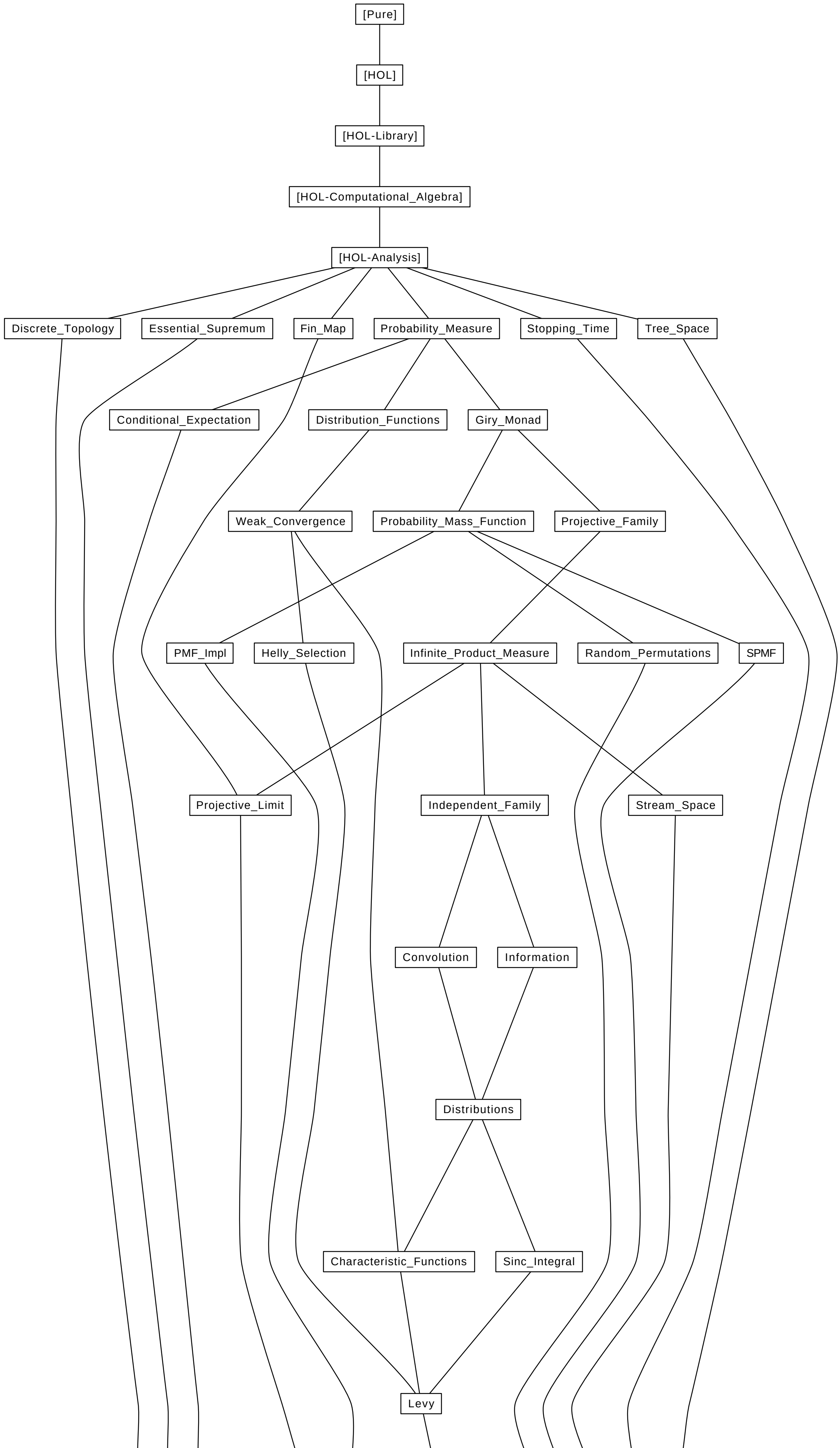
Measure and Probability Theory

October 8, 2017

Contents


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

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* $\Longrightarrow$ *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 ≡ $X \in$ *measurable M* M'
abbreviation (**in** *prob-space*) *expectation* ≡ $integral^L$ *M*
abbreviation (**in** *prob-space*) *variance X* ≡ $integral^L$ *M* ($\lambda x.$ (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*: $f \in$ *measurable M* M' **shows** *prob-space* (*distr M* M' *f*)
⟨*proof*⟩

lemma *prob-space-distrD*:
assumes *f*: $f \in$ *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* $\leq$ *1*
⟨*proof*⟩

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

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

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

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

lemma (**in** *prob-space*) *AE-in-set-eq-1*:
assumes *A*[*measurable*]: *A* ∈ *events* **shows** (*AE x in M. x* ∈ *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* ∈ *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-filter M* ≠ *bot*

⟨*proof*⟩

lemma (**in** *prob-space*) *AE-contr*:
assumes *ae*: $AE\ \omega\ in\ M.\ P\ \omega$ $AE\ \omega\ in\ M.\ \neg\ P\ \omega$
shows *False*
⟨*proof*⟩

lemma (**in** *prob-space*) *integral-ge-const*:
fixes c :: *real*
shows $integrable\ M\ f \Longrightarrow (AE\ x\ in\ M.\ c \leq f\ x) \Longrightarrow c \leq (\int x.\ f\ x\ \partial M)$
⟨*proof*⟩

lemma (**in** *prob-space*) *integral-le-const*:
fixes c :: *real*
shows $integrable\ M\ f \Longrightarrow (AE\ x\ in\ M.\ f\ x \leq c) \Longrightarrow (\int x.\ f\ x\ \partial M) \leq c$
⟨*proof*⟩

lemma (**in** *prob-space*) *nn-integral-ge-const*:
$(AE\ x\ in\ M.\ c \leq f\ x) \Longrightarrow c \leq (\int^+ x.\ f\ x\ \partial M)$
⟨*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: $integrable\ M\ X$ $AE\ x\ in\ M.\ X\ x \in I$
assumes I: $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$
assumes q: $integrable\ M\ (\lambda x.\ q\ (X\ x))$ $convex\text{-}on\ I\ q$
shows $q\ (expectation\ X) \leq expectation\ (\lambda x.\ q\ (X\ x))$
⟨*proof*⟩

1.1 Introduce binder for probability

syntax
-prob :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* (('$\mathcal{P}$'((/- *in* -./ -)')))

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* ⇒ *logic* (('$\mathcal{P}$'(- *in* -. - |/ -')))

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*$\}$ $S \in$ *events prob S* = *1*
⟨*proof*⟩

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

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

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

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

lemma (**in** *prob-space*) *prob-eq-0*:
$A \in$ *sets M* $\Longrightarrow$ *prob A* = *0* $\longleftrightarrow$ (*AE x in M.* $x \notin A$)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-eq-1*:
$A \in$ *sets M* $\Longrightarrow$ *prob A* = *1* $\longleftrightarrow$ (*AE x in M.* $x \in A$)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-sums*:
assumes *P*: $\bigwedge n.$ $\{x \in$*space M. P n x*$\} \in$ *events*

assumes Q: $\{x{\in}space\ M.\ Q\ x\} \in events$
assumes ae: $AE\ x\ in\ M.\ (\forall n.\ P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists!n.\ P\ n\ x))$
shows $(\lambda n.\ \mathcal{P}(x\ in\ M.\ P\ n\ x))\ sums\ \mathcal{P}(x\ in\ M.\ Q\ x)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-sum*:
assumes [*simp*, *intro*]: *finite I*
assumes P: $\bigwedge n.\ n \in I \Longrightarrow \{x{\in}space\ M.\ P\ n\ x\} \in events$
assumes Q: $\{x{\in}space\ M.\ Q\ x\} \in events$
assumes ae: $AE\ x\ in\ M.\ (\forall n{\in}I.\ P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists!n{\in}I.\ P\ n\ x))$
shows $\mathcal{P}(x\ in\ M.\ Q\ x) = (\sum n{\in}I.\ \mathcal{P}(x\ in\ M.\ P\ n\ x))$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-EX-countable*:
assumes $sets$: $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}space\ M.\ P\ i\ x\} \in sets\ M$ **and** I: *countable I*
assumes $disj$: $AE\ x\ in\ M.\ \forall i{\in}I.\ \forall j{\in}I.\ P\ i\ x \longrightarrow P\ j\ x \longrightarrow i = j$
shows $\mathcal{P}(x\ in\ M.\ \exists i{\in}I.\ P\ i\ x) = (\int^{+} i.\ \mathcal{P}(x\ in\ M.\ P\ i\ x)\ \partial count\text{-}space\ I)$
⟨*proof*⟩

lemma (**in** *prob-space*) *cond-prob-eq-AE*:
assumes P: $AE\ x\ in\ M.\ Q\ x \longrightarrow P\ x \longleftrightarrow P'\ x\ \{x{\in}space\ M.\ P\ x\} \in events$ $\{x{\in}space\ M.\ P'\ x\} \in events$
assumes Q: $AE\ x\ in\ M.\ Q\ x \longleftrightarrow Q'\ x\ \{x{\in}space\ M.\ Q\ x\} \in events\ \{x{\in}space$ $M.\ Q'\ x\} \in events$
shows $cond\text{-}prob\ M\ P\ Q = cond\text{-}prob\ M\ P'\ Q'$
⟨*proof*⟩

lemma (**in** *prob-space*) *joint-distribution-Times-le-fst*:
$random\text{-}variable\ MX\ X \Longrightarrow random\text{-}variable\ MY\ Y \Longrightarrow A \in sets\ MX \Longrightarrow B \in$ $sets\ MY$
$\Longrightarrow emeasure\ (distr\ M\ (MX \bigotimes_M MY)\ (\lambda x.\ (X\ x,\ Y\ x)))\ (A \times B) \le emeasure$ $(distr\ M\ MX\ X)\ A$
⟨*proof*⟩

lemma (**in** *prob-space*) *joint-distribution-Times-le-snd*:
$random\text{-}variable\ MX\ X \Longrightarrow random\text{-}variable\ MY\ Y \Longrightarrow A \in sets\ MX \Longrightarrow B \in$ $sets\ MY$
$\Longrightarrow emeasure\ (distr\ M\ (MX \bigotimes_M MY)\ (\lambda x.\ (X\ x,\ Y\ x)))\ (A \times B) \le emeasure$ $(distr\ M\ MY\ Y)\ B$
⟨*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*: $0 \leq variance\ (X{::}'a \Rightarrow real)$
⟨*proof*⟩

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

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

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*⟩

1.2 Distributions

definition *distributed* :: $'a\ measure \Rightarrow 'b\ measure \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow ennreal) \Rightarrow bool$
where
distributed M N X f ⟷
$distr\ M\ N\ X = density\ N\ f \wedge f \in borel\text{-}measurable\ N \wedge X \in measurable\ M\ N$

lemma
assumes *distributed M N X f*
shows *distributed-distr-eq-density*: $distr\ M\ N\ X = density\ N\ f$
and *distributed-measurable*: $X \in measurable\ M\ N$
and *distributed-borel-measurable*: $f \in borel\text{-}measurable\ N$
⟨*proof*⟩

lemma
assumes *D*: *distributed M N X f*
shows *distributed-measurable'*[*measurable-dest*]:

$g \in$ *measurable* $L\ M \Longrightarrow (\lambda x.\ X\ (g\ x)) \in$ *measurable* $L\ N$
and *distributed-borel-measurable′*[*measurable-dest*]:
$h \in$ *measurable* $L\ N \Longrightarrow (\lambda x.\ f\ (h\ x)) \in$ *borel-measurable* L
⟨*proof*⟩

lemma *distributed-real-measurable*:
$(\bigwedge x.\ x \in$ *space* $N \Longrightarrow 0 \le f\ x) \Longrightarrow$ *distributed* $M\ N\ X\ (\lambda x.$ *ennreal* $(f\ x)) \Longrightarrow$
$f \in$ *borel-measurable* N
⟨*proof*⟩

lemma *distributed-real-measurable′*:
$(\bigwedge x.\ x \in$ *space* $N \Longrightarrow 0 \le f\ x) \Longrightarrow$ *distributed* $M\ N\ X\ (\lambda x.$ *ennreal* $(f\ x)) \Longrightarrow$
$h \in$ *measurable* $L\ N \Longrightarrow (\lambda x.\ f\ (h\ x)) \in$ *borel-measurable* L
⟨*proof*⟩

lemma *joint-distributed-measurable1*:
distributed $M\ (S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x))\ f \Longrightarrow h1 \in$ *measurable* $N\ M \Longrightarrow$
$(\lambda x.\ X\ (h1\ x)) \in$ *measurable* $N\ S$
⟨*proof*⟩

lemma *joint-distributed-measurable2*:
distributed $M\ (S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x))\ f \Longrightarrow h2 \in$ *measurable* $N\ M \Longrightarrow$
$(\lambda x.\ Y\ (h2\ x)) \in$ *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* $M\ (X -\text{`}\ \{a\} \cap$ *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*:
$distributed\ M\ N\ X\ f \Longrightarrow g \in borel\text{-}measurable\ N \Longrightarrow (\int^{+} x.\ f\ x * g\ x\ \partial N) = (\int^{+} x.\ g\ (X\ x)\ \partial M)$
⟨*proof*⟩

lemma *distributed-integral*:
$distributed\ M\ N\ X\ f \Longrightarrow g \in borel\text{-}measurable\ N \Longrightarrow (\bigwedge x.\ x \in space\ N \Longrightarrow 0 \leq f\ x) \Longrightarrow$
$(\int x.\ f\ x * g\ x\ \partial N) = (\int x.\ g\ (X\ x)\ \partial M)$
⟨*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: $f \in borel\text{-}measurable\ P$
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 $sigma\text{-}finite\text{-}measure\ S\ sigma\text{-}finite\text{-}measure\ T$
assumes $X[measurable]$: $X \in measurable\ M\ S$ **and** $Y[measurable]$: $Y \in measurable\ M\ T$
assumes $[measurable]$: $f \in borel\text{-}measurable\ (S \bigotimes_M T)$ **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* $\Rightarrow$ *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 $M\ X\ f \longleftrightarrow$
    $(\forall x.\ 0 \leq f\ x) \wedge$
    distributed M (*count-space* $(X$‘*space* $M))$ X $(\lambda x.$ *ennreal* $(f\ x)) \wedge$
    finite $(X$‘*space* $M)$

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

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

lemma *simple-distributed-finite*[*dest*]: *simple-distributed* $M\ X\ P \Longrightarrow$ *finite* $(X$‘*space* $M)$

⟨*proof*⟩

lemma (**in** *prob-space*) *distributed-simple-function-superset*:
assumes *X*: *simple-function M X* $\bigwedge x.\ x \in X$ ' *space M* $\Longrightarrow$ *P x* = *measure M* (*X* −' {*x*} ∩ *space M*)
assumes *A*: *X'space M* ⊆ *A finite A*
defines *S* ≡ *count-space A* **and** *P'* ≡ (λ*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*
$\bigwedge x.\ 0 \le P\ x$
$\bigwedge x.\ x \in X$ ' *space M* $\Longrightarrow$ *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* (λ*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 $\Longrightarrow$ *simple-function M X*
⟨*proof*⟩

lemma *simple-distributed-measure*:
simple-distributed M X P $\Longrightarrow$ *a* ∈ *X'space M* $\Longrightarrow$ *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*) $\bigotimes_M$ *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*) $\bigotimes_M$ *count-space* (*Y'space M*) $\bigotimes_M$ *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* ∈ *borel-measurable M1 AE x in M1*. *0* ≤ *f x*
and *eq*: ⋀*A*. *A* ∈ *E* ⟹ *emeasure M* (*X* −' *A* ∩ *space M*) = (∫⁺ *x*. *f x* ∗ *indicator A x* ∂*M1*)
shows *distributed M M1 X f*
⟨*proof*⟩

lemma *distributedI-borel-atMost*:
fixes *f* :: *real* ⇒ *real*
assumes [*measurable*]: *X* ∈ *borel-measurable M*
and [*measurable*]: *f* ∈ *borel-measurable borel* **and** *f*[*simp*]: *AE x in lborel*. *0* ≤ *f x*
and *g-eq*: ⋀*a*. (∫⁺*x*. *f x* ∗ *indicator* {..*a*} *x* ∂*lborel*) = *ennreal* (*g a*)
and *M-eq*: ⋀*a*. *emeasure M* {*x*∈*space M*. *X x* ≤ *a*} = *ennreal* (*g a*)
shows *distributed M lborel X f*
⟨*proof*⟩

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

lemma *prob-space-uniform-measure*:
assumes *A*: *emeasure M A* ≠ *0 emeasure M A* ≠ ∞
shows *prob-space* (*uniform-measure M A*)
⟨*proof*⟩

lemma *prob-space-uniform-count-measure*: *finite A* ⟹ *A* ≠ {} ⟹ *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*∈*space M*. *Q x*}. *P x*) = $\mathcal{P}$(*x in M*. *P x* | *Q x*)
⟨*proof*⟩

lemma *prob-space-point-measure*:
finite S ⟹ (⋀*s*. *s* ∈ *S* ⟹ *0* ≤ *p s*) ⟹ (∑*s*∈*S*. *p s*) = *1* ⟹ *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* ∈ *J* → *K finite K*
shows *distr* (*PiM K M*) (Pi_M *J* (λ*x*. *M* (*t x*))) (λω. λ*n*∈*J*. ω (*t n*)) = *PiM J* (λ*x*. *M* (*t x*))
⟨*proof*⟩

lemma (**in** *product-prob-space*) *distr-restrict*:
J ⊆ *K* ⟹ *finite K* ⟹ (Π_M *i*∈*J*. *M i*) = *distr* (Π_M *i*∈*K*. *M i*) (Π_M *i*∈*J*. *M i*) (λ*f*. *restrict f J*)
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-prod-emb*[*simp*]:
assumes *L*: *J* ⊆ *L finite L* **and** *X*: *X* ∈ *sets* (Pi_M *J M*)
shows *emeasure* (Pi_M *L M*) (*prod-emb L M J X*) = *emeasure* (Pi_M *J M*) *X*
⟨*proof*⟩

lemma *emeasure-distr-restrict*:
assumes *I* ⊆ *K* **and** *Q*[*measurable-cong*]: *sets Q* = *sets* (*PiM K M*) **and** *A*[*measurable*]: *A* ∈ *sets* (*PiM I M*)
shows *emeasure* (*distr Q* (*PiM I M*) (λω. *restrict* ω *I*)) *A* = *emeasure Q* (*prod-emb K M I A*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-space-completion*: *prob-space* (*completion M*)
⟨*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
  $cdf :: real\ measure \Rightarrow real \Rightarrow real$
where
  $cdf\ M \equiv \lambda x.\ measure\ M\ \{..x\}$

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*]: $a \in sets\ borel \Longrightarrow a \in sets\ M$
  ⟨*proof*⟩

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

lemma *cdf-nondecreasing*: $x \le y \Longrightarrow cdf\ M\ x \le cdf\ M\ y$
  ⟨*proof*⟩

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

lemma *cdf-nonneg*: $cdf\ M\ x \ge 0$
  ⟨*proof*⟩

lemma *cdf-bounded*: $cdf\ M\ x \le measure\ M\ (space\ M)$
  ⟨*proof*⟩

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

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

lemma *cdf-lim-neg-infty*: $((\lambda i.\ cdf\ M\ (-\ real\ i)) \longrightarrow 0)$
⟨*proof*⟩

lemma *cdf-lim-at-bot*: $(cdf\ M \longrightarrow 0)\ at\text{-}bot$
⟨*proof*⟩

lemma *cdf-is-right-cont*: $continuous\ (at\text{-}right\ a)\ (cdf\ M)$
⟨*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 ∈ *borel-measurable borel* ⟹ *f* ∈ *borel-measurable M*
⟨*proof*⟩

end

lemma (**in** *prob-space*) *real-distribution-distr* [*intro*, *simp*]:
random-variable borel X ⟹ *real-distribution* (*distr M borel X*)
⟨*proof*⟩

2.2 Uniqueness

lemma (**in** *finite-borel-measure*) *emeasure-Ioc*:
assumes $a \le b$ **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* : $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$ **and**
right-cont-F : $\bigwedge a.$ *continuous* (*at-right* a) F **and**
lim-F-at-bot : ($F \longrightarrow 0$) *at-bot*
shows *emeasure-interval-measure-Iic*: *emeasure* (*interval-measure* F) $\{..\ x\} = F\ x$
and *measure-interval-measure-Iic*: *measure* (*interval-measure* F) $\{..\ x\} = F\ x$
⟨*proof*⟩

lemma *cdf-interval-measure*:
($\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$) $\Longrightarrow$ ($\bigwedge a.$ *continuous* (*at-right* a) F) $\Longrightarrow$ ($F \longrightarrow 0$) *at-bot* $\Longrightarrow$ *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* ⇒ (*real* ⇒ *real*)) ⇒ (*real* ⇒ *real*) ⇒ *bool*
where
weak-conv F-seq F ≡ ∀ *x*. *isCont F x* ⟶ ($\lambda n.$ *F-seq n x*) ⟶ *F x*

5 Weak Convergence of Distributions

definition
weak-conv-m :: (*nat* ⇒ *real measure*) ⇒ *real measure* ⇒ *bool*
where
weak-conv-m M-seq M ≡ *weak-conv* ($\lambda n.$ *cdf* (*M-seq n*)) (*cdf M*)

6 Skorohod's theorem

locale *right-continuous-mono* =
fixes f :: *real* ⇒ *real* **and** $a\ b$ :: *real*
assumes *cont*: $\bigwedge x.$ *continuous* (*at-right* x) f

assumes *mono*: *mono f*
assumes *bot*: $(f \longrightarrow a)$ *at-bot*
assumes *top*: $(f \longrightarrow b)$ *at-top*
begin

abbreviation I :: *real* ⇒ *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-on* $I\ \{a <..< b\}$
⟨*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-measurable borel*
⟨*proof*⟩

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

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

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

end

context
fixes μ :: *nat* ⇒ *real measure*
and M :: *real measure*
assumes μ: $\bigwedge n.$ *real-distribution* $(\mu\ n)$
assumes M: *real-distribution M*
assumes μ*-to-M*: *weak-conv-m* μ *M*

begin

theorem *Skorohod*:
$\exists$ (Ω :: *real measure*) (*Y-seq* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *real*) (*Y* :: *real* $\Rightarrow$ *real*).
  prob-space Ω $\wedge$
  ($\forall$ *n*. *Y-seq n* $\in$ *measurable* Ω *borel*) $\wedge$
  ($\forall$ *n*. *distr* Ω *borel* (*Y-seq n*) = μ *n*) $\wedge$
  Y $\in$ *measurable* Ω *lborel* $\wedge$
  distr Ω *borel Y* = *M* $\wedge$
  ($\forall$ *x* $\in$ *space* Ω. (λ*n*. *Y-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-countable-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-measurable borel*
 shows
  (λ *n*. *integral*^{L} (μ *n*) *f*) $\longrightarrow$ *integral*^{L} *M f*
⟨*proof*⟩

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

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

end

definition
 cts-step :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real*
where
 cts-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-step a b*)
⟨*proof*⟩

lemma (**in** *real-distribution*) *cdf-cts-step*:
assumes [*arith*]: $x < y$
shows *cdf M x* $\leq$ *integral*^{L} *M* (*cts-step x y*) **and** *integral*^{L} *M* (*cts-step x y*) $\leq$ *cdf M y*
⟨*proof*⟩

context
fixes *M-seq* :: *nat* ⇒ *real measure*
and *M* :: *real measure*
assumes *distr-M-seq* [*simp*]: ⋀*n*. *real-distribution* (*M-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

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

begin

7 Sub-probability spaces

locale *subprob-space* = *finite-measure* +
assumes *emeasure-space-le-1*: *emeasure M* (*space M*) $\leq$ *1*
assumes *subprob-not-empty*: *space M* $\neq$ {}

lemma *subprob-spaceI*[*Pure.intro*!]:
assumes $*$: *emeasure M* (*space M*) $\leq$ *1*
assumes *space M* $\neq$ {}
shows *subprob-space M*
⟨*proof*⟩

lemma (**in** *subprob-space*) *emeasure-subprob-space-less-top*: *emeasure M A* $\neq$ *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* ⇒ *real* **and** g :: *real* ⇒ *real* **and** g' :: *real* ⇒ *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* g $\{a..b\}$ **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* ($\lambda x.\ f\ (g\ x) * g'\ x$)) $\{a..b\}$
⟨*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* $M1 \bigotimes_M M2$
⟨*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*: $A \cap$ *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* ∈ *measurable M* (*subprob-algebra N*)
begin

lemma *subprob-space-kernel*: *a* ∈ *space M* ⟹ *subprob-space* (*K a*)
⟨*proof*⟩

lemma *sets-kernel*: *a* ∈ *space M* ⟹ *sets* (*K a*) = *sets N*
⟨*proof*⟩

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

end

lemma *measurable-subprob-algebra*:
(⋀*a*. *a* ∈ *space M* ⟹ *subprob-space* (*K a*)) ⟹
(⋀*a*. *a* ∈ *space M* ⟹ *sets* (*K a*) = *sets N*) ⟹
(⋀*A*. *A* ∈ *sets N* ⟹ (λ*a*. *emeasure* (*K a*) *A*) ∈ *borel-measurable M*) ⟹
K ∈ *measurable M* (*subprob-algebra N*)
⟨*proof*⟩

lemma *measurable-submarkov*:
K ∈ *measurable M* (*subprob-algebra M*) ⟷
(∀ *x*∈*space M*. *subprob-space* (*K x*) ∧ *sets* (*K x*) = *sets M*) ∧
(∀ *A*∈*sets M*. (λ*x*. *emeasure* (*K x*) *A*) ∈ *measurable M borel*)
⟨*proof*⟩

lemma *measurable-subprob-algebra-generated*:

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

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

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

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

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

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

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

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

lemma *restrict-space-measurable*:
assumes *X*: $X \neq \{\}\ X \in sets\ K$
assumes *N*: $N \in measurable\ M\ (subprob\text{-}algebra\ K)$

shows (λx. *restrict-space* ($N\,x$) X) $\in$ *measurable M* (*subprob-algebra* (*restrict-space K X*))
⟨*proof*⟩

8 Properties of return

definition *return* :: $'a$ *measure* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *measure* **where**
return R x = *measure-of* (*space R*) (*sets R*) (λA. *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* $\Longrightarrow$ *return M* = *return N*
⟨*proof*⟩

lemma *return-cong*: *sets A* = *sets B* $\Longrightarrow$ *return A x* = *return B x*
⟨*proof*⟩

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

lemma *prob-space-return*: $x \in$ *space M* $\Longrightarrow$ *prob-space* (*return M x*)
⟨*proof*⟩

lemma *subprob-space-return*: $x \in$ *space M* $\Longrightarrow$ *subprob-space* (*return M x*)
⟨*proof*⟩

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

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

lemma *AE-return*:
assumes [*simp*]: $x \in$ *space M* **and** [*measurable*]: *Measurable.pred M P*
shows (*AE y in return M x*. *P y*) $\longleftrightarrow$ *P x*

⟨*proof*⟩

lemma *nn-integral-return*:
assumes $x \in$ *space M* $g \in$ *borel-measurable M*
shows $(\int^{+} a.\ g\ a\ \partial return\ M\ x) = g\ x$
⟨*proof*⟩

lemma *integral-return*:
fixes $g :: - \Rightarrow 'a :: \{banach,\ second\text{-}countable\text{-}topology\}$
assumes $x \in$ *space M* $g \in$ *borel-measurable M*
shows $(\int a.\ g\ a\ \partial return\ M\ x) = g\ x$
⟨*proof*⟩

lemma *return-measurable*[*measurable*]: *return N* $\in$ *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 ($\lambda x.$ *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 ($\lambda x.$ $integral^N$ (*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 ($\lambda x.$ *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 ($\lambda x.$ *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*⟩

9 Join

definition *join* :: $'a$ *measure measure* $\Rightarrow$ $'a$ *measure* **where**
join M = *measure-of* (*space* (*select-sets* M)) (*sets* (*select-sets* M)) (λB. $\int^+ M'$. *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: $A \in$ *sets* N
shows *emeasure* (*join* M) A = ($\int^+ M'$. *emeasure* M' A ∂M)
⟨*proof*⟩

lemma *measurable-join*:
join $\in$ *measurable* (*subprob-algebra* (*subprob-algebra* N)) (*subprob-algebra* N)
⟨*proof*⟩

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

lemma *measurable-join1*:
⟦ $f \in$ *measurable* N K; *sets* M = *sets* (*subprob-algebra* N) ⟧
$\Longrightarrow$ $f \in$ *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*: $sets\ M = \{\} \Longrightarrow sets\ (bind\ M\ f) = \{\{\}\}$
⟨*proof*⟩

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

lemma *sets-bind*[*simp*, *measurable-cong*]:
assumes f: $\bigwedge x.\ x \in space\ M \Longrightarrow sets\ (f\ x) = sets\ N$ **and** M: $space\ M \neq \{\}$
shows $sets\ (bind\ M\ f) = sets\ N$
⟨*proof*⟩

lemma *space-bind*[*simp*]:
assumes $\bigwedge x.\ x \in space\ M \Longrightarrow sets\ (f\ x) = sets\ N$ **and** $space\ M \neq \{\}$
shows $space\ (bind\ M\ f) = space\ N$
⟨*proof*⟩

lemma *bind-cong-All*:
assumes $\forall x \in space\ M.\ f\ x = g\ x$
shows $bind\ M\ f = bind\ M\ g$
⟨*proof*⟩

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

lemma *bind-nonempty'*:
assumes $f \in measurable\ M\ (subprob\text{-}algebra\ N)\ x \in space\ M$
shows $bind\ M\ f = join\ (distr\ M\ (subprob\text{-}algebra\ N)\ f)$
⟨*proof*⟩

lemma *bind-nonempty''*:
assumes $f \in measurable\ M\ (subprob\text{-}algebra\ N)\ space\ M \neq \{\}$
shows $bind\ M\ f = join\ (distr\ M\ (subprob\text{-}algebra\ N)\ f)$
⟨*proof*⟩

lemma *emeasure-bind*:
$[\![space\ M \neq \{\};\ f \in measurable\ M\ (subprob\text{-}algebra\ N);X \in sets\ N]\!]$
$\Longrightarrow emeasure\ (M \gg\!= f)\ X = \int^+ x.\ emeasure\ (f\ x)\ X\ \partial M$
⟨*proof*⟩

lemma *nn-integral-bind*:
assumes f: $f \in borel\text{-}measurable\ B$
assumes N: $N \in measurable\ M\ (subprob\text{-}algebra\ B)$
shows $(\int^+ x.\ f\ x\ \partial(M \gg\!= N)) = (\int^+ x.\ \int^+ y.\ f\ y\ \partial N\ x\ \partial M)$
⟨*proof*⟩

lemma *AE-bind*:
assumes N[*measurable*]: $N \in measurable\ M\ (subprob\text{-}algebra\ B)$
assumes P[*measurable*]: $Measurable.pred\ B\ P$

shows (*AE x in M* $\gg=$ *N*. *P x*) $\longleftrightarrow$ (*AE x in M*. *AE y in N x*. *P y*)
⟨*proof*⟩

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

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

lemma *measurable-bind2*:
assumes $f \in$ *measurable M* (*subprob-algebra N*) **and** $g \in$ *measurable N* (*subprob-algebra R*)
shows (λx. *bind* (*f x*) *g*) $\in$ *measurable M* (*subprob-algebra R*)
⟨*proof*⟩

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

lemma
fixes *f* :: - $\Rightarrow$ *real*
assumes *f-measurable* [*measurable*]: $f \in$ *borel-measurable K*
and *f-bounded*: $\bigwedge x.\ x \in$ *space K* $\Longrightarrow |f\ x| \leq B$
and *N* [*measurable*]: $N \in$ *measurable M* (*subprob-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* ∂M (**is** *?integral*)
⟨*proof*⟩

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

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

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

lemma *bind-distr*:
assumes *f*[*measurable*]: *f* ∈ *measurable M X*
assumes *N*[*measurable*]: *N* ∈ *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* ∈ *measurable M* (*subprob-algebra K*)

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

lemma *bind-restrict-space*:
assumes A: $A \cap space\ M \neq \{\}\ A \cap space\ M \in sets\ M$
and f: $f \in measurable\ (restrict\text{-}space\ M\ A)\ (subprob\text{-}algebra\ N)$
shows $restrict\text{-}space\ M\ A \gg= f = M \gg= (\lambda x.\ if\ x \in A\ then\ f\ x\ else\ null\text{-}measure\ (f\ (SOME\ x.\ x \in A \wedge x \in space\ M)))$
(**is** $?lhs = ?rhs$ **is** $- = M \gg= ?f$)
⟨*proof*⟩

lemma *bind-const'*: $[\![prob\text{-}space\ M;\ subprob\text{-}space\ N]\!] \Longrightarrow M \gg= (\lambda x.\ N) = N$
⟨*proof*⟩

lemma *bind-return-distr*:
$space\ M \neq \{\} \Longrightarrow f \in measurable\ M\ N \Longrightarrow bind\ M\ (return\ N \circ f) = distr\ M\ N\ f$
⟨*proof*⟩

lemma *bind-return-distr'*:
$space\ M \neq \{\} \Longrightarrow f \in measurable\ M\ N \Longrightarrow bind\ M\ (\lambda x.\ return\ N\ (f\ x)) = distr\ M\ N\ f$
⟨*proof*⟩

lemma *bind-assoc*:
fixes $f :: 'a \Rightarrow 'b\ measure$ **and** $g :: 'b \Rightarrow 'c\ measure$
assumes $M1$: $f \in measurable\ M\ (subprob\text{-}algebra\ N)$ **and** $M2$: $g \in measurable\ N\ (subprob\text{-}algebra\ R)$
shows $bind\ (bind\ M\ f)\ g = bind\ M\ (\lambda x.\ bind\ (f\ x)\ g)$
⟨*proof*⟩

lemma *double-bind-assoc*:
assumes Mg: $g \in measurable\ N\ (subprob\text{-}algebra\ N')$
assumes Mf: $f \in measurable\ M\ (subprob\text{-}algebra\ M')$
assumes Mh: $case\text{-}prod\ h \in measurable\ (M \bigotimes_M M')\ N$
shows $do\ \{x \leftarrow M;\ y \leftarrow f\ x;\ g\ (h\ x\ y)\} = do\ \{x \leftarrow M;\ y \leftarrow f\ x;\ return\ N\ (h\ x\ y)\} \gg= g$
⟨*proof*⟩

lemma (**in** *prob-space*) *M-in-subprob*[*measurable* (*raw*)]: $M \in space\ (subprob\text{-}algebra\ M)$
⟨*proof*⟩

lemma (**in** *pair-prob-space*) *pair-measure-eq-bind*:
$(M1 \bigotimes_M M2) = (M1 \gg= (\lambda x.\ M2 \gg= (\lambda y.\ return\ (M1 \bigotimes_M M2)\ (x,\ y))))$
⟨*proof*⟩

lemma (**in** *pair-prob-space*) *bind-rotate*:
assumes $C[measurable]$: $(\lambda(x, y).\ C\ x\ y) \in measurable\ (M1 \bigotimes_M M2)\ (subprob\text{-}algebra\ N)$
shows $(M1 \gg= (\lambda x.\ M2 \gg= (\lambda y.\ C\ x\ y))) = (M2 \gg= (\lambda y.\ M1 \gg= (\lambda x.\ C\ x\ y)))$
⟨*proof*⟩

lemma *bind-return''*: $sets\ M = sets\ N \Longrightarrow M \gg= return\ N = M$
⟨*proof*⟩

lemma (**in** *prob-space*) *distr-const*[*simp*]:
$c \in space\ N \Longrightarrow distr\ M\ N\ (\lambda x.\ c) = return\ N\ c$
⟨*proof*⟩

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

lemma *null-measure-in-space-subprob-algebra* [*simp*]:
$null\text{-}measure\ M \in space\ (subprob\text{-}algebra\ M) \longleftrightarrow space\ M \neq \{\}$
⟨*proof*⟩

9.1 Giry monad on probability spaces

definition *prob-algebra* :: $'a\ measure \Rightarrow\ 'a\ measure\ measure$ **where**
$prob\text{-}algebra\ K = restrict\text{-}space\ (subprob\text{-}algebra\ K)\ \{M.\ prob\text{-}space\ M\}$

lemma *space-prob-algebra*: $space\ (prob\text{-}algebra\ M) = \{N.\ sets\ N = sets\ M \wedge prob\text{-}space\ N\}$
⟨*proof*⟩

lemma *measurable-measure-prob-algebra*[*measurable*]:
$a \in sets\ A \Longrightarrow (\lambda M.\ Sigma\text{-}Algebra.measure\ M\ a) \in prob\text{-}algebra\ A \rightarrow_M borel$
⟨*proof*⟩

lemma *measurable-prob-algebraD*:
$f \in N \rightarrow_M prob\text{-}algebra\ M \Longrightarrow f \in N \rightarrow_M subprob\text{-}algebra\ M$
⟨*proof*⟩

lemma *measure-measurable-prob-algebra2*:
$Sigma\ (space\ M)\ A \in sets\ (M \bigotimes_M N) \Longrightarrow L \in M \rightarrow_M prob\text{-}algebra\ N \Longrightarrow$
$(\lambda x.\ Sigma\text{-}Algebra.measure\ (L\ x)\ (A\ x)) \in borel\text{-}measurable\ M$
⟨*proof*⟩

lemma *measurable-prob-algebraI*:
$(\bigwedge x.\ x \in space\ N \Longrightarrow prob\text{-}space\ (f\ x)) \Longrightarrow f \in N \rightarrow_M subprob\text{-}algebra\ M \Longrightarrow f \in N \rightarrow_M prob\text{-}algebra\ M$
⟨*proof*⟩

lemma *measurable-distr-prob-space*:
assumes *f*: $f \in M \rightarrow_M N$
shows $(\lambda M'.\ distr\ M'\ N\ f) \in prob\text{-}algebra\ M \rightarrow_M prob\text{-}algebra\ N$
⟨*proof*⟩

lemma *measurable-return-prob-space*[*measurable*]: $return\ N \in N \rightarrow_M prob\text{-}algebra\ N$
⟨*proof*⟩

lemma *measurable-distr-prob-space2*[*measurable (raw)*]:
assumes *f*: $g \in L \rightarrow_M prob\text{-}algebra\ M\ (\lambda(x, y).\ f\ x\ y) \in L \bigotimes_M M \rightarrow_M N$
shows $(\lambda x.\ distr\ (g\ x)\ N\ (f\ x)) \in L \rightarrow_M prob\text{-}algebra\ N$
⟨*proof*⟩

lemma *measurable-bind-prob-space*:
assumes *f*: $f \in M \rightarrow_M prob\text{-}algebra\ N$ **and** *g*: $g \in N \rightarrow_M prob\text{-}algebra\ R$
shows $(\lambda x.\ bind\ (f\ x)\ g) \in M \rightarrow_M prob\text{-}algebra\ R$
⟨*proof*⟩

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

lemma *measurable-prob-algebra-generated*:
assumes *eq*: $sets\ N = sigma\text{-}sets\ \Omega\ G$ **and** $Int\text{-}stable\ G\ G \subseteq Pow\ \Omega$
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 $K \in measurable\ M\ (prob\text{-}algebra\ N)$
⟨*proof*⟩

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

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

lemma *measurable-pair-prob*[*measurable*]:
$f \in M \rightarrow_M prob\text{-}algebra\ N \Longrightarrow g \in M \rightarrow_M prob\text{-}algebra\ L \Longrightarrow (\lambda x.\ f\ x \bigotimes_M g\ x) \in M \rightarrow_M prob\text{-}algebra\ (N \bigotimes_M L)$
⟨*proof*⟩

lemma *emeasure-bind-prob-algebra*:
assumes *A*: $A \in space\ (prob\text{-}algebra\ N)$

assumes B: $B \in N \rightarrow_M$ *prob-algebra* L
assumes X: $X \in$ *sets* L
shows *emeasure* (*bind* A B) X = ($\int^+ x$. *emeasure* (B x) X ∂A)
⟨*proof*⟩

lemma *prob-space-bind'*:
assumes A: $A \in$ *space* (*prob-algebra* M) **and** B: $B \in M \rightarrow_M$ *prob-algebra* N
shows *prob-space* ($A \gg= B$)
⟨*proof*⟩

lemma *sets-bind'*:
assumes A: $A \in$ *space* (*prob-algebra* M) **and** B: $B \in M \rightarrow_M$ *prob-algebra* N
shows *sets* ($A \gg= B$) = *sets* N
⟨*proof*⟩

lemma *bind-cong-AE'*:
assumes M: $M \in$ *space* (*prob-algebra* L)
and f: $f \in L \rightarrow_M$ *prob-algebra* N **and** g: $g \in L \rightarrow_M$ *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-space* A) f = N
⟨*proof*⟩

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

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

lemma *bind-cong-strong*: $M = N \Longrightarrow (\bigwedge x.\ x \in$ *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-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-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*⟩

end

10 Projective Family

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

lemma *vimage-restrict-preseve-mono*:
assumes *J*: $J \subseteq I$
and *sets*: $A \subseteq (\Pi_E\ i{\in}J.\ S\ i)$ $B \subseteq (\Pi_E\ i{\in}J.\ S\ i)$ **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-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*) ≠ {}
⟨*proof*⟩

abbreviation
emb L K ≡ *prod-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* ⊆ *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* ⟹ *X* = *Y*
⟨*proof*⟩

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

inductive-set *generator* :: ('i ⇒ 'a) *set set* **where**
finite J ⟹ $J \subseteq I$ ⟹ $X \in$ *sets* (Pi_M *J M*) ⟹ *emb I J X* ∈ *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-sets* (*space* (*PiM I M*)) *generator*
⟨*proof*⟩

definition *mu-G* (μ*G*) **where**
μ*G A* = (*THE x*. ∀ *J*⊆*I*. *finite J* ⟶ (∀ *X*∈*sets* (Pi_M *J M*). *A* = *emb I J X* ⟶ *x* = *emeasure* (*P J*) *X*))

definition *lim* :: ('i ⇒ 'a) *measure* **where**
lim = *extend-measure* (*space* (*PiM I M*)) *generator* (λ*x*. *x*) μ*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 μG (*emb I J X*) = *emeasure* (*P J*) *X*
⟨*proof*⟩

lemma *positive-mu-G*: *positive generator* μG
⟨*proof*⟩

lemma *additive-mu-G*: *additive generator* μG
⟨*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 (λ*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* λ*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 ∈ measurable (PiM {0..< n} M) (subprob-algebra (PiM {0..<Suc n} M))
⟨*proof*⟩

lemma *space-eP*:
x ∈ space (PiM {0..<n} M) ⟹ space (eP n x) = space (PiM {0..<Suc n} M)
⟨*proof*⟩

lemma *sets-eP*[*measurable*]:
x ∈ space (PiM {0..<n} M) ⟹ sets (eP n x) = sets (PiM {0..<Suc n} M)
⟨*proof*⟩

lemma *prob-space-eP*: *x ∈ space (PiM {0..<n} M) ⟹ prob-space (eP n x)*
⟨*proof*⟩

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

lemma *emeasure-eP*:
assumes *ω*[*simp*]: *ω ∈ space (PiM {0..<n} M)* **and** *A*[*measurable*]: *A ∈ sets (PiM {0..<Suc n} M)*
shows *eP n ω A = P n ω ((λx. ω(n := x)) −‘ A ∩ space (M n))*
⟨*proof*⟩

primrec *C* :: *nat ⇒ nat ⇒ (nat ⇒ 'a) ⇒ (nat ⇒ 'a) measure* **where**
C n 0 ω = return (PiM {0..<n} M) ω
| *C n (Suc m) ω = C n m ω ⤜ eP (n + m)*

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

lemma *space-C*:
x ∈ space (PiM {0..<n} M) ⟹ space (C n m x) = space (PiM {0..<n + m} M)
⟨*proof*⟩

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

lemma *prob-space-C*: *x ∈ space (PiM {0..<n} M) ⟹ prob-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 $m \leq m'$ **and** *f*[*measurable*]: $f \in$ *borel-measurable* (*PiM* {*0..<n+m*} *M*)
and *nonneg*: $\bigwedge x.\ x \in$ *space* (*PiM* {*0..<n+m*} *M*) $\Longrightarrow 0 \leq f\ x$
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-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*) (λx. *restrict x* {*0..<n+m*})
⟨*proof*⟩

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

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

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

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

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

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

definition *CI* :: *nat set* $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) *measure* **where**
CI J = *distr* (*C 0* (*up-to J*) (λx. *undefined*)) (*PiM J M*) (λf. *restrict f J*)

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

lemma *emeasure-CI'*:
finite J ⟹ *X* ∈ *sets* (*PiM J M*) ⟹ *CI J X* = *C 0* (*up-to J*) (λ-. *undefined*) (*PF.emb* {*0..<up-to J*} *J X*)
⟨*proof*⟩

lemma *emeasure-CI*:
J ⊆ {*0..<n*} ⟹ *X* ∈ *sets* (*PiM J M*) ⟹ *CI J X* = *C 0 n* (λ-. *undefined*) (*PF.emb* {*0..<n*} *J X*)
⟨*proof*⟩

lemma *lim*:
assumes *J*: *finite J* **and** *X*: *X* ∈ *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*) (λ*x*. *restrict x J*) = *CI J*
⟨*proof*⟩

end

lemma (**in** *product-prob-space*) *emeasure-lim-emb*:
assumes ∗: *finite J J* ⊆ *I X* ∈ *sets* (*PiM J M*)
shows *emeasure lim* (*emb I J X*) = *emeasure* (Pi_M *J M*) *X*
⟨*proof*⟩

end

11 Infinite Product Measure

theory *Infinite-Product-Measure*
imports *Probability-Measure Projective-Family*
begin

lemma (**in** *product-prob-space*) *distr-PiM-restrict-finite*:
assumes *finite J J* ⊆ *I*
shows *distr* (*PiM I M*) (*PiM J M*) (λ*x*. *restrict x J*) = *PiM J M*
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-PiM-emb'*:
J ⊆ *I* ⟹ *finite J* ⟹ *X* ∈ *sets* (*PiM J M*) ⟹ *emeasure* (Pi_M *I M*) (*emb I J X*) = *PiM J M X*
⟨*proof*⟩

lemma (**in** *product-prob-space*) *emeasure-PiM-emb*:
J ⊆ *I* ⟹ *finite J* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *X i* ∈ *sets* (*M i*)) ⟹

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* ⊆ *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: $i \in I\ A \in$ *sets* $(M\ i)$
shows *emeasure* $(Pi_M\ I\ M)$ $\{x{\in}$*space* $(Pi_M\ I\ M).\ x\ i \in A\}$ = *emeasure* $(M\ i)$ A
⟨*proof*⟩

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

lemma *sets-Collect-single'*:
$i \in I \Longrightarrow \{x{\in}$*space* $(M\ i).\ P\ x\} \in$ *sets* $(M\ i) \Longrightarrow \{x{\in}$*space* (*PiM I M*)$.\ P\ (x\ i)\} \in$ *sets* (*PiM I M*)
⟨*proof*⟩

lemma (**in** *finite-product-prob-space*) *finite-measure-PiM-emb*:
$(\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *sets* $(M\ i)) \Longrightarrow$ *measure* (*PiM I M*) $(Pi_E\ I\ A) = (\prod i{\in}I.$ *measure* $(M\ i)\ (A\ i))$
⟨*proof*⟩

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

lemma (**in** *product-prob-space*) *PiM-eq*:
assumes M': *sets* M' = *sets* (*PiM I M*)
assumes *eq*: $\bigwedge J\ F.$ *finite* $J \Longrightarrow J \subseteq I \Longrightarrow (\bigwedge j.\ j \in J \Longrightarrow F\ j \in$ *sets* $(M\ j)) \Longrightarrow$
emeasure M' (*prod-emb* $I\ M\ J\ (\Pi_E\ j{\in}J.\ F\ j)) = (\prod j{\in}J.$ *emeasure* $(M\ j)\ (F\ j))$

shows $M' = (PiM\ I\ M)$
⟨*proof*⟩

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

lemma *emeasure-PiM-emb*:
assumes M: $\bigwedge i.\ i \in I \Longrightarrow prob\text{-}space\ (M\ i)$
assumes J: $J \subseteq I$ *finite* J **and** A: $\bigwedge i.\ i \in J \Longrightarrow A\ i \in sets\ (M\ i)$
shows $emeasure\ (Pi_M\ I\ M)\ (prod\text{-}emb\ I\ M\ J\ (Pi_E\ J\ A)) = (\prod i{\in}J.\ emeasure\ (M\ i)\ (A\ i))$
⟨*proof*⟩

lemma *distr-pair-PiM-eq-PiM*:
fixes i' :: $'i$ **and** I :: $'i$ *set* **and** M :: $'i \Rightarrow 'a$ *measure*
assumes M: $\bigwedge i.\ i \in I \Longrightarrow prob\text{-}space\ (M\ i)\ prob\text{-}space\ (M\ i')$
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: $\bigwedge i.\ i \in K \Longrightarrow prob\text{-}space\ (M\ i)$
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: $\bigwedge i.\ i \in I \Longrightarrow prob\text{-}space\ (M\ i)$
assumes $i \in I$
shows $distr\ (Pi_M\ I\ M)\ (M\ i)\ (\lambda\omega.\ \omega\ i) = M\ i$
⟨*proof*⟩

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

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

11.1 Sequence space

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

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

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

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

lemma *measurable-comb-seq'*[*measurable* (*raw*)]:
assumes f: $f \in measurable\ N\ (\Pi_M\ i{\in}UNIV.\ M)$ **and** g: $g \in measurable\ N\ (\Pi_M\ i{\in}UNIV.\ M)$
shows $(\lambda x.\ comb\text{-}seq\ i\ (f\ x)\ (g\ x)) \in measurable\ N\ (\Pi_M\ i{\in}UNIV.\ M)$
⟨*proof*⟩

lemma *comb-seq-0*: $comb\text{-}seq\ 0\ \omega\ \omega' = \omega'$
⟨*proof*⟩

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

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

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

lemma *comb-seq-add*: $comb\text{-}seq\ n\ \omega\ \omega'\ (i + n) = \omega'\ i$
⟨*proof*⟩

lemma *case-nat-comb-seq*: $case\text{-}nat\ s'\ (comb\text{-}seq\ n\ \omega\ \omega')\ (i + n) = case\text{-}nat\ (case\text{-}nat\ s'\ \omega\ n)\ \omega'\ i$
⟨*proof*⟩

lemma *case-nat-comb-seq'*:
$case\text{-}nat\ s\ (comb\text{-}seq\ i\ \omega\ \omega') = comb\text{-}seq\ (Suc\ i)\ (case\text{-}nat\ s\ \omega)\ \omega'$
⟨*proof*⟩

locale *sequence-space* = *product-prob-space* $\lambda i.\ M\ UNIV :: nat\ set$ **for** M
begin

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

lemma *infprod-in-sets*[*intro*]:

fixes $E :: nat \Rightarrow 'a\ set$ **assumes** E: $\bigwedge i.\ E\ i \in sets\ M$
shows $Pi\ UNIV\ E \in sets\ S$
⟨*proof*⟩

lemma *measure-PiM-countable*:
fixes $E :: nat \Rightarrow 'a\ set$ **assumes** E: $\bigwedge i.\ E\ i \in sets\ M$
shows $(\lambda n.\ \prod i{\le}n.\ measure\ M\ (E\ i)) \longrightarrow measure\ S\ (Pi\ UNIV\ E)$
⟨*proof*⟩

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

lemma *PiM-comb-seq*:
$distr\ (S \bigotimes_M S)\ S\ (\lambda(\omega, \omega').\ comb\text{-}seq\ i\ \omega\ \omega') = S$ (**is** $?D = -$)
⟨*proof*⟩

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

end

end

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

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

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

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

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

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

lemma (**in** *prob-space*) *indep-eventsI*:
$(\bigwedge i.\ i \in I \Longrightarrow F\ i \in sets\ M) \Longrightarrow (\bigwedge J.\ J \subseteq I \Longrightarrow finite\ J \Longrightarrow J \neq \{\} \Longrightarrow prob\ (\bigcap i{\in}J.\ F\ i) = (\prod i{\in}J.\ prob\ (F\ i))) \Longrightarrow indep\text{-}events\ F\ I$
⟨*proof*⟩

definition (**in** *prob-space*)
$indep\text{-}event\ A\ B \longleftrightarrow indep\text{-}events\ (case\text{-}bool\ A\ B)\ UNIV$

lemma (**in** *prob-space*) *indep-sets-cong*:
$I = J \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow F\ i = G\ i) \Longrightarrow indep\text{-}sets\ F\ I \longleftrightarrow indep\text{-}sets\ G\ J$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-events-finite-index-events*:
$indep\text{-}events\ F\ I \longleftrightarrow (\forall J{\subseteq}I.\ J \neq \{\} \longrightarrow finite\ J \longrightarrow indep\text{-}events\ F\ J)$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-finite-index-sets*:
$indep\text{-}sets\ F\ I \longleftrightarrow (\forall J{\subseteq}I.\ J \neq \{\} \longrightarrow finite\ J \longrightarrow indep\text{-}sets\ F\ J)$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-mono-index*:
$J \subseteq I \Longrightarrow indep\text{-}sets\ F\ I \Longrightarrow indep\text{-}sets\ F\ J$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-mono-sets*:
assumes *indep*: $indep\text{-}sets\ F\ I$
assumes *mono*: $\bigwedge i.\ i{\in}I \Longrightarrow G\ i \subseteq F\ i$
shows $indep\text{-}sets\ G\ I$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets-mono*:
assumes *indep*: $indep\text{-}sets\ F\ I$
assumes *mono*: $J \subseteq I\ \bigwedge i.\ i{\in}J \Longrightarrow G\ i \subseteq F\ i$
shows $indep\text{-}sets\ G\ J$
⟨*proof*⟩

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

lemma (**in** *prob-space*) *indep-setsD*:
assumes $indep\text{-}sets\ F\ I$ **and** $J \subseteq I\ J \neq \{\}\ finite\ J\ \forall j{\in}J.\ A\ j \in F\ j$
shows $prob\ (\bigcap j{\in}J.\ A\ j) = (\prod j{\in}J.\ prob\ (A\ j))$
⟨*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* (λ*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* (λ*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* (λ*i*. *sigma-sets* (*space M*) (*F i*)) *I* ⟷ *indep-sets F I*
⟨*proof*⟩

definition (**in** *prob-space*)
indep-vars-def2: *indep-vars M' X I* ⟷
(∀ *i*∈*I*. *random-variable* (*M' i*) (*X i*)) ∧
indep-sets (λ*i*. { *X i* −‘ *A* ∩ *space M* | *A*. *A* ∈ *sets* (*M' i*)}) *I*

definition (**in** *prob-space*)
indep-var Ma A Mb B ⟷ *indep-vars* (*case-bool Ma Mb*) (*case-bool A B*) *UNIV*

lemma (**in** *prob-space*) *indep-vars-def*:
indep-vars M' X I ⟷
(∀ *i*∈*I*. *random-variable* (*M' i*) (*X i*)) ∧
indep-sets (λ*i*. *sigma-sets* (*space M*) { *X i* −‘ *A* ∩ *space M* | *A*. *A* ∈ *sets* (*M'* *i*)}) *I*
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-var-eq*:
indep-var S X T Y $\longleftrightarrow$
($random\text{-}variable\ S\ X \wedge random\text{-}variable\ T\ Y$) $\wedge$
indep-set
($sigma\text{-}sets\ (space\ M)\ \{\ X\ -`\ A \cap space\ M\ |\ A.\ A \in sets\ S\}$)
($sigma\text{-}sets\ (space\ M)\ \{\ Y\ -`\ A \cap space\ M\ |\ A.\ A \in sets\ T\}$)
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-sets2-eq*:
$indep\text{-}set\ A\ B \longleftrightarrow A \subseteq events \wedge B \subseteq events \wedge (\forall a{\in}A.\ \forall b{\in}B.\ prob\ (a \cap b) = prob\ a * prob\ b)$
⟨*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 $indep\text{-}events\ (\lambda i.\ \{x{\in}space\ M.\ P\ i\ (X\ i\ x)\})\ I$
⟨*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*: $indep\text{-}sets\ E\ (\bigcup j{\in}J.\ I\ j)$
assumes *Int-stable*: $\bigwedge i\ j.\ j \in J \Longrightarrow i \in I\ j \Longrightarrow Int\text{-}stable\ (E\ i)$
assumes *disjoint*: *disjoint-family-on I J*
shows $indep\text{-}sets\ (\lambda j.\ sigma\text{-}sets\ (space\ M)\ (\bigcup i{\in}I\ j.\ E\ i))\ J$
⟨*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 $indep\text{-}vars\ (\lambda j.\ PiM\ (K\ j)\ M')\ (\lambda j\ \omega.\ restrict\ (\lambda i.\ X\ i\ \omega)\ (K\ j))\ L$
⟨*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 $indep\text{-}var\ (PiM\ A\ M')\ (\lambda\omega.\ restrict\ (\lambda i.\ X\ i\ \omega)\ A)\ (PiM\ B\ M')\ (\lambda\omega.\ restrict\ (\lambda i.\ X\ i\ \omega)\ B)$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-vars-subset*:
assumes $indep\text{-}vars\ M'\ X\ I\ J \subseteq I$
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) (*UNION* $\{n..\}$ A))

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 $A :: nat \Rightarrow {'a}\ set\ set$
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* $X = 0 \lor$ *prob* $X = 1$
⟨*proof*⟩

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

lemma (**in** *prob-space*) *borel-0-1-law-AE*:
fixes $P :: nat \Rightarrow {'a} \Rightarrow bool$
assumes *indep-events* $(\lambda m.\ \{x\in$*space* $M.\ P\ m\ x\})$ *UNIV* (**is** *indep-events ?P -*)
shows (*AE x in M*. *infinite* $\{m.\ P\ m\ x\}$) $\lor$ (*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': $\bigwedge i.\ i \in I \Longrightarrow sets\ (M'\ i) = sigma\text{-}sets\ (space\ (M'\ i))\ (E\ i)$
and rv: $\bigwedge i.\ i \in I \Longrightarrow random\text{-}variable\ (M'\ i)\ (X\ i)$
and $Int\text{-}stable$: $\bigwedge i.\ i \in I \Longrightarrow Int\text{-}stable\ (E\ i)$
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 $indep\text{-}vars\ M'\ X\ I \longleftrightarrow$
$(\forall A{\in}(\Pi\ i{\in}I.\ E\ i).\ prob\ (\bigcap j{\in}I.\ X\ j\ -\text{`}\ A\ j \cap space\ M) = (\prod j{\in}I.\ prob\ (X\ j\ -\text{`}\ A\ j \cap space\ M)))$
⟨*proof*⟩

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

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

lemma (**in** *prob-space*) *indep-var-compose*:
assumes $indep\text{-}var\ M1\ X1\ M2\ X2\ Y1 \in measurable\ M1\ N1\ Y2 \in measurable\ M2\ N2$
shows $indep\text{-}var\ N1\ (Y1 \circ X1)\ N2\ (Y2 \circ X2)$
⟨*proof*⟩

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

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

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

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

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

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

lemma (**in** *prob-space*) *indep-vars-iff-distr-eq-PiM*:
fixes *I* :: *′i set* **and** *X* :: *′i ⇒ ′a ⇒ ′b*
assumes $I \neq \{\}$
assumes *rv*: $\bigwedge i.$ *random-variable* (*M′ i*) (*X i*)
shows *indep-vars M′ X I* ⟷
distr M (Π_M *i∈I. M′ i*) (*λx. λi∈I. X i x*) = (Π_M *i∈I. distr M* (*M′ i*) (*X i*))
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-varD*:
assumes *indep*: *indep-var Ma A Mb B*
assumes *sets*: *Xa ∈ sets Ma Xb ∈ sets Mb*
shows *prob* ((*λx.* (*A x*, *B x*)) −‘ (*Xa × Xb*) ∩ *space M*) =
prob (*A* −‘ *Xa* ∩ *space M*) ∗ *prob* (*B* −‘ *Xb* ∩ *space M*)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-indep-random-variable*:
assumes *ind*[*simp*]: *indep-var N X N Y*
assumes [*simp*]: *A ∈ sets N B ∈ sets N*
shows $\mathcal{P}$(*x in M. X x ∈ A ∧ Y x ∈ B*) = $\mathcal{P}$(*x in M. X x ∈ A*) ∗ $\mathcal{P}$(*x in M. Y x ∈ 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 ⟷ *random-variable S X* ∧ *random-variable T Y* ∧
distr M S X $\bigotimes_M$ *distr M T Y* = *distr M* (*S* $\bigotimes_M$ *T*) (*λx.* (*X x*, *Y x*)) (**is** - ⟷ - ∧ - ∧ *?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 $(\int^+\omega.\ (\prod i{\in}I.\ X\ i\ \omega)\ \partial M) = (\prod i{\in}I.\ \int^+\omega.\ X\ i\ \omega\ \partial M)$
⟨*proof*⟩

lemma (**in** *prob-space*)
fixes *X* :: *'i* ⇒ *'a* ⇒ *'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*: $(\int \omega.\ (\prod i{\in}I.\ X\ i\ \omega)\ \partial M) = (\prod i{\in}I.\ \int \omega.\ X\ i\ \omega\ \partial M)$ (**is** *?eq*)
and *indep-vars-integrable*: *integrable M* $(\lambda\omega.\ (\prod i{\in}I.\ X\ i\ \omega))$ (**is** *?int*)
⟨*proof*⟩

lemma (**in** *prob-space*)
fixes *X1 X2* :: *'a* ⇒ *'b*::{*real-normed-field*, *banach*, *second-countable-topology*}
assumes *indep-var borel X1 borel X2 integrable M X1 integrable M X2*
shows *indep-var-lebesgue-integral*: $(\int \omega.\ X1\ \omega * X2\ \omega\ \partial M) = (\int \omega.\ X1\ \omega\ \partial M) * (\int \omega.\ X2\ \omega\ \partial M)$ (**is** *?eq*)
and *indep-var-integrable*: *integrable M* $(\lambda\omega.\ X1\ \omega * X2\ \omega)$ (**is** *?int*)
⟨*proof*⟩

end

13 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* ⇒ *'a measure* ⇒ *'a measure* (**infix** ⋆ *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 $(\int^+ x.\ f\ x\ \partial convolution\ M\ N) = (\int^+ x.\ \int^+ y.\ f\ (x + y)\ \partial N\ \partial M)$
⟨*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* $(M \star N)\ A = \int^+ x.\ (emeasure\ N\ \{a.\ a + x \in A\})\ \partial M$
⟨*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* $(M \star N)\ A = \int^+ x.\ \int^+ y.\ (indicator\ \ A\ (x + y))\ \partial N\ \ \partial M$
⟨*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* $(M \star N)$
⟨*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* $(L \star (M \star N\ ))\ A = \int^+ x.\ \int^+ y.\ \int^+ z.\ indicator\ A\ (x + y + z)\ \partial N\ \partial M\ \partial L$
⟨*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* $((L \star M) \star N\ )\ A = \int^+ x.\ \int^+ y.\ \int^+ z.\ indicator\ A\ (x + y + z)\ \partial N\ \partial M\ \partial L$
⟨*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* ∈ *borel-measurable borel g* ∈ *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* ∂*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* ($\lambda x.$ *X x* + *Y x*) ($\lambda x.$ $\int^+ y.$ *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*]: $g \in$ *borel-measurable borel*
assumes *gt-0*[*simp*]: $\bigwedge x.\ 0 \leq f\ x\ \bigwedge x.\ 0 \leq g\ x$
assumes *prob-space* (*density lborel f*) (**is** *prob-space ?F*)
assumes *prob-space* (*density lborel g*) (**is** *prob-space ?G*)
shows *prob-space* (*density lborel* ($\lambda x.\int^{+} y.\ f\ (x - y) * g\ y\ \partial lborel$)) (**is** *prob-space ?D*)
⟨*proof*⟩

end

14 Information theory

theory *Information*
imports
Independent-Family
begin

lemma *log-le*: $1 < a \Longrightarrow 0 < x \Longrightarrow x \leq y \Longrightarrow \log a\ x \leq \log a\ y$
⟨*proof*⟩

lemma *log-less*: $1 < a \Longrightarrow 0 < x \Longrightarrow x < y \Longrightarrow \log a\ x < \log a\ y$
⟨*proof*⟩

lemma *sum-cartesian-product'*:
$(\sum x{\in}A \times B.\ f\ x) = (\sum x{\in}A.\ sum\ (\lambda y.\ f\ (x, y))\ B)$
⟨*proof*⟩

lemma *split-pairs*:
$((A, B) = X) \longleftrightarrow (fst\ X = A \wedge snd\ X = B)$ **and**
$(X = (A, B)) \longleftrightarrow (fst\ X = A \wedge snd\ X = B)$ ⟨*proof*⟩

14.1 Information theory

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

context *information-space*
begin

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

lemma *log-neg-const*:
assumes $x \leq 0$
shows $\log b\ x = \log b\ 0$
⟨*proof*⟩

lemma *log-mult-eq*:
$\log b\ (A * B) = (if\ 0 < A * B\ then\ \log b\ |A| + \log b\ |B|\ else\ \log b\ 0)$
⟨*proof*⟩

lemma *log-inverse-eq*:
  $log\ b\ (inverse\ B) = (if\ 0 < B\ then\ -\ log\ b\ B\ else\ log\ b\ 0)$
  ⟨*proof*⟩

lemma *log-divide-eq*:
  $log\ b\ (A\ /\ B) = (if\ 0 < A * B\ then\ log\ b\ |A| - log\ b\ |B|\ else\ log\ b\ 0)$
  ⟨*proof*⟩

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

end

14.2 Kullback−Leibler divergence

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

definition
  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* ⇒ *real*
  assumes $1 < b$
  assumes *f*[*measurable*]: *f* ∈ *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*) ∂*M*)
  ⟨*proof*⟩

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

lemma (**in** *information-space*) *KL-gt-0*:
  fixes *D* :: *'a* ⇒ *real*
  assumes *prob-space* (*density M D*)
  assumes *D*: *D* ∈ *borel-measurable M AE x in M*. $0 \leq D\ x$
  assumes *int*: *integrable M* (λ*x*. *D x* ∗ *log b* (*D x*))
  assumes *A*: *density M D* ≠ *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 ⇒ real*
assumes *prob-space (density M D)*
assumes *D*: *D ∈ borel-measurable M AE x in M. 0 ≤ D x*
assumes *int*: *integrable M (λx. D x ∗ log b (D x))*
shows *KL-divergence b M (density M D) = 0 ⟷ density M D = M*
⟨*proof*⟩

lemma (**in** *information-space*) *KL-eq-0-iff-eq-ac*:
fixes *D* :: *'a ⇒ real*
assumes *prob-space N*
assumes *ac*: *absolutely-continuous M N sets N = sets M*
assumes *int*: *integrable N (entropy-density b M N)*
shows *KL-divergence b M N = 0 ⟷ N = M*
⟨*proof*⟩

lemma (**in** *information-space*) *KL-nonneg*:
assumes *prob-space (density M D)*
assumes *D*: *D ∈ borel-measurable M AE x in M. 0 ≤ D x*
assumes *int*: *integrable M (λx. D x ∗ log b (D x))*
shows *0 ≤ KL-divergence b M (density M D)*
⟨*proof*⟩

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

14.3 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 ($\lambda a.$ *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 $\Longrightarrow$ *integrable S* ($\lambda x.\ Px\ x * log\ b\ (Px\ x)$)
  ⟨*proof*⟩

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

14.4 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*) (λx. (*X x*, *Y x*)))

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

abbreviation (**in** *information-space*)
mutual-information-Pow ($\mathcal{I}$′(- ; -′)) **where**
$\mathcal{I}$(*X* ; *Y*) ≡ *mutual-information b* (*count-space* (*X*‘*space M*)) (*count-space* (*Y*‘*space M*)) *X Y*

lemma (**in** *information-space*)
fixes *Pxy* :: ′*b* × ′*c* ⇒ *real* **and** *Px* :: ′*b* ⇒ *real* **and** *Py* :: ′*c* ⇒ *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*) (λx. (*X x*, *Y x*)) *Pxy*
defines *f* ≡ λx. *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* ≤ *mutual-information b S T X Y*
⟨*proof*⟩

lemma (**in** *information-space*)
fixes *Pxy* :: ′*b* × ′*c* ⇒ *real* **and** *Px* :: ′*b* ⇒ *real* **and** *Py* :: ′*c* ⇒ *real*
assumes *sigma-finite-measure S sigma-finite-measure T*
assumes *Px*: *distributed M S X Px* **and** *Px-nn*: ⋀*x*. *x* ∈ *space S* ⟹ *0* ≤ *Px x*
and *Py*: *distributed M T Y Py* **and** *Py-nn*: ⋀*y*. *y* ∈ *space T* ⟹ *0* ≤ *Py y*
and *Pxy*: *distributed M* (*S* $\bigotimes_M$ *T*) (λx. (*X x*, *Y x*)) *Pxy*
and *Pxy-nn*: ⋀*x y*. *x* ∈ *space S* ⟹ *y* ∈ *space T* ⟹ *0* ≤ *Pxy* (*x*, *y*)
defines *f* ≡ λx. *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* ⟹ *0* ≤ *mutual-information b S T X Y*
⟨*proof*⟩

lemma (**in** *information-space*)
fixes *Pxy* :: ′*b* × ′*c* ⇒ *real* **and** *Px* :: ′*b* ⇒ *real* **and** *Py* :: ′*c* ⇒ *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 $\mathcal{I}(X\ ;\ Y) = (\sum (x,\ y) \in (\lambda x.\ (X\ x,\ Y\ x))$ '*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*⟩

14.5 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 ($\mathcal{H}$'(-')) **where**
$\mathcal{H}(X) \equiv$ *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*: $\bigwedge x.\ x \in space\ MX \Longrightarrow 0 \leq f\ x$
shows *entropy-distr*: *entropy b MX X* = $-$ ($\int x.\ f\ x * log\ b\ (f\ x)\ \partial MX$) (**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*]: $\bigwedge x.\ x \in$

space MX $\Longrightarrow$ *0* $\leq$ *Px x*
and *fin*: *emeasure MX* $\{x \in$ *space MX*. *Px x* $\neq$ *0*$\}$ $\neq$ *top*
and *int*: *integrable MX* $(\lambda x. - Px\ x * log\ b\ (Px\ x))$
shows *entropy b MX X* $\leq$ *log b* (*measure MX* $\{x \in$ *space MX*. *Px x* $\neq$ *0*$\}$)
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-le-space*:
fixes *Px* :: *'b* $\Rightarrow$ *real* **and** *MX* :: *'b measure*
assumes *X*: *distributed M MX X Px* **and** *Px-nn*[*simp*]: $\bigwedge x.\ x \in$ *space MX* $\Longrightarrow$ *0* $\leq$ *Px x*
and *fin*: *finite-measure MX*
and *int*: *integrable MX* $(\lambda x. - Px\ x * log\ b\ (Px\ x))$
shows *entropy b MX X* $\leq$ *log b* (*measure MX* (*space MX*))
⟨*proof*⟩

lemma (**in** *information-space*) *entropy-uniform*:
assumes *X*: *distributed M MX X* (λx. *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 $\Longrightarrow$ $\mathcal{H}(X)$ = − ($\sum$ *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* ∩ {*x*. *f x* ≠ *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*⟩

14.6 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* (λx. (*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* $(T \bigotimes_M P)$ $(\lambda x.\ (Y\ x,\ Z\ x))$ *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* $(S \bigotimes_M P)$ $(\lambda x.\ (X\ x,\ Z\ x))$ *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* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda x.\ (X\ x,\ Y\ x,\ Z\ x))$ *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* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda(x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxyz\ (x,\ y,\ z)\ /\ (Px\ x * Pyz\ (y,\ z))))$
assumes *I2*: *integrable* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda(x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxz\ (x,\ z)\ /\ (Px\ x * Pz\ z)))$
shows *conditional-mutual-information-generic-eq*: *conditional-mutual-information b S T P X Y Z*
$= (\int (x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxyz\ (x,\ y,\ z)\ /\ (Pxz\ (x,\ z) * (Pyz\ (y,z)\ /\ Pz\ z)))\ \partial(S \bigotimes_M T \bigotimes_M P))$ (**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'*: $0 \leq$ *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 $\mathcal{I}(X\ ;\ Y\ |\ Z) =$

($\sum (x, y, z) \in (\lambda x. (X\ x, Y\ x, Z\ x))$‘*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 $0 \leq \mathcal{I}(X ; Y \mid Z)$
⟨*proof*⟩

14.7 Conditional Entropy

definition (**in** *prob-space*)
conditional-entropy b S T X Y = $-$ ($\int (x, y)$. *log b* (*enn2real* (*RN-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**
$\mathcal{H}(X \mid Y) \equiv$ *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* :: - $\Rightarrow$ *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)
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* ($S \bigotimes_M T$) (λx. *Pxy x* $*$ *log b* (*Pxy x*))
assumes *I2*: *integrable* ($S \bigotimes_M T$) (λx. *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 $\mathcal{H}(X \mid Y) = \textit{entropy } b\ (\textit{count-space } (X`\textit{space } M) \bigotimes_M \textit{count-space } (Y`\textit{space } M))\ (\lambda x.\ (X\ x,\ Y\ x)) - \mathcal{H}(Y)$
⟨*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 $\mathcal{H}(X \mid Y) = -(\sum (x, y) \in (\lambda x.\ (X\ x,\ Y\ x))\ `\ \textit{space } M.\ \textit{Pxy } (x, y) * \log b\ (\textit{Pxy } (x, y) / \textit{Py } y))$
⟨*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*⟩

14.8 Equalities

lemma (**in** *information-space*) *mutual-information-eq-entropy-conditional-entropy-distr*:
fixes $Px :: 'b \Rightarrow \textit{real}$ **and** $Py :: 'c \Rightarrow \textit{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \textit{real}$
assumes S: *sigma-finite-measure S* **and** T: *sigma-finite-measure T*
assumes *Px*[*measurable*]: *distributed M S X Px*
and *Px-nn*[*simp*]: $\bigwedge x.\ x \in \textit{space } S \Longrightarrow 0 \leq Px\ x$
and *Py*[*measurable*]: *distributed M T Y Py*
and *Py-nn*[*simp*]: $\bigwedge x.\ x \in \textit{space } T \Longrightarrow 0 \leq Py\ x$
and *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*[*simp*]: $\bigwedge x\ y.\ x \in \textit{space } S \Longrightarrow y \in \textit{space } T \Longrightarrow 0 \leq Pxy\ (x, y)$
assumes *Ix*: $\textit{integrable}(S \bigotimes_M T)\ (\lambda x.\ Pxy\ x * \log b\ (Px\ (\textit{fst } x)))$
assumes *Iy*: $\textit{integrable}(S \bigotimes_M T)\ (\lambda x.\ Pxy\ x * \log b\ (Py\ (\textit{snd } x)))$
assumes *Ixy*: $\textit{integrable}(S \bigotimes_M T)\ (\lambda x.\ Pxy\ x * \log b\ (Pxy\ x))$
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 $Px :: 'b \Rightarrow \textit{real}$ **and** $Py :: 'c \Rightarrow \textit{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \textit{real}$
assumes S: *sigma-finite-measure S* **and** T: *sigma-finite-measure T*
assumes *Px*: *distributed M S X Px* $\bigwedge x.\ x \in \textit{space } S \Longrightarrow 0 \leq Px\ x$
and *Py*: *distributed M T Y Py* $\bigwedge x.\ x \in \textit{space } T \Longrightarrow 0 \leq Py\ x$
assumes *Pxy*: *distributed M* $(S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
$\bigwedge x.\ x \in \textit{space } (S \bigotimes_M T) \Longrightarrow 0 \leq Pxy\ x$
assumes *Ix*: $\textit{integrable}(S \bigotimes_M T)\ (\lambda x.\ Pxy\ x * \log b\ (Px\ (\textit{fst } x)))$
assumes *Iy*: $\textit{integrable}(S \bigotimes_M T)\ (\lambda x.\ Pxy\ x * \log b\ (Py\ (\textit{snd } x)))$
assumes *Ixy*: $\textit{integrable}(S \bigotimes_M T)\ (\lambda x.\ Pxy\ x * \log b\ (Pxy\ x))$

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* ⇒ *real* **and** *Py* :: *'c* ⇒ *real* **and** *Pxy* :: (*'b* × *'c*) ⇒ *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 *X*: *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

15 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*⟩

15.1 Erlang

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

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

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

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

lemma *erlang-density-nonneg*[*simp*]: $0 \leq l \Longrightarrow 0 \leq$ *erlang-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-CDF k l a* = *erlang-CDF k 1* ($l * a$)
⟨*proof*⟩

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

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

lemma *emeasure-erlang-density*:
$0 < l \Longrightarrow$ *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-density k l x* $*$ *x* ^ *i*) ∂*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*]: $0 < l$ $0 \leq a$
shows $\mathcal{P}$(*x in M*. *X x* $\leq$ *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*]: $0 < l$ $0 \leq a$
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 $X[measurable]$: $X \in$ *borel-measurable M* **and** $[arith]$: $0 < l$
and *X-distr*: $\bigwedge a.\ 0 \le a \Longrightarrow$ *emeasure M* $\{x{\in}space\ M.\ X\ x \le a\}$ = *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)* $\longleftrightarrow$
$(X \in$ *borel-measurable M* $\wedge\ 0 < l\ \wedge\ (\forall\ a{\ge}0.\ \mathcal{P}(x\ in\ M.\ X\ x \le a) =$ *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*$[arith]$: $0 < \alpha \quad 0 < l$
shows *distributed M lborel* $(\lambda x.\ \alpha * X\ x)$ *(erlang-density k* $(l\ /\ \alpha))$
⟨*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* $(\lambda x.\ X\ x\ \hat{}\ i)$ $(fact\ (k + i)\ /\ (fact\ k * l\ \hat{}\ i))$
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-ith-moment-integrable*:
$0 < l \Longrightarrow$ *distributed M lborel X (erlang-density k l)* $\Longrightarrow$ *integrable M* $(\lambda x.\ X\ x\ \hat{}\ i)$
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-ith-moment*:
$0 < l \Longrightarrow$ *distributed M lborel X (erlang-density k l)* $\Longrightarrow$
expectation $(\lambda x.\ X\ x\ \hat{}\ i) = fact\ (k + i)\ /\ (fact\ k * l\ \hat{}\ i)$
⟨*proof*⟩

lemma (**in** *prob-space*) *erlang-distributed-variance*:
assumes $[arith]$: $0 < l$ **and** *distributed M lborel X (erlang-density k l)*
shows *variance X* $= (k + 1)\ /\ l^2$
⟨*proof*⟩

15.2 Exponential distribution

abbreviation *exponential-density* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
exponential-density $\equiv$ *erlang-density 0*

lemma *exponential-density-def*:
exponential-density l x $= (if\ x < 0\ then\ 0\ else\ l * exp\ (-\ x * l))$
⟨*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* ∈ *borel-measurable M* **and** [*arith*]: *0* < *l*
and *X-distr*: ⋀*a*. *0* ≤ *a* ⟹ *emeasure M* {*x*∈*space M*. *X x* ≤ *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*) ⟷
(*X* ∈ *borel-measurable M* ∧ (∀ *a*≥*0*. 𝒫(*x in M*. *X x* ≤ *a*) = *1* − *exp* (− *a* ∗ *l*)))
⟨*proof*⟩

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

lemma *exponential-density-nonneg*: *0* < *l* ⟹ *0* ≤ *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-density* k_1 l ($x - y$)) $*$ *ennreal* (*erlang-density* k_2 l y) ∂*lborel*) =
(*erlang-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*⟩

15.3 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* ∩ *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*]: $X \in$ *borel-measurable M* **and** *A*: *emeasure lborel A* = *ennreal r* $0 < r$
and [*measurable*]: $A \in$ *sets borel*
assumes *distr*: $\bigwedge a.$ *emeasure M* {*x*∈*space M*. *X x* ≤ *a*} = *emeasure lborel* (*A* ∩ {.. *a*}) / *r*
shows *distributed M lborel X* ($\lambda x.$ *indicator A x* / *measure lborel A*)
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-distrI-borel-atLeastAtMost*:
fixes *a b* :: *real*
assumes *X*: $X \in$ *borel-measurable M* **and** $a < b$
assumes *distr*: $\bigwedge t.\ a \leq t \Longrightarrow t \leq b \Longrightarrow \mathcal{P}(x\ in\ M.\ X\ x \leq t) = (t - a) / (b - a)$
shows *distributed M lborel X* ($\lambda x.$ *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* (λx. *indicator* $\{a .. b\}$ *x* / *measure lborel* $\{a .. b\}$)
assumes *t*: $a \leq t$ $t \leq b$
shows $\mathcal{P}(x\ in\ M.\ X\ x \leq t) = (t - a) / (b - a)$
⟨*proof*⟩

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

lemma (**in** *prob-space*) *uniform-distributed-iff*:
fixes *a b* :: *real*
shows *distributed M lborel X* (λx. *indicator* $\{a..b\}$ *x* / *measure lborel* $\{a..b\}$) $\longleftrightarrow$
($X \in$ *borel-measurable M* $\wedge$ $a < b$ $\wedge$ ($\forall t \in \{a .. b\}$. $\mathcal{P}(x\ in\ M.\ X\ x \leq t) = (t - a) / (b - a)$))
⟨*proof*⟩

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

15.4 Normal distribution

definition *normal-density* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
normal-density $\mu\ \sigma\ x = 1 / sqrt\ (2 * pi * \sigma^2) * exp\ (-(x - \mu)^2 / (2 * \sigma^2))$

abbreviation *std-normal-density* :: *real* $\Rightarrow$ *real* **where**
std-normal-density $\equiv$ *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*]: $0 \leq$ *normal-density* $\mu\ \sigma\ x$
⟨*proof*⟩

lemma *normal-density-pos*: $0 < \sigma \Longrightarrow 0 <$ *normal-density* $\mu\ \sigma\ x$
⟨*proof*⟩

lemma *borel-measurable-normal-density*[*measurable*]: *normal-density* $\mu\ \sigma \in$ *borel-measurable borel*
⟨*proof*⟩

lemma *gaussian-moment-0*:
has-bochner-integral lborel (λx. *indicator* {*0..*} $x *_R$ *exp* $(- x^2)$) (*sqrt pi* / *2*)
⟨*proof*⟩

lemma *gaussian-moment-1*:
has-bochner-integral lborel (λx::*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 (λx::*real*. *indicator* {*0..*} $x *_R$ (*exp* $(-x^2)*x$ˆ(*2* ∗ k)))
((*sqrt pi* / *2*) ∗ (*fact* (*2* ∗ k) / (*2* ˆ (*2* ∗ k) ∗ *fact* k)))
(**is** *?even*)
and *gaussian-moment-odd-pos*:
has-bochner-integral lborel (λx::*real*. *indicator* {*0..*} $x *_R$ (*exp* $(-x^2)*x$ˆ(*2* ∗ k + *1*))) (*fact* k / *2*)
(**is** *?odd*)
⟨*proof*⟩

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

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

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

lemma *normal-moment-odd*:
has-bochner-integral lborel (λx. *normal-density* $\mu\ \sigma\ x * (x - \mu)$ˆ(*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 $*$ $(x - \mu)$ ^k)
⟨*proof*⟩

lemma *integrable-normal-moment-abs*: *integrable lborel* (λx. *normal-density* μ σ x $*$ $|x - \mu|$ ^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* ^ (*2* ∗ *k*)) (*fact* (*2* ∗ *k*) / (*2*^*k* ∗ *fact k*))
⟨*proof*⟩

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

lemma *std-normal-moment-odd*:
has-bochner-integral lborel (λx. *std-normal-density x* ∗ *x*^(*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*]: $0 < \sigma$ $\alpha \neq 0$
shows *distributed M lborel* ($\lambda x.$ $\beta + \alpha * X\ x$) (*normal-density* ($\beta + \alpha * \mu$) ($|\alpha| * \sigma$))
⟨*proof*⟩

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

lemma *conv-normal-density-zero-mean*:
assumes [*simp*, *arith*]: $0 < \sigma$ $0 < \tau$
shows ($\lambda x.$ $\int^+ y.$ *ennreal* (*normal-density* 0 σ ($x - y$) $*$ *normal-density* 0 τ y) ∂*lborel*) =
normal-density 0 (*sqrt* ($\sigma^2 + \tau^2$)) (**is** *?LHS* = *?RHS*)
⟨*proof*⟩

lemma *conv-std-normal-density*:
($\lambda x.$ $\int^+ y.$ *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* ($\mu + \nu$) (*sqrt* ($\sigma^2 + \tau^2$)))
⟨*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* ($\mu - \nu$) (*sqrt* ($\sigma^2 + \tau^2$)))
⟨*proof*⟩

lemma (**in** *prob-space*) *sum-indep-normal*:
assumes *finite I* $I \neq \{\}$ *indep-vars* ($\lambda i.$ *borel*) *X I*
assumes $\bigwedge i.\ i \in I \Longrightarrow 0 < \sigma\ i$

assumes *normal*: $\bigwedge i.\ i \in I \Longrightarrow$ *distributed M lborel* $(X\ i)$ (*normal-density* $(\mu\ i)$ $(\sigma\ i))$
shows *distributed M lborel* $(\lambda x.\ \sum i{\in}I.\ X\ i\ x)$ (*normal-density* $(\sum i{\in}I.\ \mu\ i)$ (*sqrt* $(\sum i{\in}I.\ (\sigma\ i)^2)))$
⟨*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 $\Longrightarrow$ *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* $(2 * pi * exp\ 1 * \sigma^2)\ /\ 2$
⟨*proof*⟩

end

16 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* $(\lambda n.\ P\ (2 * n))$ *sequentially* **and** *O*: *eventually* $(\lambda n.\ P\ (2 * n + 1))$ *sequentially*
shows *eventually P sequentially*

⟨*proof*⟩

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

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

abbreviation *iexp* :: *real* ⇒ *complex* **where**
$iexp \equiv (\lambda x.\ exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ x))$

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 $(CLBINT\ x{=}a..b.\ iexp\ x) = \mathrm{i} * iexp\ a - \mathrm{i} * iexp\ b$
⟨*proof*⟩

16.2 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: $f \in borel\text{-}measurable\ M\ \bigwedge x.\ Im\ (f\ x) = 0$
shows $integrable\ M\ (\lambda x.\ exp\ (\mathrm{i} * (f\ x)))$
⟨*proof*⟩

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

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

lemma (**in** *real-distribution*) *char-measurable* [*measurable*]: $char\ M \in borel\text{-}measurable$
borel
⟨*proof*⟩

16.3 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*⟩

16.4 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* =
(∑*k* ≤ *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* = (∑*k*≤*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* − (∑*k* ≤ *n*. (i ∗ *x*)^*k* / *fact k*)) ≤ |*x*|^(*Suc n*) / *fact* (*Suc n*)
⟨*proof*⟩

lemma *iexp-approx2*: *cmod* (*iexp x* − (∑*k* ≤ *n*. (i ∗ *x*)^*k* / *fact k*)) ≤ *2* ∗ |*x*|^*n*

/ *fact n*
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-approx1*:
assumes *integrable-moments*: $\bigwedge k.\ k \le n \Longrightarrow$ *integrable M* ($\lambda x.\ x\hat{}k$)
shows *cmod* (*char M t* − ($\sum k \le n$. ((i ∗ *t*) ˆ*k* / *fact k*) ∗ *expectation* ($\lambda x.\ x\hat{}k$)))
≤
(*2*∗|*t*| ˆ*n* / *fact n*) ∗ *expectation* (λx. |*x*| ˆ*n*) (**is** *cmod* (*char M t* − *?t1*) ≤ -)
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-approx2*:
assumes *integrable-moments*: $\bigwedge k.\ k \le n \Longrightarrow$ *integrable M* (λx. *x* ˆ *k*)
shows *cmod* (*char M t* − ($\sum k \le n$. ((i ∗ *t*) ˆ*k* / *fact k*) ∗ *expectation* ($\lambda x.\ x\hat{}k$)))
≤
(|*t*| ˆ*n* / *fact* (*Suc n*)) ∗ *expectation* (λx. *min* (*2* ∗ |*x*| ˆ*n* ∗ *Suc n*) (|*t*| ∗ |*x*| ˆ*Suc n*))
(**is** *cmod* (*char M t* − *?t1*) ≤ -)
⟨*proof*⟩

lemma (**in** *real-distribution*) *char-approx3*:
fixes *t*
assumes
integrable-1: *integrable M* (λx. *x*) **and**
integral-1: *expectation* (λx. *x*) = *0* **and**
integrable-2: *integrable M* (λx. *x*ˆ*2*) **and**
integral-2: *variance* (λx. *x*) = *σ2*
shows *cmod* (*char M t* − (*1* − *t*ˆ*2* ∗ *σ2* / *2*)) ≤
(*t*ˆ*2* / *6*) ∗ *expectation* (λx. *min* (*6* ∗ *x*ˆ*2*) (*abs t* ∗ (*abs x*)ˆ*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* (λx. (*X x*) ˆ*2*) *expectation X* = *0*
and *var-X*: *variance X* = *σ2*
and *μ-def*: *μ* = *distr M borel X*
shows *cmod* (*char μ t* − (*1* − *t*ˆ*2* ∗ *σ2* / *2*)) ≤
(*t*ˆ*2* / *6*) ∗ *expectation* (λx. *min* (*6* ∗ (*X x*) ˆ*2*) (|*t*| ∗ |*X x*| ˆ*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* (λx. *X x* ˆ *k*)
and *rv-X*[*measurable*]: *random-variable borel X*

and *μ-distr* : *distr M borel X* = *μ*
shows *cmod* (*char μ t* − (∑*k* ≤ *n*. ((i ∗ *t*) ˆ*k* / *fact k*) ∗ *expectation* (λ*x*. (*X x*) ˆ*k*))) ≤
(*2* ∗ |*t*| ˆ*n* / *fact n*) ∗ *expectation* (λ*x*. |*X x*| ˆ*n*)
⟨*proof*⟩

16.5 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*ˆ(*2* ∗ *k*)) = *fact* (*2* ∗ *k*) / (*2*ˆ*k* ∗ *fact k*)
and *integrable std-normal-distribution* (λ*x*. *x*ˆ(*2* ∗ *k*))
⟨*proof*⟩

lemma *integrable-std-normal-distribution-moment*: *integrable std-normal-distribution* (λ*x*. *x*ˆ*k*)
⟨*proof*⟩

lemma *integral-std-normal-distribution-moment-odd*:
odd k ⟹ *integral*^{L} *std-normal-distribution* (λ*x*. *x*ˆ*k*) = *0*
⟨*proof*⟩

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

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

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

end

17 Helly's selection theorem

The set of bounded, monotone, right continuous functions is sequentially compact

theory *Helly-Selection*
  imports *HOL−Library.Diagonal-Subsequence Weak-Convergence*
begin

lemma *minus-one-less*: $x - 1 < (x{::}real)$
  ⟨*proof*⟩

theorem *Helly-selection*:
  fixes $f :: nat \Rightarrow real \Rightarrow real$
  assumes *rcont*: $\bigwedge n\ x.\ continuous\ (at\text{-}right\ x)\ (f\ n)$
  assumes *mono*: $\bigwedge n.\ mono\ (f\ n)$
  assumes *bdd*: $\bigwedge n\ x.\ |f\ n\ x| \le M$
  shows $\exists s.\ strict\text{-}mono\ (s{::}nat \Rightarrow nat) \wedge (\exists F.\ (\forall x.\ continuous\ (at\text{-}right\ x)\ F) \wedge mono\ F \wedge (\forall x.\ |F\ x| \le M) \wedge$
    $(\forall x.\ continuous\ (at\ x)\ F \longrightarrow (\lambda n.\ f\ (s\ n)\ x) \longrightarrow F\ x))$
⟨*proof*⟩

definition
  $tight :: (nat \Rightarrow real\ measure) \Rightarrow bool$
where
  $tight\ \mu \equiv (\forall n.\ real\text{-}distribution\ (\mu\ n)) \wedge (\forall (\varepsilon{::}real)>0.\ \exists a\ b{::}real.\ a < b \wedge (\forall n.\ measure\ (\mu\ n)\ \{a{<}..b\} > 1 - \varepsilon))$

theorem *tight-imp-convergent-subsubsequence*:
  assumes μ: $tight\ \mu\ strict\text{-}mono\ s$
  shows $\exists r\ M.\ strict\text{-}mono\ (r :: nat \Rightarrow nat) \wedge real\text{-}distribution\ M \wedge weak\text{-}conv\text{-}m\ (\mu \circ s \circ r)\ M$
⟨*proof*⟩

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

end

18 Integral of sinc

theory *Sinc-Integral*
imports *Distributions*
begin

18.1 Various preparatory integrals

Naming convention The theorem name consists of the following parts:

- Kind of integral: *has-bochner-integral* / *integrable* / *LBINT*
- Interval: Interval (0 / infinity / open / closed) (infinity / open / closed)
- Name of the occurring constants: power, exp, m (for minus), scale, sin, ...

lemma *has-bochner-integral-I0i-power-exp-m'*:
has-bochner-integral lborel (*λx. x^k ∗ exp* (*−x*) *∗ indicator* {*0* ..} *x::real*) (*fact k*)
⟨*proof*⟩

lemma *has-bochner-integral-I0i-power-exp-m*:
has-bochner-integral lborel (*λx. x^k ∗ exp* (*−x*) *∗ indicator* {*0* <..} *x::real*) (*fact k*)
⟨*proof*⟩

lemma *integrable-I0i-exp-mscale*: *0* < (*u::real*) ⟹ *set-integrable lborel* {*0* <..} (*λx. exp* (*−*(*x ∗ u*)))
⟨*proof*⟩

lemma *LBINT-I0i-exp-mscale*: *0* < (*u::real*) ⟹ *LBINT x=0..∞. exp* (*−*(*x ∗ u*)) = *1* / *u*
⟨*proof*⟩

lemma *LBINT-I0c-exp-mscale-sin*:
LBINT x=0..t. exp (*−*(*u ∗ x*)) *∗ sin x* =
(*1* / (*1* + *u^2*)) *∗* (*1* − *exp* (*−*(*u ∗ t*)) *∗* (*u ∗ sin t* + *cos t*)) (**is** - = *?F t*)
⟨*proof*⟩

lemma *LBINT-I0i-exp-mscale-sin*:
assumes *0* < *x*
shows *LBINT u=0..∞.* |*exp* (*−u ∗ x*) *∗ sin x*| = |*sin x*| / *x*
⟨*proof*⟩

lemma
shows *integrable-inverse-1-plus-square*:
set-integrable lborel (*einterval* (*−∞*) *∞*) (*λx. inverse* (*1* + *x^2*))
and *LBINT-inverse-1-plus-square*:

LBINT x=−∞..∞. inverse (1 + x^2) = pi
⟨*proof*⟩

lemma
shows *integrable-I0i-1-div-plus-square*:
interval-lebesgue-integrable lborel 0 ∞ (λx. 1 / (1 + x^2))
and *LBINT-I0i-1-div-plus-square*:
LBINT x=0..∞. 1 / (1 + x^2) = pi / 2
⟨*proof*⟩

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

abbreviation *sinc* :: *real* ⇒ *real* **where**
sinc ≡ (λ*x*. *if x* = *0 then 1 else sin x* / *x*)

lemma *sinc-at-0*: ((λ*x*. *sin x* / *x*::*real*) ⟶ *1*) (*at 0*)
⟨*proof*⟩

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

lemma *continuous-on-sinc*[*continuous-intros*]:
continuous-on S f ⟹ *continuous-on S* (λ*x*. *sinc* (*f x*))
⟨*proof*⟩

lemma *borel-measurable-sinc*[*measurable*]: *sinc* ∈ *borel-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*ˆ*2*))) ⟶ *0*) *at-top*
⟨*proof*⟩

lemma *Si-at-top-integrable*:
assumes $t \geq 0$
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*⟩

19.1 The final theorems: boundedness and scalability

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

lemma *LBINT-I0c-sin-scale-divide*:
assumes $T \geq 0$
shows *LBINT t=0..T. sin* (*t* ∗ ϑ) / *t* = *sgn* ϑ ∗ *Si* (*T* ∗ |ϑ|)
⟨*proof*⟩

end

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

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

20.1 The Levy inversion theorem

lemma *Levy-Inversion-aux1*:
fixes *a b* :: *real*
assumes $a \leq b$
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 $a \le b$ **and** $t \neq 0$
shows $cmod\ ((iexp\ (t * b) - iexp\ (t * a)) / (\mathrm{i} * t)) \le b - a$ (**is** $?F \le -$)
⟨*proof*⟩

theorem (**in** *real-distribution*) *Levy-Inversion*:
fixes $a\ b :: real$
assumes $a \le b$
defines $\mu \equiv measure\ M$ **and** $\varphi \equiv char\ M$
assumes $\mu\ \{a\} = 0$ **and** $\mu\ \{b\} = 0$
shows $(\lambda T.\ 1 / (2 * pi) * (CLBINT\ t{=}{-}T..T.\ (iexp\ (-(t * a)) - iexp\ (-(t * b))) / (\mathrm{i} * t) * \varphi\ t))$
$\longrightarrow \mu\ \{a{<}..b\}$
(**is** $(\lambda T.\ 1 / (2 * pi) * (CLBINT\ t{=}{-}T..T.\ ?F\ t * \varphi\ t)) \longrightarrow of\text{-}real\ (\mu\ \{a{<}..b\})$)
⟨*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*⟩

20.2 The Levy continuity theorem

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

end

21 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-vars* ($\lambda i.$ *borel*) *X UNIV*
and *X-integrable*: $\bigwedge n.$ *integrable M* (*X n*)
and *X-mean-0*: $\bigwedge n.$ *expectation* (*X n*) = *0*
and *σ-pos*: $\sigma > 0$
and *X-square-integrable*: $\bigwedge n.$ *integrable M* ($\lambda x.$ $(X\ n\ x)^2$)
and *X-variance*: $\bigwedge n.$ *variance* (*X n*) = σ^2
and *X-distrib*: $\bigwedge n.$ *distr M borel* (*X n*) = μ
defines $S\ n \equiv \lambda x. \sum i<n.\ X\ i\ x$
shows *weak-conv-m* ($\lambda n.$ *distr M borel* ($\lambda x.$ *S n x* / *sqrt* ($n * \sigma^2$))) *std-normal-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-vars* ($\lambda i.$ *borel*) *X UNIV*
and *X-integrable*: $\bigwedge n.$ *integrable M* (*X n*)
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*) = σ^2
and *X-distrib*: $\bigwedge n.$ *distr M borel* (*X n*) = μ
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*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-discrete* :: *'a discrete set* ⇒ *bool* **where**
  (*open*::*'a discrete set* ⇒ *bool*) *U* ⟷ (∀ *x*∈*U*. *eventually* (λ(*x'*, *y*). *x'* = *x* ⟶ *y* ∈ *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

22 Probability mass function

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

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 $(AE\ x\ in\ M.\ measure\ M\ \{x\} \neq 0) \longleftrightarrow (\exists S.\ countable\ S \wedge (AE\ x\ in\ M.\ x \in S))$
⟨*proof*⟩

22.1 PMF as measure

typedef $'a\ pmf = \{M :: 'a\ measure.\ prob\text{-}space\ M \wedge sets\ M = UNIV \wedge (AE\ x\ in\ M.\ measure\ M\ \{x\} \neq 0)\}$
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*]: $\bigwedge y.\ y \in A \Longrightarrow (\lambda x.\ f\ (x,\ y)) \in$ *measurable M L*
shows $f \in$ *measurable* ($M \bigotimes_M$ *restrict-space* (*measure-pmf N*) *A*) *L* (**is** $f \in$ *measurable ?M* -)
⟨*proof*⟩

lemma *measurable-pair-restrict-pmf1*:
assumes *countable A*
assumes [*measurable*]: $\bigwedge x.\ x \in A \Longrightarrow (\lambda y.\ f\ (x,\ y)) \in$ *measurable N L*
shows $f \in$ *measurable* (*restrict-space* (*measure-pmf M*) $A \bigotimes_M N$) *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*: ($\bigwedge y.\ y \in$ *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*: $x \in$ *set-pmf p* $\Longrightarrow$ *0* $<$ *pmf p x*
⟨*proof*⟩

lemma *pmf-nonneg*[*simp*]: *0* $\leq$ *pmf p x*
⟨*proof*⟩

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

lemma *pmf-pos* [*simp*]: *pmf p x* $\neq$ *0* $\Longrightarrow$ *pmf p x* $>$ *0*
⟨*proof*⟩

lemma *pmf-le-1*: *pmf p x* $\leq$ *1*
⟨*proof*⟩

lemma *set-pmf-not-empty*: *set-pmf M* $\neq$ {}
⟨*proof*⟩

lemma *set-pmf-iff*: $x \in$ *set-pmf M* $\longleftrightarrow$ *pmf M x* $\neq$ *0*
⟨*proof*⟩

lemma *pmf-positive-iff*: *0* $<$ *pmf p x* $\longleftrightarrow$ $x \in$ *set-pmf p*
⟨*proof*⟩

lemma *set-pmf-eq*: *set-pmf M* = {*x*. *pmf M x* $\neq$ *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* $\Longrightarrow$ *emeasure* (*measure-pmf M*) *S* = ($\sum$ *s*$\in$*S*. *pmf M s*)
⟨*proof*⟩

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

lemma *sum-pmf-eq-1*:
assumes *finite A set-pmf p* $\subseteq$ *A*
shows ($\sum$ *x*$\in$*A*. *pmf p x*) = *1*

⟨*proof*⟩

lemma *nn-integral-measure-pmf-support*:
fixes $f :: 'a \Rightarrow ennreal$
assumes f: *finite* A **and** nn: $\bigwedge x.\ x \in A \Longrightarrow 0 \leq f\ x$ $\bigwedge x.\ x \in set\text{-}pmf\ M \Longrightarrow x \notin A \Longrightarrow f\ x = 0$
shows $(\int^+ x.\ f\ x\ \partial measure\text{-}pmf\ M) = (\sum x{\in}A.\ f\ x * pmf\ M\ x)$
⟨*proof*⟩

lemma *nn-integral-measure-pmf-finite*:
fixes $f :: 'a \Rightarrow ennreal$
assumes f: *finite* $(set\text{-}pmf\ M)$ **and** nn: $\bigwedge x.\ x \in set\text{-}pmf\ M \Longrightarrow 0 \leq f\ x$
shows $(\int^+ x.\ f\ x\ \partial measure\text{-}pmf\ M) = (\sum x{\in}set\text{-}pmf\ M.\ f\ x * pmf\ M\ x)$
⟨*proof*⟩

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

lemma *integral-measure-pmf-real*:
assumes [*simp*]: *finite* A **and** $\bigwedge a.\ a \in set\text{-}pmf\ M \Longrightarrow f\ a \neq 0 \Longrightarrow a \in A$
shows $(\int x.\ f\ x\ \partial measure\text{-}pmf\ M) = (\sum a{\in}A.\ f\ a * pmf\ M\ a)$
⟨*proof*⟩

lemma *integrable-pmf*: *integrable* $(count\text{-}space\ X)\ (pmf\ M)$
⟨*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*⟩

22.2 Monad Interpretation

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

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

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

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

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

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

lemma *bind-pmf-const*[*simp*]: $\textit{bind-pmf}\ M\ (\lambda x.\ c) = c$
⟨*proof*⟩

lemma *set-bind-pmf*[*simp*]: $\textit{set-pmf}\ (\textit{bind-pmf}\ M\ N) = (\bigcup M{\in}\textit{set-pmf}\ M.\ \textit{set-pmf}\ (N\ M))$
⟨*proof*⟩

lemma *bind-pmf-cong* [*fundef-cong*]:
assumes $p = q$
shows $(\bigwedge x.\ x \in \textit{set-pmf}\ q \Longrightarrow f\ x = g\ x) \Longrightarrow \textit{bind-pmf}\ p\ f = \textit{bind-pmf}\ q\ g$
⟨*proof*⟩

lemma *bind-pmf-cong-simp*:
$p = q \Longrightarrow (\bigwedge x.\ x \in \textit{set-pmf}\ q =\!\textit{simp}\!=\!> f\ x = g\ x) \Longrightarrow \textit{bind-pmf}\ p\ f = \textit{bind-pmf}\ q\ g$
⟨*proof*⟩

lemma *measure-pmf-bind*: $\textit{measure-pmf}\ (\textit{bind-pmf}\ M\ f) = (\textit{measure-pmf}\ M \gg\!= (\lambda x.\ \textit{measure-pmf}\ (f\ x)))$
⟨*proof*⟩

lemma *nn-integral-bind-pmf* [*simp*]: $(\int^+ x.\ f\ x\ \partial bind\text{-}pmf\ M\ N) = (\int^+ x.\ \int^+ y.\ f\ y\ \partial N\ x\ \partial M)$
⟨*proof*⟩

lemma *emeasure-bind-pmf* [*simp*]: *emeasure* (*bind-pmf M N*) $X = (\int^+ x.\ emeasure\ (N\ x)\ X\ \partial M)$
⟨*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-pmf* (*map-pmf f M*) *g* = *bind-pmf M* (λ*x*. *g* (*f x*))
⟨*proof*⟩

lemma *map-pmf-transfer* [*transfer-rule*]:
rel-fun op = (*rel-fun cr-pmf cr-pmf*) (λ*f M*. *distr M* (*count-space UNIV*) *f*) *map-pmf*
⟨*proof*⟩

lemma *map-pmf-rep-eq*:
measure-pmf (*map-pmf f M*) = *distr* (*measure-pmf M*) (*count-space UNIV*) *f*
⟨*proof*⟩

lemma *map-pmf-id* [*simp*]: *map-pmf id* = *id*
⟨*proof*⟩

lemma *map-pmf-ident* [*simp*]: *map-pmf* (λ*x*. *x*) = (λ*x*. *x*)
⟨*proof*⟩

lemma *map-pmf-compose*: *map-pmf* (*f* ∘ *g*) = *map-pmf f* ∘ *map-pmf g*

⟨*proof*⟩

lemma *map-pmf-comp*: *map-pmf f* (*map-pmf g M*) = *map-pmf* (λ*x*. *f* (*g x*)) *M*
⟨*proof*⟩

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

lemma *pmf-set-map*: *set-pmf* ∘ *map-pmf f* = *op* ' *f* ∘ *set-pmf*
⟨*proof*⟩

lemma *set-map-pmf*[*simp*]: *set-pmf* (*map-pmf f M*) = *f*'*set-pmf M*
⟨*proof*⟩

lemma *emeasure-map-pmf*[*simp*]: *emeasure* (*map-pmf f M*) *X* = *emeasure M* (*f* −' *X*)
⟨*proof*⟩

lemma *measure-map-pmf*[*simp*]: *measure* (*map-pmf f M*) *X* = *measure M* (*f* −' *X*)
⟨*proof*⟩

lemma *nn-integral-map-pmf*[*simp*]: (∫⁺*x*. *f x* ∂*map-pmf g M*) = (∫⁺*x*. *f* (*g x*) ∂*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 *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* $\subseteq$ *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* $\leq$ *f x* $\Longrightarrow$ ($\int^+ x$. *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* $\longleftrightarrow$ *x* = *y*
⟨*proof*⟩

lemma *map-pmf-eq-return-pmf-iff*:
map-pmf f p = *return-pmf x* $\longleftrightarrow$ ($\forall$ *y* $\in$ *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* $\times$ *set-pmf B*
⟨*proof*⟩

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

lemma *nn-integral-pair-pmf'*: ($\int^+ x$. *f x* ∂*pair-pmf A B*) = ($\int^+ a$. $\int^+ b$. *f* (*a*, *b*) ∂B ∂A)
⟨*proof*⟩

lemma *bind-pair-pmf*:
assumes $M[measurable]$: $M \in measurable\ (count\text{-}space\ UNIV \bigotimes_M count\text{-}space\ UNIV)\ (subprob\text{-}algebra\ N)$
shows $measure\text{-}pmf\ (pair\text{-}pmf\ A\ B) \gg= M = (measure\text{-}pmf\ A \gg= (\lambda x.\ measure\text{-}pmf\ B \gg= (\lambda y.\ M\ (x,\ y))))$
(**is** $?L = ?R$)
⟨*proof*⟩

lemma *map-fst-pair-pmf*: $map\text{-}pmf\ fst\ (pair\text{-}pmf\ A\ B) = A$
⟨*proof*⟩

lemma *map-snd-pair-pmf*: $map\text{-}pmf\ snd\ (pair\text{-}pmf\ A\ B) = B$
⟨*proof*⟩

lemma *nn-integral-pmf'*:
$inj\text{-}on\ f\ A \Longrightarrow (\int^+ x.\ pmf\ p\ (f\ x)\ \partial count\text{-}space\ A) = emeasure\ p\ (f\ `\ A)$
⟨*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*: $pmf\ M\ p = 0 \longleftrightarrow p \notin set\text{-}pmf\ M$
⟨*proof*⟩

lemma *pmf-map-inj*: $inj\text{-}on\ f\ (set\text{-}pmf\ M) \Longrightarrow x \in set\text{-}pmf\ M \Longrightarrow pmf\ (map\text{-}pmf\ f\ M)\ (f\ x) = pmf\ M\ x$
⟨*proof*⟩

lemma *pmf-map-inj'*: $inj\ f \Longrightarrow pmf\ (map\text{-}pmf\ f\ M)\ (f\ x) = pmf\ M\ x$
⟨*proof*⟩

lemma *pmf-map-outside*: $x \notin f\ `\ set\text{-}pmf\ M \Longrightarrow pmf\ (map\text{-}pmf\ f\ M)\ x = 0$
⟨*proof*⟩

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

22.3 PMFs as function

context
fixes $f :: 'a \Rightarrow real$
assumes *nonneg*: $\bigwedge x.\ 0 \leq f\ x$
assumes *prob*: $(\int^+ x.\ f\ x\ \partial count\text{-}space\ UNIV) = 1$
begin

lift-definition *embed-pmf* :: *'a pmf* **is** *density* (*count-space UNIV*) (*ennreal* $\circ$ *f*)
⟨*proof*⟩

lemma *pmf-embed-pmf*: *pmf embed-pmf x* = *f x*
⟨*proof*⟩

lemma *set-embed-pmf*: *set-pmf embed-pmf* = {*x*. *f x* $\neq$ *0*}
⟨*proof*⟩

end

lemma *embed-pmf-transfer*:
rel-fun (*eq-onp* (λf. ($\forall x$. *0* $\leq$ *f x*) $\wedge$ ($\int^+ x$. *ennreal* (*f x*) ∂*count-space UNIV*) = *1*)) *pmf-as-measure.cr-pmf* (λf. *density* (*count-space UNIV*) (*ennreal* $\circ$ *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* $\Rightarrow$ *real*. ($\forall x$. *0* $\leq$ *f x*) $\wedge$ ($\int^+ x$. *ennreal* (*f x*) ∂*count-space UNIV*) = *1*}
⟨*proof*⟩

end

lemma *nn-integral-measure-pmf*: ($\int^+ x$. *f x* ∂*measure-pmf p*) = $\int^+ x$. *ennreal* (*pmf p x*) $*$ *f x* ∂*count-space UNIV*
⟨*proof*⟩

lemma *integral-measure-pmf*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
assumes *A*: *finite A*
shows ($\bigwedge a$. *a* $\in$ *set-pmf M* $\Longrightarrow$ *f a* $\neq$ *0* $\Longrightarrow$ *a* $\in$ *A*) $\Longrightarrow$ (*LINT x*|*M*. *f x*) = ($\sum a{\in}A$. *pmf M a* $*_R$ *f a*)
⟨*proof*⟩

lemma *expectation-return-pmf* [*simp*]:
fixes *f* :: *'a* $\Rightarrow$ *'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* $\Rightarrow$ *'b pmf*
and *h* :: *'b* $\Rightarrow$ *'c*::{*banach*, *second-countable-topology*}
assumes *finite A* $\bigwedge x$. *x* $\in$ *A* $\Longrightarrow$ *finite* (*set-pmf* (*f x*)) *set-pmf p* $\subseteq$ *A*
shows *measure-pmf.expectation* (*p* $\gg=$ *f*) *h* =

$(\sum a{\in}A.\ pmf\ p\ a *_R\ measure\text{-}pmf.expectation\ (f\ a)\ h)$
⟨*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 $continuous\text{-}on\ A\ (\lambda a.\ LINT\ i|M.\ f\ i\ a)$
⟨*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 $continuous\text{-}on\ UNIV\ (\lambda b.\ LBINT\ x{:}\{a..b\}.\ f\ x)$
⟨*proof*⟩

locale *pmf-as-function*
begin

setup-lifting *td-pmf-embed-pmf*

lemma *set-pmf-transfer*[*transfer-rule*]:
assumes *bi-total A*
shows $rel\text{-}fun\ (pcr\text{-}pmf\ A)\ (rel\text{-}set\ A)\ (\lambda f.\ \{x.\ f\ x \neq 0\})\ set\text{-}pmf$
⟨*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*: $bind\text{-}pmf\ A\ (\lambda x.\ bind\text{-}pmf\ B\ (C\ x)) = bind\text{-}pmf\ B\ (\lambda y.\ bind\text{-}pmf\ A\ (\lambda x.\ C\ x\ y))$
⟨*proof*⟩

lemma *pair-map-pmf1*: $pair\text{-}pmf\ (map\text{-}pmf\ f\ A)\ B = map\text{-}pmf\ (apfst\ f)\ (pair\text{-}pmf$

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*⟩

22.4 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*⟩

22.5 Relator

inductive *rel-pmf* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *'a pmf* ⇒ *'b pmf* ⇒ *bool*
for *R p q*
where
⟦ ⋀*x y*. (*x*, *y*) ∈ *set-pmf pq* ⟹ *R x y*;
map-pmf fst pq = *p*; *map-pmf snd pq* = *q* ⟧
⟹ *rel-pmf R p q*

lemma *rel-pmfI*:
  assumes *R*: *rel-set R* (*set-pmf p*) (*set-pmf q*)
  assumes *eq*: $\bigwedge x\ y.\ x \in$ *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* : *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 ($\lambda x\ y.\ P \wedge Q\ x\ y$) $x\ y \longleftrightarrow P \wedge$ *rel-pmf* $Q\ x\ y$
rel-pmf ($\lambda x\ y.\ Q\ x\ y \wedge P$) $x\ y \longleftrightarrow P \wedge$ *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) $M \longleftrightarrow (\forall a{\in}M.\ R\ x\ a)$
⟨*proof*⟩

lemma *rel-pmf-return-pmf2*: *rel-pmf* $R\ M$ (*return-pmf* x) $\longleftrightarrow (\forall a{\in}M.\ R\ a\ x)$
⟨*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* ($\lambda x\ y.$ *False*) $x\ y$ = *False*
⟨*proof*⟩

lemma *rel-pmf-rel-prod*:
rel-pmf (*rel-prod* $R\ S$) (*pair-pmf* $A\ A'$) (*pair-pmf* $B\ B'$) $\longleftrightarrow$ *rel-pmf* $R\ A\ B \wedge$ *rel-pmf* $S\ A'\ B'$
⟨*proof*⟩

lemma *rel-pmf-reflI*:
assumes $\bigwedge x.\ x \in$ *set-pmf* $p \Longrightarrow P\ x\ x$
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*: $\bigwedge x.\ x \in$ *set-pmf* $p \Longrightarrow$ *pmf* $p\ x$ = *pmf* $q\ (f\ x)$
shows *rel-pmf* ($\lambda x\ y.\ f\ x = y$) $p\ q$
⟨*proof*⟩

context
begin

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

definition *join-pmf* M = *bind-pmf* M ($\lambda x.\ x$)

lemma *bind-eq-join-pmf*: *bind-pmf* $M\ f$ = *join-pmf* (*map-pmf* $f\ M$)
⟨*proof*⟩

lemma *join-eq-bind-pmf*: *join-pmf M = bind-pmf M id*
⟨*proof*⟩

lemma *pmf-join*: *pmf (join-pmf N) i* = ($\int$ *M. pmf M i* ∂*measure-pmf N*)
⟨*proof*⟩

lemma *ennreal-pmf-join*: *ennreal* (*pmf* (*join-pmf N*) *i*) = ($\int^+$*M. pmf M i* ∂*measure-pmf N*)
⟨*proof*⟩

lemma *set-pmf-join-pmf*[*simp*]: *set-pmf (join-pmf f)* = ($\bigcup$*p*$\in$*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*: $\bigwedge$*x y. R x y* $\Longrightarrow$ *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, http://dx.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*} $\leq$ *measure q* {*y. R x y*} (**is** *?thesis1*)
and *measure-Ioi*: *measure p* {*y. R x y* $\wedge$ $\neg$ *R y x*} $\leq$ *measure q* {*y. R x y* $\wedge$ $\neg$ *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*⟩

22.6 Distributions

context
begin

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

22.6.1 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*]: $0 < p \Longrightarrow p < 1 \Longrightarrow$ *set-pmf* (*bernoulli-pmf p*) = *UNIV*
⟨*proof*⟩

lemma *nn-integral-bernoulli-pmf* [*simp*]:
assumes [*simp*]: $0 \leq p$ $p \leq 1$ $\bigwedge x.\ 0 \leq f\ x$
shows $(\int^+ x.\ f\ x\ \partial bernoulli\text{-}pmf\ p) = f\ True * p + f\ False * (1 - p)$
⟨*proof*⟩

lemma *integral-bernoulli-pmf* [*simp*]:
assumes [*simp*]: $0 \leq p$ $p \leq 1$
shows $(\int x.\ f\ x\ \partial bernoulli\text{-}pmf\ p) = f\ True * p + f\ False * (1 - p)$
⟨*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*⟩

22.6.2 Geometric Distribution

context
fixes *p* :: *real* **assumes** *p*[*arith*]: $0 < p$ $p \leq 1$
begin

lift-definition *geometric-pmf* :: *nat pmf* **is** $\lambda n.\ (1 - p)\ \hat{}\ n * p$
⟨*proof*⟩

lemma *pmf-geometric*[*simp*]: *pmf geometric-pmf n* = $(1 - p)\ \hat{}\ n * p$
⟨*proof*⟩

end

lemma *set-pmf-geometric*: $0 < p \Longrightarrow p < 1 \Longrightarrow$ *set-pmf* (*geometric-pmf p*) = *UNIV*
⟨*proof*⟩

22.6.3 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

22.6.4 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* ($S \cap A$) / *card S*
⟨*proof*⟩

lemma *measure-pmf-of-set*: *measure* (*measure-pmf pmf-of-set*) *A* = *card* ($S \cap A$) / *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 $A \neq \{\}$ *finite A* $\bigwedge x.\ x \in A \Longrightarrow$ *finite* (*set-pmf* (*f x*))
shows *measure-pmf.expectation* (*pmf-of-set A* ⤜ *f*) *h* =
($\sum a \in A.$ *measure-pmf.expectation* (*f a*) *h* $/_R$ *real* (*card A*))
⟨*proof*⟩

lemma *map-pmf-of-set*:
assumes *finite A A* $\neq$ {}
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* $\neq$ {} *finite A*
shows *pmf* (*bind-pmf* (*pmf-of-set A*) *f*) *x* =
($\sum$ *xa*$\in$*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* $\neq$ {} *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* (*UNION A f*) *A* $\neq$ {} $\bigwedge$*x*. *x* $\in$ *A* $\Longrightarrow$ *f x* $\neq$ {}
$\bigwedge$*x*. *x* $\in$ *A* $\Longrightarrow$ *card* (*f x*) = *n disjoint-family-on f A*
shows *pmf-of-set* (*UNION A f*) = *do* {*x* $\leftarrow$ *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*⟩

22.6.5 Poisson Distribution

context
fixes *rate* :: *real* **assumes** *rate-pos*: *0* < *rate*
begin

lift-definition *poisson-pmf* :: *nat pmf* **is** λ*k*. *rate* ˆ *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

22.6.6 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*⟩

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*⟩

22.7 PMFs from assiciation lists

definition *pmf-of-list* :: ('a × real) list ⇒ 'a pmf **where**
pmf-of-list xs = embed-pmf (λx. sum-list (map snd (filter (λz. 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 *⋀x. x ∈ set (map snd xs) ⟹ x ≥ 0*
assumes *sum-list (map snd xs) = 1*
shows *(∫⁺ x. ennreal (sum-list (map snd [z←xs . fst z = x])) ∂count-space UNIV) = 1*
⟨*proof*⟩

lemma *pmf-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows *pmf (pmf-of-list xs) x = sum-list (map snd (filter (λz. fst z = x) xs))*
⟨*proof*⟩

end

lemma *set-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows *set-pmf (pmf-of-list xs) ⊆ set (map fst xs)*
⟨*proof*⟩

lemma *finite-set-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows *finite (set-pmf (pmf-of-list xs))*
⟨*proof*⟩

lemma *emeasure-Int-set-pmf*:
emeasure (measure-pmf p) (A ∩ set-pmf p) = emeasure (measure-pmf p) A
⟨*proof*⟩

lemma *measure-Int-set-pmf*:
measure (measure-pmf p) (A ∩ set-pmf p) = measure (measure-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-pmf p) A = measure (measure-pmf p) B*
⟨*proof*⟩

lemma *emeasure-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows *emeasure (pmf-of-list xs) A = ennreal (sum-list (map snd (filter (λx. fst x ∈ 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 (λx. fst x ∈ A) xs))*
⟨*proof*⟩

lemma *sum-list-nonneg-eq-zero-iff*:
fixes *xs :: 'a :: linordered-ab-group-add list*
shows $(\bigwedge x.\ x \in set\ xs \Longrightarrow x \geq 0) \Longrightarrow sum\text{-}list\ xs = 0 \longleftrightarrow set\ xs \subseteq \{0\}$
⟨*proof*⟩

lemma *sum-list-filter-nonzero*:
$sum\text{-}list\ (filter\ (\lambda x.\ x \neq 0)\ xs) = sum\text{-}list\ xs$
⟨*proof*⟩

lemma *set-pmf-of-list-eq*:
assumes *pmf-of-list-wf xs* $\bigwedge x.\ x \in snd\ `\ set\ xs \Longrightarrow x > 0$
shows *set-pmf (pmf-of-list xs) = fst ‘ set xs*
⟨*proof*⟩

lemma *pmf-of-list-remove-zeros*:
assumes *pmf-of-list-wf xs*
defines $xs' \equiv filter\ (\lambda z.\ snd\ z \neq 0)\ xs$
shows *pmf-of-list-wf xs′ pmf-of-list xs′ = pmf-of-list xs*
⟨*proof*⟩

end

23 Code generation for PMFs

theory *PMF-Impl*
imports *Probability-Mass-Function HOL−Library.AList-Mapping*
begin

23.1 General code generation setup

definition *pmf-of-mapping* :: ('*a*, *real*) *mapping* ⇒ '*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*. $x \geq 0$)
shows *nn-integral* (*count-space UNIV*) (λ*k*. *ennreal* (*Mapping.lookup-default 0 m k*)) =
ennreal ($\sum$ *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*. $p \geq 0$)
assumes ($\sum$ *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*: ($\int^+$ *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 $\bigwedge x.\ x \in \mathit{Mapping.keys}\ m \Longrightarrow \mathit{Mapping.lookup\text{-}default}\ 0\ m\ x = \mathit{pmf}\ p\ x$
assumes $\mathit{Mapping.keys}\ m = \mathit{set\text{-}pmf}\ p$
shows $\mathit{mapping\text{-}of\text{-}pmf}\ p = m$
⟨*proof*⟩

lemma *return-pmf-code* [*code abstract*]:
$\mathit{mapping\text{-}of\text{-}pmf}\ (\mathit{return\text{-}pmf}\ x) = \mathit{Mapping.update}\ x\ 1\ \mathit{Mapping.empty}$
⟨*proof*⟩

lemma *pmf-of-set-code-aux*:
assumes $A \neq \{\}\ \mathit{set}\ \mathit{xs} = A\ \mathit{distinct}\ \mathit{xs}$
shows $\mathit{mapping\text{-}of\text{-}pmf}\ (\mathit{pmf\text{-}of\text{-}set}\ A) = \mathit{Mapping.tabulate}\ \mathit{xs}\ (\lambda\text{-}.\ 1\ /\ \mathit{real}\ (\mathit{length}\ \mathit{xs}))$
⟨*proof*⟩

definition *pmf-of-set-impl* **where**
$\mathit{pmf\text{-}of\text{-}set\text{-}impl}\ A = \mathit{mapping\text{-}of\text{-}pmf}\ (\mathit{pmf\text{-}of\text{-}set}\ A)$

lemma *pmf-of-set-impl-code-alt*:
assumes $A \neq \{\}\ \mathit{finite}\ A$
shows $\mathit{pmf\text{-}of\text{-}set\text{-}impl}\ A =$
$(\mathit{let}\ p = 1\ /\ \mathit{real}\ (\mathit{card}\ A)$
$\mathit{in}\ \ \mathit{Finite\text{-}Set.fold}\ (\lambda x.\ \mathit{Mapping.update}\ x\ p)\ \mathit{Mapping.empty}\ A)$
⟨*proof*⟩

lemma *pmf-of-set-impl-code* [*code*]:
$\mathit{pmf\text{-}of\text{-}set\text{-}impl}\ (\mathit{set}\ \mathit{xs}) =$
$(\mathit{if}\ \mathit{xs} = []\ \mathit{then}$
$\mathit{Code.abort}\ (\mathit{STR}\ ''\mathit{pmf\text{-}of\text{-}set\ of\ empty\ set}'')\ (\lambda\text{-}.\ \mathit{mapping\text{-}of\text{-}pmf}\ (\mathit{pmf\text{-}of\text{-}set}\ (\mathit{set}\ \mathit{xs})))$
$\mathit{else}\ \mathit{let}\ \mathit{xs}' = \mathit{remdups}\ \mathit{xs};\ p = 1\ /\ \mathit{real}\ (\mathit{length}\ \mathit{xs}')\ \mathit{in}$
$\mathit{Mapping.tabulate}\ \mathit{xs}'\ (\lambda\text{-}.\ p))$
⟨*proof*⟩

lemma *pmf-of-set-code* [*code abstract*]:
$\mathit{mapping\text{-}of\text{-}pmf}\ (\mathit{pmf\text{-}of\text{-}set}\ A) = \mathit{pmf\text{-}of\text{-}set\text{-}impl}\ A$
⟨*proof*⟩

lemma *pmf-of-multiset-pmf-of-mapping*:
assumes $A \neq \{\#\}\ \mathit{set}\ \mathit{xs} = \mathit{set\text{-}mset}\ A\ \mathit{distinct}\ \mathit{xs}$
shows $\mathit{mapping\text{-}of\text{-}pmf}\ (\mathit{pmf\text{-}of\text{-}multiset}\ A) = \mathit{Mapping.tabulate}\ \mathit{xs}\ (\lambda x.\ \mathit{count}\ A\ x\ /\ \mathit{real}\ (\mathit{size}\ A))$
⟨*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 (λx. Mapping.map-default x 0 (op + 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' (λx. 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 (op + :: real ⇒ -))

Mapping.empty

context
begin

interpretation *fold-combine-plus*: *combine-mapping-abel-semigroup op + :: 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 =*
$(\sum y{\in}A.$ *Mapping.lookup-default 0 (f y) x*)
⟨*proof*⟩ **lemma** *keys-fold-combine-plus*:
finite A ⟹ Mapping.keys (fold-combine-plus f A) = $(\bigcup x{\in}A.$ *Mapping.keys (f x))*
⟨*proof*⟩ **lemma** *fold-combine-plus-code* [*code*]:
fold-combine-plus g (set xs) = foldr (λx. Mapping.combine op+ (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-. op ∗ (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 ∈ $(\bigcup y{\in}A.$ *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) = $(\bigcup x{\in}A.$ *set-pmf (f x))*
⟨*proof*⟩

lemma *lookup-default-bind-pmf-aux*:
Mapping.lookup-default 0 (bind-pmf-aux p f A) x =
(if x ∈ $(\bigcup y{\in}A.$ *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 (λ*x*. *Mapping.map-values* (λ-. *op* ∗ (*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 (λ*x*. *Mapping.map-values* (λ-. *op* ∗ (*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**
λ*p A*. *if A* ∩ *set-pmf p* = {} *then None else*
Some (λ*x*. *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 = ($\sum$ *x*∈*C*. *pmf p x*)
in if prob = *0 then*
None
else
Some (*Mapping.map-values* (λ- *y*. *y* / *prob*)
(*Mapping.filter* (λ*k* -. *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 = ($\sum$ *x*∈*C*. *pmf p x*)
in if prob = *0 then*
None

else
Some (*Mapping.map-values* (λ- *y*. *y* / *prob*)
(*Mapping.filter* (λ*k* -. *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* ˆ *k* ∗ (*1* − *p*) ˆ
(*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*⟩

23.2 Code abbreviations for integrals and probabilities

Integrals and probabilities are defined for general measures, so we cannot give any code equations directly. We can, however, specialise these constants them to PMFs, give code equations for these specialised constants, and tell the code generator to unfold the original constants to the specialised ones whenever possible.

definition *pmf-integral* **where**
pmf-integral p f = *lebesgue-integral* (*measure-pmf p*) (*f* :: - ⇒ *real*)

definition *pmf-set-integral* **where**
pmf-set-integral p f A = *lebesgue-integral* (*measure-pmf p*) (λ*x*. *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*) = ($\sum$ *x*∈*set xs*. *pmf p x* ∗ *f x*)
⟨*proof*⟩

lemma *pmf-prob-code-alt-finite*:
finite A ⟹ *pmf-prob p A* = ($\sum$ *x*∈*A*. *pmf p x*)
⟨*proof*⟩

lemma *pmf-prob-code* [*code*]:
pmf-prob p (*set xs*) = ($\sum$ *x*∈*set xs*. *pmf p x*)
pmf-prob p (*List.coset xs*) = *1* − ($\sum$ *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*#}

definition (**in** *term-syntax*)
pmfify :: (*'a*::*typerep multiset* × (*unit* ⇒ *Code-Evaluation.term*)) ⇒
'a × (*unit* ⇒ *Code-Evaluation.term*) ⇒
'a pmf × (*unit* ⇒ *Code-Evaluation.term*) **where**
[*code-unfold*]: *pmfify A x* =

Code-Evaluation.valtermify pmf-of-multiset {·}
(*Code-Evaluation.valtermify* (*op* +) {·} *A* {·}
(*Code-Evaluation.valtermify single* {·} *x*))

notation *fcomp* (**infixl** ∘> *60*)
notation *scomp* (**infixl** ∘→ *60*)

instantiation *pmf* :: (*random*) *random*
begin

definition
Quickcheck-Random.random i =
Quickcheck-Random.random i ∘→ (λ*A*.
Quickcheck-Random.random i ∘→ (λ*x*. *Pair* (*pmfify A x*)))

instance ⟨*proof*⟩

end

no-notation *fcomp* (**infixl** ∘> *60*)
no-notation *scomp* (**infixl** ∘→ *60*)

instantiation *pmf* :: (*full-exhaustive*) *full-exhaustive*
begin

definition *full-exhaustive-pmf* :: (*'a pmf* × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒ *natural* ⇒ (*bool* × *term list*) *option*
where
full-exhaustive-pmf f i =
Quickcheck-Exhaustive.full-exhaustive (λ*A*.
Quickcheck-Exhaustive.full-exhaustive (λ*x*. *f* (*pmfify A x*)) *i*) *i*

instance ⟨*proof*⟩

end

end

24 Finite Maps

theory *Fin-Map*
imports *HOL−Analysis.Finite-Product-Measure HOL−Library.Finite-Map*
begin

The *fmap* type can be instantiated to *polish-space*, needed for the proof of projective limit. *extensional* functions are used for the representation in order to stay close to the developments of (finite) products Pi_E and their sigma-algebra Pi_M.

type-notation *fmap* ((- $\Rightarrow_F$ /-) [*22*, *21*] *21*)

unbundle *fmap.lifting*

24.1 Domain and Application

lift-definition *domain*::('*i* $\Rightarrow_F$ '*a*) $\Rightarrow$ '*i set* **is** *dom* ⟨*proof*⟩

lemma *finite-domain*[*simp*, *intro*]: *finite* (*domain P*)
⟨*proof*⟩

lift-definition *proj* :: ('*i* $\Rightarrow_F$ '*a*) $\Rightarrow$ '*i* $\Rightarrow$ '*a* ('((-)')$_F$ [*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*⟩

24.2 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 (*finmap-of inds f*)$_F$ = *restrict f 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*⟩

24.3 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**
$Pi'\ I\ A = \{\ P.\ domain\ P = I \wedge (\forall\, i.\ i \in I \longrightarrow (P)_F\ i \in A\ i)\ \}$

syntax
$\text{-}Pi'$ :: [*pttrn*, $'a\ set$, $'b\ set$] => ($'a$ => $'b$) *set* ((*3*Π′ -∈-./ -) *10*)
translations
Π′ $x{\in}A.\ B$ == *CONST* $Pi'\ A\ (\lambda x.\ B)$

24.3.1 Basic Properties of Pi'

lemma Pi'-*I*[*intro*!]: $domain\ f = A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\ x \in B\ x) \Longrightarrow f \in Pi'\ A\ B$
⟨*proof*⟩

lemma Pi'-I'[*simp*]: $domain\ f = A \Longrightarrow (\bigwedge x.\ x \in A \longrightarrow f\ x \in B\ x) \Longrightarrow f \in Pi'\ A\ B$
⟨*proof*⟩

lemma Pi'-*mem*: $f{\in}\ Pi'\ A\ B \Longrightarrow x \in A \Longrightarrow f\ x \in B\ x$
⟨*proof*⟩

lemma Pi'-*iff*: $f \in Pi'\ I\ X \longleftrightarrow domain\ f = I \wedge (\forall\, i{\in}I.\ f\ i \in X\ i)$
⟨*proof*⟩

lemma $Pi'E$ [*elim*]:
$f \in Pi'\ A\ B \Longrightarrow (f\ x \in B\ x \Longrightarrow domain\ f = A \Longrightarrow Q) \Longrightarrow (x \notin A \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

lemma *in*-Pi'-*cong*:
$domain\ f = domain\ g \Longrightarrow (\bigwedge\ w.\ w \in A \Longrightarrow f\ w = g\ w) \Longrightarrow f \in Pi'\ A\ B \longleftrightarrow g \in Pi'\ A\ B$
⟨*proof*⟩

lemma Pi'-*eq-empty*[*simp*]:
assumes *finite* A **shows** $(Pi'\ A\ B) = \{\} \longleftrightarrow (\exists\, x{\in}A.\ B\ x = \{\})$
⟨*proof*⟩

lemma Pi'-*mono*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x \subseteq C\ x) \Longrightarrow Pi'\ A\ B \subseteq Pi'\ A\ C$
⟨*proof*⟩

lemma *Pi*-Pi': *finite* $A \Longrightarrow (Pi_E\ A\ B) = proj\ `\ Pi'\ A\ B$
⟨*proof*⟩

24.4 Topological Space of Finite Maps

instantiation *fmap* :: (*type*, *topological-space*) *topological-space*
begin

definition *open-fmap* :: $('a \Rightarrow_F 'b)$ *set* $\Rightarrow$ *bool* **where**
[*code del*]: *open-fmap* = *generate-topology* $\{Pi'\ a\ b | a\ b.\ \forall i{\in}a.\ open\ (b\ i)\}$

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*⟩

24.5 Metric Space of Finite Maps

instantiation *fmap* :: (*type*, *metric-space*) *dist*
begin

definition *dist-fmap* **where**
$dist\ P\ Q = Max\ (range\ (\lambda i.\ dist\ ((P)_F\ i)\ ((Q)_F\ i))) + (if\ domain\ P = domain\ Q\ then\ 0\ else\ 1)$

instance ⟨*proof*⟩
end

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

definition [*code del*]:
$(uniformity :: (('a, 'b)\ fmap \times ('a \Rightarrow_F 'b))\ filter) =$

(*INF e*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

instance
⟨*proof*⟩
end

declare *uniformity-Abort*[**where** *'a*=(*'a* $\Rightarrow_F$ *'b*::*metric-space*), *code*]

instantiation *fmap* :: (*type*, *metric-space*) *metric-space*
begin

lemma *finite-proj-image'*: *x* ∉ *domain P* ⟹ *finite* ($(P)_F$ ' *S*)
⟨*proof*⟩

lemma *finite-proj-image*: *finite* ($(P)_F$ ' *S*)
⟨*proof*⟩

lemma *finite-proj-diag*: *finite* ((λ*i*. *d* ($(P)_F$ *i*) ($(Q)_F$ *i*)) ' *S*)
⟨*proof*⟩

lemma *dist-le-1-imp-domain-eq*:
shows *dist P Q* < *1* ⟹ *domain P* = *domain Q*
⟨*proof*⟩

lemma *dist-proj*:
shows *dist* ($(x)_F$ *i*) ($(y)_F$ *i*) ≤ *dist x y*
⟨*proof*⟩

lemma *dist-finmap-lessI*:
assumes *domain P* = *domain Q*
assumes *0* < *e*
assumes ⋀*i*. *i* ∈ *domain P* ⟹ *dist* (*P i*) (*Q i*) < *e*
shows *dist P Q* < *e*
⟨*proof*⟩

instance
⟨*proof*⟩

end

24.6 Complete Space of Finite Maps

lemma *tendsto-finmap*:
fixes *f*::*nat* ⇒ (*'i* $\Rightarrow_F$ (*'a*::*metric-space*))
assumes *ind-f*: ⋀*n*. *domain* (*f n*) = *domain g*
assumes *proj-g*: ⋀*i*. *i* ∈ *domain g* ⟹ (λ*n*. (*f n*) *i*) ⟶ *g i*
shows *f* ⟶ *g*
⟨*proof*⟩

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

24.7 Second Countable Space of Finite Maps

instantiation *fmap* :: (*countable*, *second-countable-topology*) *second-countable-topology*
begin

definition *basis-proj*::′*b set set*
  where *basis-proj* = (*SOME B*. *countable B* ∧ *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

24.8 Polish Space of Finite Maps

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

24.9 Product Measurable Space of Finite Maps

definition *PiF I M* ≡
  sigma ($\bigcup$ *J* ∈ *I*. (Π′ *j*∈*J*. *space* (*M j*))) {(Π′ *j*∈*J*. *X j*) |*X J*. *J* ∈ *I* ∧ *X* ∈ (Π *j*∈*J*. *sets* (*M j*))}

abbreviation
  Pi_F *I M* ≡ *PiF I M*

syntax
  -PiF :: *pttrn* ⇒ ′*i set* ⇒ ′*a measure* ⇒ (′*i* => ′*a*) *measure* ((*3*Π_F -∈-./ -) *10*)
translations
  Π_F *x*∈*I*. *M* == *CONST PiF I* (%*x*. *M*)

lemma *PiF-gen-subset*: {(Π′ *j*∈*J*. *X j*) |*X J*. *J* ∈ *I* ∧ *X* ∈ (Π *j*∈*J*. *sets* (*M j*))} ⊆
    Pow ($\bigcup$ *J* ∈ *I*. (Π′ *j*∈*J*. *space* (*M j*)))
  ⟨*proof*⟩

lemma *space-PiF*: *space* (*PiF I M*) = ($\bigcup$ *J* ∈ *I*. (Π′ *j*∈*J*. *space* (*M j*)))
  ⟨*proof*⟩

lemma *sets-PiF*:
  sets (*PiF I M*) = *sigma-sets* ($\bigcup$ *J* ∈ *I*. (Π′ *j*∈*J*. *space* (*M j*)))
    {(Π′ *j*∈*J*. *X j*) |*X J*. *J* ∈ *I* ∧ *X* ∈ (Π *j*∈*J*. *sets* (*M j*))}
  ⟨*proof*⟩

lemma *sets-PiF-singleton*:
  sets (*PiF* {*I*} *M*) = *sigma-sets* (Π′ *j*∈*I*. *space* (*M j*))
    {(Π′ *j*∈*I*. *X j*) |*X*. *X* ∈ (Π *j*∈*I*. *sets* (*M j*))}
  ⟨*proof*⟩

lemma *in-sets-PiFI*:
  assumes *X* = (*Pi*′ *J S*) *J* ∈ *I* ⋀*i*. *i*∈*J* ⟹ *S i* ∈ *sets* (*M i*)
  shows *X* ∈ *sets* (*PiF I M*)
  ⟨*proof*⟩

lemma *product-in-sets-PiFI*:
  assumes *J* ∈ *I* ⋀*i*. *i*∈*J* ⟹ *S i* ∈ *sets* (*M i*)
  shows (*Pi*′ *J S*) ∈ *sets* (*PiF I M*)
  ⟨*proof*⟩

lemma *singleton-space-subset-in-sets*:
  fixes *J*
  assumes *J* ∈ *I*
  assumes *finite J*

shows *space* (*PiF* {*J*} *M*) ∈ *sets* (*PiF I M*)
⟨*proof*⟩

lemma *singleton-subspace-set-in-sets*:
assumes *A*: *A* ∈ *sets* (*PiF* {*J*} *M*)
assumes *finite J*
assumes *J* ∈ *I*
shows *A* ∈ *sets* (*PiF I M*)
⟨*proof*⟩

lemma *finite-measurable-singletonI*:
assumes *finite I*
assumes ⋀*J*. *J* ∈ *I* ⟹ *finite J*
assumes *MN*: ⋀*J*. *J* ∈ *I* ⟹ *A* ∈ *measurable* (*PiF* {*J*} *M*) *N*
shows *A* ∈ *measurable* (*PiF I M*) *N*
⟨*proof*⟩

lemma *countable-finite-comprehension*:
fixes *f* :: ′*a*::*countable set* ⇒ -
assumes ⋀*s*. *P s* ⟹ *finite s*
assumes ⋀*s*. *P s* ⟹ *f s* ∈ *sets M*
shows ⋃{*f s*|*s*. *P s*} ∈ *sets M*
⟨*proof*⟩

lemma *space-subset-in-sets*:
fixes *J*::′*a*::*countable set set*
assumes *J* ⊆ *I*
assumes ⋀*j*. *j* ∈ *J* ⟹ *finite j*
shows *space* (*PiF J M*) ∈ *sets* (*PiF I M*)
⟨*proof*⟩

lemma *subspace-set-in-sets*:
fixes *J*::′*a*::*countable set set*
assumes *A*: *A* ∈ *sets* (*PiF J M*)
assumes *J* ⊆ *I*
assumes ⋀*j*. *j* ∈ *J* ⟹ *finite j*
shows *A* ∈ *sets* (*PiF I M*)
⟨*proof*⟩

lemma *countable-measurable-PiFI*:
fixes *I*::′*a*::*countable set set*
assumes *MN*: ⋀*J*. *J* ∈ *I* ⟹ *finite J* ⟹ *A* ∈ *measurable* (*PiF* {*J*} *M*) *N*
shows *A* ∈ *measurable* (*PiF I M*) *N*
⟨*proof*⟩

lemma *measurable-PiF*:
assumes *f*: ⋀*x*. *x* ∈ *space N* ⟹ *domain* (*f x*) ∈ *I* ∧ (∀ *i*∈*domain* (*f x*). (*f x*) *i* ∈ *space* (*M i*))
assumes *S*: ⋀*J S*. *J* ∈ *I* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *S i* ∈ *sets* (*M i*)) ⟹

$f -` (Pi' \; J \; S) \cap space \; N \in sets \; N$
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 $finmap\text{-}of \; J \in measurable \; (Pi_M \; J \; M) \; (PiF \; \{J\} \; M)$
⟨*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 $finmap\text{-}of \; J \in measurable \; (PiM \; J \; M) \; (PiF \; II \; M)$
⟨*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 $sets\ (Pi_M\ ?I\ ?M) = sigma\text{-}sets\ (\Pi_E\ i{\in}?I.\ space\ (?M\ i))\ \{\{f \in \Pi_E\ i{\in}?I.\ space\ (?M\ i).\ f\ i \in A\}\ |i\ A.\ i \in ?I \wedge A \in sets\ (?M\ i)\}$

lemma *sets-PiF-single*:
assumes $finite\ I\ I \neq \{\}$
shows $sets\ (PiF\ \{I\}\ M) =$
$sigma\text{-}sets\ (\Pi'\ i{\in}I.\ space\ (M\ i))$
$\{\{f{\in}\Pi'\ i{\in}I.\ space\ (M\ i).\ f\ i \in A\}\ |\ i\ A.\ i \in I \wedge A \in sets\ (M\ i)\}$
(**is** $- = sigma\text{-}sets\ ?\Omega\ ?R$)
⟨*proof*⟩

adapted from $(\bigwedge i.\ i \in ?I \Longrightarrow ?A\ i = ?B\ i) \Longrightarrow Pi_E\ ?I\ ?A = Pi_E\ ?I\ ?B$

lemma *Pi′-cong*:
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i = g\ i$
shows $Pi'\ I\ f = Pi'\ I\ g$
⟨*proof*⟩

adapted from $[\![finite\ ?I;\ \bigwedge i\ n\ m.\ [\![i \in ?I;\ n \le m]\!] \Longrightarrow ?A\ n\ i \subseteq ?A\ m\ i]\!] \Longrightarrow (\bigcup_n\ Pi\ ?I\ (?A\ n)) = (\Pi\ i{\in}?I.\ \bigcup_n\ ?A\ n\ i)$

lemma *Pi′-UN*:
fixes $A :: nat \Rightarrow 'i \Rightarrow 'a\ set$
assumes *finite I*
assumes *mono*: $\bigwedge i\ n\ m.\ i \in I \Longrightarrow n \le m \Longrightarrow A\ n\ i \subseteq A\ m\ i$
shows $(\bigcup n.\ Pi'\ I\ (A\ n)) = Pi'\ I\ (\lambda i.\ \bigcup n.\ A\ n\ i)$
⟨*proof*⟩

adapted from $[\![\bigwedge i.\ i \in ?I \Longrightarrow \exists S{\subseteq}?E\ i.\ countable\ S \wedge ?\Omega\ i = \bigcup S;\ \bigwedge i.\ i \in ?I \Longrightarrow ?E\ i \subseteq Pow\ (?\Omega\ i);\ \bigwedge j.\ j \in ?J \Longrightarrow finite\ j;\ \bigcup ?J = ?I]\!] \Longrightarrow sets\ (Pi_M\ ?I\ (\lambda i.\ sigma\ (?\Omega\ i)\ (?E\ i))) = sets\ (sigma\ (Pi_E\ ?I\ ?\Omega)\ \{\{f \in Pi_E\ ?I\ ?\Omega.\ \forall i{\in}j.\ f\ i \in A\ i\}\ |A\ j.\ j \in ?J \wedge A \in Pi\ j\ ?E\})$

lemma *sigma-fprod-algebra-sigma-eq*:
fixes $E :: 'i \Rightarrow 'a\ set\ set$ **and** $S :: 'i \Rightarrow nat \Rightarrow 'a\ set$
assumes [*simp*]: $finite\ I\ I \neq \{\}$
and *S-union*: $\bigwedge i.\ i \in I \Longrightarrow (\bigcup j.\ S\ i\ j) = space\ (M\ i)$
and *S-in-E*: $\bigwedge i.\ i \in I \Longrightarrow range\ (S\ i) \subseteq E\ i$
assumes *E-closed*: $\bigwedge i.\ i \in I \Longrightarrow E\ i \subseteq Pow\ (space\ (M\ i))$
and *E-generates*: $\bigwedge i.\ i \in I \Longrightarrow sets\ (M\ i) = sigma\text{-}sets\ (space\ (M\ i))\ (E\ i)$
defines $P == \{\ Pi'\ I\ F\ |\ F.\ \forall i{\in}I.\ F\ i \in E\ i\ \}$
shows $sets\ (PiF\ \{I\}\ M) = sigma\text{-}sets\ (space\ (PiF\ \{I\}\ M))\ P$
⟨*proof*⟩

lemma *product-open-generates-sets-PiF-single*:
assumes $I \neq \{\}$

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* $\Rightarrow_F$ '*a*::*polish-space*) *measure*) =
PiF (*Collect finite*) (λ-. *borel* :: '*a measure*)
⟨*proof*⟩

24.10 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'*: $inj\text{-}on\ (\lambda g.\ compose\ (f\ `\ J)\ g\ f')\ (extensional\ J)$
⟨*proof*⟩

lemma *inj-on-fm*:
assumes $\bigwedge i.\ space\ (M\ i) = UNIV$
shows $inj\text{-}on\ fm\ (space\ (Pi_M\ J\ M))$
⟨*proof*⟩

to measure functions

definition $mf = (\lambda g.\ compose\ J\ g\ f)\ o\ proj$

lemma *mf-fm*:
assumes $x \in space\ (Pi_M\ J\ (\lambda\text{-}.\ M))$
shows $mf\ (fm\ x) = x$
⟨*proof*⟩

lemma *mf-measurable*:
assumes $space\ M = UNIV$
shows $mf \in measurable\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))\ (PiM\ J\ (\lambda\text{-}.\ M))$
⟨*proof*⟩

lemma *fm-image-measurable*:
assumes $space\ M = UNIV$
assumes $X \in sets\ (Pi_M\ J\ (\lambda\text{-}.\ M))$
shows $fm\ `\ X \in sets\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))$
⟨*proof*⟩

lemma *fm-image-measurable-finite*:
assumes $space\ M = UNIV$
assumes $X \in sets\ (Pi_M\ J\ (\lambda\text{-}.\ M::'c\ measure))$
shows $fm\ `\ X \in sets\ (PiF\ (Collect\ finite)\ (\lambda\text{-}.\ M::'c\ measure))$
⟨*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

25 Projective Limit

theory *Projective-Limit*
imports
Fin-Map
Infinite-Product-Measure
HOL−Library.Diagonal-Subsequence
begin

25.1 Sequences of Finite Maps in Compact Sets

locale *finmap-seqs-into-compact* =
fixes *K*::*nat* ⇒ (*nat* $\Rightarrow_F$ '*a*::*metric-space*) *set* **and** *f*::*nat* ⇒ (*nat* $\Rightarrow_F$ '*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** $\bigwedge m.\ m \geq n \Longrightarrow (f\ m)_F\ t \in (\lambda k.\ (k)_F\ t)\ `\ K\ n$
⟨*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** $l \in S\ strict\text{-}mono\ (r{::}nat{\Rightarrow}nat)\ ((f \circ r) \longrightarrow l)\ sequentially$
⟨*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*⟩

25.2 Daniell-Kolmogorov Theorem

Existence of Projective Limit

locale *polish-projective* = *projective-family* $I\ P\ \lambda\text{-}.\ borel{::}'a{::}polish\text{-}space\ measure$
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* ⊆ *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

26 Random Permutations

theory *Random-Permutations*
imports
~~/*src*/*HOL*/*Probability*/*Probability-Mass-Function*
HOL−*Library.Multiset-Permutations*
begin

Choosing a set permutation (i.e. a distinct list with the same elements as the set) uniformly at random is the same as first choosing the first element of the list and then choosing the rest of the list as a permutation of the remaining set.

lemma *random-permutation-of-set*:
assumes *finite A A* ≠ {}
shows *pmf-of-set* (*permutations-of-set A*) =
do {
x ← *pmf-of-set A*;
xs ← *pmf-of-set* (*permutations-of-set* (*A* − {*x*}));
return-pmf (*x*#*xs*)
} (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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 combined 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

27 Discrete subprobability distribution

theory *SPMF* **imports**
Probability-Mass-Function
HOL−Library.Complete-Partial-Order2
HOL−Library.Rewrite
begin

27.1 Auxiliary material

lemma *cSUP-singleton* [*simp*]: (*SUP x*:{*x*}. *f x* :: - :: *conditionally-complete-lattice*) = *f x*
⟨*proof*⟩

27.1.1 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 *ennreal-enn2real-if*: $ennreal\ (enn2real\ r) = (if\ r = \top\ then\ 0\ else\ r)$
⟨*proof*⟩

lemma *e2ennreal-0* [*simp*]: $e2ennreal\ 0 = 0$
⟨*proof*⟩

lemma *enn2real-bot* [*simp*]: $enn2real\ \bot = 0$
⟨*proof*⟩

lemma *continuous-at-ennreal*[*continuous-intros*]: $continuous\ F\ f \Longrightarrow continuous\ F\ (\lambda x.\ ennreal\ (f\ x))$
⟨*proof*⟩

lemma *ennreal-Sup*:
assumes $*$: $(SUP\ a{:}A.\ ennreal\ a) \neq \top$
and $A \neq \{\}$
shows $ennreal\ (Sup\ A) = (SUP\ a{:}A.\ ennreal\ a)$
⟨*proof*⟩

lemma *ennreal-SUP*:
$[\![\ (SUP\ a{:}A.\ ennreal\ (f\ a)) \neq \top;\ A \neq \{\}\ ]\!] \Longrightarrow ennreal\ (SUP\ a{:}A.\ f\ a) = (SUP\ a{:}A.\ ennreal\ (f\ a))$
⟨*proof*⟩

lemma *ennreal-lt-0*: $x < 0 \Longrightarrow ennreal\ x = 0$
⟨*proof*⟩

27.1.2 More about *'a option*

lemma *None-in-map-option-image* [*simp*]: $None \in map\text{-}option\ f\ `\ A \longleftrightarrow None \in A$
⟨*proof*⟩

lemma *Some-in-map-option-image* [*simp*]: $Some\ x \in map\text{-}option\ f\ `\ A \longleftrightarrow (\exists y.\ x = f\ y \wedge Some\ y \in A)$
⟨*proof*⟩

lemma *case-option-collapse*: $case\text{-}option\ x\ (\lambda\text{-}.\ x) = (\lambda\text{-}.\ x)$
⟨*proof*⟩

lemma *case-option-id*: $case\text{-}option\ None\ Some = id$
⟨*proof*⟩

inductive *ord-option* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ option \Rightarrow 'b\ option \Rightarrow bool$
for *ord* :: $'a \Rightarrow 'b \Rightarrow bool$
where
None: *ord-option ord None x*
| *Some*: *ord x y* $\Longrightarrow$ *ord-option ord* (*Some x*) (*Some y*)

inductive-simps *ord-option-simps* [*simp*]:
ord-option ord None x
ord-option ord x None
ord-option ord (*Some x*) (*Some y*)
ord-option ord (*Some x*) *None*

inductive-simps *ord-option-eq-simps* [*simp*]:
ord-option op = *None y*
ord-option op = (*Some x*) *y*

lemma *ord-option-reflI*: ($\bigwedge y$. $y \in$ *set-option x* $\Longrightarrow$ *ord y y*) $\Longrightarrow$ *ord-option ord x x*
⟨*proof*⟩

lemma *reflp-ord-option*: *reflp ord* $\Longrightarrow$ *reflp* (*ord-option ord*)
⟨*proof*⟩

lemma *ord-option-trans*:
⟦ *ord-option ord x y*; *ord-option ord y z*;
$\bigwedge a\ b\ c$. ⟦ $a \in$ *set-option x*; $b \in$ *set-option y*; $c \in$ *set-option z*; *ord a b*; *ord b c* ⟧ $\Longrightarrow$ *ord a c* ⟧
$\Longrightarrow$ *ord-option ord x z*
⟨*proof*⟩

lemma *transp-ord-option*: *transp ord* $\Longrightarrow$ *transp* (*ord-option ord*)
⟨*proof*⟩

lemma *antisymp-ord-option*: *antisymp ord* $\Longrightarrow$ *antisymp* (*ord-option ord*)
⟨*proof*⟩

lemma *ord-option-chainD*:
Complete-Partial-Order.chain (*ord-option ord*) *Y*
$\Longrightarrow$ *Complete-Partial-Order.chain ord* $\{x.$ *Some x* $\in Y\}$
⟨*proof*⟩

definition *lub-option* :: $('a\ set \Rightarrow 'b) \Rightarrow 'a\ option\ set \Rightarrow 'b\ option$
where *lub-option lub Y* = (*if* $Y \subseteq \{None\}$ *then None else Some* (*lub* $\{x.$ *Some x* $\in Y\}$))

lemma *map-lub-option*: *map-option f* (*lub-option lub Y*) = *lub-option* ($f \circ lub$) *Y*
⟨*proof*⟩

lemma *lub-option-upper*:

assumes *Complete-Partial-Order.chain* (*ord-option ord*) Y $x \in Y$
and *lub-upper*: $\bigwedge Y\ x.$ ⟦ *Complete-Partial-Order.chain ord* Y; $x \in Y$ ⟧ $\Longrightarrow$ *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* $\circ$ *op* ‘ 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* ($\lambda x\ y.$ *ord* x (f y)) x y
⟨*proof*⟩

lemma *ord-option-map1*: *ord-option ord* (*map-option* f x) y = *ord-option* ($\lambda x\ y.$ *ord* (f x) y) x y
⟨*proof*⟩

lemma *option-ord-Some1-iff*: *option-ord* (*Some* x) $y \longleftrightarrow y =$ *Some* x
⟨*proof*⟩

27.1.3 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* ===> *op* =) (λ*x*. *x* ∈ *A*) (λ*y*. *y* ∈ *B*)

lemma *rel-predI*: (*R* ===> *op* =) (λ*x*. *x* ∈ *A*) (λ*y*. *y* ∈ *B*) ⟹ *rel-pred R A B*
⟨*proof*⟩

lemma *rel-predD*: ⟦ *rel-pred R A B*; *R x y* ⟧ ⟹ *x* ∈ *A* ⟷ *y* ∈ *B*
⟨*proof*⟩

lemma *Collect-parametric*: ((*A* ===> *op* =) ===> *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

27.1.4 Monotonicity rules

lemma *monotone-gfp-eadd1*: *monotone op* ≥ *op* ≥ (λ*x*. *x* + *y* :: *enat*)
⟨*proof*⟩

lemma *monotone-gfp-eadd2*: *monotone op* ≥ *op* ≥ (λ*y*. *x* + *y* :: *enat*)
⟨*proof*⟩

lemma *mono2mono-gfp-eadd*[*THEN gfp.mono2mono2*, *cont-intro*, *simp*]:
shows *monotone-eadd*: *monotone* (*rel-prod op* ≥ *op* ≥) *op* ≥ (λ(*x*, *y*). *x* + *y* :: *enat*)
⟨*proof*⟩

lemma *eadd-gfp-partial-function-mono* [*partial-function-mono*]:
⟦ *monotone* (*fun-ord op* ≥) *op* ≥ *f*; *monotone* (*fun-ord op* ≥) *op* ≥ *g* ⟧
⟹ *monotone* (*fun-ord op* ≥) *op* ≥ (λ*x*. *f x* + *g x* :: *enat*)
⟨*proof*⟩

lemma *mono2mono-ereal*[*THEN lfp.mono2mono*]:
shows *monotone-ereal*: *monotone op* ≤ *op* ≤ *ereal*
⟨*proof*⟩

lemma *mono2mono-ennreal*[*THEN lfp.mono2mono*]:
shows *monotone-ennreal*: *monotone op* ≤ *op* ≤ *ennreal*
⟨*proof*⟩

27.1.5 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* $\Longrightarrow$ *rel-set* $(\lambda x\ y.\ y = f\ x)$ *A B*
⟨*proof*⟩

27.2 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* $\neq \bot$
⟨*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*: ($\int^+$ *x*. *f x* ∂*measure-spmf p*) = $\int^+$ *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 ($\int$ *x*. *f x* ∂*measure-spmf p*) = $\int$ *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*: ($\int^+$ *x*. *spmf p x* ∂*count-space UNIV*) ≠ ⊤
⟨*proof*⟩

lemma *SUP-spmf-neq-top'*: $(SUP\ p{:}Y.\ ennreal\ (spmf\ p\ x)) \neq \top$
⟨*proof*⟩

lemma *SUP-spmf-neq-top*: $(SUP\ i.\ ennreal\ (spmf\ (Y\ i)\ x)) \neq \top$
⟨*proof*⟩

lemma *SUP-emeasure-spmf-neq-top*: $(SUP\ p{:}Y.\ emeasure\ (measure\text{-}spmf\ p)\ A) \neq \top$
⟨*proof*⟩

27.3 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'*:

$(\int^+ x.\ f\ x\ \partial measure\text{-}spmf\ p) = \int^+ x.\ ennreal\ (spmf\ p\ x) * f\ x\ \partial count\text{-}space\ (set\text{-}spmf\ p)$
⟨*proof*⟩

27.4 Functorial structure

abbreviation $map\text{-}spmf :: ('a \Rightarrow 'b) \Rightarrow 'a\ spmf \Rightarrow 'b\ spmf$
where $map\text{-}spmf\ f \equiv map\text{-}pmf\ (map\text{-}option\ f)$

context begin
⟨*ML*⟩

lemma *map-comp*: $map\text{-}spmf\ f\ (map\text{-}spmf\ g\ p) = map\text{-}spmf\ (f \circ g)\ p$
⟨*proof*⟩

lemma *map-id0*: $map\text{-}spmf\ id = id$
⟨*proof*⟩

lemma *map-id* [*simp*]: $map\text{-}spmf\ id\ p = p$
⟨*proof*⟩

lemma *map-ident* [*simp*]: $map\text{-}spmf\ (\lambda x.\ x)\ p = p$
⟨*proof*⟩

end

lemma *set-map-spmf* [*simp*]: $set\text{-}spmf\ (map\text{-}spmf\ f\ p) = f\ `\ set\text{-}spmf\ p$
⟨*proof*⟩

lemma *map-spmf-cong*:
$[\![p = q; \bigwedge x.\ x \in set\text{-}spmf\ q \Longrightarrow f\ x = g\ x]\!]$
$\Longrightarrow map\text{-}spmf\ f\ p = map\text{-}spmf\ g\ q$
⟨*proof*⟩

lemma *map-spmf-cong-simp*:
$[\![p = q; \bigwedge x.\ x \in set\text{-}spmf\ q =simp=> f\ x = g\ x]\!]$
$\Longrightarrow map\text{-}spmf\ f\ p = map\text{-}spmf\ g\ q$
⟨*proof*⟩

lemma *map-spmf-idI*: $(\bigwedge x.\ x \in set\text{-}spmf\ p \Longrightarrow f\ x = x) \Longrightarrow map\text{-}spmf\ f\ p = p$
⟨*proof*⟩

lemma *emeasure-map-spmf*:
$emeasure\ (measure\text{-}spmf\ (map\text{-}spmf\ f\ p))\ A = emeasure\ (measure\text{-}spmf\ p)\ (f -`\ A)$
⟨*proof*⟩

lemma *measure-map-spmf*: $measure\ (measure\text{-}spmf\ (map\text{-}spmf\ f\ p))\ A = measure\ (measure\text{-}spmf\ p)\ (f -`\ A)$

⟨*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)*; $x \in$ *set-spmf M* ⟧ $\Longrightarrow$ *spmf (map-spmf f M) (f x) = spmf M x*
⟨*proof*⟩

lemma *spmf-map-inj'*: *inj f* $\Longrightarrow$ *spmf (map-spmf f M) (f x) = spmf M x*
⟨*proof*⟩

lemma *spmf-map-outside*: $x \notin f$ ‘ *set-spmf M* $\Longrightarrow$ *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*⟩

27.5 Monad operations

27.5.1 Return

abbreviation *return-spmf* :: $'a \Rightarrow$ *'a spmf*
where *return-spmf x* $\equiv$ *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*⟩

27.5.2 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* ⤜ (*f* :: *'a* ⇒ -) = *return-pmf None*
⟨*proof*⟩

lemma *return-bind-spmf* [*simp*]: *return-spmf x* ⤜ *f* = *f x*
⟨*proof*⟩

lemma *bind-return-spmf* [*simp*]: *x* ⤜ *return-spmf* = *x*
⟨*proof*⟩

lemma *bind-spmf-assoc* [*simp*]:
fixes *x* :: *'a spmf* **and** *f* :: *'a* ⇒ *'b spmf* **and** *g* :: *'b* ⇒ *'c spmf*
shows (*x* ⤜ *f*) ⤜ *g* = *x* ⤜ (λ*y*. *f y* ⤜ *g*)
⟨*proof*⟩

lemma *pmf-bind-spmf-None*: *pmf* (*p* ⤜ *f*) *None* = *pmf p None* + ∫ *x*. *pmf* (*f x*) *None* ∂*measure-spmf p*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *spmf-bind*: *spmf* (*p* ⤜ *f*) *y* = ∫ *x*. *spmf* (*f x*) *y* ∂*measure-spmf p*
⟨*proof*⟩

lemma *ennreal-spmf-bind*: *ennreal* (*spmf* (*p* ⤜ *f*) *x*) = ∫⁺ *y*. *spmf* (*f y*) *x* ∂*measure-spmf p*
⟨*proof*⟩

lemma *measure-spmf-bind-pmf*: *measure-spmf* (*p* ⤜ *f*) = *measure-pmf p* ⤜ *measure-spmf* ∘ *f*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *measure-spmf-bind*: *measure-spmf* (*p* ⤜ *f*) = *measure-spmf p* ⤜ *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 $\bigwedge y.\ y \in$ *set-spmf p* ⟹ *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* $\gg=$ (λ*x*. *return-spmf* (*f x*)))
⟨*proof*⟩

lemma *bind-spmf-cong*:
⟦ *p* = *q*; $\bigwedge x.\ x \in$ *set-spmf q* ⟹ *f x* = *g x* ⟧
⟹ *bind-spmf p f* = *bind-spmf q g*
⟨*proof*⟩

lemma *bind-spmf-cong-simp*:
⟦ *p* = *q*; $\bigwedge x.\ x \in$ *set-spmf q* =*simp*=> *f x* = *g x* ⟧
⟹ *bind-spmf p f* = *bind-spmf q g*
⟨*proof*⟩

lemma *set-bind-spmf*: *set-spmf* (*M* $\gg=$ *f*) = *set-spmf M* $\gg=$ (*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*⟩

27.6 Relator

abbreviation *rel-spmf* :: (′*a* ⇒ ′*b* ⇒ *bool*) ⇒ ′*a spmf* ⇒ ′*b spmf* ⇒ *bool*
where *rel-spmf R* ≡ *rel-pmf* (*rel-option R*)

lemma *rel-pmf-mono*:
⟦*rel-pmf A f g*; $\bigwedge x\ y.\ A\ x\ y \Longrightarrow B\ x\ y$ ⟧ ⟹ *rel-pmf B f g*
⟨*proof*⟩

lemma *rel-spmf-mono*:
⟦*rel-spmf A f g*; $\bigwedge x\ y.\ A\ x\ y \Longrightarrow B\ x\ y$ ⟧ ⟹ *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**
⋀*x y*. (*x*, *y*) ∈ *set-spmf pq* ⟹ *P x y*
p = *map-spmf fst pq*
q = *map-spmf snd pq*
⟨*proof*⟩

lemma *rel-spmf-simps*:
rel-spmf R p q ⟷ (∃ *pq*. (∀ (*x*, *y*)∈*set-spmf pq*. *R x y*) ∧ *map-spmf fst pq* = *p* ∧ *map-spmf snd pq* = *q*)
⟨*proof*⟩

lemma *spmf-rel-map*:
shows *spmf-rel-map1*: ⋀*R f x*. *rel-spmf R* (*map-spmf f x*) = *rel-spmf* (λ*x*. *R* (*f x*)) *x*
and *spmf-rel-map2*: ⋀*R x g y*. *rel-spmf R x* (*map-spmf g y*) = *rel-spmf* (λ*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 op* = = *op* =
⟨*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* ===> *op* =) ===> *rel-spmf A* ===> *rel-spmf B* ===> *op* =) *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*:
$\llbracket$ *rel-spmf R p q*; $\bigwedge x\ y.\ R\ x\ y \Longrightarrow$ *rel-spmf P* (*f x*) (*g y*) $\rrbracket$
$\Longrightarrow$ *rel-spmf P* ($p \gg= f$) ($q \gg= g$)
⟨*proof*⟩

lemma *rel-spmf-bind-reflI*:
($\bigwedge x.\ x \in$ *set-spmf p* $\Longrightarrow$ *rel-spmf P* (*f x*) (*g x*)) $\Longrightarrow$ *rel-spmf P* ($p \gg= f$) ($p \gg= g$)
⟨*proof*⟩

lemma *rel-pmf-return-pmfI*: *P x y* $\Longrightarrow$ *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* ===> *op* =) (λp. *measure* (*measure-pmf p*)) (λq. *measure* (*measure-pmf q*))
⟨*proof*⟩

lemma *measure-spmf-parametric*:
(*rel-spmf A* ===> *rel-pred A* ===> *op* =) (λp. *measure* (*measure-spmf p*)) (λq. *measure* (*measure-spmf q*))
⟨*proof*⟩

end

27.7 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*⟩

27.8 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-le-0*: *weight-spmf p* ≤ *0* ⟷ *weight-spmf p* = *0*
⟨*proof*⟩

lemma *weight-spmf-lt-0*: ¬ *weight-spmf p* < *0*
⟨*proof*⟩

lemma *spmf-le-weight*: $spmf\ p\ x \leq weight\text{-}spmf\ p$
⟨*proof*⟩

lemma *weight-spmf-eq-0*: $weight\text{-}spmf\ p = 0 \longleftrightarrow p = return\text{-}pmf\ None$
⟨*proof*⟩

lemma *weight-bind-spmf*: $weight\text{-}spmf\ (x \gg= f) = lebesgue\text{-}integral\ (measure\text{-}spmf\ x)\ (weight\text{-}spmf \circ f)$
⟨*proof*⟩

lemma *rel-spmf-weightD*: $rel\text{-}spmf\ A\ p\ q \Longrightarrow weight\text{-}spmf\ p = weight\text{-}spmf\ q$
⟨*proof*⟩

lemma *rel-spmf-bij-betw*:
  assumes *f*: $bij\text{-}betw\ f\ (set\text{-}spmf\ p)\ (set\text{-}spmf\ q)$
  and *eq*: $\bigwedge x.\ x \in set\text{-}spmf\ p \Longrightarrow spmf\ p\ x = spmf\ q\ (f\ x)$
  shows $rel\text{-}spmf\ (\lambda x\ y.\ f\ x = y)\ p\ q$
⟨*proof*⟩

27.9 From density to spmfs

context fixes $f :: 'a \Rightarrow real$ **begin**

definition $embed\text{-}spmf :: 'a\ spmf$
where $embed\text{-}spmf = embed\text{-}pmf\ (\lambda x.\ case\ x\ of\ None \Rightarrow 1 - enn2real\ (\int^+ x.\ ennreal\ (f\ x)\ \partial count\text{-}space\ UNIV)\ |\ Some\ x' \Rightarrow max\ 0\ (f\ x'))$

context
  assumes *prob*: $(\int^+ x.\ ennreal\ (f\ x)\ \partial count\text{-}space\ UNIV) \leq 1$
begin

lemma *nn-integral-embed-spmf-eq-1*:
  $(\int^+ x.\ ennreal\ (case\ x\ of\ None \Rightarrow 1 - enn2real\ (\int^+ x.\ ennreal\ (f\ x)\ \partial count\text{-}space\ UNIV)\ |\ Some\ x' \Rightarrow max\ 0\ (f\ x'))\ \partial count\text{-}space\ UNIV) = 1$
  (**is** $?lhs = -$ **is** $(\int^+ x.\ ?f\ x\ \partial ?M) = -$)
⟨*proof*⟩

lemma *pmf-embed-spmf-None*: $pmf\ embed\text{-}spmf\ None = 1 - enn2real\ (\int^+ x.\ ennreal\ (f\ x)\ \partial count\text{-}space\ UNIV)$
⟨*proof*⟩

lemma *spmf-embed-spmf* [*simp*]: $spmf\ embed\text{-}spmf\ x = max\ 0\ (f\ x)$
⟨*proof*⟩

end

end

lemma *embed-spmf-K-0*[*simp*]: *embed-spmf* (λ-. *0*) = *return-pmf None* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

27.10 Ordering on spmfs

rel-pmf does not preserve a ccpo structure. Counterexample by Saheb-Djahromi: Take prefix order over *bool llist* and the set *range* (λ*n* :: *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* ⇒ *'a* ⇒ *bool*) ⇒ *'a spmf* ⇒ *'a spmf* ⇒ *bool*
where *ord-spmf ord* ≡ *rel-pmf* (*ord-option ord*)

locale *ord-spmf-syntax* **begin**
notation *ord-spmf* (**infix** ⊑1 *60*)
end

lemma *ord-spmf-map-spmf1*: *ord-spmf R* (*map-spmf f p*) = *ord-spmf* (λ*x*. *R* (*f x*)) *p*
⟨*proof*⟩

lemma *ord-spmf-map-spmf2*: *ord-spmf R p* (*map-spmf f q*) = *ord-spmf* (λ*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* (λ*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* ⇒ *'a* ⇒ *bool* (**structure**) **begin**
interpretation *ord-spmf-syntax* ⟨*proof*⟩

lemma *ord-spmfI*:
⟦ ⋀*x y*. (*x*, *y*) ∈ *set-spmf pq* ⟹ *ord x y*; *map-spmf fst pq* = *p*; *map-spmf snd pq* = *q* ⟧
⟹ *p* ⊑ *q*

⟨*proof*⟩

lemma *ord-spmf-None* [*simp*]: *return-pmf None* $\sqsubseteq$ *x*
⟨*proof*⟩

lemma *ord-spmf-reflI*: ($\bigwedge x.\ x \in$ *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*) $\longleftrightarrow$ ($\forall x \in$*set-spmf p*. *R x y*)
⟨*proof*⟩

lemma *ord-spmf-mono*: ⟦ *ord-spmf A p q*; $\bigwedge x\ y.$ *A x y* $\Longrightarrow$ *B x y* ⟧ $\Longrightarrow$ *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*: $\bigwedge x\ y.$ *R x y* $\Longrightarrow$ *ord-spmf P* (*f x*) (*g y*)
shows *ord-spmf P* (*p* $\gg=$ *f*) (*q* $\gg=$ *g*)
⟨*proof*⟩

lemma *ord-spmf-bind-reflI*:
($\bigwedge x.\ x \in$ *set-spmf p* $\Longrightarrow$ *ord-spmf R* (*f x*) (*g x*))
$\Longrightarrow$ *ord-spmf R* (*p* $\gg=$ *f*) (*p* $\gg=$ *g*)
⟨*proof*⟩

lemma *ord-pmf-increaseI*:
assumes *le*: $\bigwedge x.$ *spmf p x* $\leq$ *spmf q x*
and *refl*: $\bigwedge x.\ x \in$ *set-spmf p* $\Longrightarrow$ *R x x*
shows *ord-spmf R p q*
⟨*proof*⟩

lemma *ord-spmf-eq-leD*:
assumes *ord-spmf op* = *p q*
shows *spmf p x* $\leq$ *spmf q x*

⟨*proof*⟩

lemma *ord-spmf-eqD-set-spmf*: *ord-spmf op* = *p q* ⟹ *set-spmf p* ⊆ *set-spmf q*
⟨*proof*⟩

lemma *ord-spmf-eqD-emeasure*:
ord-spmf op = *p q* ⟹ *emeasure* (*measure-spmf p*) *A* ≤ *emeasure* (*measure-spmf q*) *A*
⟨*proof*⟩

lemma *ord-spmf-eqD-measure-spmf*: *ord-spmf op* = *p q* ⟹ *measure-spmf p* ≤ *measure-spmf q*
⟨*proof*⟩

27.11 CCPO structure for the flat ccpo *ord-option op* =

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 op* =) *Y* **begin**

lemma *chain-ord-spmf-eqD*: *Complete-Partial-Order.chain* (*op* ≤) ((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 op* = *p q*
and *None*: *pmf p None* = *pmf q None*
shows *p* = *q*
⟨*proof*⟩

lemma *ord-spmf-eqD-pmf-None*:
assumes *ord-spmf op* = *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 $Y \neq \{\}$
shows *spmf lub-spmf x* = (*SUP p* : *Y*. *spmf p x*)
⟨*proof*⟩

lemma *ennreal-spmf-lub-spmf*: $Y \neq \{\} \Longrightarrow$ *ennreal* (*spmf lub-spmf x*) = (*SUP p*:*Y*. *ennreal* (*spmf p x*))
⟨*proof*⟩

lemma *lub-spmf-upper*:
assumes *p*: $p \in Y$
shows *ord-spmf op* = *p lub-spmf*
⟨*proof*⟩

lemma *lub-spmf-least*:
assumes *z*: $\bigwedge x.\ x \in Y \Longrightarrow$ *ord-spmf op* = *x z*
shows *ord-spmf op* = *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*:*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*:*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*:*Y*. *weight-spmf y*)
⟨*proof*⟩

lemma *measure-spmf-lub-spmf*:
assumes *Y*: $Y \neq \{\}$
shows *measure-spmf lub-spmf* = (*SUP p*:*Y*. *measure-spmf p*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

end

lemma *partial-function-definitions-spmf*: *partial-function-definitions* (*ord-spmf op* =) *lub-spmf*
(**is** *partial-function-definitions ?R* -)
⟨*proof*⟩

lemma *ccpo-spmf*: *class.ccpo lub-spmf* (*ord-spmf op* =) (*mk-less* (*ord-spmf op* =))
⟨*proof*⟩

interpretation *spmf*: *partial-function-definitions ord-spmf op* = *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 op* =)) (*ord-spmf op* =)

lemma *lub-spmf-const* [*simp*]: *lub-spmf* {*p*} = *p*
⟨*proof*⟩

lemma *bind-spmf-mono'*:
assumes *fg*: *ord-spmf op* = *f g*
and *hk*: $\bigwedge x :: 'a.$ *ord-spmf op* = (*h x*) (*k x*)
shows *ord-spmf op* = (*f* ≫= *h*) (*g* ≫= *k*)
⟨*proof*⟩

lemma *bind-spmf-mono* [*partial-function-mono*]:
assumes *mf*: *mono-spmf B* **and** *mg*: $\bigwedge y.$ *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 op* =) (*ord-spmf op* =) (λ*y*. *bind-spmf y g*)
⟨*proof*⟩

lemma *monotone-bind-spmf2*:
assumes *g*: $\bigwedge x.$ *monotone ord* (*ord-spmf op* =) (λ*y*. *g y x*)
shows *monotone ord* (*ord-spmf op* =) (λ*y*. *bind-spmf p* (*g y*))
⟨*proof*⟩

lemma *bind-lub-spmf*:
assumes *chain*: *Complete-Partial-Order.chain* (*ord-spmf op* =) *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 op* =) *Y*
⟹ *map-spmf f* (*lub-spmf Y*) = *lub-spmf* (*map-spmf f* ‘ *Y*)
⟨*proof*⟩

lemma *mcont-bind-spmf1*: *mcont lub-spmf* (*ord-spmf op* =) *lub-spmf* (*ord-spmf op* =) (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 op* =) (*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 op* =) *f*
and *g*: ⋀*y*. *mcont luba orda lub-spmf* (*ord-spmf op* =) (*g y*)
shows *mcont luba orda lub-spmf* (*ord-spmf op* =) (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 op* =) *g*
⟹ *mcont luba orda lub-spmf* (*ord-spmf op* =) (λ*x*. *map-spmf f* (*g x*))
⟨*proof*⟩

lemma *monotone-set-spmf*: *monotone* (*ord-spmf op* =) *op* ⊆ *set-spmf*
⟨*proof*⟩

lemma *cont-set-spmf*: *cont lub-spmf* (*ord-spmf op* =) *Union op* ⊆ *set-spmf*
⟨*proof*⟩

lemma *mcont2mcont-set-spmf*[*THEN mcont2mcont*, *cont-intro*]:
shows *mcont-set-spmf*: *mcont lub-spmf* (*ord-spmf op* =) *Union op* ⊆ *set-spmf*
⟨*proof*⟩

lemma *monotone-spmf*: *monotone* (*ord-spmf op* =) *op* ≤ (λ*p*. *spmf p x*)
⟨*proof*⟩

lemma *cont-spmf*: *cont lub-spmf* (*ord-spmf op* =) *Sup op* ≤ (λ*p*. *spmf p x*)
⟨*proof*⟩

lemma *mcont-spmf*: *mcont lub-spmf* (*ord-spmf op* =) *Sup op* ≤ (λ*p*. *spmf p x*)
⟨*proof*⟩

lemma *cont-ennreal-spmf*: *cont lub-spmf* (*ord-spmf op* =) *Sup op* ≤ (λ*p*. *ennreal* (*spmf p x*))
⟨*proof*⟩

lemma *mcont2mcont-ennreal-spmf* [*THEN mcont2mcont*, *cont-intro*]:
shows *mcont-ennreal-spmf*: *mcont lub-spmf* (*ord-spmf op* =) *Sup op* ≤ (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*⟩

27.11.1 Admissibility of *rel-spmf*

lemma *rel-spmf-measureD*:
assumes *rel-spmf R p q*
shows *measure* (*measure-spmf p*) *A* ≤ *measure* (*measure-spmf q*) {*y*. ∃*x*∈*A*. *R x y*} (**is** *?lhs* ≤ *?rhs*)
⟨*proof*⟩

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 op* =) (*ord-spmf op* =)) (*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 op* =) *f*; *mcont lub ord lub-spmf* (*ord-spmf op* =) *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 op* =) (*ord-spmf op* =) *F*
and *g*: ⋀*x*. *monotone* (*ord-spmf op* =) (*ord-spmf op* =) *G*
and *param*: (*rel-spmf R* ===> *rel-spmf R*) *F G*
shows (*rel-spmf R*) (*ccpo.fixp lub-spmf* (*ord-spmf op* =) *F*) (*ccpo.fixp lub-spmf* (*ord-spmf op* =) *G*)
⟨*proof*⟩

lemma *fixp-spmf-parametric*:
assumes *f*: ⋀*x*. *mono-spmf* (λ*f*. *F f x*)
and *g*: ⋀*x*. *mono-spmf* (λ*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

27.12 Restrictions on spmfs

definition *restrict-spmf* :: *'a spmf* ⇒ *'a set* ⇒ *'a spmf* (**infixl** ↿ *110*)
where *p* ↿ *A* = *map-pmf* (λ*x*. *x* ⤜ (λ*y*. *if y* ∈ *A then Some y else None*)) *p*

lemma *set-restrict-spmf* [*simp*]: *set-spmf* (*p* ↿ *A*) = *set-spmf p* ∩ *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* ∩ *B*)

⟨*proof*⟩

lemma *restrict-spmf-empty* [*simp*]: $p \upharpoonright \{\} = \textit{return-pmf None}$
⟨*proof*⟩

lemma *restrict-spmf-UNIV* [*simp*]: $p \upharpoonright \textit{UNIV} = p$
⟨*proof*⟩

lemma *spmf-restrict-spmf-outside* [*simp*]: $x \notin A \Longrightarrow \textit{spmf}\ (p \upharpoonright A)\ x = 0$
⟨*proof*⟩

lemma *emeasure-restrict-spmf* [*simp*]:
$\textit{emeasure}\ (\textit{measure-spmf}\ (p \upharpoonright A))\ X = \textit{emeasure}\ (\textit{measure-spmf}\ p)\ (X \cap A)$
⟨*proof*⟩

lemma *measure-restrict-spmf* [*simp*]:
$\textit{measure}\ (\textit{measure-spmf}\ (p \upharpoonright A))\ X = \textit{measure}\ (\textit{measure-spmf}\ p)\ (X \cap A)$
⟨*proof*⟩

lemma *spmf-restrict-spmf*: $\textit{spmf}\ (p \upharpoonright A)\ x = (\textit{if}\ x \in A\ \textit{then spmf}\ p\ x\ \textit{else}\ 0)$
⟨*proof*⟩

lemma *spmf-restrict-spmf-inside* [*simp*]: $x \in A \Longrightarrow \textit{spmf}\ (p \upharpoonright A)\ x = \textit{spmf}\ p\ x$
⟨*proof*⟩

lemma *pmf-restrict-spmf-None*: $\textit{pmf}\ (p \upharpoonright A)\ \textit{None} = \textit{pmf}\ p\ \textit{None} + \textit{measure}\ (\textit{measure-spmf}\ p)\ (-\ A)$
⟨*proof*⟩

lemma *restrict-spmf-trivial*: $(\bigwedge x.\ x \in \textit{set-spmf}\ p \Longrightarrow x \in A) \Longrightarrow p \upharpoonright A = p$
⟨*proof*⟩

lemma *restrict-spmf-trivial'*: $\textit{set-spmf}\ p \subseteq A \Longrightarrow p \upharpoonright A = p$
⟨*proof*⟩

lemma *restrict-return-spmf*: $\textit{return-spmf}\ x \upharpoonright A = (\textit{if}\ x \in A\ \textit{then return-spmf}\ x\ \textit{else return-pmf None})$
⟨*proof*⟩

lemma *restrict-return-spmf-inside* [*simp*]: $x \in A \Longrightarrow \textit{return-spmf}\ x \upharpoonright A = \textit{return-spmf}\ x$
⟨*proof*⟩

lemma *restrict-return-spmf-outside* [*simp*]: $x \notin A \Longrightarrow \textit{return-spmf}\ x \upharpoonright A = \textit{return-pmf None}$
⟨*proof*⟩

lemma *restrict-spmf-return-pmf-None* [*simp*]: $\textit{return-pmf None} \upharpoonright A = \textit{return-pmf None}$

⟨*proof*⟩

lemma *restrict-bind-pmf*: *bind-pmf p g* ↿ *A* = *p* ⤜ (*λx. g x* ↿ *A*)
⟨*proof*⟩

lemma *restrict-bind-spmf*: *bind-spmf p g* ↿ *A* = *p* ⤜ (*λx. g x* ↿ *A*)
⟨*proof*⟩

lemma *bind-restrict-pmf*: *bind-pmf* (*p* ↿ *A*) *g* = *p* ⤜ (*λx. if x* ∈ *Some* ‘ *A then g x else g None*)
⟨*proof*⟩

lemma *bind-restrict-spmf*: *bind-spmf* (*p* ↿ *A*) *g* = *p* ⤜ (*λx. if x* ∈ *A then g x else return-pmf None*)
⟨*proof*⟩

lemma *spmf-map-restrict*: *spmf* (*map-spmf fst* (*p* ↿ (*snd* −‘ {*y*}))) *x* = *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*⟩

27.13 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 $\Longrightarrow$ *map-spmf f* (*spmf-of-set A*) = *spmf-of-set* (*f ‘ A*)
⟨*proof*⟩

lemma *spmf-of-pmf-pmf-of-set* [*simp*]:
⟦ *finite A*; *A* ≠ {} ⟧ $\Longrightarrow$ *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* ≠ {} ⟧ $\Longrightarrow$ *weight-spmf* (*spmf-of-set A*) = *1*
⟨*proof*⟩

lemma *weight-spmf-of-set-infinite* [*simp*]: *infinite A* $\Longrightarrow$ *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* $\Longrightarrow$ *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*: $\bigwedge x.\ x \in A \Longrightarrow R\ x\ (f\ x)$
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* $B \neq \{\}$
shows *map-spmf* ($\lambda x.\ x \in A$) (*spmf-of-set B*) = *spmf-of-pmf* (*bernoulli-pmf* (*card* ($A \cap B$) / *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* (*op* = *c*) *coin-spmf* = *coin-spmf*
⟨*proof*⟩

lemma *bind-coin-spmf-eq-const*: *coin-spmf* ≫= (λx :: *bool*. *return-spmf* ($b = x$)) = *coin-spmf*
⟨*proof*⟩

lemma *bind-coin-spmf-eq-const'*: *coin-spmf* ≫= (λx :: *bool*. *return-spmf* ($x = b$)) = *coin-spmf*
⟨*proof*⟩

27.14 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 ⟷ *None* ∉ *set-pmf p*
⟨proof⟩

lemma *lossless-spmf-of-set* [*simp*]: *lossless-spmf* (*spmf-of-set A*) ⟷ *finite A* ∧ *A* ≠ {}
⟨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*) ⟷ (∀ *x*∈*set-pmf p*. *lossless-spmf* (*f x*))
⟨proof⟩

lemma *lossless-spmf-conv-spmf-of-pmf*: *lossless-spmf p* ⟷ (∃ *p'*. *p* = *spmf-of-pmf p'*)
⟨proof⟩

lemma *spmf-False-conv-True*: *lossless-spmf p* ⟹ *spmf p False* = *1* − *spmf p True*
⟨proof⟩

lemma *spmf-True-conv-False*: *lossless-spmf p* ⟹ *spmf p True* = *1* − *spmf p False*
⟨proof⟩

lemma *bind-eq-return-spmf*:
bind-spmf p f = *return-spmf x* ⟷ (∀ *y*∈*set-spmf p*. *f y* = *return-spmf x*) ∧ *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*⟩

27.15 Scaling

definition *scale-spmf* :: *real* $\Rightarrow$ *'a spmf* $\Rightarrow$ *'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*:
($\int^+$ *x. min (inverse (weight-spmf p)) (max 0 r)* $*$ *spmf p x* ∂*count-space UNIV*) $\leq$ *1* (**is** *?lhs* $\leq$ -)
⟨*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* $\leq$ *x*; *x* $\leq$ *1* ⟧ $\Longrightarrow$ *1 / x* $\leq$ *1* $\longleftrightarrow$ *x = 1* $\vee$ *x = 0*
⟨*proof*⟩

lemma *spmf-scale-spmf'*: *r* $\leq$ *1* $\Longrightarrow$ *spmf (scale-spmf r p) x = max 0 r* $*$ *spmf p x*
⟨*proof*⟩

lemma *scale-spmf-neg*: *r* $\leq$ *0* $\Longrightarrow$ *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'*:
r ≤ *1* ⟹ *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* (λ*x*. *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* (λ*x*. *scale-spmf r* (*f x*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *bind-spmf-const*: *bind-spmf p* (λ*x*. *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 $r > 0 \Longrightarrow$ *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*]:
⟦ $0 \leq r$; $r \leq 1$ ⟧ ⟹ *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*]:
⟦ $0 \leq r$; $r \leq 1$; $0 \leq r'$; $r' \leq 1$ ⟧
⟹ *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*⟩

27.16 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*⟩

27.17 Product spmf

definition *pair-spmf* :: *'a spmf* ⇒ *'b spmf* ⇒ (*'a* × *'b*) *spmf*
where *pair-spmf p q* = *bind-pmf* (*pair-pmf p q*) (λ*xy*. *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* (λ*x*. *bind-spmf q* (λ*y*. *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*:
$r \leq 1 \Longrightarrow$ *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*⟩

27.18 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*⟩

27.19 Try

definition *try-spmf* :: *'a spmf* ⇒ *'a spmf* ⇒ *'a spmf* (*TRY - ELSE -* [*0*,*60*] *59*)
where *try-spmf p q* = *bind-pmf p* (λ*x*. *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* (λ*x*. *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* (λ*x*. *TRY* (*f x*) *ELSE q*)
⟨*proof*⟩

lemma *try-spmf-bind-out*:
lossless-spmf p ⟹ *bind-spmf p* (λ*x*. *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*⟩

27.20 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

theory *Stream-Space*
imports
Infinite-Product-Measure
HOL−Library.Stream
HOL−Library.Linear-Temporal-Logic-on-Streams
begin

lemma *stream-eq-Stream-iff*: *s* = *x* ## *t* ⟷ (*shd s* = *x* ∧ *stl s* = *t*)
⟨*proof*⟩

lemma *Stream-snth*: (*x* ## *s*) !! *n* = (*case n of 0* ⇒ *x* | *Suc n* ⇒ *s* !! *n*)
⟨*proof*⟩

definition *to-stream* :: (*nat* ⇒ *'a*) ⇒ *'a stream* **where**
to-stream X = *smap X nats*

lemma *to-stream-nat-case*: *to-stream* (*case-nat x X*) = *x* ## *to-stream X*
⟨*proof*⟩

lemma *to-stream-in-streams*: *to-stream X* ∈ *streams S* ⟷ (∀ *n*. *X n* ∈ *S*)
⟨*proof*⟩

definition *stream-space* :: *'a measure* ⇒ *'a stream measure* **where**
stream-space M =
distr (Π_M *i*∈*UNIV*. *M*) (*vimage-algebra* (*streams* (*space M*)) *snth* (Π_M *i*∈*UNIV*. *M*)) *to-stream*

lemma *space-stream-space*: *space* (*stream-space M*) = *streams* (*space M*)
⟨*proof*⟩

lemma *streams-stream-space*[*intro*]: *streams* (*space M*) ∈ *sets* (*stream-space M*)
⟨*proof*⟩

lemma *stream-space-Stream*:
x ## ω ∈ *space* (*stream-space M*) ⟷ *x* ∈ *space M* ∧ ω ∈ *space* (*stream-space M*)
⟨*proof*⟩

lemma *stream-space-eq-distr*: *stream-space M* = *distr* (Π_M *i*∈*UNIV*. *M*) (*stream-space M*) *to-stream*
⟨*proof*⟩

lemma *sets-stream-space-cong*[*measurable-cong*]:
sets M = *sets N* ⟹ *sets* (*stream-space M*) = *sets* (*stream-space N*)
⟨*proof*⟩

lemma *measurable-snth-PiM*: (λω *n*. ω !! *n*) ∈ *measurable* (*stream-space M*) (Π_M *i*∈*UNIV*. *M*)
⟨*proof*⟩

lemma *measurable-snth*[*measurable*]: (λω. ω !! *n*) ∈ *measurable* (*stream-space M*) *M*
⟨*proof*⟩

lemma *measurable-shd*[*measurable*]: *shd* ∈ *measurable* (*stream-space M*) *M*
⟨*proof*⟩

lemma *measurable-stream-space2*:
assumes *f-snth*: ⋀*n*. (λ*x*. *f x* !! *n*) ∈ *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 $f \in measurable\ N\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-sdrop*[*measurable*]: $sdrop\ n \in measurable\ (stream\text{-}space\ M)\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-stl*[*measurable*]: $(\lambda\omega.\ stl\ \omega) \in measurable\ (stream\text{-}space\ M)\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-to-stream*[*measurable*]: $to\text{-}stream \in measurable\ (\Pi_M\ i{\in}UNIV.\ M)\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-Stream*[*measurable (raw)*]:
assumes f[*measurable*]: $f \in measurable\ N\ M$
assumes g[*measurable*]: $g \in measurable\ N\ (stream\text{-}space\ M)$
shows $(\lambda x.\ f\ x\ \#\#\ g\ x) \in measurable\ N\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-smap*[*measurable*]:
assumes X[*measurable*]: $X \in measurable\ N\ M$
shows $smap\ X \in measurable\ (stream\text{-}space\ N)\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-stake*[*measurable*]:
$stake\ i \in measurable\ (stream\text{-}space\ (count\text{-}space\ UNIV))\ (count\text{-}space\ (UNIV :: 'a::countable\ list\ set))$
⟨*proof*⟩

lemma *measurable-shift*[*measurable*]:
assumes f: $f \in measurable\ N\ (stream\text{-}space\ M)$
assumes [*measurable*]: $g \in measurable\ N\ (stream\text{-}space\ M)$
shows $(\lambda x.\ stake\ n\ (f\ x)\ @-\ g\ x) \in measurable\ N\ (stream\text{-}space\ M)$
⟨*proof*⟩

lemma *measurable-case-stream-replace*[*measurable (raw)*]:
$(\lambda x.\ f\ x\ (shd\ (g\ x))\ (stl\ (g\ x))) \in measurable\ M\ N \Longrightarrow (\lambda x.\ case\text{-}stream\ (f\ x)\ (g\ x)) \in measurable\ M\ N$
⟨*proof*⟩

lemma *measurable-ev-at*[*measurable*]:
assumes [*measurable*]: $Measurable.pred\ (stream\text{-}space\ M)\ P$
shows $Measurable.pred\ (stream\text{-}space\ M)\ (ev\text{-}at\ P\ n)$
⟨*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*]: $t \in$ *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*:
($\lambda(\omega 1, \omega 2)$. *szip* $\omega 1$ $\omega 2$) $\in$ *measurable* (*stream-space M* $\bigotimes_M$ *stream-space N*) (*stream-space* (*M* $\bigotimes_M$ *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*]: $f \in$ *borel-measurable* (*stream-space M*)
shows ($\int^+ X.\ f\ X\ \partial$*stream-space M*) = ($\int^+ x.\ (\int^+ X.\ f\ (x\ \#\#\ X)\ \partial$*stream-space M*) ∂M)
⟨*proof*⟩

lemma (**in** *prob-space*) *emeasure-stream-space*:
assumes $X[measurable]$: $X \in$ *sets* (*stream-space M*)
shows *emeasure* (*stream-space M*) $X = (\int^+ t.$ *emeasure* (*stream-space M*) $\{x \in$*space* (*stream-space M*). t ## $x \in X$ } ∂M)
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-stream-space*:
assumes $P[measurable]$: $\{x \in$*space* (*stream-space M*). $P\ x\} \in$ *sets* (*stream-space M*)
shows $\mathcal{P}$(x *in stream-space M*. $P\ x$) = ($\int^+ t$. $\mathcal{P}$(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 $X[measurable]$: $X \in$ *sets M* **shows** *streams* $X \in$ *sets* (*stream-space M*)
⟨*proof*⟩

lemma *sets-stream-space-in-sets*:
assumes *space*: *space* N = *streams* (*space M*)
assumes *sets*: $\bigwedge i$. (λx. x !! i) $\in$ *measurable N M*
shows *sets* (*stream-space M*) $\subseteq$ *sets N*
⟨*proof*⟩

lemma *sets-stream-space-eq*: *sets* (*stream-space M*) =
sets (*SUP i*:*UNIV*. *vimage-algebra* (*streams* (*space M*)) (λs. s !! i) M)
⟨*proof*⟩

lemma *sets-restrict-stream-space*:
assumes $S[measurable]$: $S \in$ *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) = *op* ## x ‘ *sstart S xs*

lemma *in-sstart*[*simp*]: $s \in sstart\ S\ (x\ \#\ xs) \longleftrightarrow shd\ s = x \wedge stl\ s \in sstart\ S\ xs$
⟨*proof*⟩

lemma *sstart-in-streams*: $xs \in lists\ S \Longrightarrow sstart\ S\ xs \subseteq streams\ S$
⟨*proof*⟩

lemma *sstart-eq*: $x \in streams\ S \Longrightarrow x \in sstart\ S\ xs = (\forall\, i{<}length\ xs.\ x\ !!\ i = xs\ !\ i)$
⟨*proof*⟩

lemma *sstart-sets*: $sstart\ S\ xs \in sets\ (stream\text{-}space\ (count\text{-}space\ UNIV))$
⟨*proof*⟩

lemma *sigma-sets-singletons*:
assumes *countable S*
shows $sigma\text{-}sets\ S\ ((\lambda s.\ \{s\})\text{'}S) = Pow\ S$
⟨*proof*⟩

lemma *sets-count-space-eq-sigma*:
$countable\ S \Longrightarrow sets\ (count\text{-}space\ S) = sets\ (sigma\ S\ ((\lambda s.\ \{s\})\text{'}S))$
⟨*proof*⟩

lemma *sets-stream-space-sstart*:
assumes *S*[*simp*]: *countable S*
shows $sets\ (stream\text{-}space\ (count\text{-}space\ S)) = sets\ (sigma\ (streams\ S)\ (sstart\ S\text{'}lists\ S \cup \{\{\}\}))$
⟨*proof*⟩

lemma *Int-stable-sstart*: $Int\text{-}stable\ (sstart\ S\text{'}lists\ S \cup \{\{\}\})$
⟨*proof*⟩

lemma *stream-space-eq-sstart*:
assumes *S*[*simp*]: *countable S*
assumes *P*: *prob-space M prob-space N*
assumes *ae*: $AE\ x\ in\ M.\ x \in streams\ S\ AE\ x\ in\ N.\ x \in streams\ S$
assumes *sets-M*: $sets\ M = sets\ (stream\text{-}space\ (count\text{-}space\ UNIV))$
assumes *sets-N*: $sets\ N = sets\ (stream\text{-}space\ (count\text{-}space\ UNIV))$
assumes $*$: $\bigwedge xs.\ xs \neq [] \Longrightarrow xs \in lists\ S \Longrightarrow emeasure\ M\ (sstart\ S\ xs) = emeasure\ N\ (sstart\ S\ xs)$
shows $M = N$
⟨*proof*⟩

lemma *sets-sstart*[*measurable*]: $sstart\ \Omega\ xs \in sets\ (stream\text{-}space\ (count\text{-}space\ UNIV))$
⟨*proof*⟩

primrec *scylinder* :: $'a\ set \Rightarrow\ 'a\ set\ list \Rightarrow\ 'a\ stream\ set$
where
$scylinder\ S\ [] = streams\ S$
$|\ scylinder\ S\ (A\ \#\ As) = \{\omega{\in}streams\ S.\ shd\ \omega \in A \wedge stl\ \omega \in scylinder\ S\ As\}$

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* $\rightarrow_M$ *prob-algebra* (*stream-space M*). ∃ *B*′∈*M* $\rightarrow_M$ *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*: ⋀*Y*. *countable Y* ⟹ *countable* (*F Y*)
and *cont*: *Order-Continuity.sup-continuous F*
shows *countable* (*lfp F*)
⟨*proof*⟩

lemma *countable-lfp-apply*:
assumes *step*: ⋀*Y x*. (⋀*x*. *countable* (*Y x*)) ⟹ *countable* (*F Y x*)
and *cont*: *Order-Continuity.sup-continuous F*
shows *countable* (*lfp F x*)
⟨*proof*⟩

primrec *left* :: $'a$ *tree* $\Rightarrow$ $'a$ *tree*
where
left (*Node l v r*) = *l*
| *left Leaf* = *Leaf*

primrec *right* :: $'a$ *tree* $\Rightarrow$ $'a$ *tree*
where
right (*Node l v r*) = *r*
| *right Leaf* = *Leaf*

inductive-set *trees* :: $'a$ *set* $\Rightarrow$ $'a$ *tree set* **for** *S* :: $'a$ *set* **where**
[*intro*!]: *Leaf* $\in$ *trees S*
| *l* $\in$ *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* (λT. $T \cup$ {*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'* $\in$ *trees-cyl t* $\longleftrightarrow$
($\exists$ *l v r*. *t* = *Node l v r* $\wedge$ *l'* $\in$ *trees-cyl l* $\wedge$ *r'* $\in$ *trees-cyl r* $\wedge$ *v'* $\in$ *v*)
⟨*proof*⟩

lemma *trees-cyl-sub-trees*:
assumes $t \in$ *trees* A $A \subseteq$ *Pow* B **shows** *trees-cyl* $t \subseteq$ *trees* B
⟨*proof*⟩

lemma *trees-cyl-sets-in-space*: *trees-cyl* ‘ *trees* (*sets* M) $\subseteq$ *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* $\in$ *space* (*tree-sigma* M)
⟨*proof*⟩

lemma *Leaf-in-tree-sigma* [*measurable*, *simp*, *intro*]: {*Leaf*} $\in$ *sets* (*tree-sigma* M)
⟨*proof*⟩

lemma *trees-cyl-map-treeI*: $t \in$ *trees-cyl* (*map-tree* ($\lambda x.\ A$) t) **if** $*$: $t \in$ *trees* A
⟨*proof*⟩

lemma *trees-cyl-map-in-sets*:
($\bigwedge x.\ x \in$ *set-tree* $t \Longrightarrow f\, x \in$ *sets* M) $\Longrightarrow$ *trees-cyl* (*map-tree* $f\, t$) $\in$ *sets* (*tree-sigma* M)
⟨*proof*⟩

lemma *Node-in-tree-sigma*:
assumes L: $X \in$ *sets* ($M \bigotimes_M$ (*tree-sigma* $M \bigotimes_M$ *tree-sigma* M))
shows {*Node* $l\ v\ r$ | $l\ v\ r$. $(v, l, r) \in X$} $\in$ *sets* (*tree-sigma* M)
⟨*proof*⟩

lemma *measurable-left*[*measurable*]: *left* $\in$ *tree-sigma* $M \rightarrow_M$ *tree-sigma* M
⟨*proof*⟩

lemma *measurable-right*[*measurable*]: *right* $\in$ *tree-sigma* $M \rightarrow_M$ *tree-sigma* M
⟨*proof*⟩

lemma *measurable-root-val'*: *root-val* $\in$ *restrict-space* (*tree-sigma* M) ($-${*Leaf*}) $\rightarrow_M M$
⟨*proof*⟩

lemma *measurable-restrict-mono*:
assumes f: $f \in$ *restrict-space* $M\ A \rightarrow_M N$ **and** $B \subseteq A$
shows $f \in$ *restrict-space* $M\ B \rightarrow_M N$
⟨*proof*⟩

lemma *measurable-root-val*[*measurable* (*raw*)]:
  assumes $f \in X \to_M$ *tree-sigma M*
    and $\bigwedge x.\ x \in$ *space X* $\Longrightarrow f\ x \neq$ *Leaf*
  shows ($\lambda\omega$. *root-val* ($f\ \omega$)) $\in X \to_M M$
⟨*proof*⟩

lemma *measurable-Node* [*measurable*]:
  (λ(*l*,*x*,*r*). *Node l x r*) $\in$ *tree-sigma M* $\bigotimes_M M \bigotimes_M$ *tree-sigma M* $\to_M$ *tree-sigma M*
⟨*proof*⟩

lemma *measurable-Node'* [*measurable* (*raw*)]:
  assumes [*measurable*]: $l \in B \to_M$ *tree-sigma A*
  assumes [*measurable*]: $x \in B \to_M A$
  assumes [*measurable*]: $r \in B \to_M$ *tree-sigma A*
  shows (λy. *Node* ($l\ y$) ($x\ y$) ($r\ y$)) $\in B \to_M$ *tree-sigma A*
⟨*proof*⟩

lemma *measurable-rec-tree*[*measurable* (*raw*)]:
  assumes *t*: $t \in B \to_M$ *tree-sigma M*
  assumes *l*: $l \in B \to_M A$
  assumes *n*: (λ(*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* $\bigotimes_M A \bigotimes_M A) \to_M A$ (**is** *?N* $\in$ *?M* $\to_M A$)
  shows (λx. *rec-tree* ($l\ x$) ($n\ x$) ($t\ x$)) $\in B \to_M A$
⟨*proof*⟩

lemma *measurable-case-tree* [*measurable* (*raw*)]:
  assumes $t \in B \to_M$ *tree-sigma M*
  assumes $l \in B \to_M A$
  assumes (λ(*x*, *l*, *v*, *r*). *n x l v r*)
      $\in B \bigotimes_M$ *tree-sigma M* $\bigotimes_M M \bigotimes_M$ *tree-sigma M* $\to_M A$
  shows (λx. *case-tree* ($l\ x$) ($n\ x$) ($t\ x$)) $\in B \to_M$ (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* ⇒ *'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*]: $f \in$ *borel-measurable F* $g \in$ *borel-measurable M*
shows ($\int^+ x.\ f\ x * $ *nn-cond-exp M F g x* ∂M) = ($\int^+ x.\ f\ x * g\ x\ \partial M$)
⟨*proof*⟩

lemma *nn-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*]: $f \in$ *borel-measurable M* $g \in$ *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* $\partial M) = (\int x.\ f\ x * g\ x\ \partial M)$
⟨*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* $\partial M)$
⟨*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* $\partial M) = (\int x.\ f\ x\ \partial M)$
⟨*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* ($\lambda x.$ *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* ($\lambda x.$ *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* ($\lambda x.$ *f x* $-$ *g x*) *x* = *real-cond-exp M F f x* $-$ *real-cond-exp M F g x*
⟨*proof*⟩

lemma *real-cond-exp-pos* [*intro*]:
assumes *AE x in M. f x* $\geq$ *0* **and** [*measurable*]: *f* $\in$ *borel-measurable M*
shows *AE x in M. real-cond-exp M F f x* $\geq$ *0*
⟨*proof*⟩

lemma *real-cond-exp-mono*:
assumes *AE x in M. f x* $\leq$ *g x* **and** [*measurable*]: *integrable M f integrable M g*
shows *AE x in M. real-cond-exp M F f x* $\leq$ *real-cond-exp M F g x*
⟨*proof*⟩

lemma (**in** $-$) *measurable-P-restriction* [*measurable* (*raw*)]:
assumes [*measurable*]: *Measurable.pred M P A* $\in$ *sets M*
shows $\{x \in A.\ P\ x\} \in$ *sets M*
⟨*proof*⟩

lemma *real-cond-exp-gr-c*:
assumes [*measurable*]: *integrable M f*
and *AE x in M. f x* > *c*
shows *AE x in M. real-cond-exp M F f x* > *c*
⟨*proof*⟩

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* $\geq$ *c*
shows *AE x in M. real-cond-exp M F f x* $\geq$ *c*
⟨*proof*⟩

lemma *real-cond-exp-le-c*:
assumes [*measurable*]: *integrable M f*
and *AE x in M. f x* $\leq$ *c*
shows *AE x in M. real-cond-exp M F f x* $\leq$ *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*]: $\bigwedge$*i. integrable M (f i)*
shows *AE x in M. real-cond-exp M F* (λ*x.* $\sum$ *i*$\in$*I. f i x*) *x* = ($\sum$ *i*$\in$*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* $\in$ *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* $\in$ *borel-measurable borel*
shows *AE x in M. real-cond-exp M F X x* $\in$ *I*
AE x in M. q (real-cond-exp M F X x) $\leq$ *real-cond-exp M F* (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*} ∨ *I* = {*a* <..} ∨ *I* = {..< *b*} ∨ *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* ∉ *M* $\rightarrow_M$ *borel* ⟹ *esssup M f* = *top*
⟨*proof*⟩

lemma *esssup-eq-AE*:
assumes *f*: *f* ∈ *M* $\rightarrow_M$ *borel* **shows** *esssup M f* = *Inf* {*z*. *AE x in M*. *f x* ≤ *z*}
⟨*proof*⟩

lemma *esssup-eq*: *f* ∈ *M* $\rightarrow_M$ *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* :: $('t{::}linorder \Rightarrow 'a\ measure) \Rightarrow ('a \Rightarrow 't) \Rightarrow bool$
where
  $stopping\text{-}time\ F\ T = (\forall\, t.\ Measurable.pred\ (F\ t)\ (\lambda x.\ T\ x \le t))$

lemma *stopping-time-cong*: $(\bigwedge t\, x.\ x \in space\ (F\ t) \Longrightarrow T\,x = S\,x) \Longrightarrow stopping\text{-}time\ F\ T = stopping\text{-}time\ F\ S$
  ⟨*proof*⟩

lemma *stopping-timeD*: $stopping\text{-}time\ F\ T \Longrightarrow Measurable.pred\ (F\ t)\ (\lambda x.\ T\,x \le t)$
  ⟨*proof*⟩

lemma *stopping-timeD2*: $stopping\text{-}time\ F\ T \Longrightarrow Measurable.pred\ (F\ t)\ (\lambda x.\ t < T\,x)$
  ⟨*proof*⟩

lemma *stopping-timeI*[*intro?*]: $(\bigwedge t.\ Measurable.pred\ (F\ t)\ (\lambda x.\ T\ x \le t)) \Longrightarrow stopping\text{-}time\ F\ T$
  ⟨*proof*⟩

lemma *measurable-stopping-time*:
  fixes $T :: 'a \Rightarrow 't{::}\{linorder\text{-}topology,\ second\text{-}countable\text{-}topology\}$
  assumes T: $stopping\text{-}time\ F\ T$
    and M: $\bigwedge t.\ sets\ (F\ t) \subseteq sets\ M$ $\bigwedge t.\ space\ (F\ t) = space\ M$
  shows $T \in M \rightarrow_M borel$
⟨*proof*⟩

lemma *stopping-time-const*: $stopping\text{-}time\ F\ (\lambda x.\ c)$
  ⟨*proof*⟩

lemma *stopping-time-min*:
  $stopping\text{-}time\ F\ T \Longrightarrow stopping\text{-}time\ F\ S \Longrightarrow stopping\text{-}time\ F\ (\lambda x.\ min\ (T\ x)\ (S\ x))$
  ⟨*proof*⟩

lemma *stopping-time-max*:
  $stopping\text{-}time\ F\ T \Longrightarrow stopping\text{-}time\ F\ S \Longrightarrow stopping\text{-}time\ F\ (\lambda x.\ max\ (T\ x)\ (S\ x))$
  ⟨*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) = Ω
⟨*proof*⟩

lemma *measure-pre-sigma*[*simp*]: *emeasure* (*pre-sigma* T) = (λ-. *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*) ($\lambda\omega.\ T\ \omega < t$)
⟨*proof*⟩

lemma *stopping-time-eq-const*: *stopping-time F T* ⟹ *Measurable.pred* (*F t*) ($\lambda\omega.$ $T\ \omega = t$)
  ⟨*proof*⟩

lemma *stopping-time-less*:
  assumes *T*: *stopping-time F T* **and** *S*: *stopping-time F S*
  shows *Measurable.pred* (*pre-sigma T*) ($\lambda\omega.\ T\ \omega < S\ \omega$)
⟨*proof*⟩

end

lemma *stopping-time-SUP-enat*:
  fixes *T* :: *nat* ⇒ (*'a* ⇒ *enat*)
  shows ($\bigwedge i.$ *stopping-time F* (*T i*)) ⟹ *stopping-time F* (*SUP i. T i*)
  ⟨*proof*⟩

lemma *less-eSuc-iff*: $a <$ *eSuc* $b \longleftrightarrow (a \leq b \wedge a \neq \infty)$
  ⟨*proof*⟩

lemma *stopping-time-Inf-enat*:
  fixes *F* :: *enat* ⇒ *'a measure*
  assumes *F*: *filtration* Ω *F*
  assumes *P*: $\bigwedge i.$ *Measurable.pred* (*F i*) (*P i*)
  shows *stopping-time F* ($\lambda\omega.$ *Inf* {*i. P i* ω})
⟨*proof*⟩

lemma *stopping-time-Inf-nat*:
  fixes *F* :: *nat* ⇒ *'a measure*
  assumes *F*: *filtration* Ω *F*
  assumes *P*: $\bigwedge i.$ *Measurable.pred* (*F i*) (*P i*) **and** *wf*: $\bigwedge i\ \omega.\ \omega \in \Omega \Longrightarrow \exists n.\ P$ $n\ \omega$
  shows *stopping-time F* ($\lambda\omega.$ *Inf* {*i. P i* ω})
  ⟨*proof*⟩

end

theory *Probability*
imports
Central-Limit-Theorem
Discrete-Topology
PMF-Impl
Projective-Limit
Random-Permutations
SPMF
Stream-Space
Tree-Space
Conditional-Expectation
Essential-Supremum
Stopping-Time
begin

end