$ *y* = *Some x*
⟨*proof*⟩

25.1.3 A relator for sets that treats sets like predicates

context includes *lifting-syntax*
begin

definition *rel-pred* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ '*a set* ⇒ '*b set* ⇒ *bool*
where *rel-pred R A B* = (*R* ===> (=)) (λ*x*. *x* ∈ *A*) (λ*y*. *y* ∈ *B*)

lemma *rel-predI*: (*R* ===> (=)) (λ*x*. *x* ∈ *A*) (λ*y*. *y* ∈ *B*) $\Longrightarrow$ *rel-pred R A B*
⟨*proof*⟩

lemma *rel-predD*: ⟦ *rel-pred R A B*; *R x y* ⟧ ⟹ $x \in A \longleftrightarrow y \in B$
⟨*proof*⟩

lemma *Collect-parametric*: ((*A* ===> (=)) ===> *rel-pred A*) *Collect Collect*
— Declare this rule as *transfer-rule* only locally because it blows up the search space for *transfer* (in combination with *Collect-transfer*)
⟨*proof*⟩

end

25.1.4 Monotonicity rules

lemma *monotone-gfp-eadd1*: *monotone* (≥) (≥) (λ*x*. *x* + *y* :: *enat*)
⟨*proof*⟩

lemma *monotone-gfp-eadd2*: *monotone* (≥) (≥) (λ*y*. *x* + *y* :: *enat*)
⟨*proof*⟩

lemma *mono2mono-gfp-eadd*[*THEN gfp.mono2mono2*, *cont-intro*, *simp*]:
shows *monotone-eadd*: *monotone* (*rel-prod* (≥) (≥)) (≥) (λ(*x*, *y*). *x* + *y* :: *enat*)
⟨*proof*⟩

lemma *eadd-gfp-partial-function-mono* [*partial-function-mono*]:
⟦ *monotone* (*fun-ord* (≥)) (≥) *f*; *monotone* (*fun-ord* (≥)) (≥) *g* ⟧
⟹ *monotone* (*fun-ord* (≥)) (≥) (λ*x*. *f x* + *g x* :: *enat*)
⟨*proof*⟩

lemma *mono2mono-ereal*[*THEN lfp.mono2mono*]:
shows *monotone-ereal*: *monotone* (≤) (≤) *ereal*
⟨*proof*⟩

lemma *mono2mono-ennreal*[*THEN lfp.mono2mono*]:
shows *monotone-ennreal*: *monotone* (≤) (≤) *ennreal*
⟨*proof*⟩

25.1.5 Bijections

lemma *bi-unique-rel-set-bij-betw*:
assumes *unique*: *bi-unique R*
and *rel*: *rel-set R A B*
shows ∃*f*. *bij-betw f A B* ∧ (∀ *x*∈*A*. *R x* (*f x*))
⟨*proof*⟩

lemma *bij-betw-rel-setD*: *bij-betw f A B* ⟹ *rel-set* (λ*x y*. *y* = *f x*) *A B*
⟨*proof*⟩

25.2 Subprobability mass function

type-synonym *'a spmf* = *'a option pmf*
translations (*type*) *'a spmf* ↽ (*type*) *'a option pmf*

definition *measure-spmf* :: *'a spmf* ⇒ *'a measure*
where *measure-spmf p* = *distr* (*restrict-space* (*measure-pmf p*) (*range Some*)) (*count-space UNIV*) *the*

abbreviation *spmf* :: *'a spmf* ⇒ *'a* ⇒ *real*
where *spmf p x* ≡ *pmf p* (*Some x*)

lemma *space-measure-spmf*: *space* (*measure-spmf p*) = *UNIV*
⟨*proof*⟩

lemma *sets-measure-spmf* [*simp*, *measurable-cong*]: *sets* (*measure-spmf p*) = *sets* (*count-space UNIV*)
⟨*proof*⟩

lemma *measure-spmf-not-bot* [*simp*]: *measure-spmf p* ≠ ⊥
⟨*proof*⟩

lemma *measurable-the-measure-pmf-Some* [*measurable*, *simp*]:
the ∈ *measurable* (*restrict-space* (*measure-pmf p*) (*range Some*)) (*count-space UNIV*)
⟨*proof*⟩

lemma *measurable-spmf-measure1*[*simp*]: *measurable* (*measure-spmf M*) *N* = *UNIV* → *space N*
⟨*proof*⟩

lemma *measurable-spmf-measure2*[*simp*]: *measurable N* (*measure-spmf M*) = *measurable N* (*count-space UNIV*)
⟨*proof*⟩

lemma *subprob-space-measure-spmf* [*simp*, *intro*!]: *subprob-space* (*measure-spmf p*)
⟨*proof*⟩

interpretation *measure-spmf*: *subprob-space measure-spmf p* **for** *p*
⟨*proof*⟩

lemma *finite-measure-spmf* [*simp*]: *finite-measure* (*measure-spmf p*)
⟨*proof*⟩

lemma *spmf-conv-measure-spmf*: *spmf p x* = *measure* (*measure-spmf p*) {*x*}
⟨*proof*⟩

lemma *emeasure-measure-spmf-conv-measure-pmf*:
emeasure (*measure-spmf p*) *A* = *emeasure* (*measure-pmf p*) (*Some* ‘ *A*)
⟨*proof*⟩

lemma *measure-measure-spmf-conv-measure-pmf*:
measure (*measure-spmf p*) *A* = *measure* (*measure-pmf p*) (*Some* ‘ *A*)

⟨*proof*⟩

lemma *emeasure-spmf-map-pmf-Some* [*simp*]:
emeasure (*measure-spmf* (*map-pmf Some p*)) *A* = *emeasure* (*measure-pmf p*) *A*
⟨*proof*⟩

lemma *measure-spmf-map-pmf-Some* [*simp*]:
measure (*measure-spmf* (*map-pmf Some p*)) *A* = *measure* (*measure-pmf p*) *A*
⟨*proof*⟩

lemma *nn-integral-measure-spmf*: (∫⁺ *x*. *f x* ∂*measure-spmf p*) = ∫⁺ *x*. *ennreal* (*spmf p x*) ∗ *f x* ∂*count-space UNIV*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *integral-measure-spmf*:
assumes *integrable* (*measure-spmf p*) *f*
shows (∫ *x*. *f x* ∂*measure-spmf p*) = ∫ *x*. *spmf p x* ∗ *f x* ∂*count-space UNIV*
⟨*proof*⟩

lemma *emeasure-spmf-single*: *emeasure* (*measure-spmf p*) {*x*} = *spmf p x*
⟨*proof*⟩

lemma *measurable-measure-spmf*[*measurable*]:
(λ*x*. *measure-spmf* (*M x*)) ∈ *measurable* (*count-space UNIV*) (*subprob-algebra* (*count-space UNIV*))
⟨*proof*⟩

lemma *nn-integral-measure-spmf-conv-measure-pmf*:
assumes [*measurable*]: *f* ∈ *borel-measurable* (*count-space UNIV*)
shows *nn-integral* (*measure-spmf p*) *f* = *nn-integral* (*restrict-space* (*measure-pmf p*) (*range Some*)) (*f* ∘ *the*)
⟨*proof*⟩

lemma *measure-spmf-in-space-subprob-algebra* [*simp*]:
measure-spmf p ∈ *space* (*subprob-algebra* (*count-space UNIV*))
⟨*proof*⟩

lemma *nn-integral-spmf-neq-top*: (∫⁺ *x*. *spmf p x* ∂*count-space UNIV*) ≠ ⊤
⟨*proof*⟩

lemma *SUP-spmf-neq-top'*: (*SUP p*∈*Y*. *ennreal* (*spmf p x*)) ≠ ⊤
⟨*proof*⟩

lemma *SUP-spmf-neq-top*: (*SUP i*. *ennreal* (*spmf* (*Y i*) *x*)) ≠ ⊤
⟨*proof*⟩

lemma *SUP-emeasure-spmf-neq-top*: (*SUP p*∈*Y*. *emeasure* (*measure-spmf p*) *A*) ≠ ⊤

⟨*proof*⟩

25.3 Support

definition *set-spmf* :: *'a spmf* ⇒ *'a set*
where *set-spmf p* = *set-pmf p* ⤜ *set-option*

lemma *set-spmf-rep-eq*: *set-spmf p* = {*x*. *measure* (*measure-spmf p*) {*x*} ≠ *0*}
⟨*proof*⟩

lemma *in-set-spmf*: *x* ∈ *set-spmf p* ⟷ *Some x* ∈ *set-pmf p*
⟨*proof*⟩

lemma *AE-measure-spmf-iff* [*simp*]: (*AE x in measure-spmf p*. *P x*) ⟷ (∀ *x*∈*set-spmf p*. *P x*)
⟨*proof*⟩

lemma *spmf-eq-0-set-spmf*: *spmf p x* = *0* ⟷ *x* ∉ *set-spmf p*
⟨*proof*⟩

lemma *in-set-spmf-iff-spmf*: *x* ∈ *set-spmf p* ⟷ *spmf p x* ≠ *0*
⟨*proof*⟩

lemma *set-spmf-return-pmf-None* [*simp*]: *set-spmf* (*return-pmf None*) = {}
⟨*proof*⟩

lemma *countable-set-spmf* [*simp*]: *countable* (*set-spmf p*)
⟨*proof*⟩

lemma *spmf-eqI*:
assumes ⋀*i*. *spmf p i* = *spmf q i*
shows *p* = *q*
⟨*proof*⟩

lemma *integral-measure-spmf-restrict*:
fixes *f* :: *'a* ⇒ *'b* :: {*banach*, *second-countable-topology*}
shows (∫ *x*. *f x* ∂*measure-spmf M*) = (∫ *x*. *f x* ∂*restrict-space* (*measure-spmf M*) (*set-spmf M*))
⟨*proof*⟩

lemma *nn-integral-measure-spmf'*:
(∫⁺ *x*. *f x* ∂*measure-spmf p*) = ∫⁺ *x*. *ennreal* (*spmf p x*) ∗ *f x* ∂*count-space* (*set-spmf p*)
⟨*proof*⟩

25.4 Functorial structure

abbreviation *map-spmf* :: (*'a* ⇒ *'b*) ⇒ *'a spmf* ⇒ *'b spmf*
where *map-spmf f* ≡ *map-pmf* (*map-option f*)

context begin
⟨*ML*⟩

lemma *map-comp*: *map-spmf f* (*map-spmf g p*) = *map-spmf* (*f* ∘ *g*) *p*
⟨*proof*⟩

lemma *map-id0*: *map-spmf id* = *id*
⟨*proof*⟩

lemma *map-id* [*simp*]: *map-spmf id p* = *p*
⟨*proof*⟩

lemma *map-ident* [*simp*]: *map-spmf* (λ*x*. *x*) *p* = *p*
⟨*proof*⟩

end

lemma *set-map-spmf* [*simp*]: *set-spmf* (*map-spmf f p*) = *f* ‘ *set-spmf p*
⟨*proof*⟩

lemma *map-spmf-cong*:
⟦ *p* = *q*; ⋀*x*. *x* ∈ *set-spmf q* ⟹ *f x* = *g x* ⟧ ⟹ *map-spmf f p* = *map-spmf g q*
⟨*proof*⟩

lemma *map-spmf-cong-simp*:
⟦ *p* = *q*; ⋀*x*. *x* ∈ *set-spmf q* =*simp*=> *f x* = *g x* ⟧
⟹ *map-spmf f p* = *map-spmf g q*
⟨*proof*⟩

lemma *map-spmf-idI*: (⋀*x*. *x* ∈ *set-spmf p* ⟹ *f x* = *x*) ⟹ *map-spmf f p* = *p*
⟨*proof*⟩

lemma *emeasure-map-spmf*:
emeasure (*measure-spmf* (*map-spmf f p*)) *A* = *emeasure* (*measure-spmf p*) (*f* −‘ *A*)
⟨*proof*⟩

lemma *measure-map-spmf*: *measure* (*measure-spmf* (*map-spmf f p*)) *A* = *measure* (*measure-spmf p*) (*f* −‘ *A*)
⟨*proof*⟩

lemma *measure-map-spmf-conv-distr*:
measure-spmf (*map-spmf f p*) = *distr* (*measure-spmf p*) (*count-space UNIV*) *f*
⟨*proof*⟩

lemma *spmf-map-pmf-Some* [*simp*]: *spmf* (*map-pmf Some p*) *i* = *pmf p i*
⟨*proof*⟩

lemma *spmf-map-inj*: ⟦ *inj-on f* (*set-spmf M*); *x* ∈ *set-spmf M* ⟧ ⟹ *spmf* (*map-spmf*

f M) (*f x*) = *spmf M x*
⟨*proof*⟩

lemma *spmf-map-inj'*: *inj f* ⟹ *spmf* (*map-spmf f M*) (*f x*) = *spmf M x*
⟨*proof*⟩

lemma *spmf-map-outside*: *x* ∉ *f* ‘ *set-spmf M* ⟹ *spmf* (*map-spmf f M*) *x* = *0*
⟨*proof*⟩

lemma *ennreal-spmf-map*: *ennreal* (*spmf* (*map-spmf f p*) *x*) = *emeasure* (*measure-spmf p*) (*f* −‘ {*x*})
⟨*proof*⟩

lemma *spmf-map*: *spmf* (*map-spmf f p*) *x* = *measure* (*measure-spmf p*) (*f* −‘ {*x*})
⟨*proof*⟩

lemma *ennreal-spmf-map-conv-nn-integral*:
ennreal (*spmf* (*map-spmf f p*) *x*) = *integral*^{N} (*measure-spmf p*) (*indicator* (*f* −‘ {*x*}))
⟨*proof*⟩

25.5 Monad operations

25.5.1 Return

abbreviation *return-spmf* :: *'a* ⇒ *'a spmf*
where *return-spmf x* ≡ *return-pmf* (*Some x*)

lemma *pmf-return-spmf*: *pmf* (*return-spmf x*) *y* = *indicator* {*y*} (*Some x*)
⟨*proof*⟩

lemma *measure-spmf-return-spmf*: *measure-spmf* (*return-spmf x*) = *Giry-Monad.return* (*count-space UNIV*) *x*
⟨*proof*⟩

lemma *measure-spmf-return-pmf-None* [*simp*]: *measure-spmf* (*return-pmf None*) = *null-measure* (*count-space UNIV*)
⟨*proof*⟩

lemma *set-return-spmf* [*simp*]: *set-spmf* (*return-spmf x*) = {*x*}
⟨*proof*⟩

25.5.2 Bind

definition *bind-spmf* :: *'a spmf* ⇒ (*'a* ⇒ *'b spmf*) ⇒ *'b spmf*
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* $\gg\!=$ (*f* :: *'a* $\Rightarrow$ -) = *return-pmf None*
⟨*proof*⟩

lemma *return-bind-spmf* [*simp*]: *return-spmf x* $\gg\!=$ *f* = *f x*
⟨*proof*⟩

lemma *bind-return-spmf* [*simp*]: *x* $\gg\!=$ *return-spmf* = *x*
⟨*proof*⟩

lemma *bind-spmf-assoc* [*simp*]:
fixes *x* :: *'a spmf* **and** *f* :: *'a* $\Rightarrow$ *'b spmf* **and** *g* :: *'b* $\Rightarrow$ *'c spmf*
shows (*x* $\gg\!=$ *f*) $\gg\!=$ *g* = *x* $\gg\!=$ (λ*y*. *f y* $\gg\!=$ *g*)
⟨*proof*⟩

lemma *pmf-bind-spmf-None*: *pmf* (*p* $\gg\!=$ *f*) *None* = *pmf p None* + $\int$ *x*. *pmf* (*f x*) *None* ∂*measure-spmf p*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *spmf-bind*: *spmf* (*p* $\gg\!=$ *f*) *y* = $\int$ *x*. *spmf* (*f x*) *y* ∂*measure-spmf p*
⟨*proof*⟩

lemma *ennreal-spmf-bind*: *ennreal* (*spmf* (*p* $\gg\!=$ *f*) *x*) = $\int^+$ *y*. *spmf* (*f y*) *x* ∂*measure-spmf p*
⟨*proof*⟩

lemma *measure-spmf-bind-pmf*: *measure-spmf* (*p* $\gg\!=$ *f*) = *measure-pmf p* $\gg\!=$ *measure-spmf* ∘ *f*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *measure-spmf-bind*: *measure-spmf* (*p* $\gg\!=$ *f*) = *measure-spmf p* $\gg\!=$ *measure-spmf* ∘ *f*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *map-spmf-bind-spmf*: *map-spmf f* (*bind-spmf p g*) = *bind-spmf p* (*map-spmf f* ∘ *g*)
⟨*proof*⟩

lemma *bind-map-spmf*: *map-spmf f p* $\gg\!=$ *g* = *p* $\gg\!=$ *g* ∘ *f*
⟨*proof*⟩

lemma *spmf-bind-leI*:
assumes $\bigwedge$*y*. *y* ∈ *set-spmf p* $\Longrightarrow$ *spmf* (*f y*) *x* ≤ *r*
and *0* ≤ *r*
shows *spmf* (*bind-spmf p f*) *x* ≤ *r*
⟨*proof*⟩

lemma *map-spmf-conv-bind-spmf*: *map-spmf f p* = (*p* ⤜ (λ*x*. *return-spmf* (*f x*)))
⟨*proof*⟩

lemma *bind-spmf-cong*:
⟦ *p* = *q*; ⋀*x*. *x* ∈ *set-spmf q* ⟹ *f x* = *g x* ⟧ ⟹ *bind-spmf p f* = *bind-spmf q g*
⟨*proof*⟩

lemma *bind-spmf-cong-simp*:
⟦ *p* = *q*; ⋀*x*. *x* ∈ *set-spmf q* =*simp*=> *f x* = *g x* ⟧
⟹ *bind-spmf p f* = *bind-spmf q g*
⟨*proof*⟩

lemma *set-bind-spmf*: *set-spmf* (*M* ⤜ *f*) = *set-spmf M* ⤜ (*set-spmf* ∘ *f*)
⟨*proof*⟩

lemma *bind-spmf-const-return-None* [*simp*]: *bind-spmf p* (λ-. *return-pmf None*) = *return-pmf None*
⟨*proof*⟩

lemma *bind-commute-spmf*:
bind-spmf p (λ*x*. *bind-spmf q* (*f x*)) = *bind-spmf q* (λ*y*. *bind-spmf p* (λ*x*. *f x y*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

25.6 Relator

abbreviation *rel-spmf* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ '*a spmf* ⇒ '*b spmf* ⇒ *bool*
where *rel-spmf R* ≡ *rel-pmf* (*rel-option R*)

lemma *rel-spmf-mono*:
⟦*rel-spmf A f g*; ⋀*x y*. *A x y* ⟹ *B x y* ⟧ ⟹ *rel-spmf B f g*
⟨*proof*⟩

lemma *rel-spmf-mono-strong*:
⟦ *rel-spmf A f g*; ⋀*x y*. ⟦ *A x y*; *x* ∈ *set-spmf f*; *y* ∈ *set-spmf g* ⟧ ⟹ *B x y* ⟧ ⟹
rel-spmf B f g
⟨*proof*⟩

lemma *rel-spmf-reflI*: (⋀*x*. *x* ∈ *set-spmf p* ⟹ *P x x*) ⟹ *rel-spmf P p p*
⟨*proof*⟩

lemma *rel-spmfI* [*intro?*]:
⟦ ⋀*x y*. (*x*, *y*) ∈ *set-spmf pq* ⟹ *P x y*; *map-spmf fst pq* = *p*; *map-spmf snd pq* = *q* ⟧
⟹ *rel-spmf P p q*
⟨*proof*⟩

lemma *rel-spmfE* [*elim?*, *consumes 1*, *case-names rel-spmf*]:

assumes *rel-spmf P p q*
obtains *pq* **where**
$\bigwedge x\ y.\ (x,\ y) \in$ *set-spmf pq* $\Longrightarrow$ *P x y*
p = *map-spmf fst pq*
q = *map-spmf snd pq*
⟨*proof*⟩

lemma *rel-spmf-simps*:
rel-spmf R p q $\longleftrightarrow$ $(\exists\, pq.\ (\forall\, (x,\ y) \in$ *set-spmf pq*. *R x y*$) \wedge$ *map-spmf fst pq* = *p* $\wedge$ *map-spmf snd pq* = *q*)
⟨*proof*⟩

lemma *spmf-rel-map*:
shows *spmf-rel-map1*: $\bigwedge R\ f\ x.$ *rel-spmf R* (*map-spmf f x*) = *rel-spmf* ($\lambda x.$ *R* (*f x*)) *x*
and *spmf-rel-map2*: $\bigwedge R\ x\ g\ y.$ *rel-spmf R x* (*map-spmf g y*) = *rel-spmf* ($\lambda x\ y.$ *R x* (*g y*)) *x y*
⟨*proof*⟩

lemma *spmf-rel-conversep*: *rel-spmf* R^{-1-1} = (*rel-spmf R*)$^{-1-1}$
⟨*proof*⟩

lemma *spmf-rel-eq*: *rel-spmf* (=) = (=)
⟨*proof*⟩

context includes *lifting-syntax*
begin

lemma *bind-spmf-parametric* [*transfer-rule*]:
(*rel-spmf A* ===> (*A* ===> *rel-spmf B*) ===> *rel-spmf B*) *bind-spmf bind-spmf*
⟨*proof*⟩

lemma *return-spmf-parametric*: (*A* ===> *rel-spmf A*) *return-spmf return-spmf*
⟨*proof*⟩

lemma *map-spmf-parametric*: ((*A* ===> *B*) ===> *rel-spmf A* ===> *rel-spmf B*) *map-spmf map-spmf*
⟨*proof*⟩

lemma *rel-spmf-parametric*:
((*A* ===> *B* ===> (=)) ===> *rel-spmf A* ===> *rel-spmf B* ===> (=)) *rel-spmf rel-spmf*
⟨*proof*⟩

lemma *set-spmf-parametric* [*transfer-rule*]:
(*rel-spmf A* ===> *rel-set A*) *set-spmf set-spmf*
⟨*proof*⟩

lemma *return-spmf-None-parametric*:

(*rel-spmf A*) (*return-pmf None*) (*return-pmf None*)
⟨*proof*⟩

end

lemma *rel-spmf-bindI*:
⟦ *rel-spmf R p q*; ⋀*x y*. *R x y* ⟹ *rel-spmf P* (*f x*) (*g y*) ⟧
⟹ *rel-spmf P* (*p* ⤜ *f*) (*q* ⤜ *g*)
⟨*proof*⟩

lemma *rel-spmf-bind-reflI*:
(⋀*x*. *x* ∈ *set-spmf p* ⟹ *rel-spmf P* (*f x*) (*g x*)) ⟹ *rel-spmf P* (*p* ⤜ *f*) (*p* ⤜ *g*)
⟨*proof*⟩

lemma *rel-pmf-return-pmfI*: *P x y* ⟹ *rel-pmf P* (*return-pmf x*) (*return-pmf y*)
⟨*proof*⟩

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*:
(*rel-pmf A* ===> *rel-pred A* ===> (=)) (λ*p*. *measure* (*measure-pmf p*)) (λ*q*. *measure* (*measure-pmf q*))
⟨*proof*⟩

lemma *measure-spmf-parametric*:
(*rel-spmf A* ===> *rel-pred A* ===> (=)) (λ*p*. *measure* (*measure-spmf p*)) (λ*q*. *measure* (*measure-spmf q*))
⟨*proof*⟩

end

25.7 From *'a pmf* to *'a spmf*

definition *spmf-of-pmf* :: *'a pmf* ⇒ *'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*
⟨*proof*⟩

lemma *spmf-spmf-of-pmf* [*simp*]: *spmf* (*spmf-of-pmf p*) *x* = *pmf p x*
⟨*proof*⟩

lemma *pmf-spmf-of-pmf-None* [*simp*]: *pmf* (*spmf-of-pmf p*) *None* = *0*
⟨*proof*⟩

lemma *emeasure-spmf-of-pmf* [*simp*]: *emeasure* (*measure-spmf* (*spmf-of-pmf p*)) *A* = *emeasure* (*measure-pmf p*) *A*
⟨*proof*⟩

lemma *measure-spmf-spmf-of-pmf* [*simp*]: *measure-spmf* (*spmf-of-pmf p*) = *measure-pmf p*
⟨*proof*⟩

lemma *map-spmf-of-pmf* [*simp*]: *map-spmf f* (*spmf-of-pmf p*) = *spmf-of-pmf* (*map-pmf f p*)
⟨*proof*⟩

lemma *rel-spmf-spmf-of-pmf* [*simp*]: *rel-spmf R* (*spmf-of-pmf p*) (*spmf-of-pmf q*) = *rel-pmf R p q*
⟨*proof*⟩

lemma *spmf-of-pmf-return-pmf* [*simp*]: *spmf-of-pmf* (*return-pmf x*) = *return-spmf x*
⟨*proof*⟩

lemma *bind-spmf-of-pmf* [*simp*]: *bind-spmf* (*spmf-of-pmf p*) *f* = *bind-pmf p f*
⟨*proof*⟩

lemma *set-spmf-bind-pmf*: *set-spmf* (*bind-pmf p f*) = *Set.bind* (*set-pmf p*) (*set-spmf* ∘ *f*)
⟨*proof*⟩

lemma *spmf-of-pmf-bind*: *spmf-of-pmf* (*bind-pmf p f*) = *bind-pmf p* (λ*x*. *spmf-of-pmf* (*f x*))
⟨*proof*⟩

lemma *bind-pmf-return-spmf*: *p* ⤜ (λ*x*. *return-spmf* (*f x*)) = *spmf-of-pmf* (*map-pmf f p*)
⟨*proof*⟩

25.8 Weight of a subprobability

abbreviation *weight-spmf* :: *'a spmf* ⇒ *real*
where *weight-spmf p* ≡ *measure* (*measure-spmf p*) (*space* (*measure-spmf p*))

lemma *weight-spmf-def*: *weight-spmf p* = *measure* (*measure-spmf p*) *UNIV*
⟨*proof*⟩

lemma *weight-spmf-le-1*: *weight-spmf p* ≤ *1*
⟨*proof*⟩

lemma *weight-return-spmf* [*simp*]: *weight-spmf* (*return-spmf x*) = *1*
⟨*proof*⟩

lemma *weight-return-pmf-None* [*simp*]: *weight-spmf* (*return-pmf None*) = *0*
⟨*proof*⟩

lemma *weight-map-spmf* [*simp*]: *weight-spmf* (*map-spmf f p*) = *weight-spmf p*
⟨*proof*⟩

lemma *weight-spmf-of-pmf* [*simp*]: *weight-spmf* (*spmf-of-pmf p*) = *1*
⟨*proof*⟩

lemma *weight-spmf-nonneg*: *weight-spmf p* ≥ *0*
⟨*proof*⟩

lemma (**in** *finite-measure*) *integrable-weight-spmf* [*simp*]:
(λ*x*. *weight-spmf* (*f x*)) ∈ *borel-measurable M* ⟹ *integrable M* (λ*x*. *weight-spmf* (*f x*))
⟨*proof*⟩

lemma *weight-spmf-eq-nn-integral-spmf*: *weight-spmf p* = ∫⁺ *x*. *spmf p x* ∂*count-space UNIV*
⟨*proof*⟩

lemma *weight-spmf-eq-nn-integral-support*:
weight-spmf p = ∫⁺ *x*. *spmf p x* ∂*count-space* (*set-spmf p*)
⟨*proof*⟩

lemma *pmf-None-eq-weight-spmf*: *pmf p None* = *1* − *weight-spmf p*
⟨*proof*⟩

lemma *weight-spmf-conv-pmf-None*: *weight-spmf p* = *1* − *pmf p None*
⟨*proof*⟩

lemma *weight-spmf-lt-0*: ¬ *weight-spmf p* < *0*
⟨*proof*⟩

lemma *spmf-le-weight*: *spmf p x* ≤ *weight-spmf p*
⟨*proof*⟩

lemma *weight-spmf-eq-0*: *weight-spmf p* = *0* ⟷ *p* = *return-pmf None*
⟨*proof*⟩

lemma *weight-bind-spmf*: *weight-spmf* (*x* ⤜ *f*) = *lebesgue-integral* (*measure-spmf x*) (*weight-spmf* ∘ *f*)
⟨*proof*⟩

lemma *rel-spmf-weightD*: *rel-spmf A p q* ⟹ *weight-spmf p* = *weight-spmf q*
⟨*proof*⟩

lemma *rel-spmf-bij-betw*:
assumes *f*: *bij-betw f* (*set-spmf p*) (*set-spmf q*)

and *eq*: $\bigwedge x.\ x \in$ *set-spmf p* $\Longrightarrow$ *spmf p x* = *spmf q* (*f x*)
shows *rel-spmf* ($\lambda x\ y.\ f\ x = y$) *p q*
⟨*proof*⟩

25.9 From density to spmfs

context fixes *f* :: *'a* ⇒ *real* **begin**

definition *embed-spmf* :: *'a spmf*
where *embed-spmf* = *embed-pmf* ($\lambda x.$ *case x of None* ⇒ *1* − *enn2real* ($\int^+ x.$ *ennreal* (*f x*) *∂count-space UNIV*) | *Some x'* ⇒ *max 0* (*f x'*))

context
assumes *prob*: ($\int^+ x.$ *ennreal* (*f x*) *∂count-space UNIV*) ≤ *1*
begin

lemma *nn-integral-embed-spmf-eq-1*:
($\int^+ x.$ *ennreal* (*case x of None* ⇒ *1* − *enn2real* ($\int^+ x.$ *ennreal* (*f x*) *∂count-space UNIV*) | *Some x'* ⇒ *max 0* (*f x'*)) *∂count-space UNIV*) = *1*
(**is** *?lhs* = - **is** ($\int^+ x.$ *?f x ∂?M*) = -)
⟨*proof*⟩

lemma *pmf-embed-spmf-None*: *pmf embed-spmf None* = *1* − *enn2real* ($\int^+ x.$ *ennreal* (*f x*) *∂count-space UNIV*)
⟨*proof*⟩

lemma *spmf-embed-spmf* [*simp*]: *spmf embed-spmf x* = *max 0* (*f x*)
⟨*proof*⟩

end

end

lemma *embed-spmf-K-0*[*simp*]: *embed-spmf* (λ-. *0*) = *return-pmf None*
⟨*proof*⟩

25.10 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* (λn :: *nat*. *uniform* (*llist-n n*)) where *llist-n* is the set of all *llist*s 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* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ spmf \Rightarrow 'a\ spmf \Rightarrow bool$
  where *ord-spmf ord* $\equiv$ *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* (λx. *R* (*f x*)) *p*
  ⟨*proof*⟩

lemma *ord-spmf-map-spmf2*: *ord-spmf R p* (*map-spmf f q*) = *ord-spmf* ($\lambda x\ y$. *R x* (*f y*)) *p q*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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* ⟨*proof*⟩

lemma *ord-spmfI*:
  ⟦ $\bigwedge x\ y$. (*x*, *y*) ∈ *set-spmf pq* $\Longrightarrow$ *ord x y*; *map-spmf fst pq* = *p*; *map-spmf snd pq* = *q* ⟧
  $\Longrightarrow$ *p* $\sqsubseteq$ *q*
  ⟨*proof*⟩

lemma *ord-spmf-None* [*simp*]: *return-pmf None* $\sqsubseteq$ *x*
  ⟨*proof*⟩

lemma *ord-spmf-reflI*: ($\bigwedge x$. *x* ∈ *set-spmf p* $\Longrightarrow$ *ord x x*) $\Longrightarrow$ *p* $\sqsubseteq$ *p*
  ⟨*proof*⟩

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*⟩

end

lemma *ord-spmf-return-spmf2*: *ord-spmf R p* (*return-spmf y*) ⟷ (∀ *x*∈*set-spmf p*. *R x y*)
⟨*proof*⟩

lemma *ord-spmf-mono*: ⟦ *ord-spmf A p q*; ⋀*x y*. *A x y* ⟹ *B x y* ⟧ ⟹ *ord-spmf B p q*
⟨*proof*⟩

lemma *ord-spmf-compp*: *ord-spmf* (*A OO B*) = *ord-spmf A OO ord-spmf B*
⟨*proof*⟩

lemma *ord-spmf-bindI*:
assumes *pq*: *ord-spmf R p q*
and *fg*: ⋀*x y*. *R x y* ⟹ *ord-spmf P* (*f x*) (*g y*)
shows *ord-spmf P* (*p* ⤜ *f*) (*q* ⤜ *g*)
⟨*proof*⟩

lemma *ord-spmf-bind-reflI*:
(⋀*x*. *x* ∈ *set-spmf p* ⟹ *ord-spmf R* (*f x*) (*g x*)) ⟹ *ord-spmf R* (*p* ⤜ *f*) (*p* ⤜ *g*)
⟨*proof*⟩

lemma *ord-pmf-increaseI*:
assumes *le*: ⋀*x*. *spmf p x* ≤ *spmf q x*
and *refl*: ⋀*x*. *x* ∈ *set-spmf p* ⟹ *R x x*
shows *ord-spmf R p q*
⟨*proof*⟩

lemma *ord-spmf-eq-leD*:
assumes *ord-spmf* (=) *p q*
shows *spmf p x* ≤ *spmf q x*
⟨*proof*⟩

lemma *ord-spmf-eqD-set-spmf*: *ord-spmf* (=) *p q* ⟹ *set-spmf p* ⊆ *set-spmf q*
⟨*proof*⟩

lemma *ord-spmf-eqD-emeasure*:
ord-spmf (=) *p q* ⟹ *emeasure* (*measure-spmf p*) *A* ≤ *emeasure* (*measure-spmf q*) *A*
⟨*proof*⟩

lemma *ord-spmf-eqD-measure-spmf*: *ord-spmf* (=) *p q* ⟹ *measure-spmf p* ≤ *measure-spmf q*
⟨*proof*⟩

25.11 CCPO structure for the flat ccpo *ord-option* (=)

context fixes *Y* :: *'a spmf set* **begin**

definition *lub-spmf* :: *'a spmf*
where *lub-spmf* = *embed-spmf* (λ*x*. *enn2real* (*SUP p* ∈ *Y*. *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-spmf* {} = *return-pmf None*
⟨*proof*⟩

context assumes *chain*: *Complete-Partial-Order.chain* (*ord-spmf* (=)) *Y*
begin

lemma *chain-ord-spmf-eqD*: *Complete-Partial-Order.chain* (≤) ((λ*p x*. *ennreal* (*spmf p x*)) ‘ *Y*)
(**is** *Complete-Partial-Order.chain* - (*?f* ‘ -))
⟨*proof*⟩

lemma *ord-spmf-eq-pmf-None-eq*:
assumes *le*: *ord-spmf* (=) *p q*
and *None*: *pmf p None* = *pmf q None*
shows *p* = *q*
⟨*proof*⟩

lemma *ord-spmf-eqD-pmf-None*:
assumes *ord-spmf* (=) *x y*
shows *pmf x None* ≥ *pmf y None*
⟨*proof*⟩

Chains on *'a spmf* maintain countable support. Thanks to Johannes Hölzl for the proof idea.

lemma *spmf-chain-countable*: *countable* (⋃*p*∈*Y*. *set-spmf p*)
⟨*proof*⟩

lemma *lub-spmf-subprob*: (∫$^+$ *x*. (*SUP p* ∈ *Y*. *ennreal* (*spmf p x*)) ∂*count-space UNIV*) ≤ *1*
⟨*proof*⟩

lemma *spmf-lub-spmf*:
assumes *Y* ≠ {}
shows *spmf lub-spmf x* = (*SUP p* ∈ *Y*. *spmf p x*)
⟨*proof*⟩

lemma *ennreal-spmf-lub-spmf*: *Y* ≠ {} ⟹ *ennreal* (*spmf lub-spmf x*) = (*SUP p*∈*Y*. *ennreal* (*spmf p x*))
⟨*proof*⟩

lemma *lub-spmf-upper*:
assumes *p*: *p* ∈ *Y*
shows *ord-spmf* (=) *p lub-spmf*
⟨*proof*⟩

lemma *lub-spmf-least*:
assumes *z*: $\bigwedge x.\ x \in Y \Longrightarrow$ *ord-spmf* (=) *x z*
shows *ord-spmf* (=) *lub-spmf z*
⟨*proof*⟩

lemma *set-lub-spmf*: *set-spmf lub-spmf* = ($\bigcup p{\in}Y.$ *set-spmf p*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *emeasure-lub-spmf*:
assumes *Y*: $Y \neq \{\}$
shows *emeasure* (*measure-spmf lub-spmf*) *A* = (*SUP* $y{\in}Y.$ *emeasure* (*measure-spmf y*) *A*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *measure-lub-spmf*:
assumes *Y*: $Y \neq \{\}$
shows *measure* (*measure-spmf lub-spmf*) *A* = (*SUP* $y{\in}Y.$ *measure* (*measure-spmf y*) *A*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *weight-lub-spmf*:
assumes *Y*: $Y \neq \{\}$
shows *weight-spmf lub-spmf* = (*SUP* $y{\in}Y.$ *weight-spmf y*)
⟨*proof*⟩

lemma *measure-spmf-lub-spmf*:
assumes *Y*: $Y \neq \{\}$
shows *measure-spmf lub-spmf* = (*SUP* $p{\in}Y.$ *measure-spmf p*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

end

lemma *partial-function-definitions-spmf*: *partial-function-definitions* (*ord-spmf* (=)) *lub-spmf*
(**is** *partial-function-definitions ?R* -)
⟨*proof*⟩

lemma *ccpo-spmf*: *class.ccpo lub-spmf* (*ord-spmf* (=)) (*mk-less* (*ord-spmf* (=)))
⟨*proof*⟩

interpretation *spmf*: *partial-function-definitions ord-spmf* (=) *lub-spmf*
rewrites *lub-spmf* {} ≡ *return-pmf None*
⟨*proof*⟩

⟨*ML*⟩

declare *spmf.leq-refl*[*simp*]
declare *admissible-leI*[*OF ccpo-spmf*, *cont-intro*]

abbreviation *mono-spmf* ≡ *monotone* (*fun-ord* (*ord-spmf* (=))) (*ord-spmf* (=))

lemma *lub-spmf-const* [*simp*]: *lub-spmf* {*p*} = *p*
⟨*proof*⟩

lemma *bind-spmf-mono'*:
assumes *fg*: *ord-spmf* (=) *f g*
and *hk*: ⋀*x* :: *'a*. *ord-spmf* (=) (*h x*) (*k x*)
shows *ord-spmf* (=) (*f* ⤜ *h*) (*g* ⤜ *k*)
⟨*proof*⟩

lemma *bind-spmf-mono* [*partial-function-mono*]:
assumes *mf*: *mono-spmf B* **and** *mg*: ⋀*y*. *mono-spmf* (λ*f*. *C y f*)
shows *mono-spmf* (λ*f*. *bind-spmf* (*B f*) (λ*y*. *C y f*))
⟨*proof*⟩

lemma *monotone-bind-spmf1*: *monotone* (*ord-spmf* (=)) (*ord-spmf* (=)) (λ*y*. *bind-spmf y g*)
⟨*proof*⟩

lemma *monotone-bind-spmf2*:
assumes *g*: ⋀*x*. *monotone ord* (*ord-spmf* (=)) (λ*y*. *g y x*)
shows *monotone ord* (*ord-spmf* (=)) (λ*y*. *bind-spmf p* (*g y*))
⟨*proof*⟩

lemma *bind-lub-spmf*:
assumes *chain*: *Complete-Partial-Order.chain* (*ord-spmf* (=)) *Y*
shows *bind-spmf* (*lub-spmf Y*) *f* = *lub-spmf* ((λ*p*. *bind-spmf p f*) ‘ *Y*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *map-lub-spmf*:
Complete-Partial-Order.chain (*ord-spmf* (=)) *Y*
⟹ *map-spmf f* (*lub-spmf Y*) = *lub-spmf* (*map-spmf f* ‘ *Y*)
⟨*proof*⟩

lemma *mcont-bind-spmf1*: *mcont lub-spmf* (*ord-spmf* (=)) *lub-spmf* (*ord-spmf* (=)) (λ*y*. *bind-spmf y f*)
⟨*proof*⟩

lemma *bind-lub-spmf2*:
assumes *chain*: *Complete-Partial-Order.chain ord Y*
and *g*: ⋀*y*. *monotone ord* (*ord-spmf* (=)) (*g y*)
shows *bind-spmf x* (λ*y*. *lub-spmf* (*g y* ‘ *Y*)) = *lub-spmf* ((λ*p*. *bind-spmf x* (λ*y*. *g y p*)) ‘ *Y*)

(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *mcont-bind-spmf* [*cont-intro*]:
assumes *f*: *mcont luba orda lub-spmf* (*ord-spmf* (=)) *f*
and *g*: ⋀*y*. *mcont luba orda lub-spmf* (*ord-spmf* (=)) (*g y*)
shows *mcont luba orda lub-spmf* (*ord-spmf* (=)) (λ*x*. *bind-spmf* (*f x*) (λ*y*. *g y x*))
⟨*proof*⟩

lemma *bind-pmf-mono* [*partial-function-mono*]:
(⋀*y*. *mono-spmf* (λ*f*. *C y f*)) ⟹ *mono-spmf* (λ*f*. *bind-pmf p* (λ*x*. *C x f*))
⟨*proof*⟩

lemma *map-spmf-mono* [*partial-function-mono*]: *mono-spmf B* ⟹ *mono-spmf* (λ*g*. *map-spmf f* (*B g*))
⟨*proof*⟩

lemma *mcont-map-spmf* [*cont-intro*]:
mcont luba orda lub-spmf (*ord-spmf* (=)) *g*
⟹ *mcont luba orda lub-spmf* (*ord-spmf* (=)) (λ*x*. *map-spmf f* (*g x*))
⟨*proof*⟩

lemma *monotone-set-spmf*: *monotone* (*ord-spmf* (=)) (⊆) *set-spmf*
⟨*proof*⟩

lemma *cont-set-spmf*: *cont lub-spmf* (*ord-spmf* (=)) *Union* (⊆) *set-spmf*
⟨*proof*⟩

lemma *mcont2mcont-set-spmf*[*THEN mcont2mcont*, *cont-intro*]:
shows *mcont-set-spmf*: *mcont lub-spmf* (*ord-spmf* (=)) *Union* (⊆) *set-spmf*
⟨*proof*⟩

lemma *monotone-spmf*: *monotone* (*ord-spmf* (=)) (≤) (λ*p*. *spmf p x*)
⟨*proof*⟩

lemma *cont-spmf*: *cont lub-spmf* (*ord-spmf* (=)) *Sup* (≤) (λ*p*. *spmf p x*)
⟨*proof*⟩

lemma *mcont-spmf*: *mcont lub-spmf* (*ord-spmf* (=)) *Sup* (≤) (λ*p*. *spmf p x*)
⟨*proof*⟩

lemma *cont-ennreal-spmf*: *cont lub-spmf* (*ord-spmf* (=)) *Sup* (≤) (λ*p*. *ennreal* (*spmf p x*))
⟨*proof*⟩

lemma *mcont2mcont-ennreal-spmf* [*THEN mcont2mcont*, *cont-intro*]:
shows *mcont-ennreal-spmf*: *mcont lub-spmf* (*ord-spmf* (=)) *Sup* (≤) (λ*p*. *ennreal* (*spmf p x*))

⟨*proof*⟩

lemma *nn-integral-map-spmf* [*simp*]: *nn-integral* (*measure-spmf* (*map-spmf f p*)) *g* = *nn-integral* (*measure-spmf p*) (*g* ∘ *f*)
⟨*proof*⟩

25.11.1 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*⟩

locale *rel-spmf-characterisation* =
assumes *rel-pmf-measureI*:
⋀(*R* :: *'a option* ⇒ *'b option* ⇒ *bool*) *p q*.
(⋀*A*. *measure* (*measure-pmf p*) *A* ≤ *measure* (*measure-pmf q*) {*y*. ∃ *x*∈*A*. *R x y*})
⟹ *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*: ⋀*A*. *measure* (*measure-spmf p*) *A* ≤ *measure* (*measure-spmf q*) {*y*. ∃ *x*∈*A*. *R x y*}
assumes *eq2*: *weight-spmf q* ≤ *weight-spmf p*
shows *rel-spmf R p q*
⟨*proof*⟩

lemma *admissible-rel-spmf*:
ccpo.admissible (*prod-lub lub-spmf lub-spmf*) (*rel-prod* (*ord-spmf* (=)) (*ord-spmf* (=))) (*case-prod* (*rel-spmf R*))
(**is** *ccpo.admissible ?lub ?ord ?P*)
⟨*proof*⟩

lemma *admissible-rel-spmf-mcont* [*cont-intro*]:
⟦ *mcont lub ord lub-spmf* (*ord-spmf* (=)) *f*; *mcont lub ord lub-spmf* (*ord-spmf* (=)) *g* ⟧
⟹ *ccpo.admissible lub ord* (λ*x*. *rel-spmf R* (*f x*) (*g x*))
⟨*proof*⟩

context includes *lifting-syntax*
begin

lemma *fixp-spmf-parametric'*:
assumes *f*: ⋀*x*. *monotone* (*ord-spmf* (=)) (*ord-spmf* (=)) *F*

and g: $\bigwedge x.$ *monotone* (*ord-spmf* (=)) (*ord-spmf* (=)) G
and *param*: (*rel-spmf R* ===> *rel-spmf R*) F G
shows (*rel-spmf R*) (*ccpo.fixp lub-spmf* (*ord-spmf* (=)) F) (*ccpo.fixp lub-spmf* (*ord-spmf* (=)) G)
⟨*proof*⟩

lemma *fixp-spmf-parametric*:
assumes f: $\bigwedge x.$ *mono-spmf* ($\lambda f.$ F f x)
and g: $\bigwedge x.$ *mono-spmf* ($\lambda f.$ G f x)
and *param*: ((A ===> *rel-spmf R*) ===> A ===> *rel-spmf R*) F G
shows (A ===> *rel-spmf R*) (*spmf.fixp-fun F*) (*spmf.fixp-fun G*)
⟨*proof*⟩

end

end

end

25.12 Restrictions on spmfs

definition *restrict-spmf* :: *'a spmf* ⇒ *'a set* ⇒ *'a spmf* (**infixl** ‹⟜› *110*)
where p ⟜ A = *map-pmf* ($\lambda x.$ x ⤜ ($\lambda y.$ *if* $y \in A$ *then Some* y *else None*)) p

lemma *set-restrict-spmf* [*simp*]: *set-spmf* (p ⟜ A) = *set-spmf* $p \cap A$
⟨*proof*⟩

lemma *restrict-map-spmf*: *map-spmf f p* ⟜ A = *map-spmf f* (p ⟜ (f −‘ A))
⟨*proof*⟩

lemma *restrict-restrict-spmf* [*simp*]: p ⟜ A ⟜ B = p ⟜ ($A \cap B$)
⟨*proof*⟩

lemma *restrict-spmf-empty* [*simp*]: p ⟜ {} = *return-pmf None*
⟨*proof*⟩

lemma *restrict-spmf-UNIV* [*simp*]: p ⟜ *UNIV* = p
⟨*proof*⟩

lemma *spmf-restrict-spmf-outside* [*simp*]: $x \notin A \Longrightarrow$ *spmf* (p ⟜ A) x = *0*
⟨*proof*⟩

lemma *emeasure-restrict-spmf* [*simp*]: *emeasure* (*measure-spmf* (p ⟜ A)) X = *emeasure* (*measure-spmf p*) ($X \cap A$)
⟨*proof*⟩

lemma *measure-restrict-spmf* [*simp*]:
measure (*measure-spmf* (p ⟜ A)) X = *measure* (*measure-spmf p*) ($X \cap A$)
⟨*proof*⟩

lemma *spmf-restrict-spmf*: *spmf* $(p \upharpoonright A)\ x = (\mathit{if}\ x \in A\ \mathit{then}\ \mathit{spmf}\ p\ x\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *spmf-restrict-spmf-inside* [*simp*]: $x \in A \Longrightarrow \mathit{spmf}\ (p \upharpoonright A)\ x = \mathit{spmf}\ p\ x$
⟨*proof*⟩

lemma *pmf-restrict-spmf-None*: *pmf* $(p \upharpoonright A)$ *None* = *pmf p None* + *measure* (*measure-spmf p*) $(-A)$
⟨*proof*⟩

lemma *restrict-spmf-trivial*: $(\bigwedge x.\ x \in \mathit{set\text{-}spmf}\ p \Longrightarrow x \in A) \Longrightarrow p \upharpoonright A = p$
⟨*proof*⟩

lemma *restrict-spmf-trivial'*: *set-spmf* $p \subseteq A \Longrightarrow p \upharpoonright A = p$
⟨*proof*⟩

lemma *restrict-return-spmf*: *return-spmf* $x \upharpoonright A = (\mathit{if}\ x \in A\ \mathit{then}\ \mathit{return\text{-}spmf}\ x\ \mathit{else}\ \mathit{return\text{-}pmf}\ \mathit{None})$
⟨*proof*⟩

lemma *restrict-return-spmf-inside* [*simp*]: $x \in A \Longrightarrow \mathit{return\text{-}spmf}\ x \upharpoonright A = \mathit{return\text{-}spmf}\ x$
⟨*proof*⟩

lemma *restrict-return-spmf-outside* [*simp*]: $x \notin A \Longrightarrow \mathit{return\text{-}spmf}\ x \upharpoonright A = \mathit{return\text{-}pmf}\ \mathit{None}$
⟨*proof*⟩

lemma *restrict-spmf-return-pmf-None* [*simp*]: *return-pmf None* $\upharpoonright A$ = *return-pmf None*
⟨*proof*⟩

lemma *restrict-bind-pmf*: *bind-pmf p g* $\upharpoonright A = p \gg\!= (\lambda x.\ g\ x \upharpoonright A)$
⟨*proof*⟩

lemma *restrict-bind-spmf*: *bind-spmf p g* $\upharpoonright A = p \gg\!= (\lambda x.\ g\ x \upharpoonright A)$
⟨*proof*⟩

lemma *bind-restrict-pmf*: *bind-pmf* $(p \upharpoonright A)\ g = p \gg\!= (\lambda x.\ \mathit{if}\ x \in \mathit{Some}\ `\ A\ \mathit{then}\ g\ x\ \mathit{else}\ g\ \mathit{None})$
⟨*proof*⟩

lemma *bind-restrict-spmf*: *bind-spmf* $(p \upharpoonright A)\ g = p \gg\!= (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ g\ x\ \mathit{else}\ \mathit{return\text{-}pmf}\ \mathit{None})$
⟨*proof*⟩

lemma *spmf-map-restrict*: *spmf* $(\mathit{map\text{-}spmf}\ \mathit{fst}\ (p \upharpoonright (\mathit{snd} -` \{y\})))\ x = \mathit{spmf}\ p\ (x, y)$

⟨*proof*⟩

lemma *measure-eqI-restrict-spmf*:
assumes *rel-spmf R* (*restrict-spmf p A*) (*restrict-spmf q B*)
shows *measure* (*measure-spmf p*) *A* = *measure* (*measure-spmf q*) *B*
⟨*proof*⟩

25.13 Subprobability distributions of sets

definition *spmf-of-set* :: *'a set* ⇒ *'a spmf*
where
spmf-of-set A = (*if finite A* ∧ *A* ≠ {} *then spmf-of-pmf* (*pmf-of-set A*) *else return-pmf None*)

lemma *spmf-of-set*: *spmf* (*spmf-of-set A*) *x* = *indicator A x* / *card A*
⟨*proof*⟩

lemma *pmf-spmf-of-set-None* [*simp*]: *pmf* (*spmf-of-set A*) *None* = *indicator* {*A*. *infinite A* ∨ *A* = {}} *A*
⟨*proof*⟩

lemma *set-spmf-of-set*: *set-spmf* (*spmf-of-set A*) = (*if finite A then A else* {})
⟨*proof*⟩

lemma *set-spmf-of-set-finite* [*simp*]: *finite A* ⟹ *set-spmf* (*spmf-of-set A*) = *A*
⟨*proof*⟩

lemma *spmf-of-set-singleton*: *spmf-of-set* {*x*} = *return-spmf x*
⟨*proof*⟩

lemma *map-spmf-of-set-inj-on* [*simp*]:
inj-on f A ⟹ *map-spmf f* (*spmf-of-set A*) = *spmf-of-set* (*f* ‘ *A*)
⟨*proof*⟩

lemma *spmf-of-pmf-pmf-of-set* [*simp*]:
⟦ *finite A*; *A* ≠ {} ⟧ ⟹ *spmf-of-pmf* (*pmf-of-set A*) = *spmf-of-set A*
⟨*proof*⟩

lemma *weight-spmf-of-set*:
weight-spmf (*spmf-of-set A*) = (*if finite A* ∧ *A* ≠ {} *then 1 else 0*)
⟨*proof*⟩

lemma *weight-spmf-of-set-finite* [*simp*]: ⟦ *finite A*; *A* ≠ {} ⟧ ⟹ *weight-spmf* (*spmf-of-set A*) = *1*
⟨*proof*⟩

lemma *weight-spmf-of-set-infinite* [*simp*]: *infinite A* ⟹ *weight-spmf* (*spmf-of-set A*) = *0*
⟨*proof*⟩

lemma *measure-spmf-spmf-of-set*:
measure-spmf (*spmf-of-set A*) = (*if finite A* ∧ *A* ≠ {} *then measure-pmf* (*pmf-of-set A*) *else null-measure* (*count-space UNIV*))
⟨*proof*⟩

lemma *emeasure-spmf-of-set*:
emeasure (*measure-spmf* (*spmf-of-set S*)) *A* = *card* (*S* ∩ *A*) / *card S*
⟨*proof*⟩

lemma *measure-spmf-of-set*:
measure (*measure-spmf* (*spmf-of-set S*)) *A* = *card* (*S* ∩ *A*) / *card S*
⟨*proof*⟩

lemma *nn-integral-spmf-of-set*: *nn-integral* (*measure-spmf* (*spmf-of-set A*)) *f* = *sum f A* / *card A*
⟨*proof*⟩

lemma *integral-spmf-of-set*: $integral^L$ (*measure-spmf* (*spmf-of-set A*)) *f* = *sum f A* / *card A*
⟨*proof*⟩

notepad begin — *pmf-of-set* is not fully parametric.
⟨*proof*⟩
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*⟩

lemma *rel-spmf-of-set-bij*:
assumes *f*: *bij-betw f A B*
and *R*: ⋀*x*. *x* ∈ *A* ⟹ *R x* (*f x*)
shows *rel-spmf R* (*spmf-of-set A*) (*spmf-of-set B*)
⟨*proof*⟩

context includes *lifting-syntax*
begin

lemma *rel-spmf-of-set*:
assumes *bi-unique R*
shows (*rel-set R* ===> *rel-spmf R*) *spmf-of-set spmf-of-set*
⟨*proof*⟩

end

lemma *map-mem-spmf-of-set*:
assumes *finite B B* ≠ {}
shows *map-spmf* (λ*x*. *x* ∈ *A*) (*spmf-of-set B*) = *spmf-of-pmf* (*bernoulli-pmf* (*card* (*A* ∩ *B*) / *card B*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

abbreviation *coin-spmf* :: *bool spmf*
where *coin-spmf* ≡ *spmf-of-set UNIV*

lemma *map-eq-const-coin-spmf*: *map-spmf* ((=) *c*) *coin-spmf* = *coin-spmf*
⟨*proof*⟩

lemma *bind-coin-spmf-eq-const*: *coin-spmf* ⤜ (λ*x* :: *bool*. *return-spmf* (*b* = *x*)) = *coin-spmf*
⟨*proof*⟩

lemma *bind-coin-spmf-eq-const'*: *coin-spmf* ⤜ (λ*x* :: *bool*. *return-spmf* (*x* = *b*)) = *coin-spmf*
⟨*proof*⟩

25.14 Losslessness

definition *lossless-spmf* :: *'a spmf* ⇒ *bool*
where *lossless-spmf p* ⟷ *weight-spmf p* = *1*

lemma *lossless-iff-pmf-None*: *lossless-spmf p* ⟷ *pmf p None* = *0*
⟨*proof*⟩

lemma *lossless-return-spmf* [*iff*]: *lossless-spmf* (*return-spmf x*)
⟨*proof*⟩

lemma *lossless-return-pmf-None* [*iff*]: ¬ *lossless-spmf* (*return-pmf None*)
⟨*proof*⟩

lemma *lossless-map-spmf* [*simp*]: *lossless-spmf* (*map-spmf f p*) ⟷ *lossless-spmf p*
⟨*proof*⟩

lemma *lossless-bind-spmf* [*simp*]:
lossless-spmf (*p* ⤜ *f*) ⟷ *lossless-spmf p* ∧ (∀ *x*∈*set-spmf p*. *lossless-spmf* (*f x*))
⟨*proof*⟩

lemma *lossless-weight-spmfD*: *lossless-spmf p* ⟹ *weight-spmf p* = *1*
⟨*proof*⟩

lemma *lossless-iff-set-pmf-None*:

lossless-spmf p $\longleftrightarrow$ *None* $\notin$ *set-pmf p*
⟨*proof*⟩

lemma *lossless-spmf-of-set* [*simp*]: *lossless-spmf* (*spmf-of-set A*) $\longleftrightarrow$ *finite A* $\wedge$ *A* $\neq$ {}
⟨*proof*⟩

lemma *lossless-spmf-spmf-of-spmf* [*simp*]: *lossless-spmf* (*spmf-of-pmf p*)
⟨*proof*⟩

lemma *lossless-spmf-bind-pmf* [*simp*]:
lossless-spmf (*bind-pmf p f*) $\longleftrightarrow$ ($\forall$ *x*$\in$*set-pmf p. lossless-spmf* (*f x*))
⟨*proof*⟩

lemma *lossless-spmf-conv-spmf-of-pmf*: *lossless-spmf p* $\longleftrightarrow$ ($\exists$ *p′. p* = *spmf-of-pmf p′*)
⟨*proof*⟩

lemma *spmf-False-conv-True*: *lossless-spmf p* $\Longrightarrow$ *spmf p False* = *1* − *spmf p True*
⟨*proof*⟩

lemma *spmf-True-conv-False*: *lossless-spmf p* $\Longrightarrow$ *spmf p True* = *1* − *spmf p False*
⟨*proof*⟩

lemma *bind-eq-return-spmf*:
bind-spmf p f = *return-spmf x* $\longleftrightarrow$ ($\forall$ *y*$\in$*set-spmf p. f y* = *return-spmf x*) $\wedge$ *lossless-spmf p*
⟨*proof*⟩

lemma *rel-spmf-return-spmf2*:
rel-spmf R p (*return-spmf x*) $\longleftrightarrow$ *lossless-spmf p* $\wedge$ ($\forall$ *a*$\in$*set-spmf p. R a x*)
⟨*proof*⟩

lemma *rel-spmf-return-spmf1*:
rel-spmf R (*return-spmf x*) *p* $\longleftrightarrow$ *lossless-spmf p* $\wedge$ ($\forall$ *a*$\in$*set-spmf p. R x a*)
⟨*proof*⟩

lemma *rel-spmf-bindI1*:
assumes *f*: $\bigwedge$*x. x* $\in$ *set-spmf p* $\Longrightarrow$ *rel-spmf R* (*f x*) *q*
and *p*: *lossless-spmf p*
shows *rel-spmf R* (*bind-spmf p f*) *q*
⟨*proof*⟩

lemma *rel-spmf-bindI2*:
⟦ $\bigwedge$*x. x* $\in$ *set-spmf q* $\Longrightarrow$ *rel-spmf R p* (*f x*); *lossless-spmf q* ⟧
$\Longrightarrow$ *rel-spmf R p* (*bind-spmf q f*)
⟨*proof*⟩

25.15 Scaling

definition *scale-spmf* :: *real* ⇒ *'a spmf* ⇒ *'a spmf*
where
scale-spmf r p = *embed-spmf* (λ*x*. *min* (*inverse* (*weight-spmf p*)) (*max 0 r*) ∗ *spmf p x*)

lemma *scale-spmf-le-1*:
(∫⁺ *x*. *min* (*inverse* (*weight-spmf p*)) (*max 0 r*) ∗ *spmf p x* ∂*count-space UNIV*) ≤ *1* (**is** *?lhs* ≤ -)
⟨*proof*⟩

lemma *spmf-scale-spmf*: *spmf* (*scale-spmf r p*) *x* = *max 0* (*min* (*inverse* (*weight-spmf p*)) *r*) ∗ *spmf p x* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *real-inverse-le-1-iff*: **fixes** *x* :: *real*
shows ⟦ *0* ≤ *x*; *x* ≤ *1* ⟧ ⟹ *1* / *x* ≤ *1* ⟷ *x* = *1* ∨ *x* = *0*
⟨*proof*⟩

lemma *spmf-scale-spmf'*: *r* ≤ *1* ⟹ *spmf* (*scale-spmf r p*) *x* = *max 0 r* ∗ *spmf p x*
⟨*proof*⟩

lemma *scale-spmf-neg*: *r* ≤ *0* ⟹ *scale-spmf r p* = *return-pmf None*
⟨*proof*⟩

lemma *scale-spmf-return-None* [*simp*]: *scale-spmf r* (*return-pmf None*) = *return-pmf None*
⟨*proof*⟩

lemma *scale-spmf-conv-bind-bernoulli*:
assumes *r* ≤ *1*
shows *scale-spmf r p* = *bind-pmf* (*bernoulli-pmf r*) (λ*b*. *if b then p else return-pmf None*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *nn-integral-spmf*: (∫⁺ *x*. *spmf p x* ∂*count-space A*) = *emeasure* (*measure-spmf p*) *A*
⟨*proof*⟩

lemma *measure-spmf-scale-spmf*: *measure-spmf* (*scale-spmf r p*) = *scale-measure* (*min* (*inverse* (*weight-spmf p*)) *r*) (*measure-spmf p*)
⟨*proof*⟩

lemma *measure-spmf-scale-spmf'*:
assumes *r* ≤ *1*
shows *measure-spmf* (*scale-spmf r p*) = *scale-measure r* (*measure-spmf p*)
⟨*proof*⟩

lemma *scale-spmf-1* [*simp*]: *scale-spmf 1 p = p*
⟨*proof*⟩

lemma *scale-spmf-0* [*simp*]: *scale-spmf 0 p = return-pmf None*
⟨*proof*⟩

lemma *bind-scale-spmf*:
assumes *r*: *r ≤ 1*
shows *bind-spmf (scale-spmf r p) f = bind-spmf p (λx. scale-spmf r (f x))*
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *scale-bind-spmf*:
assumes *r ≤ 1*
shows *scale-spmf r (bind-spmf p f) = bind-spmf p (λx. scale-spmf r (f x))*
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *bind-spmf-const*: *bind-spmf p (λx. q) = scale-spmf (weight-spmf p) q* (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *map-scale-spmf*: *map-spmf f (scale-spmf r p) = scale-spmf r (map-spmf f p)* (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *set-scale-spmf*: *set-spmf (scale-spmf r p) = (if r > 0 then set-spmf p else {})*
⟨*proof*⟩

lemma *set-scale-spmf'* [*simp*]: *0 < r ⟹ set-spmf (scale-spmf r p) = set-spmf p*
⟨*proof*⟩

lemma *rel-spmf-scaleI*:
assumes *r > 0 ⟹ rel-spmf A p q*
shows *rel-spmf A (scale-spmf r p) (scale-spmf r q)*
⟨*proof*⟩

lemma *weight-scale-spmf*: *weight-spmf (scale-spmf r p) = min 1 (max 0 r * weight-spmf p)*
⟨*proof*⟩

lemma *weight-scale-spmf'* [*simp*]:
⟦ *0 ≤ r*; *r ≤ 1* ⟧ *⟹ weight-spmf (scale-spmf r p) = r * weight-spmf p*
⟨*proof*⟩

lemma *pmf-scale-spmf-None*:
*pmf (scale-spmf k p) None = 1 − min 1 (max 0 k * (1 − pmf p None))*
⟨*proof*⟩

lemma *scale-scale-spmf*:
*scale-spmf r (scale-spmf r′ p) = scale-spmf (r * max 0 (min (inverse (weight-spmf p)) r′)) p*
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *scale-scale-spmf′* [*simp*]:
assumes *0 ≤ r r ≤ 1 0 ≤ r′ r′ ≤ 1*
shows *scale-spmf r (scale-spmf r′ p) = scale-spmf (r * r′) p*
⟨*proof*⟩

lemma *scale-spmf-eq-same*: *scale-spmf r p = p ⟷ weight-spmf p = 0 ∨ r = 1 ∨ r ≥ 1 ∧ weight-spmf p = 1*
(**is** *?lhs ⟷ ?rhs*)
⟨*proof*⟩

lemma *map-const-spmf-of-set*:
⟦ *finite A*; *A ≠ {}* ⟧ ⟹ *map-spmf* (λ-. *c*) (*spmf-of-set A*) = *return-spmf c*
⟨*proof*⟩

25.16 Conditional spmfs

lemma *set-pmf-Int-Some*: *set-pmf p ∩ Some ‘ A = {} ⟷ set-spmf p ∩ A = {}*
⟨*proof*⟩

lemma *measure-spmf-zero-iff*: *measure* (*measure-spmf p*) *A = 0 ⟷ set-spmf p ∩ A = {}*
⟨*proof*⟩

definition *cond-spmf* :: *'a spmf ⇒ 'a set ⇒ 'a spmf*
where *cond-spmf p A = (if set-spmf p ∩ 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 ∩ A*
⟨*proof*⟩

lemma *cond-map-spmf* [*simp*]: *cond-spmf* (*map-spmf f p*) *A* = *map-spmf f* (*cond-spmf p* (*f −‘ A*))
⟨*proof*⟩

lemma *spmf-cond-spmf* [*simp*]:
spmf (*cond-spmf p A*) *x* = (*if x ∈ A then spmf p x / measure* (*measure-spmf p*) *A else 0*)
⟨*proof*⟩

lemma *bind-eq-return-pmf-None*:
bind-spmf p f = return-pmf None ⟷ (∀ *x*∈*set-spmf p*. *f x = return-pmf None*)
⟨*proof*⟩

lemma *return-pmf-None-eq-bind*:
return-pmf None = bind-spmf p f ⟷ (∀ x∈set-spmf p. f x = return-pmf None)
⟨*proof*⟩

25.17 Product spmf

definition *pair-spmf* :: *'a spmf ⇒ 'b spmf ⇒ ('a × 'b) spmf*
where *pair-spmf p q = bind-pmf (pair-pmf p q) (λxy. case xy of (Some x, Some y) ⇒ return-spmf (x, y) | - ⇒ return-pmf None)*

lemma *map-fst-pair-spmf* [*simp*]: *map-spmf fst (pair-spmf p q) = scale-spmf (weight-spmf q) p*
⟨*proof*⟩

lemma *map-snd-pair-spmf* [*simp*]: *map-spmf snd (pair-spmf p q) = scale-spmf (weight-spmf p) q*
⟨*proof*⟩

lemma *set-pair-spmf* [*simp*]: *set-spmf (pair-spmf p q) = set-spmf p × set-spmf q*
⟨*proof*⟩

lemma *spmf-pair* [*simp*]: *spmf (pair-spmf p q) (x, y) = spmf p x * spmf q y* (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *pair-map-spmf2*: *pair-spmf p (map-spmf f q) = map-spmf (apsnd f) (pair-spmf p q)*
⟨*proof*⟩

lemma *pair-map-spmf1*: *pair-spmf (map-spmf f p) q = map-spmf (apfst f) (pair-spmf p q)*
⟨*proof*⟩

lemma *pair-map-spmf*: *pair-spmf (map-spmf f p) (map-spmf g q) = map-spmf (map-prod f g) (pair-spmf p q)*
⟨*proof*⟩

lemma *pair-spmf-alt-def*: *pair-spmf p q = bind-spmf p (λx. bind-spmf q (λy. return-spmf (x, y)))*
⟨*proof*⟩

lemma *weight-pair-spmf* [*simp*]: *weight-spmf (pair-spmf p q) = weight-spmf p * weight-spmf q*
⟨*proof*⟩

lemma *pair-scale-spmf1*:
r ≤ 1 ⟹ pair-spmf (scale-spmf r p) q = scale-spmf r (pair-spmf p q)
⟨*proof*⟩

lemma *pair-scale-spmf2*:
$r \leq 1 \Longrightarrow$ *pair-spmf p* (*scale-spmf r q*) = *scale-spmf r* (*pair-spmf p q*)
⟨*proof*⟩

lemma *pair-spmf-return-None1* [*simp*]: *pair-spmf* (*return-pmf None*) *p* = *return-pmf None*
⟨*proof*⟩

lemma *pair-spmf-return-None2* [*simp*]: *pair-spmf p* (*return-pmf None*) = *return-pmf None*
⟨*proof*⟩

lemma *pair-spmf-return-spmf1*: *pair-spmf* (*return-spmf x*) *q* = *map-spmf* (*Pair x*) *q*
⟨*proof*⟩

lemma *pair-spmf-return-spmf2*: *pair-spmf p* (*return-spmf y*) = *map-spmf* (λ*x*. (*x*, *y*)) *p*
⟨*proof*⟩

lemma *pair-spmf-return-spmf* [*simp*]: *pair-spmf* (*return-spmf x*) (*return-spmf y*) = *return-spmf* (*x*, *y*)
⟨*proof*⟩

lemma *rel-pair-spmf-prod*:
rel-spmf (*rel-prod A B*) (*pair-spmf p q*) (*pair-spmf p′ q′*) ⟷
rel-spmf A (*scale-spmf* (*weight-spmf q*) *p*) (*scale-spmf* (*weight-spmf q′*) *p′*) ∧
rel-spmf B (*scale-spmf* (*weight-spmf p*) *q*) (*scale-spmf* (*weight-spmf p′*) *q′*)
(**is** *?lhs* ⟷ *?rhs* **is** - ⟷ *?A* ∧ *?B* **is** - ⟷ *rel-spmf* - *?p ?p′* ∧ *rel-spmf* - *?q ?q′*)
⟨*proof*⟩

lemma *pair-pair-spmf*:
pair-spmf (*pair-spmf p q*) *r* = *map-spmf* (λ(*x*, (*y*, *z*)). ((*x*, *y*), *z*)) (*pair-spmf p* (*pair-spmf q r*))
⟨*proof*⟩

lemma *pair-commute-spmf*:
pair-spmf p q = *map-spmf* (λ(*y*, *x*). (*x*, *y*)) (*pair-spmf q p*)
⟨*proof*⟩

25.18 Assertions

definition *assert-spmf* :: *bool* ⇒ *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
⟨*proof*⟩

lemma *in-set-assert-spmf* [*simp*]: *x ∈ set-spmf (assert-spmf p) ⟷ p*
⟨*proof*⟩

lemma *set-spmf-assert-spmf-eq-empty* [*simp*]: *set-spmf (assert-spmf b) = {} ⟷ ¬ b*
⟨*proof*⟩

lemma *lossless-assert-spmf* [*iff*]: *lossless-spmf (assert-spmf b) ⟷ b*
⟨*proof*⟩

25.19 Try

definition *try-spmf* :: *'a spmf ⇒ 'a spmf ⇒ 'a spmf*
(‹(‹*open-block notation=‹mixfix try-spmf›› TRY - ELSE -*)› [*0,60*] *59*)
where *TRY p ELSE q = bind-pmf p (λx. case x of None ⇒ q | Some y ⇒ return-spmf y)*

lemma *try-spmf-lossless* [*simp*]:
assumes *lossless-spmf p*
shows *TRY p ELSE q = p*
⟨*proof*⟩

lemma *try-spmf-return-spmf1*: *TRY return-spmf x ELSE q = return-spmf x*
⟨*proof*⟩

lemma *try-spmf-return-None* [*simp*]: *TRY return-pmf None ELSE q = q*
⟨*proof*⟩

lemma *try-spmf-return-pmf-None2* [*simp*]: *TRY p ELSE return-pmf None = p*
⟨*proof*⟩

lemma *map-try-spmf*: *map-spmf f (try-spmf p q) = try-spmf (map-spmf f p) (map-spmf f q)*
⟨*proof*⟩

lemma *try-spmf-bind-pmf*: *TRY (bind-pmf p f) ELSE q = bind-pmf p (λx. TRY (f x) ELSE q)*
⟨*proof*⟩

lemma *try-spmf-bind-spmf-lossless*:
lossless-spmf p ⟹ TRY (bind-spmf p f) ELSE q = bind-spmf p (λx. TRY (f x) ELSE q)
⟨*proof*⟩

lemma *try-spmf-bind-out*:
lossless-spmf p ⟹ bind-spmf p (λx. TRY (f x) ELSE q) = TRY (bind-spmf p

f) *ELSE q*
⟨*proof*⟩

lemma *lossless-try-spmf* [*simp*]:
lossless-spmf (*TRY p ELSE q*) ⟷ *lossless-spmf p* ∨ *lossless-spmf q*
⟨*proof*⟩

context includes *lifting-syntax*
begin

lemma *try-spmf-parametric* [*transfer-rule*]:
(*rel-spmf A* ===> *rel-spmf A* ===> *rel-spmf A*) *try-spmf try-spmf*
⟨*proof*⟩

end

lemma *try-spmf-cong*:
⟦ *p* = *p*′; ¬ *lossless-spmf p*′ ⟹ *q* = *q*′ ⟧ ⟹ *TRY p ELSE q* = *TRY p*′ *ELSE q*′
⟨*proof*⟩

lemma *rel-spmf-try-spmf*:
⟦ *rel-spmf R p p*′; ¬ *lossless-spmf p*′ ⟹ *rel-spmf R q q*′ ⟧
⟹ *rel-spmf R* (*TRY p ELSE q*) (*TRY p*′ *ELSE q*′)
⟨*proof*⟩

lemma *spmf-try-spmf*:
spmf (*TRY p ELSE q*) *x* = *spmf p x* + *pmf p None* ∗ *spmf q x*
⟨*proof*⟩

lemma *try-scale-spmf-same* [*simp*]: *lossless-spmf p* ⟹ *TRY scale-spmf k p ELSE p* = *p*
⟨*proof*⟩

lemma *pmf-try-spmf-None* [*simp*]: *pmf* (*TRY p ELSE q*) *None* = *pmf p None* ∗ *pmf q None* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *try-bind-spmf-lossless2*:
lossless-spmf q ⟹ *TRY* (*bind-spmf p f*) *ELSE q* = *TRY* (*p* ⤜ (λ*x*. *TRY* (*f x*) *ELSE q*)) *ELSE q*
⟨*proof*⟩

lemma *try-bind-spmf-lossless2*′:
fixes *f* :: ′*a* ⇒ ′*b spmf* **shows**
⟦ *NO-MATCH* (λ*x* :: ′*a*. *try-spmf* (*g x* :: ′*b spmf*) (*h x*)) *f*; *lossless-spmf q* ⟧
⟹ *TRY* (*bind-spmf p f*) *ELSE q* = *TRY* (*p* ⤜ (λ*x* :: ′*a*. *TRY* (*f x*) *ELSE q*)) *ELSE q*
⟨*proof*⟩

lemma *try-bind-assert-spmf*:
TRY (*assert-spmf b* ⤜ *f*) *ELSE q* = (*if b then TRY* (*f* ()) *ELSE q else q*)
⟨*proof*⟩

25.20 Miscellaneous

lemma assumes *rel-spmf* (λ*x y*. *bad1 x* = *bad2 y* ∧ (¬ *bad2 y* ⟶ *A x* ⟷ *B y*)) *p q* (**is** *rel-spmf ?A* - -)
shows *fundamental-lemma-bad*: *measure* (*measure-spmf p*) {*x*. *bad1 x*} = *measure* (*measure-spmf q*) {*y*. *bad2 y*} (**is** *?bad*)
and *fundamental-lemma*: |*measure* (*measure-spmf p*) {*x*. *A x*} − *measure* (*measure-spmf q*) {*y*. *B y*}| ≤
measure (*measure-spmf p*) {*x*. *bad1 x*} (**is** *?fundamental*)
⟨*proof*⟩

end

26 Indexed products of PMFs

theory *Product-PMF*
imports *Probability-Mass-Function Independent-Family*
begin

Conflicting notation from *HOL−Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (**infixr** ‹*abs′-summable′-on*› *46*)

26.1 Preliminaries

lemma *pmf-expectation-eq-infsetsum*: *measure-pmf.expectation p f* = *infsetsum* (λ*x*. *pmf p x* ∗ *f x*) *UNIV*
⟨*proof*⟩

lemma *measure-pmf-prob-product*:
assumes *countable A countable B*
shows *measure-pmf.prob* (*pair-pmf M N*) (*A* × *B*) = *measure-pmf.prob M A* ∗ *measure-pmf.prob N B*
⟨*proof*⟩

26.2 Definition

In analogy to Pi_M, we define an indexed product of PMFs. In the literature, this is typically called taking a vector of independent random variables. Note that the components do not have to be identically distributed.

The operation takes an explicit index set A and a function f that maps each element from A to a PMF and defines the product measure $\bigotimes_{i \in A} f(i)$, which is represented as a (′*a* ⇒ ′*b*) *pmf*.

Note that unlike Pi_M, this only works for *finite* index sets. It could be extended to countable sets and beyond, but the construction becomes somewhat more involved.

definition *Pi-pmf* :: *'a set* ⇒ *'b* ⇒ (*'a* ⇒ *'b pmf*) ⇒ (*'a* ⇒ *'b*) *pmf* **where**
Pi-pmf A dflt p =
embed-pmf (λ*f*. *if* (∀ *x*. *x* ∉ *A* ⟶ *f x* = *dflt*) *then* ∏ *x*∈*A*. *pmf* (*p x*) (*f x*) *else 0*)

A technical subtlety that needs to be addressed is this: Intuitively, the functions in the support of a product distribution have domain A. However, since HOL is a total logic, these functions must still return *some* value for inputs outside A. The product measure Pi_M simply lets these functions return *undefined* in these cases. We chose a different solution here, which is to supply a default value *dflt* that is returned in these cases.

As one possible application, one could model the result of n different independent coin tosses as *Pi-pmf* {*0*..<*n*} *False* (λ-. *bernoulli-pmf* (*1* / *2*)). This returns a function of type *nat* ⇒ *bool* that maps every natural number below n to the result of the corresponding coin toss, and every other natural number to *False*.

lemma *pmf-Pi*:
assumes *A*: *finite A*
shows *pmf* (*Pi-pmf A dflt p*) *f* =
(*if* (∀ *x*. *x* ∉ *A* ⟶ *f x* = *dflt*) *then* ∏ *x*∈*A*. *pmf* (*p x*) (*f x*) *else 0*)
⟨*proof*⟩

lemma *Pi-pmf-cong*:
assumes *A* = *A'* *dflt* = *dflt'* ⋀*x*. *x* ∈ *A* ⟹ *f x* = *f' x*
shows *Pi-pmf A dflt f* = *Pi-pmf A' dflt' f'*
⟨*proof*⟩

lemma *pmf-Pi'*:
assumes *finite A* ⋀*x*. *x* ∉ *A* ⟹ *f x* = *dflt*
shows *pmf* (*Pi-pmf A dflt p*) *f* = (∏ *x*∈*A*. *pmf* (*p x*) (*f x*))
⟨*proof*⟩

lemma *pmf-Pi-outside*:
assumes *finite A* ∃ *x*. *x* ∉ *A* ∧ *f x* ≠ *dflt*
shows *pmf* (*Pi-pmf A dflt p*) *f* = *0*
⟨*proof*⟩

lemma *pmf-Pi-empty* [*simp*]: *Pi-pmf* {} *dflt p* = *return-pmf* (λ-. *dflt*)
⟨*proof*⟩

lemma *set-Pi-pmf-subset*: *finite A* ⟹ *set-pmf* (*Pi-pmf A dflt p*) ⊆ {*f*. ∀ *x*. *x* ∉ *A* ⟶ *f x* = *dflt*}
⟨*proof*⟩

26.3 Dependent product sets with a default

The following describes a dependent product of sets where the functions are required to return the default value *dflt* outside their domain, in analogy to Pi_E, which uses *undefined*.

definition *PiE-dflt*
  where *PiE-dflt A dflt B* = $\{f.\ \forall x.\ (x \in A \longrightarrow f\ x \in B\ x) \wedge (x \notin A \longrightarrow f\ x =$ *dflt*$)\}$

lemma *restrict-PiE-dflt*: $(\lambda h.$ *restrict h A*$)$ ‘ *PiE-dflt A dflt B* = *PiE A B*
⟨*proof*⟩

lemma *dflt-image-PiE*: $(\lambda h\ x.$ *if x* $\in$ *A then h x else dflt*$)$ ‘ *PiE A B* = *PiE-dflt A dflt B*
  (**is** *?f* ‘ *?X* = *?Y*)
⟨*proof*⟩

lemma *finite-PiE-dflt* [*intro*]:
  assumes *finite A* $\bigwedge x.$ *x* $\in$ *A* $\Longrightarrow$ *finite* (*B x*)
  shows *finite* (*PiE-dflt A d B*)
⟨*proof*⟩

lemma *card-PiE-dflt*:
  assumes *finite A* $\bigwedge x.$ *x* $\in$ *A* $\Longrightarrow$ *finite* (*B x*)
  shows *card* (*PiE-dflt A d B*) = $(\prod x \in A.$ *card* (*B x*)$)$
⟨*proof*⟩

lemma *PiE-dflt-empty-iff* [*simp*]: *PiE-dflt A dflt B* = $\{\}$ $\longleftrightarrow$ $(\exists x \in A.$ *B x* = $\{\})$
  ⟨*proof*⟩

lemma *set-Pi-pmf-subset′*:
  assumes *finite A*
  shows *set-pmf* (*Pi-pmf A dflt p*) $\subseteq$ *PiE-dflt A dflt* (*set-pmf* $\circ$ *p*)
  ⟨*proof*⟩

lemma *set-Pi-pmf*:
  assumes *finite A*
  shows *set-pmf* (*Pi-pmf A dflt p*) = *PiE-dflt A dflt* (*set-pmf* $\circ$ *p*)
⟨*proof*⟩

The probability of an independent combination of events is precisely the product of the probabilities of each individual event.

lemma *measure-Pi-pmf-PiE-dflt*:
  assumes [*simp*]: *finite A*
  shows *measure-pmf.prob* (*Pi-pmf A dflt p*) (*PiE-dflt A dflt B*) =
    $(\prod x \in A.$ *measure-pmf.prob* (*p x*) (*B x*)$)$
⟨*proof*⟩

lemma *measure-Pi-pmf-Pi*:

fixes *t*::*nat*
assumes [*simp*]: *finite A*
shows *measure-pmf.prob* (*Pi-pmf A dflt p*) (*Pi A B*) =
($\prod$ *x*∈*A*. *measure-pmf.prob* (*p x*) (*B x*)) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

26.4 Common PMF operations on products

Pi-pmf distributes over the 'bind' operation in the Giry monad:

lemma *Pi-pmf-bind*:
assumes *finite A*
shows *Pi-pmf A d* (λ*x*. *bind-pmf* (*p x*) (*q x*)) =
do {*f* ← *Pi-pmf A d'* *p*; *Pi-pmf A d* (λ*x*. *q x* (*f x*))} (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *Pi-pmf-return-pmf* [*simp*]:
assumes *finite A*
shows *Pi-pmf A dflt* (λ*x*. *return-pmf* (*f x*)) = *return-pmf* (λ*x*. *if x* ∈ *A then f x else dflt*)
⟨*proof*⟩

Analogously any componentwise mapping can be pulled outside the product:

lemma *Pi-pmf-map*:
assumes [*simp*]: *finite A* **and** *f dflt* = *dflt'*
shows *Pi-pmf A dflt'* (λ*x*. *map-pmf f* (*g x*)) = *map-pmf* (λ*h*. *f* ∘ *h*) (*Pi-pmf A dflt g*)
⟨*proof*⟩

We can exchange the default value in a product of PMFs like this:

lemma *Pi-pmf-default-swap*:
assumes *finite A*
shows *map-pmf* (λ*f x*. *if x* ∈ *A then f x else dflt'*) (*Pi-pmf A dflt p*) =
Pi-pmf A dflt' p (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

The following rule allows reindexing the product:

lemma *Pi-pmf-bij-betw*:
assumes *finite A bij-betw h A B* ⋀*x*. *x* ∉ *A* ⟹ *h x* ∉ *B*
shows *Pi-pmf A dflt* (λ-. *f*) = *map-pmf* (λ*g*. *g* ∘ *h*) (*Pi-pmf B dflt* (λ-. *f*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

A product of uniform random choices is again a uniform distribution.

lemma *Pi-pmf-of-set*:
assumes *finite A* ⋀*x*. *x* ∈ *A* ⟹ *finite* (*B x*) ⋀*x*. *x* ∈ *A* ⟹ *B x* ≠ {}
shows *Pi-pmf A d* (λ*x*. *pmf-of-set* (*B x*)) = *pmf-of-set* (*PiE-dflt A d B*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

26.5 Merging and splitting PMF products

The following lemma shows that we can add a single PMF to a product:

lemma *Pi-pmf-insert*:
assumes *finite A* $x \notin A$
shows *Pi-pmf* (*insert x A*) *dflt p* = *map-pmf* (λ(*y*,*f*). *f*(*x*:=*y*)) (*pair-pmf* (*p x*) (*Pi-pmf A dflt p*))
⟨*proof*⟩

lemma *Pi-pmf-insert'*:
assumes *finite A* $x \notin A$
shows *Pi-pmf* (*insert x A*) *dflt p* =
do {*y* ← *p x*; *f* ← *Pi-pmf A dflt p*; *return-pmf* (*f*(*x* := *y*))}
⟨*proof*⟩

lemma *Pi-pmf-singleton*:
Pi-pmf {*x*} *dflt p* = *map-pmf* (λ*a b*. *if b* = *x then a else dflt*) (*p x*)
⟨*proof*⟩

Projecting a product of PMFs onto a component yields the expected result:

lemma *Pi-pmf-component*:
assumes *finite A*
shows *map-pmf* (λ*f*. *f x*) (*Pi-pmf A dflt p*) = (*if x* ∈ *A then p x else return-pmf dflt*)
⟨*proof*⟩

We can take merge two PMF products on disjoint sets like this:

lemma *Pi-pmf-union*:
assumes *finite A finite B A* ∩ *B* = {}
shows *Pi-pmf* (*A* ∪ *B*) *dflt p* =
map-pmf (λ(*f*,*g*) *x*. *if x* ∈ *A then f x else g x*)
(*pair-pmf* (*Pi-pmf A dflt p*) (*Pi-pmf B dflt p*)) (**is** - = *map-pmf* (*?h A*) (*?q A*))
⟨*proof*⟩

We can also project a product to a subset of the indices by mapping all the other indices to the default value:

lemma *Pi-pmf-subset*:
assumes *finite A A'* ⊆ *A*
shows *Pi-pmf A' dflt p* = *map-pmf* (λ*f x*. *if x* ∈ *A' then f x else dflt*) (*Pi-pmf A dflt p*)
⟨*proof*⟩

lemma *Pi-pmf-subset'*:
fixes *f* :: *'a* ⇒ *'b pmf*
assumes *finite A B* ⊆ *A* ⋀*x*. *x* ∈ *A* − *B* ⟹ *f x* = *return-pmf dflt*
shows *Pi-pmf A dflt f* = *Pi-pmf B dflt f*
⟨*proof*⟩

lemma *Pi-pmf-if-set*:
assumes *finite A*
shows *Pi-pmf A dflt* (λx. *if b x then f x else return-pmf dflt*) =
Pi-pmf {$x \in A$. *b x*} *dflt f*
⟨*proof*⟩

lemma *Pi-pmf-if-set'*:
assumes *finite A*
shows *Pi-pmf A dflt* (λx. *if b x then return-pmf dflt else f x*) =
Pi-pmf {$x \in A$. $\neg$*b x*} *dflt f*
⟨*proof*⟩

Lastly, we can delete a single component from a product:

lemma *Pi-pmf-remove*:
assumes *finite A*
shows *Pi-pmf* ($A - \{x\}$) *dflt p* = *map-pmf* (λf. $f(x := dflt)$) (*Pi-pmf A dflt p*)
⟨*proof*⟩

26.6 Additional properties

lemma *nn-integral-prod-Pi-pmf*:
assumes *finite A*
shows *nn-integral* (*Pi-pmf A dflt p*) (λy. $\prod x \in A$. *f x* (*y x*)) = ($\prod x \in A$. *nn-integral* (*p x*) (*f x*))
⟨*proof*⟩

lemma *integrable-prod-Pi-pmf*:
fixes *f* :: $'a \Rightarrow 'b \Rightarrow 'c$ :: {*real-normed-field*, *second-countable-topology*, *banach*}
assumes *finite A* **and** $\bigwedge x$. $x \in A \Longrightarrow$ *integrable* (*measure-pmf* (*p x*)) (*f x*)
shows *integrable* (*measure-pmf* (*Pi-pmf A dflt p*)) (λh. $\prod x \in A$. *f x* (*h x*))
⟨*proof*⟩

lemma *expectation-prod-Pi-pmf*:
fixes *f* :: - ⇒ - ⇒ *real*
assumes *finite A*
assumes $\bigwedge x$. $x \in A \Longrightarrow$ *integrable* (*measure-pmf* (*p x*)) (*f x*)
assumes $\bigwedge x\ y$. $x \in A \Longrightarrow y \in$ *set-pmf* (*p x*) $\Longrightarrow$ *f x y* $\geq$ *0*
shows *measure-pmf.expectation* (*Pi-pmf A dflt p*) (λy. $\prod x \in A$. *f x* (*y x*)) =
($\prod x \in A$. *measure-pmf.expectation* (*p x*) (λv. *f x v*))
⟨*proof*⟩

lemma *indep-vars-Pi-pmf*:
assumes *fin*: *finite I*
shows *prob-space.indep-vars* (*measure-pmf* (*Pi-pmf I dflt p*))
(λ-. *count-space UNIV*) ($\lambda x\ f$. *f x*) *I*
⟨*proof*⟩

lemma
fixes $h :: 'a :: comm\text{-}monoid\text{-}add \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
assumes *fin*: *finite I*
assumes *integrable*: $\bigwedge i.\ i \in I \Longrightarrow integrable\ (measure\text{-}pmf\ (D\ i))\ h$
shows *integrable-sum-Pi-pmf*: *integrable* (*Pi-pmf I dflt D*) ($\lambda g.\ \sum i{\in}I.\ h\ (g\ i)$)
and *expectation-sum-Pi-pmf*:
measure-pmf.expectation (*Pi-pmf I dflt D*) ($\lambda g.\ \sum i{\in}I.\ h\ (g\ i)$) =
($\sum i{\in}I.$ *measure-pmf.expectation* (*D i*) *h*)
⟨*proof*⟩

26.7 Applications

Choosing a subset of a set uniformly at random is equivalent to tossing a fair coin independently for each element and collecting all the elements that came up heads.

lemma *pmf-of-set-Pow-conv-bernoulli*:
assumes *finite* (*A* :: *'a set*)
shows *map-pmf* ($\lambda b.\ \{x{\in}A.\ b\ x\}$) (*Pi-pmf A P* (λ-. *bernoulli-pmf* (*1/2*))) = *pmf-of-set* (*Pow A*)
⟨*proof*⟩

A binomial distribution can be seen as the number of successes in n independent coin tosses.

lemma *binomial-pmf-altdef'*:
fixes *A* :: *'a set*
assumes *finite A* **and** *card A* = *n* **and** *p*: $p \in \{0..1\}$
shows *binomial-pmf n p* =
map-pmf ($\lambda f.$ *card* $\{x{\in}A.\ f\ x\}$) (*Pi-pmf A dflt* (λ-. *bernoulli-pmf p*)) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

27 Hoeffding's Lemma and Hoeffding's Inequality

theory *Hoeffding*
imports *Product-PMF Independent-Family*
begin

Hoeffding's inequality shows that a sum of bounded independent random variables is concentrated around its mean, with an exponential decay of the tail probabilities.

27.1 Hoeffding's Lemma

lemma *convex-on-exp*:
fixes *l* :: *real*

assumes $l \geq 0$
shows *convex-on UNIV* $(\lambda x.\ exp(l*x))$
⟨*proof*⟩

lemma *mult-const-minus-self-real-le*:
fixes x :: *real*
shows $x * (c - x) \leq c^2\ /\ 4$
⟨*proof*⟩

lemma *Hoeffdings-lemma-aux*:
fixes $h\ p$ :: *real*
assumes $h \geq 0$ **and** $p \geq 0$
defines $L \equiv (\lambda h.\ -h * p + ln\ (1 + p * (exp\ h - 1)))$
shows $L\ h \leq h^2\ /\ 8$
⟨*proof*⟩

locale *interval-bounded-random-variable* = *prob-space* +
fixes f :: $'a \Rightarrow$ *real* **and** $a\ b$ :: *real*
assumes *random-variable* [*measurable*]: *random-variable borel f*
assumes *AE-in-interval*: *AE x in M. f x* $\in \{a..b\}$
begin

lemma *integrable* [*intro*]: *integrable M f*
⟨*proof*⟩

We first show Hoeffding's lemma for distributions whose expectation is 0. The general case will easily follow from this later.

lemma *Hoeffdings-lemma-nn-integral-0*:
assumes $l > 0$ **and** *E0*: *expectation f* $= 0$
shows *nn-integral M* $(\lambda x.\ exp\ (l * f\ x)) \leq ennreal\ (exp\ (l^2 * (b - a)^2\ /\ 8))$
⟨*proof*⟩

context
begin

interpretation *shift*: *interval-bounded-random-variable M* $\lambda x.\ f\ x - \mu\ a - \mu\ b - \mu$
rewrites $b - \mu - (a - \mu) \equiv b - a$
⟨*proof*⟩

lemma *expectation-shift*: *expectation* $(\lambda x.\ f\ x -$ *expectation f*$) = 0$
⟨*proof*⟩

lemmas *Hoeffdings-lemma-nn-integral* = *shift.Hoeffdings-lemma-nn-integral-0*[*OF - expectation-shift*]

end

end

27.2 Hoeffding's Inequality

Consider n independent real random variables $X_1, \ldots, X_n$ that each almost surely lie in a compact interval $[a_i, b_i]$. Hoeffding's inequality states that the distribution of the sum of the X_i is tightly concentrated around the sum of the expected values: the probability of it being above or below the sum of the expected values by more than some ε decreases exponentially with ε.

locale *indep-interval-bounded-random-variables* = *prob-space* +
  fixes I :: $'b$ *set* **and** X :: $'b \Rightarrow 'a \Rightarrow$ *real*
  fixes a b :: $'b \Rightarrow$ *real*
  assumes *fin*: *finite* I
  assumes *indep*: *indep-vars* (λ-. *borel*) X I
  assumes *AE-in-interval*: $\bigwedge i.\ i \in I \Longrightarrow$ *AE* x *in* $M.\ X\ i\ x \in \{a\ i..b\ i\}$
begin

lemma *random-variable* [*measurable*]:
  assumes *i*: $i \in I$
  shows *random-variable borel* (X i)
  ⟨*proof*⟩

lemma *bounded-random-variable* [*intro*]:
  assumes *i*: $i \in I$
  shows *interval-bounded-random-variable* M (X i) (a i) (b i)
  ⟨*proof*⟩

end

locale *Hoeffding-ineq* = *indep-interval-bounded-random-variables* +
  fixes μ :: *real*
  defines $\mu \equiv (\sum i \in I.$ *expectation* (X i))
begin

theorem *Hoeffding-ineq-ge*:
  assumes $\varepsilon \geq 0$
  assumes $(\sum i \in I.\ (b\ i - a\ i)^2) > 0$
  shows *prob* $\{x \in$ *space* $M.\ (\sum i \in I.\ X\ i\ x) \geq \mu + \varepsilon\} \leq$ *exp* $(-2 * \varepsilon^2$ / $(\sum i \in I.$ $(b\ i - a\ i)^2))$
proof (*cases* $\varepsilon = 0$)
  case [*simp*]: *True*
  have *prob* $\{x \in$ *space* $M.\ (\sum i \in I.\ X\ i\ x) \geq \mu + \varepsilon\} \leq 1$
    by *simp*
  thus *?thesis* **by** *simp*
next
  case *False*
  with ‹$\varepsilon \geq 0$› **have** ε: $\varepsilon > 0$

by *auto*

define d **where** $d = (\sum i{\in}I.\ (b\ i - a\ i)^2)$
define $l :: real$ **where** $l = 4 * \varepsilon\ /\ d$
have d: $d > 0$
using *assms* **by** (*simp add*: *d-def*)
have l: $l > 0$
using ε d **by** (*simp add*: *l-def*)
define μ' **where** $\mu' = (\lambda i.\ expectation\ (X\ i))$

have $\{x{\in}space\ M.\ (\sum i{\in}I.\ X\ i\ x) \geq \mu + \varepsilon\} = \{x{\in}space\ M.\ (\sum i{\in}I.\ X\ i\ x) - \mu \geq \varepsilon\}$
by (*simp add*: *algebra-simps*)
hence $ennreal\ (prob\ \{x{\in}space\ M.\ (\sum i{\in}I.\ X\ i\ x) \geq \mu + \varepsilon\}) = emeasure\ M\ \ldots$
by (*simp add*: *emeasure-eq-measure*)
also have $\ldots \leq ennreal\ (exp\ (-l{*}\varepsilon)) * (\int^+ x{\in}space\ M.\ exp\ (l * ((\sum i{\in}I.\ X\ i\ x) - \mu))\ \partial M)$
by (*intro Chernoff-ineq-nn-integral-ge l*) *auto*
also have $(\lambda x.\ (\sum i{\in}I.\ X\ i\ x) - \mu) = (\lambda x.\ (\sum i{\in}I.\ X\ i\ x - \mu'\ i))$
by (*simp add*: *μ-def sum-subtractf μ'-def*)
also have $(\int^+ x{\in}space\ M.\ exp\ (l * ((\sum i{\in}I.\ X\ i\ x - \mu'\ i)))\ \partial M) =$
$(\int^+ x.\ (\prod i{\in}I.\ ennreal\ (exp\ (l * (X\ i\ x - \mu'\ i))))\ \partial M)$
by (*intro nn-integral-cong*)
(*simp-all add*: *sum-distrib-left ring-distribs exp-diff exp-sum fin prod-ennreal*)
also have $\ldots = (\prod i{\in}I.\ \int^+ x.\ ennreal\ (exp\ (l * (X\ i\ x - \mu'\ i)))\ \partial M)$
by (*intro indep-vars-nn-integral fin indep-vars-compose2*[*OF indep*]) *auto*
also have $ennreal\ (exp\ (-l * \varepsilon)) * \ldots \leq$
$ennreal\ (exp\ (-l * \varepsilon)) * (\prod i{\in}I.\ ennreal\ (exp\ (l^2 * (b\ i - a\ i)^2\ /\ 8)))$
proof (*intro mult-left-mono prod-mono-ennreal*)
fix i **assume** i: $i \in I$
from i **interpret** *interval-bounded-random-variable M X i a i b i* **..**
show $(\int^+ x.\ ennreal\ (exp\ (l * (X\ i\ x - \mu'\ i)))\ \partial M) \leq ennreal\ (exp\ (l^2 * (b\ i - a\ i)^2\ /\ 8))$
unfolding *μ'-def* **by** (*rule Hoeffdings-lemma-nn-integral*) *fact+*
qed *auto*
also have $\ldots = ennreal\ (exp\ (-l{*}\varepsilon) * (\prod i{\in}I.\ exp\ (l^2 * (b\ i - a\ i)^2\ /\ 8)))$
by (*simp add*: *prod-ennreal prod-nonneg flip*: *ennreal-mult*)
also have $exp\ (-l{*}\varepsilon) * (\prod i{\in}I.\ exp\ (l^2 * (b\ i - a\ i)^2\ /\ 8)) = exp\ (d * l^2\ /\ 8 - l * \varepsilon)$
by (*simp add*: *exp-diff exp-minus sum-divide-distrib sum-distrib-left*
sum-distrib-right exp-sum fin divide-simps mult-ac d-def)
also have $d * l^2\ /\ 8 - l * \varepsilon = -2 * \varepsilon^2\ /\ d$
using d **by** (*simp add*: *l-def field-simps power2-eq-square*)
finally show *?thesis*
by (*subst* (*asm*) *ennreal-le-iff*) (*simp-all add*: *d-def*)
qed

corollary *Hoeffding-ineq-le*:
assumes ε: $\varepsilon \geq 0$

assumes $(\sum i{\in}I.\ (b\ i - a\ i)^2) > 0$
shows *prob* $\{x{\in}space\ M.\ (\sum i{\in}I.\ X\ i\ x) \leq \mu - \varepsilon\} \leq exp\ (-2 * \varepsilon^2\ /\ (\sum i{\in}I.\ (b\ i - a\ i)^2))$
⟨*proof*⟩

corollary *Hoeffding-ineq-abs-ge*:
assumes ε: $\varepsilon \geq 0$
assumes $(\sum i{\in}I.\ (b\ i - a\ i)^2) > 0$
shows *prob* $\{x{\in}space\ M.\ |(\sum i{\in}I.\ X\ i\ x) - \mu| \geq \varepsilon\} \leq 2 * exp\ (-2 * \varepsilon^2\ /\ (\sum i{\in}I.\ (b\ i - a\ i)^2))$
⟨*proof*⟩

end

27.3 Hoeffding's inequality for i.i.d. bounded random variables

If we have n even identically-distributed random variables, the statement of Hoeffding's lemma simplifies a bit more: it shows that the probability that the average of the X_i is more than ε above the expected value is no greater than $e^{\frac{-2ny^2}{(b-a)^2}}$.

This essentially gives us a more concrete version of the weak law of large numbers: the law states that the probability vanishes for $n \to \infty$ for any $\varepsilon > 0$. Unlike Hoeffding's inequality, it does not assume the variables to have bounded support, but it does not provide concrete bounds.

locale *iid-interval-bounded-random-variables* = *prob-space* +
fixes I :: *'b set* **and** X :: *'b* ⇒ *'a* ⇒ *real* **and** Y :: *'a* ⇒ *real*
fixes $a\ b$ :: *real*
assumes *fin*: *finite I*
assumes *indep*: *indep-vars* (λ-. *borel*) *X I*
assumes *distr-X*: $i \in I \Longrightarrow$ *distr M borel* $(X\ i) =$ *distr M borel Y*
assumes *rv-Y* [*measurable*]: *random-variable borel Y*
assumes *AE-in-interval*: *AE x in M.* $Y\ x \in \{a..b\}$
begin

lemma *random-variable* [*measurable*]:
assumes *i*: $i \in I$
shows *random-variable borel* $(X\ i)$
⟨*proof*⟩

sublocale *X*: *indep-interval-bounded-random-variables M I X* λ-. *a* λ-. *b*
⟨*proof*⟩

lemma *expectation-X* [*simp*]:
assumes *i*: $i \in I$
shows *expectation* $(X\ i) =$ *expectation Y*
⟨*proof*⟩

end

locale *Hoeffding-ineq-iid* = *iid-interval-bounded-random-variables* +
  fixes μ :: *real*
  defines $\mu \equiv$ *expectation Y*
begin

sublocale *X*: *Hoeffding-ineq M I X* λ-. *a* λ-. *b real* (*card I*) $*$ μ
  ⟨*proof*⟩

corollary
  assumes ε: $\varepsilon \geq 0$
  assumes $a < b$ $I \neq \{\}$
  defines $n \equiv$ *card I*
  shows *Hoeffding-ineq-ge*:
        prob $\{x \in$ *space M*. $(\sum i \in I.\ X\ i\ x) \geq n * \mu + \varepsilon\} \leq$
          exp $(-2 * \varepsilon^2 \ /\ (n * (b - a)^2))$ (**is** *?le*)
    and *Hoeffding-ineq-le*:
        prob $\{x \in$ *space M*. $(\sum i \in I.\ X\ i\ x) \leq n * \mu - \varepsilon\} \leq$
          exp $(-2 * \varepsilon^2 \ /\ (n * (b - a)^2))$ (**is** *?ge*)
    and *Hoeffding-ineq-abs-ge*:
        prob $\{x \in$ *space M*. $|(\sum i \in I.\ X\ i\ x) - n * \mu| \geq \varepsilon\} \leq$
          $2 *$ *exp* $(-2 * \varepsilon^2 \ /\ (n * (b - a)^2))$ (**is** *?abs-ge*)
⟨*proof*⟩

lemma
  assumes ε: $\varepsilon \geq 0$
  assumes $a < b$ $I \neq \{\}$
  defines $n \equiv$ *card I*
  shows *Hoeffding-ineq-ge′*:
        prob $\{x \in$ *space M*. $(\sum i \in I.\ X\ i\ x)\ /\ n \geq \mu + \varepsilon\} \leq$
          exp $(-2 * n * \varepsilon^2 \ /\ (b - a)^2)$ (**is** *?ge*)
    and *Hoeffding-ineq-le′*:
        prob $\{x \in$ *space M*. $(\sum i \in I.\ X\ i\ x)\ /\ n \leq \mu - \varepsilon\} \leq$
          exp $(-2 * n * \varepsilon^2 \ /\ (b - a)^2)$ (**is** *?le*)
    and *Hoeffding-ineq-abs-ge′*:
        prob $\{x \in$ *space M*. $|(\sum i \in I.\ X\ i\ x)\ /\ n - \mu| \geq \varepsilon\} \leq$
          $2 *$ *exp* $(-2 * n * \varepsilon^2 \ /\ (b - a)^2)$ (**is** *?abs-ge*)
⟨*proof*⟩

end

27.4 Hoeffding's Inequality for the Binomial distribution

We can now apply Hoeffding's inequality to the Binomial distribution, which can be seen as the sum of n i.i.d. coin flips (the support of each of which is contained in $[0, 1]$).

locale *binomial-distribution* =
  fixes n :: *nat* **and** p :: *real*
  assumes *p*: $p \in \{0..1\}$
begin

context
  fixes *coins* :: $(nat \Rightarrow bool)$ *pmf* **and** μ
  assumes *n*: $n > 0$
  defines *coins* $\equiv$ *Pi-pmf* $\{..<n\}$ *False* $(\lambda$-. *bernoulli-pmf p*$)$
begin

lemma *coins-component*:
  assumes *i*: $i < n$
  shows *distr* (*measure-pmf coins*) *borel* $(\lambda f.$ *if f i then 1 else 0*$)$ =
    distr (*measure-pmf* (*bernoulli-pmf p*)) *borel* $(\lambda b.$ *if b then 1 else 0*$)$
⟨*proof*⟩

lemma *prob-binomial-pmf-conv-coins*:
  measure-pmf.prob (*binomial-pmf n p*) $\{x.$ *P* (*real x*)$\}$ =
   measure-pmf.prob coins $\{x.$ *P* $(\sum i<n.$ *if x i then 1 else 0*$)\}$
⟨*proof*⟩

interpretation *Hoeffding-ineq-iid*
  coins $\{..<n\}$ $\lambda i\ f.$ *if f i then 1 else 0* $\lambda f.$ *if f 0 then 1 else 0 0 1 p*
⟨*proof*⟩

corollary
  fixes ε :: *real*
  assumes ε: $\varepsilon \geq 0$
  shows *prob-ge*: *measure-pmf.prob* (*binomial-pmf n p*) $\{x.\ x \geq n * p + \varepsilon\} \leq exp$ $(-2 * \varepsilon^2 / n)$
    and *prob-le*: *measure-pmf.prob* (*binomial-pmf n p*) $\{x.\ x \leq n * p - \varepsilon\} \leq exp$ $(-2 * \varepsilon^2 / n)$
    and *prob-abs-ge*:
      measure-pmf.prob (*binomial-pmf n p*) $\{x.\ |x - n * p| \geq \varepsilon\} \leq 2 * exp\ (-2$ $* \varepsilon^2 / n)$
⟨*proof*⟩

corollary
  fixes ε :: *real*
  assumes ε: $\varepsilon \geq 0$
  shows *prob-ge′*: *measure-pmf.prob* (*binomial-pmf n p*) $\{x.\ x / n \geq p + \varepsilon\} \leq exp$ $(-2 * n * \varepsilon^2)$
    and *prob-le′*: *measure-pmf.prob* (*binomial-pmf n p*) $\{x.\ x / n \leq p - \varepsilon\} \leq exp$ $(-2 * n * \varepsilon^2)$
    and *prob-abs-ge′*:
      measure-pmf.prob (*binomial-pmf n p*) $\{x.\ |x / n - p| \geq \varepsilon\} \leq 2 * exp\ (-2$ $* n * \varepsilon^2)$
⟨*proof*⟩

end

end

27.5 Tail bounds for the negative binomial distribution

Since the tail probabilities of a negative Binomial distribution are equal to the tail probabilities of some Binomial distribution, we can obtain tail bounds for the negative Binomial distribution through the Hoeffding tail bounds for the Binomial distribution.

context
  fixes p q :: *real*
  assumes *p*: $p \in \{0<..<1\}$
  defines $q \equiv 1 - p$
begin

lemma *prob-neg-binomial-pmf-ge-bound*:
  fixes n :: *nat* **and** k :: *real*
  defines $\mu \equiv real\ n * q\ /\ p$
  assumes *k*: $k \geq 0$
  shows *measure-pmf.prob* (*neg-binomial-pmf n p*) $\{x.\ real\ x \geq \mu + k\}$
      $\leq exp\ (-\ 2 * p\ \hat{}\ 3 * k^2\ /\ (n + p * k))$
⟨*proof*⟩

lemma *prob-neg-binomial-pmf-le-bound*:
  fixes n :: *nat* **and** k :: *real*
  defines $\mu \equiv real\ n * q\ /\ p$
  assumes *k*: $k \geq 0$
  shows *measure-pmf.prob* (*neg-binomial-pmf n p*) $\{x.\ real\ x \leq \mu - k\}$
      $\leq exp\ (-2 * p\ \hat{}\ 3 * k^2\ /\ (n - p * k))$
⟨*proof*⟩

Due to the function $exp(-l/x)$ being concave for $x \geq \frac{l}{2}$, the above two bounds can be combined into the following one for moderate values of k. (cf. https://math.stackexchange.com/questions/1565559)

lemma *prob-neg-binomial-pmf-abs-ge-bound*:
  fixes n :: *nat* **and** k :: *real*
  defines $\mu \equiv real\ n * q\ /\ p$
  assumes $k \geq 0$ **and** *n-ge*: $n \geq p * k * (p^2 * k + 1)$
  shows *measure-pmf.prob* (*neg-binomial-pmf n p*) $\{x.\ |real\ x - \mu| \geq k\} \leq$
      $2 * exp\ (-2 * p\ \hat{}\ 3 * k\ \hat{}\ 2\ /\ n)$
⟨*proof*⟩

end

end

theory *Stream-Space*
imports
Infinite-Product-Measure
HOL−Library.Stream
HOL−Library.Linear-Temporal-Logic-on-Streams
begin

lemma *stream-eq-Stream-iff*: $s = x \mathbin{\#\#} t \longleftrightarrow (shd\ s = x \wedge stl\ s = t)$
⟨*proof*⟩

lemma *Stream-snth*: $(x \mathbin{\#\#} s) !! n = (case\ n\ of\ 0 \Rightarrow x \mid Suc\ n \Rightarrow s !! n)$
⟨*proof*⟩

definition *to-stream* :: $(nat \Rightarrow 'a) \Rightarrow 'a\ stream$ **where**
to-stream $X = smap\ X\ nats$

lemma *to-stream-nat-case*: $to\text{-}stream\ (case\text{-}nat\ x\ X) = x \mathbin{\#\#} to\text{-}stream\ X$
⟨*proof*⟩

lemma *to-stream-in-streams*: $to\text{-}stream\ X \in streams\ S \longleftrightarrow (\forall n.\ X\ n \in S)$
⟨*proof*⟩

definition *stream-space* :: $'a\ measure \Rightarrow 'a\ stream\ measure$ **where**
stream-space $M =$
distr $(\Pi_M\ i \in UNIV.\ M)$ $(vimage\text{-}algebra\ (streams\ (space\ M))\ snth\ (\Pi_M\ i \in UNIV.\ M))\ to\text{-}stream$

lemma *space-stream-space*: $space\ (stream\text{-}space\ M) = streams\ (space\ M)$
⟨*proof*⟩

lemma *streams-stream-space*[*intro*]: $streams\ (space\ M) \in sets\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *stream-space-Stream*:
$x \mathbin{\#\#} \omega \in space\ (stream\text{-}space\ M) \longleftrightarrow x \in space\ M \wedge \omega \in space\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *stream-space-eq-distr*: $stream\text{-}space\ M = distr\ (\Pi_M\ i \in UNIV.\ M)\ (stream\text{-}space\ M)\ to\text{-}stream$
⟨*proof*⟩

lemma *sets-stream-space-cong*[*measurable-cong*]:
$sets\ M = sets\ N \Longrightarrow sets\ (stream\text{-}space\ M) = sets\ (stream\text{-}space\ N)$
⟨*proof*⟩

lemma *measurable-snth-PiM*: $(\lambda\omega\ n.\ \omega !! n) \in measurable\ (stream\text{-}space\ M)\ (\Pi_M\ i \in UNIV.\ M)$

⟨*proof*⟩

lemma *measurable-snth*[*measurable*]: $(\lambda\omega.\ \omega\ !!\ n) \in$ *measurable* (*stream-space M*) *M*
⟨*proof*⟩

lemma *measurable-shd*[*measurable*]: *shd* ∈ *measurable* (*stream-space M*) *M*
⟨*proof*⟩

lemma *measurable-stream-space2*:
assumes *f-snth*: $\bigwedge n.\ (\lambda x.\ f\ x\ !!\ n) \in$ *measurable N M*
shows *f* ∈ *measurable N* (*stream-space M*)
⟨*proof*⟩

lemma *measurable-stream-coinduct*[*consumes 1*, *case-names shd stl*, *coinduct set*: *measurable*]:
assumes *F f*
assumes *h*: $\bigwedge f.\ F\ f \Longrightarrow (\lambda x.\ shd\ (f\ x)) \in$ *measurable N M*
assumes *t*: $\bigwedge f.\ F\ f \Longrightarrow F\ (\lambda x.\ stl\ (f\ x))$
shows *f* ∈ *measurable N* (*stream-space M*)
⟨*proof*⟩

lemma *measurable-sdrop*[*measurable*]: *sdrop n* ∈ *measurable* (*stream-space M*) (*stream-space M*)
⟨*proof*⟩

lemma *measurable-stl*[*measurable*]: $(\lambda\omega.\ stl\ \omega) \in$ *measurable* (*stream-space M*) (*stream-space M*)
⟨*proof*⟩

lemma *measurable-to-stream*[*measurable*]: *to-stream* ∈ *measurable* (Π_M *i*∈*UNIV*. *M*) (*stream-space M*)
⟨*proof*⟩

lemma *measurable-Stream*[*measurable* (*raw*)]:
assumes *f*[*measurable*]: *f* ∈ *measurable N M*
assumes *g*[*measurable*]: *g* ∈ *measurable N* (*stream-space M*)
shows $(\lambda x.\ f\ x\ \#\#\ g\ x) \in$ *measurable N* (*stream-space M*)
⟨*proof*⟩

lemma *measurable-smap*[*measurable*]:
assumes *X*[*measurable*]: *X* ∈ *measurable N M*
shows *smap X* ∈ *measurable* (*stream-space N*) (*stream-space M*)
⟨*proof*⟩

lemma *measurable-stake*[*measurable*]:
stake i ∈ *measurable* (*stream-space* (*count-space UNIV*)) (*count-space* (*UNIV* :: *'a*::*countable list set*))
⟨*proof*⟩

lemma *measurable-shift*[*measurable*]:
assumes *f*: $f \in$ *measurable N* (*stream-space M*)
assumes [*measurable*]: $g \in$ *measurable N* (*stream-space M*)
shows ($\lambda x.$ *stake n* (*f x*) @− *g x*) ∈ *measurable N* (*stream-space M*)
⟨*proof*⟩

lemma *measurable-case-stream-replace*[*measurable* (*raw*)]:
($\lambda x.$ *f x* (*shd* (*g x*)) (*stl* (*g x*))) ∈ *measurable M N* ⟹ ($\lambda x.$ *case-stream* (*f x*) (*g x*)) ∈ *measurable M N*
⟨*proof*⟩

lemma *measurable-ev-at*[*measurable*]:
assumes [*measurable*]: *Measurable.pred* (*stream-space M*) *P*
shows *Measurable.pred* (*stream-space M*) (*ev-at P n*)
⟨*proof*⟩

lemma *measurable-alw*[*measurable*]:
Measurable.pred (*stream-space M*) *P* ⟹ *Measurable.pred* (*stream-space M*) (*alw P*)
⟨*proof*⟩

lemma *measurable-ev*[*measurable*]:
Measurable.pred (*stream-space M*) *P* ⟹ *Measurable.pred* (*stream-space M*) (*ev P*)
⟨*proof*⟩

lemma *measurable-until*:
assumes [*measurable*]: *Measurable.pred* (*stream-space M*) φ *Measurable.pred* (*stream-space M*) ψ
shows *Measurable.pred* (*stream-space M*) (φ *until* ψ)
⟨*proof*⟩

lemma *measurable-holds* [*measurable*]: *Measurable.pred M P* ⟹ *Measurable.pred* (*stream-space M*) (*holds P*)
⟨*proof*⟩

lemma *measurable-hld*[*measurable*]: **assumes** [*measurable*]: *t* ∈ *sets M* **shows** *Measurable.pred* (*stream-space M*) (*HLD t*)
⟨*proof*⟩

lemma *measurable-nxt*[*measurable* (*raw*)]:
Measurable.pred (*stream-space M*) *P* ⟹ *Measurable.pred* (*stream-space M*) (*nxt P*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *measurable-szip*:
(λ(*ω1*, *ω2*). *szip ω1 ω2*) ∈ *measurable* (*stream-space M* ⨂$_M$ *stream-space N*) (*stream-space* (*M* ⨂$_M$ *N*))
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-space-stream-space*: *prob-space* (*stream-space M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *nn-integral-stream-space*:
assumes [*measurable*]: *f* ∈ *borel-measurable* (*stream-space M*)
shows (∫$^+$*X*. *f X* ∂*stream-space M*) = (∫$^+$*x*. (∫$^+$*X*. *f* (*x* ## *X*) ∂*stream-space M*) ∂*M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *emeasure-stream-space*:
assumes *X*[*measurable*]: *X* ∈ *sets* (*stream-space M*)
shows *emeasure* (*stream-space M*) *X* = (∫$^+$*t*. *emeasure* (*stream-space M*) {*x*∈*space* (*stream-space M*). *t* ## *x* ∈ *X* } ∂*M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-stream-space*:
assumes *P*[*measurable*]: {*x*∈*space* (*stream-space M*). *P x*} ∈ *sets* (*stream-space M*)
shows *P*(*x in stream-space M*. *P x*) = (∫$^+$*t*. *P*(*x in stream-space M*. *P* (*t* ## *x*)) ∂*M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *AE-stream-space*:
assumes [*measurable*]: *Measurable.pred* (*stream-space M*) *P*
shows (*AE X in stream-space M*. *P X*) = (*AE x in M*. *AE X in stream-space M*. *P* (*x* ## *X*))
⟨*proof*⟩

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*⟩

lemma *streams-sets*:
assumes *X*[*measurable*]: *X* ∈ *sets M* **shows** *streams X* ∈ *sets* (*stream-space M*)
⟨*proof*⟩

lemma *sets-stream-space-in-sets*:
assumes *space*: *space N* = *streams* (*space M*)
assumes *sets*: ⋀*i*. (λ*x*. *x* !! *i*) ∈ *measurable N M*
shows *sets* (*stream-space M*) ⊆ *sets N*

⟨*proof*⟩

lemma *sets-stream-space-eq*: *sets* (*stream-space M*) =
sets (*SUP i∈UNIV*. *vimage-algebra* (*streams* (*space M*)) (λ*s*. *s* !! *i*) *M*)
⟨*proof*⟩

lemma *sets-restrict-stream-space*:
assumes *S*[*measurable*]: *S* ∈ *sets M*
shows *sets* (*restrict-space* (*stream-space M*) (*streams S*)) = *sets* (*stream-space* (*restrict-space M S*))
⟨*proof*⟩

primrec *sstart* :: *'a set* ⇒ *'a list* ⇒ *'a stream set* **where**
sstart S [] = *streams S*
| [*simp del*]: *sstart S* (*x* # *xs*) = (##) *x* ‘ *sstart S xs*

lemma *in-sstart*[*simp*]: *s* ∈ *sstart S* (*x* # *xs*) ⟷ *shd s* = *x* ∧ *stl s* ∈ *sstart S xs*
⟨*proof*⟩

lemma *sstart-in-streams*: *xs* ∈ *lists S* ⟹ *sstart S xs* ⊆ *streams S*
⟨*proof*⟩

lemma *sstart-eq*: *x* ∈ *streams S* ⟹ *x* ∈ *sstart S xs* = (∀ *i*<*length xs*. *x* !! *i* = *xs* ! *i*)
⟨*proof*⟩

lemma *sstart-sets*: *sstart S xs* ∈ *sets* (*stream-space* (*count-space UNIV*))
⟨*proof*⟩

lemma *sigma-sets-singletons*:
assumes *countable S*
shows *sigma-sets S* ((λ*s*. {*s*})‘*S*) = *Pow S*
⟨*proof*⟩

lemma *sets-count-space-eq-sigma*:
countable S ⟹ *sets* (*count-space S*) = *sets* (*sigma S* ((λ*s*. {*s*})‘*S*))
⟨*proof*⟩

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*⟩

lemma *Int-stable-sstart*: *Int-stable* (*sstart S*‘*lists S* ∪ {{}})
⟨*proof*⟩

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 ∈ streams S AE x in N. x ∈ 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 ∗: *⋀xs. xs ≠ [] ⟹ xs ∈ lists S ⟹ emeasure M (sstart S xs) = emeasure N (sstart S xs)*
shows *M = N*
⟨*proof*⟩

lemma *sets-sstart*[*measurable*]: *sstart Ω xs ∈ sets (stream-space (count-space UNIV))*
⟨*proof*⟩

primrec *scylinder* :: *'a set ⇒ 'a set list ⇒ 'a stream set*
where
scylinder S [] = streams S
| *scylinder S (A # As) = {ω∈streams S. shd ω ∈ A ∧ stl ω ∈ scylinder S As}*

lemma *scylinder-streams*: *scylinder S xs ⊆ streams S*
⟨*proof*⟩

lemma *sets-scylinder*: *(∀ x∈set xs. x ∈ sets S) ⟹ scylinder (space S) xs ∈ sets (stream-space S)*
⟨*proof*⟩

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 ∗: *⋀xs. xs ≠ [] ⟹ xs ∈ lists G ⟹ emeasure M (scylinder (space S) xs) = emeasure N (scylinder (space S) xs)*
shows *M = N*
⟨*proof*⟩

lemma *stream-space-coinduct*:
fixes *R* :: *'a stream measure ⇒ 'a stream measure ⇒ bool*
assumes *R A B*
assumes *R*: *⋀A B. R A B ⟹ ∃ K∈space (prob-algebra M). ∃ A'∈M →M prob-algebra (stream-space M). ∃ B'∈M →M prob-algebra (stream-space M).*
(AE y in K. R (A' y) (B' y) ∨ A' y = B' y) ∧
A = do { y ← K; ω ← A' y; return (stream-space M) (y ## ω) } ∧
B = do { y ← K; ω ← B' y; return (stream-space M) (y ## ω) }
shows *A = B*
⟨*proof*⟩

end

theory *Tree-Space*
imports *HOL−Analysis.Analysis HOL−Library.Tree*
begin

lemma *countable-lfp*:
assumes *step*: $\bigwedge Y.$ *countable* $Y \Longrightarrow$ *countable* $(F\ Y)$
and *cont*: *Order-Continuity.sup-continuous* F
shows *countable* $(lfp\ F)$
⟨*proof*⟩

lemma *countable-lfp-apply*:
assumes *step*: $\bigwedge Y\ x.\ (\bigwedge x.$ *countable* $(Y\ x)) \Longrightarrow$ *countable* $(F\ Y\ x)$
and *cont*: *Order-Continuity.sup-continuous* F
shows *countable* $(lfp\ F\ x)$
⟨*proof*⟩

inductive-set *trees* :: $'a$ *set* $\Rightarrow$ $'a$ *tree set* **for** S :: $'a$ *set* **where**
[*intro*!]: *Leaf* $\in$ *trees* S
| $l \in$ *trees* $S \Longrightarrow r \in$ *trees* $S \Longrightarrow v \in S \Longrightarrow$ *Node* $l\ v\ r \in$ *trees* S

lemma *Node-in-trees-iff*[*simp*]: *Node* $l\ v\ r \in$ *trees* $S \longleftrightarrow (l \in$ *trees* $S \wedge v \in S \wedge$ $r \in$ *trees* $S)$
⟨*proof*⟩

lemma *trees-sub-lfp*: *trees* $S \subseteq$ *lfp* $(\lambda T.\ T \cup \{Leaf\} \cup (\bigcup l{\in}T.\ (\bigcup v{\in}S.\ (\bigcup r{\in}T.$ $\{Node\ l\ v\ r\}))))$
⟨*proof*⟩

lemma *countable-trees*: *countable* $A \Longrightarrow$ *countable* $(trees\ A)$
⟨*proof*⟩

lemma *trees-UNIV*[*simp*]: *trees UNIV* $=$ *UNIV*
⟨*proof*⟩

instance *tree* :: $(countable)$ *countable*
⟨*proof*⟩

lemma *map-in-trees*[*intro*]: $(\bigwedge x.\ x \in$ *set-tree* $t \Longrightarrow f\ x \in S) \Longrightarrow$ *map-tree* $f\ t \in$ *trees* S
⟨*proof*⟩

primrec *trees-cyl* :: $'a$ *set tree* $\Rightarrow$ $'a$ *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′ ∈ trees-cyl t ⟷*
(∃ l v r. t = Node l v r ∧ l′ ∈ trees-cyl l ∧ r′ ∈ trees-cyl r ∧ v′ ∈ v)
⟨*proof*⟩

lemma *trees-cyl-sub-trees*:
assumes *t ∈ trees A A ⊆ Pow B* **shows** *trees-cyl t ⊆ trees B*
⟨*proof*⟩

lemma *trees-cyl-sets-in-space*: *trees-cyl ‘ trees (sets M) ⊆ Pow (trees (space M))*
⟨*proof*⟩

lemma *space-tree-sigma*: *space (tree-sigma M) = trees (space M)*
⟨*proof*⟩

lemma *sets-tree-sigma-eq*: *sets (tree-sigma M) = sigma-sets (trees (space M)) (trees-cyl ‘ trees (sets M))*
⟨*proof*⟩

lemma *Leaf-in-space-tree-sigma* [*measurable*, *simp*, *intro*]: *Leaf ∈ space (tree-sigma M)*
⟨*proof*⟩

lemma *Leaf-in-tree-sigma* [*measurable*, *simp*, *intro*]: *{Leaf} ∈ sets (tree-sigma M)*
⟨*proof*⟩

lemma *trees-cyl-map-treeI*: *t ∈ trees-cyl (map-tree (λx. A) t)* **if** *∗*: *t ∈ trees A*
⟨*proof*⟩

lemma *trees-cyl-map-in-sets*:
(⋀x. x ∈ set-tree t ⟹ f x ∈ sets M) ⟹ trees-cyl (map-tree f t) ∈ sets (tree-sigma M)
⟨*proof*⟩

lemma *Node-in-tree-sigma*:
assumes *L*: *X ∈ sets (M ⨂M (tree-sigma M ⨂M tree-sigma M))*
shows *{Node l v r | l v r. (v, l, r) ∈ X} ∈ sets (tree-sigma M)*
⟨*proof*⟩

lemma *measurable-left*[*measurable*]: *left ∈ tree-sigma M →M tree-sigma M*
⟨*proof*⟩

lemma *measurable-right*[*measurable*]: *right ∈ tree-sigma M →M tree-sigma M*
⟨*proof*⟩

lemma *measurable-value′*: *value ∈ restrict-space (tree-sigma M) (−{Leaf}) →M M*
⟨*proof*⟩

lemma *measurable-value*[*measurable* (*raw*)]:
  assumes $f \in X \rightarrow_M$ *tree-sigma* M
    and $\bigwedge x.\ x \in$ *space* $X \Longrightarrow f\ x \neq$ *Leaf*
  shows $(\lambda\omega.$ *value* $(f\ \omega)) \in X \rightarrow_M M$
⟨*proof*⟩

lemma *measurable-Node* [*measurable*]:
  $(\lambda(l,x,r).$ *Node* $l\ x\ r) \in$ *tree-sigma* $M \bigotimes_M M \bigotimes_M$ *tree-sigma* $M \rightarrow_M$ *tree-sigma* M
⟨*proof*⟩

lemma *measurable-Node'* [*measurable* (*raw*)]:
  assumes [*measurable*]: $l \in B \rightarrow_M$ *tree-sigma* A
  assumes [*measurable*]: $x \in B \rightarrow_M A$
  assumes [*measurable*]: $r \in B \rightarrow_M$ *tree-sigma* A
  shows $(\lambda y.$ *Node* $(l\ y)\ (x\ y)\ (r\ y)) \in B \rightarrow_M$ *tree-sigma* A
⟨*proof*⟩

lemma *measurable-rec-tree*[*measurable* (*raw*)]:
  assumes *t*: $t \in B \rightarrow_M$ *tree-sigma* M
  assumes *l*: $l \in B \rightarrow_M A$
  assumes *n*: $(\lambda(x,\ l,\ v,\ r,\ al,\ ar).\ n\ x\ l\ v\ r\ al\ ar) \in$
    $(B \bigotimes_M$ *tree-sigma* $M \bigotimes_M M \bigotimes_M$ *tree-sigma* $M \bigotimes_M A \bigotimes_M A) \rightarrow_M A$ (**is** *?N* $\in$ *?M* $\rightarrow_M A$)
  shows $(\lambda x.$ *rec-tree* $(l\ x)\ (n\ x)\ (t\ x)) \in B \rightarrow_M A$
⟨*proof*⟩

lemma *measurable-case-tree* [*measurable* (*raw*)]:
  assumes $t \in B \rightarrow_M$ *tree-sigma* M
  assumes $l \in B \rightarrow_M A$
  assumes $(\lambda(x,\ l,\ v,\ r).\ n\ x\ l\ v\ r)$
      $\in B \bigotimes_M$ *tree-sigma* $M \bigotimes_M M \bigotimes_M$ *tree-sigma* $M \rightarrow_M A$
  shows $(\lambda x.$ *case-tree* $(l\ x)\ (n\ x)\ (t\ x)) \in B \rightarrow_M (A :: 'a$ *measure*$)$
⟨*proof*⟩

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* ⇒ *'a measure* ⇒ *bool* **where**
subalgebra M F = ((*space F* = *space M*) ∧ (*sets F* ⊆ *sets M*))

lemma *sub-measure-space*:
assumes *i*: *subalgebra M F*
shows *measure-space* (*space M*) (*sets F*) (*emeasure M*)
⟨*proof*⟩

definition *restr-to-subalg*::*'a measure* ⇒ *'a measure* ⇒ *'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*
⟨*proof*⟩

lemma *sets-restr-to-subalg* [*measurable-cong*]:
assumes *subalgebra M F*
shows *sets* (*restr-to-subalg M F*) = *sets F*
⟨*proof*⟩

lemma *emeasure-restr-to-subalg*:
assumes *subalgebra M F*
A ∈ *sets F*
shows *emeasure* (*restr-to-subalg M F*) *A* = *emeasure M A*
⟨*proof*⟩

lemma *null-sets-restr-to-subalg*:
assumes *subalgebra M F*
shows *null-sets* (*restr-to-subalg M F*) = *null-sets M* ∩ *sets F*
⟨*proof*⟩

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*⟩

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*⟩

lemma *prob-space-restr-to-subalg*:
assumes *subalgebra M F*
prob-space M
shows *prob-space* (*restr-to-subalg M F*)
⟨*proof*⟩

lemma *finite-measure-restr-to-subalg*:
assumes *subalgebra M F*
finite-measure M
shows *finite-measure* (*restr-to-subalg M F*)
⟨*proof*⟩

lemma *measurable-in-subalg*:
assumes *subalgebra M F*
$f \in$ *measurable F N*
shows $f \in$ *measurable* (*restr-to-subalg M F*) *N*
⟨*proof*⟩

lemma *measurable-in-subalg'*:
assumes *subalgebra M F*
$f \in$ *measurable* (*restr-to-subalg M F*) *N*
shows $f \in$ *measurable F N*
⟨*proof*⟩

lemma *measurable-from-subalg*:
assumes *subalgebra M F*
$f \in$ *measurable F N*
shows $f \in$ *measurable M N*
⟨*proof*⟩

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 $(\int^+ x.\ f\ x\ \partial(\textit{restr-to-subalg}\ M\ F)) = (\int^+ x.\ f\ x\ \partial M)$
⟨*proof*⟩

The following is the direct transposition of `integral_subalgebra` (from `Bochner_Integration`) in the current notations.

lemma *integral-subalgebra2*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes *subalgebra M F* **and**
[*measurable*]: $f \in$ *borel-measurable F*
shows $(\int x.\ f\ x\ \partial(\textit{restr-to-subalg}\ M\ F)) = (\int x.\ f\ x\ \partial M)$
⟨*proof*⟩

lemma *integrable-from-subalg*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes *subalgebra M F*
integrable (*restr-to-subalg M F*) *f*
shows *integrable M f*
⟨*proof*⟩

lemma *integrable-in-subalg*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes [*measurable*]: *subalgebra M F*
f ∈ *borel-measurable F*
integrable M f
shows *integrable* (*restr-to-subalg M F*) *f*
⟨*proof*⟩

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 fdM, restrict it to the sigma-algebra F, and apply the Radon-Nikodym theorem to write it as $gdM_{|F}$ 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* ⇒ *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ (*'a* ⇒ *ennreal*)
where
nn-cond-exp M F f =
(*if f* ∈ *borel-measurable M* ∧ *subalgebra M F*
then RN-deriv (*restr-to-subalg M F*) (*restr-to-subalg* (*density M f*) *F*)
else (λ-. *0*))

lemma
shows *borel-measurable-nn-cond-exp* [*measurable*]: *nn-cond-exp M F f* ∈ *borel-measurable F*
and *borel-measurable-nn-cond-exp2* [*measurable*]: *nn-cond-exp M F f* ∈ *borel-measurable M*
⟨*proof*⟩

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 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*⟩

sublocale *sigma-finite-subalgebra* ⊆ *sigma-finite-measure*
⟨*proof*⟩

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 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*⟩

sublocale *finite-measure-subalgebra* ⊆ *sigma-finite-subalgebra*
⟨*proof*⟩

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* ∈ *borel-measurable F g* ∈ *borel-measurable M*
shows (∫⁺ *x*. *f x* ∗ *nn-cond-exp M F g x* ∂*M*) = (∫⁺ *x*. *f x* ∗ *g x* ∂*M*)
⟨*proof*⟩

lemma *nn-cond-exp-charact*:
assumes ⋀*A*. *A* ∈ *sets F* ⟹ (∫⁺ *x* ∈ *A*. *f x* ∂*M*) = (∫⁺ *x* ∈ *A*. *g x* ∂*M*) **and**
[*measurable*]: *f* ∈ *borel-measurable M g* ∈ *borel-measurable F*
shows *AE x in M*. *g x* = *nn-cond-exp M F f x*
⟨*proof*⟩

lemma *nn-cond-exp-F-meas*:
assumes $f \in$ *borel-measurable* F
shows *AE x in M*. $f\ x =$ *nn-cond-exp M F f x*
⟨*proof*⟩

lemma *nn-cond-exp-prod*:
assumes [*measurable*]: $f \in$ *borel-measurable* F $g \in$ *borel-measurable* M
shows *AE x in M*. $f\ x *$ *nn-cond-exp M F g x* = *nn-cond-exp M F* $(\lambda x.\ f\ x * g\ x)\ x$
⟨*proof*⟩

lemma *nn-cond-exp-sum*:
assumes [*measurable*]: $f \in$ *borel-measurable* M $g \in$ *borel-measurable* M
shows *AE x in M*. *nn-cond-exp M F f x* + *nn-cond-exp M F g x* = *nn-cond-exp M F* $(\lambda x.\ f\ x + g\ x)\ x$
⟨*proof*⟩

lemma *nn-cond-exp-cong*:
assumes *AE x in M*. $f\ x = g\ x$
and [*measurable*]: $f \in$ *borel-measurable* M $g \in$ *borel-measurable* M
shows *AE x in M*. *nn-cond-exp M F f x* = *nn-cond-exp M F g x*
⟨*proof*⟩

lemma *nn-cond-exp-mono*:
assumes *AE x in M*. $f\ x \leq g\ x$
and [*measurable*]: $f \in$ *borel-measurable* M $g \in$ *borel-measurable* M
shows *AE x in M*. *nn-cond-exp M F f x* $\leq$ *nn-cond-exp M F g x*
⟨*proof*⟩

lemma *nested-subalg-is-sigma-finite*:
assumes *subalgebra M G subalgebra G F*
shows *sigma-finite-subalgebra M G*
⟨*proof*⟩

lemma *nn-cond-exp-nested-subalg*:
assumes *subalgebra M G subalgebra G F*
and [*measurable*]: $f \in$ *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*⟩

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 =
(λ*x*. *enn2real*(*nn-cond-exp M F* (λ*x*. *ennreal* (*f x*)) *x*) − *enn2real*(*nn-cond-exp M F* (λ*x*. *ennreal* (−*f x*)) *x*))

lemma
shows *borel-measurable-cond-exp* [*measurable*]: *real-cond-exp M F f* ∈ *borel-measurable F*
and *borel-measurable-cond-exp2* [*measurable*]: *real-cond-exp M F f* ∈ *borel-measurable M*
⟨*proof*⟩

context *sigma-finite-subalgebra*
begin

lemma *real-cond-exp-abs*:
assumes [*measurable*]: *f* ∈ *borel-measurable M*
shows *AE x in M*. *abs*(*real-cond-exp M F f x*) ≤ *nn-cond-exp M F* (λ*x*. *ennreal* (*abs*(*f x*))) *x*
⟨*proof*⟩

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* (λ*x*. *f x* ∗ *g x*) **and** *f-pos*[*simp*]: ⋀*x*. *f x* ≥ *0* **and**
[*measurable*]: *f* ∈ *borel-measurable F g* ∈ *borel-measurable M*
shows *integrable M* (λ*x*. *f x* ∗ *real-cond-exp M F g x*)
(∫ *x*. *f x* ∗ *real-cond-exp M F g x* ∂*M*) = (∫ *x*. *f x* ∗ *g x* ∂*M*)
⟨*proof*⟩

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* ($\lambda x.\ f\ x *$ *real-cond-exp M F g x*)
($\int x.\ f\ x *$ *real-cond-exp M F g x* ∂M) = ($\int x.\ f\ x * g\ x\ \partial M$)
⟨*proof*⟩

lemma *real-cond-exp-intA*:
assumes [*measurable*]: *integrable M f A* $\in$ *sets F*
shows ($\int x \in A.\ f\ x\ \partial M$) = ($\int x \in A.$ *real-cond-exp M F f x* ∂M)
⟨*proof*⟩

lemma *real-cond-exp-int* [*intro*]:
assumes *integrable M f*
shows *integrable M* (*real-cond-exp M F f*) ($\int x.$ *real-cond-exp M F f x* ∂M) = ($\int x.\ f\ x\ \partial M$)
⟨*proof*⟩

lemma *real-cond-exp-charact*:
assumes $\bigwedge A.\ A \in$ *sets F* $\Longrightarrow$ ($\int x \in A.\ f\ x\ \partial M$) = ($\int x \in A.\ g\ x\ \partial M$)
and [*measurable*]: *integrable M f integrable M g*
$g \in$ *borel-measurable F*
shows *AE x in M. real-cond-exp M F f x* = *g x*
⟨*proof*⟩

lemma *real-cond-exp-F-meas* [*intro*, *simp*]:
assumes *integrable M f*
$f \in$ *borel-measurable F*
shows *AE x in M. real-cond-exp M F f x* = *f x*
⟨*proof*⟩

lemma *real-cond-exp-mult*:
assumes [*measurable*]:$f \in$ *borel-measurable F* $g \in$ *borel-measurable M integrable M* ($\lambda x.\ f\ x * g\ x$)
shows *AE x in M. real-cond-exp M F* ($\lambda x.\ f\ x * g\ x$) *x* = *f x* * *real-cond-exp M F g x*
⟨*proof*⟩

lemma *real-cond-exp-add* [*intro*]:
assumes [*measurable*]: *integrable M f integrable M g*
shows *AE x in M. real-cond-exp M F* ($\lambda x.\ f\ x + g\ x$) *x* = *real-cond-exp M F f x* + *real-cond-exp M F g x*
⟨*proof*⟩

lemma *real-cond-exp-cong*:
assumes *ae*: *AE x in M. f x* = *g x* **and** [*measurable*]: $f \in$ *borel-measurable M g* $\in$ *borel-measurable M*
shows *AE x in M. real-cond-exp M F f x* = *real-cond-exp M F g x*
⟨*proof*⟩

lemma *real-cond-exp-cmult* [*intro*, *simp*]:
fixes *c*::*real*
assumes *integrable M f*
shows *AE x in M. real-cond-exp M F* (λ*x. c* ∗ *f x*) *x* = *c* ∗ *real-cond-exp M F f x*
⟨*proof*⟩

lemma *real-cond-exp-cdiv* [*intro*, *simp*]:
fixes *c*::*real*
assumes *integrable M f*
shows *AE x in M. real-cond-exp M F* (λ*x. f x* / *c*) *x* = *real-cond-exp M F f x* / *c*
⟨*proof*⟩

lemma *real-cond-exp-diff* [*intro*, *simp*]:
assumes [*measurable*]: *integrable M f integrable M g*
shows *AE x in M. real-cond-exp M F* (λ*x. f x* − *g x*) *x* = *real-cond-exp M F f x* − *real-cond-exp M F g x*
⟨*proof*⟩

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*⟩

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*⟩

lemma (**in** −) *measurable-P-restriction* [*measurable* (*raw*)]:
assumes [*measurable*]: *Measurable.pred M P A* ∈ *sets M*
shows {*x* ∈ *A. P x*} ∈ *sets M*
⟨*proof*⟩

lemma *real-cond-exp-gr-c*:
assumes [*measurable*]: *integrable M f*
and *AE*: *AE x in M. f x* > *c*
shows *AE x in M. real-cond-exp M F f x* > *c*
⟨*proof*⟩

lemma *real-cond-exp-less-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*⟩

lemma *real-cond-exp-ge-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*⟩

lemma *real-cond-exp-le-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*⟩

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*⟩

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*⟩

lemma *real-cond-exp-sum* [*intro*, *simp*]:
fixes *f*::*'b* ⇒ *'a* ⇒ *real*
assumes [*measurable*]: ⋀*i. integrable M (f i)*
shows *AE x in M. real-cond-exp M F* (λ*x.* ∑ *i*∈*I. f i x*) *x* = (∑ *i*∈*I. real-cond-exp M F* (*f i*) *x*)
⟨*proof*⟩

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* ⇒ *real*
assumes *X*: *integrable M X AE x in M. X x* ∈ *I*
assumes *I*: *I* = {*a* <..< *b*} ∨ *I* = {*a* <..} ∨ *I* = {..< *b*} ∨ *I* = *UNIV*
assumes *q*: *integrable M* (λ*x. q* (*X x*)) *convex-on I q q* ∈ *borel-measurable borel*
shows *AE x in M. real-cond-exp M F X x* ∈ *I*
AE x in M. q (*real-cond-exp M F X x*) ≤ *real-cond-exp M F* (λ*x. q* (*X x*)) *x*
⟨*proof*⟩

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* ⇒ *real*
assumes X: *integrable M X AE x in M. X x* ∈ I
assumes I: $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$
assumes q: *integrable M* (λ*x. q* (*X x*)) *convex-on I q q* ∈ *borel-measurable borel*
assumes H: *emeasure M* (*space M*) = ∞ ⟹ *0* ∈ I
shows *integrable M* (λ*x. q* (*real-cond-exp M F X x*))
⟨*proof*⟩

end

end

theory *Essential-Supremum*
imports *HOL−Analysis.Analysis*
begin

lemma *ae-filter-eq-bot-iff*: *ae-filter M* = *bot* ⟷ *emeasure M* (*space M*) = *0*
⟨*proof*⟩

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 prerequisiste to define the space L^∞.

definition *esssup*::*'a measure* ⇒ (*'a* ⇒ *'b*::{*second-countable-topology*, *dense-linorder*, *linorder-topology*, *complete-linorder*}) ⇒ *'b*
where *esssup M f* = (*if f* ∈ *borel-measurable M then Limsup* (*ae-filter M*) *f else top*)

lemma *esssup-non-measurable*: $f \notin M \to_M$ *borel* ⟹ *esssup M f* = *top*
⟨*proof*⟩

lemma *esssup-eq-AE*:
assumes f: $f \in M \to_M$ *borel* **shows** *esssup M f* = *Inf* {*z. AE x in M. f x* ≤ *z*}
⟨*proof*⟩

lemma *esssup-eq*: $f \in M \to_M$ *borel* ⟹ *esssup M f* = *Inf* {*z. emeasure M* {*x* ∈

space M. f x > z} = 0}
⟨*proof*⟩

lemma *esssup-zero-measure*:
emeasure M {x ∈ space M. f x > esssup M f} = 0
⟨*proof*⟩

lemma *esssup-AE*: *AE x in M. f x ≤ esssup M f*
⟨*proof*⟩

lemma *esssup-pos-measure*:
f ∈ borel-measurable M ⟹ z < esssup M f ⟹ emeasure M {x ∈ space M. f x > z} > 0
⟨*proof*⟩

lemma *esssup-I* [*intro*]: *f ∈ borel-measurable M ⟹ AE x in M. f x ≤ c ⟹ esssup M f ≤ c*
⟨*proof*⟩

lemma *esssup-AE-mono*: *f ∈ borel-measurable M ⟹ AE x in M. f x ≤ g x ⟹ esssup M f ≤ esssup M g*
⟨*proof*⟩

lemma *esssup-mono*: *f ∈ borel-measurable M ⟹ (⋀x. f x ≤ g x) ⟹ esssup M f ≤ esssup M g*
⟨*proof*⟩

lemma *esssup-AE-cong*:
f ∈ borel-measurable M ⟹ g ∈ borel-measurable M ⟹ AE x in M. f x = g x ⟹ esssup M f = esssup M g
⟨*proof*⟩

lemma *esssup-const*: *emeasure M (space M) ≠ 0 ⟹ esssup M (λx. c) = c*
⟨*proof*⟩

lemma *esssup-cmult*: **assumes** *c > (0::real)* **shows** *esssup M (λx. c * f x::ereal) = c * esssup M f*
⟨*proof*⟩

lemma *esssup-add*:
esssup M (λx. f x + g x::ereal) ≤ esssup M f + esssup M g
⟨*proof*⟩

lemma *esssup-zero-space*:
emeasure M (space M) = 0 ⟹ f ∈ borel-measurable M ⟹ esssup M f = (−∞::ereal)
⟨*proof*⟩

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*::*linorder* ⇒ '*a measure*) ⇒ ('*a* ⇒ '*t*) ⇒ *bool*
where
stopping-time F T = (∀ *t*. *Measurable.pred* (*F t*) (λ*x*. *T x* ≤ *t*))

lemma *stopping-time-cong*: (⋀*t x*. *x* ∈ *space* (*F t*) ⟹ *T x* = *S x*) ⟹ *stopping-time F T* = *stopping-time F S*
⟨*proof*⟩

lemma *stopping-timeD*: *stopping-time F T* ⟹ *Measurable.pred* (*F t*) (λ*x*. *T x* ≤ *t*)
⟨*proof*⟩

lemma *stopping-timeD2*: *stopping-time F T* ⟹ *Measurable.pred* (*F t*) (λ*x*. *t* < *T x*)
⟨*proof*⟩

lemma *stopping-timeI*[*intro?*]: (⋀*t*. *Measurable.pred* (*F t*) (λ*x*. *T x* ≤ *t*)) ⟹ *stopping-time F T*
⟨*proof*⟩

lemma *measurable-stopping-time*:
fixes *T* :: '*a* ⇒ '*t*::{*linorder-topology*, *second-countable-topology*}
assumes *T*: *stopping-time F T*
and *M*: ⋀*t*. *sets* (*F t*) ⊆ *sets M* ⋀*t*. *space* (*F t*) = *space M*
shows *T* ∈ *M* →$_M$ *borel*
⟨*proof*⟩

lemma *stopping-time-const*: *stopping-time F* (λ*x*. *c*)
⟨*proof*⟩

lemma *stopping-time-min*:
stopping-time F T ⟹ *stopping-time F S* ⟹ *stopping-time F* (λ*x*. *min* (*T x*) (*S x*))
⟨*proof*⟩

lemma *stopping-time-max*:
stopping-time F T ⟹ *stopping-time F S* ⟹ *stopping-time F* (λ*x*. *max* (*T x*) (*S x*))
⟨*proof*⟩

31 Filtration

locale *filtration* =
  fixes Ω :: *'a set* **and** F :: *'t*::{*linorder-topology*, *second-countable-topology*} $\Rightarrow$ *'a measure*
  assumes *space-F*: $\bigwedge i.$ *space* $(F\ i) = \Omega$
  assumes *sets-F-mono*: $\bigwedge i\ j.\ i \leq j \Longrightarrow$ *sets* $(F\ i) \leq$ *sets* $(F\ j)$
begin

31.1 σ-algebra of a Stopping Time

definition *pre-sigma* :: $('a \Rightarrow 't) \Rightarrow$ *'a measure*
where
  pre-sigma $T =$ *sigma* Ω $\{A.\ \forall t.\ \{\omega{\in}A.\ T\ \omega \leq t\} \in$ *sets* $(F\ t)\}$

lemma *space-pre-sigma*: *space* (*pre-sigma* T) $= \Omega$
  ⟨*proof*⟩

lemma *measure-pre-sigma*[*simp*]: *emeasure* (*pre-sigma* T) $= (\lambda\text{-}.\ 0)$
  ⟨*proof*⟩

lemma *sigma-algebra-pre-sigma*:
  assumes T: *stopping-time* $F\ T$
  shows *sigma-algebra* Ω $\{A.\ \forall t.\ \{\omega{\in}A.\ T\ \omega \leq t\} \in$ *sets* $(F\ t)\}$
  ⟨*proof*⟩

lemma *sets-pre-sigma*: *stopping-time* $F\ T \Longrightarrow$ *sets* (*pre-sigma* T) $= \{A.\ \forall t.\ \{\omega{\in}A.\ T\ \omega \leq t\} \in$ *sets* $(F\ t)\}$
  ⟨*proof*⟩

lemma *sets-pre-sigmaI*: *stopping-time* $F\ T \Longrightarrow (\bigwedge t.\ \{\omega{\in}A.\ T\ \omega \leq t\} \in$ *sets* $(F\ t)) \Longrightarrow A \in$ *sets* (*pre-sigma* T)
  ⟨*proof*⟩

lemma *pred-pre-sigmaI*:
  assumes T: *stopping-time* $F\ T$
  shows $(\bigwedge t.$ *Measurable.pred* $(F\ t)\ (\lambda\omega.\ P\ \omega \wedge T\ \omega \leq t)) \Longrightarrow$ *Measurable.pred* (*pre-sigma* T) P
  ⟨*proof*⟩

lemma *sets-pre-sigmaD*: *stopping-time* $F\ T \Longrightarrow A \in$ *sets* (*pre-sigma* T) $\Longrightarrow \{\omega{\in}A.\ T\ \omega \leq t\} \in$ *sets* $(F\ t)$
  ⟨*proof*⟩

lemma *stopping-time-le-const*: *stopping-time* $F\ T \Longrightarrow s \leq t \Longrightarrow$ *Measurable.pred* $(F\ t)\ (\lambda\omega.\ T\ \omega \leq s)$
  ⟨*proof*⟩

lemma *measurable-stopping-time-pre-sigma*:
  assumes T: *stopping-time* $F\ T$ **shows** $T \in$ *pre-sigma* $T \rightarrow_M$ *borel*

⟨*proof*⟩

lemma *mono-pre-sigma*:
assumes T: *stopping-time F T* **and** S: *stopping-time F S*
and *le*: $\bigwedge\omega.\ \omega \in \Omega \Longrightarrow T\ \omega \leq S\ \omega$
shows *sets* (*pre-sigma T*) ⊆ *sets* (*pre-sigma S*)
⟨*proof*⟩

lemma *stopping-time-less-const*:
assumes T: *stopping-time F T* **shows** *Measurable.pred* (*F t*) (λω. *T ω* < *t*)
⟨*proof*⟩

lemma *stopping-time-eq-const*: *stopping-time F T* ⟹ *Measurable.pred* (*F t*) (λω. *T ω* = *t*)
⟨*proof*⟩

lemma *stopping-time-less*:
assumes T: *stopping-time F T* **and** S: *stopping-time F S*
shows *Measurable.pred* (*pre-sigma T*) (λω. *T ω* < *S ω*)
⟨*proof*⟩

end

lemma *stopping-time-SUP-enat*:
fixes *T* :: *nat* ⇒ (′*a* ⇒ *enat*)
shows (⋀*i*. *stopping-time F* (*T i*)) ⟹ *stopping-time F* (*SUP i*. *T i*)
⟨*proof*⟩

lemma *less-eSuc-iff*: *a* < *eSuc b* ⟷ (*a* ≤ *b* ∧ *a* ≠ ∞)
⟨*proof*⟩

lemma *stopping-time-Inf-enat*:
fixes *F* :: *enat* ⇒ ′*a measure*
assumes *F*: *filtration* Ω *F*
assumes *P*: ⋀*i*. *Measurable.pred* (*F i*) (*P i*)
shows *stopping-time F* (λω. *Inf* {*i*. *P i ω*})
⟨*proof*⟩

lemma *stopping-time-Inf-nat*:
fixes *F* :: *nat* ⇒ ′*a measure*
assumes *F*: *filtration* Ω *F*
assumes *P*: ⋀*i*. *Measurable.pred* (*F i*) (*P i*) **and** *wf*: ⋀*i ω*. *ω* ∈ Ω ⟹ ∃ *n*. *P n ω*
shows *stopping-time F* (λω. *Inf* {*i*. *P i ω*})
⟨*proof*⟩

end

theory *Probability*
imports
Central-Limit-Theorem
Discrete-Topology
PMF-Impl
Projective-Limit
Random-Permutations
SPMF
Product-PMF
Hoeffding
Stream-Space
Tree-Space
Conditional-Expectation
Essential-Supremum
Stopping-Time
begin

end