S-Finite Measure Monad on Quasi-Borel Spaces

Michikazu Hirata, Yasuhiko Minamide

March 19, 2025

Abstract

The s-finite measure monad on quasi-Borel spaces provides a suitable denotational model for higher-order probabilistic programs with conditioning. This entry is a formalization of the s-finite measure monad and related notions, including s-finite measures, s-finite kernels, and a proof automation for quasi-Borel spaces which is an extension of our previous entry *quasi-Borel spaces*. We also implement several examples of probabilistic programs in previous works and prove their property.

This work is a part of the work by Hirata, Minamide, and Sato, *Semantic Foundations of Higher-Order Probabilistic Programs in Isabelle/HOL* which will be presented at the 14th Conference on Interactive Theorem Proving (ITP2023).

Contents

For the terminology of s-finite measures/kernels, we refer to the work by Staton [4]. For the definition of the s-finite measure monad, we refer to the lecture note by Yang [6]. The construction of the s-finite measure monad is based on the detailed pencil-and-paper proof by Tetsuya Sato.

1 Lemmas

theory *Lemmas-S-Finite-Measure-Monad*
imports *HOL−Probability.Probability Standard-Borel-Spaces.StandardBorel*
begin

lemma *integrable-mono-measure*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes [*measurable-cong*,*measurable*]:*sets M* = *sets N M* ≤ *N integrable N f*
shows *integrable M f*
using *assms*(*3*) *nn-integral-mono-measure*[*OF assms*(*1*,*2*),*of* λ*x*. *ennreal* (*norm* (*f x*))]
by(*auto simp*: *integrable-iff-bounded*)

lemma *AE-mono-measure*:
assumes *sets M* = *sets N M* ≤ *N AE x in N*. *P x*
shows *AE x in M*. *P x*
by (*metis* (*no-types*, *lifting*) *AE-E Collect-cong assms eventually-ae-filter le-measure le-zero-eq null-setsI sets-eq-imp-space-eq*)

lemma *finite-measure-return*:*finite-measure* (*return M x*)
by(*auto intro*!: *finite-measureI*) (*metis ennreal-top-neq-one ennreal-zero-neq-top indicator-eq-0-iff indicator-eq-1-iff*)

lemma *nn-integral-return*′:
assumes *x* ∉ *space M*

shows $(\int^+ x.\ g\ x\ \partial return\ M\ x) = 0$
proof −
have *emeasure* (*return M x*) *A* = *0* **for** *A*
by(*cases A* ∈ *sets M*,*insert assms*) (*auto simp*: *indicator-def emeasure-notin-sets dest*: *sets.sets-into-space*)
thus *?thesis*
by(*auto simp*: *nn-integral-def simple-integral-def*) (*meson SUP-least le-zero-eq*)
qed

lemma *pair-measure-return*: *return M l* $\bigotimes_M$ *return N r* = *return* (*M* $\bigotimes_M$ *N*) (*l*,*r*)
proof(*safe intro*!: *measure-eqI*)
fix *A*
assume *A* ∈ *sets* (*return M l* $\bigotimes_M$ *return N r*)
then have *A*[*measurable*]:*A* ∈ *sets* (*M* $\bigotimes_M$ *N*) **by** *simp*
note [*measurable-cong*] = *sets-return*[*of M*] *sets-return*[*of N*]
interpret *finite-measure return N r* **by**(*simp add*: *finite-measure-return*)
consider *l* ∉ *space M* | *r* ∉ *space N* | *l* ∈ *space M r* ∈ *space N* **by** *auto*
then show *emeasure* (*return M l* $\bigotimes_M$ *return N r*) *A* = *emeasure* (*return* (*M* $\bigotimes_M$ *N*) (*l*, *r*)) *A* (**is** *?lhs* = *?rhs*)
by(*cases*, *insert sets.sets-into-space*[*OF A*]) (*auto simp*: *emeasure-pair-measure nn-integral-return*′ *space-pair-measure nn-integral-return*, *auto simp*: *indicator-def*)
qed *simp-all*

lemma *null-measure-distr*: *distr* (*null-measure M*) *N f* = *null-measure N*
by(*auto intro*!: *measure-eqI simp*: *distr-def emeasure-sigma*)

lemma *integral-measurable-subprob-algebra2*:
fixes *f* :: - ⇒ - ⇒ -::{*banach*,*second-countable-topology*}
assumes [*measurable*]: (λ(*x*, *y*). *f x y*) ∈ *borel-measurable* (*M* $\bigotimes_M$ *N*) *L* ∈ *measurable M* (*subprob-algebra N*)
shows (λ*x*. *integral*^{L} (*L x*) (*f x*)) ∈ *borel-measurable M*
proof −
note *integral-measurable-subprob-algebra*[*measurable*]
note *measurable-distr2*[*measurable*]
have (λ*x*. *integral*^{L} (*distr* (*L x*) (*M* $\bigotimes_M$ *N*) (λ*y*. (*x*, *y*))) (λ(*x*, *y*). *f x y*)) ∈ *borel-measurable M*
by *measurable*
then show (λ*x*. *integral*^{L} (*L x*) (*f x*)) ∈ *borel-measurable M*
by (*rule measurable-cong*[*THEN iffD1*, *rotated*])
(*simp add*: *integral-distr*)
qed

lemma *distr-id*′:
assumes *sets N* = *sets M*
and ⋀*x*. *x* ∈ *space N* ⟹ *f x* = *x*
shows *distr N M f* = *N*
by(*simp add*: *distr-cong*[*OF refl refl*,*of N f id*,*simplified*,*OF assms*(*2*),*simplified*] *distr-id2*[*OF assms*(*1*)[*symmetric*]] *id-def*)

lemma *measure-density-times*:
assumes [*measurable*]:$S \in$ *sets* M $X \in$ *sets* M $r \neq \infty$
shows *measure* (*density* M (λx. *indicator* S $x * r$)) X = *enn2real* r * *measure* M ($S \cap X$)
proof −
have [*simp*]:*density* M (λx. *indicator* S $x * r$) = *density* (*density* M (*indicator* S)) (λ-. r)
by(*simp add*: *density-density-eq*)
show *?thesis*
by(*simp add*: *measure-density-const*[*OF* - *assms*(*3*)] *measure-restricted*)
qed

lemma *complete-the-square*:
fixes a b c x :: *real*
assumes $a \neq 0$
shows $a*x^2 + b * x + c = a * (x + (b / (2*a)))^2 - ((b^2 - 4* a * c)/(4*a))$
using *assms* **by**(*simp add*: *comm-semiring-1-class.power2-sum power2-eq-square*[*of* $b / (2 * a)$] *ring-class.ring-distribs*(*1*) *division-ring-class.diff-divide-distrib power2-eq-square*[*of* b])

lemma *complete-the-square2'*:
fixes a b c x :: *real*
assumes $a \neq 0$
shows $a*x^2 - 2 * b * x + c = a * (x - (b / a))^2 - ((b^2 - a*c)/a)$
using *complete-the-square*[*OF assms*,**where** b=$-2 * b$ **and** x=x **and** c=c]
by(*simp add*: *division-ring-class.diff-divide-distrib assms*)

lemma *normal-density-mu-x-swap*:
normal-density μ σ x = *normal-density* x σ μ
by(*simp add*: *normal-density-def power2-commute*)

lemma *normal-density-plus-shift*: *normal-density* μ σ (x + y) = *normal-density* (μ − x) σ y
by(*simp add*: *normal-density-def add.commute diff-diff-eq2*)

lemma *normal-density-times*:
assumes $\sigma > 0$ $\sigma' > 0$
shows *normal-density* μ σ x * *normal-density* μ' σ' x = (*1* / *sqrt* (*2* * *pi* * ($\sigma^2 + \sigma'^2$))) * *exp* (− (μ − μ')^2 / (*2* * ($\sigma^2 + \sigma'^2$))) * *normal-density* (($\mu*\sigma'^2 + \mu'*\sigma^2$)/($\sigma^2 + \sigma'^2$)) ($\sigma$ * σ' / *sqrt* ($\sigma^2 + \sigma'^2$)) x
(**is** *?lhs* = *?rhs*)
proof −
have *non0*: $2*\sigma^2 \neq 0$ $2*\sigma'^2 \neq 0$ $\sigma^2 + \sigma'^2 \neq 0$
using *assms* **by** *auto*
have *?lhs* = *exp* (− ((x − μ)^2 / (*2* * σ^2))) * *exp* (− ((x − μ')^2 / (*2* * σ'^2))) / (*sqrt* (*2* * *pi* * σ^2) * *sqrt* (*2* * *pi* * σ'^2))
by(*simp add*: *normal-density-def*)
also have ... = *exp* (− ((x − μ)^2 / (*2* * σ^2)) − ((x − μ')^2 / (*2* * σ'^2))) / (*sqrt*

$(2 * pi * \sigma^2) * sqrt\ (2 * pi * \sigma'^2))$
by(*simp add*: *exp-add*[*of* $-$ $((x - \mu)^2 / (2 * \sigma^2)) - ((x - \mu')^2 / (2 * \sigma'^2))$,*simplified add-uminus-conv-diff*])
also have ... = *exp* $(- (x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))^2) - (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2)))\ / (sqrt\ (2 * pi * \sigma^2) * sqrt\ (2 * pi * \sigma'^2))$
proof $-$
have $((x - \mu)^2 / (2 * \sigma^2)) + ((x - \mu')^2 / (2 * \sigma'^2)) = (x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))^2) + (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2))$
(**is** *?lhs'* = *?rhs'*)
proof $-$
have *?lhs'* = $(2 * ((x - \mu)^2 * \sigma'^2) + 2 * ((x - \mu')^2 * \sigma^2)) / (4 * (\sigma^2 * \sigma'^2))$
by(*simp add*: *field-class.add-frac-eq*[*OF non0*(*1*,*2*)])
also have ... = $((x - \mu)^2 * \sigma'^2 + (x - \mu')^2 * \sigma^2) / (2 * (\sigma^2 * \sigma'^2))$
by(*simp add*: *power2-eq-square division-ring-class.add-divide-distrib*)
also have ... = $((\sigma^2 + \sigma'^2) * x^2 - 2 * (\mu * \sigma'^2 + \mu' * \sigma^2) * x\ + (\mu'^2 * \sigma^2 + \mu^2 * \sigma'^2)) / (2 * (\sigma^2 * \sigma'^2))$
by(*simp add*: *comm-ring-1-class.power2-diff ring-class.left-diff-distrib semiring-class.distrib-right*)
also have ... = $((\sigma^2 + \sigma'^2) * (x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 - ((\mu * \sigma'^2 + \mu' * \sigma^2)^2 - (\sigma^2 + \sigma'^2) * (\mu'^2 * \sigma^2 + \mu^2 * \sigma'^2)) / (\sigma^2 + \sigma'^2)) / (2 * (\sigma^2 * \sigma'^2))$
by(*simp only*: *complete-the-square2'*[*OF non0*(*3*),*of x* $(\mu * \sigma'^2 + \mu' * \sigma^2)$ $(\mu'^2 * \sigma^2 + \mu^2 * \sigma'^2)$])
also have ... = $((\sigma^2 + \sigma'^2) * (x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2) / (2 * (\sigma^2 * \sigma'^2)) - (((\mu * \sigma'^2 + \mu' * \sigma^2)^2 - (\sigma^2 + \sigma'^2) * (\mu'^2 * \sigma^2 + \mu^2 * \sigma'^2)) / (\sigma^2 + \sigma'^2)) / (2 * (\sigma^2 * \sigma'^2))$
by(*simp add*: *division-ring-class.diff-divide-distrib*)
also have ... = $(x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * ((\sigma * \sigma') / sqrt\ (\sigma^2 + \sigma'^2))^2) - (((\mu * \sigma'^2 + \mu' * \sigma^2)^2 - (\sigma^2 + \sigma'^2) * (\mu'^2 * \sigma^2 + \mu^2 * \sigma'^2)) / (\sigma^2 + \sigma'^2)) / (2 * (\sigma^2 * \sigma'^2))$
by(*simp add*: *monoid-mult-class.power2-eq-square*[*of* $(\sigma * \sigma') / sqrt\ (\sigma^2 + \sigma'^2)$] *ab-semigroup-mult-class.mult.commute*[*of* $\sigma^2 + \sigma'^2$])
(*simp add*: *monoid-mult-class.power2-eq-square*[*of* σ] *monoid-mult-class.power2-eq-square*[*of* σ'])
also have ... = $(x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))^2) - ((\mu * \sigma'^2)^2 + (\mu' * \sigma^2)^2 + 2 * (\mu * \sigma'^2) * (\mu' * \sigma^2) - (\sigma^2 * (\mu'^2 * \sigma^2) + \sigma^2 * (\mu^2 * \sigma'^2) + (\sigma'^2 * (\mu'^2 * \sigma^2) + \sigma'^2 * (\mu^2 * \sigma'^2)))) / ((\sigma^2 + \sigma'^2) * (2 * (\sigma^2 * \sigma'^2)))$
by(*simp add*: *comm-semiring-1-class.power2-sum*[*of* $\mu * \sigma'^2$ $\mu' * \sigma^2$] *semiring-class.distrib-right*[*of* σ^2 σ'^2 $\mu'^2 * \sigma^2 + \mu^2 * \sigma'^2$])
(*simp add*: *semiring-class.distrib-left*[*of* - $\mu'^2 * \sigma^2$ $\mu^2 * \sigma'^2$])
also have ... = $(x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))^2) + ((\sigma^2 * \sigma'^2) * \mu^2 + (\sigma^2 * \sigma'^2) * \mu'^2 - (\sigma^2 * \sigma'^2) * 2 * (\mu * \mu')) / ((\sigma^2 + \sigma'^2) * (2 * (\sigma^2 * \sigma'^2)))$
by(*simp add*: *monoid-mult-class.power2-eq-square division-ring-class.minus-divide-left*)
also have ... = $(x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))^2) + (\mu^2 + \mu'^2 - 2 * (\mu * \mu')) / ((\sigma^2 + \sigma'^2) * 2)$

using *assms* **by**(*simp add*: *division-ring-class.add-divide-distrib division-ring-class.diff-divide-distrib*)
also have ... = *?rhs'*
by(*simp add*: *comm-ring-1-class.power2-diff ab-semigroup-mult-class.mult.commute*[*of* *2*])
finally show *?thesis* **.**
qed
thus *?thesis*
by *simp*
qed
also have ... = (*exp* (− (*μ* − *μ'*)^2 / (*2* ∗ (*σ*^{2} + *σ'*^{2}))) / (*sqrt* (*2* ∗ *pi* ∗ *σ*^{2}) ∗ *sqrt* (*2* ∗ *pi* ∗ *σ'*^{2}))) ∗ *sqrt* (*2* ∗ *pi* ∗ (*σ* ∗ *σ'* / *sqrt* (*σ*^{2} + *σ'*^{2}))^2) ∗ *normal-density* ((*μ* ∗ *σ'*^{2} + *μ'* ∗ *σ*^{2}) / (*σ*^{2} + *σ'*^{2})) (*σ* ∗ *σ'* / *sqrt* (*σ*^{2} + *σ'*^{2})) *x*
by(*simp add*: *exp-add*[*of* − (*x* − (*μ* ∗ *σ'*^{2} + *μ'* ∗ *σ*^{2}) / (*σ*^{2} + *σ'*^{2}))^2 / (*2* ∗ (*σ* ∗ *σ'* / *sqrt* (*σ*^{2} + *σ'*^{2}))^2) − (*μ* − *μ'*)^2 / (*2* ∗ (*σ*^{2} + *σ'*^{2})),*simplified*] *normal-density-def*)
also have ... = *?rhs*
proof −
have *exp* (− (*μ* − *μ'*)^2 / (*2* ∗ (*σ*^{2} + *σ'*^{2}))) / (*sqrt* (*2* ∗ *pi* ∗ *σ*^{2}) ∗ *sqrt* (*2* ∗ *pi* ∗ *σ'*^{2})) ∗ *sqrt* (*2* ∗ *pi* ∗ (*σ* ∗ *σ'* / *sqrt* (*σ*^{2} + *σ'*^{2}))^2) = *1* / *sqrt* (*2* ∗ *pi* ∗ (*σ*^{2} + *σ'*^{2})) ∗ *exp* (− (*μ* − *μ'*)^2 / (*2* ∗ (*σ*^{2} + *σ'*^{2})))
using *assms* **by**(*simp add*: *real-sqrt-mult*)
thus *?thesis*
by *simp*
qed
finally show *?thesis* **.**
qed

lemma *KL-normal-density*:
assumes [*arith*]: *b* > *0 d* > *0*
shows *KL-divergence* (*exp 1*) (*density lborel* (*normal-density a b*)) (*density lborel* (*normal-density c d*)) = *ln* (*b* / *d*) + (*d*^{2} + (*c* − *a*)^2) / (*2* ∗ *b*^{2}) − *1* / *2* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∫ *x*. *normal-density c d x* ∗ *ln* (*normal-density c d x* / *normal-density a b x*) ∂*lborel*)
by(*unfold log-ln*,*rule lborel.KL-density-density*) (*use order.strict-implies-not-eq*[*OF normal-density-pos*[*of b a*]] **in** *auto*)
also have ... = (∫ *x*. *normal-density c d x* ∗ *ln* (*normal-density c d x*) − *normal-density c d x* ∗ *ln* (*normal-density a b x*) ∂*lborel*)
by (*metis* (*no-types*, *opaque-lifting*) *assms ln-div normal-density-pos order.irrefl right-diff-distrib'*)
also have ... = (∫ *x*. *normal-density c d x* ∗ *ln* (*exp* (− (*x* − *c*)^2 / (*2* ∗ *d*^{2})) / *sqrt* (*2* ∗ *pi* ∗ *d*^{2})) − *normal-density c d x* ∗ *ln* (*exp* (− (*x* − *a*)^2 / (*2* ∗ *b*^{2})) / *sqrt* (*2* ∗ *pi* ∗ *b*^{2})) ∂*lborel*)
by(*simp add*: *normal-density-def*)
also have ... = (∫ *x*. *normal-density c d x* ∗ (− (*x* − *c*)^2 / (*2* ∗ *d*^{2}) − *ln* (*sqrt* (*2* ∗ *pi* ∗ *d*^{2}))) − (*normal-density c d x* ∗ (− (*x* − *a*)^2 / (*2* ∗ *b*^{2}) − *ln* (*sqrt* (*2* ∗ *pi* ∗ *b*^{2})))) ∂*lborel*)
by(*simp add*: *ln-div*)
also have ... = (∫ *x*. *normal-density c d x* ∗ (*ln* (*sqrt* (*2* ∗ *pi* ∗ *b*^{2})) − *ln* (*sqrt*

$(2 * pi * d^2))) + (normal\text{-}density\ c\ d\ x * ((x - a)^2 / (2 * b^2)) - normal\text{-}density\ c\ d\ x * ((x - c)^2 / (2 * d^2)))\ \partial lborel$)
by(*auto intro*!: *Bochner-Integration.integral-cong simp*: *right-diff-distrib*)
also have ... = ($\int x.\ normal\text{-}density\ c\ d\ x * (ln\ (sqrt\ (2 * pi * b^2)) - ln\ (sqrt\ (2 * pi * d^2))) + (normal\text{-}density\ c\ d\ x * ((x - c)^2 / (2 * b^2) + (2 * x * (c - a) + a\hat{}2 - c\hat{}2) / (2 * b^2)) - normal\text{-}density\ c\ d\ x * ((x - c)^2 / (2 * d^2)))\ \partial lborel$)
by(*auto intro*!: *Bochner-Integration.integral-cong simp*: *add-divide-distrib*[*symmetric*] *power2-diff*) (*simp add*: *right-diff-distrib*)
also have ... = ($\int x.\ (ln\ (sqrt\ (2 * pi * b^2)) - ln\ (sqrt\ (2 * pi * d^2))) * normal\text{-}density\ c\ d\ x + ((1 / (2 * b^2) * (normal\text{-}density\ c\ d\ x * (x - c)^2) + (2 * (c - a)) / (2 * b^2) * (normal\text{-}density\ c\ d\ x * x) + (a\hat{}2 - c\hat{}2) / (2 * b^2) * (normal\text{-}density\ c\ d\ x)) - 1 / (2 * d^2) * (normal\text{-}density\ c\ d\ x * (x - c)^2))\ \partial lborel$)
by(*auto intro*!: *Bochner-Integration.integral-cong simp*: *add-divide-distrib*[*symmetric*] *ring-distribs*)
also have ... = ($\int x.\ (ln\ (sqrt\ (2 * pi * b^2)) - ln\ (sqrt\ (2 * pi * d^2))) * normal\text{-}density\ c\ d\ x\ \partial lborel) + ((((\int x.\ 1 / (2 * b^2) * (normal\text{-}density\ c\ d\ x * (x - c)^2)\ \partial lborel) + (\int x.\ (2 * (c - a)) / (2 * b^2) * (normal\text{-}density\ c\ d\ x * x)\ \partial lborel) + (\int x.\ (a\hat{}2 - c\hat{}2) / (2 * b^2) * (normal\text{-}density\ c\ d\ x)\ \partial lborel)) - (\int x.\ 1 / (2 * d^2) * (normal\text{-}density\ c\ d\ x * (x - c)^2)\ \partial lborel))$
using *integrable-normal-moment-nz-1*[*OF assms*(*2*)] *integrable-normal-moment*[*OF assms*(*2*),**where** *k*=*2*] **by** *simp*
also have ... = $ln\ (sqrt\ (2 * pi * b^2)) - ln\ (sqrt\ (2 * pi * d^2)) + 1 / (2 * b^2) * d^2 + (2 * c - 2 * a) / (2 * b^2) * c + (a^2 - c^2) / (2 * b^2) - 1 / (2 * d^2) * d^2$
by(*simp add*: *integral-normal-moment-even*[*OF assms*(*2*),*of - 1*,*simplified*] *integral-normal-moment-nz-1*[*OF assms*(*2*)] *del*: *times-divide-eq-left*)
also have ... = $ln\ (b / d) + 1 / (2 * b^2) * d^2 + (2 * c - 2 * a) / (2 * b^2) * c + (a^2 - c^2) / (2 * b^2) - 1 / (2 * d^2) * d^2$
by(*simp add*: *ln-sqrt ln-mult power2-eq-square diff-divide-distrib*[*symmetric*] *ln-div*)
also have ... = *?rhs*
by(*auto simp*: *add-divide-distrib*[*symmetric*] *power2-diff left-diff-distrib*) (*simp add*: *power2-eq-square*)
finally show *?thesis* **.**
qed

lemma *count-space-prod*:*count-space* (*UNIV* :: (*'a* :: *countable*) *set*) $\bigotimes_M$ *count-space* (*UNIV* :: (*'b* :: *countable*) *set*) = *count-space UNIV*
by(*auto simp*: *pair-measure-countable*)

lemma *measure-pair-pmf*:
fixes *p* :: (*'a* :: *countable*) *pmf* **and** *q* :: (*'b* :: *countable*) *pmf*
shows *measure-pmf p* $\bigotimes_M$ *measure-pmf q* = *measure-pmf* (*pair-pmf p q*) (**is** *?lhs* = *?rhs*)
proof −
interpret *pair-prob-space measure-pmf p measure-pmf q*
by *standard*
have *?lhs* = *measure-pmf p* $\gg=$ ($\lambda x.$ *measure-pmf q* $\gg=$ ($\lambda y.$ *return* (*measure-pmf p* $\bigotimes_M$ *measure-pmf q*) (*x*, *y*)))

by(*rule pair-measure-eq-bind*)
also have ... = *?rhs*
by(*simp add*: *measure-pmf-bind pair-pmf-def return-pmf.rep-eq cong*: *return-cong*[*OF sets-pair-measure-cong*[*OF sets-measure-pmf-count-space*[*of p*] *sets-measure-pmf-count-space*[*of q*],*simplified count-space-prod*]])
finally show *?thesis* **.**
qed

lemma *distr-PiM-distr*:
assumes *finite I* $\bigwedge i.\ i \in I \Longrightarrow$ *sigma-finite-measure* (*distr* (*M i*) (*N i*) (*f i*))
and $\bigwedge i.\ i \in I \Longrightarrow f\ i \in M\ i \rightarrow_M N\ i$
shows *distr* (Π_M *i*∈*I*. *M i*) (Π_M *i*∈*I*. *N i*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) = (Π_M *i*∈*I*. *distr* (*M i*) (*N i*) (*f i*))
proof −
define *M′* **where** *M′* ≡ (λ*i*. *if i* ∈ *I then M i else null-measure* (*M i*))
have *f*[*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in M'\ i \rightarrow_M N\ i$ **and** [*measurable-cong*]: $\bigwedge i.$ *sets* (*M′ i*) = *sets* (*M i*) **and** [*simp*]: $\bigwedge i.\ i \in I \Longrightarrow M'\ i = M\ i$
by(*auto simp*: *M′-def assms*)
interpret *product-sigma-finite* λ*i*. *distr* (*M′ i*) (*N i*) (*f i*)
by(*auto simp*: *product-sigma-finite-def M′-def assms*(*2*)) (*auto intro*!: *finite-measure.sigma-finite-measure finite-measureI simp*: *null-measure-distr*)
interpret *ps*: *product-sigma-finite M′*
by(*auto simp*: *product-sigma-finite-def M′-def intro*!: *finite-measure.sigma-finite-measure*[*of null-measure -*] *finite-measureI sigma-finite-measure-distr*[*OF assms*(*2*)])
have *distr* (Π_M *i*∈*I*. *M i*) (Π_M *i*∈*I*. *N i*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) = *distr* (Π_M *i*∈*I*. *M′ i*) (Π_M *i*∈*I*. *N i*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*))
by(*simp cong*: *PiM-cong*)
also have ... = (Π_M *i*∈*I*. *distr* (*M′ i*) (*N i*) (*f i*))
proof(*rule PiM-eqI*[*OF assms*(*1*)])
fix *A*
assume $\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *sets* (*distr* (*M′ i*) (*N i*) (*f i*))
hence *h*[*measurable*]:$\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *sets* (*N i*)
by *simp*
have [*simp*]:(λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) −‘ (*Pi*$_E$ *I A*) ∩ *space* (*Pi*$_M$ *I M′*) = (Π_E *i*∈*I*. *f i* −‘ *A i* ∩ *space* (*M′ i*))
by(*auto simp*: *space-PiM*)
show *emeasure* (*distr* (*Pi*$_M$ *I M′*) (*Pi*$_M$ *I N*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*))) (*Pi*$_E$ *I A*) = ($\prod$ *i*∈*I*. *emeasure* (*distr* (*M′ i*) (*N i*) (*f i*)) (*A i*))
by(*auto simp*: *emeasure-distr assms*(*1*) *ps.emeasure-PiM*[*OF assms*(*1*)])
qed(*simp-all cong*: *sets-PiM-cong*)
also have ... = (Π_M *i*∈*I*. *distr* (*M i*) (*N i*) (*f i*))
by(*auto cong*: *PiM-cong*)
finally show *?thesis* **.**
qed

lemma *distr-PiM-distr-prob*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *prob-space* (*M i*)
and $\bigwedge i.\ i \in I \Longrightarrow f\ i \in M\ i \rightarrow_M N\ i$
shows *distr* (Π_M *i*∈*I*. *M i*) (Π_M *i*∈*I*. *N i*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) = (Π_M *i*∈*I*.

distr (*M i*) (*N i*) (*f i*))
proof −
define *M′* **where** *M′* ≡ (λ*i*. *if i* ∈ *I then M i else return* (*count-space UNIV*) *undefined*)
define *N′* **where** *N′* ≡ (λ*i*. *if i* ∈ *I then N i else return* (*count-space UNIV*) *undefined*)
interpret *p*: *product-prob-space* λ*i*. *distr* (*M′ i*) (*N′ i*) (*f i*)
by(*auto simp*: *product-prob-space-def product-prob-space-axioms-def product-sigma-finite-def M′-def prob-space-return N′-def assms intro*!: *prob-space-imp-sigma-finite prob-space.prob-space-distr*)
interpret *p′*: *product-prob-space M′*
by(*auto simp*: *product-prob-space-def product-prob-space-axioms-def product-sigma-finite-def M′-def prob-space-return assms intro*!: *prob-space-imp-sigma-finite*)
have *f*[*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *M′ i* →$_M$ *N′ i*
by(*auto simp*: *assms M′-def N′-def*)
have [*simp*]: *p.emb I* = *prod-emb I N′*
by *standard* (*auto simp*: *prod-emb-def*)
have *distr* (Π$_M$ *i*∈*I*. *M i*) (Π$_M$ *i*∈*I*. *N i*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) = *distr* (Π$_M$ *i*∈*I*. *M′ i*) (Π$_M$ *i*∈*I*. *N′ i*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*))
by(*simp add*: *M′-def N′-def cong*: *PiM-cong*)
also have ... = (Π$_M$ *i*∈*I*. *distr* (*M′ i*) (*N′ i*) (*f i*))
proof(*rule p.PiM-eq*)
fix *J F*
assume *h*[*measurable*]: *finite J J* ⊆ *I* ⋀*j*. *j* ∈ *J* ⟹ *F j* ∈ *p.M.events j*
then have [*measurable*]: ⋀*j*. *j* ∈ *J* ⟹ *F j* ∈ *sets* (*N′ j*) **by** *simp*
show *emeasure* (*distr* (*Pi$_M$ I M′*) (*Pi$_M$ I N′*) (λ*xi*. λ*i*∈*I*. *f i* (*xi i*))) (*p.emb I J* (*Pi$_E$ J F*)) = (∏*j*∈*J*. *emeasure* (*distr* (*M′ j*) (*N′ j*) (*f j*)) (*F j*)) (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *emeasure* (*Pi$_M$ I M′*) ((λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) −‘ (*prod-emb I N′ J* (*Pi$_E$ J F*)) ∩ *space* (*Pi$_M$ I M′*))
by(*simp add*: *emeasure-distr h*)
also have ... = *emeasure* (*Pi$_M$ I M′*) (*prod-emb I M′ J* (Π$_E$ *i*∈*J*. *f i* −‘ (*F i*) ∩ *space* (*M′ i*)))
proof −
have [*simp*]:(λ*xi*. λ*i*∈*I*. *f i* (*xi i*)) −‘ (*prod-emb I N′ J* (*Pi$_E$ J F*)) ∩ *space* (*Pi$_M$ I M′*) = *prod-emb I M′ J* (Π$_E$ *i*∈*J*. *f i* −‘ (*F i*) ∩ *space* (*M′ i*))
using *measurable-space*[*OF f*] *h*(*1*,*2*,*3*)
by(*fastforce simp*: *space-PiM prod-emb-def PiE-def extensional-def Pi-def M′-def N′-def*)
show *?thesis* **by** *simp*
qed
also have ... = (∏ *i*∈*J*. *emeasure* (*M′ i*) (*f i* −‘ (*F i*) ∩ *space* (*M′ i*)))
by(*rule p′.emeasure-PiM-emb*,*insert h*(*2*)) (*auto simp*: *h*(*1*))
also have ... = *?rhs*
using *h*(*2*) **by**(*auto simp*: *emeasure-distr intro*!: *comm-monoid-mult-class.prod.cong*)
finally show *?thesis* **.**
qed
qed (*simp cong*: *sets-PiM-cong*)
also have ... = (Π$_M$ *i*∈*I*. *distr* (*M i*) (*N i*) (*f i*))

by(*simp add*: *M'-def N'-def cong*: *distr-cong PiM-cong*)
finally show *?thesis* **.**
qed

end

2 Kernels

theory *Kernels*
imports *Lemmas-S-Finite-Measure-Monad*
begin

2.1 S-Finite Measures

locale *s-finite-measure* =
fixes *M* :: *'a measure*
assumes *s-finite-sum*: ∃ *Mi* :: *nat* ⇒ *'a measure*. (∀ *i*. *sets* (*Mi i*) = *sets M*) ∧ (∀ *i*. *finite-measure* (*Mi i*)) ∧ (∀ *A*∈*sets M*. *M A* = (∑ *i*. *Mi i A*))

lemma(**in** *sigma-finite-measure*) *s-finite-measure*: *s-finite-measure M*
proof
obtain *A* :: *nat* ⇒ - **where** *A*: *range A* ⊆ *sets M* ⋃ (*range A*) = *space M* ⋀*i*. *emeasure M* (*A i*) ≠ ∞ *disjoint-family A*
by(*metis sigma-finite-disjoint*)
define *Mi* **where** *Mi* ≡ (λ*i*. *measure-of* (*space M*) (*sets M*) (λ*a*. *M* (*a* ∩ *A i*)))
have *emeasure-Mi*:*Mi i a* = *M* (*a* ∩ *A i*) **if** *a* ∈ *sets M* **for** *i a*
proof −
have *positive* (*sets* (*Mi i*)) (λ*a*. *M* (*a* ∩ *A i*)) *countably-additive* (*sets* (*Mi i*)) (λ*a*. *M* (*a* ∩ *A i*))
unfolding *positive-def countably-additive-def*
proof *safe*
fix *B* :: *nat* ⇒ -
assume *range B* ⊆ *sets* (*Mi i*) *disjoint-family B*
with *A*(*1*) **have** *range* (λ*j*. *B j* ∩ *A i*) ⊆ *sets M disjoint-family* (λ*j*. *B j* ∩ *A i*)
by(*fastforce simp*: *Mi-def disjoint-family-on-def*)+
thus (∑ *j*. *M* (*B j* ∩ *A i*)) = *M* (⋃ (*range B*) ∩ *A i*)
by (*metis UN-extend-simps*(*4*) *suminf-emeasure*)
qed *simp*
from *emeasure-measure-of*[*OF - - this*] *that* **show** *?thesis*
by(*auto simp add*: *Mi-def sets.space-closed*)
qed
have *sets-Mi*:*sets* (*Mi i*) = *sets M* **for** *i* **by**(*simp add*: *Mi-def*)
show ∃ *Mi*. (∀ *i*. *sets* (*Mi i*) = *sets M*) ∧ (∀ *i*. *finite-measure* (*Mi i*)) ∧ (∀ *A*∈*sets M*. *emeasure M A* = (∑ *i*. *emeasure* (*Mi i*) *A*))
proof(*safe intro*!: *exI*[**where** *x*=*Mi*])
fix *i*
show *finite-measure* (*Mi i*)
using *A* **by**(*auto intro*!: *finite-measureI simp*: *sets-eq-imp-space-eq*[*OF sets-Mi*]

emeasure-Mi)
next
fix *B*
assume *B*:*B* ∈ *sets M*
with *A*(*1*,*4*) **have** *range* (λ*i*. *B* ∩ *A i*) ⊆ *sets M disjoint-family* (λ*i*. *B* ∩ *A i*)
by(*auto simp*: *disjoint-family-on-def*)
then show *M B* = (∑ *i*. (*Mi i*) *B*)
by(*simp add*: *emeasure-Mi*[*OF B*] *suminf-emeasure A*(*2*) *B*)
qed(*simp-all add*: *sets-Mi*)
qed

lemmas(**in** *finite-measure*) *s-finite-measure-finite-measure* = *s-finite-measure*

lemmas(**in** *subprob-space*) *s-finite-measure-subprob* = *s-finite-measure*

lemmas(**in** *prob-space*) *s-finite-measure-prob* = *s-finite-measure*

sublocale *sigma-finite-measure* ⊆ *s-finite-measure*
by(*rule s-finite-measure*)

lemma *s-finite-measureI*:
assumes ⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. *finite-measure* (*Mi i*) ⋀*A*. *A*∈*sets M* ⟹ *M A* = (∑ *i*. *Mi i A*)
shows *s-finite-measure M*
by *standard* (*use assms* **in** *blast*)

lemma *s-finite-measure-prodI*:
assumes ⋀*i j*. *sets* (*Mij i j*) = *sets M* ⋀*i j*. *Mij i j* (*space M*) < ∞ ⋀*A*. *A* ∈ *sets M* ⟹ *M A* = (∑ *i*. (∑ *j*. *Mij i j A*))
shows *s-finite-measure M*
proof −
define *Mi'* **where** *Mi'* ≡ (λ*n*. *case-prod Mij* (*prod-decode n*))
have *sets-Mi'*[*measurable-cong*]:⋀*i*. *sets* (*Mi' i*) = *sets M*
using *assms*(*1*) **by**(*simp add*: *Mi'-def split-beta'*)
have *Mi'-finite*:⋀*i*. *finite-measure* (*Mi' i*)
using *assms*(*2*) *sets-eq-imp-space-eq*[*OF sets-Mi'*[*symmetric*]] *top.not-eq-extremum*
by(*fastforce intro*!: *finite-measureI simp*: *Mi'-def split-beta'*)
show *?thesis*
proof(*safe intro*!: *s-finite-measureI*[**where** *Mi*=*Mi'*] *sets-Mi' Mi'-finite*)
fix *A*
show *A* ∈ *sets M* ⟹ *M A* = (∑ *i*. *Mi' i A*)
by(*simp add*: *assms*(*3*) *suminf-ennreal-2dimen*[**where** *f*=λ(*x*,*y*). *Mij x y A*, *simplified*,*OF refl*,*symmetric*] *Mi'-def split-beta'*)
qed
qed

corollary *s-finite-measure-s-finite-sumI*:
assumes ⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. *s-finite-measure* (*Mi i*) ⋀*A*. *A* ∈ *sets M* ⟹ *M A* = (∑ *i*. *Mi i A*)

shows *s-finite-measure M*
proof −
from *s-finite-measure.s-finite-sum*[*OF assms*(*2*)]
obtain *Mij* **where** *Mij*[*measurable*]: $\bigwedge i\ j.\ sets\ (Mij\ i\ j) = sets\ M\ \bigwedge i\ j.\ finite\text{-}measure\ (Mij\ i\ j)\ \bigwedge i\ j\ A.\ A \in sets\ M \Longrightarrow Mi\ i\ A = (\sum j.\ Mij\ i\ j\ A)$
by (*metis assms*(*1*))
show *?thesis*
using *finite-measure.emeasure-finite*[*OF Mij*(*2*)]
by(*auto intro*!: *s-finite-measure-prodI*[**where** *Mij* = *Mij*] *simp*: *assms*(*3*) *Mij top.not-eq-extremum*)
qed

lemma *s-finite-measure-finite-sumI*:
assumes *finite I* $\bigwedge i.\ i \in I \Longrightarrow$ *s-finite-measure* (*Mi i*) $\bigwedge i.\ i \in I \Longrightarrow$ *sets* (*Mi i*) = *sets M*
and $\bigwedge A.\ A \in sets\ M \Longrightarrow M\ A = (\sum i{\in}I.\ Mi\ i\ A)$
shows *s-finite-measure M*
proof −
let *?Mi* = $\lambda n.$ *if n* < *card I then Mi* (*from-nat-into I n*) *else null-measure M*
show *?thesis*
proof(*rule s-finite-measure-s-finite-sumI*[*of ?Mi*])
show $\bigwedge i.$ *s-finite-measure* (*?Mi i*)
by (*metis* (*full-types*) *assms*(*2*) *bot-nat-0.extremum-strict card.empty emeasure-null-measure ennreal-top-neq-zero finite-measure.s-finite-measure-finite-measure finite-measureI from-nat-into infinity-ennreal-def*)
next
show $\bigwedge i.$ *sets* (*?Mi i*) = *sets M*
by (*metis* (*full-types*) *assms*(*3*) *card-gt-0-iff dual-order.strict-trans2 from-nat-into less-zeroE linorder-le-less-linear sets-null-measure*)
next
fix *A*
assume *A* ∈ *sets M*
then have *M A* = $(\sum i{\in}I.\ Mi\ i\ A)$
by *fact*
also have ... = $(\sum n{<}card\ I.\ Mi\ (from\text{-}nat\text{-}into\ I\ n)\ A)$
by(*rule sum.card-from-nat-into*[*symmetric*])
also have ... = $(\sum n{<}card\ I.\ ?Mi\ n\ A)$
by *simp*
also have ... = $(\sum n.\ ?Mi\ n\ A)$
by(*rule suminf-finite*[*symmetric*]) *auto*
finally show *M A* = $(\sum n.\ ?Mi\ n\ A)$ **.**
qed
qed

lemma *countable-space-s-finite-measure*:
assumes *countable* (*space M*) *sets M* = *Pow* (*space M*)
shows *s-finite-measure M*
proof −
define *Mi* **where** *Mi* ≡ (λ*i. measure-of* (*space M*) (*sets M*) (λ*A. emeasure M*

```
(A ∩ {from-nat-into (space M) i})))
  have sets-Mi[measurable-cong,simp]: sets (Mi i) = sets M for i
    by(auto simp: Mi-def)
  have emeasure-Mi: emeasure (Mi i) A = emeasure M (A ∩ {from-nat-into (space
M) i}) if [measurable]: A ∈ sets M and i:i ∈ to-nat-on (space M) ` (space M) for
i A
  proof −
    have from-nat-into (space M) i ∈ space M
      by (simp add: from-nat-into-def i inv-into-into)
    hence [measurable]: {from-nat-into (space M) i} ∈ sets M
      using assms(2) by auto
     have 1:countably-additive (sets M) (λA. emeasure M (A ∩ {from-nat-into
(space M) i}))
      unfolding countably-additive-def
    proof safe
      fix B :: nat ⇒ -
      assume range B ⊆ sets M disjoint-family B
       then have [measurable]:⋀i. B i ∈ sets M and disjoint-family (λj. B j ∩
{from-nat-into (space M) i})
        by(auto simp: disjoint-family-on-def)
     then have (∑ j. emeasure M (B j ∩ {from-nat-into (space M) i})) = emeasure
M (⋃ (range (λj. B j ∩ {from-nat-into (space M) i})))
        by(intro suminf-emeasure) auto
      thus (∑ j. emeasure M (B j ∩ {from-nat-into (space M) i})) = emeasure M
(⋃ (range B) ∩ {from-nat-into (space M) i})
        by simp
    qed
    have 2:positive (sets M) (λA. emeasure M (A ∩ {from-nat-into (space M) i}))
      by(auto simp: positive-def)
    show ?thesis
     by(simp add: Mi-def emeasure-measure-of-sigma[OF sets.sigma-algebra-axioms
2 1])
  qed
  define Mi' where Mi' ≡ (λi. if i ∈ to-nat-on (space M) ` (space M) then Mi i
else null-measure M)
  have [measurable-cong, simp]: sets (Mi' i) = sets M for i
    by(auto simp: Mi'-def)
  show ?thesis
  proof(rule s-finite-measure-s-finite-sumI[where Mi=Mi'])
    fix A
    assume A[measurable]: A ∈ sets M
    show emeasure M A = (∑ i. emeasure (Mi' i) A) (is ?lhs = ?rhs)
    proof −
      have ?lhs = (∫⁺ x. emeasure M {x} ∂count-space A)
       using sets.sets-into-space[OF A] by(auto intro!: emeasure-countable-singleton
simp: assms(2) countable-subset[OF - assms(1)])
     also have ... = (∫⁺ x. emeasure (Mi (to-nat-on (space M) x)) A ∂count-space
A)
      proof(safe intro!: nn-integral-cong)
```

```
      fix x
      assume x ∈ space (count-space A)
      then have 1:x ∈ A by simp
      hence 2:to-nat-on (space M) x ∈ to-nat-on (space M) ‘ (space M)
        using A assms(2) by auto
      have [simp]: from-nat-into (space M) (to-nat-on (space M) x) = x
       by (metis 1 2 A assms(1) eq-from-nat-into-iff in-mono sets.sets-into-space)
      show emeasure M {x} = emeasure (Mi (to-nat-on (space M) x)) A
        using 1 by(simp add: emeasure-Mi[OF A 2])
    qed
      also have ... = (∫+ x∈A. emeasure (Mi (to-nat-on (space M) x)) A
∂count-space UNIV)
      by (simp add: nn-integral-count-space-indicator)
   also have ... = (∫+ i∈to-nat-on (space M) ‘ A. emeasure (Mi i) A ∂count-space
UNIV)
    by(rule nn-integral-count-compose-inj[OF inj-on-subset[OF inj-on-to-nat-on[OF
assms(1)] sets.sets-into-space[OF A]]])
      also have ... = (∫+ i∈to-nat-on (space M) ‘ A. emeasure (Mi' i) A
∂count-space UNIV)
    proof −
      {
        fix x
        assume x ∈ A
        then have to-nat-on (space M) x ∈ to-nat-on (space M) ‘ (space M)
          using sets.sets-into-space[OF A] by auto
     hence emeasure (Mi (to-nat-on (space M) x)) A = emeasure (Mi' (to-nat-on
(space M) x)) A
          by(auto simp: Mi'-def)
      }
      thus ?thesis
        by(auto intro!: nn-integral-cong simp: indicator-def)
    qed
    also have ... = (∫+ i. emeasure (Mi' i) A ∂count-space UNIV)
    proof −
      {
        fix i
        assume i:i ∉ to-nat-on (space M) ‘ A
         have from-nat-into (space M) i ∉ A if i ∈ to-nat-on (space M) ‘ (space
M)
          by (metis i image-eqI that to-nat-on-from-nat-into)
        with emeasure-Mi have emeasure (Mi' i) A = 0
          by(auto simp: Mi'-def)
      }
      thus ?thesis
        by(auto intro!: nn-integral-cong simp: indicator-def)
    qed
    also have ... = ?rhs
      by(rule nn-integral-count-space-nat)
    finally show ?thesis .
```

```
    qed
  next
    fix i
    show s-finite-measure (Mi' i)
    proof −
      {
        fix x
        assume h:x ∈ space M i = to-nat-on (space M) x
        then have i:i ∈ to-nat-on (space M) ‘ space M
          by blast
        have x: from-nat-into (space M) i = x
          using h by (simp add: assms(1))
        consider M {x} = 0 | M {x} ≠ 0 M {x} < ∞ | M {x} = ∞
          using top.not-eq-extremum by fastforce
        hence s-finite-measure (Mi (to-nat-on (space M) x))
        proof cases
          case 1
          then have [simp]:Mi i = null-measure M
           by(auto intro!: measure-eqI simp: emeasure-Mi[OF - i] x Int-insert-right)
          show ?thesis
         by(auto simp: h(2)[symmetric] intro!: finite-measure.s-finite-measure-finite-measure
finite-measureI)
        next
          case 2
          then show ?thesis
            unfolding h(2)[symmetric]
         by(auto intro!: finite-measure.s-finite-measure-finite-measure finite-measureI
simp: sets-eq-imp-space-eq[OF sets-Mi] emeasure-Mi[OF - i] x h(1))
        next
          case 3
          show ?thesis
            unfolding h(2)[symmetric] s-finite-measure-def
          proof(safe intro!: exI[where x=λj. return M x] prob-space.finite-measure
prob-space-return h(1))
            fix A
            assume A ∈ sets (Mi i)
            then have [measurable]: A ∈ sets M
              by(simp add: Mi-def)
            thus emeasure (Mi i) A = (∑ i. emeasure (return M x) A)
                 by(simp add: emeasure-Mi[OF - i] x) (cases x ∈ A,auto simp: 3
nn-integral-count-space-nat[symmetric])
          qed(auto simp: Mi-def)
        qed
      }
      thus ?thesis
     by(auto simp: Mi'-def) (auto intro!: finite-measure.s-finite-measure-finite-measure
finite-measureI)
    qed
  qed simp
```

qed

lemma *s-finite-measure-subprob-space*:
s-finite-measure M $\longleftrightarrow$ ($\exists$ *Mi* :: *nat* $\Rightarrow$ *'a measure*. ($\forall$ *i*. *sets* (*Mi i*) = *sets M*) $\wedge$ ($\forall$ *i*. (*Mi i*) (*space M*) $\leq$ *1*) $\wedge$ ($\forall$ *A*$\in$*sets M*. *M A* = ($\sum$ *i*. *Mi i A*)))
proof
assume $\exists$ *Mi*. ($\forall$ *i*. *sets* (*Mi i*) = *sets M*) $\wedge$ ($\forall$ *i*. *emeasure* (*Mi i*) (*space M*) $\leq$ *1*) $\wedge$ ($\forall$ *A*$\in$*sets M*. *M A* = ($\sum$ *i*. (*Mi i*) *A*))
then obtain *Mi* **where** *1*:$\bigwedge$*i*. *sets* (*Mi i*) = *sets M* $\bigwedge$*i*. *emeasure* (*Mi i*) (*space M*) $\leq$ *1* ($\forall$ *A*$\in$*sets M*. *M A* = ($\sum$ *i*. (*Mi i*) *A*))
by *auto*
thus *s-finite-measure M*
by(*auto simp*: *s-finite-measure-def sets-eq-imp-space-eq*[*OF 1*(*1*)] *intro*!: *finite-measureI exI*[**where** *x*=*Mi*]) (*metis ennreal-one-less-top linorder-not-le*)
next
assume *s-finite-measure M*
then obtain *Mi'* **where** *Mi'*: $\bigwedge$*i*. *sets* (*Mi' i*) = *sets M* $\bigwedge$*i*. *finite-measure* (*Mi' i*) $\bigwedge$*A*. *A*$\in$*sets M* $\Longrightarrow$ *M A* = ($\sum$ *i*. *Mi' i A*)
by (*metis s-finite-measure.s-finite-sum*)
obtain *u* **where** *u*:$\bigwedge$*i*. *u i* < ∞ $\bigwedge$*i*. *Mi' i* (*space M*) = *u i*
using *Mi'*(*2*) *finite-measure.emeasure-finite top.not-eq-extremum* **by** *fastforce*
define *Mmn* **where** *Mmn* $\equiv$ (λ(*m*,*n*). *if n* < *nat* $\lceil$*enn2real* (*u m*)$\rceil$ *then scale-measure* (*1* / *ennreal* (*real-of-int* $\lceil$*enn2real* (*u m*)$\rceil$)) (*Mi' m*) *else* (*sigma* (*space M*) (*sets M*)))
have *sets-Mmn* : *sets* (*Mmn k*) = *sets M* **for** *k* **by**(*simp add*: *Mmn-def split-beta Mi'*)
have *emeasure-Mmn*: (*Mmn* (*m*, *n*)) *A* = (*Mi' m A*) / *ennreal* (*real-of-int* $\lceil$*enn2real* (*u m*)$\rceil$) **if** *n* < *nat* $\lceil$*enn2real* (*u m*)$\rceil$ *A* $\in$ *sets M* **for** *n m A*
by(*auto simp*: *Mmn-def that ennreal-divide-times*)
have *emeasure-Mmn-less1*: (*Mmn* (*m*, *n*)) *A* $\leq$ *1* **for** *m n A*
proof (*cases n* < *nat* $\lceil$*enn2real* (*u m*)$\rceil$ $\wedge$ *A* $\in$ *sets M*)
case *h*:*True*
have (*Mi' m*) *A* $\leq$ *ennreal* (*real-of-int* $\lceil$*enn2real* (*u m*)$\rceil$)
by(*rule order.trans*[*OF emeasure-mono*[*OF sets.sets-into-space sets.top*]]) (*insert u*(*1*) *h*, *auto simp*: *u*(*2*)[*symmetric*] *enn2real-le top.not-eq-extremum sets-eq-imp-space-eq*[*OF Mi'*(*1*)] *Mi'*(*1*))
with *h* **show** *?thesis*
by(*simp add*: *emeasure-Mmn*) (*metis divide-le-posI-ennreal dual-order.eq-iff ennreal-zero-divide mult.right-neutral not-gr-zero zero-le*)
qed(*auto simp*: *Mmn-def emeasure-sigma emeasure-notin-sets Mi'*)
have *Mi'-sum*:*Mi' m A* = ($\sum$ *n*. *Mmn* (*m*, *n*) *A*) **if** *A* $\in$ *sets M* **for** *m A*
proof −
have ($\sum$ *n*. *Mmn* (*m*, *n*) *A*) = ($\sum$ *n*. *Mmn* (*m*, *n* + *nat* $\lceil$*enn2real* (*u m*)$\rceil$) *A*) + ($\sum$ *n*< *nat* $\lceil$*enn2real* (*u m*)$\rceil$. *Mmn* (*m*, *n*) *A*)
by(*simp add*: *suminf-offset*[**where** *f*=λ*n*. *Mmn* (*m*, *n*) *A*])
also have ... = ($\sum$ *n*< *nat* $\lceil$*enn2real* (*u m*)$\rceil$. *Mmn* (*m*, *n*) *A*)
by(*simp add*: *emeasure-sigma Mmn-def*)
also have ... = ($\sum$ *n*< *nat* $\lceil$*enn2real* (*u m*)$\rceil$. (*Mi' m A*) / *ennreal* (*real-of-int* $\lceil$*enn2real* (*u m*)$\rceil$))

by(*rule Finite-Cartesian-Product.sum-cong-aux*) (*auto simp*: *emeasure-Mmn that*)
also have ... = *Mi' m A*
proof (*cases nat* ⌈*enn2real* (*u m*)⌉)
case *0*
with *u*[*of m*] **show** *?thesis*
by *simp* (*metis Mi'*(*1*) *emeasure-mono enn2real-positive-iff less-le-not-le linorder-less-linear not-less-zero sets.sets-into-space sets.top that*)
next
case (*Suc n'*)
then have *ennreal* (*real-of-int* ⌈*enn2real* (*u m*)⌉) > *0*
using *ennreal-less-zero-iff* **by** *fastforce*
with *u*(*1*)[*of m*] **have** *of-nat* (*nat* ⌈*enn2real* (*u m*)⌉) / *ennreal* (*real-of-int* ⌈*enn2real* (*u m*)⌉) = *1*
by (*simp add*: *ennreal-eq-0-iff ennreal-of-nat-eq-real-of-nat*)
thus *?thesis*
by (*simp add*: *ennreal-divide-times*[*symmetric*])
qed
finally show *?thesis* **..**
qed
define *Mi* **where** *Mi* ≡ (λ*i*. *Mmn* (*prod-decode i*))
show ∃ *Mi*. (∀ *i*. *sets* (*Mi i*) = *sets M*) ∧ (∀ *i*. *emeasure* (*Mi i*) (*space M*) ≤ *1*) ∧ (∀ *A*∈*sets M*. *M A* = (∑ *i*. (*Mi i*) *A*))
by(*auto intro*!: *exI*[**where** *x*=*Mi*] *simp*: *Mi-def sets-Mmn suminf-ennreal-2dimen*[*OF Mi'-sum*] *Mi'*(*3*)) (*metis emeasure-Mmn-less1 prod-decode-aux.cases*)
qed

lemma(**in** *s-finite-measure*) *finite-measures*:
obtains *Mi* **where** ⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. (*Mi i*) (*space M*) ≤ *1* ⋀*A*. *M A* = (∑ *i*. *Mi i A*)
proof −
obtain *Mi* **where** *Mi*:⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. (*Mi i*) (*space M*) ≤ *1* ⋀*A*. *A* ∈ *sets M* ⟹ *M A* = (∑ *i*. *Mi i A*)
using *s-finite-measure-axioms* **by**(*metis s-finite-measure-subprob-space*)
hence *M A* = (∑ *i*. *Mi i A*) **for** *A*
by(*cases A* ∈ *sets M*) (*auto simp*: *emeasure-notin-sets*)
with *Mi*(*1*,*2*) **show** *?thesis*
using *that* **by** *blast*
qed

lemma(**in** *s-finite-measure*) *finite-measures-ne*:
assumes *space M* ≠ {}
obtains *Mi* **where** ⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. *subprob-space* (*Mi i*) ⋀*A*. *M A* = (∑ *i*. *Mi i A*)
by (*metis assms finite-measures sets-eq-imp-space-eq subprob-spaceI*)

lemma(**in** *s-finite-measure*) *finite-measures'*:
obtains *Mi* **where** ⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. *finite-measure* (*Mi i*) ⋀*A*. *M A* = (∑ *i*. *Mi i A*)

by (*metis ennreal-top-neq-one finite-measureI finite-measures infinity-ennreal-def sets-eq-imp-space-eq top.extremum-uniqueI*)

lemma(**in** *s-finite-measure*) *s-finite-measure-distr*:
assumes *f*[*measurable*]:$f \in M \to_M N$
shows *s-finite-measure* (*distr M N f*)
proof −
obtain *Mi* **where** *Mi*[*measurable-cong*]:$\bigwedge i.\ sets\ (Mi\ i) = sets\ M \bigwedge i.\ finite\text{-}measure\ (Mi\ i) \bigwedge A.\ M\ A = (\sum i.\ Mi\ i\ A)$
by(*metis finite-measures*′)
show *?thesis*
by(*auto intro*!: *s-finite-measureI*[**where** *Mi*=($\lambda i.\ distr\ (Mi\ i)\ N\ f$)] *finite-measure.finite-measure-distr*[*OF Mi*(*2*)] *simp*: *emeasure-distr Mi*(*3*) *sets-eq-imp-space-eq*[*OF Mi*(*1*)])
qed

lemma *nn-integral-measure-suminf*:
assumes [*measurable-cong*]:$\bigwedge i.\ sets\ (Mi\ i) = sets\ M$ **and** $\bigwedge A.\ A \in sets\ M \Longrightarrow M\ A = (\sum i.\ Mi\ i\ A)$ $f \in borel\text{-}measurable\ M$
shows $(\sum i.\ \int^+ x.\ f\ x\ \partial(Mi\ i)) = (\int^+ x.\ f\ x\ \partial M)$
using *assms*(*3*)
proof *induction*
case (*cong f g*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *nn-integral-cong sets-eq-imp-space-eq suminf-cong*)
next
case (*set A*)
then show *?case*
using *assms*(*1*,*2*) **by** *simp*
next
case (*mult u c*)
then show *?case*
by(*simp add*: *nn-integral-cmult*)
next
case (*add u v*)
then show *?case*
by(*simp add*: *nn-integral-add suminf-add*[*symmetric*])
next
case *ih*:(*seq U*)
have $(\sum i.\ integral^N\ (Mi\ i)\ (\bigsqcup\ range\ U)) = (\sum i.\ \int^+ x.\ (\bigsqcup j.\ U\ j\ x)\ \ \partial(Mi\ i))$
by(*auto intro*!: *suminf-cong*) (*metis SUP-apply*)
also have ... $= (\sum i.\ \bigsqcup j.\ \int^+ x.\ U\ j\ x\ \partial(Mi\ i))$
using *ih* **by**(*auto intro*!: *suminf-cong nn-integral-monotone-convergence-SUP*)
also have ... $= (\bigsqcup j.\ (\sum i.\ \int^+ x.\ U\ j\ x\ \partial(Mi\ i)))$
using *ih*(*3*) **by**(*auto intro*!: *ennreal-suminf-SUP-eq incseq-nn-integral*)
also have ... $= (\bigsqcup j.\ integral^N\ M\ (U\ j))$
by(*simp add*: *ih*)
also have ... $= (\int^+ x.\ (\bigsqcup j.\ U\ j\ x)\ \partial M)$
using *ih* **by**(*auto intro*!: *nn-integral-monotone-convergence-SUP*[*symmetric*])

also have ... = $integral^N\ M\ (\bigsqcup\ range\ U)$
by(*metis SUP-apply*)
finally show *?case* **.**
qed

A *density M f* of *s*-finite measure M and $f \in$ *borel-measurable M* is again s-finite. We do not require additional assumption, unlike σ-finite measures.

lemma(**in** *s-finite-measure*) *s-finite-measure-density*:
assumes *f*[*measurable*]:*f* ∈ *borel-measurable M*
shows *s-finite-measure* (*density M f*)
proof −
obtain *Mi* **where** *Mi*[*measurable-cong*]:⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. *finite-measure* (*Mi i*) ⋀*A*. *M A* = (∑ *i*. *Mi i A*)
by(*metis finite-measures*′)
show *?thesis*
proof(*rule s-finite-measure-s-finite-sumI*[**where** *Mi*=λ*i*. *density* (*Mi i*) *f*])
show *s-finite-measure* (*density* (*Mi i*) *f*) **for** *i*
proof −
define *Mij* **where** *Mij* = (λ*j*::*nat*. *if j* = *0 then density* (*Mi i*) (λ*x*. ∞ ∗ *indicator* {*x*∈*space M*. *f x* = ∞} *x*)
else if j = *1 then density* (*Mi i*) (λ*x*. *f x* ∗ *indicator* {*x*∈*space M*. *f x* < ∞} *x*)
else null-measure M)
have *sets-Mij*[*measurable-cong*]: *sets* (*Mij j*) = *sets M* **for** *j*
by(*auto simp*: *Mij-def Mi*)
have *emeasure-Mi*:*density* (*Mi i*) *f A* = (∑ *j*. *Mij j A*) (**is** *?lhs* = *?rhs*) **if** *A*[*measurable*]: *A* ∈ *sets M* **for** *A*
proof −
have *?lhs* = (∫ ⁺*x* ∈ *A*. *f x* ∂*Mi i*)
by(*simp add*: *emeasure-density*)
also have ... = (∫ ⁺*x*. ∞ ∗ *indicator* {*x*∈*space M*. *f x* = ∞} *x* ∗ *indicator A x* + *f x* ∗ *indicator* {*x*∈*space M*. *f x* < ∞} *x* ∗ *indicator A x* ∂*Mi i*)
by(*auto intro*!: *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF Mi*(*1*)] *indicator-def*) (*simp add*: *top.not-eq-extremum*)
also have ... = *density* (*Mi i*) (λ*x*. ∞ ∗ *indicator* {*x*∈*space M*. *f x* = ∞} *x*) *A* + *density* (*Mi i*) (λ*x*. *f x* ∗ *indicator* {*x*∈*space M*. *f x* < ∞} *x*) *A*
by(*simp add*: *nn-integral-add emeasure-density*)
also have ... = *?rhs*
using *suminf-finite*[*of* {..<*Suc* (*Suc 0*)} λ*j*. *Mij j A*] **by**(*auto simp*: *Mij-def*)
finally show *?thesis* **.**
qed
show *?thesis*
proof(*rule s-finite-measure-s-finite-sumI*[*OF* - - *emeasure-Mi*])
fix *j* :: *nat*
consider *j* = *0* | *j* = *1* | *j* ≠ *0 j* ≠ *1* **by** *auto*
then show *s-finite-measure* (*Mij j*)
proof *cases*
case *1*
have *2*:*Mij j A* = (∑ *k*. *density* (*Mi i*) (*indicator* {*x*∈*space M*. *f x* = ⊤})

A) **if** *A*[*measurable*]:*A* ∈ *sets M* **for** *A*
by(*auto simp*: *Mij-def 1 emeasure-density nn-integral-suminf*[*symmetric*] *sets-eq-imp-space-eq*[*OF Mi*(*1*)] *indicator-def intro*!: *nn-integral-cong*) (*simp add*: *nn-integral-count-space-nat*[*symmetric*])
show *?thesis*
by(*auto simp*: *s-finite-measure-def 2 Mi*(*1*)[*of i*] *sets-Mij*[*of j*] *intro*!: *exI*[**where** *x*=λ*k*. *density* (*Mi i*) (*indicator* {*x*∈*space M*. *f x* = ∞})] *finite-measure.finite-measure-restricted Mi*(*2*))
next
case *2*
show *?thesis*
by(*auto intro*!: *sigma-finite-measure.s-finite-measure AE-mono-measure*[*OF Mi*(*1*)] *sum-le-suminf*[**where** *I*={*i*} **and** *f*=λ*i*. *Mi i* -,*simplified*] *simp*: *sigma-finite-measure.sigma-finite-iff-de finite-measure.sigma-finite-measure*[*OF Mi*(*2*)[*of i*]]] *le-measure*[*OF Mi*(*1*)] *Mi indicator-def 2 Mij-def*) *auto*
next
case *3*
then show *?thesis*
by(*auto simp*: *Mij-def intro*!: *finite-measure.s-finite-measure-finite-measure finite-measureI*)
qed
qed(*auto simp*: *sets-Mij Mi*)
qed
qed(*auto simp*: *emeasure-density nn-integral-measure-suminf*[*OF Mi*(*1*,*3*)] *Mi*(*1*))
qed

lemma
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
assumes [*measurable-cong*]:⋀*i*. *sets* (*Mi i*) = *sets M* **and** ⋀*A*. *A*∈*sets M* ⟹ *M A* = (∑ *i*. *Mi i A*) *integrable M f*
shows *lebesgue-integral-measure-suminf*:(∑ *i*. ∫ *x*. *f x* ∂(*Mi i*)) = (∫ *x*. *f x* ∂*M*) (**is** *?suminf*)
and *lebesgue-integral-measure-suminf-summable-norm*: *summable* (λ*i*. *norm* (∫ *x*. *f x* ∂(*Mi i*))) (**is** *?summable2*)
and *lebesgue-integral-measure-suminf-summable-norm-in*: *summable* (λ*i*. ∫ *x*. *norm* (*f x*) ∂(*Mi i*)) (**is** *?summable*)
proof −
have *Mi*:*Mi i* ≤ *M* **for** *i*
using *assms*(*2*) *ennreal-suminf-lessD linorder-not-le* **by**(*fastforce simp*: *assms*(*1*) *le-measure*[*OF assms*(*1*)])
have *sum2*: *summable* (λ*i*. *norm* (∫ *x*. *g x* ∂(*Mi i*))) **if** *summable* (λ*i*. ∫ *x*. *norm* (*g x*) ∂(*Mi i*)) **for** *g* :: *'a* ⇒ *'b*
proof(*rule summable-suminf-not-top*)
have (∑ *i*. *ennreal* (*norm* (∫ *x*. *g x* ∂(*Mi i*)))) ≤ (∑ *i*. *ennreal* (∫ *x*. *norm* (*g x*) ∂(*Mi i*)))
by(*auto intro*!: *suminf-le*)
thus (∑ *i*. *ennreal* (*norm* (∫ *x*. *g x* ∂(*Mi i*)))) ≠ ⊤
by (*metis ennreal-suminf-neq-top*[*OF that*] *Bochner-Integration.integral-nonneg neq-top-trans norm-ge-zero*)

```
  qed simp
  have ?suminf ∧ ?summable
    using assms(3)
  proof induction
    case h[measurable]:(base A c)
    have Mi-fin:Mi i A < ∞ for i
        by(rule order.strict-trans1[OF - h(2)], auto simp: le-measureD3[OF Mi
assms(1)])
    have 1: (∫ x. (indicat-real A x *R c) ∂Mi i) = measure (Mi i) A *R c for i
      using Mi-fin by simp
    have 2:summable (λi. ∫ x. norm (indicat-real A x *R c) ∂Mi i)
    proof(rule summable-suminf-not-top)
      show (∑ i. ennreal (∫ x. norm (indicat-real A x *R c) ∂Mi i)) ≠ ⊤ (is ?l ≠
-)
      proof −
        have ?l = (∑ i. Mi i A ) * norm c
          using Mi-fin by(auto intro!: suminf-cong simp: measure-def enn2real-mult
ennreal-mult)
        also have ... = M A * norm c
          by(simp add: assms(2))
        also have ... ≠ ⊤
          using h(2) by (simp add: ennreal-mult-less-top top.not-eq-extremum)
        finally show ?thesis .
      qed
    qed simp
    have 3: (∑ i. ∫ x. indicat-real A x *R c ∂Mi i) = (∫ x. indicat-real A x *R c
∂M) (is ?l = ?r)
    proof −
      have [simp]: summable (λi. enn2real (Mi i A))
      using Mi-fin h by(auto intro!: summable-suminf-not-top simp: assms(2)[symmetric])
      have ?l = (∑ i. measure (Mi i) A) *R c
        by(auto intro!: suminf-cong simp: 1 measure-def suminf-scaleR-left)
      also have ... = ?r
        using h(2) Mi-fin by(simp add: ennreal-inj[where a=(∑ i. measure (Mi i)
A) and b=measure M A,OF suminf-nonneg measure-nonneg,symmetric,simplified
measure-def] measure-def suminf-ennreal2[symmetric] assms(2)[symmetric])
      finally show ?thesis .
    qed
    from 2 3 show ?case by simp
  next
    case ih[measurable]:(add f g)
    have 1:summable (λi. ∫ x. norm (f x + g x) ∂Mi i)
    proof(rule summable-suminf-not-top)
      show (∑ i. ennreal (∫ x. norm (f x + g x) ∂Mi i)) ≠ ⊤ (is ?l ≠ -)
      proof −
        have ?l = (∑ i. (∫ +x. ennreal (norm (f x + g x)) ∂Mi i))
          using ih by(auto intro!: suminf-cong nn-integral-eq-integral[symmetric]
integrable-mono-measure[OF assms(1) Mi])
        also have ... ≤ (∑ i. (∫ +x. ennreal (norm (f x) + norm (g x)) ∂Mi i))
```

by(*auto intro*!: *suminf-le nn-integral-mono norm-triangle-ineq simp del*: *ennreal-plus*)
also have ... = $(\sum i.\ (\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial Mi\ i)) + (\sum i.\ (\int^+ x.\ ennreal\ (norm\ (g\ x))\ \partial Mi\ i))$
by(*auto intro*!: *suminf-cong simp*: *nn-integral-add suminf-add*)
also have ... = $(\sum i.\ ennreal\ (\int x.\ norm\ (f\ x)\ \partial Mi\ i)) + (\sum i.\ ennreal\ (\int x.\ norm\ (g\ x)\ \partial Mi\ i))$
using *ih* **by**(*simp add*: *nn-integral-eq-integral integrable-mono-measure*[*OF assms*(*1*) *Mi*])
also have ... < $\top$
using *ennreal-suminf-neq-top*[*OF conjunct2*[*OF ih*(*3*)]] *ennreal-suminf-neq-top*[*OF conjunct2*[*OF ih*(*4*)]]
by (*meson Bochner-Integration.integral-nonneg ennreal-add-eq-top norm-ge-zero top.not-eq-extremum*)
finally show *?thesis*
using *order.strict-iff-order* **by** *blast*
qed
qed *simp*
with *ih* **show** *?case*
by(*auto simp*: *Bochner-Integration.integral-add*[*OF integrable-mono-measure*[*OF assms*(*1*) *Mi ih*(*1*)] *integrable-mono-measure*[*OF assms*(*1*) *Mi ih*(*2*)]] *suminf-add*[*symmetric*,*OF summable-norm-cancel*[*OF sum2*[*OF conjunct2*[*OF ih*(*3*)]]] *summable-norm-cancel*[*OF sum2*[*OF conjunct2*[*OF ih*(*4*)]]]])
next
case *ih*[*measurable*]:(*lim f fn*)
have *1*:*summable* $(\lambda i.\ \int x.\ norm\ (f\ x)\ \partial(Mi\ i))$
proof(*rule summable-suminf-not-top*)
show $(\sum i.\ ennreal\ (\int x.\ norm\ (f\ x)\ \partial(Mi\ i))) \neq \top$ (**is** *?lhs* ≠ -)
proof −
have *?lhs* = $(\sum i.\ \int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial Mi\ i)$
by(*auto intro*!: *suminf-cong nn-integral-eq-integral*[*symmetric*] *integrable-mono-measure*[*OF assms*(*1*) *Mi*] *simp*: *ih*)
also have ... = $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M)$
by(*simp add*: *nn-integral-measure-suminf*[*OF assms*(*1*,*2*)])
also have ... = $ennreal\ (\int x.\ norm\ (f\ x)\ \partial M)$
by(*auto intro*!: *nn-integral-eq-integral ih*(*4*))
also have ... < $\top$ **by** *simp*
finally show *?lhs* ≠ $\top$
using *linorder-neq-iff* **by** *blast*
qed
qed *simp*
have $(\sum i.\ \int x.\ f\ x\ \partial(Mi\ i)) = (\int i.\ \int x.\ f\ x\ \partial(Mi\ i)\ \partial(count\text{-}space\ UNIV))$
by(*rule integral-count-space-nat*[*symmetric*]) (*simp add*: *integrable-count-space-nat-iff sum2*[*OF 1*])
also have ... = $lim\ (\lambda m.\ \int i.\ \int x.\ fn\ m\ x\ \partial(Mi\ i)\ \partial(count\text{-}space\ UNIV))$
proof(*rule limI*[*OF integral-dominated-convergence*[**where** *w*=$\lambda i.\ 2 * (\int x.\ norm\ (f\ x)\ \partial(Mi\ i))$],*symmetric*],*auto simp*: *AE-count-space integrable-count-space-nat-iff 1*)
show $(\lambda m.\ \int x.\ fn\ m\ x\ \partial(Mi\ i)) \longrightarrow \int x.\ f\ x\ \partial(Mi\ i)$ **for** *i*

by(*rule integral-dominated-convergence*[**where** *w*=λ*x*. *2* ∗ *norm* (*f x*)],*insert ih*) (*auto intro*!: *integrable-mono-measure*[*OF assms*(*1*) *Mi*] *simp*: *sets-eq-imp-space-eq*[*OF assms*(*1*)])
next
fix *i j*
show *norm* (∫ *x*. *fn j x* ∂(*Mi i*)) ≤ *2* ∗ (∫ *x*. *norm* (*f x*) ∂(*Mi i*)) (**is** *?l* ≤ *?r*)
proof −
have *?l* ≤ (∫ *x*. *norm* (*fn j x*) ∂(*Mi i*))
by *simp*
also have ... ≤ (∫ *x*. *2* ∗ *norm* (*f x*) ∂(*Mi i*))
by(*rule integral-mono*,*insert ih*) (*auto intro*!: *integrable-mono-measure*[*OF assms*(*1*) *Mi*] *simp*: *sets-eq-imp-space-eq*[*OF assms*(*1*)])
finally show *?l* ≤ *?r* **by** *simp*
qed
qed
also have ... = *lim* (λ*m*. (∑ *i*. ∫ *x*. *fn m x* ∂(*Mi i*)))
proof −
have (∫ *i*. ∫ *x*. *fn m x* ∂(*Mi i*) ∂(*count-space UNIV*)) = (∑ *i*. ∫ *x*. *fn m x* ∂(*Mi i*)) **for** *m*
by(*auto intro*!: *integral-count-space-nat sum2 simp*: *integrable-count-space-nat-iff*) (*use ih*(*5*) **in** *auto*)
thus *?thesis* **by** *simp*
qed
also have ... = *lim* (λ*m*. ∫ *x*. *fn m x* ∂*M*)
by(*simp add*: *ih*(*5*))
also have ... = (∫ *x*. *f x* ∂*M*)
using *ih* **by**(*auto intro*!: *limI*[*OF integral-dominated-convergence*[**where** *w*=λ*x*. *2* ∗ *norm* (*f x*)]])
finally show *?case*
using *1* **by** *auto*
qed
thus *?suminf ?summable ?summable2*
by(*simp-all add*: *sum2*)
qed

lemma (**in** *s-finite-measure*) *measurable-emeasure-Pair'*:
assumes *Q* ∈ *sets* (*N* ⨂$_M$ *M*)
shows (λ*x*. *emeasure M* (*Pair x* −‘ *Q*)) ∈ *borel-measurable N* (**is** *?s Q* ∈ -)
proof −
obtain *Mi* **where** *Mi*:⋀*i*. *sets* (*Mi i*) = *sets M* ⋀*i*. *finite-measure* (*Mi i*) ⋀*A*. *M A* = (∑ *i*. *Mi i A*)
by(*metis finite-measures'*)
show *?thesis*
using *Mi*(*1*,*2*) *assms finite-measure.finite-measure-cut-measurable*[*of Mi - Q N*]
by(*simp add*: *Mi*(*3*))
qed

lemma (**in** *s-finite-measure*) *measurable-emeasure'*[*measurable* (*raw*)]:

assumes *space*: $\bigwedge x.\ x \in space\ N \Longrightarrow A\ x \subseteq space\ M$
assumes *A*: $\{x \in space\ (N \bigotimes_M M).\ snd\ x \in A\ (fst\ x)\} \in sets\ (N \bigotimes_M M)$
shows $(\lambda x.\ emeasure\ M\ (A\ x)) \in borel\text{-}measurable\ N$
proof −
from *space* **have** $\bigwedge x.\ x \in space\ N \Longrightarrow Pair\ x\ -`\ \{x \in space\ (N \bigotimes_M M).\ snd\ x \in A\ (fst\ x)\} = A\ x$
by (*auto simp*: *space-pair-measure*)
with *measurable-emeasure-Pair'*[*OF A*] **show** *?thesis*
by (*auto cong*: *measurable-cong*)
qed

lemma(**in** *s-finite-measure*) *emeasure-pair-measure'*:
assumes $X \in sets\ (N \bigotimes_M M)$
shows $emeasure\ (N \bigotimes_M M)\ X = (\int^+ x.\ \int^+ y.\ indicator\ X\ (x, y)\ \partial M\ \partial N)$
(**is** - = $?\mu\ X$)
proof (*rule emeasure-measure-of*[*OF pair-measure-def*])
show *positive* $(sets\ (N \bigotimes_M M))\ ?\mu$
by (*auto simp*: *positive-def*)
have *eq*[*simp*]: $\bigwedge A\ x\ y.\ indicator\ A\ (x, y) = indicator\ (Pair\ x\ -`\ A)\ y$
by (*auto simp*: *indicator-def*)
show *countably-additive* $(sets\ (N \bigotimes_M M))\ ?\mu$
proof (*rule countably-additiveI*)
fix $F :: nat \Rightarrow ('b \times 'a)\ set$ **assume** *F*: *range F* $\subseteq sets\ (N \bigotimes_M M)$ *disjoint-family F*
from *F* **have** $*$: $\bigwedge i.\ F\ i \in sets\ (N \bigotimes_M M)$ **by** *auto*
moreover have $\bigwedge x.\ disjoint\text{-}family\ (\lambda i.\ Pair\ x\ -`\ F\ i)$
by (*intro disjoint-family-on-bisimulation*[*OF F*(*2*)]) *auto*
moreover have $\bigwedge x.\ range\ (\lambda i.\ Pair\ x\ -`\ F\ i) \subseteq sets\ M$
using *F* **by** (*auto simp*: *sets-Pair1*)
ultimately show $(\sum n.\ ?\mu\ (F\ n)) = ?\mu\ (\bigcup i.\ F\ i)$
by (*auto simp add*: *nn-integral-suminf*[*symmetric*] *vimage-UN suminf-emeasure*
intro!: *nn-integral-cong nn-integral-indicator*[*symmetric*])
qed
show $\{a \times b \mid a\ b.\ a \in sets\ N \wedge b \in sets\ M\} \subseteq Pow\ (space\ N \times space\ M)$
using *sets.space-closed*[*of N*] *sets.space-closed*[*of M*] **by** *auto*
qed *fact*

lemma (**in** *s-finite-measure*) *emeasure-pair-measure-alt'*:
assumes *X*: $X \in sets\ (N \bigotimes_M M)$
shows $emeasure\ (N \bigotimes_M M)\ X = (\int^+ x.\ emeasure\ M\ (Pair\ x\ -`\ X)\ \partial N)$
proof −
have [*simp*]: $\bigwedge x\ y.\ indicator\ X\ (x, y) = indicator\ (Pair\ x\ -`\ X)\ y$
by (*auto simp*: *indicator-def*)
show *?thesis*
using *X* **by** (*auto intro*!: *nn-integral-cong simp*: *emeasure-pair-measure' sets-Pair1*)
qed

proposition (**in** *s-finite-measure*) *emeasure-pair-measure-Times'*:

```
  assumes A: A ∈ sets N and B: B ∈ sets M
  shows emeasure (N ⨂M M) (A × B) = emeasure N A * emeasure M B
proof −
  have emeasure (N ⨂M M) (A × B) = (∫+x. emeasure M B * indicator A x
∂N)
    using A B by (auto intro!: nn-integral-cong simp: emeasure-pair-measure-alt')
  also have ... = emeasure M B * emeasure N A
    using A by (simp add: nn-integral-cmult-indicator)
  finally show ?thesis
    by (simp add: ac-simps)
qed

lemma(in s-finite-measure) measure-times:
  assumes[measurable]: A ∈ sets N B ∈ sets M
  shows measure (N ⨂M M) (A × B) = measure N A * measure M B
  by(auto simp: measure-def emeasure-pair-measure-Times' enn2real-mult)

lemma pair-measure-s-finite-measure-suminf:
  assumes Mi[measurable-cong]:⋀i. sets (Mi i) = sets M ⋀i. finite-measure (Mi
i) ⋀A. M A = (∑ i. Mi i A)
      and Ni[measurable-cong]:⋀i. sets (Ni i) = sets N ⋀i. finite-measure (Ni i)
⋀A. N A = (∑ i. Ni i A)
    shows (M ⨂M N) A = (∑ i j. (Mi i ⨂M Ni j) A) (is ?lhs = ?rhs)
proof −
  interpret N: s-finite-measure N
   by(auto intro!: s-finite-measureI[where Mi=Mi] s-finite-measureI[where Mi=Ni]
assms)
  show ?thesis
  proof(cases A ∈ sets (M ⨂M N))
    case [measurable]:True
    show ?thesis
    proof −
      have ?lhs = (∫+x. N (Pair x −‘ A) ∂M)
        by(simp add: N.emeasure-pair-measure-alt')
      also have ... = (∑ i. ∫+x. N (Pair x −‘ A) ∂Mi i)
        using N.measurable-emeasure-Pair'[of A]
        by(auto intro!: nn-integral-measure-suminf[OF Mi(1,3),symmetric])
      also have ... = (∑ i. ∫+x. (∑ j. Ni j (Pair x −‘ A)) ∂Mi i)
        by(simp add: Ni(3))
      also have ... = (∑ i j. ∫+x. Ni j (Pair x −‘ A) ∂Mi i)
      using s-finite-measure.measurable-emeasure-Pair'[OF finite-measure.s-finite-measure-finite-measure[OF
Ni(2)],of A]
        by(auto simp: nn-integral-suminf intro!: suminf-cong)
      also have ... = ?rhs
      by(auto intro!: suminf-cong simp: s-finite-measure.emeasure-pair-measure-alt'[OF
finite-measure.s-finite-measure-finite-measure[OF Ni(2)]])
      finally show ?thesis .
    qed
  next
```

```
    case False
    with Mi(1) Ni(1) show ?thesis
      by(simp add: emeasure-notin-sets)
  qed
qed

lemma pair-measure-s-finite-measure-suminf':
  assumes Mi[measurable-cong]:⋀i. sets (Mi i) = sets M ⋀i. finite-measure (Mi
i) ⋀A. M A = (∑ i. Mi i A)
      and Ni[measurable-cong]:⋀i. sets (Ni i) = sets N ⋀i. finite-measure (Ni i)
⋀A. N A = (∑ i. Ni i A)
    shows (M ⨂M N) A = (∑ i j. (Mi j ⨂M Ni i) A) (is ?lhs = ?rhs)
proof −
  interpret N: s-finite-measure N
   by(auto intro!: s-finite-measureI[where Mi=Mi] s-finite-measureI[where Mi=Ni]
assms)
  show ?thesis
  proof(cases A ∈ sets (M ⨂M N))
    case [measurable]:True
    show ?thesis
    proof −
      have ?lhs = (∫+x. N (Pair x −‘ A) ∂M)
        by(simp add: N.emeasure-pair-measure-alt')
      also have ... = (∫+x. (∑ i. Ni i (Pair x −‘ A)) ∂M)
        by(auto intro!: nn-integral-cong simp: Ni)
      also have ... = (∑ i. (∫+x. Ni i (Pair x −‘ A) ∂M))
      by(auto intro!: nn-integral-suminf simp: finite-measure.finite-measure-cut-measurable[OF
Ni(2)])
      also have ... = (∑ i j. ∫+x. Ni i (Pair x −‘ A) ∂Mi j)
         by(auto intro!: suminf-cong nn-integral-measure-suminf[symmetric] simp:
finite-measure.finite-measure-cut-measurable[OF Ni(2)] Mi)
      also have ... = ?rhs
      by(auto intro!: suminf-cong simp: s-finite-measure.emeasure-pair-measure-alt'[OF
finite-measure.s-finite-measure-finite-measure[OF Ni(2)]])
      finally show ?thesis .
    qed
  next
    case False
    with Mi(1) Ni(1) show ?thesis
      by(simp add: emeasure-notin-sets)
  qed
qed

lemma pair-measure-s-finite-measure:
  assumes s-finite-measure M and s-finite-measure N
  shows s-finite-measure (M ⨂M N)
proof −
  obtain Mi where Mi[measurable-cong]:⋀i. sets (Mi i) = sets M ⋀i. finite-measure
(Mi i) ⋀A. M A = (∑ i. Mi i A)
```

by(*metis s-finite-measure.finite-measures'*[*OF assms*(*1*)])
obtain *Ni* **where** *Ni*[*measurable-cong*]:$\bigwedge i.$ *sets* (*Ni i*) = *sets N* $\bigwedge i.$ *finite-measure* (*Ni i*) $\bigwedge A.$ *N A* = ($\sum i.$ *Ni i A*)
by(*metis s-finite-measure.finite-measures'*[*OF assms*(*2*)])
show *?thesis*
proof(*rule s-finite-measure-prodI*[**where** *Mij*=$\lambda i\ j.$ *Mi i* $\bigotimes_M$ *Ni j*])
show *emeasure* (*Mi i* $\bigotimes_M$ *Ni j*) (*space* (*M* $\bigotimes_M$ *N*)) < ∞ **for** *i j*
using *finite-measure.emeasure-finite*[*OF Mi*(*2*)[*of i*]] *finite-measure.emeasure-finite*[*OF Ni*(*2*)[*of j*]]
by(*auto simp*: *sets-eq-imp-space-eq*[*OF Mi*(*1*)[*of i*],*symmetric*] *sets-eq-imp-space-eq*[*OF Ni*(*1*)[*of j*],*symmetric*] *space-pair-measure s-finite-measure.emeasure-pair-measure-Times'*[*OF finite-measure.s-finite-measure-finite-measure*[*OF Ni*(*2*)[*of j*]]] *ennreal-mult-less-top top.not-eq-extremum*)
qed(*auto simp*: *pair-measure-s-finite-measure-suminf Mi Ni*)
qed

lemma(**in** *s-finite-measure*) *borel-measurable-nn-integral-fst'*:
assumes [*measurable*]: *f* ∈ *borel-measurable* (*N* $\bigotimes_M$ *M*)
shows ($\lambda x.$ $\int^+ y.$ *f* (*x*, *y*) ∂*M*) ∈ *borel-measurable N*
proof −
obtain *Mi* **where** *Mi*[*measurable-cong*]:$\bigwedge i.$ *sets* (*Mi i*) = *sets M* $\bigwedge i.$ *finite-measure* (*Mi i*) $\bigwedge A.$ *M A* = ($\sum i.$ *Mi i A*)
by(*metis finite-measures'*)
show *?thesis*
by(*rule measurable-cong*[**where** *g*=$\lambda x.$ $\sum i.$ $\int^+ y.$ *f* (*x*, *y*) ∂*Mi i*,*THEN iffD2*])
(*auto simp*: *nn-integral-measure-suminf*[*OF Mi*(*1*,*3*)] *intro*!: *borel-measurable-suminf-order sigma-finite-measure.borel-measurable-nn-integral-fst*[*OF finite-measure.sigma-finite-measure*[*OF Mi*(*2*)]])
qed

lemma (**in** *s-finite-measure*) *nn-integral-fst'*:
assumes *f*: *f* ∈ *borel-measurable* (*M1* $\bigotimes_M$ *M*)
shows ($\int^+ x.$ $\int^+ y.$ *f* (*x*, *y*) ∂*M* ∂*M1*) = *integral*^{N} (*M1* $\bigotimes_M$ *M*) *f* (**is** *?I f* = -)
using *f* **proof** *induct*
case (*cong u v*)
then have *?I u* = *?I v*
by (*intro nn-integral-cong*) (*auto simp*: *space-pair-measure*)
with *cong* **show** *?case*
by (*simp cong*: *nn-integral-cong*)
qed (*simp-all add*: *emeasure-pair-measure' nn-integral-cmult nn-integral-add nn-integral-monotone-convergence-SUP measurable-compose-Pair1 borel-measurable-nn-integral-fst' nn-integral-mono incseq-def le-fun-def image-comp cong*: *nn-integral-cong*)

lemma (**in** *s-finite-measure*) *borel-measurable-nn-integral'*[*measurable* (*raw*)]:
case-prod f ∈ *borel-measurable* (*N* $\bigotimes_M$ *M*) ⟹ ($\lambda x.$ $\int^+ y.$ *f x y* ∂*M*) ∈ *borel-measurable N*

using *borel-measurable-nn-integral-fst'*[*of case-prod f N*] **by** *simp*

lemma *distr-pair-swap-s-finite*:
assumes *s-finite-measure M1* **and** *s-finite-measure M2*
shows *M1* $\bigotimes_M$ *M2* = *distr* (*M2* $\bigotimes_M$ *M1*) (*M1* $\bigotimes_M$ *M2*) (λ(*x*, *y*). (*y*, *x*)) (**is** *?P* = *?D*)
proof −
{
from *s-finite-measure.finite-measures'*[*OF assms*(*1*)] *s-finite-measure.finite-measures'*[*OF assms*(*2*)]
obtain *Mi1 Mi2*
where *Mi1*:⋀*i*. *sets* (*Mi1 i*) = *sets M1* ⋀*i*. *finite-measure* (*Mi1 i*) ⋀*A*. *M1 A* = (∑ *i*. *Mi1 i A*)
and *Mi2*:⋀*i*. *sets* (*Mi2 i*) = *sets M2* ⋀*i*. *finite-measure* (*Mi2 i*) ⋀*A*. *M2 A* = (∑ *i*. *Mi2 i A*)
by *metis*
fix *A*
assume *A*[*measurable*]:*A* ∈ *sets* (*M1* $\bigotimes_M$ *M2*)
have *emeasure* (*M1* $\bigotimes_M$ *M2*) *A* = *emeasure* (*M2* $\bigotimes_M$ *M1*) ((λ(*x*, *y*). (*y*, *x*)) −‘ *A* ∩ *space* (*M2* $\bigotimes_M$ *M1*))
proof −
{
fix *i j*
interpret *pair-sigma-finite Mi1 i Mi2 j*
by(*auto simp*: *pair-sigma-finite-def Mi1*(*2*) *Mi2*(*2*) *finite-measure.sigma-finite-measure*)
have *emeasure* (*Mi1 i* $\bigotimes_M$ *Mi2 j*) *A* = *emeasure* (*Mi2 j* $\bigotimes_M$ *Mi1 i*) ((λ(*x*, *y*). (*y*, *x*)) −‘ *A* ∩ *space* (*M2* $\bigotimes_M$ *M1*))
using *Mi1*(*1*) *Mi2*(*1*) **by**(*simp add*: *arg-cong*[*OF distr-pair-swap*,*of emeasure*] *emeasure-distr sets-eq-imp-space-eq*[*OF sets-pair-measure-cong*[*OF Mi2*(*1*) *Mi1*(*1*)]])
}
thus *?thesis*
by(*auto simp*: *pair-measure-s-finite-measure-suminf'*[*OF Mi2 Mi1*] *pair-measure-s-finite-measure-suminf*[*Mi1 Mi2*] *intro*!: *suminf-cong*)
qed
}
thus *?thesis*
by(*auto intro*!: *measure-eqI simp*: *emeasure-distr*)
qed

proposition *nn-integral-snd'*:
assumes *s-finite-measure M1 s-finite-measure M2*
and *f*[*measurable*]: *f* ∈ *borel-measurable* (*M1* $\bigotimes_M$ *M2*)
shows (∫⁺ *y*. (∫⁺ *x*. *f* (*x*, *y*) ∂*M1*) ∂*M2*) = *integral*^{N} (*M1* $\bigotimes_M$ *M2*) *f*
proof −
interpret *M1*: *s-finite-measure M1* **by** *fact*
interpret *M2*: *s-finite-measure M2* **by** *fact*
note *measurable-pair-swap*[*OF f*]
from *M1.nn-integral-fst'*[*OF this*]
have (∫⁺ *y*. (∫⁺ *x*. *f* (*x*, *y*) ∂*M1*) ∂*M2*) = (∫⁺ (*x*, *y*). *f* (*y*, *x*) ∂(*M2* $\bigotimes_M$

M1))
  by *simp*
 also have ($\int^+$ (*x*, *y*). *f* (*y*, *x*) ∂(*M2* $\bigotimes_M$ *M1*)) = *integral*^{N} (*M1* $\bigotimes_M$ *M2*) *f*
  by (*subst distr-pair-swap-s-finite*[*OF assms*(*1*,*2*)]) (*auto simp add*: *nn-integral-distr intro*!: *nn-integral-cong*)
 finally show *?thesis* **.**
qed

lemma (**in** *s-finite-measure*) *borel-measurable-lebesgue-integrable'*[*measurable* (*raw*)]:
 fixes *f* :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
 assumes [*measurable*]: *case-prod f* ∈ *borel-measurable* (*N* $\bigotimes_M$ *M*)
 shows *Measurable.pred N* (λ*x*. *integrable M* (*f x*))
proof −
 have [*simp*]: ⋀*x*. *x* ∈ *space N* ⟹ *integrable M* (*f x*) ⟷ ($\int^+ y$. *norm* (*f x y*) ∂*M*) < ∞
  unfolding *integrable-iff-bounded* **by** *simp*
 show *?thesis*
  by (*simp cong*: *measurable-cong*)
qed

lemma (**in** *s-finite-measure*) *measurable-measure'*[*measurable* (*raw*)]:
 (⋀*x*. *x* ∈ *space N* ⟹ *A x* ⊆ *space M*) ⟹
  {*x* ∈ *space* (*N* $\bigotimes_M$ *M*). *snd x* ∈ *A* (*fst x*)} ∈ *sets* (*N* $\bigotimes_M$ *M*) ⟹
  (λ*x*. *measure M* (*A x*)) ∈ *borel-measurable N*
 unfolding *measure-def* **by** (*intro measurable-emeasure' borel-measurable-enn2real*) *auto*

proposition (**in** *s-finite-measure*) *borel-measurable-lebesgue-integral'*[*measurable* (*raw*)]:
 fixes *f* :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
 assumes *f*[*measurable*]: *case-prod f* ∈ *borel-measurable* (*N* $\bigotimes_M$ *M*)
 shows (λ*x*. ∫ *y*. *f x y* ∂*M*) ∈ *borel-measurable N*
proof −
 from *borel-measurable-implies-sequence-metric*[*OF f*, *of 0*]
 obtain *s* **where** *s*: ⋀*i*. *simple-function* (*N* $\bigotimes_M$ *M*) (*s i*)
  and ∀ *x*∈*space* (*N* $\bigotimes_M$ *M*). (λ*i*. *s i x*) ⟶ (*case x of* (*x*, *y*) ⇒ *f x y*)
  and ∀ *i*. ∀ *x*∈*space* (*N* $\bigotimes_M$ *M*). *dist* (*s i x*) *0* ≤ *2* ∗ *dist* (*case x of* (*x*, *xa*) ⇒ *f x xa*) *0*
  by *auto*
 then have ∗:
  ⋀*x y*. *x* ∈ *space N* ⟹ *y* ∈ *space M* ⟹ (λ*i*. *s i* (*x*, *y*)) ⟶ *f x y*
  ⋀*i x y*. *x* ∈ *space N* ⟹ *y* ∈ *space M* ⟹ *norm* (*s i* (*x*, *y*)) ≤ *2* ∗ *norm* (*f x y*)
  by (*auto simp*: *space-pair-measure*)

 have [*measurable*]: ⋀*i*. *s i* ∈ *borel-measurable* (*N* $\bigotimes_M$ *M*)
  by (*rule borel-measurable-simple-function*) *fact*

 have ⋀*i*. *s i* ∈ *measurable* (*N* $\bigotimes_M$ *M*) (*count-space UNIV*)
  by (*rule measurable-simple-function*) *fact*

```
  define f' where [abs-def]: f' i x =
    (if integrable M (f x) then Bochner-Integration.simple-bochner-integral M (λy.
s i (x, y)) else 0) for i x

  { fix i x assume x ∈ space N
    then have Bochner-Integration.simple-bochner-integral M (λy. s i (x, y)) =
      (∑ z∈s i ‘ (space N × space M). measure M {y ∈ space M. s i (x, y) = z}
*R z)
      using s[THEN simple-functionD(1)]
      unfolding simple-bochner-integral-def
      by (intro sum.mono-neutral-cong-left)
        (auto simp: eq-commute space-pair-measure image-iff cong: conj-cong) }
  note eq = this

  show ?thesis
  proof (rule borel-measurable-LIMSEQ-metric)
    fix i show f' i ∈ borel-measurable N
      unfolding f'-def by (simp-all add: eq cong: measurable-cong if-cong)
  next
    fix x assume x: x ∈ space N
    { assume int-f: integrable M (f x)
      have int-2f: integrable M (λy. 2 * norm (f x y))
        by (intro integrable-norm integrable-mult-right int-f)
      have (λi. integral^L M (λy. s i (x, y))) ⟶ integral^L M (f x)
      proof (rule integral-dominated-convergence)
        from int-f show f x ∈ borel-measurable M by auto
        show ⋀i. (λy. s i (x, y)) ∈ borel-measurable M
          using x by simp
        show AE xa in M. (λi. s i (x, xa)) ⟶ f x xa
          using x * by auto
        show ⋀i. AE xa in M. norm (s i (x, xa)) ≤ 2 * norm (f x xa)
          using x * by auto
      qed fact
      moreover
      { fix i
        have Bochner-Integration.simple-bochner-integrable M (λy. s i (x, y))
        proof (rule simple-bochner-integrableI-bounded)
          have (λy. s i (x, y)) ‘ space M ⊆ s i ‘ (space N × space M)
            using x by auto
          then show simple-function M (λy. s i (x, y))
            using simple-functionD(1)[OF s(1), of i] x
            by (intro simple-function-borel-measurable)
              (auto simp: space-pair-measure dest: finite-subset)
          have (∫+ y. ennreal (norm (s i (x, y))) ∂M) ≤ (∫+ y. 2 * norm (f x y)
∂M)
            using x * by (intro nn-integral-mono) auto
          also have (∫+ y. 2 * norm (f x y) ∂M) < ∞
            using int-2f unfolding integrable-iff-bounded by simp
          finally show (∫+ xa. ennreal (norm (s i (x, xa))) ∂M) < ∞ .
```

```
      qed
    then have integral^L M (λy. s i (x, y)) = Bochner-Integration.simple-bochner-integral
M (λy. s i (x, y))
      by (rule simple-bochner-integrable-eq-integral[symmetric]) }
    ultimately have (λi. Bochner-Integration.simple-bochner-integral M (λy. s i
(x, y))) ⟶ integral^L M (f x)
      by simp }
  then
  show (λi. f' i x) ⟶ integral^L M (f x)
    unfolding f'-def
    by (cases integrable M (f x)) (simp-all add: not-integrable-integral-eq)
 qed
qed

lemma integrable-product-swap-s-finite:
 fixes f :: - ⇒ -::{banach, second-countable-topology}
 assumes M1:s-finite-measure M1 and M2:s-finite-measure M2
    and integrable (M1 ⨂_M M2) f
  shows integrable (M2 ⨂_M M1) (λ(x,y). f (y,x))
proof −
  have ∗: (λ(x,y). f (y,x)) = (λx. f (case x of (x,y)⇒(y,x))) by (auto simp:
fun-eq-iff)
 show ?thesis unfolding ∗
  by (rule integrable-distr[OF measurable-pair-swap'])
    (simp add: distr-pair-swap-s-finite[OF M1 M2,symmetric] assms)
qed

lemma integrable-product-swap-iff-s-finite:
 fixes f :: - ⇒ -::{banach, second-countable-topology}
 assumes M1:s-finite-measure M1 and M2:s-finite-measure M2
 shows integrable (M2 ⨂_M M1) (λ(x,y). f (y,x)) ⟷ integrable (M1 ⨂_M M2)
f
proof −
 from integrable-product-swap-s-finite[OF M2 M1,of λ(x,y). f (y,x)] integrable-product-swap-s-finite[OF
M1 M2,of f]
 show ?thesis by auto
qed

lemma integral-product-swap-s-finite:
 fixes f :: - ⇒ -::{banach, second-countable-topology}
 assumes M1:s-finite-measure M1 and M2:s-finite-measure M2
    and f: f ∈ borel-measurable (M1 ⨂_M M2)
 shows (∫ (x,y). f (y,x) ∂(M2 ⨂_M M1)) = integral^L (M1 ⨂_M M2) f
proof −
  have ∗: (λ(x,y). f (y,x)) = (λx. f (case x of (x,y)⇒(y,x))) by (auto simp:
fun-eq-iff)
 show ?thesis unfolding ∗
  by (simp add: integral-distr[symmetric, OF measurable-pair-swap' f] distr-pair-swap-s-finite[OF
M1 M2,symmetric])
```

qed

theorem(**in** *s-finite-measure*) *Fubini-integrable'*:
fixes f :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: $f \in$ *borel-measurable* ($M1 \bigotimes_M M$)
and *integ1*: *integrable M1* ($\lambda x.$ $\int$ $y.$ *norm* (f (x, y)) ∂M)
and *integ2*: *AE x in M1*. *integrable M* ($\lambda y.$ f (x, y))
shows *integrable* ($M1 \bigotimes_M M$) f
proof (*rule integrableI-bounded*)
have ($\int^+$ $p.$ *norm* (f p) ∂($M1 \bigotimes_M M$)) = ($\int^+$ $x.$ ($\int^+$ $y.$ *norm* (f (x, y)) ∂M) $\partial M1$)
by (*simp add*: *nn-integral-fst'*[*symmetric*])
also have ... = ($\int^+$ $x.$ |$\int$ $y.$ *norm* (f (x, y)) ∂M| $\partial M1$)
proof(*rule nn-integral-cong-AE*)
show *AE x in M1*. ($\int^+$ $y.$ *ennreal* (*norm* (f (x, y))) ∂M) = *ennreal* |*LINT* y|M. *norm* (f (x, y))|
using *integ2*
proof *eventually-elim*
fix x **assume** *integrable M* ($\lambda y.$ f (x, y))
then have f: *integrable M* ($\lambda y.$ *norm* (f (x, y)))
by *simp*
then have ($\int^+ y.$ *ennreal* (*norm* (f (x, y))) ∂M) = *ennreal* (*LINT* y|M. *norm* (f (x, y)))
by (*rule nn-integral-eq-integral*) *simp*
also have ... = *ennreal* |*LINT* y|M. *norm* (f (x, y))|
using f **by** *simp*
finally show ($\int^+ y.$ *ennreal* (*norm* (f (x, y))) ∂M) = *ennreal* |*LINT* y|M. *norm* (f (x, y))| **.**
qed
qed
also have ... < ∞
using *integ1* **by** (*simp add*: *integrable-iff-bounded integral-nonneg-AE*)
finally show ($\int^+$ $p.$ *norm* (f p) ∂($M1 \bigotimes_M M$)) < ∞ **.**
qed *fact*

lemma(**in** *s-finite-measure*) *emeasure-pair-measure-finite'*:
assumes A: $A \in$ *sets* ($M1 \bigotimes_M M$) **and** *finite*: *emeasure* ($M1 \bigotimes_M M$) $A < \infty$
shows *AE x in M1*. *emeasure M* {y∈*space M*. (x, y) ∈ A} < ∞
proof −
from *emeasure-pair-measure-alt'*[*OF A*] *finite*
have ($\int^+$ $x.$ *emeasure M* (*Pair x* −‘ A) $\partial M1$) ≠ ∞
by *simp*
then have *AE x in M1*. *emeasure M* (*Pair x* −‘ A) ≠ ∞
by (*rule nn-integral-PInf-AE*[*rotated*]) (*intro measurable-emeasure-Pair'* A)
moreover have $\bigwedge x.$ $x \in$ *space M1* ⟹ *Pair x* −‘ A = {y∈*space M*. (x, y) ∈ A}
using *sets.sets-into-space*[*OF A*] **by** (*auto simp*: *space-pair-measure*)
ultimately show *?thesis* **by** (*auto simp*: *less-top*)
qed

lemma(**in** *s-finite-measure*) *AE-integrable-fst'''*:
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *integrable* (*M1* ⨂$_M$ *M*) *f*
shows *AE x in M1*. *integrable M* (λ*y*. *f* (*x*, *y*))
proof −
have (∫$^+$*x*. (∫$^+$*y*. *norm* (*f* (*x*, *y*)) ∂*M*) ∂*M1*) = (∫$^+$*x*. *norm* (*f x*) ∂(*M1* ⨂$_M$ *M*))
by (*rule nn-integral-fst'*) *simp*
also have (∫$^+$*x*. *norm* (*f x*) ∂(*M1* ⨂$_M$ *M*)) ≠ ∞
using *f* **unfolding** *integrable-iff-bounded* **by** *simp*
finally have *AE x in M1*. (∫$^+$*y*. *norm* (*f* (*x*, *y*)) ∂*M*) ≠ ∞
by (*intro nn-integral-PInf-AE borel-measurable-nn-integral'*)
(*auto simp*: *measurable-split-conv*)
with *AE-space* **show** *?thesis*
by *eventually-elim*
(*auto simp*: *integrable-iff-bounded measurable-compose*[*OF* - *borel-measurable-integrable*[*OF f*]] *less-top*)
qed

lemma(**in** *s-finite-measure*) *integrable-fst-norm'*:
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *integrable* (*M1* ⨂$_M$ *M*) *f*
shows *integrable M1* (λ*x*. ∫ *y*. *norm* (*f* (*x*, *y*)) ∂*M*)
unfolding *integrable-iff-bounded*
proof
show (λ*x*. ∫ *y*. *norm* (*f* (*x*, *y*)) ∂*M*) ∈ *borel-measurable M1*
by (*rule borel-measurable-lebesgue-integral'*) *simp*
have (∫$^+$ *x*. *ennreal* (*norm* (∫ *y*. *norm* (*f* (*x*, *y*)) ∂*M*)) ∂*M1*) = (∫$^+$*x*. (∫$^+$*y*. *norm* (*f* (*x*, *y*)) ∂*M*) ∂*M1*)
using *AE-integrable-fst'''*[*OF f*] **by** (*auto intro*!: *nn-integral-cong-AE simp*: *nn-integral-eq-integral*)
also have (∫$^+$*x*. (∫$^+$*y*. *norm* (*f* (*x*, *y*)) ∂*M*) ∂*M1*) = (∫$^+$*x*. *norm* (*f x*) ∂(*M1* ⨂$_M$ *M*))
by (*rule nn-integral-fst'*) *simp*
also have (∫$^+$*x*. *norm* (*f x*) ∂(*M1* ⨂$_M$ *M*)) < ∞
using *f* **unfolding** *integrable-iff-bounded* **by** *simp*
finally show (∫$^+$ *x*. *ennreal* (*norm* (∫ *y*. *norm* (*f* (*x*, *y*)) ∂*M*)) ∂*M1*) < ∞ .
qed

lemma(**in** *s-finite-measure*) *integrable-fst'''*:
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *integrable* (*M1* ⨂$_M$ *M*) *f*
shows *integrable M1* (λ*x*. ∫ *y*. *f* (*x*, *y*) ∂*M*)
by(*auto intro*!: *Bochner-Integration.integrable-bound*[*OF integrable-fst-norm'*[*OF f*]])

proposition(**in** *s-finite-measure*) *integral-fst'''*:
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*: *integrable* (*M1* ⨂$_M$ *M*) *f*

shows $(\int x.\ (\int y.\ f\ (x,\ y)\ \partial M)\ \partial M1) = integral^L\ (M1 \bigotimes_M M)\ f$
using f **proof** *induct*
case (*base A c*)
have *A*[*measurable*]: $A \in sets\ (M1 \bigotimes_M M)$ **by** *fact*

have *eq*: $\bigwedge x\ y.\ x \in space\ M1 \Longrightarrow indicator\ A\ (x,\ y) = indicator\ \{y \in space\ M.\ (x,\ y) \in A\}\ y$
using *sets.sets-into-space*[*OF A*] **by** (*auto split*: *split-indicator simp*: *space-pair-measure*)

have *int-A*: *integrable* $(M1 \bigotimes_M M)$ (*indicator A* :: - $\Rightarrow$ *real*)
using *base* **by** (*rule integrable-real-indicator*)
have $(\int\ x.\ \int\ y.\ indicator\ A\ (x,\ y) *_R c\ \partial M\ \partial M1) = (\int x.\ measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\} *_R c\ \partial M1)$
proof (*intro integral-cong-AE*)
from *AE-integrable-fst'''*[*OF int-A*] *AE-space*
show *AE x in M1*. $(\int y.\ indicator\ A\ (x,\ y) *_R c\ \partial M) = measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\} *_R c$
by *eventually-elim* (*simp add*: *eq integrable-indicator-iff*)
qed *simp-all*
also have ... $= measure\ (M1 \bigotimes_M M)\ A *_R c$
proof (*subst integral-scaleR-left*)
have $(\int^+ x.\ ennreal\ (measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\})\ \partial M1) =$
$(\int^+ x.\ emeasure\ M\ \{y \in space\ M.\ (x,\ y) \in A\}\ \partial M1)$
using *emeasure-pair-measure-finite'*[*OF base*]
by (*intro nn-integral-cong-AE*, *eventually-elim*) (*simp add*: *emeasure-eq-ennreal-measure*)
also have ... $= emeasure\ (M1 \bigotimes_M M)\ A$
using *sets.sets-into-space*[*OF A*]
by (*subst emeasure-pair-measure-alt'*)
(*auto intro*!: *nn-integral-cong arg-cong*[**where** *f*=*emeasure M*] *simp*: *space-pair-measure*)
finally have $*$: $(\int^+ x.\ ennreal\ (measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\})\ \partial M1) = emeasure\ (M1 \bigotimes_M M)\ A$ **.**

from *base* $*$ **show** *integrable M1* $(\lambda x.\ measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\})$
by (*simp add*: *integrable-iff-bounded*)
then have $(\int x.\ measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\}\ \partial M1) =$
$(\int^+ x.\ ennreal\ (measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\})\ \partial M1)$
by (*rule nn-integral-eq-integral*[*symmetric*]) *simp*
also note $*$
finally show $(\int x.\ measure\ M\ \{y \in space\ M.\ (x,\ y) \in A\}\ \partial M1) *_R c = measure\ (M1 \bigotimes_M M)\ A *_R c$
using *base* **by** (*simp add*: *emeasure-eq-ennreal-measure*)
qed
also have ... $= (\int\ a.\ indicator\ A\ a *_R c\ \partial(M1 \bigotimes_M M))$
using *base* **by** *simp*
finally show *?case* **.**
next
case (*add f g*)
then have [*measurable*]: $f \in borel\text{-}measurable\ (M1 \bigotimes_M M)$ $g \in borel\text{-}measurable$

```
(M1 ⨂M M)
    by auto
  have (∫ x. ∫ y. f (x, y) + g (x, y) ∂M ∂M1) =
    (∫ x. (∫ y. f (x, y) ∂M) + (∫ y. g (x, y) ∂M) ∂M1)
    apply (rule integral-cong-AE)
    apply simp-all
    using AE-integrable-fst'''[OF add(1)] AE-integrable-fst'''[OF add(3)]
    apply eventually-elim
    apply simp
    done
  also have ... = (∫ x. f x ∂(M1 ⨂M M)) + (∫ x. g x ∂(M1 ⨂M M))
   using integrable-fst'''[OF add(1)] integrable-fst'''[OF add(3)] add(2,4) by simp
  finally show ?case
    using add by simp
next
  case (lim f s)
 then have [measurable]: f ∈ borel-measurable (M1 ⨂M M) ⋀i. s i ∈ borel-measurable
(M1 ⨂M M)
    by auto

  show ?case
  proof (rule LIMSEQ-unique)
    show (λi. integral^L (M1 ⨂M M) (s i)) ⟶ integral^L (M1 ⨂M M) f
    proof (rule integral-dominated-convergence)
      show integrable (M1 ⨂M M) (λx. 2 * norm (f x))
        using lim(5) by auto
    qed (insert lim, auto)
    have (λi. ∫ x. ∫ y. s i (x, y) ∂M ∂M1) ⟶ ∫ x. ∫ y. f (x, y) ∂M ∂M1
    proof (rule integral-dominated-convergence)
      have AE x in M1. ∀i. integrable M (λy. s i (x, y))
        unfolding AE-all-countable using AE-integrable-fst'''[OF lim(1)] ..
      with AE-space AE-integrable-fst'''[OF lim(5)]
      show AE x in M1. (λi. ∫ y. s i (x, y) ∂M) ⟶ ∫ y. f (x, y) ∂M
      proof eventually-elim
        fix x assume x: x ∈ space M1 and
          s: ∀i. integrable M (λy. s i (x, y)) and f: integrable M (λy. f (x, y))
        show (λi. ∫ y. s i (x, y) ∂M) ⟶ ∫ y. f (x, y) ∂M
        proof (rule integral-dominated-convergence)
          show integrable M (λy. 2 * norm (f (x, y)))
            using f by auto
          show AE xa in M. (λi. s i (x, xa)) ⟶ f (x, xa)
            using x lim(3) by (auto simp: space-pair-measure)
          show ⋀i. AE xa in M. norm (s i (x, xa)) ≤ 2 * norm (f (x, xa))
            using x lim(4) by (auto simp: space-pair-measure)
        qed (insert x, measurable)
      qed
      show integrable M1 (λx. (∫ y. 2 * norm (f (x, y)) ∂M))
        by (intro integrable-mult-right integrable-norm integrable-fst''' lim)
      fix i show AE x in M1. norm (∫ y. s i (x, y) ∂M) ≤ (∫ y. 2 * norm (f (x,
```

$y))$ ∂M)
using *AE-space AE-integrable-fst'''*[*OF lim*(*1*), *of i*] *AE-integrable-fst'''*[*OF lim*(*5*)]
proof *eventually-elim*
fix x **assume** x: $x \in$ *space M1*
and s: *integrable M* ($\lambda y.\ s\ i\ (x,\ y)$) **and** f: *integrable M* ($\lambda y.\ f\ (x,\ y)$)
from s **have** *norm* ($\int\ y.\ s\ i\ (x,\ y)\ \partial M$) $\leq$ ($\int^+ y.$ *norm* ($s\ i\ (x,\ y)$) ∂M)
by (*rule integral-norm-bound-ennreal*)
also have $\ldots \leq$ ($\int^+ y.\ 2 * norm\ (f\ (x,\ y))\ \partial M$)
using *x lim* **by** (*auto intro*!: *nn-integral-mono simp*: *space-pair-measure*)
also have $\ldots =$ ($\int y.\ 2 * norm\ (f\ (x,\ y))\ \partial M$)
using f **by** (*intro nn-integral-eq-integral*) *auto*
finally show *norm* ($\int\ y.\ s\ i\ (x,\ y)\ \partial M$) $\leq$ ($\int\ y.\ 2 * norm\ (f\ (x,\ y))\ \partial M$)
by *simp*
qed
qed *simp-all*
then show ($\lambda i.\ integral^L\ (M1 \bigotimes_M M)\ (s\ i)$) $\longrightarrow \int\ x.\ \int\ y.\ f\ (x,\ y)\ \partial M\ \partial M1$
using *lim* **by** *simp*
qed
qed

lemma (**in** *s-finite-measure*)
fixes f :: - $\Rightarrow$ - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes f: *integrable* (*M1* $\bigotimes_M$ *M*) (*case-prod f*)
shows *AE-integrable-fst''*: *AE x in M1*. *integrable M* ($\lambda y.\ f\ x\ y$)
and *integrable-fst''*: *integrable M1* ($\lambda x.\ \int y.\ f\ x\ y\ \partial M$)
and *integrable-fst-norm*: *integrable M1* ($\lambda x.\ \int y.\ norm\ (f\ x\ y)\ \partial M$)
and *integral-fst''*: ($\int x.\ (\int y.\ f\ x\ y\ \partial M)\ \partial M1$) = $integral^L$ (*M1* $\bigotimes_M$ *M*) ($\lambda(x,\ y).\ f\ x\ y$)
using *AE-integrable-fst'''*[*OF f*] *integrable-fst'''*[*OF f*] *integral-fst'''*[*OF f*] *integrable-fst-norm'*[*OF f*] **by** *auto*

lemma
fixes f :: - $\Rightarrow$ - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes *M1*:*s-finite-measure M1* **and** *M2*:*s-finite-measure M2*
and *f*[*measurable*]: *integrable* (*M1* $\bigotimes_M$ *M2*) (*case-prod f*)
shows *AE-integrable-snd-s-finite*: *AE y in M2*. *integrable M1* ($\lambda x.\ f\ x\ y$) (**is** *?AE*)
and *integrable-snd-s-finite*: *integrable M2* ($\lambda y.\ \int x.\ f\ x\ y\ \partial M1$) (**is** *?INT*)
and *integrable-snd-norm-s-finite*: *integrable M2* ($\lambda y.\ \int x.\ norm\ (f\ x\ y)\ \partial M1$) (**is** *?INT2*)
and *integral-snd-s-finite*: ($\int y.\ (\int x.\ f\ x\ y\ \partial M1)\ \partial M2$) = $integral^L$ (*M1* $\bigotimes_M$ *M2*) (*case-prod f*) (**is** *?EQ*)
proof −
interpret *Q*: *s-finite-measure M1* **by** *fact*
have *Q-int*: *integrable* (*M2* $\bigotimes_M$ *M1*) ($\lambda(x,\ y).\ f\ y\ x$)
using *f* **unfolding** *integrable-product-swap-iff-s-finite*[*OF M1 M2*,*symmetric*] **by** *simp*
show *?AE* **using** *Q.AE-integrable-fst'''*[*OF Q-int*] **by** *simp*

show *?INT* **using** *Q.integrable-fst'''*[*OF Q-int*] **by** *simp*
show *?INT2* **using** *Q.integrable-fst-norm*[*OF Q-int*] **by** *simp*
show *?EQ* **using** *Q.integral-fst'''*[*OF Q-int*]
using *integral-product-swap-s-finite*[*OF M1 M2*,*of case-prod f*] **by** *simp*
qed

proposition *Fubini-integral'*:
fixes *f* :: - ⇒ - ⇒ - :: {*banach*, *second-countable-topology*}
assumes *M1*:*s-finite-measure M1* **and** *M2*:*s-finite-measure M2*
and *f*: *integrable* (*M1* ⨂$_M$ *M2*) (*case-prod f*)
shows (∫ *y*. (∫ *x*. *f x y* ∂*M1*) ∂*M2*) = (∫ *x*. (∫ *y*. *f x y* ∂*M2*) ∂*M1*)
unfolding *integral-snd-s-finite*[*OF assms*] *s-finite-measure.integral-fst'*'[*OF assms*(*2*,*3*)]
..

locale *product-s-finite* =
fixes *M* :: *'i* ⇒ *'a measure*
assumes *s-finite-measures*: ⋀*i*. *s-finite-measure* (*M i*)

sublocale *product-s-finite* ⊆ *M?*: *s-finite-measure M i* **for** *i*
by (*rule s-finite-measures*)

locale *finite-product-s-finite* = *product-s-finite M* **for** *M* :: *'i* ⇒ *'a measure* +
fixes *I* :: *'i set*
assumes *finite-index*: *finite I*

lemma (**in** *product-s-finite*) *emeasure-PiM*:
finite I ⟹ (⋀*i*. *i*∈*I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *emeasure* (*PiM I M*) (*Pi*$_E$ *I A*)
= (∏ *i*∈*I*. *emeasure* (*M i*) (*A i*))
proof (*induct I arbitrary*: *A rule*: *finite-induct*)
case (*insert i I*)
interpret *finite-product-s-finite M I* **by** *standard fact*
have *finite* (*insert i I*) **using** ‹*finite I*› **by** *auto*
interpret *I'*: *finite-product-s-finite M insert i I* **by** *standard fact*
let *?h* = (λ(*f*, *y*). *f*(*i* := *y*))

let *?P* = *distr* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) (*Pi*$_M$ (*insert i I*) *M*) *?h*
let *?μ* = *emeasure ?P*
let *?I* = {*j* ∈ *insert i I*. *emeasure* (*M j*) (*space* (*M j*)) ≠ *1*}
let *?f* = λ*J E j*. *if j* ∈ *J then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*))

have *emeasure* (*Pi*$_M$ (*insert i I*) *M*) (*prod-emb* (*insert i I*) *M* (*insert i I*) (*Pi*$_E$ (*insert i I*) *A*)) =
(∏ *i*∈*insert i I*. *emeasure* (*M i*) (*A i*))
proof (*subst emeasure-extend-measure-Pair*[*OF PiM-def*])
fix *J E* **assume** (*J* ≠ {} ∨ *insert i I* = {}) ∧ *finite J* ∧ *J* ⊆ *insert i I* ∧ *E* ∈ (Π *j*∈*J*. *sets* (*M j*))
then have *J*: *J* ≠ {} *finite J J* ⊆ *insert i I* **and** *E*: ∀ *j*∈*J*. *E j* ∈ *sets* (*M j*)
by *auto*

let *?p* = *prod-emb* (*insert i I*) *M J* (*Pi*$_E$ *J E*)
let *?p′* = *prod-emb I M* (*J* − {*i*}) (Π$_E$ *j*∈*J*−{*i*}. *E j*)
have *?μ ?p* =
emeasure (*Pi*$_M$ *I M* ⨂$_M$ (*M i*)) (*?h* −‘ *?p* ∩ *space* (*Pi*$_M$ *I M* ⨂$_M$ *M i*))
by (*intro emeasure-distr measurable-add-dim sets-PiM-I*) *fact*+
also have *?h* −‘ *?p* ∩ *space* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) = *?p′* × (*if i* ∈ *J then E i else space* (*M i*))
using *J E*[*rule-format*, *THEN sets.sets-into-space*]
by (*force simp*: *space-pair-measure space-PiM prod-emb-iff PiE-def Pi-iff split*: *if-split-asm*)
also have *emeasure* (*Pi*$_M$ *I M* ⨂$_M$ (*M i*)) (*?p′* × (*if i* ∈ *J then E i else space* (*M i*))) =
emeasure (*Pi*$_M$ *I M*) *?p′* ∗ *emeasure* (*M i*) (*if i* ∈ *J then* (*E i*) *else space* (*M i*))
using *J E* **by** (*intro M.emeasure-pair-measure-Times′ sets-PiM-I*) *auto*
also have *?p′* = (Π$_E$ *j*∈*I*. *if j* ∈ *J*−{*i*} *then E j else space* (*M j*))
using *J E*[*rule-format*, *THEN sets.sets-into-space*]
by (*auto simp*: *prod-emb-iff PiE-def Pi-iff split*: *if-split-asm*) *blast*+
also have *emeasure* (*Pi*$_M$ *I M*) (Π$_E$ *j*∈*I*. *if j* ∈ *J*−{*i*} *then E j else space* (*M j*)) =
(∏ *j*∈*I*. *if j* ∈ *J*−{*i*} *then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*)))
using *E* **by** (*subst insert*) (*auto intro*!: *prod.cong*)
also have (∏*j*∈*I*. *if j* ∈ *J* − {*i*} *then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*))) ∗
emeasure (*M i*) (*if i* ∈ *J then E i else space* (*M i*)) = (∏*j*∈*insert i I*. *?f J E j*)
using *insert* **by** (*auto simp*: *mult.commute intro*!: *arg-cong2*[**where** *f*=(∗)] *prod.cong*)
also have ... = (∏*j*∈*J* ∪ *?I*. *?f J E j*)
using *insert*(*1*,*2*) *J E* **by** (*intro prod.mono-neutral-right*) *auto*
finally show *?μ ?p* = ... **.**

show *prod-emb* (*insert i I*) *M J* (*Pi*$_E$ *J E*) ∈ *Pow* (Π$_E$ *i*∈*insert i I*. *space* (*M i*))
using *J E*[*rule-format*, *THEN sets.sets-into-space*] **by** (*auto simp*: *prod-emb-iff PiE-def*)
next
show *positive* (*sets* (*Pi*$_M$ (*insert i I*) *M*)) *?μ countably-additive* (*sets* (*Pi*$_M$ (*insert i I*) *M*)) *?μ*
using *emeasure-positive*[*of ?P*] *emeasure-countably-additive*[*of ?P*] **by** *simp-all*
next
show (*insert i I* ≠ {} ∨ *insert i I* = {}) ∧ *finite* (*insert i I*) ∧
insert i I ⊆ *insert i I* ∧ *A* ∈ (Π *j*∈*insert i I*. *sets* (*M j*))
using *insert* **by** *auto*
qed (*auto intro*!: *prod.cong*)
with *insert* **show** *?case*
by (*subst* (*asm*) *prod-emb-PiE-same-index*) (*auto intro*!: *sets.sets-into-space*)
qed *simp*

lemma (**in** *finite-product-s-finite*) *measure-times*:
(⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *emeasure* (*Pi*$_M$ *I M*) (*Pi*$_E$ *I A*) = (∏ *i*∈*I*. *emeasure* (*M i*) (*A i*))
using *emeasure-PiM*[*OF finite-index*] **by** *auto*

lemma (**in** *product-s-finite*) *nn-integral-empty*:
0 ≤ *f* (λ*k*. *undefined*) ⟹ *integral*^{N} (*Pi*$_M$ {} *M*) *f* = *f* (λ*k*. *undefined*)
by (*simp add*: *PiM-empty nn-integral-count-space-finite*)

Every s-finite measure is represented as the push-forward measure of a σ-finite measure.

definition *Mi-to-NM* :: (*nat* ⇒ '*a measure*) ⇒ '*a measure* ⇒ (*nat* × '*a*) *measure*
where
Mi-to-NM Mi M ≡ *measure-of* (*space* (*count-space UNIV* ⨂$_M$ *M*)) (*sets* (*count-space UNIV* ⨂$_M$ *M*)) (λ*A*. ∑ *i*. *distr* (*Mi i*) (*count-space UNIV* ⨂$_M$ *M*) (λ*x*. (*i*,*x*)) *A*)

lemma
shows *sets-Mi-to-NM*[*measurable-cong*,*simp*]: *sets* (*Mi-to-NM Mi M*) = *sets* (*count-space UNIV* ⨂$_M$ *M*)
and *space-Mi-to-NM*[*simp*]: *space* (*Mi-to-NM Mi M*) = *space* (*count-space UNIV* ⨂$_M$ *M*)
by(*simp-all add*: *Mi-to-NM-def*)

context
fixes *M* :: '*a measure* **and** *Mi* :: *nat* ⇒ '*a measure*
assumes *sets-Mi*[*measurable-cong*,*simp*]: ⋀*i*. *sets* (*Mi i*) = *sets M*
and *emeasure-Mi*: ⋀*A*. *A* ∈ *sets M* ⟹ *M A* = (∑ *i*. *Mi i A*)
begin

lemma *emeasure-Mi-to-NM*:
assumes [*measurable*]: *A* ∈ *sets* (*count-space UNIV* ⨂$_M$ *M*)
shows *emeasure* (*Mi-to-NM Mi M*) *A* = (∑ *i*. *distr* (*Mi i*) (*count-space UNIV* ⨂$_M$ *M*) (λ*x*. (*i*,*x*)) *A*)
proof(*rule emeasure-measure-of*[**where** Ω=*space* (*count-space UNIV* ⨂$_M$ *M*) **and** *A*=*sets* (*count-space UNIV* ⨂$_M$ *M*)])
show *countably-additive* (*sets* (*Mi-to-NM Mi M*)) (λ*A*. ∑ *i*. *emeasure* (*distr* (*Mi i*) (*count-space UNIV* ⨂$_M$ *M*) (*Pair i*)) *A*)
unfolding *countably-additive-def*
proof *safe*
fix *A* :: *nat* ⇒ (*nat* × -) *set*
assume *range A* ⊆ *sets* (*Mi-to-NM Mi M*) **and** *dA*:*disjoint-family A*
hence [*measurable*]: ⋀*i*. *A i* ∈ *sets* (*count-space UNIV* ⨂$_M$ *M*)
by *auto*
show (∑ *j i*. *emeasure* (*distr* (*Mi i*) (*count-space UNIV* ⨂$_M$ *M*) (*Pair i*)) (*A j*)) = (∑ *i*. *emeasure* (*distr* (*Mi i*) (*count-space UNIV* ⨂$_M$ *M*) (*Pair i*)) (⋃ (*range A*))) (**is** *?lhs* = *?rhs*)

proof −
have *?lhs* = $(\sum i\ j.$ *emeasure* (*distr* (*Mi i*) (*count-space UNIV* $\bigotimes_M$ *M*) (*Pair i*)) (*A j*))
by(*auto simp*: *nn-integral-count-space-nat*[*symmetric*] *pair-sigma-finite-def sigma-finite-measure-count-space intro*!: *pair-sigma-finite.Fubini'*)
also have ... = *?rhs*
proof(*rule suminf-cong*)
fix *n*
have [*simp*]:*Pair n* −‘ $\bigcup$ (*range A*) = ($\bigcup$ (*range* ($\lambda j.$ *Pair n* −‘ *A j*)))
by *auto*
show ($\sum j.$ *emeasure* (*distr* (*Mi n*) (*count-space UNIV* $\bigotimes_M$ *M*) (*Pair n*)) (*A j*)) = *emeasure* (*distr* (*Mi n*) (*count-space UNIV* $\bigotimes_M$ *M*) (*Pair n*)) ($\bigcup$ (*range A*))
using *dA* **by**(*fastforce intro*!: *suminf-emeasure simp*: *disjoint-family-on-def emeasure-distr*)
qed
finally show *?thesis* **.**
qed
qed
qed(*auto simp*: *positive-def sets.space-closed Mi-to-NM-def*)

lemma *sigma-finite-Mi-to-NM-measure*:
assumes $\bigwedge i.$ *finite-measure* (*Mi i*)
shows *sigma-finite-measure* (*Mi-to-NM Mi M*)
proof −
{
fix *n*
assume *emeasure* (*Mi-to-NM Mi M*) ({*n*} × *space M*) = ⊤
moreover have *emeasure* (*Mi-to-NM Mi M*) ({*n*} × *space M*) = *emeasure* (*Mi n*) (*space M*)
by(*simp add*: *emeasure-Mi-to-NM emeasure-distr suminf-offset*[*of* - *Suc n*])
ultimately have *False*
using *finite-measure.finite-emeasure-space*[*OF assms*[*of n*]] **by**(*auto simp*: *sets-eq-imp-space-eq*[*OF sets-Mi*])
}
thus *?thesis*
by(*auto intro*!: *exI*[**where** *x*=$\bigcup i.$ {{*i*} × *space M*}] *simp*: *space-pair-measure sigma-finite-measure-def*)
qed

lemma *distr-Mi-to-NM-M*: *distr* (*Mi-to-NM Mi M*) *M snd* = *M*
proof −
have [*simp*]:*Pair i* −‘ *snd* −‘ *A* ∩ *Pair i* −‘ *space* (*count-space UNIV* $\bigotimes_M$ *M*) = *A* **if** *A* ∈ *sets M* **for** *A* **and** *i* :: *nat*
using *sets.sets-into-space*[*OF that*] **by**(*auto simp*: *space-pair-measure*)
show *?thesis*
by(*auto intro*!: *measure-eqI simp*: *emeasure-distr emeasure-Mi-to-NM emeasure-Mi*)

qed

end

context
fixes μ :: *'a measure*
assumes *standard-borel-ne*: *standard-borel-ne* μ
and *s-finite*: *s-finite-measure* μ
begin

interpretation μ : *s-finite-measure* μ **by** *fact*

interpretation *n-*μ: *standard-borel-ne count-space* (*UNIV* :: *nat set*) $\bigotimes_M$ μ
by (*simp add*: *pair-standard-borel-ne standard-borel-ne*)

lemma *exists-push-forward*:
$\exists(\mu'::$ *real measure*$)\ f.\ f \in$ *borel* $\to_M$ $\mu \wedge$ *sets* $\mu' =$ *sets borel* $\wedge$ *sigma-finite-measure* μ'
$\wedge$ *distr* μ' μ $f = \mu$
proof −
obtain μi **where** μi: $\bigwedge i.$ *sets* (μi *i*) = *sets* μ $\bigwedge i.$ *finite-measure* (μi *i*) $\bigwedge A.$ μ *A* = ($\sum i.$ μi *i A*)
by(*metis* μ*.finite-measures'*)
show *?thesis*
proof(*safe intro*!: *exI*[**where** *x*=*distr* (*Mi-to-NM* μi μ) *borel n-*μ*.to-real*] *exI*[**where** *x*=*snd* ∘ *n-*μ*.from-real*])
have [*simp*]:*distr* (*distr* (*Mi-to-NM* μi μ) *borel n-*μ*.to-real*) (*count-space UNIV* $\bigotimes_M$ μ) *n-*μ*.from-real* = *Mi-to-NM* μi μ
by(*auto simp*: *distr-distr comp-def intro*!:*distr-id'*)
show *sigma-finite-measure* (*distr* (*Mi-to-NM* μi μ) *borel n-*μ*.to-real*)
by(*rule sigma-finite-measure-distr*[**where** *N*=*count-space UNIV* $\bigotimes_M$ μ **and** *f*=*n-*μ*.from-real*]) (*auto intro*!: *sigma-finite-Mi-to-NM-measure* μi)
next
have [*simp*]: *distr* (*Mi-to-NM* μi μ) μ (*snd* ∘ *n-*μ*.from-real* ∘ *n-*μ*.to-real*) = *distr* (*Mi-to-NM* μi μ) μ *snd*
by(*auto intro*!: *distr-cong*[*OF refl*])
show *distr* (*distr* (*Mi-to-NM* μi μ) *borel n-*μ*.to-real*) μ (*snd* ∘ *n-*μ*.from-real*) = μ
by(*auto simp*: *distr-distr distr-Mi-to-NM-M*[*OF* μi(*1*,*3*)])
qed *auto*
qed

abbreviation μ'*-and-f* ≡ (*SOME* (μ'::*real measure*,*f*). *f* ∈ *borel* $\to_M$ $\mu \wedge$ *sets* μ' = *sets borel* $\wedge$ *sigma-finite-measure* $\mu' \wedge$ *distr* μ' μ *f* = μ)

definition *sigma-pair-*μ ≡ *fst* μ'*-and-f*
definition *sigma-pair-f* ≡ *snd* μ'*-and-f*

lemma

shows *sigma-pair-f-measurable* : *sigma-pair-f* ∈ *borel* →$_M$ *μ* (**is** *?g1*)
and *sets-sigma-pair-μ*: *sets sigma-pair-μ* = *sets borel* (**is** *?g2*)
and *sigma-finite-sigma-pair-μ*: *sigma-finite-measure sigma-pair-μ* (**is** *?g3*)
and *distr-sigma-pair*: *distr sigma-pair-μ μ sigma-pair-f* = *μ* (**is** *?g4*)
proof −
have *case μ′-and-f of* (*μ′*,*f*) ⇒ *f* ∈ *borel* →$_M$ *μ* ∧ *sets μ′* = *sets borel* ∧ *sigma-finite-measure μ′* ∧ *distr μ′ μ f* = *μ*
by(*rule someI-ex*) (*use exists-push-forward* **in** *auto*)
then show *?g1 ?g2 ?g3 ?g4*
by(*auto simp*: *sigma-pair-μ-def sigma-pair-f-def split-beta*)
qed

end

definition *s-finite-measure-algebra* :: *′a measure* ⇒ *′a measure measure* **where**
s-finite-measure-algebra K =
(*SUP A* ∈ *sets K*. *vimage-algebra* {*M*. *s-finite-measure M* ∧ *sets M* = *sets K*} (*λM*. *emeasure M A*) *borel*)

lemma *space-s-finite-measure-algebra*:
space (*s-finite-measure-algebra K*) = {*M*. *s-finite-measure M* ∧ *sets M* = *sets K*}
by (*auto simp add*: *s-finite-measure-algebra-def space-Sup-eq-UN*)

lemma *s-finite-measure-algebra-cong*: *sets M* = *sets N* ⟹ *s-finite-measure-algebra M* = *s-finite-measure-algebra N*
by (*simp add*: *s-finite-measure-algebra-def*)

lemma *measurable-emeasure-s-finite-measure-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (*λM*. *emeasure M a*) ∈ *borel-measurable* (*s-finite-measure-algebra A*)
by (*auto intro*!: *measurable-Sup1 measurable-vimage-algebra1 simp*: *s-finite-measure-algebra-def*)

lemma *measurable-measure-s-finite-measure-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (*λM*. *measure M a*) ∈ *borel-measurable* (*s-finite-measure-algebra A*)
unfolding *measure-def* **by** *measurable*

lemma *s-finite-measure-algebra-measurableD*:
assumes *N*: *N* ∈ *measurable M* (*s-finite-measure-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*
using *measurable-space*[*OF N x*]
by (*auto simp*: *space-s-finite-measure-algebra intro*!: *measurable-cong-sets dest*: *sets-eq-imp-space-eq*)

context
fixes *K M N* **assumes** *K*: *K* ∈ *measurable M* (*s-finite-measure-algebra N*)

begin

lemma *s-finite-measure-algebra-kernel*: $a \in space\ M \Longrightarrow s\text{-}finite\text{-}measure\ (K\ a)$
using *measurable-space*[*OF K*] **by** (*simp add*: *space-s-finite-measure-algebra*)

lemma *s-finite-measure-algebra-sets-kernel*: $a \in space\ M \Longrightarrow sets\ (K\ a) = sets\ N$
using *measurable-space*[*OF K*] **by** (*simp add*: *space-s-finite-measure-algebra*)

lemma *measurable-emeasure-kernel-s-finite-measure-algebra*[*measurable*]:
$A \in sets\ N \Longrightarrow (\lambda a.\ emeasure\ (K\ a)\ A) \in borel\text{-}measurable\ M$
using *measurable-compose*[*OF K measurable-emeasure-s-finite-measure-algebra*] **.**

end

lemma *measurable-s-finite-measure-algebra*:
$(\bigwedge a.\ a \in space\ M \Longrightarrow s\text{-}finite\text{-}measure\ (K\ a)) \Longrightarrow$
$(\bigwedge a.\ a \in space\ M \Longrightarrow sets\ (K\ a) = sets\ N) \Longrightarrow$
$(\bigwedge A.\ A \in sets\ N \Longrightarrow (\lambda a.\ emeasure\ (K\ a)\ A) \in borel\text{-}measurable\ M) \Longrightarrow$
$K \in measurable\ M\ (s\text{-}finite\text{-}measure\text{-}algebra\ N)$
by (*auto intro*!: *measurable-Sup2 measurable-vimage-algebra2 simp*: *s-finite-measure-algebra-def*)

definition *bind-kernel* :: $'a\ measure \Rightarrow ('a \Rightarrow 'b\ measure) \Rightarrow 'b\ measure$ (**infixl** $\gg\!=_k$ *54*) **where**
bind-kernel M k = (*if space M* = {} *then count-space* {} *else*
let Y = *k* (*SOME x. x* ∈ *space M*) *in*
measure-of (*space Y*) (*sets Y*) (λ*B*. ∫⁺*x*. (*k x B*) ∂*M*))

lemma *bind-kernel-cong-All*:
assumes $\bigwedge x.\ x \in space\ M \Longrightarrow f\ x = g\ x$
shows $M \gg\!=_k f = M \gg\!=_k g$
proof(*cases space M* = {})
case *1*:*False*
have (*SOME x. x* ∈ *space M*) ∈ *space M*
by (*rule someI-ex*) (*use 1* **in** *blast*)
with *assms* **have** [*simp*]:*f* (*SOME x. x* ∈ *space M*) = *g* (*SOME x. x* ∈ *space M*)
by *simp*
have (λ*B*. ∫⁺ *x*. *emeasure* (*f x*) *B* ∂*M*) = (λ*B*. ∫⁺ *x*. *emeasure* (*g x*) *B* ∂*M*)
by *standard* (*auto intro*!: *nn-integral-cong simp*: *assms*)
thus *?thesis*
by(*auto simp*: *bind-kernel-def 1*)
qed(*simp add*: *bind-kernel-def*)

lemma *sets-bind-kernel*:
assumes $\bigwedge x.\ x \in space\ M \Longrightarrow sets\ (k\ x) = sets\ N\ space\ M \neq \{\}$
shows $sets\ (M \gg\!=_k k) = sets\ N$
proof −
have *sets* (*k* (*SOME x. x* ∈ *space M*)) = *sets N*
by(*rule someI2-ex*) (*use assms* **in** *auto*)
with *sets-eq-imp-space-eq*[*OF this*] **show** *?thesis*

by(*simp add*: *bind-kernel-def assms*(*2*))
qed

2.2 Measure Kernel

locale *measure-kernel* =
fixes $X :: 'a$ *measure* **and** $Y :: 'b$ *measure* **and** $\kappa :: 'a \Rightarrow 'b$ *measure*
assumes *kernel-sets*[*measurable-cong*]: $\bigwedge x.\ x \in$ *space* $X \Longrightarrow$ *sets* $(\kappa\ x) =$ *sets* Y
and *emeasure-measurable*[*measurable*]: $\bigwedge B.\ B \in$ *sets* $Y \Longrightarrow (\lambda x.$ *emeasure* $(\kappa\ x)\ B) \in$ *borel-measurable* X
and *Y-not-empty*: *space* $X \neq \{\} \Longrightarrow$ *space* $Y \neq \{\}$
begin

lemma *kernel-space* :$\bigwedge x.\ x \in$ *space* $X \Longrightarrow$ *space* $(\kappa\ x) =$ *space* Y
by (*meson kernel-sets sets-eq-imp-space-eq*)

lemma *measure-measurable*:
assumes $B \in$ *sets* Y
shows $(\lambda x.$ *measure* $(\kappa\ x)\ B) \in$ *borel-measurable* X
using *emeasure-measurable*[*OF assms*] **by** (*simp add*: *Sigma-Algebra.measure-def*)

lemma *set-nn-integral-measure*:
assumes [*measurable-cong*]: *sets* $\mu =$ *sets* X **and** [*measurable*]: $A \in$ *sets* $X\ B \in$ *sets* Y
defines $\nu \equiv$ *measure-of* (*space* Y) (*sets* Y) $(\lambda B.\ \int^+ x{\in}A.\ (\kappa\ x\ B)\ \partial\mu)$
shows $\nu\ B = (\int^+ x{\in}A.\ (\kappa\ x\ B)\ \partial\mu)$
proof −
have *nu-sets*[*measurable-cong*]: *sets* $\nu =$ *sets* Y
by(*simp add*: ν-*def*)
have *positive* (*sets* Y) $(\lambda B.\ \int^+ x{\in}A.\ (\kappa\ x\ B)\ \partial\mu)$
by(*simp add*: *positive-def*)
moreover have *countably-additive* (*sets* Y) $(\lambda B.\ \int^+ x{\in}A.\ (\kappa\ x\ B)\ \partial\mu)$
unfolding *countably-additive-def*
proof *safe*
fix $C :: nat \Rightarrow$ -
assume *h*:*range* $C \subseteq$ *sets* Y *disjoint-family* C
thus $(\sum i.\ \int^+ x{\in}A.\ (\kappa\ x)\ (C\ i)\partial\mu) = (\int^+ x{\in}A.\ (\kappa\ x)\ (\bigcup\ (range\ C))\partial\mu)$
by(*auto intro*!: *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF assms*(*1*)] *kernel-sets suminf-emeasure nn-integral-suminf*[*symmetric*])
qed
ultimately show *?thesis*
using ν-*def assms*(*3*) *emeasure-measure-of-sigma sets.sigma-algebra-axioms* **by** *blast*
qed

corollary *nn-integral-measure*:
assumes *sets* $\mu =$ *sets* $X\ B \in$ *sets* Y
defines $\nu \equiv$ *measure-of* (*space* Y) (*sets* Y) $(\lambda B.\ \int^+ x.\ (\kappa\ x\ B)\ \partial\mu)$
shows $\nu\ B = (\int^+ x.\ (\kappa\ x\ B)\ \partial\mu)$

using *set-nn-integral-measure*[*OF assms*(*1*) *sets.top assms*(*2*)]
by(*simp add*: ν-*def sets-eq-imp-space-eq*[*OF assms*(*1*),*symmetric*])

lemma *distr-measure-kernel*:
assumes [*measurable*]:$f \in Y \to_M Z$
shows *measure-kernel X Z* (λx. *distr* (κ *x*) *Z f*)
unfolding *measure-kernel-def*
proof *safe*
fix *B*
assume *B*[*measurable*]: *B* $\in$ *sets Z*
show (λx. *emeasure* (*distr* (κ *x*) *Z f*) *B*) $\in$ *borel-measurable X*
by(*rule measurable-cong*[**where** *g*= (λx. κ *x* (*f* −‘ *B* ∩ *space Y*)),*THEN iffD2*])
(*auto simp*: *emeasure-distr sets-eq-imp-space-eq*[*OF kernel-sets*])
next
show $\bigwedge x$. *space Z* = {} $\Longrightarrow$ *x* $\in$ *space X* $\Longrightarrow$ *x* $\in$ {}
by (*metis Y-not-empty assms measurable-empty-iff*)
qed *auto*

lemma *measure-kernel-comp*:
assumes [*measurable*]: *f* $\in$ *W* $\to_M$ *X*
shows *measure-kernel W Y* (λx. κ (*f x*))
using *measurable-space*[*OF assms*] *kernel-sets Y-not-empty*
by(*auto simp*: *measure-kernel-def*)

lemma *emeasure-bind-kernel*:
assumes *sets* μ = *sets X B* $\in$ *sets Y*
shows (μ $\gg\!=_k$ κ) *B* = ($\int^+ x$. (κ *x B*) $\partial\mu$)
proof(*cases space X* = {})
assume *space X* = {}
then show *?thesis*
by (*metis assms*(*1*) *bind-kernel-def emeasure-empty emeasure-space nn-integral-empty*
sets-eq-imp-space-eq space-bot space-empty zero-order(*2*))
next
assume *h*:*space X* ≠ {}
have *sets* (κ (*SOME x*. *x* $\in$ *space* μ)) = *sets Y*
by(*rule someI2-ex*) (*use h kernel-sets sets-eq-imp-space-eq*[*OF assms*(*1*)] **in**
auto)
with *sets-eq-imp-space-eq*[*OF this*] **show** *?thesis*
by(*simp add*: *bind-kernel-def sets-eq-imp-space-eq*[*OF assms*(*1*)] *h nn-integral-measure*[*OF*
assms(*1*,*2*)])
qed

lemma *measure-bind-kernel*:
assumes [*measurable-cong*]:*sets* μ = *sets X* **and** [*measurable*]:*B* $\in$ *sets Y*
and *AE x in* μ. κ *x B* < ∞
shows *measure* (μ $\gg\!=_k$ κ) *B* = ($\int$ *x*. *measure* (κ *x*) *B* $\partial\mu$)
using *assms*(*3*) **by**(*auto simp*: *emeasure-bind-kernel*[*OF assms*(*1*,*2*)] *measure-def*
integral-eq-nn-integral
intro!: *arg-cong*[*of - - enn2real*] *nn-integral-cong-AE*)

lemma *sets-bind-kernel*:
assumes *space X* ≠ {} *sets* μ = *sets X*
shows *sets* (μ ⤜$_k$ κ) = *sets Y*
using *sets-bind-kernel*[*of* μ κ, *OF kernel-sets*,*simplified sets-eq-imp-space-eq*[*OF assms*(*2*)]]
by(*auto simp*: *assms*(*1*))

lemma *distr-bind-kernel*:
assumes *space X* ≠ {} **and** [*measurable-cong*]:*sets* μ = *sets X* **and** [*measurable*]: *f* ∈ *Y* →$_M$ *Z*
shows *distr* (μ ⤜$_k$ κ) *Z f* = μ ⤜$_k$ (λ*x*. *distr* (κ *x*) *Z f*)
proof −
{
fix *A*
assume *A*[*measurable*]:*A* ∈ *sets Z*
have *sets*[*measurable-cong*]:*sets* (μ ⤜$_k$ κ) = *sets Y*
by(*rule sets-bind-kernel*[*OF assms*(*1*,*2*)])
have *emeasure* (*distr* (μ ⤜$_k$ κ) *Z f*) *A* = *emeasure* (μ ⤜$_k$ (λ*x*. *distr* (κ *x*) *Z f*)) *A* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∫$^+$ *x*. *emeasure* (κ *x*) (*f* −‘ *A* ∩ *space Y*) ∂μ)
by(*simp add*: *emeasure-distr sets-eq-imp-space-eq*[*OF sets*] *emeasure-bind-kernel*[*OF assms*(*2*)])
also have ... = (∫$^+$ *x*. *emeasure* (*distr* (κ *x*) *Z f*) *A* ∂μ)
by(*auto simp*: *emeasure-distr sets-eq-imp-space-eq*[*OF assms*(*2*)] *sets-eq-imp-space-eq*[*OF kernel-sets*] *intro*!: *nn-integral-cong*)
also have ... = *?rhs*
by(*simp add*: *measure-kernel.emeasure-bind-kernel*[*OF distr-measure-kernel*[*OF assms*(*3*)] *assms*(*2*)])
finally show *?thesis* **.**
qed
}
thus *?thesis*
by(*auto intro*!: *measure-eqI simp*: *measure-kernel.sets-bind-kernel*[*OF distr-measure-kernel*[*OF assms*(*3*)] *assms*(*1*,*2*)])
qed

lemma *bind-kernel-distr*:
assumes [*measurable*]: *f* ∈ *W* →$_M$ *X* **and** *space W* ≠ {}
shows *distr W X f* ⤜$_k$ κ = *W* ⤜$_k$ (λ*x*. κ (*f x*))
proof −
have *X*: *space X* ≠ {}
using *measurable-space*[*OF assms*(*1*)] *assms*(*2*) **by** *auto*
show *?thesis*
by(*rule measure-eqI*, *insert X*)
(*auto simp*: *sets-bind-kernel*[*OF X*] *measure-kernel.sets-bind-kernel*[*OF measure-kernel-comp*[*OF assms*(*1*)] *assms*(*2*)]
emeasure-bind-kernel nn-integral-distr measure-kernel.emeasure-bind-kernel[*OF*

measure-kernel-comp[*OF assms*(*1*)]])
qed

lemma *bind-kernel-return*:
assumes $x \in$ *space* X
shows *return* $X\ x \gg\!=_k \kappa = \kappa\ x$
proof −
have X: *space* $X \neq \{\}$
using *assms* **by** *auto*
show *?thesis*
by(*rule measure-eqI*)
(*auto simp*: *sets-bind-kernel*[*OF X sets-return*] *kernel-sets*[*OF assms*] *emeasure-bind-kernel*[*OF sets-return*] *nn-integral-return*[*OF assms*])
qed

lemma *nn-integral-measurable-kernel*:
assumes $f \in$ *borel-measurable* Y
shows $(\lambda x.\ (\int^+ y.\ f\ y\ \partial(\kappa\ x))) \in$ *borel-measurable* X
using *assms*
proof *induction*
case (*cong f g*)
then show *?case*
by(*auto intro*!: *measurable-cong*[**where** $f{=}\lambda x.\ \int^+ y.\ f\ y\ \partial(\kappa\ x)$ **and** $g{=}\ \lambda x.\ \int^+ y.\ g\ y\ \partial(\kappa\ x)$,*THEN iffD2*] *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF kernel-sets*])
next
case (*set A*)
then show *?case*
by(*auto intro*!: *measurable-cong*[**where** $f{=}\lambda x.\ integral^N\ (\kappa\ x)\ (indicator\ A)$ **and** $g{=}\lambda x.\ \kappa\ x\ A$,*THEN iffD2*])
next
case (*mult u c*)
then show *?case*
by(*auto intro*!: *measurable-cong*[**where** $f{=}\lambda x.\ \int^+ y.\ c * u\ y\ \partial\kappa\ x$ **and** $g{=}\lambda x.\ c * \int^+ y.\ u\ y\ \partial\kappa\ x$,*THEN iffD2*] *simp*: *nn-integral-cmult*)
next
case (*add u v*)
then show *?case*
by(*auto intro*!: *measurable-cong*[**where** $f{=}\lambda x.\ \int^+ y.\ v\ y + u\ y\ \partial\kappa\ x$ **and** $g{=}\lambda x.\ (\int^+ y.\ v\ y\ \partial\kappa\ x) + (\int^+ y.\ u\ y\ \partial\kappa\ x)$,*THEN iffD2*] *simp*: *nn-integral-add*)
next
case (*seq U*)
then show *?case*
by(*intro measurable-cong*[**where** $f{=}\lambda x.\ integral^N\ (\kappa\ x)\ (\bigsqcup\ range\ U)$ **and** $g{=}\lambda x.\ \bigsqcup i.\ integral^N\ (\kappa\ x)\ (U\ i)$,*THEN iffD2*])
(*auto simp*: *nn-integral-monotone-convergence-SUP*[*of U*,*simplified SUP-apply*[*symmetric*]])
qed

corollary *integrable-measurable-kernel*:

fixes $f :: 'b \Rightarrow 'c$::{*banach*, *second-countable-topology*}
assumes [*measurable*]:$f \in$ *borel-measurable* Y
shows *Measurable.pred* X (λx. *integrable* (κ x) f)
proof(*subst measurable-cong*)
show $\bigwedge x.\ x \in$ *space* $X \Longrightarrow$ *integrable* (κ x) $f \longleftrightarrow$ ($\int^+ y.$ *ennreal* (*norm* (f y)) $\partial\kappa$ x) $< \infty$
by(*auto simp*: *integrable-iff-bounded*)
qed(*use nn-integral-measurable-kernel* **in** *auto*)

lemma *integral-measurable-kernel*:
fixes $f :: 'b \Rightarrow 'c$::{*banach*, *second-countable-topology*}
assumes f[*measurable*]: $f \in$ *borel-measurable* Y
shows (λx. ($\int$ y. f y $\partial(\kappa$ $x)$)) $\in$ *borel-measurable* X
proof −
from *borel-measurable-implies-sequence-metric*[*OF* f, *of* 0]
obtain F **where** F: $\bigwedge i$. *simple-function* Y (F i)
$\bigwedge x.\ x \in$ *space* $Y \Longrightarrow$ ($\lambda i.\ F$ i x) $\longrightarrow$ f x
$\bigwedge i$ $x.\ x \in$ *space* $Y \Longrightarrow$ *norm* (F i x) $\leq$ 2 $*$ *norm* (f x)
unfolding *norm-conv-dist* **by** *blast*

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

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

have (λx. F' (κ x) i) $\in X \rightarrow_M$ *borel* **for** i
proof (*rule measurable-cong*[*THEN iffD2*])
fix x
assume x:$x \in$ *space* X
note [*simp*] = *kernel-space*[*OF* x] *kernel-sets*[*OF* x]
have F' (κ x) i = (*if integrable* (κ x) f *then Bochner-Integration.simple-bochner-integral* (κ x) (F i) *else* 0)
proof (*cases integrable* (κ x) f)
assume f:*integrable* (κ x) f
have *Bochner-Integration.simple-bochner-integral* (κ x) (F i) = *integral*^{L} (κ x) (F i)
proof(*intro simple-bochner-integrable-eq-integral*)
have $*$:*integrable* (κ x) ($\lambda y.$ 2 $*$ *norm* (f y))
using f **by** *simp*
have *integrable* (κ x) (F i)
by(*rule Bochner-Integration.integrable-bound*[*OF* $*$])
(*auto simp add*: $F(1)$ $F(3)$ *borel-measurable-simple-function simple-function-cong-algebra*)
thus *Bochner-Integration.simple-bochner-integrable* (κ x) (F i)
by (*simp add*: $F(1)$ *integrable-iff-bounded simple-bochner-integrableI-bounded simple-function-cong-algebra*)
qed

```
    thus ?thesis
      by(simp add: F'-def)
  qed(simp add: F'-def)
  then show F' (κ x) i = (if integrable (κ x) f then ∑ y∈F i ‘ space Y. measure
(κ x) {x∈space Y. F i x = y} *R y else 0)
    unfolding simple-bochner-integral-def by simp
next
  have [measurable]: (λx. measure (κ x) {x∈space Y. F i x = y} *R y) ∈
borel-measurable X if y:y ∈ F i ‘ space Y for y
    using borel-measurable-simple-function[OF F(1)[of i]]
    by(auto intro!: borel-measurable-scaleR measure-measurable)
  show (λx. if integrable (κ x) f then ∑ y∈F i ‘ space Y. measure (κ x) {x∈space
Y. F i x = y} *R y else 0) ∈ borel-measurable X
    using integrable-measurable-kernel[OF assms] by measurable
qed
moreover
have F' (κ x) ⟶ integral^L (κ x) f if x: x ∈ space X for x
proof cases
  note [simp] = kernel-space[OF x] kernel-sets[OF x]
  assume integrable (κ x) f then show ?thesis
    unfolding F'-def using F(1)[THEN borel-measurable-simple-function] F
   by (auto intro!: integral-dominated-convergence[where w=λx. 2 * norm (f x)]
          cong: measurable-cong-sets)
qed (auto simp: F'-def not-integrable-integral-eq)
ultimately show ?thesis
  by (rule borel-measurable-LIMSEQ-metric)
qed

lemma density-measure-kernel':
  assumes f[measurable]: f ∈ Y →M borel
  shows measure-kernel X Y (λx. density (κ x) f)
proof(cases space X = {})
  assume space X = {}
  then show ?thesis
    by(auto simp: measure-kernel-def dest:space-empty)
next
  assume X: space X ≠ {}
  show measure-kernel X Y (λx. density (κ x) f)
  proof
    fix B
    assume [measurable]: B ∈ sets Y
    show (λx. emeasure (density (κ x) f) B) ∈ borel-measurable X
    proof(subst measurable-cong)
      show (λx. set-nn-integral (κ x) B f) ∈ borel-measurable X
        by(auto intro!: nn-integral-measurable-kernel)
    qed(auto simp: emeasure-density)
  qed(auto simp: kernel-sets Y-not-empty)
qed
```

lemma *nn-integral-bind-kernel*:
assumes $f \in$ *borel-measurable* Y *sets* $\mu =$ *sets* X
shows $(\int^+ y.\ f\ y\ \partial(\mu \gg\!=_k \kappa)) = (\int^+ x.\ (\int^+ y.\ f\ y\ \partial(\kappa\ x))\ \partial\mu)$
proof(*cases space* $X = \{\}$)
case *True*
then show *?thesis*
by(*simp add*: *sets-eq-imp-space-eq*[*OF assms*(*2*)] *bind-kernel-def nn-integral-empty*)
next
case *X*:*False*
then have μ:*space* $\mu \neq \{\}$ **by**(*simp add*: *sets-eq-imp-space-eq*[*OF assms*(*2*)])
note *1*[*measurable-cong*] = *assms*(*2*) *sets-bind-kernel*[*OF X assms*(*2*)]
from *assms*(*1*) **show** *?thesis*
proof *induction*
case *ih*:(*cong f g*)
have $(\int^+ y.\ f\ y\ \partial(\mu \gg\!=_k \kappa)) = (\int^+ y.\ g\ y\ \partial(\mu \gg\!=_k \kappa))$ $(\int^+ x.\ integral^N\ (\kappa\ x)\ f\ \partial\mu) = (\int^+ x.\ integral^N\ (\kappa\ x)\ g\ \partial\mu)$
by(*auto intro*!: *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF 1*(*2*)] *sets-eq-imp-space-eq*[*OF assms*(*2*)] *sets-eq-imp-space-eq*[*OF kernel-sets*] *ih*(*3*))
then show *?case*
by(*simp add*: *ih*)
next
case (*set A*)
then show *?case*
by(*auto simp*: *emeasure-bind-kernel*[*OF 1*(*1*)] *sets-eq-imp-space-eq*[*OF 1*(*1*)] *intro*!: *nn-integral-cong*)
next
case *ih*:(*mult u c*)
then have $(\int^+ x.\ \int^+ y.\ c * u\ y\ \partial\kappa\ x\ \partial\mu) = (\int^+ x.\ c * \int^+ y.\ u\ y\ \partial\kappa\ x\ \partial\mu)$
by(*auto intro*!: *nn-integral-cong nn-integral-cmult simp*: *sets-eq-imp-space-eq*[*OF 1*(*1*)])
with *ih nn-integral-measurable-kernel*[*of* $\lambda y.\ u\ y$] **show** *?case*
by(*auto simp*: *nn-integral-cmult intro*!: *nn-integral-cong*)
next
case *ih*:(*add u v*)
then have $(\int^+ x.\ \int^+ y.\ v\ y + u\ y\ \partial\kappa\ x\ \partial\mu) = (\int^+ x.\ (\int^+ y.\ v\ y\ \partial\kappa\ x) + (\int^+ y.\ u\ y\ \partial\kappa\ x)\ \partial\mu)$
by(*auto intro*!: *nn-integral-cong simp*: *nn-integral-add sets-eq-imp-space-eq*[*OF 1*(*1*)])
with *ih nn-integral-measurable-kernel*[*of* $\lambda y.\ u\ y$] *nn-integral-measurable-kernel*[*of* $\lambda y.\ v\ y$]
show *?case*
by(*simp add*: *nn-integral-add*)
next
case *ih*[*measurable*]:(*seq U*)
show *?case* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $((\bigsqcup i.\ integral^N\ (\mu \gg\!=_k \kappa)\ (U\ i)))$
by(*rule nn-integral-monotone-convergence-SUP*[*of U*,*simplified SUP-apply*[*of U UNIV*,*symmetric*]]) (*use ih* **in** *auto*)

```
    also have ... = (⨆ i. ∫⁺ x. (∫⁺ y. U i y ∂κ x) ∂μ)
      by(simp add: ih)
    also have ... = (∫⁺ x. (⨆ i. (∫⁺ y. U i y ∂κ x)) ∂μ)
    proof(rule nn-integral-monotone-convergence-SUP[symmetric])
      show incseq (λi x. ∫⁺ y. U i y ∂κ x)
          by standard+ (auto intro!: le-funI nn-integral-mono simp:le-funD[OF
incseqD[OF ih(3)]])
    qed(use nn-integral-measurable-kernel[of λy. U - y] in simp)
    also have ... = ?rhs
    by(intro nn-integral-cong nn-integral-monotone-convergence-SUP[of U,simplified
SUP-apply[of U UNIV,symmetric],OF ih(3),symmetric])
        (auto simp: sets-eq-imp-space-eq[OF 1(1)])
    finally show ?thesis .
  qed
 qed
qed

lemma bind-kernel-measure-kernel:
 assumes measure-kernel Y Z k'
 shows measure-kernel X Z (λx. κ x ⤜ₖ k')
proof(cases space X = {})
 case True
 then show ?thesis
  by(auto simp: measure-kernel-def measurable-def)
next
 case X:False
 then have Y: space Y ≠ {}
  by(simp add: Y-not-empty)
 interpret k': measure-kernel Y Z k' by fact
 show ?thesis
 proof
  fix B
  assume B ∈ sets Z
  with k'.emeasure-bind-kernel[OF kernel-sets,of - B] show (λx. emeasure (κ x
⤜ₖ k') B) ∈ borel-measurable X
   by(auto intro!: measurable-cong[where f=λx. emeasure (κ x ⤜ₖ k') B and
g=λx. ∫⁺ y. emeasure (k' y) B ∂κ x,THEN iffD2] nn-integral-measurable-kernel
          simp: sets-eq-imp-space-eq[OF kernel-sets] Y)
 qed(use k'.Y-not-empty Y k'.sets-bind-kernel[OF Y kernel-sets] in auto)
qed

lemma restrict-measure-kernel: measure-kernel (restrict-space X A) Y κ
proof
 fix B
 assume B ∈ sets Y
 from emeasure-measurable[OF this] show (λx. emeasure (κ x) B) ∈ borel-measurable
(restrict-space X A)
  using measurable-restrict-space1 by blast
qed(insert Y-not-empty,auto simp add: space-restrict-space kernel-sets)
```

end

lemma *measure-kernel-cong-sets*:
assumes *sets X = sets X' sets Y = sets Y'*
shows *measure-kernel X Y = measure-kernel X' Y'*
by *standard* (*simp add*: *measure-kernel-def measurable-cong-sets*[*OF assms*(*1*) *refl*] *sets-eq-imp-space-eq*[*OF assms*(*1*)] *assms*(*2*) *sets-eq-imp-space-eq*[*OF assms*(*2*)])

lemma *measure-kernel-cong*:
assumes $\bigwedge x.\ x \in$ *space X* $\Longrightarrow$ *k x = k' x*
shows *measure-kernel X Y k = measure-kernel X Y k'*
using *assms* **by**(*auto cong*: *measurable-cong simp*: *measure-kernel-def*)

lemma *measure-kernel-pair-countble1*:
assumes *countable A* $\bigwedge i.\ i \in A \Longrightarrow$ *measure-kernel X Y* ($\lambda x.\ k\ (i,x)$)
shows *measure-kernel* (*count-space A* $\bigotimes_M$ *X*) *Y k*
using *assms* **by**(*auto simp*: *measure-kernel-def space-pair-measure intro*!: *measurable-pair-measure-countable1*)

lemma *measure-kernel-empty-trivial*:
assumes *space X = {}*
shows *measure-kernel X Y k*
using *assms* **by**(*auto simp*: *measure-kernel-def measurable-def*)

lemma *measure-kernel-const'*: *space Y* $\neq$ *{}* $\Longrightarrow$ *sets* μ *= sets Y* $\Longrightarrow$ *measure-kernel X Y* ($\lambda r.\ \mu$)
by(*auto simp*: *measure-kernel-def*)

2.3 Finite Kernel

locale *finite-kernel = measure-kernel* +
assumes *finite-measure-spaces*: $\exists\, r{<}\infty.\ \forall\, x{\in}$ *space X*. κ *x* (*space Y*) $<$ *r*
begin

lemma *finite-measures*:
assumes *x* $\in$ *space X*
shows *finite-measure* (κ *x*)
proof−
obtain *r* **where** κ *x* (*space Y*) $<$ *r*
using *finite-measure-spaces assms* **by** *metis*
then show *?thesis*
by(*auto intro*!: *finite-measureI simp*: *sets-eq-imp-space-eq*[*OF kernel-sets*[*OF assms*]])
qed

end

lemma *finite-kernel-empty-trivial*: *space X* = *{}* $\Longrightarrow$ *finite-kernel X Y f*

by(*auto simp*: *finite-kernel-def finite-kernel-axioms-def measure-kernel-empty-trivial intro*!: *exI*[**where** *x=1*])

lemma *finite-kernel-cong-sets*:
assumes *sets X = sets X' sets Y = sets Y'*
shows *finite-kernel X Y = finite-kernel X' Y'*
by *standard* (*auto simp*: *measure-kernel-cong-sets*[*OF assms*] *finite-kernel-def finite-kernel-axioms-def sets-eq-imp-space-eq*[*OF assms*(*1*)] *sets-eq-imp-space-eq*[*OF assms*(*2*)])

2.4 Sub-Probability Kernel

locale *subprob-kernel* = *measure-kernel* +
assumes *subprob-spaces*: $\bigwedge x.\ x \in space\ X \Longrightarrow subprob\text{-}space\ (\kappa\ x)$
begin
lemma *subprob-space*:
$\bigwedge x.\ x \in space\ X \Longrightarrow \kappa\ x\ (space\ Y) \leq 1$
by (*simp add*: *subprob-space.subprob-emeasure-le-1 subprob-spaces*)

lemma *subprob-measurable*[*measurable*]:
$\kappa \in X \rightarrow_M subprob\text{-}algebra\ Y$
by(*auto intro*!: *measurable-subprob-algebra-generated*[*OF sets.sigma-sets-eq*[*symmetric*] *sets.Int-stable sets.space-closed*] *simp*: *subprob-spaces kernel-sets emeasure-measurable*)

lemma *finite-kernel*: *finite-kernel X Y* κ
by(*auto simp*: *finite-kernel-def finite-kernel-axioms-def intro*!: *measure-kernel-axioms exI*[**where** *x=2*] *order.strict-trans1*[*OF subprob-space.subprob-emeasure-le-1*[*OF subprob-spaces*]])

sublocale *finite-kernel*
by (*rule finite-kernel*)

end

lemma *subprob-kernel-def'*:
subprob-kernel X Y $\kappa \longleftrightarrow \kappa \in X \rightarrow_M subprob\text{-}algebra\ Y$
by(*auto simp*: *subprob-kernel.subprob-measurable subprob-kernel-def subprob-kernel-axioms-def measure-kernel-def measurable-subprob-algebra measurable-empty-iff space-subprob-algebra-empty-iff*)
(*auto simp*: *subprob-measurableD*(*2*) *subprob-space-kernel*)

lemmas *subprob-kernelI* = *measurable-subprob-algebra*[*simplified subprob-kernel-def'*[*symmetric*]]

lemma *subprob-kernel-cong-sets*:
assumes *sets X = sets X' sets Y = sets Y'*
shows *subprob-kernel X Y = subprob-kernel X' Y'*
by *standard* (*auto simp*: *subprob-kernel-def' subprob-algebra-cong*[*OF assms*(*2*)] *measurable-cong-sets*[*OF assms*(*1*) *refl*])

lemma *subprob-kernel-empty-trivial*:

assumes *space X = {}*
shows *subprob-kernel X Y k*
using *assms* **by**(*auto simp*: *subprob-kernel-def subprob-kernel-axioms-def intro*!: *measure-kernel-empty-trivial*)

lemma *bind-kernel-bind*:
assumes $f \in M \rightarrow_M$ *subprob-algebra N*
shows $M \gg=_k f = M \gg= f$
proof(*cases space M = {}*)
case *True*
then show *?thesis*
by(*simp add*: *bind-kernel-def bind-def*)
next
case *h*:*False*
interpret *subprob-kernel M N f*
using *assms*(*1*) **by**(*simp add*: *subprob-kernel-def'*)
show *?thesis*
by(*rule measure-eqI*,*insert sets-kernel*[*OF assms*])
(*auto simp*: *h sets-bind-kernel emeasure-bind-kernel emeasure-bind*[*OF h assms*])
qed

lemma(**in** *measure-kernel*) *subprob-kernel-sum*:
assumes $\bigwedge x.\ x \in$ *space X* $\Longrightarrow$ *finite-measure* $(\kappa\ x)$
obtains *ki* **where** $\bigwedge i.$ *subprob-kernel X Y* $(ki\ i)$ $\bigwedge A\ x.\ x \in$ *space X* $\Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
proof −
obtain *u* **where** *u*: $\bigwedge x.\ x \in$ *space X* $\Longrightarrow u\ x < \infty$ $\bigwedge x.\ x \in$ *space X* $\Longrightarrow u\ x = \kappa\ x$ (*space Y*)
using *finite-measure.emeasure-finite*[*OF assms*]
by (*simp add*: *top.not-eq-extremum*)
have [*measurable*]: *u* ∈ *borel-measurable X*
by(*simp cong*: *measurable-cong add*: *u*(*2*))
define *ki* **where** *ki* ≡ (λ*i x*. *if i* < *nat* ⌈*enn2real* (*u x*)⌉ *then scale-measure* (*1* / *ennreal* (*real-of-int* ⌈*enn2real* (*u x*)⌉)) (κ *x*) *else* (*sigma* (*space Y*) (*sets Y*)))
have *1*:$\bigwedge i\ x.\ x \in$ *space X* $\Longrightarrow$ *sets* (*ki i x*) = *sets Y*
by(*auto simp*: *ki-def kernel-sets*)
have *subprob-kernel X Y* (*ki i*) **for** *i*
proof −
{
fix *i B*
assume [*measurable*]: *B* ∈ *sets Y*
have (λ*x*. *emeasure* (*ki i x*) *B*) = (λ*x*. *if i* < *nat* ⌈*enn2real* (*u x*)⌉ *then* (*1* / *ennreal* (*real-of-int* ⌈*enn2real* (*u x*)⌉)) ∗ *emeasure* (κ *x*) *B else 0*)
by(*auto simp*: *ki-def emeasure-sigma*)
also have ... ∈ *borel-measurable X*
by *simp*
finally have (λ*x*. *emeasure* (*ki i x*) *B*) ∈ *borel-measurable X* **.**
}

```
    moreover {
      fix i x
      assume x:x ∈ space X
      have emeasure (ki i x) (space Y) ≤ 1
        by(cases u x = 0,auto simp: ki-def emeasure-sigma u(2)[OF x,symmetric])
(metis u(1)[OF x,simplified] divide-ennreal-def divide-le-posI-ennreal enn2real-le
le-of-int-ceiling mult.commute mult.right-neutral not-gr-zero order.strict-iff-not)
      hence subprob-space (ki i x)
      using x Y-not-empty by(fastforce intro!: subprob-spaceI simp: sets-eq-imp-space-eq[OF
1[OF x]])
    }
    ultimately show ?thesis
        by(auto simp: subprob-kernel-def measure-kernel-def 1 Y-not-empty sub-
prob-kernel-axioms-def)
  qed
  moreover have κ x A = (∑ i. ki i x A) if x:x ∈ space X for x A
  proof (cases A ∈ sets Y)
    case A[measurable]:True
    have emeasure (κ x) A = (∑ i<nat ⌈enn2real (u x)⌉. emeasure (ki i x) A)
    proof(cases u x = 0)
      case True
      then show ?thesis
      using u(2)[OF that] by simp (metis A emeasure-eq-0 kernel-sets sets.sets-into-space
sets.top x)
    next
      case u0:False
      hence real-of-int ⌈enn2real (u x)⌉ > 0
        by (metis enn2real-nonneg ennreal-0 ennreal-enn2real-if infinity-ennreal-def
linorder-not-le nat-0-iff nle-le of-int-le-0-iff of-nat-eq-0-iff real-nat-ceiling-ge u(1)
x)
       with u(1)[OF x] have of-nat (nat ⌈enn2real (u x)⌉) / ennreal (real-of-int
⌈enn2real (u x)⌉) = 1
        by(simp add: ennreal-eq-0-iff ennreal-of-nat-eq-real-of-nat)
      thus ?thesis
       by(simp add: ki-def ennreal-divide-times[symmetric] mult.assoc[symmetric])
    qed
    then show ?thesis
     by(auto simp: suminf-offset[of λi. emeasure (ki i x) A nat ⌈enn2real (u x)⌉])
(simp add: ki-def emeasure-sigma)
  next
    case False
    then show ?thesis
      using kernel-sets[OF x] 1[OF x]
      by(simp add: emeasure-notin-sets)
  qed
  ultimately show ?thesis
    using that by blast
qed
```

2.5 Probability Kernel

locale *prob-kernel* = *measure-kernel* +
  assumes *prob-spaces*: $\bigwedge x.\ x \in space\ X \Longrightarrow prob\text{-}space\ (\kappa\ x)$
begin

lemma *prob-space*:
 $\bigwedge x.\ x \in space\ X \Longrightarrow \kappa\ x\ (space\ Y) = 1$
  using *kernel-space prob-space.emeasure-space-1 prob-spaces* **by** *fastforce*

lemma *prob-measurable*[*measurable*]:
 $\kappa \in X \to_M prob\text{-}algebra\ Y$
 by(*auto intro*!: *measurable-prob-algebra-generated*[*OF sets.sigma-sets-eq*[*symmetric*]
sets.Int-stable sets.space-closed] *simp*: *prob-spaces kernel-sets emeasure-measurable*)

lemma *subprob-kernel*: *subprob-kernel X Y* κ
  by (*simp add*: *measurable-prob-algebraD subprob-kernel-def'*)

sublocale *subprob-kernel*
  by (*simp add*: *subprob-kernel*)

lemma *restrict-probability-kernel*:
  prob-kernel (*restrict-space X A*) *Y* κ
 by(*auto simp*: *prob-kernel-def restrict-measure-kernel prob-kernel-axioms-def space-restrict-space
prob-spaces*)

end

lemma *prob-kernel-def'*:
 prob-kernel X Y $\kappa \longleftrightarrow \kappa \in X \to_M prob\text{-}algebra\ Y$
proof
  assume *h*:$\kappa \in X \to_M prob\text{-}algebra\ Y$
  show *prob-kernel X Y* κ
    using *subprob-measurableD*(*2*)[*OF measurable-prob-algebraD*[*OF h*]] *measur-
able-space*[*OF h*] *measurable-emeasure-kernel*[*OF measurable-prob-algebraD*[*OF h*]]
   by(*auto simp*: *prob-kernel-def measure-kernel-def prob-kernel-axioms-def space-prob-algebra
*) (*metis prob-space.not-empty sets-eq-imp-space-eq*)
qed(*auto simp*: *prob-kernel.prob-measurable prob-kernel-def prob-kernel-axioms-def
measure-kernel-def*)

lemma *bind-kernel-return''*:
  assumes *sets M = sets N*
  shows $M \gg\!=_k return\ N = M$
proof(*cases space M* = {})
  case *True*
  then show *?thesis*
   by(*simp add*: *bind-kernel-def space-empty*[*symmetric*])
next
  case *False*

then have *1*: *space N* ≠ {}
by(*simp add*: *sets-eq-imp-space-eq*[*OF assms*])
interpret *prob-kernel N N return N*
by(*simp add*: *prob-kernel-def'*)
show *?thesis*
by(*rule measure-eqI*) (*auto simp*: *emeasure-bind-kernel sets-bind-kernel*[*OF 1 assms*] *assms*)
qed

2.6 S-Finite Kernel

locale *s-finite-kernel* = *measure-kernel* +
assumes *s-finite-kernel-sum*: ∃ *ki*. (∀ *i*. *finite-kernel X Y* (*ki i*) ∧ (∀ *x*∈*space X*. ∀ *A*∈*sets Y*. *κ x A* = (∑ *i*. *ki i x A*)))

lemma *s-finite-kernel-subI*:
assumes ⋀*x*. *x* ∈ *space X* ⟹ *sets* (*κ x*) = *sets Y* ⋀*i*. *subprob-kernel X Y* (*ki i*) ⋀*x A*. *x* ∈ *space X* ⟹ *A* ∈ *sets Y* ⟹ *emeasure* (*κ x*) *A* = (∑ *i*. *ki i x A*)
shows *s-finite-kernel X Y κ*
proof −
interpret *measure-kernel X Y κ*
proof
show *B* ∈ *sets Y* ⟹ (λ*x*. *emeasure* (*κ x*) *B*) ∈ *borel-measurable X* **for** *B*
using *assms*(*2*) **by**(*simp add*: *assms*(*3*) *subprob-kernel-def' cong*: *measurable-cong*)
next
show *space X* ≠ {} ⟹ *space Y* ≠ {}
using *assms*(*2*)[*of 0*] **by**(*auto simp*: *subprob-kernel-def measure-kernel-def*)
qed *fact*
show *?thesis*
by (*auto simp*: *s-finite-kernel-def measure-kernel-axioms s-finite-kernel-axioms-def assms*(*2*,*3*) *intro*!: *exI*[**where** *x*=*ki*] *subprob-kernel.finite-kernel*)
qed

context *s-finite-kernel*
begin

lemma *s-finite-kernels-fin*:
obtains *ki* **where** ⋀*i*. *finite-kernel X Y* (*ki i*) ⋀*x A*. *x* ∈ *space X* ⟹ *κ x A* = (∑ *i*. *ki i x A*)
proof −
obtain *ki* **where** *ki*:⋀*i*. *finite-kernel X Y* (*ki i*) ⋀*x A*. *x* ∈ *space X* ⟹ *A* ∈ *sets Y* ⟹ *κ x A* = (∑ *i*. *ki i x A*)
by(*metis s-finite-kernel-sum*)
hence *κ x A* = (∑ *i*. *ki i x A*) **if** *x* ∈ *space X* **for** *x A*
by(*cases A* ∈ *sets Y*, *insert that kernel-sets*[*OF that*]) (*auto simp*: *finite-kernel-def measure-kernel-def emeasure-notin-sets*)
with *ki* **show** *?thesis*
using *that* **by** *auto*

qed

lemma *s-finite-kernels*:
obtains *ki* **where** $\bigwedge i.$ *subprob-kernel X Y* (*ki i*) $\bigwedge x\ A.\ x \in$ *space X* $\Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
proof −
obtain *ki* **where** *ki*:$\bigwedge i.$ *finite-kernel X Y* (*ki i*) $\bigwedge x\ A.\ x \in$ *space X* $\Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
by(*metis s-finite-kernels-fin*)
have $\exists$ *kij*. $(\forall j.$ *subprob-kernel X Y* (*kij j*)) $\wedge$ $(\forall x\ A.\ x \in$ *space X* $\longrightarrow$ *ki i x A* $= (\sum j.\ kij\ j\ x\ A))$ **for** *i*
using *measure-kernel.subprob-kernel-sum*[*of X Y ki i*, *OF - finite-kernel.finite-measures*[*OF ki*(*1*)[*of i*]]] *ki*(*1*)[*of i*] **by**(*metis finite-kernel-def*)
then obtain *kij* **where** *kij*: $\bigwedge i\ j.$ *subprob-kernel X Y* (*kij i j*) $\bigwedge x\ A\ i.\ x \in$ *space X* $\Longrightarrow$ *ki i x A* $= (\sum j.\ kij\ i\ j\ x\ A)$
by *metis*
have $\bigwedge i.$ *subprob-kernel X Y* (*case-prod kij* (*prod-decode i*))
using *kij*(*1*) **by**(*auto simp*: *split-beta*)
moreover have *x* $\in$ *space X* $\Longrightarrow \kappa\ x\ A = (\sum i.$ *case-prod kij* (*prod-decode i*) *x A*) **for** *x A*
using *suminf-ennreal-2dimen*[*of* $\lambda i.$ *ki i x A* $\lambda(i,j).$ *kij i j x A*]
by(*auto simp*: *ki*(*2*) *kij*(*2*) *split-beta'*)
ultimately show *?thesis*
using *that* **by** *fastforce*
qed

lemma *image-s-finite-measure*:
assumes *x* $\in$ *space X*
shows *s-finite-measure* (κ *x*)
proof −
obtain *ki* **where** *ki*:$\bigwedge i.$ *subprob-kernel X Y* (*ki i*) $\bigwedge x\ A.\ x \in$ *space X* $\Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
by(*metis s-finite-kernels*)
show *?thesis*
using *ki*(*1*)[*simplified subprob-kernel-def'*] *measurable-space*[*OF ki*(*1*)[*simplified subprob-kernel-def'*] *assms*]
by(*auto intro*!: *s-finite-measureI*[**where** *Mi*=$\lambda i.$ *ki i x*] *subprob-space.axioms*(*1*) *simp*: *kernel-sets*[*OF assms*] *space-subprob-algebra ki*(*2*)[*OF assms*])
qed

corollary *kernel-measurable-s-finite*[*measurable*]:$\kappa \in X \rightarrow_M$ *s-finite-measure-algebra Y*
by(*auto intro*!: *measurable-s-finite-measure-algebra simp*: *kernel-sets image-s-finite-measure*)

lemma *comp-measurable*:
assumes *f*[*measurable*]:*f* $\in M \rightarrow_M X$
shows *s-finite-kernel M Y* ($\lambda x.\ \kappa\ (f\ x)$)
proof −
obtain *ki* **where** *ki*:$\bigwedge i.$ *subprob-kernel X Y* (*ki i*) $\bigwedge x\ A.\ x \in$ *space X* $\Longrightarrow \kappa\ x$

$A = (\sum i.\ ki\ i\ x\ A)$
by(*metis s-finite-kernels*)
show *?thesis*
using *ki*(*1*) *measurable-space*[*OF f*] **by**(*auto intro*!: *s-finite-kernel-subI*[**where** *ki*=$\lambda i\ x.\ ki\ i\ (f\ x)$] *simp*: *subprob-kernel-def'* *ki*(*2*) *kernel-sets*)
qed

lemma *distr-s-finite-kernel*:
assumes *f*[*measurable*]: $f \in Y \to_M Z$
shows *s-finite-kernel X Z* ($\lambda x.$ *distr* ($\kappa\ x$) *Z f*)
proof −
obtain *ki* **where** *ki*:$\bigwedge i.$ *subprob-kernel X Y* (*ki i*) $\bigwedge x\ A.\ x \in space\ X \Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
by(*metis s-finite-kernels*)
hence *1*:$x \in space\ X \Longrightarrow space\ (ki\ i\ x) = space\ Y$ **for** *x i*
by(*auto simp*: *subprob-kernel-def' intro*!: *subprob-measurableD*(*1*)[*of - X Y*])
have [*measurable*]:$B \in sets\ Z \Longrightarrow (\lambda x.\ emeasure\ (distr\ (\kappa\ x)\ Z\ f)\ B) \in borel\text{-}measurable\ X$ **for** *B*
by(*rule measurable-cong*[**where** g=$\lambda x.\ \kappa\ x\ (f -` B \cap space\ Y)$, *THEN iffD2*]) (*auto simp*: *emeasure-distr sets-eq-imp-space-eq*[*OF kernel-sets*])
show *?thesis*
using *ki*(*1*) *measurable-distr*[*OF f*] **by**(*auto intro*!: *s-finite-kernel-subI*[**where** *ki*=$\lambda i\ x.\ distr\ (ki\ i\ x)\ Z f$] *simp*: *subprob-kernel-def' emeasure-distr ki*(*2*) *sets-eq-imp-space-eq*[*OF kernel-sets*] *1*)
qed

lemma *comp-s-finite-measure*:
assumes *s-finite-measure* μ **and** [*measurable-cong*]: *sets* μ = *sets X*
shows *s-finite-measure* ($\mu \gg\!=_k \kappa$)
proof(*cases space X* = {})
case *1*:*True*
show *?thesis*
by(*auto simp*: *sets-eq-imp-space-eq*[*OF assms*(*2*)] *1 bind-kernel-def intro*!: *finite-measure.s-finite-measure-finite-measure finite-measureI*)
next
case *0*:*False*
then have *1*: *space* $\mu \neq \{\}$
by(*simp add*: *sets-eq-imp-space-eq*[*OF assms*(*2*)])
have *2*: *sets* (κ (*SOME x. x* ∈ *space* μ)) = *sets Y*
by(*rule someI2-ex*, *insert 1 kernel-sets*) (*auto simp*: *sets-eq-imp-space-eq*[*OF assms*(*2*)])
have *sets-bind*[*measurable-cong*]: *sets* ($\mu \gg\!=_k \kappa$) = *sets Y*
by(*simp add*: *bind-kernel-def 1 sets-eq-imp-space-eq*[*OF 2*] *2*)
obtain μi **where** *mui*[*measurable-cong*]: $\bigwedge i.$ *sets* ($\mu i\ i$) = *sets X* $\bigwedge i.$ ($\mu i\ i$) (*space X*) ≤ *1* $\bigwedge A.\ \mu\ A = (\sum i.\ \mu i\ i\ A)$
using *s-finite-measure.finite-measures*[*OF assms*(*1*)] *assms*(*2*) *sets-eq-imp-space-eq*[*OF assms*(*2*)] **by** *metis*
obtain *ki* **where** *ki*:$\bigwedge i.$ *subprob-kernel X Y* (*ki i*) $\bigwedge x\ A.\ x \in space\ X \Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$

by(*metis s-finite-kernels*)
define *Mi* **where** *Mi* ≡ (λ*n*. (λ(*i*,*j*). *measure-of* (*space Y*) (*sets Y*) (λ*A*. ∫⁺*x*. (*ki i x A*) ∂(μ*i j*))) (*prod-decode n*))
have *emeasure*:(μ ⤜ₖ κ) *A* = (∑ *i*. (*Mi i*) *A*) (**is** *?lhs* = *?rhs*) **if** *A* ∈ *sets Y* **for** *A*
proof −
have *?lhs* = (∫⁺*x*. (κ *x A*) ∂μ)
by(*simp add*: *emeasure-bind-kernel*[*OF assms*(*2*) *that*])
also have ... = (∫⁺*x*. (∑ *i*. (*ki i x A*)) ∂μ)
by(*auto intro*!: *nn-integral-cong simp*: *ki sets-eq-imp-space-eq*[*OF assms*(*2*)])
also have ... = (∑ *i*. ∫⁺*x*. (*ki i x A*) ∂μ)
by(*auto intro*!: *nn-integral-suminf*) (*metis ki*(*1*) *assms*(*2*) *measurable-cong-sets measure-kernel.emeasure-measurable subprob-kernel-def that*)
also have ... = *?rhs*
unfolding *Mi-def*
proof(*rule suminf-ennreal-2dimen*[*symmetric*])
fix *m*
interpret *kim*: *subprob-kernel X Y ki m*
by(*simp add*: *ki*)
show (∫⁺ *x*. (*ki m x*) *A* ∂μ) = (∑ *n*. *emeasure* (*case* (*m*, *n*) *of* (*i*, *j*) ⇒ *measure-of* (*space Y*) (*sets Y*) (λ*A*. ∫⁺ *x*. *emeasure* (*ki i x*) *A* ∂μ*i j*)) *A*)
using *kim.emeasure-measurable*[*OF that*] **by**(*simp add*: *kim.nn-integral-measure*[*OF mui*(*1*) *that*] *nn-integral-measure-suminf*[*OF mui*(*1*)[*simplified assms*(*2*)[*symmetric*]] *mui*(*3*)])
qed
finally show *?thesis* **.**
qed
have *fin*:*finite-measure* (*Mi i*) **for** *i*
proof(*rule prod.exhaust*[**where** *y*=*prod-decode i*])
fix *j1 j2*
interpret *kij*: *subprob-kernel X Y ki j1*
by(*simp add*: *ki*)
assume *pd*:*prod-decode i* = (*j1*, *j2*)
have *Mi i* (*space* (*Mi i*)) = (∫⁺*x*. (*ki j1 x* (*space Y*)) ∂μ*i j2*)
by(*auto simp*: *Mi-def pd kij.nn-integral-measure*[*OF mui*(*1*) *sets.top*])
also have ... ≤ (∫⁺*x*. *1* ∂μ*i j2*)
by(*intro nn-integral-mono*) (*metis kij.subprob-space mui*(*1*) *sets-eq-imp-space-eq*)
also have ... ≤ *1*
using *mui* **by** (*simp add*: *sets-eq-imp-space-eq*[*OF mui*(*1*)])
finally show *finite-measure* (*Mi i*)
by (*metis ennreal-one-less-top finite-measureI infinity-ennreal-def less-le-not-le*)
qed
have *3*: *sets* (*Mi i*) = *sets* (μ ⤜ₖ κ) **for** *i*
by(*simp add*: *Mi-def split-beta sets-bind*)
show *s-finite-measure* (μ ⤜ₖ κ)
using *emeasure fin 3* **by** (*auto intro*!: *exI*[**where** *x*=*Mi*] *simp*: *s-finite-measure-def sets-bind*)
qed

end

lemma *s-finite-kernel-empty-trivial*:
assumes *space X = {}*
shows *s-finite-kernel X Y k*
using *assms* **by**(*auto simp*: *s-finite-kernel-def s-finite-kernel-axioms-def intro*!: *measure-kernel-empty-trivial finite-kernel-empty-trivial*)

lemma *s-finite-kernel-def'*: *s-finite-kernel X Y κ* ⟷ ((∀ *x*. *x* ∈ *space X* ⟶ *sets* (*κ x*) = *sets Y*) ∧ (∃ *ki*. (∀ *i*. *subprob-kernel X Y* (*ki i*)) ∧ (∀ *x A*. *x* ∈ *space X* ⟶ *A* ∈ *sets Y* ⟶ *emeasure* (*κ x*) *A* = (∑ *i*. *ki i x A*)))) (**is** *?l* ⟷ *?r*)
proof
assume *?l*
then interpret *s-finite-kernel X Y κ* **.**
from *s-finite-kernels* **obtain** *ki* **where** *ki*:⋀*i*. *subprob-kernel X Y* (*ki i*) ⋀*x A*. *x* ∈ *space X* ⟹ *emeasure* (*κ x*) *A* = (∑ *i*. *emeasure* (*ki i x*) *A*)
by *metis*
thus *?r*
by(*auto simp*: *kernel-sets*)
qed(*auto intro*!: *s-finite-kernel-subI*)

lemma(**in** *finite-kernel*) *s-finite-kernel-finite-kernel*: *s-finite-kernel X Y κ*
proof
consider *space X* = {} | *space X* ≠ {} **by** *auto*
then show ∃ *ki*. ∀ *i*. *finite-kernel X Y* (*ki i*) ∧ (∀ *x*∈*space X*. ∀ *A*∈*sets Y*. (*κ x*) *A* = (∑ *i*. (*ki i x*) *A*))
proof *cases*
case *1*
then show *?thesis*
by(*auto simp*: *finite-kernel-def measure-kernel-def finite-kernel-axioms-def measurable-def intro*!: *exI*[**where** *x=0*])
next
case *2*
then have *y*:*space Y* ≠ {} **by**(*simp add*: *Y-not-empty*)
define *ki* **where** *ki i* ≡ *case i of 0* ⇒ *κ* | *Suc* - ⇒ (λ-. *sigma* (*space Y*) (*sets Y*)) **for** *i*
have *finite-kernel X Y* (*ki i*) **for** *i*
by (*cases i*, *auto simp*: *ki-def finite-kernel-axioms*) (*auto simp*: *emeasure-sigma finite-kernel-def measure-kernel-def finite-kernel-axioms-def y intro*!: *finite-measureI exI*[**where** *x=1*])
moreover have (*κ x*) *A* = (∑ *i*. (*ki i x*) *A*) **for** *x A*
by(*simp add*: *suminf-offset*[**where** *i=Suc 0* **and** *f*=λ*i*. *ki i x A*,*simplified*],*simp add*: *ki-def emeasure-sigma*)
ultimately show *?thesis* **by** *auto*
qed
qed

lemmas(**in** *subprob-kernel*) *s-finite-kernel-subprob-kernel* = *s-finite-kernel-finite-kernel*
lemmas(**in** *prob-kernel*) *s-finite-kernel-prob-kernel* = *s-finite-kernel-subprob-kernel*

sublocale *finite-kernel* $\subseteq$ *s-finite-kernel*
by(*rule s-finite-kernel-finite-kernel*)

lemma *s-finite-kernel-cong-sets*:
assumes *sets X = sets X' sets Y = sets Y'*
shows *s-finite-kernel X Y = s-finite-kernel X' Y'*
by *standard* (*simp add*: *s-finite-kernel-def measurable-cong-sets*[*OF assms*(*1*) *refl*] *sets-eq-imp-space-eq*[*OF assms*(*1*)] *assms*(*2*) *measure-kernel-cong-sets*[*OF assms*] *s-finite-kernel-axioms-def finite-kernel-cong-sets*[*OF assms*])

lemma(**in** *s-finite-kernel*) *s-finite-kernel-cong*:
assumes $\bigwedge x.\ x \in$ *space X* $\Longrightarrow \kappa\ x = g\ x$
shows *s-finite-kernel X Y g*
using *assms s-finite-kernel-axioms* **by**(*auto simp*: *s-finite-kernel-def s-finite-kernel-axioms-def measure-kernel-def cong*: *measurable-cong*)

lemma(**in** *s-finite-measure*) *s-finite-kernel-const*:
assumes *space M* $\neq \{\}$
shows *s-finite-kernel X M* ($\lambda x.\ M$)
proof
obtain *Mi* **where** *Mi*:$\bigwedge i.$ *sets* (*Mi i*) = *sets M* $\bigwedge i.$ (*Mi i*) (*space M*) $\leq$ *1* $\bigwedge A.$ *M A* = ($\sum i.$ *Mi i A*)
by(*metis finite-measures*)
hence $\bigwedge i.$ *subprob-kernel X M* ($\lambda x.$ *Mi i*)
by(*auto simp*: *subprob-kernel-def' space-subprob-algebra sets-eq-imp-space-eq*[*OF Mi*(*1*)] *assms intro*!:*measurable-const subprob-spaceI*)
thus $\exists ki.\ \forall i.$ *finite-kernel X M* (*ki i*) $\wedge$ ($\forall x{\in}$*space X*. $\forall A{\in}$*sets M*. *M A* = ($\sum i.$ (*ki i x*) *A*))
by(*auto intro*!: *exI*[**where** *x*=$\lambda i\ x.$ *Mi i*] *Mi*(*3*) *subprob-kernel.finite-kernel*)
qed (*auto simp*: *assms*)

corollary(**in** *s-finite-measure*) *s-finite-kernel-const'*:
assumes *sets M = sets N space N* $\neq \{\}$
shows *s-finite-kernel X N* ($\lambda x.$ *M*)
using *assms*(*2*) *sets-eq-imp-space-eq*[*OF assms*(*1*)]
by(*auto simp*: *s-finite-kernel-cong-sets*[*OF refl assms*(*1*)[*symmetric*]] *intro*!: *s-finite-kernel-const*)

lemma *s-finite-kernel-pair-countble1*:
assumes *countable A* $\bigwedge i.\ i \in A \Longrightarrow$ *s-finite-kernel X Y* ($\lambda x.$ *k* (*i*,*x*))
shows *s-finite-kernel* (*count-space A* $\bigotimes_M$ *X*) *Y k*
proof −
have $\exists ki.$ ($\forall j.$ *subprob-kernel X Y* (*ki j*)) $\wedge$ ($\forall x\ B.\ x \in$ *space X* $\longrightarrow B \in$ *sets Y* $\longrightarrow$ *k* (*i*,*x*) *B* = ($\sum j.$ *ki j x B*)) **if** $i \in A$ **for** *i*
using *s-finite-kernel.s-finite-kernels*[*OF assms*(*2*)[*OF that*]] **by** *metis*
then obtain *ki* **where** *ki*:$\bigwedge i\ j.\ i \in A \Longrightarrow$ *subprob-kernel X Y* (*ki i j*) $\bigwedge i\ x\ B.\ i \in A \Longrightarrow x \in$ *space X* $\Longrightarrow B \in$ *sets Y* $\Longrightarrow$ *k* (*i*,*x*) *B* = ($\sum j.$ *ki i j x B*)
by *metis*
then show *?thesis*

using *assms(2)* **by**(*auto simp*: *s-finite-kernel-def' measure-kernel-pair-countble1*[*OF assms(1)*] *subprob-kernel-def' space-pair-measure intro*!: *exI*[**where** *x*=λ*j* (*i*,*x*). *ki i j x*] *measurable-pair-measure-countable1 assms(1)*)
qed

lemma *s-finite-kernel-s-finite-kernel*:
assumes ⋀*i*. *s-finite-kernel X Y* (*ki i*) ⋀*x*. *x* ∈ *space X* ⟹ *sets* (*k x*) = *sets Y* ⋀*x A*. *x* ∈ *space X* ⟹ *A* ∈ *sets Y* ⟹ *emeasure* (*k x*) *A* = (∑ *i*. (*ki i*) *x A*)
shows *s-finite-kernel X Y k*
proof −
have ∃ *kij*. (∀ *j*. *subprob-kernel X Y* (*kij j*)) ∧ (∀ *x A*. *x* ∈ *space X* ⟶ *ki i x A* = (∑ *j*. *kij j x A*)) **for** *i*
using *s-finite-kernel.s-finite-kernels*[*OF assms(1)*[*of i*]] **by** *metis*
then obtain *kij* **where** *kij*:⋀*i j*. *subprob-kernel X Y* (*kij i j*) ⋀*i x A*. *x* ∈ *space X* ⟹ *ki i x A* = (∑ *j*. *kij i j x A*)
by *metis*
define *ki'* **where** *ki'* ≡ (λ*n*. *case-prod kij* (*prod-decode n*))
have *emeasure-sumk'*:*emeasure* (*k x*) *A* = (∑ *i*. *emeasure* (*ki' i x*) *A*) **if** *x*:*x* ∈ *space X* **and** *A*: *A* ∈ *sets Y* **for** *x A*
by(*auto simp*: *assms(3)*[*OF that*] *kij(2)*[*OF x*] *ki'-def intro*!: *suminf-ennreal-2dimen*[*symmetric*])
have *subprob-kernel X Y* (*ki' i*) **for** *i*
using *kij(1)* **by**(*auto simp*: *ki'-def split-beta'*)
thus *?thesis*
by(*auto simp*: *s-finite-kernel-def' measure-kernel-def assms(2) s-finite-kernel-axioms-def emeasure-sumk' intro*!: *exI*[**where** *x*=*ki'*])
qed

lemma *s-finite-kernel-finite-sumI*:
assumes [*measurable-cong*]: ⋀*x*. *x* ∈ *space X* ⟹ *sets* (κ *x*) = *sets Y*
and ⋀*i*. *i* ∈ *I* ⟹ *subprob-kernel X Y* (*ki i*) ⋀*x A*. *x* ∈ *space X* ⟹ *A* ∈ *sets Y* ⟹ *emeasure* (κ *x*) *A* = (∑ *i*∈*I*. *ki i x A*) *finite I I* ≠ {}
shows *s-finite-kernel X Y* κ
proof −
consider *space X* = {} | *space X* ≠ {} **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by(*rule s-finite-kernel-empty-trivial*)
next
case *2*
then have *Y*:*space Y* ≠ {}
using *assms measure-kernel.Y-not-empty* **by** (*fastforce simp*: *subprob-kernel-def*)
define *ki'* **where** *ki'* ≡ (λ*i x*. *if i* < *card I then ki* (*from-nat-into I i*) *x else null-measure Y*)
have [*simp*]:*subprob-kernel X Y* (*ki' i*) **for** *i*
by(*cases i* < *card I*) (*simp add*: *ki'-def from-nat-into assms*, *auto simp*: *ki'-def subprob-kernel-def measure-kernel-def subprob-kernel-axioms-def Y intro*!: *subprob-spaceI*)

have [*simp*]: $(\sum i.\ emeasure\ (ki'\ i\ x)\ A) = (\sum i{\in}I.\ ki\ i\ x\ A)$ **for** $x\ A$
using *suminf-finite*[*of* $\{..<card\ I\}$ $\lambda i.\ (if\ i < card\ I\ then\ ki\ (from\text{-}nat\text{-}into\ I\ i)\ x\ else\ null\text{-}measure\ Y)\ A$]
by(*auto simp*: *sum.reindex-bij-betw*[*OF bij-betw-from-nat-into-finite*[*OF assms*(*4*)],*symmetric*] *ki'-def*)
have [*measurable*]:$B \in sets\ Y \Longrightarrow (\lambda x.\ emeasure\ (\kappa\ x)\ B) \in borel\text{-}measurable\ X$ **for** B
using *assms*(*2*) **by**(*auto simp*: *assms*(*3*) *subprob-kernel-def'* *cong*: *measurable-cong*)
show *?thesis*
by (*auto simp*: *s-finite-kernel-def'* *intro*!: *exI*[**where** $x{=}ki'$] *assms*)
qed
qed

Each kernel does not need to be bounded by a uniform upper-bound in the definition of *s-finite-kernel*

lemma *s-finite-kernel-finite-bounded-sum*:
assumes [*measurable-cong*]: $\bigwedge x.\ x \in space\ X \Longrightarrow sets\ (\kappa\ x) = sets\ Y$
and $\bigwedge i.\ measure\text{-}kernel\ X\ Y\ (ki\ i)$ $\bigwedge x\ A.\ x \in space\ X \Longrightarrow A \in sets\ Y \Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$ $\bigwedge i\ x.\ x \in space\ X \Longrightarrow ki\ i\ x\ (space\ Y) < \infty$
shows *s-finite-kernel* $X\ Y\ \kappa$
proof(*cases* $space\ X = \{\}$)
case *True*
then show *?thesis*
by(*simp add*: *s-finite-kernel-empty-trivial*)
next
case *X*:*False*
then have *Y*: $space\ Y \neq \{\}$
using *assms*(*2*)[*of 0*] **by**(*simp add*: *measure-kernel-def*)
show *?thesis*
proof(*rule s-finite-kernel-s-finite-kernel*[**where** *ki*=*ki*,*OF - assms*(*1*) *assms*(*3*)])
fix *i*
interpret *m*: *measure-kernel X Y ki i* **by** *fact*
define *kij* **where** $kij \equiv (\lambda(j :: nat)\ x.\ if\ j < nat\ \lceil enn2real\ (ki\ i\ x\ (space\ Y))\rceil\ then\ scale\text{-}measure\ (1\ /\ ennreal\ \lceil enn2real\ (ki\ i\ x\ (space\ Y))\rceil)\ (ki\ i\ x)\ else\ sigma\ (space\ Y)\ (sets\ Y))$
have *sets-kij*: $sets\ (kij\ j\ x) = sets\ Y$ **if** $x \in space\ X$ **for** $j\ x$
by(*auto simp*: *m.kernel-sets*[*OF that*] *kij-def*)
have *emeasure-kij*: $ki\ i\ x\ A = (\sum j.\ kij\ j\ x\ A)$ **if** $x \in space\ X\ A \in sets\ Y$ **for** $x\ A$
proof −
have $(\sum j.\ kij\ j\ x\ A) = (\sum j{<}\ nat\ \lceil enn2real\ (ki\ i\ x\ (space\ Y))\rceil.\ scale\text{-}measure\ (1\ /\ ennreal\ \lceil enn2real\ (ki\ i\ x\ (space\ Y))\rceil)\ (ki\ i\ x)\ A)$
by(*simp add*: *suminf-offset*[**where** $i{=}nat\ \lceil enn2real\ (ki\ i\ x\ (space\ Y))\rceil$ **and** $f{=}\lambda j.\ kij\ j\ x\ A$], *simp add*: *kij-def emeasure-sigma*)
also have ... = *ki i x A*
proof(*cases nat* $\lceil enn2real\ (ki\ i\ x\ (space\ Y))\rceil$)
case *0*
then show *?thesis*

by *simp* (*metis assms*(*4*) *emeasure-eq-0 enn2real-le ennreal-0 infinity-ennreal-def le-zero-eq linorder-not-le m.kernel-space nle-le sets.sets-into-space sets.top that*)
next
case (*Suc n'*)
then have *ennreal* (*real-of-int* ⌈*enn2real* (*emeasure* (*ki i x*) (*space Y*))⌉) > *0*
using *ennreal-less-zero-iff* **by** *fastforce*
with *assms*(*4*)[*OF that*(*1*),*of i*] **have** [*simp*]: *of-nat* (*nat* ⌈*enn2real* (*emeasure* (*ki i x*) (*space Y*))⌉) / *ennreal* (*real-of-int* ⌈*enn2real* (*emeasure* (*ki i x*) (*space Y*))⌉) = *1*
by (*simp add*: *ennreal-eq-0-iff ennreal-of-nat-eq-real-of-nat*)
show *?thesis*
by(*simp add*: *mult.assoc*[*symmetric*] *ennreal-times-divide*)
qed
finally show *?thesis* **by** *simp*
qed
have *sk*: *subprob-kernel X Y* (*kij j*) **for** *j*
proof −
{
fix *B*
assume [*measurable*]:*B* ∈ *sets Y*
have *emeasure* (*kij j x*) *B* = (*if j* < *nat* ⌈*enn2real* (*ki i x* (*space Y*))⌉ *then* (*ki i x*) *B* / *ennreal* (*real-of-int* ⌈*enn2real* (*ki i x* (*space Y*))⌉) *else 0*) **if** *x* ∈ *space X* **for** *x*
by(*auto simp*: *kij-def emeasure-sigma divide-ennreal-def mult.commute*)
hence (λ*x*. *emeasure* (*kij j x*) *B*) ∈ *borel-measurable X*
by(*auto simp*: *kij-def cong*: *measurable-cong*)
}
moreover {
fix *x*
assume *x*:*x* ∈ *space X*
have *subprob-space* (*kij j x*)
proof −
have *emeasure* (*kij j x*) (*space Y*) ≤ *1*
proof −
{
assume *1*:*j* < *nat* ⌈*enn2real* (*emeasure* (*ki i x*) (*space Y*))⌉
then have *emeasure* (*ki i x*) (*space Y*) > *0*
by (*metis ceiling-zero enn2real-0 nat-zero-as-int not-gr-zero not-less-zero*)
with *assms*(*4*)[*OF x*] **have** [*simp*]:*emeasure* (*ki i x*) (*space Y*) / *emeasure* (*ki i x*) (*space Y*) = *1*
by *simp*
have [*simp*]:*emeasure* (*ki i x*) (*space Y*) / *ennreal* (*real-of-int* ⌈*enn2real* (*ki i x* (*space Y*))⌉) ≤ *1*
proof(*rule order.trans*[**where** *b*=*emeasure* (*ki i x*) (*space Y*) / *ki i x* (*space Y*),*OF divide-le-posI-ennreal*])
show *0* < *ennreal* (*real-of-int* ⌈*enn2real* (*ki i x* (*space Y*))⌉)
using *1 assms*(*4*)[*OF x*] *enn2real-positive-iff top.not-eq-extremum*

by *fastforce*
next
have *1*:*ennreal* (*real-of-int* ⌈*enn2real* (*ki i x* (*space Y*))⌉) ≥ *ki i x* (*space Y*)
using *assms*(*4*)[*OF x*] *enn2real-le* **by** (*simp add*: *linorder-neq-iff*)
have *ennreal* (*real-of-int* ⌈*enn2real* (*ki i x* (*space Y*))⌉) / *ki i x* (*space Y*) ≥ *1*
by(*rule order.trans*[*OF - divide-right-mono-ennreal*[*OF 1*,*of ki i x* (*space Y*)]]) *simp*
thus *emeasure* (*ki i x*) (*space Y*) ≤ *ennreal* (*real-of-int* ⌈*enn2real* (*ki i x* (*space Y*))⌉) ∗ (*emeasure* (*ki i x*) (*space Y*) / *ki i x* (*space Y*))
by (*simp add*: *1*)
qed *simp*
have *1* / *ennreal* (*real-of-int* ⌈*enn2real* (*emeasure* (*ki i x*) (*space Y*))⌉) ∗ *emeasure* (*ki i x*) (*space Y*) ≤ *1*
by (*simp add*: *ennreal-divide-times*)
}
thus *?thesis*
by(*auto simp*: *kij-def emeasure-sigma*)
qed
thus *?thesis*
by(*auto intro*!: *subprob-spaceI simp*: *sets-eq-imp-space-eq*[*OF sets-kij*[*OF x*,*of j*]] *Y*)
qed
}
ultimately show *?thesis*
by(*auto simp*: *subprob-kernel-def measure-kernel-def sets-kij m.Y-not-empty subprob-kernel-axioms-def*)
qed
show *s-finite-kernel X Y* (*ki i*)
by(*auto intro*!: *s-finite-kernel-subI simp*: *emeasure-kij sk m.kernel-sets*)
qed *simp-all*
qed

lemma(**in** *measure-kernel*) *s-finite-kernel-finite-bounded*:
assumes ⋀*x*. *x* ∈ *space X* ⟹ *κ x* (*space Y*) < ∞
shows *s-finite-kernel X Y κ*
proof(*cases space X* = {})
case *True*
then show *?thesis*
by(*simp add*: *s-finite-kernel-empty-trivial*)
next
case *False*
then have *Y*:*space Y* ≠ {} **by**(*simp add*: *Y-not-empty*)
have *measure-kernel X Y* (*case i of 0* ⇒ *κ* | *Suc x* ⇒ λ*x*. *null-measure Y*) **for** *i*
by(*cases i*,*auto simp*: *measure-kernel-axioms*) (*auto simp*: *measure-kernel-def Y*)
moreover have *κ x A* = (∑ *i*. *emeasure* ((*case i of 0* ⇒ *κ* | *Suc x* ⇒ λ*x*. *null-measure Y*) *x*) *A*) **for** *x A*

by(*simp add*: *suminf-offset*[**where** *i*=*Suc 0*])
moreover have $x \in space\ X \Longrightarrow emeasure\ ((case\ i\ of\ 0 \Rightarrow \kappa \mid Suc\ x \Rightarrow \lambda x.\ null\text{-}measure\ Y)\ x)\ (space\ Y) < \top$ **for** *x i*
by(*cases i*) (*use assms* **in** *auto*)
ultimately show *?thesis*
by(*auto intro*!: *s-finite-kernel-finite-bounded-sum*[**where** *ki*=$\lambda i.\ case\ i\ of\ 0 \Rightarrow \kappa \mid Suc\ \text{-} \Rightarrow (\lambda x.\ null\text{-}measure\ Y)$ **and** *X*=*X* **and** *Y*=*Y*] *simp*: *kernel-sets*)
qed

lemma(**in** *s-finite-kernel*) *density-s-finite-kernel*:
assumes *f*[*measurable*]: *case-prod f* $\in X \bigotimes_M Y \to_M$ *borel*
shows *s-finite-kernel X Y* ($\lambda x.\ density\ (\kappa\ x)\ (f\ x)$)
proof(*cases space X* = {})
case *True*
then show *?thesis*
by(*simp add*: *s-finite-kernel-empty-trivial*)
next
case *False*
note *Y* = *Y-not-empty*[*OF this*]
obtain *ki'* **where** *ki'*: $\bigwedge i.\ subprob\text{-}kernel\ X\ Y\ (ki'\ i)\ \bigwedge x\ A.\ x \in space\ X \Longrightarrow \kappa\ x\ A = (\sum i.\ ki'\ i\ x\ A)$
by(*metis s-finite-kernels*)
hence *sets-ki'*[*measurable-cong*]:$\bigwedge x\ i.\ x \in space\ X \Longrightarrow sets\ (ki'\ i\ x) = sets\ Y$
by(*auto simp*: *subprob-kernel-def measure-kernel-def*)
define *ki* **where** *ki* ≡ ($\lambda i\ x.\ density\ (ki'\ i\ x)\ (f\ x)$)
have *sets-ki*: *x* ∈ *space X* ⟹ *sets* (*ki i x*) = *sets Y* **for** *i x*
using *ki'*(*1*) **by**(*auto simp*: *subprob-kernel-def measure-kernel-def ki-def*)
have *emeasure-k*:*density* (*κ x*) (*f x*) *A* = ($\sum i.\ ki\ i\ x\ A$) **if** *x*:*x* ∈ *space X* **and** *A*[*measurable*]:*A* ∈ *sets Y* **for** *x A*
using *kernel-sets*[*OF x*] *x ki'*(*1*) *sets-ki'*[*OF x*] **by**(*auto simp*: *emeasure-density nn-integral-measure-suminf*[*OF - ki'*(*2*),*of x*] *ki-def*)
show *?thesis*
proof(*rule s-finite-kernel-s-finite-kernel*[**where** *ki*=*ki*,*OF - - emeasure-k*])
fix *i*
note *nn-integral-measurable-subprob-algebra2*[*OF - ki'*(*1*)[*of i*,*simplified sub-prob-kernel-def'*],*measurable*]
define *kij* **where** *kij* ≡ ($\lambda j\ x.\ if\ j = 0\ then\ density\ (ki'\ i\ x)\ (\lambda y.\ \infty * indicator\ \{y \in space\ Y.\ f\ x\ y = \infty\}\ y)$
else if j = (*Suc 0*) *then density* (*ki' i x*) ($\lambda y.\ f\ x\ y * indicator\ \{y \in space\ Y.\ f\ x\ y < \infty\}\ y$)
else null-measure Y)
have *emeasure-kij*: *ki i x A* = ($\sum j.\ kij\ j\ x\ A$) (**is** *?lhs* = *?rhs*) **if** *x*:*x* ∈ *space X* **and** [*measurable*]: *A* ∈ *sets Y* **for** *x A*
proof −
have *?lhs* = ($\int^+ y \in A.\ f\ x\ y\ \partial ki'\ i\ x$)
using *sets-ki*[*OF x*,*of i*] *x* **by**(*auto simp*: *ki-def emeasure-density*)
also have *...* = ($\int^+ y.\ (\infty * indicator\ \{y \in space\ Y.\ f\ x\ y = \infty\}\ y * indicator\ A\ y + f\ x\ y * indicator\ \{y \in space\ Y.\ f\ x\ y < \infty\}\ y * indicator\ A\ y)\ \partial ki'\ i\ x$)
by(*auto intro*!: *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF sets-ki'*[*OF*

```
x]] indicator-def) (simp add: top.not-eq-extremum)
      also have ... = density (ki' i x) (λy. ∞ * indicator {y∈space Y. f x y = ∞}
y) A + density (ki' i x) (λy. f x y * indicator {y∈space Y. f x y < ∞} y) A
      using sets-ki[OF x,of i] x by(auto simp: ki-def emeasure-density nn-integral-add)
      also have ... = ?rhs
       using suminf-finite[of {..<Suc (Suc 0)} λj. kij j x A] by(simp add: kij-def)
      finally show ?thesis .
    qed
    have sets-kij[measurable-cong]:x ∈ space X ⟹ sets (kij j x) = sets Y for j x
      by(auto simp: kij-def sets-ki')
    show s-finite-kernel X Y (ki i)
    proof(rule s-finite-kernel-s-finite-kernel[where ki=kij,OF - - emeasure-kij])
      fix j
      consider j = 0 | j = Suc 0 | j ≠ 0 j ≠ Suc 0 by auto
      then show s-finite-kernel X Y (kij j)
      proof cases
        case 1
        have emeasure-ki: emeasure (kij j x) A = (∑ j. emeasure (density (ki' i x)
(indicator {y ∈ space Y. f x y = ⊤})) A) if x:x ∈ space X and [measurable]: A ∈
sets Y for x A
         using sets-ki'[OF x] x by(auto simp: 1 kij-def emeasure-density nn-integral-suminf[symmetric]
indicator-def intro!: nn-integral-cong) (simp add: nn-integral-count-space-nat[symmetric])
        have [simp]:subprob-kernel X Y (λx. density (ki' i x) (indicator {y ∈ space
Y. f x y = ⊤}))
        proof −
          have [simp]:x ∈ space X ⟹ set-nn-integral (ki' i x) (space Y) (indicator
{y ∈ space Y. f x y = ⊤}) ≤ 1 for x
          by(rule order.trans[OF nn-integral-mono[where v=λx. 1]],insert ki'(1)[of
i]) (auto simp: indicator-def subprob-kernel-def subprob-kernel-axioms-def intro!:
subprob-space.emeasure-space-le-1)
          show ?thesis
             by(auto simp: subprob-kernel-def measure-kernel-def emeasure-density
subprob-kernel-axioms-def sets-ki' sets-eq-imp-space-eq[OF sets-ki'] Y cong: mea-
surable-cong intro!: subprob-spaceI)
        qed
        show ?thesis
        by (auto simp: s-finite-kernel-def' sets-kij intro!: exI[where x=λk x. density
(ki' i x) (indicator {y ∈ space Y. f x y = ⊤})] simp: emeasure-ki )
      next
        case j:2
        have emeasure-ki: emeasure (kij j x) A = (∑ k. density (ki' i x) (λy. f x y
* indicator {y ∈ space Y. of-nat k ≤ f x y ∧ f x y < 1 + of-nat k} y) A) if x:x ∈
space X and [measurable]:A ∈ sets Y for x A
        proof −
          have [simp]: f x y * indicator {y ∈ space Y. f x y < ⊤} y * indicator A y
= f x y * (∑ k. indicator {y ∈ space Y. of-nat k ≤ f x y ∧ f x y < 1 + of-nat k}
y) * indicator A y if y:y ∈ space Y for y
          proof(cases f x y < ⊤)
            case f:True
```

define l **where** $l \equiv$ *floor* (*enn2real* ($f\ x\ y$))
have *nat* $l \leq$ *enn2real* ($f\ x\ y$) *enn2real* ($f\ x\ y$) $<$ 1 + *nat* l
by (*simp-all add*: *l-def*) *linarith*
with y **have** *l*:*of-nat* (*nat* l) $\leq f\ x\ y\ f\ x\ y <$ 1 + *of-nat* (*nat* l)
using *Orderings.order-eq-iff enn2real-positive-iff ennreal-enn2real-if ennreal-of-nat-eq-real-of-nat linorder-not-le of-nat-0-le-iff f* **by** *fastforce*+
then have ($\sum j$. *indicator* $\{y \in$ *space* Y. *of-nat* $j \leq f\ x\ y \wedge f\ x\ y <$ 1 + *of-nat* $j\}$ y :: *ennreal*) = ($\sum j$. *if* j = *nat* l *then* 1 *else* 0)
by(*auto intro*!: *suminf-cong simp*: *indicator-def* y) (*metis Suc-leI linorder-neqE-nat linorder-not-less of-nat-Suc of-nat-le-iff order-trans*)
also have ... = 1
using *suminf-finite*[**where** N={*nat* l} **and** f=λj. *if* j = *nat* l *then* 1 *else* (0 :: *ennreal*)] **by** *simp*
finally show *?thesis*
by(*auto*, *insert f*) (*auto simp*: *indicator-def*)
qed(*use top.not-eq-extremum* **in** *fastforce*)
show *?thesis*
using *sets-ki*[*OF x*] *sets-ki'*[*OF x*] x **by**(*auto simp*: *kij-def j emeasure-density nn-integral-suminf*[*symmetric*] *sets-eq-imp-space-eq*[*OF sets-ki'*[*OF x*]] *intro*!: *nn-integral-cong*)
qed
show *?thesis*
proof(*rule s-finite-kernel-finite-bounded-sum*[*OF sets-kij - emeasure-ki*])
fix k
show *measure-kernel X Y* (λx. *density* (*ki'* $i\ x$) (λy. $f\ x\ y$ * *indicator* $\{y \in$ *space* Y. *of-nat* $k \leq f\ x\ y \wedge f\ x\ y <$ 1 + *of-nat* $k\}$ y))
using *sets-ki'*[*of - i*] **by**(*auto simp*: *measure-kernel-def emeasure-density Y cong*: *measurable-cong*)
next
fix $k\ x$
assume x:$x \in$*space X*
have *emeasure* (*density* (*ki'* $i\ x$) (λy. $f\ x\ y$ * *indicator* $\{y \in$ *space* Y. *of-nat* $k \leq f\ x\ y \wedge f\ x\ y <$ 1 + *of-nat* $k\}$ y)) (*space* Y) $\leq$ 1 + *of-nat* k
by(*auto simp*: *emeasure-density x*,*rule order.trans*[*OF nn-integral-mono*[**where** v=λx. 1 + *of-nat* k]]) (*insert subprob-kernel.subprob-space*[*OF ki'*(1)[*of i*] x], *auto simp*: *indicator-def subprob-kernel-def subprob-kernel-axioms-def sets-eq-imp-space-eq*[*OF sets-ki'*[*OF x*]] *intro*!: *mult-mono*[**where** *d*=*1* :: *ennreal*,*OF order.refl*,*simplified*])
also have ... $< \infty$
by (*simp add*: *of-nat-less-top*)
finally show *emeasure* (*density* (*ki'* $i\ x$) (λy. $f\ x\ y$ * *indicator* $\{y \in$ *space* Y. *of-nat* $k \leq f\ x\ y \wedge f\ x\ y <$ 1 + *of-nat* $k\}$ y)) (*space* Y) $< \infty$.
qed *auto*
next
case *3*
then show *?thesis*
by(*auto simp*: *kij-def s-finite-kernel-cong-sets*[*of X X Y*,*OF - sets-null-measure*[*symmetric*]] *Y intro*!: *s-finite-measure.s-finite-kernel-const finite-measure.s-finite-measure-finite-measure finite-measureI*)
qed

qed(*auto simp*: *sets-ki*)
qed(*auto simp*: *kernel-sets*)
qed

lemma(**in** *s-finite-kernel*) *nn-integral-measurable-f*:
assumes [*measurable*]:(λ(*x*,*y*). *f x y*) ∈ *borel-measurable* (*X* ⨂$_M$ *Y*)
shows (λx. $\int^+ y$. *f x y* ∂(κ *x*)) ∈ *borel-measurable X*
proof −
obtain κi **where** κi:⋀*i*. *subprob-kernel X Y* (κi *i*) ⋀*x A*. *x* ∈ *space X* ⟹ κ *x A* = (∑ *i*. κi *i x A*)
by(*metis s-finite-kernels*)
show *?thesis*
proof(*rule measurable-cong*[*THEN iffD2*])
fix *x*
assume *x* ∈ *space X*
with κi **show** ($\int^+$ *y*. *f x y* ∂κ *x*) = (∑ *i*. $\int^+$ *y*. *f x y* ∂κi *i x*)
by(*auto intro*!: *nn-integral-measure-suminf*[*symmetric*] *simp*: *subprob-kernel-def kernel-sets measure-kernel-def*)
next
show (λx. ∑ *i*. $\int^+$ *y*. *f x y* ∂κi *i x*) ∈ *borel-measurable X*
using κi(*1*) *nn-integral-measurable-subprob-algebra2*[*OF assms*] **by**(*simp add*: *subprob-kernel-def'*)
qed
qed

lemma(**in** *s-finite-kernel*) *nn-integral-measurable-f'*:
assumes *f* ∈ *borel-measurable* (*X* ⨂$_M$ *Y*)
shows (λx. $\int^+ y$. *f* (*x*, *y*) ∂(κ *x*)) ∈ *borel-measurable X*
using *nn-integral-measurable-f*[**where** *f*=*curry f*,*simplified*,*OF assms*] **by** *simp*

lemma(**in** *s-finite-kernel*) *bind-kernel-s-finite-kernel'*:
assumes *s-finite-kernel* (*X* ⨂$_M$ *Y*) *Z* (*case-prod g*)
shows *s-finite-kernel X Z* (λx. κ *x* ⤜$_k$ *g x*)
proof(*cases space X* = {})
case *True*
then show *?thesis*
by (*simp add*: *s-finite-kernel-empty-trivial*)
next
case *X*:*False*
then have *Y*:*space Y* ≠ {}
by(*simp add*: *Y-not-empty*)
from *s-finite-kernels* **obtain** *ki* **where** *ki*:
⋀*i*. *subprob-kernel X Y* (*ki i*) ⋀*x A*. *x* ∈ *space X* ⟹ κ *x A* = (∑ *i*. *ki i x A*)
by *metis*
interpret *g*:*s-finite-kernel X* ⨂$_M$ *Y Z case-prod g* **by** *fact*
from *g.s-finite-kernels*[*simplified space-pair-measure*] **obtain** *gi* **where** *gi*:
⋀*i*. *subprob-kernel* (*X* ⨂$_M$ *Y*) *Z* (*gi i*) ⋀*x y A*. *x* ∈ *space X* ⟹ *y* ∈ *space Y* ⟹ *g x y A* = (∑ *i*. *gi i* (*x*,*y*) *A*)
by *auto metis*

define *kgi* **where** *kgi* = (λ*i x*. *case prod-decode i of* (*i*,*j*) ⇒ (*ki j x* ⤜ *curry* (*gi i*) *x*))
have *emeasure*:*emeasure* (*κ x* ⤜$_k$ *g x*) *A* = (∑ *i*. *emeasure* (*kgi i x*) *A*) (**is** *?lhs* = *?rhs*) **if** *x*:*x* ∈ *space X* **and** *A*:*A* ∈ *sets Z* **for** *x A*
proof −
interpret *gx*: *s-finite-kernel Y Z g x*
using *g.comp-measurable*[*OF measurable-Pair1′*[*OF x*]] **by** *auto*
have *?lhs* = (∫$^+$ *y*. *g x y A* ∂*κ x*)
using *gx.emeasure-bind-kernel*[*OF kernel-sets*[*OF x*] *A*]
by(*auto simp*: *sets-eq-imp-space-eq*[*OF kernel-sets*[*OF x*]] *Y*)
also have ... = (∫$^+$ *y*. (∑ *i*. *gi i* (*x*, *y*) *A*) ∂*κ x*)
by(*auto intro*!: *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF kernel-sets*[*OF x*]] *gi*(*2*)[*OF x*])
also have ... = (∑ *i*. ∫$^+$ *y*. *gi i* (*x*, *y*) *A* ∂*κ x*)
using *gi*(*1*) *x A* **by**(*auto intro*!: *nn-integral-suminf simp*: *subprob-kernel-def′*)
also have ... = (∑ *i*. (∑ *j*. ∫$^+$ *y*. *gi i* (*x*, *y*) *A* ∂*ki j x*))
by(*rule suminf-cong*, *rule nn-integral-measure-suminf*[*symmetric*], *insert kernel-sets*[*OF x*] *ki gi*(*1*) *x A*)
(*auto simp*: *subprob-kernel-def measure-kernel-def measurable-cong-sets*[*OF sets-pair-measure-cong*[*OF refl kernel-sets*[*OF x*]]] *intro*!: *measurable-Pair2*[*OF - x*])
also have ... = (∑ *i*. (∑ *j*. *emeasure* (*ki j x* ⤜ (*curry* (*gi i*) *x*)) *A*))
using *sets-eq-imp-space-eq*[*of ki - x Y*] *ki*(*1*) *x gi*(*1*) *measurable-cong-sets*[*of - - subprob-algebra Z subprob-algebra Z*, *OF sets-pair-measure-cong*[*of X X Y ki - x*]]
by(*auto intro*!: *suminf-cong emeasure-bind*[*OF - - A*,*symmetric*] *measurable-Pair2*[*OF - x*] *simp*: *curry-def subprob-kernel-def*[*of X*] *subprob-kernel-def′*[*of X* ⨂$_M$ *Y*] *measure-kernel-def Y*)
also have ... = *?rhs*
unfolding *kgi-def* **by**(*rule suminf-ennreal-2dimen*[*symmetric*]) (*simp add*: *curry-def*)
finally show *?thesis* **.**
qed
have *sets*: *sets* (*κ x* ⤜$_k$ *g x*) = *sets Z* **if** *x*:*x* ∈ *space X* **for** *x*
proof −
interpret *gx*: *s-finite-kernel Y Z g x*
using *g.comp-measurable*[*OF measurable-Pair1′*[*OF x*]] **by** *auto*
show *?thesis*
by(*simp add*: *gx.sets-bind-kernel*[*OF - kernel-sets*[*OF x*]] *Y*)
qed
have *sk*:*subprob-kernel X Z* (*kgi i*) **for** *i*
using *ki*(*1*)[*of snd* (*prod-decode i*)] *gi*(*1*)[*of fst* (*prod-decode i*)]
by(*auto simp*: *subprob-kernel-def′ kgi-def split-beta′ curry-def*)
show *?thesis*
using *sk* **by**(*auto simp*: *s-finite-kernel-def′ emeasure sets subprob-kernel-def′ intro*!: *exI*[**where** *x*=*kgi*] *measurable-cong*[**where** *g*=λ*x*. ∑ *i*. *emeasure* (*kgi i x*) - **and** *f*=λ*x*. *emeasure* (*κ x* ⤜$_k$ *g x*) -,*THEN iffD2*])
qed

corollary(**in** *s-finite-kernel*) *bind-kernel-s-finite-kernel*:
assumes *s-finite-kernel Y Z k′*
shows *s-finite-kernel X Z* ($\lambda x.\ \kappa\ x \gg\!=_k\ k'$)
by(*auto intro*!: *bind-kernel-s-finite-kernel′ s-finite-kernel.comp-measurable*[*OF assms measurable-snd*] *simp*: *split-beta′*)

lemma(**in** *s-finite-kernel*) *bind-kernel-assoc*:
assumes *s-finite-kernel Y Z k′ sets μ* = *sets X*
shows $\mu \gg\!=_k (\lambda x.\ \kappa\ x \gg\!=_k k') = \mu \gg\!=_k \kappa \gg\!=_k k'$
proof(*cases space X* = {})
case *X*:*False*
then have *μ*: *space μ* ≠ {} **and** *Y*:*space Y* ≠ {}
by(*simp-all add*: *Y-not-empty sets-eq-imp-space-eq*[*OF assms*(*2*)])
interpret *k′*:*s-finite-kernel Y Z k′* **by** *fact*
interpret *k″*: *s-finite-kernel X Z λx. κ x* $\gg\!=_k$ *k′*
by(*rule bind-kernel-s-finite-kernel*[*OF assms*(*1*)])
show *?thesis*
proof(*rule measure-eqI*)
fix *A*
assume *A* ∈ *sets* ($\mu \gg\!=_k (\lambda x.\ \kappa\ x \gg\!=_k k')$)
then have *A*[*measurable*]: *A* ∈ *sets Z*
by(*simp add*: *k″.sets-bind-kernel*[*OF X assms*(*2*)])
show *emeasure* ($\mu \gg\!=_k (\lambda x.\ \kappa\ x \gg\!=_k k')$) *A* = *emeasure* ($\mu \gg\!=_k \kappa \gg\!=_k k'$) *A*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\int^+ x.$ *emeasure* ($\kappa\ x \gg\!=_k k'$) *A* $\partial\mu$)
by(*rule k″.emeasure-bind-kernel*[*OF assms*(*2*) *A*])
also have ... = ($\int^+ x. \int^+ y.$ *k′ y A* $\partial\kappa\ x\ \partial\mu$)
using *k′.emeasure-bind-kernel*[*OF kernel-sets A*]
by(*auto intro*!: *nn-integral-cong simp*: *sets-eq-imp-space-eq*[*OF assms*(*2*)] *sets-eq-imp-space-eq*[*OF kernel-sets*] *Y*)
also have ... = ($\int^+ y.$ *k′ y A* $\partial(\mu \gg\!=_k \kappa)$)
by(*simp add*: *nn-integral-bind-kernel*[*OF k′.emeasure-measurable*[*OF A*] *assms*(*2*)])
also have ... = *?rhs*
by(*simp add*: *k′.emeasure-bind-kernel*[*OF sets-bind-kernel*[*OF X assms*(*2*)] *A*] *sets-eq-imp-space-eq*[*OF sets-bind-kernel*[*OF X assms*(*2*)]] *Y*)
finally show *?thesis* **.**
qed
qed(*auto simp*: *k′.sets-bind-kernel*[*OF Y sets-bind-kernel*[*OF X assms*(*2*)]] *k″.sets-bind-kernel*[*OF X assms*(*2*)])
qed(*simp add*: *bind-kernel-def sets-eq-imp-space-eq*[*OF assms*(*2*)])

lemma *s-finite-kernel-pair-measure*:
assumes *s-finite-kernel X Y k* **and** *s-finite-kernel X Z k′*
shows *s-finite-kernel X* (*Y* $\bigotimes_M$ *Z*) ($\lambda x.\ k\ x \bigotimes_M k'\ x$)
proof −
interpret *k*: *s-finite-kernel X Y k* **by** *fact*
interpret *k′*: *s-finite-kernel X Z k′* **by** *fact*

from *k.s-finite-kernels k'.s-finite-kernels* **obtain** *ki ki'*
where *ki*:$\bigwedge i$. *subprob-kernel X Y* (*ki i*) $\bigwedge x$ *A*. *x* ∈ *space X* ⟹ *k x A* = ($\sum i$. *ki i x A*)
and *ki'*:$\bigwedge i$. *subprob-kernel X Z* (*ki' i*) $\bigwedge x$ *A*. *x* ∈ *space X* ⟹ *k' x A* = ($\sum i$. *ki' i x A*)
by *metis*
then have *1*[*measurable-cong*]: $\bigwedge x$ *i*. *x* ∈ *space X* ⟹ *sets* (*ki i x*) = *sets Y* $\bigwedge x$ *i*. *x* ∈ *space X* ⟹ *sets* (*ki' i x*) = *sets Z*
by(*auto simp*: *subprob-kernel-def measure-kernel-def*)
define *kki* **where** *kki* ≡ (λ*i x*. (λ(*j*,*i*). *ki i x* ⨂$_M$ *ki' j x*) (*prod-decode i*))
have *kki1*: $\bigwedge i$. *subprob-kernel X* (*Y* ⨂$_M$ *Z*) (*kki i*)
using *ki*(*1*) *ki'*(*1*) **by**(*auto simp*: *subprob-kernel-def' kki-def split-beta intro*!: *measurable-pair-measure*)
have *kki2*: (*k x* ⨂$_M$ *k' x*) *A* = ($\sum i$. (*kki i x*) *A*) (**is** *?lhs* = *?rhs*) **if** *x*:*x* ∈ *space X* **and** *A*[*measurable*]: *A* ∈ *sets* (*Y* ⨂$_M$ *Z*) **for** *x A*
proof −
have *?lhs* = (∫$^+$ *y*. ∫$^+$ *z*. *indicator A* (*y*, *z*) ∂*k' x* ∂*k x*)
using *x* **by**(*simp add*: *s-finite-measure.emeasure-pair-measure'*[*OF k'.image-s-finite-measure*])
also have ... = (∫$^+$ *y*. ($\sum j$. ∫$^+$ *z*. *indicator A* (*y*, *z*) ∂*ki' j x*) ∂*k x*)
using *ki' x* **by**(*auto intro*!: *nn-integral-cong nn-integral-measure-suminf*[*symmetric*] *simp*: *sets-eq-imp-space-eq*[*OF k.kernel-sets*[*OF x*]] *subprob-kernel-def measure-kernel-def k'.kernel-sets*)
also have ... = ($\sum j$. ∫$^+$ *y*. ∫$^+$ *z*. *indicator A* (*y*, *z*) ∂*ki' j x* ∂*k x*)
using *x* **by**(*auto intro*!: *nn-integral-suminf s-finite-measure.borel-measurable-nn-integral-fst' s-finite-kernel.image-s-finite-measure*[*OF subprob-kernel.s-finite-kernel-subprob-kernel*[*OF ki'*(*1*)]])
also have ... = ($\sum j$. ($\sum i$. (∫$^+$ *y*. ∫$^+$ *z*. *indicator A* (*y*, *z*) ∂*ki' j x* ∂*ki i x*)))
using *x ki* **by**(*auto intro*!: *suminf-cong nn-integral-measure-suminf*[*symmetric*] *s-finite-measure.borel-measurable-nn-integral-fst' simp*: *k.kernel-sets*[*OF x*] *subprob-kernel-def measure-kernel-def s-finite-kernel.image-s-finite-measure*[*OF subprob-kernel.s-finite-kernel-subprob-kernel*[*OF ki'*(*1*)]])
also have ... = ($\sum j$. ($\sum i$. (*ki i x* ⨂$_M$ *ki' j x*) *A*))
using *x* **by**(*auto simp*: *s-finite-measure.emeasure-pair-measure'*[*OF s-finite-kernel.image-s-finite-measure*[*O subprob-kernel.s-finite-kernel-subprob-kernel*[*OF ki'*(*1*)]]])
also have ... = *?rhs*
unfolding *kki-def* **by**(*rule suminf-ennreal-2dimen*[*symmetric*]) *auto*
finally show *?thesis* **.**
qed
show *?thesis*
proof
fix *B*
assume [*measurable*]:*B* ∈ *sets* (*Y* ⨂$_M$ *Z*)
show (λ*x*. *emeasure* (*k x* ⨂$_M$ *k' x*) *B*) ∈ *borel-measurable X*
by(*rule measurable-cong*[**where** *g*=λ*x*. $\sum i$. (*kki i x*) *B*,*THEN iffD2*], *insert kki1*) (*auto simp*: *subprob-kernel-def' kki2*)
qed(*auto intro*!: *exI*[**where** *x*=*kki*] *simp*: *subprob-kernel.finite-kernel kki1 kki2 k.kernel-sets k'.kernel-sets space-pair-measure k.Y-not-empty k'.Y-not-empty*)
qed

lemma *pair-measure-eq-bind-s-finite*:
assumes *s-finite-measure* μ *s-finite-measure* ν
shows $\mu \bigotimes_M \nu = \mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,y)))$
proof −
consider *space* $\mu = \{\}$ | *space* $\nu = \{\}$ | *space* $\mu \neq \{\}$ *space* $\nu \neq \{\}$
by *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by(*auto simp*: *bind-kernel-def space-pair-measure intro*!: *space-empty*)
next
case *2*
then have $\mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y))) = count\text{-}space\ \{\}$
by(*auto simp*: *bind-kernel-def space-empty*)
with *2* **show** *?thesis*
by(*auto simp*: *space-pair-measure intro*!: *space-empty*)
next
case *3*
show *?thesis*
proof(*intro measure-eqI sets-bind-kernel*[*OF - 3*(*1*),*symmetric*] *sets-bind-kernel*[*OF - 3*(*2*)])
fix *A*
assume *A*[*measurable*]: $A \in sets\ (\mu \bigotimes_M \nu)$
show *emeasure* $(\mu \bigotimes_M \nu)\ A = emeasure\ (\mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y))))\ A$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+ x.\ \int^+ y.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y)\ A\ \partial\nu\ \partial\mu)$
by(*simp add*: *s-finite-measure.emeasure-pair-measure'*[*OF assms*(*2*)])
also have ... = $(\int^+ x.\ (\nu \gg\!=_k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,y)))\ A\ \partial\mu)$
by(*auto intro*!: *nn-integral-cong measure-kernel.emeasure-bind-kernel*[*OF - - A,symmetric*] *prob-kernel.axioms*(*1*) *simp*: *prob-kernel-def' simp del*: *emeasure-return*)
also have ... = *?rhs*
by(*auto intro*!: *measure-kernel.emeasure-bind-kernel*[*OF - - A,symmetric*] *s-finite-kernel.axioms*(*1*) *s-finite-kernel.bind-kernel-s-finite-kernel'*[**where** *Y*=ν] *s-finite-measure.s-finite-kernel assms*(*2*) *3*(*2*)] *prob-kernel.s-finite-kernel-prob-kernel*[*of* $\mu \bigotimes_M \nu$] *simp*: *prob-kernel-def'*)
finally show *?thesis* **.**
qed
qed *simp*
qed
qed

lemma *bind-kernel-rotate-return*:
assumes *s-finite-measure* μ *s-finite-measure* ν
shows $\mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,y))) = \nu \gg\!=_k (\lambda y.\ \mu \gg\!=_k (\lambda x.\ return\ (\mu \bigotimes_M \nu)\ (x,y)))$
proof −

consider *space* $\mu = \{\} \mid$ *space* $\nu = \{\} \mid$ *space* $\mu \neq \{\}$ *space* $\nu \neq \{\}$
by *auto*
then show *?thesis*
proof *cases*
case *1*
then have $\nu \gg\!=_k (\lambda y.\ \mu \gg\!=_k (\lambda x.\ return\ (\mu \bigotimes_M \nu)\ (x,y))) =$ *count-space* $\{\}$
by(*auto simp*: *bind-kernel-def space-empty*)
then show *?thesis*
by(*auto simp*: *bind-kernel-def space-pair-measure 1 intro*!: *space-empty*)
next
case *2*
then have $\mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y))) =$ *count-space* $\{\}$
by(*auto simp*: *bind-kernel-def space-empty*)
with *2* **show** *?thesis*
by(*auto simp*: *space-pair-measure bind-kernel-def intro*!: *space-empty*)
next
case *3*
show *?thesis*
unfolding *pair-measure-eq-bind-s-finite*[*OF assms*,*symmetric*]
proof(*intro measure-eqI*)
fix *A*
assume *A*[*measurable*]:$A \in$ *sets* $(\mu \bigotimes_M \nu)$
show *emeasure* $(\mu \bigotimes_M \nu)\ A =$ *emeasure* $(\nu \gg\!=_k (\lambda y.\ \mu \gg\!=_k (\lambda x.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y))))\ A$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+ x.\ \int^+ y.$ *indicator* $A\ (x,\ y)\ \partial\nu\ \partial\mu)$
by(*rule s-finite-measure.emeasure-pair-measure'*[*OF assms*(*2*) *A*])
also have ... = $(\int^+ y.\ \int^+ x.$ *return* $(\mu \bigotimes_M \nu)\ (x,\ y)\ A\ \partial\mu\ \partial\nu)$
by(*simp add*: *nn-integral-snd'*[*OF assms*] *s-finite-measure.nn-integral-fst'*[*OF assms*(*2*)])
also have ... = $(\int^+ y.\ (\mu \gg\!=_k (\lambda x.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y)))\ A\ \partial\nu)$
by(*auto intro*!: *nn-integral-cong measure-kernel.emeasure-bind-kernel*[*OF - - A*,*symmetric*] *prob-kernel.axioms*(*1*) *simp add*: *prob-kernel-def' simp del*: *emeasure-return*)
also have ... = *?rhs*
by(*auto intro*!: *measure-kernel.emeasure-bind-kernel*[*OF - - A*,*symmetric*] *s-finite-kernel.axioms*(*1*) *s-finite-kernel.bind-kernel-s-finite-kernel'*[**where** *Y*=μ] *s-finite-measure.s-finite-kernel assms*(*1*) *3*(*1*)] *prob-kernel.s-finite-kernel-prob-kernel*[*of* $\nu \bigotimes_M \mu$] *simp*: *prob-kernel-def'*)
finally show *?thesis* **.**
qed
qed(*auto intro*!: *sets-bind-kernel*[*OF - 3*(*2*),*symmetric*] *sets-bind-kernel*[*OF - 3*(*1*)])
qed
qed

lemma *bind-kernel-rotate'*:
assumes *s-finite-measure* μ *s-finite-measure* ν *s-finite-kernel* $(\mu \bigotimes_M \nu)$ *Z* (*case-prod f*)

shows $\mu \gg\!= _k (\lambda x.\ \nu \gg\!= _k (\lambda y.\ f\ x\ y)) = \nu \gg\!= _k (\lambda y.\ \mu \gg\!= _k (\lambda x.\ f\ x\ y))$ (**is** *?lhs = ?rhs*)
proof −
interpret *sk*: *s-finite-kernel* $\mu \bigotimes_M \nu$ *Z case-prod f* **by** *fact*
consider *space* $\mu = \{\} \mid$ *space* $\nu = \{\} \mid$ *space* $\mu \neq \{\}$ *space* $\nu \neq \{\}$
by *auto*
then show *?thesis*
proof *cases*
case *1*
then have *?rhs = count-space* $\{\}$
by(*auto simp*: *bind-kernel-def space-empty*)
then show *?thesis*
by(*auto simp*: *bind-kernel-def space-pair-measure 1 intro*!: *space-empty*)
next
case *2*
then show *?thesis*
by(*auto simp*: *space-pair-measure bind-kernel-def intro*!: *space-empty*)
next
case *3*
show *?thesis*
proof −
have *?lhs* $= \mu \gg\!= _k (\lambda x.\ \nu \gg\!= _k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y)) \gg\!= _k$ *case-prod f*)
by(*auto intro*!: *bind-kernel-cong-All simp*: *s-finite-kernel.bind-kernel-assoc*[*OF prob-kernel.s-finite-kernel-prob-kernel assms*(*3*) *refl*,*of* ν $\lambda y.\ return\ (\mu \bigotimes_M \nu)$ (-, *y*),*simplified prob-kernel-def'*,*symmetric*] *sk.bind-kernel-return space-pair-measure*)
also have ... $= \mu \gg\!= _k (\lambda x.\ \nu \gg\!= _k (\lambda y.\ return\ (\mu \bigotimes_M \nu)\ (x,y))) \gg\!= _k$ (*case-prod f*)
by(*auto simp*: *s-finite-kernel.bind-kernel-assoc*[*OF s-finite-kernel.bind-kernel-s-finite-kernel'*[*OF s-finite-measure.s-finite-kernel-const*[*OF assms*(*2*) *3*(*2*),*of* μ] *prob-kernel.s-finite-kernel-prob-kernel*,*of* $\mu \bigotimes_M \nu$ $\lambda x\ y.\ return\ (\mu \bigotimes_M \nu)$ (*x*,*y*),*simplified*] *assms*(*3*) *refl*, *simplified prob-kernel-def'*,*symmetric*])
also have ... $= \nu \gg\!= _k (\lambda y.\ \mu \gg\!= _k (\lambda x.\ return\ (\mu \bigotimes_M \nu)\ (x,y))) \gg\!= _k$ (*case-prod f*)
by(*simp add*: *bind-kernel-rotate-return assms*)
also have ... $= \nu \gg\!= _k (\lambda y.\ \mu \gg\!= _k (\lambda x.\ return\ (\mu \bigotimes_M \nu)\ (x,\ y)) \gg\!= _k$ *case-prod f*)
by(*auto intro*!: *s-finite-kernel.bind-kernel-assoc*[*OF* - *assms*(*3*),*symmetric*] *s-finite-kernel.bind-kernel-s-finite-kernel'*[*OF s-finite-measure.s-finite-kernel-const*[*OF assms*(*1*) *3*(*1*)]] *prob-kernel.s-finite-kernel-prob-kernel*[*of* $\nu \bigotimes_M \mu$] *simp*: *prob-kernel-def'*)
also have ... = *?rhs*
by(*auto intro*!: *bind-kernel-cong-All simp*: *s-finite-kernel.bind-kernel-assoc*[*OF prob-kernel.s-finite-kernel-prob-kernel assms*(*3*) *refl*,*of* μ $\lambda x.\ return\ (\mu \bigotimes_M \nu)$ (*x*, -),*simplified prob-kernel-def'*,*symmetric*] *sk.bind-kernel-return space-pair-measure*)
finally show *?thesis* **.**
qed
qed
qed

lemma *bind-kernel-rotate*:

assumes *sets* μ = *sets* X **and** *sets* ν = *sets* Y
and *s-finite-measure* μ *s-finite-measure* ν *s-finite-kernel* ($X \bigotimes_M Y$) Z ($\lambda(x,y)$. $f\ x\ y$)
shows $\mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ f\ x\ y)) = \nu \gg\!=_k (\lambda y.\ \mu \gg\!=_k (\lambda x.\ f\ x\ y))$
by(*auto intro*!: *bind-kernel-rotate'* *assms simp*: *s-finite-kernel-cong-sets*[*OF sets-pair-measure-cong*[*OF assms*(*1*,*2*)]])

lemma(**in** *s-finite-kernel*) *emeasure-measurable'*:
assumes *A*[*measurable*]: (*SIGMA x*:*space X*. *A x*) ∈ *sets* ($X \bigotimes_M Y$)
shows (λx. *emeasure* (κ *x*) (*A x*)) ∈ *borel-measurable X*
proof −
have ∗∗: *A x* ∈ *sets Y* **if** *x* ∈ *space X* **for** *x*
proof −
have *Pair x* −‘ *Sigma* (*space X*) *A* = *A x*
using *that* **by** *auto*
with *sets-Pair1*[*OF A*, *of x*] **show** *A x* ∈ *sets Y*
by *auto*
qed

have ∗: $\bigwedge x$. *fst x* ∈ *space X* ⟹ *snd x* ∈ *A* (*fst x*) ⟷ *x* ∈ (*SIGMA x*:*space X*. *A x*)
by (*auto simp*: *fun-eq-iff*)
have ($\lambda(x,\ y)$. *indicator* (*A x*) *y*::*ennreal*) ∈ *borel-measurable* ($X \bigotimes_M Y$)
by (*measurable*,*subst measurable-cong*[*OF* ∗]) (*auto simp*: *space-pair-measure*)
then have (λx. *integral*^{N} (κ *x*) (*indicator* (*A x*))) ∈ *borel-measurable X*
by(*rule nn-integral-measurable-f*)
moreover have *integral*^{N} (κ *x*) (*indicator* (*A x*)) = *emeasure* (κ *x*) (*A x*) **if** *x* ∈ *space X* **for** *x*
using ∗∗[*OF that*] *kernel-sets*[*OF that*] **by**(*auto intro*!: *nn-integral-indicator*)
ultimately show (λx. *emeasure* (κ *x*) (*A x*)) ∈ *borel-measurable X*
by(*auto cong*: *measurable-cong*)
qed

lemma(**in** *s-finite-kernel*) *measure-measurable'*:
assumes (*SIGMA x*:*space X*. *A x*) ∈ *sets* ($X \bigotimes_M Y$)
shows (λx. *measure* (κ *x*) (*A x*)) ∈ *borel-measurable X*
using *emeasure-measurable'*[*OF assms*] **by**(*simp add*: *measure-def*)

lemma(**in** *s-finite-kernel*) *AE-pred*:
assumes *P*[*measurable*]:*Measurable.pred* ($X \bigotimes_M Y$) (*case-prod P*)
shows *Measurable.pred X* (λx. *AE y in* κ *x*. *P x y*)
proof −
have [*measurable*]:*Measurable.pred X* (λx. *emeasure* (κ *x*) {*y* ∈ *space Y*. ¬ *P x y*} = *0*)
proof(*intro pred-eq-const1*[**where** *N*=*borel*] *emeasure-measurable'*)
have (*SIGMA x*:*space X*. {*y* ∈ *space Y*. ¬ *P x y*}) = {*xy*∈*space* ($X \bigotimes_M Y$). ¬ *P* (*fst xy*) (*snd xy*)}
by (*auto simp*: *space-pair-measure*)
also have ... ∈ *sets* ($X \bigotimes_M Y$)

by *simp*
finally show (*SIGMA x:space X.* {*y* ∈ *space Y.* ¬ *P x y*}) ∈ *sets* (*X* ⨂$_M$ *Y*)
.
qed *simp*
have {*x* ∈ *space X. almost-everywhere* (κ *x*) (*P x*)} = {*x* ∈ *space X. emeasure* (κ *x*) {*y*∈*space Y.* ¬ *P x y*} = *0*}
proof *safe*
fix *x*
assume *x*:*x* ∈ *space X*
show (*AE y in* κ *x. P x y*) ⟹ *emeasure* (κ *x*) {*y* ∈ *space Y.* ¬ *P x y*} = *0*
using *emeasure-eq-0-AE*[*of* λ*y.* ¬ *P x y* κ *x*]
by(*simp add*: *sets-eq-imp-space-eq*[*OF kernel-sets*[*OF x*]])
show *emeasure* (κ *x*) {*y* ∈ *space Y.* ¬ *P x y*} = *0* ⟹ *almost-everywhere* (κ *x*) (*P x*)
using *x* **by**(*auto intro*!: *AE-I*[**where** *N*={*y* ∈ *space Y.* ¬ *P x y*}] *simp*: *sets-eq-imp-space-eq*[*OF kernel-sets*[*OF x*]] *kernel-sets*[*OF x*])
qed
also have ... ∈ *sets X*
by(*simp add*: *pred-def*)
finally show *?thesis*
by(*simp add*: *pred-def*)
qed

lemma(**in** *subprob-kernel*) *integrable-probability-kernel-pred*:
fixes *f* :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
assumes [*measurable*]:(λ(*x*,*y*). *f x y*) ∈ *borel-measurable* (*X* ⨂$_M$ *Y*)
shows *Measurable.pred X* (λ*x. integrable* (κ *x*) (*f x*))
proof(*rule measurable-cong*[*THEN iffD2*])
show *x* ∈ *space X* ⟹ *integrable* (κ *x*) (*f x*) ⟷ (∫$^+$*y. norm* (*f x y*) ∂(κ *x*)) < ∞ **for** *x*
by(*auto simp*: *integrable-iff-bounded*)
next
have (λ(*x*,*y*). *ennreal* (*norm* (*f x y*))) ∈ *borel-measurable* (*X* ⨂$_M$ *Y*)
by *measurable*
from *nn-integral-measurable-f*[*OF this*]
show *Measurable.pred X* (λ*x.* (∫$^+$ *y. ennreal* (*norm* (*f x y*)) ∂κ *x*) < ∞)
by *simp*
qed

corollary *integrable-measurable-subprob′*:
fixes *f* :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
assumes [*measurable*]:(λ(*x*,*y*). *f x y*) ∈ *borel-measurable* (*X* ⨂$_M$ *Y*) *k* ∈ *X* →$_M$ *subprob-algebra Y*
shows *Measurable.pred X* (λ*x. integrable* (*k x*) (*f x*))
by(*auto intro*!: *subprob-kernel.integrable-probability-kernel-pred*[**where** *Y*=*Y*] *simp*: *subprob-kernel-def′*)

lemma(**in** *subprob-kernel*) *integrable-probability-kernel-pred′*:
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}

assumes $f \in$ *borel-measurable* $(X \bigotimes_M Y)$
shows *Measurable.pred* X $(\lambda x.$ *integrable* $(\kappa\ x)$ $(curry\ f\ x))$
using *integrable-probability-kernel-pred*[*of curry f*] *assms* **by** *auto*

lemma(**in** *subprob-kernel*) *lebesgue-integral-measurable-f-subprob*:
fixes f :: - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes [*measurable*]:$f \in$ *borel-measurable* $(X \bigotimes_M Y)$
shows $(\lambda x. \int y.\ f\ (x,y)\ \partial(\kappa\ x)) \in$ *borel-measurable* X
by(*rule integral-measurable-subprob-algebra2*[**where** N=Y])
(*use subprob-kernel-axioms* **in** *auto simp*: *subprob-kernel-def'*)

lemma(**in** *s-finite-kernel*) *integrable-measurable-pred*[*measurable* (*raw*)]:
fixes f :: - $\Rightarrow$ - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes [*measurable*]:*case-prod* $f \in$ *borel-measurable* $(X \bigotimes_M Y)$
shows *Measurable.pred* X $(\lambda x.$ *integrable* $(\kappa\ x)$ $(f\ x))$
proof(*cases space* $X = \{\}$)
case *True*
from *space-empty*[*OF this*] **show** *?thesis*
by *simp*
next
case *h*:*False*
obtain *ki* **where** *ki*:$\bigwedge i.$ *subprob-kernel* $X\ Y$ $(ki\ i)$ $\bigwedge x\ A.\ x \in space\ X \Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
using *s-finite-kernels* **by** *metis*
have [*simp*]:*integrable* $(\kappa\ x)$ $(f\ x) = ((\sum i. \int^+ y.\ ennreal\ (norm\ (f\ x\ y))\ \partial ki\ i\ x) < \infty)$ **if** $x \in space\ X$ **for** x
using *ki*(*1*) *nn-integral-measure-suminf*[*of* $\lambda i.$ *ki i x* κ *x*,*OF - ki*(*2*)] *that kernel-sets*
by(*auto simp*: *integrable-iff-bounded subprob-kernel-def measure-kernel-def*)
note [*measurable*] = *nn-integral-measurable-subprob-algebra2*
show *?thesis*
by(*rule measurable-cong*[**where** g=$\lambda x.$ $(\sum i. \int^+ y.\ ennreal\ (norm\ (f\ x\ y))\ \partial(ki\ i\ x)) < \infty$,*THEN iffD2*]) (*insert ki*(*1*), *auto simp*: *subprob-kernel-def'*)
qed

lemma(**in** *s-finite-kernel*) *integral-measurable-f*:
fixes f :: - $\Rightarrow$ - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes [*measurable*]:*case-prod* $f \in$ *borel-measurable* $(X \bigotimes_M Y)$
shows $(\lambda x. \int y.\ f\ x\ y\ \partial(\kappa\ x)) \in$ *borel-measurable* X
proof −
obtain *ki* **where** *ki*:$\bigwedge i.$ *subprob-kernel* $X\ Y$ $(ki\ i)$ $\bigwedge x\ A.\ x \in space\ X \Longrightarrow \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)$
using *s-finite-kernels* **by** *metis*
note [*measurable*] = *integral-measurable-subprob-algebra2*

show *?thesis*
proof(*rule measurable-cong*[*THEN iffD1*])
fix x
assume *h*:$x \in space\ X$

```
    {
      assume h':integrable (κ x) (f x)
      have (∑ i. ∫ y. f x y ∂(ki i x)) = (∫ y. f x y ∂(κ x))
       using lebesgue-integral-measure-suminf[of λi. ki i x κ x,OF - ki(2) h'] ki(1)
kernel-sets[OF h] h
        by(auto simp: subprob-kernel-def measure-kernel-def)
    }
    thus (if integrable (κ x) (f x) then (∑ i. ∫ y. f x y ∂(ki i x)) else 0) = (∫ y. f
x y ∂(κ x))
      using not-integrable-integral-eq by auto
  qed(use ki(1) in auto simp: subprob-kernel-def')
qed

lemma(in s-finite-kernel) integral-measurable-f':
  fixes f :: - ⇒ -::{banach, second-countable-topology}
  assumes [measurable]:f ∈ borel-measurable (X ⨂M Y)
  shows (λx. ∫ y. f (x,y) ∂(κ x)) ∈ borel-measurable X
  using integral-measurable-f[of curry f] by simp

lemma(in measure-kernel)
  fixes f :: - ⇒ -::{banach, second-countable-topology}
  assumes [measurable-cong]: sets μ = sets X
      and integrable (μ ⤜k κ) f
     shows integrable-bind-kernelD1: integrable μ (λx. ∫ y. norm (f y) ∂κ x) (is
?g1)
      and integrable-bind-kernelD1': integrable μ (λx. ∫ y. f y ∂κ x) (is ?g1')
      and integrable-bind-kernelD2: AE x in μ. integrable (κ x) f (is ?g2)
      and integrable-bind-kernelD3: space X ≠ {} ⟹ f ∈ borel-measurable Y (is
- ⟹ ?g3)
proof −
  show 1:space X ≠ {} ⟹ ?g3
   using assms(2) sets-bind-kernel[OF - assms(1)] by(simp add: integrable-iff-bounded
cong: measurable-cong-sets)
  have integrable μ (λx. ∫ y. norm (f y) ∂κ x) ∧ integrable μ (λx. ∫ y. f y ∂κ x)
∧ (AE x in μ. integrable (κ x) f)
  proof(cases space X = {})
    assume ne: space X ≠ {}
    then have space μ ≠ {} by(simp add: sets-eq-imp-space-eq[OF assms(1)])
   note h = integral-measurable-kernel[measurable] sets-bind-kernel[OF ne assms(1),measurable-cong]
            nn-integral-measurable-kernel[measurable]
    have g2: ?g2
      unfolding integrable-iff-bounded AE-conj-iff
    proof safe
      show AE x in μ. f ∈ borel-measurable (κ x)
        using assms(2) by(auto simp: sets-eq-imp-space-eq[OF assms(1)] measur-
able-cong-sets[OF kernel-sets])
    next
      have AE x in μ. (∫+ x. ennreal (norm (f x)) ∂κ x) ≠ ∞
        by(rule nn-integral-PInf-AE)
```

(*use assms(2)* **in** *auto simp*: *integrable-iff-bounded nn-integral-bind-kernel*[*OF* - *assms(1)*])
thus *AE x in* μ. ($\int^+$ *x*. *ennreal* (*norm* (*f x*)) $\partial\kappa$ *x*) < ∞
by (*simp add*: *top.not-eq-extremum*)
qed
have [*simp*]:($\int^+$ *x*. $\int^+$ *x*. *ennreal* (*norm* (*f x*)) $\partial\kappa$ *x* $\partial\mu$) = ($\int^+$ *x*. *ennreal* ($\int$ *y*. *norm* (*f y*) $\partial\kappa$ *x*)$\partial\mu$)
using *g2* **by**(*auto intro*!: *nn-integral-cong-AE simp*: *nn-integral-eq-integral*)
have *g1*: *?g1*
using *assms(2)* **by**(*auto simp*: *integrable-iff-bounded measurable-cong-sets*[*OF h(2)*] *measurable-cong-sets*[*OF assms(1)*] *nn-integral-bind-kernel*[*OF* - *assms(1)*])
have *?g1'*
using *assms(2)* **by**(*auto intro*!: *Bochner-Integration.integrable-bound*[*OF g1*])
with *g2 g1* **show** *?thesis*
by *auto*
qed(*auto simp*: *space-empty*[*of* μ] *sets-eq-imp-space-eq*[*OF assms(1)*] *integrable-iff-bounded nn-integral-empty*)
thus *?g1 ?g1' ?g2*
by *auto*
qed

lemma(**in** *measure-kernel*)
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes [*measurable-cong*]: *sets* μ = *sets X*
and [*measurable*]:*AE x in* μ. *integrable* (κ *x*) *f integrable* μ (λ*x*. $\int$ *y*. *norm* (*f y*) $\partial\kappa$ *x*) *f* ∈ *borel-measurable Y*
shows *integrable-bind-kernel*: *integrable* (μ ⤜$_k$ κ) *f*
and *integral-bind-kernel*: ($\int$ *y*. *f y* ∂(μ ⤜$_k$ κ)) = ($\int$ *x*. ($\int$ *y*. *f y*$\partial\kappa$ *x*)∂ μ) (**is** *?eq*)
proof −
have *integrable* (μ ⤜$_k$ κ) *f* ∧ ($\int$ *y*. *f y* ∂(μ ⤜$_k$ κ)) = ($\int$ *x*. ($\int$ *y*. *f y*$\partial\kappa$ *x*)∂ μ)
proof(*cases space X* = {})
assume *ne*: *space X* ≠ {}
note *sets-bind*[*measurable-cong*] = *sets-bind-kernel*[*OF ne assms(1)*]
note *h*[*measurable*] = *integral-measurable-kernel measure-measurable*
have *1*:*integrable* (μ ⤜$_k$ κ) *f*
unfolding *integrable-iff-bounded*
proof
show ($\int^+$ *x*. *ennreal* (*norm* (*f x*)) ∂(μ ⤜$_k$ κ)) < ∞ (**is** *?l* < -)
proof −
have *?l* = ($\int^+$ *x*. *ennreal* ($\int$ *y*. *norm* (*f y*) $\partial\kappa$ *x*)$\partial\mu$)
using *assms(2)* **by**(*auto intro*!: *nn-integral-cong-AE simp*: *nn-integral-eq-integral simp*: *nn-integral-bind-kernel*[*OF* - *assms(1)*])
also have ... < ∞
using *assms(3)* **by**(*auto simp*: *integrable-iff-bounded*)
finally show *?thesis* .
qed
qed *simp*
then have *?eq*

proof *induction*
case *h*[*measurable*]:(*base A c*)
hence *1*:*integrable* (*μ* ⤜*k κ*) (*indicat-real A*)
by *simp*
have *2*:*integrable μ* (*λx. measure* (*κ x*) *A*)
by(*rule Bochner-Integration.integrable-cong*[**where** *f*=*λx. Sigma-Algebra.measure* (*κ x*) (*A* ∩ *space* (*κ x*)),*THEN iffD1*,*OF refl*])
(*insert h integrable-bind-kernelD1*[*OF assms*(*1*) *1*] *sets-eq-imp-space-eq*[*OF kernel-sets*], *auto simp*: *sets-eq-imp-space-eq*[*OF assms*(*1*)] *sets-eq-imp-space-eq*[*OF kernel-sets*] *sets-bind*)
have *AE x in μ. emeasure* (*κ x*) *A* ≠ ∞
by(*rule nn-integral-PInf-AE*,*insert h*) (*auto simp*: *emeasure-bind-kernel*[*OF assms*(*1*)] *sets-bind*)
hence *0*:*AE x in μ. emeasure* (*κ x*) *A* < ∞
by (*simp add*: *top.not-eq-extremum*)
have (∫ *x*. (∫ *y*. *indicat-real A y* ∗*R c* ∂*κ x*)∂ *μ*) = (∫ *x*. *measure* (*κ x*) *A* ∗*R c*∂*μ*)
using *h integrable-bind-kernelD2*[*OF assms*(*1*) *integrable-real-indicator*,*of A*]
by(*auto intro*!: *integral-cong-AE simp*: *sets-eq-imp-space-eq*[*OF kernel-sets*] *sets-bind sets-eq-imp-space-eq*[*OF assms*(*1*)])
also have ... = (∫ *x*. *measure* (*κ x*) *A* ∂*μ*) ∗*R c*
using *2* **by**(*auto intro*!: *integral-scaleR-left*)
finally show *?case*
using *h* **by**(*auto simp*: *measure-bind-kernel*[*OF assms*(*1*) - *0*] *sets-bind*)
next
case *ih*:(*add f g*)
show *?case*
using *ih*(*1*,*2*) *integrable-bind-kernelD2*[*OF assms*(*1*) *ih*(*1*)] *integrable-bind-kernelD2*[*OF assms*(*1*) *ih*(*2*)]
by(*auto simp*: *ih*(*3*,*4*) *Bochner-Integration.integral-add*[*OF integrable-bind-kernelD1'*[*OF assms*(*1*) *ih*(*1*)] *integrable-bind-kernelD1'*[*OF assms*(*1*) *ih*(*2*)],*symmetric*] *intro*!: *integral-cong-AE*)
next
case *ih*:(*lim f fn*)
show *?case* (**is** *?lhs* = *?rhs*)
proof −
have *conv*: *AE x in μ.* (*λn.* ∫ *y. fn n y*∂*κ x*) ⟶ (∫ *y. f y* ∂*κ x*)
proof −
have *conv*:*AE x in μ. integrable* (*κ x*) *f* ⟶ (*λn.* ∫ *y. fn n y*∂*κ x*) ⟶ (∫ *y. f y* ∂*κ x*)
proof
fix *x*
assume *h*:*x* ∈ *space μ*
then show *integrable* (*κ x*) *f* ⟶ (*λn.* ∫ *y. fn n y*∂*κ x*) ⟶ (∫ *y. f y* ∂*κ x*)
using *ih* **by**(*auto intro*!: *integral-dominated-convergence*[**where** *w*=*λx. 2* ∗ *norm* (*f x*)] *simp*: *sets-eq-imp-space-eq*[*OF sets-bind*] *sets-eq-imp-space-eq*[*OF kernel-sets*[*OF h*[*simplified sets-eq-imp-space-eq*[*OF assms*(*1*)]]]] *sets-eq-imp-space-eq*[*OF*

assms(*1*)])
qed
with *conv integrable-bind-kernelD2*[*OF assms*(*1*) *ih*(*4*)]
show *?thesis* **by** *fastforce*
qed
have *?lhs* = *lim* (λ*n*. ∫ *y*. *fn n y* ∂(μ ⋙=$_k$ κ))
by(*rule limI*[*OF integral-dominated-convergence*[**where** *w*=λ*x*. *2* ∗ *norm* (*f x*)],*symmetric*]) (*use ih* **in** *auto*)
also have ... = *lim* (λ*n*. (∫ *x*. (∫ *y*. *fn n y*∂κ *x*)∂ μ))
by(*simp add*: *ih*)
also have ... = (∫ *x*. *lim* (λ*n*. ∫ *y*. *fn n y*∂κ *x*)∂ μ)
proof(*rule limI*[*OF integral-dominated-convergence*[**where** *w*=λ*x*. ∫ *y*. *2* ∗ *norm* (*f y*) ∂κ *x*]])
fix *n*
show *AE x in* μ. *norm* (∫ *y*. *fn n y*∂κ *x*) ≤ (∫ *y*. *2* ∗ *norm* (*f y*) ∂κ *x*)
by(*rule AE-mp*[*OF integrable-bind-kernelD2*[*OF assms*(*1*) *ih*(*1*),*of n*] *AE-mp*[*OF integrable-bind-kernelD2*[*OF assms*(*1*) *ih*(*4*)]]],*standard*+,*rule order.trans*[*OF integral-norm-bound integral-mono*[*of* κ - λ*y*. *norm* (*fn n y*) -,*OF* - - *ih*(*3*)[*simplified sets-eq-imp-space-eq*[*OF sets-bind*]]]])
(*auto simp*: *sets-eq-imp-space-eq*[*OF assms*(*1*)] *sets-eq-imp-space-eq*[*OF kernel-sets*])
qed(*use ih integrable-bind-kernelD1*[*OF assms*(*1*) *ih*(*4*)] *conv limI* **in** *auto*,*fastforce*)
also have ... = *?rhs*
using *ih conv limI* **by**(*auto intro*!: *integral-cong-AE*, *blast*)
finally show *?thesis* **.**
qed
qed
with *1* **show** *?thesis*
by *auto*
qed(*auto simp*: *bind-kernel-def space-empty*[*of* μ] *sets-eq-imp-space-eq*[*OF assms*(*1*)] *integrable-iff-bounded nn-integral-empty Bochner-Integration.integral-empty*)
thus *integrable* (μ ⋙=$_k$ κ) *f ?eq*
by *auto*
qed

end

3 Quasi-Borel Spaces

theory *QuasiBorel*
imports *HOL−Probability.Probability*
begin

3.1 Definitions

3.1.1 Quasi-Borel Spaces

definition *qbs-closed1* :: (*real* ⇒ *'a*) *set* ⇒ *bool*

where *qbs-closed1 Mx* ≡ (∀ *a* ∈ *Mx*. ∀ *f* ∈ (*borel* :: *real measure*) →$_M$ (*borel* :: *real measure*). *a* ∘ *f* ∈ *Mx*)

definition *qbs-closed2* :: ['*a set*, (*real* ⇒ '*a*) *set*] ⇒ *bool*
where *qbs-closed2 X Mx* ≡ (∀ *x* ∈ *X*. (λ*r*. *x*) ∈ *Mx*)

definition *qbs-closed3* :: (*real* ⇒ '*a*) *set* ⇒ *bool*
where *qbs-closed3 Mx* ≡ (∀ *P*::*real* ⇒ *nat*. ∀ *Fi*::*nat* ⇒ *real* ⇒ '*a*.
(*P* ∈ *borel* →$_M$ *count-space UNIV*) ⟶ (∀ *i*. *Fi i* ∈ *Mx*) ⟶
(λ*r*. *Fi* (*P r*) *r*) ∈ *Mx*)

lemma *separate-measurable*:
fixes *P* :: *real* ⇒ *nat*
assumes ⋀*i*. *P* −‘ {*i*} ∈ *sets borel*
shows *P* ∈ *borel* →$_M$ *count-space UNIV*
by (*auto simp add*: *assms measurable-count-space-eq-countable*)

lemma *measurable-separate*:
fixes *P* :: *real* ⇒ *nat*
assumes *P* ∈ *borel* →$_M$ *count-space UNIV*
shows *P* −‘ {*i*} ∈ *sets borel*
by (*metis assms borel-singleton measurable-sets-borel sets.empty-sets sets-borel-eq-count-space*)

definition *is-quasi-borel X Mx* ⟷ *Mx* ⊆ *UNIV* → *X* ∧ *qbs-closed1 Mx* ∧ *qbs-closed2 X Mx* ∧ *qbs-closed3 Mx*

lemma *is-quasi-borel-intro*[*simp*]:
assumes *Mx* ⊆ *UNIV* → *X*
and *qbs-closed1 Mx qbs-closed2 X Mx qbs-closed3 Mx*
shows *is-quasi-borel X Mx*
using *assms* **by**(*simp add*: *is-quasi-borel-def*)

typedef '*a quasi-borel* = {(*X*::'*a set*, *Mx*). *is-quasi-borel X Mx*}
proof
show (*UNIV*, *UNIV*) ∈ {(*X*::'*a set*, *Mx*). *is-quasi-borel X Mx*}
by (*simp add*: *is-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def*)
qed

definition *qbs-space* :: '*a quasi-borel* ⇒ '*a set* **where**
qbs-space X ≡ *fst* (*Rep-quasi-borel X*)

definition *qbs-Mx* :: '*a quasi-borel* ⇒ (*real* ⇒ '*a*) *set* **where**
qbs-Mx X ≡ *snd* (*Rep-quasi-borel X*)

declare [[*coercion qbs-space*]]

lemma *qbs-decomp* : (*qbs-space X*,*qbs-Mx X*) ∈ {(*X*::'*a set*, *Mx*). *is-quasi-borel X Mx*}
by (*simp add*: *qbs-space-def qbs-Mx-def Rep-quasi-borel*[*simplified*])

lemma *qbs-Mx-to-X*:
assumes $\alpha \in$ *qbs-Mx X*
shows $\alpha\ r \in$ *qbs-space X*
using *qbs-decomp assms* **by**(*auto simp*: *is-quasi-borel-def*)

lemma *qbs-closed1I*:
assumes $\bigwedge \alpha\ f.\ \alpha \in Mx \Longrightarrow f \in borel \to_M borel \Longrightarrow \alpha \circ f \in Mx$
shows *qbs-closed1 Mx*
using *assms* **by**(*simp add*: *qbs-closed1-def*)

lemma *qbs-closed1-dest*[*simp*]:
assumes $\alpha \in$ *qbs-Mx X*
and $f \in borel \to_M borel$
shows $\alpha \circ f \in$ *qbs-Mx X*
using *assms qbs-decomp* **by** (*auto simp add*: *is-quasi-borel-def qbs-closed1-def*)

lemma *qbs-closed1-dest'*[*simp*]:
assumes $\alpha \in$ *qbs-Mx X*
and $f \in borel \to_M borel$
shows $(\lambda r.\ \alpha\ (f\ r)) \in$ *qbs-Mx X*
using *qbs-closed1-dest*[*OF assms*] **by** (*simp add*: *comp-def*)

lemma *qbs-closed2I*:
assumes $\bigwedge x.\ x \in X \Longrightarrow (\lambda r.\ x) \in Mx$
shows *qbs-closed2 X Mx*
using *assms* **by**(*simp add*: *qbs-closed2-def*)

lemma *qbs-closed2-dest*[*simp*]:
assumes $x \in$ *qbs-space X*
shows $(\lambda r.\ x) \in$ *qbs-Mx X*
using *assms qbs-decomp*[*of X*] **by** (*auto simp add*: *is-quasi-borel-def qbs-closed2-def*)

lemma *qbs-closed3I*:
assumes $\bigwedge(P :: real \Rightarrow nat)\ Fi.\ P \in borel \to_M$ *count-space UNIV* $\Longrightarrow (\bigwedge i.\ Fi\ i \in Mx)$
$\Longrightarrow (\lambda r.\ Fi\ (P\ r)\ r) \in Mx$
shows *qbs-closed3 Mx*
using *assms* **by**(*auto simp*: *qbs-closed3-def*)

lemma *qbs-closed3I'*:
assumes $\bigwedge(P :: real \Rightarrow nat)\ Fi.\ (\bigwedge i.\ P -` \{i\} \in sets\ borel) \Longrightarrow (\bigwedge i.\ Fi\ i \in Mx)$
$\Longrightarrow (\lambda r.\ Fi\ (P\ r)\ r) \in Mx$
shows *qbs-closed3 Mx*
using *assms* **by**(*auto intro*!: *qbs-closed3I dest*: *measurable-separate*)

lemma *qbs-closed3-dest*[*simp*]:
fixes *P*::*real* $\Rightarrow$ *nat* **and** *Fi* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ -

assumes $P \in borel \rightarrow_M count\text{-}space\ UNIV$
and $\bigwedge i.\ Fi\ i \in qbs\text{-}Mx\ X$
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in qbs\text{-}Mx\ X$
using *assms qbs-decomp*[*of X*] **by** (*auto simp add*: *is-quasi-borel-def qbs-closed3-def*)

lemma *qbs-closed3-dest'*:
fixes *P*::*real* ⇒ *nat* **and** *Fi* :: *nat* ⇒ *real* ⇒ -
assumes $\bigwedge i.\ P -` \{i\} \in sets\ borel$
and $\bigwedge i.\ Fi\ i \in qbs\text{-}Mx\ X$
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in qbs\text{-}Mx\ X$
using *qbs-closed3-dest*[*OF separate-measurable*[*OF assms*(*1*)] *assms*(*2*)] .

lemma *qbs-closed3-dest2*:
assumes *countable I*
and [*measurable*]: $P \in borel \rightarrow_M count\text{-}space\ I$
and $\bigwedge i.\ i \in I \Longrightarrow Fi\ i \in qbs\text{-}Mx\ X$
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in qbs\text{-}Mx\ X$
proof −
have *0*:$I \neq \{\}$
using *measurable-empty-iff*[*of count-space I P borel*] *assms*(*2*)
by *fastforce*
define P' **where** $P' \equiv to\text{-}nat\text{-}on\ I \circ P$
define Fi' **where** $Fi' \equiv Fi \circ (from\text{-}nat\text{-}into\ I)$
have *1*:$P' \in borel \rightarrow_M count\text{-}space\ UNIV$
by(*simp add*: *P'-def*)
have *2*:$\bigwedge i.\ Fi'\ i \in qbs\text{-}Mx\ X$
using *assms*(*3*) *from-nat-into*[*OF 0*] **by**(*simp add*: *Fi'-def*)
have $(\lambda r.\ Fi'\ (P'\ r)\ r) \in qbs\text{-}Mx\ X$
using *1 2 measurable-separate* **by** *auto*
thus *?thesis*
using *from-nat-into-to-nat-on*[*OF assms*(*1*)] *measurable-space*[*OF assms*(*2*)]
by(*auto simp*: *Fi'-def P'-def*)
qed

lemma *qbs-closed3-dest2'*:
assumes *countable I*
and [*measurable*]: $P \in borel \rightarrow_M count\text{-}space\ I$
and $\bigwedge i.\ i \in range\ P \Longrightarrow Fi\ i \in qbs\text{-}Mx\ X$
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in qbs\text{-}Mx\ X$
proof −
have *0*:*range P* ∩ *I* = *range P*
using *measurable-space*[*OF assms*(*2*)] **by** *auto*
have *1*:$P \in borel \rightarrow_M count\text{-}space\ (range\ P)$
using *restrict-count-space*[*of I range P*] *measurable-restrict-space2*[*OF - assms*(*2*),*of range P*]
by(*simp add*: *0*)
have *2*:*countable* (*range P*)
using *countable-Int2*[*OF assms*(*1*),*of range P*]
by(*simp add*: *0*)

show *?thesis*
by(*auto intro*!: *qbs-closed3-dest2*[*OF 2 1 assms*(*3*)])
qed

lemma *qbs-Mx-indicat*:
assumes $S \in$ *sets borel* $\alpha \in$ *qbs-Mx X* $\beta \in$ *qbs-Mx X*
shows $(\lambda r.$ *if* $r \in S$ *then* α r *else* β $r) \in$ *qbs-Mx X*
proof −
have $(\lambda r$::*real. if* $r \in S$ *then* α r *else* β $r) = (\lambda r.$ $(\lambda b.$ *if* b *then* α *else* $\beta)$ $(r \in S)$ $r)$
by(*auto simp*: *indicator-def*)
also have ... $\in$ *qbs-Mx X*
by(*rule qbs-closed3-dest2*[**where** *I*=*UNIV* **and** *Fi*=$\lambda b.$ *if b then* α *else* β]) (*use assms* **in** *auto*)
finally show *?thesis* **.**
qed

lemma *qbs-space-Mx*: *qbs-space X* = $\{\alpha$ x $|x$ $\alpha.$ $\alpha \in$ *qbs-Mx X*$\}$
proof *safe*
fix x
assume *1*:$x \in$ *qbs-space X*
show $\exists$ *xa* $\alpha.$ $x = \alpha$ *xa* $\wedge$ $\alpha \in$ *qbs-Mx X*
by(*auto intro*!: *exI*[**where** *x*=*0*] *exI*[**where** *x*=$(\lambda r.$ $x)$] *simp*: *1*)
qed(*simp add*: *qbs-Mx-to-X*)

lemma *qbs-space-eq-Mx*:
assumes *qbs-Mx X* = *qbs-Mx Y*
shows *qbs-space X* = *qbs-space Y*
by(*simp add*: *qbs-space-Mx assms*)

lemma *qbs-eqI*:
assumes *qbs-Mx X* = *qbs-Mx Y*
shows *X* = *Y*
by (*metis Rep-quasi-borel-inverse prod.exhaust-sel qbs-Mx-def qbs-space-def assms qbs-space-eq-Mx*[*OF assms*])

3.1.2 Empty Space

definition *empty-quasi-borel* :: *'a quasi-borel* **where**
empty-quasi-borel ≡ *Abs-quasi-borel* ({},{})

lemma
shows *eqb-space*[*simp*]: *qbs-space empty-quasi-borel* = ({} :: *'a set*)
and *eqb-Mx*[*simp*]: *qbs-Mx empty-quasi-borel* = ({} :: (*real* ⇒ *'a*) *set*)
proof −
have *Rep-quasi-borel empty-quasi-borel* = ({} :: *'a set*, {})
using *Abs-quasi-borel-inverse* **by**(*auto simp add*: *Abs-quasi-borel-inverse empty-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def is-quasi-borel-def*)
thus *qbs-space empty-quasi-borel* = ({} :: *'a set*) *qbs-Mx empty-quasi-borel* = ({}

:: (*real* ⇒ *'a*) *set*)
by(*auto simp add*: *qbs-space-def qbs-Mx-def*)
qed

lemma *qbs-empty-equiv* :*qbs-space X* = {} ⟷ *qbs-Mx X* = {}
proof *safe*
fix *x*
assume *qbs-Mx X* = {}
and *h*:*x* ∈ *qbs-space X*
have (λ*r*. *x*) ∈ *qbs-Mx X*
using *h* **by** *simp*
thus *x* ∈ {} **using** ‹*qbs-Mx X* = {}› **by** *simp*
qed(*use qbs-Mx-to-X* **in** *blast*)

lemma *empty-quasi-borel-iff*:
qbs-space X = {} ⟷ *X* = *empty-quasi-borel*
by(*auto intro*!: *qbs-eqI simp*: *qbs-empty-equiv*)

3.1.3 Unit Space

definition *unit-quasi-borel* :: *unit quasi-borel* (1$_Q$) **where**
unit-quasi-borel ≡ *Abs-quasi-borel* (*UNIV*,*UNIV*)

lemma
shows *unit-qbs-space*[*simp*]: *qbs-space unit-quasi-borel* = {()}
and *unit-qbs-Mx*[*simp*]: *qbs-Mx unit-quasi-borel* = {λ*r*. ()}
proof −
have *Rep-quasi-borel unit-quasi-borel* = (*UNIV*,*UNIV*)
using *Abs-quasi-borel-inverse* **by**(*auto simp add*: *unit-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def is-quasi-borel-def*)
thus *qbs-space unit-quasi-borel* = {()} *qbs-Mx unit-quasi-borel* = {λ*r*. ()}
by(*auto simp add*: *qbs-space-def qbs-Mx-def UNIV-unit*)
qed

3.1.4 Sub-Spaces

definition *sub-qbs* :: [*'a quasi-borel*, *'a set*] ⇒ *'a quasi-borel* **where**
sub-qbs X U ≡ *Abs-quasi-borel* (*qbs-space X* ∩ *U*,{α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)})

lemma
shows *sub-qbs-space*: *qbs-space* (*sub-qbs X U*) = *qbs-space X* ∩ *U*
and *sub-qbs-Mx*: *qbs-Mx* (*sub-qbs X U*) = {α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)}
proof −
have *qbs-closed1* {α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)} *qbs-closed2* (*qbs-space X* ∩ *U*) {α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)}
qbs-closed3 {α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)}
unfolding *qbs-closed1-def qbs-closed2-def qbs-closed3-def* **by** *auto*
hence *Rep-quasi-borel* (*sub-qbs X U*) = (*qbs-space X* ∩ *U*,{α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)})

by(*auto simp*: *sub-qbs-def is-quasi-borel-def qbs-Mx-to-X intro*!: *Abs-quasi-borel-inverse*)
thus *qbs-space* (*sub-qbs X U*) = *qbs-space X* ∩ *U qbs-Mx* (*sub-qbs X U*) = {α. α ∈ *qbs-Mx X* ∧ (∀ *r*. α *r* ∈ *U*)}
by(*simp-all add*: *qbs-Mx-def qbs-space-def*)
qed

lemma *sub-qbs*:
assumes *U* ⊆ *qbs-space X*
shows (*qbs-space* (*sub-qbs X U*), *qbs-Mx* (*sub-qbs X U*)) = (*U*, {*f* ∈ *UNIV* → *U*. *f* ∈ *qbs-Mx X*})
using *assms* **by** (*auto simp*: *sub-qbs-space sub-qbs-Mx*)

lemma *sub-qbs-ident*: *sub-qbs X* (*qbs-space X*) = *X*
by(*auto intro*!: *qbs-eqI simp*: *sub-qbs-Mx qbs-Mx-to-X*)

lemma *sub-qbs-sub-qbs*: *sub-qbs* (*sub-qbs X A*) *B* = *sub-qbs X* (*A* ∩ *B*)
by(*auto intro*!: *qbs-eqI simp*: *sub-qbs-Mx sub-qbs-space*)

3.1.5 Image Spaces

definition *map-qbs* :: ['*a* ⇒ '*b*] ⇒ '*a quasi-borel* ⇒ '*b quasi-borel* **where**
map-qbs f X = *Abs-quasi-borel* (*f* ‘ (*qbs-space X*),{*f* ∘ α |α. α∈ *qbs-Mx X*})

lemma
shows *map-qbs-space*: *qbs-space* (*map-qbs f X*) = *f* ‘ (*qbs-space X*)
and *map-qbs-Mx*: *qbs-Mx* (*map-qbs f X*) = {*f* ∘ α |α. α∈ *qbs-Mx X*}
proof −
have {*f* ∘ α |α. α∈ *qbs-Mx X*} ⊆ *UNIV* → *f* ‘ (*qbs-space X*)
using *qbs-Mx-to-X* **by** *fastforce*
moreover have *qbs-closed1* {*f* ∘ α |α. α∈ *qbs-Mx X*}
unfolding *qbs-closed1-def* **using** *qbs-closed1-dest* **by**(*fastforce simp*: *comp-def*)
moreover have *qbs-closed2* (*f* ‘ (*qbs-space X*)) {*f* ∘ α |α. α∈ *qbs-Mx X*}
unfolding *qbs-closed2-def* **by** *fastforce*
moreover have *qbs-closed3* {*f* ∘ α |α. α∈ *qbs-Mx X*}
proof(*rule qbs-closed3I'*)
fix *P* :: *real* ⇒ *nat* **and** *Fi*
assume *h*:⋀*i*::*nat*. *P* −‘ {*i*} ∈ *sets borel*
⋀*i*::*nat*. *Fi i* ∈ {*f* ∘ α |α. α∈ *qbs-Mx X*}
then obtain α*i* **where** *ha*: ⋀*i*::*nat*. α*i i* ∈ *qbs-Mx X* ⋀*i*. *Fi i* = *f* ∘ (α*i i*)
by *auto metis*
hence *1*:(λ*r*. α*i* (*P r*) *r*) ∈ *qbs-Mx X*
using *h*(*1*) *qbs-closed3-dest'* **by** *blast*
show (λ*r*. *Fi* (*P r*) *r*) ∈ {*f* ∘ α |α. α∈ *qbs-Mx X*}
by(*auto intro*!: *bexI*[**where** *x*=(λ*r*. α*i* (*P r*) *r*)] *simp add*: *1 ha comp-def*)
qed
ultimately have *Rep-quasi-borel* (*map-qbs f X*) = (*f* ‘ (*qbs-space X*),{*f* ∘ α |α. α∈ *qbs-Mx X*})
unfolding *map-qbs-def* **by**(*auto intro*!: *Abs-quasi-borel-inverse*)
thus *qbs-space* (*map-qbs f X*) = *f* ‘ (*qbs-space X*) *qbs-Mx* (*map-qbs f X*) = {*f* ∘

α $|\alpha.\ \alpha\in$ *qbs-Mx X*$\}$
by(*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

3.1.6 Binary Product Spaces

definition *pair-qbs* :: [*'a quasi-borel*, *'b quasi-borel*] $\Rightarrow$ (*'a* $\times$ *'b*) *quasi-borel* (**infixr** $\bigotimes_Q$ *80*) **where**
pair-qbs X Y = *Abs-quasi-borel* (*qbs-space X* $\times$ *qbs-space Y*, $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$)

lemma
shows *pair-qbs-space*: *qbs-space* (*X* $\bigotimes_Q$ *Y*) = *qbs-space X* $\times$ *qbs-space Y*
and *pair-qbs-Mx*: *qbs-Mx* (*X* $\bigotimes_Q$ *Y*) = $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$
proof −
have $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$ $\subseteq$ *UNIV* $\rightarrow$ *qbs-space X* $\times$ *qbs-space Y*
by (*auto simp*: *mem-Times-iff*[*of* - *qbs-space X qbs-space Y*]; *use qbs-Mx-to-X* **in** *fastforce*)
moreover have *qbs-closed1* $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$
unfolding *qbs-closed1-def* **by** (*metis* (*no-types*, *lifting*) *comp-assoc mem-Collect-eq qbs-closed1-dest*)
moreover have *qbs-closed2* (*qbs-space X* $\times$ *qbs-space Y*) $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$
unfolding *qbs-closed2-def* **by** *auto*
moreover have *qbs-closed3* $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$
proof(*safe intro*!: *qbs-closed3I*)
fix *P* :: *real* $\Rightarrow$ *nat*
fix *Fi* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *'a* $\times$ *'b*
define *Fj* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *'a* **where** *Fj* $\equiv \lambda j.$(*fst* $\circ$ *Fi j*)
assume $\forall\, i.$ *Fi i* $\in$ $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$
then have $\bigwedge i.$ *Fj i* $\in$ *qbs-Mx X* **by** (*simp add*: *Fj-def*)
moreover assume *P* $\in$ *borel* $\rightarrow_M$ *count-space UNIV*
ultimately have ($\lambda r.$ *Fj* (*P r*) *r*) $\in$ *qbs-Mx X*
by *auto*
moreover have *fst* $\circ$ ($\lambda r.$ *Fi* (*P r*) *r*) = ($\lambda r.$ *Fj* (*P r*) *r*) **by** (*auto simp add*: *Fj-def*)
ultimately show *fst* $\circ$ ($\lambda r.$ *Fi* (*P r*) *r*) $\in$ *qbs-Mx X* **by** *simp*
next
fix *P* :: *real* $\Rightarrow$ *nat*
fix *Fi* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *'a* $\times$ *'b*
define *Fj* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *'b* **where** *Fj* $\equiv \lambda j.$(*snd* $\circ$ *Fi j*)
assume $\forall\, i.$ *Fi i* $\in$ $\{f.$ *fst* $\circ$ *f* $\in$ *qbs-Mx X* $\wedge$ *snd* $\circ$ *f* $\in$ *qbs-Mx Y*$\}$
then have $\bigwedge i.$ *Fj i* $\in$ *qbs-Mx Y* **by** (*simp add*: *Fj-def*)
moreover assume *P* $\in$ *borel* $\rightarrow_M$ *count-space UNIV*
ultimately have ($\lambda r.$ *Fj* (*P r*) *r*) $\in$ *qbs-Mx Y*
by *auto*
moreover have *snd* $\circ$ ($\lambda r.$ *Fi* (*P r*) *r*) = ($\lambda r.$ *Fj* (*P r*) *r*) **by** (*auto simp add*:

Fj-def)
ultimately show *snd* ∘ (λ*r*. *Fi* (*P r*) *r*) ∈ *qbs-Mx Y* **by** *simp*
qed
ultimately have *Rep-quasi-borel* (*X* ⨂$_Q$ *Y*) = (*qbs-space X* × *qbs-space Y*, {*f*. *fst* ∘ *f* ∈ *qbs-Mx X* ∧ *snd* ∘ *f* ∈ *qbs-Mx Y*})
unfolding *pair-qbs-def* **by**(*auto intro*!: *Abs-quasi-borel-inverse is-quasi-borel-intro*)
thus *qbs-space* (*X* ⨂$_Q$ *Y*) = *qbs-space X* × *qbs-space Y qbs-Mx* (*X* ⨂$_Q$ *Y*) = {*f*. *fst* ∘ *f* ∈ *qbs-Mx X* ∧ *snd* ∘ *f* ∈ *qbs-Mx Y*}
by(*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

lemma *pair-qbs-fst*:
assumes *qbs-space Y* ≠ {}
shows *map-qbs fst* (*X* ⨂$_Q$ *Y*) = *X*
proof(*rule qbs-eqI*)
obtain *αy* **where** *hy*:*αy* ∈ *qbs-Mx Y*
using *qbs-empty-equiv*[*of Y*] *assms* **by** *auto*
show *qbs-Mx* (*map-qbs fst* (*X* ⨂$_Q$ *Y*)) = *qbs-Mx X*
by(*auto simp*: *map-qbs-Mx pair-qbs-Mx hy comp-def intro*!: *exI*[**where** *x*=λ*r*. (- *r*, *αy r*)])
qed

lemma *pair-qbs-snd*:
assumes *qbs-space X* ≠ {}
shows *map-qbs snd* (*X* ⨂$_Q$ *Y*) = *Y*
proof(*rule qbs-eqI*)
obtain *αx* **where** *hx*:*αx* ∈ *qbs-Mx X*
using *qbs-empty-equiv*[*of X*] *assms* **by** *auto*
show *qbs-Mx* (*map-qbs snd* (*X* ⨂$_Q$ *Y*)) = *qbs-Mx Y*
by(*auto simp*: *map-qbs-Mx pair-qbs-Mx hx comp-def intro*!: *exI*[**where** *x*=λ*r*. (*αx r*, - *r*)])
qed

3.1.7 Binary Coproduct Spaces

definition *copair-qbs-Mx* :: ['*a quasi-borel*, '*b quasi-borel*] ⇒ (*real* => '*a* + '*b*) *set* **where**
copair-qbs-Mx X Y ≡
{*g*. ∃ *S* ∈ *sets borel*.
(*S* = {} ⟶ (∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*)))) ∧
(*S* = *UNIV* ⟶ (∃ *α2*∈ *qbs-Mx Y*. *g* = (λ*r*. *Inr* (*α2 r*)))) ∧
((*S* ≠ {} ∧ *S* ≠ *UNIV*) ⟶
(∃ *α1*∈ *qbs-Mx X*. ∃ *α2*∈ *qbs-Mx Y*.
g = (λ*r*::*real*. (*if* (*r* ∈ *S*) *then Inl* (*α1 r*) *else Inr* (*α2 r*))))))}

definition *copair-qbs* :: ['*a quasi-borel*, '*b quasi-borel*] ⇒ ('*a* + '*b*) *quasi-borel* (**infixr** ⨁$_Q$ *65*) **where**
copair-qbs X Y ≡ *Abs-quasi-borel* (*qbs-space X* <+> *qbs-space Y*, *copair-qbs-Mx X Y*)

The following is an equivalent definition of *copair-qbs-Mx*.

definition *copair-qbs-Mx2* :: [*'a quasi-borel*, *'b quasi-borel*] ⇒ (*real* => *'a* + *'b*) *set* **where**
copair-qbs-Mx2 X Y ≡
{*g*. (*if qbs-space X* = {} ∧ *qbs-space Y* = {} *then False*
else if qbs-space X ≠ {} ∧ *qbs-space Y* = {} *then*
(∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*)))
else if qbs-space X = {} ∧ *qbs-space Y* ≠ {} *then*
(∃ *α2*∈ *qbs-Mx Y*. *g* = (λ*r*. *Inr* (*α2 r*)))
else
(∃ *S* ∈ *sets borel*. ∃ *α1*∈ *qbs-Mx X*. ∃ *α2*∈ *qbs-Mx Y*.
g = (λ*r*::*real*. (*if* (*r* ∈ *S*) *then Inl* (*α1 r*) *else Inr* (*α2 r*))))) }

lemma *copair-qbs-Mx-equiv* :*copair-qbs-Mx* (*X* :: *'a quasi-borel*) (*Y* :: *'b quasi-borel*)
= *copair-qbs-Mx2 X Y*
proof *safe*

fix *g* :: *real* ⇒ *'a* + *'b*
assume *g* ∈ *copair-qbs-Mx X Y*
then obtain *S* **where** *hs*:*S*∈ *sets borel* ∧
(*S* = {} ⟶ (∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*)))) ∧
(*S* = *UNIV* ⟶ (∃ *α2*∈ *qbs-Mx Y*. *g* = (λ*r*. *Inr* (*α2 r*)))) ∧
((*S* ≠ {} ∧ *S* ≠ *UNIV*) ⟶
(∃ *α1*∈ *qbs-Mx X*.
∃ *α2*∈ *qbs-Mx Y*.
g = (λ*r*::*real*. (*if* (*r* ∈ *S*) *then Inl* (*α1 r*) *else Inr* (*α2 r*)))))
by (*auto simp add*: *copair-qbs-Mx-def*)
consider *S* = {} | *S* = *UNIV* | *S* ≠ {} ∧ *S* ≠ *UNIV* **by** *auto*
then show *g* ∈ *copair-qbs-Mx2 X Y*
proof *cases*
assume *S* = {}
from *hs this* **have** ∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*)) **by** *simp*
then obtain *α1* **where** *h1*:*α1*∈ *qbs-Mx X* ∧ *g* = (λ*r*. *Inl* (*α1 r*)) **by** *auto*
have *qbs-space X* ≠ {}
using *qbs-empty-equiv h1*
by *auto*
then have (*qbs-space X* ≠ {} ∧ *qbs-space Y* = {}) ∨ (*qbs-space X* ≠ {} ∧
qbs-space Y ≠ {})
by *simp*
then show *g* ∈ *copair-qbs-Mx2 X Y*
proof
assume *qbs-space X* ≠ {} ∧ *qbs-space Y* = {}
then show *g* ∈ *copair-qbs-Mx2 X Y*
by(*simp add*: *copair-qbs-Mx2-def* ‹∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*))›)
next
assume *qbs-space X* ≠ {} ∧ *qbs-space Y* ≠ {}
then obtain *α2* **where** *α2* ∈ *qbs-Mx Y* **using** *qbs-empty-equiv* **by** *force*
define *S'* :: *real set*
where *S'* ≡ *UNIV*

define g' :: *real* $\Rightarrow$ $'a + 'b$
where $g' \equiv (\lambda r{::}real.\ (if\ (r \in S')\ then\ Inl\ (\alpha 1\ r)\ else\ Inr\ (\alpha 2\ r)))$
from ‹*qbs-space* $X \neq \{\} \wedge$ *qbs-space* $Y \neq \{\}$› *h1* ‹$\alpha 2 \in$ *qbs-Mx* Y›
have $g' \in$ *copair-qbs-Mx2* X Y
by(*force simp add*: S'*-def* g'*-def copair-qbs-Mx2-def*)
moreover have $g = g'$
using *h1* **by**(*simp add*: g'*-def* S'*-def*)
ultimately show *?thesis*
by *simp*
qed
next
assume $S =$ *UNIV*
from *hs this* **have** $\exists\ \alpha 2 \in$ *qbs-Mx* $Y.\ g = (\lambda r.\ Inr\ (\alpha 2\ r))$ **by** *simp*
then obtain $\alpha 2$ **where** *h2*:$\alpha 2 \in$ *qbs-Mx* $Y \wedge g = (\lambda r.\ Inr\ (\alpha 2\ r))$ **by** *auto*
have *qbs-space* $Y \neq \{\}$
using *qbs-empty-equiv h2*
by *auto*
then have (*qbs-space* $X = \{\} \wedge$ *qbs-space* $Y \neq \{\}$) $\vee$ (*qbs-space* $X \neq \{\} \wedge$ *qbs-space* $Y \neq \{\}$)
by *simp*
then show $g \in$ *copair-qbs-Mx2* X Y
proof
assume *qbs-space* $X = \{\} \wedge$ *qbs-space* $Y \neq \{\}$
then show *?thesis*
by(*simp add*: *copair-qbs-Mx2-def* ‹$\exists\ \alpha 2 \in$ *qbs-Mx* $Y.\ g = (\lambda r.\ Inr\ (\alpha 2\ r))$›)
next
assume *qbs-space* $X \neq \{\} \wedge$ *qbs-space* $Y \neq \{\}$
then obtain $\alpha 1$ **where** $\alpha 1 \in$ *qbs-Mx* X **using** *qbs-empty-equiv* **by** *force*
define S' :: *real set*
where $S' \equiv \{\}$
define g' :: *real* $\Rightarrow$ $'a + 'b$
where $g' \equiv (\lambda r{::}real.\ (if\ (r \in S')\ then\ Inl\ (\alpha 1\ r)\ else\ Inr\ (\alpha 2\ r)))$
from ‹*qbs-space* $X \neq \{\} \wedge$ *qbs-space* $Y \neq \{\}$› *h2* ‹$\alpha 1 \in$ *qbs-Mx* X›
have $g' \in$ *copair-qbs-Mx2* X Y
by(*force simp add*: S'*-def* g'*-def copair-qbs-Mx2-def*)
moreover have $g = g'$
using *h2* **by**(*simp add*: g'*-def* S'*-def*)
ultimately show *?thesis*
by *simp*
qed
next
assume $S \neq \{\} \wedge S \neq$ *UNIV*
then have
h: $\exists\ \alpha 1 \in$ *qbs-Mx* $X.$
$\exists\ \alpha 2 \in$ *qbs-Mx* $Y.$
$g = (\lambda r{::}real.\ (if\ (r \in S)\ then\ Inl\ (\alpha 1\ r)\ else\ Inr\ (\alpha 2\ r)))$
using *hs* **by** *simp*
then have *qbs-space* $X \neq \{\} \wedge$ *qbs-space* $Y \neq \{\}$
by (*metis empty-iff qbs-empty-equiv*)

thus *?thesis*
using *hs h* **by**(*auto simp add*: *copair-qbs-Mx2-def*)
qed

next
fix *g* :: *real* ⇒ *'a* + *'b*
assume *g* ∈ *copair-qbs-Mx2 X Y*
then have
h: *if qbs-space X* = {} ∧ *qbs-space Y* = {} *then False*
else if qbs-space X ≠ {} ∧ *qbs-space Y* = {} *then*
(∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*)))
else if qbs-space X = {} ∧ *qbs-space Y* ≠ {} *then*
(∃ *α2*∈ *qbs-Mx Y*. *g* = (λ*r*. *Inr* (*α2 r*)))
else
(∃ *S* ∈ *sets borel*. ∃ *α1*∈ *qbs-Mx X*. ∃ *α2*∈ *qbs-Mx Y*.
g = (λ*r*::*real*. (*if* (*r* ∈ *S*) *then Inl* (*α1 r*) *else Inr* (*α2 r*))))
by(*simp add*: *copair-qbs-Mx2-def*)
consider (*qbs-space X* = {} ∧ *qbs-space Y* = {}) |
(*qbs-space X* ≠ {} ∧ *qbs-space Y* = {}) |
(*qbs-space X* = {} ∧ *qbs-space Y* ≠ {}) |
(*qbs-space X* ≠ {} ∧ *qbs-space Y* ≠ {}) **by** *auto*
then show *g* ∈ *copair-qbs-Mx X Y*
proof *cases*
assume *qbs-space X* = {} ∧ *qbs-space Y* = {}
then show *?thesis*
using ‹*g* ∈ *copair-qbs-Mx2 X Y*› **by**(*simp add*: *copair-qbs-Mx2-def*)
next
assume *qbs-space X* ≠ {} ∧ *qbs-space Y* = {}
from *h this* **have** ∃ *α1*∈ *qbs-Mx X*. *g* = (λ*r*. *Inl* (*α1 r*)) **by** *simp*
thus *?thesis*
by(*auto simp add*: *copair-qbs-Mx-def*)
next
assume *qbs-space X* = {} ∧ *qbs-space Y* ≠ {}
from *h this* **have** ∃ *α2*∈ *qbs-Mx Y*. *g* = (λ*r*. *Inr* (*α2 r*)) **by** *simp*
thus *?thesis*
unfolding *copair-qbs-Mx-def*
by(*force simp add*: *copair-qbs-Mx-def*)
next
assume *qbs-space X* ≠ {} ∧ *qbs-space Y* ≠ {}
from *h this* **obtain** *S α1 α2* **where** *Sag*:
S ∈ *sets borel α1* ∈ *qbs-Mx X α2* ∈ *qbs-Mx Y g* = (λ*r*. *if r* ∈ *S then Inl* (*α1*
r) *else Inr* (*α2 r*))
by *auto*
consider *S* = {} | *S* = *UNIV* | *S* ≠ {} *S* ≠ *UNIV* **by** *auto*
then show *g* ∈ *copair-qbs-Mx X Y*
proof *cases*
assume *S* = {}
then have [*simp*]: (λ*r*. *if r* ∈ *S then Inl* (*α1 r*) *else Inr* (*α2 r*)) = (λ*r*. *Inr*

```
(α2 r))
        by simp
      show ?thesis
        using ‹α2 ∈ qbs-Mx Y› unfolding copair-qbs-Mx-def
        by(auto intro! : bexI[where x=UNIV] simp: Sag)
    next
      assume S = UNIV
      then have (λr. if r ∈ S then Inl (α1 r) else Inr (α2 r)) = (λr. Inl (α1 r))
        by simp
      then show ?thesis
        using Sag by(auto simp add: copair-qbs-Mx-def)
    next
      assume S ≠ {} S ≠ UNIV
      then show ?thesis
        using Sag by(auto simp add: copair-qbs-Mx-def)
    qed
  qed
qed

lemma
  shows copair-qbs-space: qbs-space (X ⨁_Q Y) = qbs-space X <+> qbs-space Y
(is ?goal1)
    and copair-qbs-Mx: qbs-Mx (X ⨁_Q Y) = copair-qbs-Mx X Y (is ?goal2)
proof −
  have copair-qbs-Mx X Y ⊆ UNIV → qbs-space X <+> qbs-space Y
  proof
    fix g
    assume g ∈ copair-qbs-Mx X Y
    then obtain S where hs:S∈ sets borel ∧
     (S = {}   ⟶ (∃ α1∈ qbs-Mx X. g = (λr. Inl (α1 r)))) ∧
     (S = UNIV ⟶ (∃ α2∈ qbs-Mx Y. g = (λr. Inr (α2 r)))) ∧
     ((S ≠ {} ∧ S ≠ UNIV) ⟶
        (∃ α1∈ qbs-Mx X.
         ∃ α2∈ qbs-Mx Y.
            g = (λr::real. (if (r ∈ S) then Inl (α1 r) else Inr (α2 r)))))
      by (auto simp add: copair-qbs-Mx-def)
    consider S = {} | S = UNIV | S ≠ {} ∧ S ≠ UNIV by auto
    then show g ∈ UNIV → qbs-space X <+> qbs-space Y
    proof cases
      assume S = {}
      then show ?thesis
        using hs qbs-Mx-to-X by auto
    next
      assume S = UNIV
      then show ?thesis
        using hs qbs-Mx-to-X by auto
    next
      assume S ≠ {} ∧ S ≠ UNIV
      then have ∃ α1∈ qbs-Mx X. ∃ α2∈ qbs-Mx Y.
```

```
        g = (λr::real. (if (r ∈ S) then Inl (α1 r) else Inr (α2 r))) using hs by
simp
    then show ?thesis
      by(auto dest: qbs-Mx-to-X)
   qed
  qed
  moreover have qbs-closed1 (copair-qbs-Mx X Y)
  proof(rule qbs-closed1I)
   fix g and f :: real ⇒ real
   assume g ∈ copair-qbs-Mx X Y and [measurable]: f ∈ borel →M borel
   then have g ∈ copair-qbs-Mx2 X Y using copair-qbs-Mx-equiv by auto
   consider (qbs-space X = {} ∧ qbs-space Y = {}) |
            (qbs-space X ≠ {} ∧ qbs-space Y = {}) |
            (qbs-space X = {} ∧ qbs-space Y ≠ {}) |
            (qbs-space X ≠ {} ∧ qbs-space Y ≠ {}) by auto
   then have g ∘ f ∈ copair-qbs-Mx2 X Y
   proof cases
    assume qbs-space X = {} ∧ qbs-space Y = {}
    then show ?thesis
    using ‹g ∈ copair-qbs-Mx2 X Y› qbs-empty-equiv by(simp add: copair-qbs-Mx2-def)
   next
    assume qbs-space X ≠ {} ∧ qbs-space Y = {}
    then obtain α1 where h1:α1∈ qbs-Mx X ∧ g = (λr. Inl (α1 r))
     using ‹g ∈ copair-qbs-Mx2 X Y› by(auto simp add: copair-qbs-Mx2-def)
    then have α1 ∘ f ∈ qbs-Mx X
     by auto
    moreover have g ∘ f = (λr. Inl ((α1 ∘ f) r))
     using h1 by auto
    ultimately show ?thesis
       using ‹qbs-space X ≠ {} ∧ qbs-space Y = {}› by(force simp add: co-
pair-qbs-Mx2-def)
   next
    assume (qbs-space X = {} ∧ qbs-space Y ≠ {})
    then obtain α2 where h2:α2∈ qbs-Mx Y ∧ g = (λr. Inr (α2 r))
     using ‹g ∈ copair-qbs-Mx2 X Y› by(auto simp add: copair-qbs-Mx2-def)
    then have α2 ∘ f ∈ qbs-Mx Y
     by auto
    moreover have g ∘ f = (λr. Inr ((α2 ∘ f) r))
     using h2 by auto
    ultimately show ?thesis
      using ‹(qbs-space X = {} ∧ qbs-space Y ≠ {})› by(force simp add: co-
pair-qbs-Mx2-def)
   next
    assume qbs-space X ≠ {} ∧ qbs-space Y ≠ {}
    then have ∃ S ∈ sets borel. ∃α1∈ qbs-Mx X. ∃α2∈ qbs-Mx Y.
       g = (λr::real. (if (r ∈ S) then Inl (α1 r) else Inr (α2 r)))
     using ‹g ∈ copair-qbs-Mx2 X Y› by(simp add: copair-qbs-Mx2-def)
    then show ?thesis
    proof safe
```

fix S $\alpha 1$ $\alpha 2$
assume [*measurable*]:$S \in$ *sets borel* **and** $\alpha 1 \in$ *qbs-Mx X* $\alpha 2 \in$ *qbs-Mx Y*
$g = (\lambda r.$ *if* $r \in S$ *then Inl* $(\alpha 1\ r)$ *else Inr* $(\alpha 2\ r))$
have $f -` S \in$ *sets borel*
using ‹$S \in$ *sets borel*› ‹$f \in$ *borel-measurable borel*› *measurable-sets-borel*
by *blast*
moreover have $\alpha 1 \circ f \in$ *qbs-Mx X*
using ‹$\alpha 1 \in$ *qbs-Mx X*› **by**(*auto simp add*: *qbs-closed1-def*)
moreover have $\alpha 2 \circ f \in$ *qbs-Mx Y*
using ‹$\alpha 2 \in$ *qbs-Mx Y*› **by**(*auto simp add*: *qbs-closed1-def*)
moreover have $(\lambda r.$ *if* $r \in S$ *then Inl* $(\alpha 1\ r)$ *else Inr* $(\alpha 2\ r)) \circ f = (\lambda r.$ *if* $r \in f -` S$ *then Inl* $((\alpha 1 \circ f)\ r)$ *else Inr* $((\alpha 2 \circ f)\ r))$
by *auto*
ultimately show $(\lambda r.$ *if* $r \in S$ *then Inl* $(\alpha 1\ r)$ *else Inr* $(\alpha 2\ r)) \circ f \in$ *copair-qbs-Mx2 X Y*
using ‹*qbs-space X* $\neq \{\} \wedge$ *qbs-space Y* $\neq \{\}$› **by**(*force simp add*: *copair-qbs-Mx2-def*)
qed
qed
thus $g \circ f \in$ *copair-qbs-Mx X Y*
using *copair-qbs-Mx-equiv* **by** *auto*
qed
moreover have *qbs-closed2* (*qbs-space X <+> qbs-space Y*) (*copair-qbs-Mx X Y*)
proof(*rule qbs-closed2I*)
fix y
assume $y \in$ *qbs-space X <+> qbs-space Y*
then consider $y \in$ *Inl* ‘ (*qbs-space X*) | $y \in$ *Inr* ‘ (*qbs-space Y*)
by *auto*
thus $(\lambda r.\ y) \in$ *copair-qbs-Mx X Y*
proof *cases*
case *1*
then obtain x **where** x: $y =$ *Inl x* $x \in$ *qbs-space X*
by *auto*
define $\alpha 1$ :: *real* $\Rightarrow$ - **where** $\alpha 1 \equiv (\lambda r.\ x)$
have $\alpha 1 \in$ *qbs-Mx X* **using** ‹$x \in$ *qbs-space X*› *qbs-decomp*
by(*force simp add*: *qbs-closed2-def* $\alpha 1$-*def*)
moreover have $(\lambda r.$ *Inl* $x) = (\lambda l.$ *Inl* $(\alpha 1\ l))$ **by** (*simp add*: $\alpha 1$-*def*)
moreover have $\{\} \in$ *sets borel* **by** *auto*
ultimately show $(\lambda r.\ y) \in$ *copair-qbs-Mx X Y*
by(*auto simp add*: *copair-qbs-Mx-def x*)
next
case *2*
then obtain x **where** x: $y =$ *Inr x* $x \in$ *qbs-space Y*
by *auto*
define $\alpha 2$ :: *real* $\Rightarrow$ - **where** $\alpha 2 \equiv (\lambda r.\ x)$
have $\alpha 2 \in$ *qbs-Mx Y* **using** ‹$x \in$ *qbs-space Y*› *qbs-decomp*
by(*force simp add*: *qbs-closed2-def* $\alpha 2$-*def*)
moreover have $(\lambda r.$ *Inr* $x) = (\lambda l.$ *Inr* $(\alpha 2\ l))$ **by** (*simp add*: $\alpha 2$-*def*)

```
    moreover have UNIV ∈ sets borel by auto
    ultimately show (λr. y) ∈ copair-qbs-Mx X Y
      unfolding copair-qbs-Mx-def
      by(auto intro!: bexI[where x=UNIV] simp: x)
  qed
 qed
 moreover have qbs-closed3 (copair-qbs-Mx X Y)
 proof(safe intro!: qbs-closed3I)
  fix P :: real ⇒ nat
  fix Fi :: nat ⇒ real ⇒- + -
  assume P ∈ borel →M count-space UNIV
      ∀ i. Fi i ∈ copair-qbs-Mx X Y
  then have ∀ i. Fi i ∈ copair-qbs-Mx2 X Y using copair-qbs-Mx-equiv by blast
  consider (qbs-space X = {} ∧ qbs-space Y = {}) |
           (qbs-space X ≠ {} ∧ qbs-space Y = {}) |
           (qbs-space X = {} ∧ qbs-space Y ≠ {}) |
           (qbs-space X ≠ {} ∧ qbs-space Y ≠ {}) by auto
  then have (λr. Fi (P r) r) ∈ copair-qbs-Mx2 X Y
  proof cases
    assume qbs-space X = {} ∧ qbs-space Y = {}
    then show ?thesis
      using ‹∀ i. Fi i ∈ copair-qbs-Mx2 X Y› qbs-empty-equiv
      by(simp add: copair-qbs-Mx2-def)
  next
    assume qbs-space X ≠ {} ∧ qbs-space Y = {}
    then have ∀ i. ∃ αi. αi ∈ qbs-Mx X ∧ Fi i = (λr. Inl (αi r))
    using ‹∀ i. Fi i ∈ copair-qbs-Mx2 X Y› by(auto simp add: copair-qbs-Mx2-def)
    then have ∃ α1. ∀ i. α1 i ∈ qbs-Mx X ∧ Fi i = (λr. Inl (α1 i r))
      by(rule choice)
    then obtain α1 :: nat ⇒ real ⇒ -
      where h1: ∀ i. α1 i ∈ qbs-Mx X ∧ Fi i = (λr. Inl (α1 i r)) by auto
    define β :: real ⇒ - where β ≡ (λr. α1 (P r) r)
    from ‹P ∈ borel →M count-space UNIV› h1
    have β ∈ qbs-Mx X by (simp add: β-def)
    moreover have (λr. Fi (P r) r) = (λr. Inl (β r))
      using h1 by(simp add: β-def)
    ultimately show ?thesis
      using ‹qbs-space X ≠ {} ∧ qbs-space Y = {}› by (auto simp add: co-
pair-qbs-Mx2-def)
  next
    assume qbs-space X = {} ∧ qbs-space Y ≠ {}
    then have ∀ i. ∃ αi. αi ∈ qbs-Mx Y ∧ Fi i = (λr. Inr (αi r))
    using ‹∀ i. Fi i ∈ copair-qbs-Mx2 X Y› by(auto simp add: copair-qbs-Mx2-def)
    then have ∃ α2. ∀ i. α2 i ∈ qbs-Mx Y ∧ Fi i = (λr. Inr (α2 i r))
      by(rule choice)
    then obtain α2 :: nat ⇒ real ⇒ -
      where h2: ∀ i. α2 i ∈ qbs-Mx Y ∧ Fi i = (λr. Inr (α2 i r)) by auto
    define β :: real ⇒ - where β ≡ (λr. α2 (P r) r)
    from ‹P ∈ borel →M count-space UNIV› h2
```

have $\beta \in$ *qbs-Mx Y* **by**(*simp add*: β*-def*)
moreover have $(\lambda r.\ Fi\ (P\ r)\ r) = (\lambda r.\ Inr\ (\beta\ r))$
using *h2* **by**(*simp add*: β*-def*)
ultimately show *?thesis*
using ‹*qbs-space X* $=$ $\{\}$ $\wedge$ *qbs-space Y* $\neq$ $\{\}$› **by** (*auto simp add*: *copair-qbs-Mx2-def*)
next
assume *qbs-space X* $\neq$ $\{\}$ $\wedge$ *qbs-space Y* $\neq$ $\{\}$
then have $\forall i.\ \exists Si.\ Si \in sets\ borel \wedge (\exists \alpha 1i \in qbs\text{-}Mx\ X.\ \exists \alpha 2i \in qbs\text{-}Mx\ Y.$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in Si)\ then\ Inl\ (\alpha 1i\ r)\ else\ Inr\ (\alpha 2i\ r))))$
using ‹$\forall i.\ Fi\ i \in copair\text{-}qbs\text{-}Mx2\ X\ Y$› **by** (*auto simp add*: *copair-qbs-Mx2-def*)
then have $\exists S.\ \forall i.\ S\ i \in sets\ borel \wedge (\exists \alpha 1i \in qbs\text{-}Mx\ X.\ \exists \alpha 2i \in qbs\text{-}Mx\ Y.$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1i\ r)\ else\ Inr\ (\alpha 2i\ r))))$
by(*rule choice*)
then obtain $S :: nat \Rightarrow real\ set$
where *hs* :$\forall i.\ S\ i \in sets\ borel \wedge (\exists \alpha 1i \in qbs\text{-}Mx\ X.\ \exists \alpha 2i \in qbs\text{-}Mx\ Y.$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1i\ r)\ else\ Inr\ (\alpha 2i\ r))))$
by *auto*
then have $\forall i.\ \exists \alpha 1i.\ \alpha 1i \in qbs\text{-}Mx\ X \wedge (\exists \alpha 2i \in qbs\text{-}Mx\ Y.$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1i\ r)\ else\ Inr\ (\alpha 2i\ r))))$
by *blast*
then have $\exists \alpha 1.\ \forall i.\ \alpha 1\ i \in qbs\text{-}Mx\ X \wedge (\exists \alpha 2i \in qbs\text{-}Mx\ Y.$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1\ i\ r)\ else\ Inr\ (\alpha 2i\ r))))$
by(*rule choice*)
then obtain $\alpha 1$ **where** *h1*: $\forall i.\ \alpha 1\ i \in qbs\text{-}Mx\ X \wedge (\exists \alpha 2i \in qbs\text{-}Mx\ Y.$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1\ i\ r)\ else\ Inr\ (\alpha 2i\ r))))$
by *auto*
define $\beta 1 :: real \Rightarrow$ - **where** $\beta 1 \equiv (\lambda r.\ \alpha 1\ (P\ r)\ r)$
from ‹$P \in borel \rightarrow_M count\text{-}space\ UNIV$› *h1*
have $\beta 1 \in qbs\text{-}Mx\ X$ **by**(*simp add*: $\beta 1$*-def*)
from *h1* **have** $\forall i.\ \exists \alpha 2i.\ \alpha 2i \in qbs\text{-}Mx\ Y \wedge$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1\ i\ r)\ else\ Inr\ (\alpha 2i\ r)))$
by *auto*
then have $\exists \alpha 2.\ \forall i.\ \alpha 2\ i \in qbs\text{-}Mx\ Y \wedge$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1\ i\ r)\ else\ Inr\ (\alpha 2\ i\ r)))$
by(*rule choice*)
then obtain $\alpha 2$
where *h2*: $\forall i.\ \alpha 2\ i \in qbs\text{-}Mx\ Y \wedge$
$Fi\ i = (\lambda r{::}real.\ (if\ (r \in S\ i)\ then\ Inl\ (\alpha 1\ i\ r)\ else\ Inr\ (\alpha 2\ i\ r)))$
by *auto*
define $\beta 2 :: real \Rightarrow$ - **where** $\beta 2 \equiv (\lambda r.\ \alpha 2\ (P\ r)\ r)$
from ‹$P \in borel \rightarrow_M count\text{-}space\ UNIV$› *h2*
have $\beta 2 \in qbs\text{-}Mx\ Y$ **by**(*simp add*: $\beta 2$*-def*)
define $A :: nat \Rightarrow real\ set$ **where** $A \equiv (\lambda i.\ S\ i \cap P -` \{i\})$
have [*measurable*]:$\bigwedge i.\ A\ i \in sets\ borel$
using *A-def hs measurable-separate*[*OF* ‹$P \in borel \rightarrow_M count\text{-}space\ UNIV$›]
by *blast*
define $S' :: real\ set$ **where** $S' \equiv \{r.\ r \in S\ (P\ r)\}$
have $S' = (\bigcup i{::}nat.\ A\ i)$

by(*auto simp add*: *S′-def A-def*)
hence $S' \in$ *sets borel* **by** *auto*
from *h2* **have** $(\lambda r.\ Fi\ (P\ r)\ r) = (\lambda r.\ (if\ r \in S'\ then\ Inl\ (\beta 1\ \ r)$
else Inr $(\beta 2\ r)))$
by(*auto simp add*: *β1-def β2-def S′-def*)
thus $(\lambda r.\ Fi\ (P\ r)\ r) \in$ *copair-qbs-Mx2 X Y*
using ‹*qbs-space X* $\neq \{\} \wedge$ *qbs-space Y* $\neq \{\}$› ‹$S' \in$ *sets borel*› ‹$\beta 1 \in$ *qbs-Mx X*› ‹$\beta 2 \in$ *qbs-Mx Y*›
by(*auto simp add*: *copair-qbs-Mx2-def*)
qed
thus $(\lambda r.\ Fi\ (P\ r)\ r) \in$ *copair-qbs-Mx X Y*
using *copair-qbs-Mx-equiv* **by** *auto*
qed
ultimately have *Rep-quasi-borel* (*copair-qbs X Y*) = (*qbs-space X* <+> *qbs-space Y*, *copair-qbs-Mx X Y*)
unfolding *copair-qbs-def* **by**(*auto intro*!: *Abs-quasi-borel-inverse*)
thus *?goal1 ?goal2*
by(*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

lemma *copair-qbs-MxD*:
assumes $g \in$ *qbs-Mx* $(X \bigoplus_Q Y)$
and $\bigwedge\alpha.\ \alpha \in$ *qbs-Mx X* $\Longrightarrow g = (\lambda r.\ Inl\ (\alpha\ r)) \Longrightarrow P\ g$
and $\bigwedge\beta.\ \beta \in$ *qbs-Mx Y* $\Longrightarrow g = (\lambda r.\ Inr\ (\beta\ r)) \Longrightarrow P\ g$
and $\bigwedge S\ \alpha\ \beta.\ (S :: real\ set) \in$ *sets borel* $\Longrightarrow S \neq \{\} \Longrightarrow S \neq UNIV \Longrightarrow \alpha \in$ *qbs-Mx X* $\Longrightarrow \beta \in$ *qbs-Mx Y* $\Longrightarrow g = (\lambda r.\ if\ r \in S\ then\ Inl\ (\alpha\ r)\ else\ Inr\ (\beta\ r)) \Longrightarrow P\ g$
shows *P g*
using *assms* **by**(*fastforce simp*: *copair-qbs-Mx copair-qbs-Mx-def*)

3.1.8 Product Spaces

definition *PiQ* :: *'a set* ⇒ (*'a* ⇒ *'b quasi-borel*) ⇒ (*'a* ⇒ *'b*) *quasi-borel* **where** *PiQ I X* ≡ *Abs-quasi-borel* $(\Pi_E\ i{\in}I.\ qbs\text{-}space\ (X\ i), \{\alpha.\ \forall i.\ (i \in I \longrightarrow (\lambda r.\ \alpha\ r\ i) \in qbs\text{-}Mx\ (X\ i)) \wedge (i \notin I \longrightarrow (\lambda r.\ \alpha\ r\ i) = (\lambda r.\ undefined))\})$

syntax
-PiQ :: *pttrn* ⇒ *'i set* ⇒ *'a quasi-borel* ⇒ (*'i* => *'a*) *quasi-borel* ((*3*Π_Q *-*∈*-./ -*) *10*)
syntax-consts
-PiQ == *PiQ*
translations
$\Pi_Q\ x{\in}I.\ X$ == *CONST PiQ I* $(\lambda x.\ X)$

lemma
shows *PiQ-space*: *qbs-space* (*PiQ I X*) = $(\Pi_E\ i{\in}I.\ qbs\text{-}space\ (X\ i))$ (**is** *?goal1*)
and *PiQ-Mx*: *qbs-Mx* (*PiQ I X*) = $\{\alpha.\ \forall i.\ (i \in I \longrightarrow (\lambda r.\ \alpha\ r\ i) \in qbs\text{-}Mx\ (X\ i)) \wedge (i \notin I \longrightarrow (\lambda r.\ \alpha\ r\ i) = (\lambda r.\ undefined))\}$ (**is** *- = ?Mx*)
proof −

```
  have ?Mx ⊆ UNIV → (Π_E i∈I. qbs-space (X i))
    using qbs-Mx-to-X[of - X -] by auto metis
  moreover have qbs-closed1 ?Mx
  proof(safe intro!: qbs-closed1I)
    fix α i and f :: real ⇒ real
    assume h[measurable]:∀ i. (i ∈ I ⟶ (λr. α r i) ∈ qbs-Mx (X i)) ∧ (i ∉ I ⟶
(λr. α r i) = (λr. undefined))
                        f ∈ borel →_M borel
    show (λr. (α ∘ f) r i) ∈ qbs-Mx (X i) if i:i ∈ I
    proof −
      have (λr. α r i) ∘ f ∈ qbs-Mx (X i)
        using h i by auto
      thus (λr. (α ∘ f) r i) ∈ qbs-Mx (X i)
        by(simp add: comp-def)
    qed
    show i ∉ I ⟹ (λr. (α ∘ f) r i) = (λr. undefined)
      by (metis comp-apply h(1))
  qed
  moreover have qbs-closed2 (Π_E i∈I. qbs-space (X i)) ?Mx
    by(rule qbs-closed2I) (auto simp: PiE-def extensional-def Pi-def)
  moreover have qbs-closed3 ?Mx
  proof(rule qbs-closed3I)
    fix P :: real ⇒ nat and Fi
    assume h:P ∈ borel →_M count-space UNIV
           ⋀i::nat. Fi i ∈ ?Mx
    show (λr. Fi (P r) r) ∈ ?Mx
    proof safe
      fix i
      assume hi:i ∈ I
      then show (λr. Fi (P r) r i) ∈ qbs-Mx (X i)
        using h qbs-closed3-dest[OF h(1),of λj r. Fi j r i]
        by auto
    next
      show ⋀i. i ∉ I ⟹ (λr. Fi (P r) r i) = (λr. undefined)
        using h by auto meson
    qed
  qed
  ultimately have Rep-quasi-borel (PiQ I X) = (Π_E i∈I. qbs-space (X i), ?Mx)
    by(auto intro!: Abs-quasi-borel-inverse is-quasi-borel-intro simp: PiQ-def)
  thus ?goal1 qbs-Mx (PiQ I X) = ?Mx
    by(simp-all add: qbs-space-def qbs-Mx-def)
qed

lemma prod-qbs-MxI:
  assumes ⋀i. i ∈ I ⟹ (λr. α r i) ∈ qbs-Mx (X i)
     and ⋀i. i ∉ I ⟹ (λr. α r i) = (λr. undefined)
    shows α ∈ qbs-Mx (PiQ I X)
  using assms by(auto simp: PiQ-Mx)
```

lemma *prod-qbs-MxD*:
assumes $\alpha \in$ *qbs-Mx* (*PiQ I X*)
shows $\bigwedge i.\ i \in I \Longrightarrow (\lambda r.\ \alpha\ r\ i) \in$ *qbs-Mx* (*X i*)
and $\bigwedge i.\ i \notin I \Longrightarrow (\lambda r.\ \alpha\ r\ i) = (\lambda r.\ undefined)$
and $\bigwedge i\ r.\ i \notin I \Longrightarrow \alpha\ r\ i = undefined$
using *assms* **by**(*auto simp*: *PiQ-Mx dest*: *fun-cong*[**where** g=$(\lambda r.\ undefined)$])

lemma *PiQ-eqI*:
assumes $\bigwedge i.\ i \in I \Longrightarrow X\ i = Y\ i$
shows *PiQ I X* = *PiQ I Y*
by(*auto intro*!: *qbs-eqI simp*: *PiQ-Mx assms*)

lemma *PiQ-empty*: *qbs-space* (*PiQ* {} *X*) = {$\lambda i.\ undefined$}
by(*auto simp*: *PiQ-space*)

lemma *PiQ-empty-Mx*: *qbs-Mx* (*PiQ* {} *X*) = {$\lambda r\ i.\ undefined$}
by(*auto simp*: *PiQ-Mx*) *meson*

3.1.9 Coproduct Spaces

definition *coPiQ-Mx* :: ['*a set*, '*a* ⇒ '*b quasi-borel*] ⇒ (*real* ⇒ '*a* × '*b*) *set* **where** *coPiQ-Mx I X* ≡ { $\lambda r.$ (*f r*, α (*f r*) *r*) |*f* α. *f* ∈ *borel* →$_M$ *count-space I* ∧ (∀ *i*∈*range f*. α *i* ∈ *qbs-Mx* (*X i*))}

definition *coPiQ-Mx'* :: ['*a set*, '*a* ⇒ '*b quasi-borel*] ⇒ (*real* ⇒ '*a* × '*b*) *set* **where** *coPiQ-Mx' I X* ≡ { $\lambda r.$ (*f r*, α (*f r*) *r*) |*f* α. *f* ∈ *borel* →$_M$ *count-space I* ∧ (∀ *i*. (*i* ∈ *range f* ∨ *qbs-space* (*X i*) ≠ {}) ⟶ α *i* ∈ *qbs-Mx* (*X i*))}

lemma *coPiQ-Mx-eq*:
coPiQ-Mx I X = *coPiQ-Mx' I X*
proof *safe*
fix α
assume α ∈ *coPiQ-Mx I X*
then obtain *f* β **where** *hfb*:
f ∈ *borel* →$_M$ *count-space I* $\bigwedge i.\ i \in range\ f \Longrightarrow \beta\ i \in$ *qbs-Mx* (*X i*) $\alpha = (\lambda r.$ (*f r*, β (*f r*) *r*))
unfolding *coPiQ-Mx-def* **by** *blast*
define β' **where** $\beta' \equiv (\lambda i.$ *if i* ∈ *range f then* β *i*
else if qbs-space (*X i*) ≠ {} *then* (*SOME* $\gamma.\ \gamma \in$ *qbs-Mx* (*X i*))
else β *i*)
have *1*:$\alpha = (\lambda r.$ (*f r*, β' (*f r*) *r*))
by(*simp add*: *hfb*(*3*) β'-*def*)
have *2*:$\bigwedge i.$ *qbs-space* (*X i*) ≠ {} ⟹ β' *i* ∈ *qbs-Mx* (*X i*)
proof −
fix *i*
assume *hne*:*qbs-space* (*X i*) ≠ {}
then obtain *x* **where** *x* ∈ *qbs-space* (*X i*) **by** *auto*
hence $(\lambda r.\ x) \in$ *qbs-Mx* (*X i*) **by** *auto*

thus $\beta'\ i \in$ *qbs-Mx* $(X\ i)$
by(*cases* $i \in$ *range* f) (*auto simp*: β'*-def hfb*(*2*) *hne intro*!: *someI2*[**where** $a{=}\lambda r.\ x$])
qed
show $\alpha \in$ *coPiQ-Mx'* $I\ X$
using *hfb*(*1*,*2*) *1 2* β'*-def* **by**(*auto simp*: *coPiQ-Mx'-def intro*!: *exI*[**where** $x{=}f$] *exI*[**where** $x{=}\beta'$])
next
fix α
assume $\alpha \in$ *coPiQ-Mx'* $I\ X$
then obtain $f\ \beta$ **where** *hfb*:
$f \in$ *borel* $\rightarrow_M$ *count-space* I $\bigwedge i.$ *qbs-space* $(X\ i) \neq \{\} \Longrightarrow \beta\ i \in$ *qbs-Mx* $(X\ i)$
$\bigwedge i.\ i \in$ *range* $f \Longrightarrow \beta\ i \in$ *qbs-Mx* $(X\ i)$ $\alpha = (\lambda r.\ (f\ r, \beta\ (f\ r)\ r))$
unfolding *coPiQ-Mx'-def* **by** *blast*
show $\alpha \in$ *coPiQ-Mx* $I\ X$
by(*auto simp*: *hfb*(*4*) *coPiQ-Mx-def intro*!: *hfb*(*1*) *hfb*(*3*))
qed

definition *coPiQ* :: [$'a$ *set*, $'a \Rightarrow$ $'b$ *quasi-borel*] $\Rightarrow$ ($'a \times 'b$) *quasi-borel* **where**
coPiQ $I\ X \equiv$ *Abs-quasi-borel* (*SIGMA* $i{:}I.$ *qbs-space* $(X\ i)$, *coPiQ-Mx* $I\ X$)

syntax
-coPiQ :: *pttrn* $\Rightarrow$ $'i$ *set* $\Rightarrow$ $'a$ *quasi-borel* $\Rightarrow$ ($'i \times 'a$) *quasi-borel* ((*3*Π_Q *-*$\in$*-./ -*) *10*)
syntax-consts
-coPiQ $\rightleftharpoons$ *coPiQ*
translations
$\Pi_Q\ x{\in}I.\ X \rightleftharpoons$ *CONST coPiQ* $I\ (\lambda x.\ X)$

lemma
shows *coPiQ-space*: *qbs-space* $($*coPiQ* $I\ X) =$ (*SIGMA* $i{:}I.$ *qbs-space* $(X\ i)$) (**is** *?goal1*)
and *coPiQ-Mx*: *qbs-Mx* $($*coPiQ* $I\ X) =$ *coPiQ-Mx* $I\ X$ (**is** *?goal2*)
proof −
have *coPiQ-Mx* $I\ X \subseteq$ *UNIV* $\rightarrow$ (*SIGMA* $i{:}I.$ *qbs-space* $(X\ i)$)
by(*fastforce simp*: *coPiQ-Mx-def dest*: *measurable-space qbs-Mx-to-X*)
moreover have *qbs-closed1* $($*coPiQ-Mx* $I\ X)$
proof(*rule qbs-closed1I*)
fix α **and** f :: *real* $\Rightarrow$ *real*
assume $\alpha \in$ *coPiQ-Mx* $I\ X$
and *1*[*measurable*]: $f \in$ *borel* $\rightarrow_M$ *borel*
then obtain $\beta\ g$ **where** *ha*:
$\bigwedge i.\ i \in$ *range* $g \Longrightarrow \beta\ i \in$ *qbs-Mx* $(X\ i)$ $\alpha = (\lambda r.\ (g\ r, \beta\ (g\ r)\ r))$ **and** [*measurable*]:$g \in$ *borel* $\rightarrow_M$ *count-space* I
by(*fastforce simp*: *coPiQ-Mx-def*)
then have $\bigwedge i.\ i \in$ *range* $g \Longrightarrow \beta\ i \circ f \in$ *qbs-Mx* $(X\ i)$
by *simp*
thus $\alpha \circ f \in$ *coPiQ-Mx* $I\ X$
unfolding *coPiQ-Mx-def* **by** (*auto intro*!: *exI*[**where** $x{=}g \circ f$] *exI*[**where**

x=$\lambda i.\ \beta\ i \circ f$] *simp*: *ha*(*2*))
qed
moreover have *qbs-closed2* (*SIGMA i*:*I*. *qbs-space* (*X i*)) (*coPiQ-Mx I X*)
proof(*safe intro*!: *qbs-closed2I*)
fix *i x*
assume $i \in I\ x \in$ *qbs-space* (*X i*)
then show ($\lambda r.\ (i,x)$) $\in$ *coPiQ-Mx I X*
by(*auto simp*: *coPiQ-Mx-def intro*!: *exI*[**where** x=$\lambda r.\ i$])
qed
moreover have *qbs-closed3* (*coPiQ-Mx I X*)
proof(*rule qbs-closed3I*)
fix *P* :: *real* $\Rightarrow$ *nat* **and** *Fi*
assume *h*[*measurable*]:*P* $\in$ *borel* $\rightarrow_M$ *count-space UNIV*
$\bigwedge i$ :: *nat*. *Fi i* $\in$ *coPiQ-Mx I X*
then have $\forall\, i.\ \exists\, fi\ \alpha i.$ *Fi i* = ($\lambda r.$ (*fi r*, αi (*fi r*) *r*)) $\wedge$ *fi* $\in$ *borel* $\rightarrow_M$ *count-space I* $\wedge$ ($\forall\, j.$ ($j \in$ *range fi* $\vee$ *qbs-space* (*X j*) $\neq$ {}) $\longrightarrow$ $\alpha i\ j \in$ *qbs-Mx* (*X j*))
by(*auto simp*: *coPiQ-Mx-eq coPiQ-Mx*′*-def*)
then obtain *fi* **where**
$\forall\, i.\ \exists \alpha i.$ *Fi i* = ($\lambda r.$ (*fi i r*, αi (*fi i r*) *r*)) $\wedge$ *fi i* $\in$ *borel* $\rightarrow_M$ *count-space I* $\wedge$ ($\forall\, j.$ ($j \in$ *range* (*fi i*) $\vee$ *qbs-space* (*X j*) $\neq$ {}) $\longrightarrow$ $\alpha i\ j \in$ *qbs-Mx* (*X j*))
by(*fastforce intro*!: *choice*)
then obtain αi **where**
$\forall\, i.$ *Fi i* = ($\lambda r.$ (*fi i r*, $\alpha i\ i$ (*fi i r*) *r*)) $\wedge$ *fi i* $\in$ *borel* $\rightarrow_M$ *count-space I* $\wedge$ ($\forall\, j.$ ($j \in$ *range* (*fi i*) $\vee$ *qbs-space* (*X j*) $\neq$ {}) $\longrightarrow$ $\alpha i\ i\ j \in$ *qbs-Mx* (*X j*))
by(*fastforce intro*!: *choice*)
then have *hf*[*measurable*]:
$\bigwedge i.$ *Fi i* = ($\lambda r.$ (*fi i r*, $\alpha i\ i$ (*fi i r*) *r*)) $\bigwedge i.$ *fi i* $\in$ *borel* $\rightarrow_M$ *count-space I* $\bigwedge i\ j.\ j \in$ *range* (*fi i*) $\Longrightarrow$ $\alpha i\ i\ j \in$ *qbs-Mx* (*X j*) $\bigwedge i\ j.$ *qbs-space* (*X j*) $\neq$ {} $\Longrightarrow$ $\alpha i\ i\ j \in$ *qbs-Mx* (*X j*)
by *auto*

define f' **where** $f' \equiv (\lambda r.\ fi\ (P\ r)\ r)$
define α' **where** $\alpha' \equiv (\lambda i\ r.\ \alpha i\ (P\ r)\ i\ r)$
have *1*:($\lambda r.$ *Fi* (*P r*) *r*) = ($\lambda r.$ (f' *r*, α' (f' *r*) *r*))
by(*simp add*: α'*-def f*′*-def hf*)
have $f' \in$ *borel* $\rightarrow_M$ *count-space I*
by(*simp add*: f'*-def*)
moreover have $\bigwedge i.\ i \in$ *range* $f' \Longrightarrow \alpha'\ i \in$ *qbs-Mx* (*X i*)
proof −
fix *i*
assume *hi*:*i* $\in$ *range* f'
then obtain *r* **where** *hr*: *i* = *fi* (*P r*) *r* **by**(*auto simp*: f'*-def*)
hence *i* $\in$ *range* (*fi* (*P r*)) **by** *simp*
hence αi (*P r*) *i* $\in$ *qbs-Mx* (*X i*) **by**(*simp add*: *hf*)
hence *qbs-space* (*X i*) $\neq$ {}
by(*auto simp*: *qbs-empty-equiv*)
hence $\bigwedge j.\ \alpha i\ j\ i \in$ *qbs-Mx* (*X i*)
by(*simp add*: *hf*(*4*))
then show $\alpha'\ i \in$ *qbs-Mx* (*X i*)

by(*auto simp*: α'-*def h*(*1*) *intro*!: *qbs-closed3-dest*[*of P* $\lambda j.\ \alpha i\ j\ i$])
qed
ultimately show ($\lambda r.$ *Fi* (*P r*) *r*) ∈ *coPiQ-Mx I X*
by(*auto simp*: *1 coPiQ-Mx-def intro*!: *exI*[**where** *x*=*f'*])
qed
ultimately have *Rep-quasi-borel* (*coPiQ I X*) = (*SIGMA i*:*I*. *qbs-space* (*X i*), *coPiQ-Mx I X*)
unfolding *coPiQ-def* **by**(*fastforce intro*!: *Abs-quasi-borel-inverse*)
thus *?goal1 ?goal2*
by(*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

lemma *coPiQ-MxI*:
assumes *f* ∈ *borel* →$_M$ *count-space I*
and ⋀*i*. *i* ∈ *range f* ⟹ α *i* ∈ *qbs-Mx* (*X i*)
shows ($\lambda r.$ (*f r*, α (*f r*) *r*)) ∈ *qbs-Mx* (*coPiQ I X*)
using *assms* **unfolding** *coPiQ-Mx-def coPiQ-Mx* **by** *blast*

lemma *coPiQ-eqI*:
assumes ⋀*i*. *i* ∈ *I* ⟹ *X i* = *Y i*
shows *coPiQ I X* = *coPiQ I Y*
using *assms* **by**(*auto intro*!: *qbs-eqI simp*: *coPiQ-Mx coPiQ-Mx-def*) (*metis UNIV-I measurable-space space-borel space-count-space*)+

3.1.10 List Spaces

We define the quasi-Borel spaces on list using the following isomorphism.

$$List(X) \cong \coprod_{n\in\mathbb{N}} \prod_{0\le i<n} X$$

definition *list-nil* :: *nat* × (*nat* ⇒ 'a) **where**
list-nil ≡ (*0*, $\lambda n.$ *undefined*)
definition *list-cons* :: ['a, *nat* × (*nat* ⇒ 'a)] ⇒ *nat* × (*nat* ⇒ 'a) **where**
list-cons x l ≡ (*Suc* (*fst l*), ($\lambda n.$ *if n* = *0 then x else* (*snd l*) (*n* − *1*)))

fun *from-list* :: 'a *list* ⇒ *nat* × (*nat* ⇒ 'a) **where**
from-list [] = *list-nil* |
from-list (*a*#*l*) = *list-cons a* (*from-list l*)

fun *to-list'* :: *nat* ⇒ (*nat* ⇒ 'a) ⇒ 'a *list* **where**
to-list' *0* - = [] |
to-list' (*Suc n*) *f* = *f 0* # *to-list' n* ($\lambda n.$ *f* (*Suc n*))

definition *to-list* :: *nat* × (*nat* ⇒ 'a) ⇒ 'a *list* **where**
to-list ≡ *case-prod to-list'*

lemma *inj-on-to-list*: *inj-on* (*to-list* :: *nat* × (*nat* ⇒ 'a) ⇒ 'a *list*) (*SIGMA n*:*UNIV*. *PiE* {..<*n*} *A*)

```
proof(safe intro!: inj-onI)
  fix n m and x y :: nat ⇒ 'a
  assume h:to-list (n, x) = to-list (m, y)
  have 1:⋀a. length (to-list (n, a)) = n for n
    by(induction n) (auto simp: to-list-def)
  show n:n = m
    using 1 arg-cong[OF h,of length] by metis
  show x ∈ PiE {..<n} A ⟹ y ∈ PiE {..<m} A ⟹ x = y
    using h unfolding n
  proof(induction m arbitrary: x y A)
    case ih:(Suc m)
    then have to-list (m, (λn. x (Suc n))) = to-list (m, (λn. y (Suc n)))
      by(auto simp: to-list-def)
    hence 1:(λn. x (Suc n)) = (λn. y (Suc n))
      using ih(2−4) by(intro ih(1)[of - λn. A (Suc n)]) auto
    show ?case
    proof
      fix n
      show x n = y n
      proof(cases n)
        assume n = 0
        then show x n = y n
          using ih(4) by(auto simp: to-list-def)
      qed(use fun-cong[OF 1] in auto)
    qed
  qed(auto simp: to-list-def)
qed
```

Definition

```
definition list-qbs :: 'a quasi-borel ⇒ 'a list quasi-borel where
list-qbs X ≡ map-qbs to-list (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X)

definition list-head :: nat × (nat ⇒ 'a) ⇒ 'a where
list-head l = snd l 0
definition list-tail :: nat × (nat ⇒ 'a) ⇒ nat × (nat ⇒ 'a) where
list-tail l = (fst l − 1, λm. (snd l) (Suc m))

lemma list-simp1: list-nil ≠ list-cons x l
  by (simp add: list-nil-def list-cons-def)

lemma list-simp2:
  assumes list-cons a al = list-cons b bl
  shows a = b al = bl
proof −
  have a = snd (list-cons a al) 0 b = snd (list-cons b bl) 0
    by (auto simp: list-cons-def)
  thus a = b
    by(simp add: assms)
next
```

```
  have fst al = fst bl
    using assms by (simp add: list-cons-def)
  moreover have snd al = snd bl
  proof
    fix n
    have snd al n = snd (list-cons a al) (Suc n)
      by (simp add: list-cons-def)
    also have ... = snd (list-cons b bl) (Suc n)
      by (simp add: assms)
    also have ... = snd bl n
      by (simp add: list-cons-def)
    finally show snd al n = snd bl n .
  qed
  ultimately show al = bl
    by (simp add: prod.expand)
qed

lemma
  shows list-simp3:list-head (list-cons a l) = a
    and list-simp4:list-tail (list-cons a l) = l
  by(simp-all add: list-head-def list-cons-def list-tail-def)

lemma list-decomp1:
  assumes l ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X)
  shows l = list-nil ∨
          (∃ a l'. a ∈ qbs-space X ∧ l' ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q
i∈{..<n}. X) ∧ l = list-cons a l')
proof(cases l)
  case hl:(Pair n f)
  show ?thesis
  proof(cases n)
    case 0
    then show ?thesis
      using assms hl by (simp add:  list-nil-def coPiQ-space PiQ-space)
  next
    case hn:(Suc n')
    define f' where f' ≡ λm. f (Suc m)
    have l = list-cons (f 0) (n',f')
      unfolding hl hn list-cons-def
    proof safe
      fix m
      show f = (λm. if m = 0 then f 0 else snd (n', f') (m − 1))
      proof
        fix m
        show f m = (if m = 0 then f 0 else snd (n', f') (m − 1))
          using assms hl by(cases m; fastforce simp: f'-def)
      qed
    qed simp
    moreover have (n', f') ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}.
```

X)
proof −
have $\bigwedge x.\ x \in \{..<n'\} \Longrightarrow f'\ x \in$ *qbs-space* X
using *assms hl hn* **by**(*fastforce simp*: *f'-def coPiQ-space PiQ-space*)
moreover {
fix x
assume *1*:$x \notin \{..<n'\}$
hence $f'\ x =$ *undefined*
using *hl assms hn* **by**(*auto simp*: *f'-def coPiQ-space PiQ-space*)
}
ultimately show *?thesis*
by(*auto simp add*: *coPiQ-space PiQ-space*)
qed
ultimately show *?thesis*
using *hl assms* **by**(*auto intro*!: *exI*[**where** x=*f 0*] *exI*[**where** x=(n',λm. *if m = 0 then undefined else f* (*Suc m*))] *simp*: *list-cons-def coPiQ-space PiQ-space*)
qed
qed

lemma *list-simp5*:
assumes $l \in$ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q i∈{..<n}. X)
and $l \neq$ *list-nil*
shows $l =$ *list-cons* (*list-head l*) (*list-tail l*)
proof −
obtain *a l'* **where** *hl*:
a ∈ *qbs-space X l'* ∈ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q i∈{..<n}. X) $l =$ *list-cons a l'*
using *list-decomp1*[*OF assms*(*1*)] *assms*(*2*) **by** *blast*
hence *list-head l = a list-tail l = l'*
by(*simp-all add*: *list-simp3 list-simp4*)
thus *?thesis*
using *hl*(*3*) *list-simp2* **by** *auto*
qed

lemma *list-simp6*:
list-nil ∈ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q i∈{..<n}. X)
by (*simp add*: *list-nil-def coPiQ-space PiQ-space*)

lemma *list-simp7*:
assumes $a \in$ *qbs-space* X
and $l \in$ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q i∈{..<n}. X)
shows *list-cons a l* ∈ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q i∈{..<n}. X)
using *assms* **by**(*fastforce simp*: *PiE-def extensional-def list-cons-def coPiQ-space PiQ-space*)

lemma *list-destruct-rule*:
assumes $l \in$ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q i∈{..<n}. X)
P list-nil
and $\bigwedge a\ l'.\ a \in$ *qbs-space* $X \Longrightarrow l' \in$ *qbs-space* (Π_Q n∈(*UNIV* :: *nat set*).Π_Q

```
i∈{..<n}. X) ⟹ P (list-cons a l')
    shows P l
  by(rule disjE[OF list-decomp1[OF assms(1)]]) (use assms in auto)

lemma list-induct-rule:
  assumes l ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X)
          P list-nil
      and ⋀a l'. a ∈ qbs-space X ⟹ l' ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q
i∈{..<n}. X) ⟹ P l' ⟹ P (list-cons a l')
    shows P l
proof(cases l)
  case hl:(Pair n f)
  then show ?thesis
    using assms(1)
  proof(induction n arbitrary: f l)
    case 0
    then show ?case
      using assms(2) by (simp add: coPiQ-space PiQ-space list-nil-def)
  next
    case ih:(Suc n)
    then obtain a l' where hl:
    a ∈ qbs-space X l' ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X) l =
list-cons a l'
      using list-decomp1 by(simp add: list-nil-def) blast
    have P l'
      using ih hl(3)
        by(auto intro!: ih(1)[OF - hl(2),of snd l'] simp: coPiQ-space PiQ-space
list-cons-def)
    from assms(3)[OF hl(1,2) this]
    show ?case
      by(simp add: hl(3))
  qed
qed

lemma to-list-simp1: to-list list-nil = []
  by(simp add: to-list-def list-nil-def)

lemma to-list-simp2:
  assumes l ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X)
  shows to-list (list-cons a l) = a # to-list l
 using assms by(auto simp:PiE-def to-list-def list-cons-def coPiQ-space PiQ-space)

lemma to-list-set:
  assumes l ∈ qbs-space (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X)
  shows set (to-list l) ⊆ qbs-space X
  by(rule list-induct-rule[OF assms]) (auto simp: to-list-simp1 to-list-simp2)

lemma from-list-length: fst (from-list l) = length l
  by(induction l, simp-all add: list-cons-def list-nil-def)
```

lemma *from-list-in-list-of*:
assumes *set l* ⊆ *qbs-space X*
shows *from-list l* ∈ *qbs-space* (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)
using *assms* **by**(*induction l*) (*auto simp*: *PiE-def extensional-def Pi-def coPiQ-space PiQ-space list-nil-def list-cons-def*)

lemma *from-list-in-list-of'*: *from-list l* ∈ *qbs-space* ((Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *Abs-quasi-borel* (*UNIV*,*UNIV*)))
proof −
have *set l* ⊆ *qbs-space* (*Abs-quasi-borel* (*UNIV*,*UNIV*))
by(*simp add*: *qbs-space-def Abs-quasi-borel-inverse*[*of* (*UNIV*,*UNIV*),*simplified is-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def*,*simplified*])
thus *?thesis*
using *from-list-in-list-of* **by** *blast*
qed

lemma *list-cons-in-list-of*:
assumes *set* (*a*#*l*) ⊆ *qbs-space X*
shows *list-cons a* (*from-list l*) ∈ *qbs-space* (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)
using *from-list-in-list-of*[*OF assms*] **by** *simp*

lemma *from-list-to-list-ident*:
to-list (*from-list l*) = *l*
by(*induction l*) (*simp add*: *to-list-def list-nil-def*,*simp add*: *to-list-simp2*[*OF from-list-in-list-of'*])

lemma *to-list-from-list-ident*:
assumes *l* ∈ *qbs-space* (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)
shows *from-list* (*to-list l*) = *l*
proof(*rule list-induct-rule*[*OF assms*])
fix *a l'*
assume *h*: *l'* ∈ *qbs-space* (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)
and *ih*:*from-list* (*to-list l'*) = *l'*
show *from-list* (*to-list* (*list-cons a l'*)) = *list-cons a l'*
by(*auto simp add*: *to-list-simp2*[*OF h*] *ih*[*simplified*])
qed (*simp add*: *to-list-simp1*)

definition *rec-list'* :: *'b* ⇒ (*'a* ⇒ (*nat* × (*nat* ⇒ *'a*)) ⇒ *'b* ⇒ *'b*) ⇒ (*nat* × (*nat* ⇒ *'a*)) ⇒ *'b* **where**
rec-list' t0 f l ≡ (*rec-list t0* (λ*x l'*. *f x* (*from-list l'*)) (*to-list l*))

lemma *rec-list'-simp1*:
rec-list' t f list-nil = *t*
by(*simp add*: *rec-list'-def to-list-simp1*)

lemma *rec-list'-simp2*:
assumes *l* ∈ *qbs-space* (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)
shows *rec-list' t f* (*list-cons x l*) = *f x l* (*rec-list' t f l*)

by(*simp add*: *rec-list'-def to-list-simp2*[*OF assms*] *to-list-from-list-ident*[*OF assms*,*simplified*])

lemma *list-qbs-space*: *qbs-space* (*list-qbs X*) = *lists* (*qbs-space X*)
using *to-list-set* **by**(*auto simp*: *list-qbs-def map-qbs-space image-def from-list-to-list-ident from-list-in-list-of subset-iff intro*!: *bexI*[**where** *x*=*from-list* -])

3.1.11 Option Spaces

The option spaces is defined using the following isomorphism.

$$Option(X) \cong X + 1$$

definition *option-qbs* :: *'a quasi-borel* ⇒ *'a option quasi-borel* **where**
option-qbs X = *map-qbs* (λ*x*. *case x of Inl y* ⇒ *Some y* | *Inr y* ⇒ *None*) (*X* $\bigoplus_Q$ 1_Q)

lemma *option-qbs-space*: *qbs-space* (*option-qbs X*) = {*Some x*|*x*. *x* ∈ *qbs-space X*} ∪ {*None*}
by(*auto simp*: *option-qbs-def map-qbs-space copair-qbs-space*) (*metis InrI image-eqI insert-iff old.sum.simps*(*6*), *metis InlI image-iff sum.case*(*1*))

3.1.12 Function Spaces

definition *exp-qbs* :: [*'a quasi-borel*, *'b quasi-borel*] ⇒ (*'a* ⇒ *'b*) *quasi-borel* (**infixr** ⇒$_Q$ *61*) **where**
X ⇒$_Q$ *Y* ≡ *Abs-quasi-borel* ({*f*. ∀α ∈ *qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*}, {*g*. ∀α∈ *borel-measurable borel*. ∀β∈ *qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y*})

lemma
shows *exp-qbs-space*: *qbs-space* (*exp-qbs X Y*) = {*f*. ∀α ∈ *qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*}
and *exp-qbs-Mx*: *qbs-Mx* (*exp-qbs X Y*) = {*g*. ∀α∈ *borel-measurable borel*. ∀β∈ *qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y*}
proof −
have {*g*:: *real* ⇒ -. ∀α∈ *borel-measurable borel*. ∀β∈ *qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y*} ⊆ *UNIV* → {*f*. ∀α ∈ *qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*}
proof *safe*
fix *g* :: *real* ⇒ - **and** *r* :: *real* **and** α
assume *h*:∀α∈*borel-measurable borel*. ∀β∈*qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y* α ∈ *qbs-Mx X*
have [*simp*]: *g r* ∘ α = (λ*l*. *g r* (α *l*)) **by** (*auto simp*: *comp-def*)
thus *g r* ∘ α ∈ *qbs-Mx Y*
using *h* **by** *auto*
qed
moreover have *qbs-closed3* {*g*. ∀α∈ *borel-measurable borel*. ∀β∈ *qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y*}
by(*rule qbs-closed3I*, *auto*) (*rule qbs-closed3-dest*,*auto*)
ultimately have *Rep-quasi-borel* (*exp-qbs X Y*) = ({*f*. ∀α ∈ *qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*}, {*g*. ∀α∈ *borel-measurable borel*. ∀β∈ *qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y*})

unfolding *exp-qbs-def* **by**(*auto intro*!: *Abs-quasi-borel-inverse is-quasi-borel-intro qbs-closed1I qbs-closed2I simp*: *comp-def*)
thus *qbs-space* (*exp-qbs X Y*) = {*f*. ∀ α ∈ *qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*}
qbs-Mx (*exp-qbs X Y*) = {*g*. ∀ α∈ *borel-measurable borel*. ∀ β∈ *qbs-Mx X*. (λ*r*. *g* (α *r*) (β *r*)) ∈ *qbs-Mx Y*}
by(*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

3.1.13 Ordering on Quasi-Borel Spaces

inductive-set *generating-Mx* :: *'a set* ⇒ (*real* ⇒ *'a*) *set* ⇒ (*real* ⇒ *'a*) *set*
for *X* :: *'a set* **and** *Mx* :: (*real* ⇒ *'a*) *set*
where
Basic: α ∈ *Mx* ⟹ α ∈ *generating-Mx X Mx*
| *Const*: *x* ∈ *X* ⟹ (λ*r*. *x*) ∈ *generating-Mx X Mx*
| *Comp* : *f* ∈ (*borel* :: *real measure*) →$_M$ (*borel* :: *real measure*) ⟹ α ∈ *generating-Mx X Mx* ⟹ α ∘ *f* ∈ *generating-Mx X Mx*
| *Part* : (⋀*i*. *Fi i* ∈ *generating-Mx X Mx*) ⟹ *P* ∈ *borel* →$_M$ *count-space* (*UNIV* :: *nat set*) ⟹ (λ*r*. *Fi* (*P r*) *r*) ∈ *generating-Mx X Mx*

lemma *generating-Mx-to-space*:
assumes *Mx* ⊆ *UNIV* → *X*
shows *generating-Mx X Mx* ⊆ *UNIV* → *X*
proof
fix α
assume α ∈ *generating-Mx X Mx*
then show α ∈ *UNIV* → *X*
by(*induct rule*: *generating-Mx.induct*) (*use assms* **in** *auto*)
qed

lemma *generating-Mx-closed1*:
qbs-closed1 (*generating-Mx X Mx*)
by (*simp add*: *generating-Mx.Comp qbs-closed1I*)

lemma *generating-Mx-closed2*:
qbs-closed2 X (*generating-Mx X Mx*)
by (*simp add*: *generating-Mx.Const qbs-closed2I*)

lemma *generating-Mx-closed3*:
qbs-closed3 (*generating-Mx X Mx*)
by(*simp add*: *qbs-closed3I generating-Mx.Part*)

lemma *generating-Mx-Mx*:
generating-Mx (*qbs-space X*) (*qbs-Mx X*) = *qbs-Mx X*
proof *safe*
fix α
assume α ∈ *generating-Mx* (*qbs-space X*) (*qbs-Mx X*)
then show α ∈ *qbs-Mx X*
by(*rule generating-Mx.induct*) (*auto intro*!: *qbs-closed1-dest*[*simplified comp-def*]

simp: *qbs-closed3-dest'*)
next
fix α
assume $\alpha \in$ *qbs-Mx X*
then show $\alpha \in$ *generating-Mx* (*qbs-space X*) (*qbs-Mx X*) **..**
qed

instantiation *quasi-borel* :: (*type*) *order-bot*
begin

inductive *less-eq-quasi-borel* :: *'a quasi-borel* ⇒ *'a quasi-borel* ⇒ *bool* **where**
qbs-space X ⊂ *qbs-space Y* ⟹ *less-eq-quasi-borel X Y*
| *qbs-space X* = *qbs-space Y* ⟹ *qbs-Mx Y* ⊆ *qbs-Mx X* ⟹ *less-eq-quasi-borel X Y*

lemma *le-quasi-borel-iff*:
X ≤ *Y* ⟷ (*if qbs-space X* = *qbs-space Y then qbs-Mx Y* ⊆ *qbs-Mx X else qbs-space X* ⊂ *qbs-space Y*)
by(*auto elim*: *less-eq-quasi-borel.cases intro*: *less-eq-quasi-borel.intros*)

definition *less-quasi-borel* :: *'a quasi-borel* ⇒ *'a quasi-borel* ⇒ *bool* **where**
less-quasi-borel X Y ⟷ (*X* ≤ *Y* ∧ ¬ *Y* ≤ *X*)

definition *bot-quasi-borel* :: *'a quasi-borel* **where**
bot-quasi-borel = *empty-quasi-borel*

instance
proof
show *bot* ≤ *a* **for** *a* :: *'a quasi-borel*
using *qbs-empty-equiv*
by(*auto simp add*: *le-quasi-borel-iff bot-quasi-borel-def*)
qed (*auto simp*: *le-quasi-borel-iff less-quasi-borel-def split*: *if-split-asm intro*: *qbs-eqI*)
end

definition *inf-quasi-borel* :: [*'a quasi-borel*, *'a quasi-borel*] ⇒ *'a quasi-borel* **where**
inf-quasi-borel X X' = *Abs-quasi-borel* (*qbs-space X* ∩ *qbs-space X'*, *qbs-Mx X* ∩ *qbs-Mx X'*)

lemma *inf-quasi-borel-correct*: *Rep-quasi-borel* (*inf-quasi-borel X X'*) = (*qbs-space X* ∩ *qbs-space X'*, *qbs-Mx X* ∩ *qbs-Mx X'*)
by(*auto intro*!: *Abs-quasi-borel-inverse simp*: *inf-quasi-borel-def is-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def dest*: *qbs-Mx-to-X*)

lemma *inf-qbs-space*[*simp*]: *qbs-space* (*inf-quasi-borel X X'*) = *qbs-space X* ∩ *qbs-space X'*
by (*simp add*: *qbs-space-def inf-quasi-borel-correct*)

lemma *inf-qbs-Mx*[*simp*]: *qbs-Mx* (*inf-quasi-borel X X'*) = *qbs-Mx X* ∩ *qbs-Mx X'*
by(*simp add*: *qbs-Mx-def inf-quasi-borel-correct*)

definition *max-quasi-borel* :: *'a set* ⇒ *'a quasi-borel* **where**
max-quasi-borel X = *Abs-quasi-borel* (*X*, *UNIV* → *X*)

lemma *max-quasi-borel-correct*: *Rep-quasi-borel* (*max-quasi-borel X*) = (*X*, *UNIV* → *X*)
by(*fastforce intro*!: *Abs-quasi-borel-inverse*
simp: *max-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def is-quasi-borel-def*)

lemma *max-qbs-space*[*simp*]: *qbs-space* (*max-quasi-borel X*) = *X*
by(*simp add*: *qbs-space-def max-quasi-borel-correct*)

lemma *max-qbs-Mx*[*simp*]: *qbs-Mx* (*max-quasi-borel X*) = *UNIV* → *X*
by(*simp add*: *qbs-Mx-def max-quasi-borel-correct*)

instantiation *quasi-borel* :: (*type*) *semilattice-sup*
begin

definition *sup-quasi-borel* :: *'a quasi-borel* ⇒ *'a quasi-borel* ⇒ *'a quasi-borel* **where**
sup-quasi-borel X Y ≡ (*if qbs-space X* = *qbs-space Y* *then inf-quasi-borel X Y*
else if qbs-space X ⊂ *qbs-space Y then Y*
else if qbs-space Y ⊂ *qbs-space X then X*
else max-quasi-borel (*qbs-space X* ∪ *qbs-space Y*))

instance
proof
fix *X Y* :: *'a quasi-borel*
let *?X* = *qbs-space X*
let *?Y* = *qbs-space Y*
consider *?X* = *?Y* | *?X* ⊂ *?Y* | *?Y* ⊂ *?X* | *?X* ⊂ *?X* ∪ *?Y* ∧ *?Y* ⊂ *?X* ∪ *?Y*
by *auto*
then show *X* ≤ *X* ⊔ *Y*
proof(*cases*)
case *1*
show *?thesis*
unfolding *sup-quasi-borel-def*
by(*rule less-eq-quasi-borel.intros*(*2*),*simp-all add*: *1*)
next
case *2*
then show *?thesis*
unfolding *sup-quasi-borel-def*
by (*simp add*: *less-eq-quasi-borel.intros*(*1*))
next
case *3*
then show *?thesis*
unfolding *sup-quasi-borel-def*
by *auto*
next

```
    case 4
    then show ?thesis
      unfolding sup-quasi-borel-def
      by(auto simp: less-eq-quasi-borel.intros(1))
  qed
next
  fix X Y :: 'a quasi-borel
  let ?X = qbs-space X
  let ?Y = qbs-space Y
  consider ?X = ?Y | ?X ⊂ ?Y | ?Y ⊂ ?X | ?X ⊂ ?X ∪ ?Y ∧ ?Y ⊂ ?X ∪ ?Y
    by auto
  then show Y ≤ X ⊔ Y
  proof(cases)
    case 1
    show ?thesis
      unfolding sup-quasi-borel-def
      by(rule less-eq-quasi-borel.intros(2)) (simp-all add: 1)
  next
    case 2
    then show ?thesis
      unfolding sup-quasi-borel-def
      by auto
  next
    case 3
    then show ?thesis
      unfolding sup-quasi-borel-def
      by (auto simp add: less-eq-quasi-borel.intros(1))
  next
    case 4
    then show ?thesis
      unfolding sup-quasi-borel-def
      by(auto simp: less-eq-quasi-borel.intros(1))
  qed
next
  fix X Y Z :: 'a quasi-borel
  assume h:X ≤ Z Y ≤ Z
  let ?X = qbs-space X
  let ?Y = qbs-space Y
  let ?Z = qbs-space Z
  consider ?X = ?Y | ?X ⊂ ?Y | ?Y ⊂ ?X | ?X ⊂ ?X ∪ ?Y ∧ ?Y ⊂ ?X ∪ ?Y
    by auto
  then show sup X Y ≤ Z
  proof cases
    case 1
    show ?thesis
      unfolding sup-quasi-borel-def
      apply(simp add: 1,rule less-eq-quasi-borel.cases[OF h(1)])
       apply(rule less-eq-quasi-borel.intros(1))
       apply auto[1]
```

```
    apply simp
    apply(rule less-eq-quasi-borel.intros(2))
     apply(simp add: 1)
    apply(rule less-eq-quasi-borel.cases[OF h(2)])
    using 1
     apply fastforce
    apply simp
    by (metis 1 h(2) inf-qbs-Mx le-inf-iff le-quasi-borel-iff)

  next
    case 2
    then show ?thesis
      unfolding sup-quasi-borel-def
      using h(2) by auto
  next
    case 3
    then show ?thesis
      unfolding sup-quasi-borel-def
      using h(1) by auto
  next
    case 4
    then have [simp]:?X ≠ ?Y ~ (?X ⊂ ?Y) ~ (?Y ⊂ ?X)
      by auto
    have [simp]:?X ⊆ ?Z ?Y ⊆ ?Z
      by (metis h(1) dual-order.order-iff-strict less-eq-quasi-borel.cases)
         (metis h(2) dual-order.order-iff-strict less-eq-quasi-borel.cases)
    then consider ?X ∪ ?Y = ?Z | ?X ∪ ?Y ⊂ ?Z
      by blast
    then show ?thesis
      unfolding sup-quasi-borel-def
      apply cases
       apply simp
       apply(rule less-eq-quasi-borel.intros(2))
        apply simp
      using qbs-Mx-to-X apply auto[1]
      by(simp add: less-eq-quasi-borel.intros(1))
  qed
qed

end

end
```

3.2 Morphisms of Quasi-Borel Spaces

```
theory QBS-Morphism

imports
 QuasiBorel
```

begin

abbreviation *qbs-morphism* :: [*'a quasi-borel*, *'b quasi-borel*] ⇒ (*'a* ⇒ *'b*) *set* (**infixr** $\rightarrow_Q$ *60*) **where**
  $X \rightarrow_Q Y \equiv$ *qbs-space* ($X \Rightarrow_Q Y$)

lemma *qbs-morphismI*: ($\bigwedge\alpha.\ \alpha \in$ *qbs-Mx* $X \Longrightarrow f \circ \alpha \in$ *qbs-Mx* Y) $\Longrightarrow f \in X \rightarrow_Q Y$
  by(*auto simp*: *exp-qbs-space*)

lemma *qbs-morphism-def*: $X \rightarrow_Q Y = \{f \in$ *qbs-space* $X \rightarrow$ *qbs-space* $Y.\ \forall \alpha \in$ *qbs-Mx* $X.\ f \circ \alpha \in$ *qbs-Mx* $Y\}$
  unfolding *exp-qbs-space*
proof *safe*
  fix *f x*
  assume *h*:$x \in$ *qbs-space* X $\forall \alpha \in$ *qbs-Mx* $X.\ f \circ \alpha \in$ *qbs-Mx* Y
  then have ($\lambda r.\ x$) $\in$ *qbs-Mx* X
    by *simp*
  hence $f \circ (\lambda r.\ x) \in$ *qbs-Mx* Y
    using *h* **by** *blast*
  with *qbs-Mx-to-X* **show** $f\ x \in$ *qbs-space* Y
    by *auto*
qed *auto*

lemma *qbs-morphism-Mx*:
  assumes $f \in X \rightarrow_Q Y$ $\alpha \in$ *qbs-Mx* X
  shows $f \circ \alpha \in$ *qbs-Mx* Y
  using *assms* **by**(*auto simp*: *qbs-morphism-def*)

lemma *qbs-morphism-space*:
  assumes $f \in X \rightarrow_Q Y$ $x \in$ *qbs-space* X
  shows $f\ x \in$ *qbs-space* Y
  using *assms* **by**(*auto simp*: *qbs-morphism-def*)

lemma *qbs-morphism-ident*[*simp*]:
  $id \in X \rightarrow_Q X$
  by(*auto intro*: *qbs-morphismI*)

lemma *qbs-morphism-ident'*[*simp*]:
  ($\lambda x.\ x$) $\in X \rightarrow_Q X$
  using *qbs-morphism-ident* **by**(*simp add*: *id-def*)

lemma *qbs-morphism-comp*:
  assumes $f \in X \rightarrow_Q Y$ $g \in Y \rightarrow_Q Z$
  shows $g \circ f \in X \rightarrow_Q Z$
  using *assms* **by** (*simp add*: *comp-assoc Pi-def qbs-morphism-def*)

lemma *qbs-morphism-compose-rev*:

assumes $f \in Y \to_Q Z$ **and** $g \in X \to_Q Y$
shows $(\lambda x.\ f\ (g\ x)) \in X \to_Q Z$
using *qbs-morphism-comp*[*OF assms*(*2*,*1*)] **by**(*simp add*: *comp-def*)

lemma *qbs-morphism-compose*:
assumes $g \in X \to_Q Y$ **and** $f \in Y \to_Q Z$
shows $(\lambda x.\ f\ (g\ x)) \in X \to_Q Z$
using *qbs-morphism-compose-rev*[*OF assms*(*2*,*1*)] **.**

lemma *qbs-morphism-cong′*:
assumes $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x = g\ x$
and $f \in X \to_Q Y$
shows $g \in X \to_Q Y$
proof(*rule qbs-morphismI*)
fix α
assume *1*:$\alpha \in$ *qbs-Mx* X
have $g \circ \alpha = f \circ \alpha$
proof
fix x
have $\alpha\ x \in$ *qbs-space* X
using *1 qbs-decomp*[*of X*] *qbs-Mx-to-X* **by** *auto*
thus $(g \circ \alpha)\ x = (f \circ \alpha)\ x$
using *assms*(*1*) **by** *simp*
qed
thus $g \circ \alpha \in$ *qbs-Mx* Y
using *1 assms*(*2*) **by**(*simp add*: *qbs-morphism-def*)
qed

lemma *qbs-morphism-cong*:
assumes $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x = g\ x$
shows $f \in X \to_Q Y \longleftrightarrow g \in X \to_Q Y$
using *assms* **by**(*auto simp*: *qbs-morphism-cong′*[*of - f g*] *qbs-morphism-cong′*[*of - g f*])

lemma *qbs-morphism-const*:
assumes $y \in$ *qbs-space* Y
shows $(\lambda x.\ y) \in X \to_Q Y$
using *assms* **by** (*auto intro*: *qbs-morphismI*)

lemma *qbs-morphism-from-empty*: *qbs-space* $X = \{\} \Longrightarrow f \in X \to_Q Y$
by(*auto intro*!: *qbs-morphismI simp*: *qbs-empty-equiv*)

lemma *unit-quasi-borel-terminal*: $\exists!\ f.\ f \in X \to_Q$ *unit-quasi-borel*
by(*fastforce simp*: *qbs-morphism-def*)

definition *to-unit-quasi-borel* :: $'a \Rightarrow$ *unit* $(!_Q)$ **where**
to-unit-quasi-borel $\equiv (\lambda r.())$

lemma *to-unit-quasi-borel-morphism*:

$!_Q \in X \to_Q$ *unit-quasi-borel*
by(*auto simp add*: *to-unit-quasi-borel-def qbs-morphism-def*)

lemma *qbs-morphism-subD*:
assumes $f \in X \to_Q$ *sub-qbs* Y A
shows $f \in X \to_Q$ Y
using *qbs-morphism-Mx*[*OF assms*] **by**(*auto intro*!: *qbs-morphismI simp*: *sub-qbs-Mx*)

lemma *qbs-morphism-subI1*:
assumes $f \in X \to_Q$ Y $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x \in A$
shows $f \in X \to_Q$ *sub-qbs* Y A
using *qbs-morphism-space*[*OF assms*(*1*)] *qbs-morphism-Mx*[*OF assms*(*1*)] *assms*(*2*)
qbs-Mx-to-X[*of - X*]
by(*auto intro*!: *qbs-morphismI simp*: *sub-qbs-Mx*)

lemma *qbs-morphism-subI2*:
assumes $f \in X \to_Q$ Y
shows $f \in$ *sub-qbs* X $A \to_Q$ Y
using *qbs-morphism-Mx*[*OF assms*] **by**(*auto intro*!: *qbs-morphismI simp*: *sub-qbs-Mx*)

lemma *qbs-morphism-subI2'*:
assumes $f \in X \to_Q$ Y *qbs-space* $Z \subseteq$ *qbs-space* X *qbs-Mx* $Z \subseteq$ *qbs-Mx* X
shows $f \in Z \to_Q$ Y
using *qbs-morphism-Mx*[*OF assms*(*1*)] *assms*(*2*,*3*) **by**(*auto intro*!: *qbs-morphismI*)

corollary *qbs-morphism-subsubI*:
assumes $f \in X \to_Q$ Y $\bigwedge x.\ x \in A \Longrightarrow x \in$ *qbs-space* $X \Longrightarrow f\ x \in B$
shows $f \in$ *sub-qbs* X $A \to_Q$ *sub-qbs* Y B
by(*rule qbs-morphism-subI1*) (*auto intro*!: *qbs-morphism-subI2 assms simp*: *sub-qbs-space*)

lemma *map-qbs-morphism-f*: $f \in X \to_Q$ *map-qbs* f X
by(*auto intro*!: *qbs-morphismI simp*: *map-qbs-Mx*)

lemma *map-qbs-morphism-inverse-f*:
assumes $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow g\ (f\ x) = x$
shows $g \in$ *map-qbs* f $X \to_Q$ X
proof −
{
fix α
assume *h*:$\alpha \in$ *qbs-Mx* X
from *qbs-Mx-to-X*[*OF this*] *assms* **have** $g \circ (f \circ \alpha) = \alpha$
by *auto*
with *h* **have** $g \circ (f \circ \alpha) \in$ *qbs-Mx* X **by** *simp*
}
thus *?thesis*
by(*auto intro*!: *qbs-morphismI simp*: *map-qbs-Mx*)
qed

lemma *pair-qbs-morphismI*:

assumes $\bigwedge \alpha\ \beta.\ \alpha \in qbs\text{-}Mx\ X \Longrightarrow \beta \in qbs\text{-}Mx\ Y$
$\Longrightarrow (\lambda r.\ f\ (\alpha\ r,\ \beta\ r)) \in qbs\text{-}Mx\ Z$
shows $f \in (X \bigotimes_Q Y) \to_Q Z$
using *assms* **by**(*fastforce intro*!: *qbs-morphismI simp*: *pair-qbs-Mx comp-def*)

lemma *pair-qbs-MxD*:
assumes $\gamma \in qbs\text{-}Mx\ (X \bigotimes_Q Y)$
obtains $\alpha\ \beta$ **where** $\alpha \in qbs\text{-}Mx\ X\ \beta \in qbs\text{-}Mx\ Y\ \gamma = (\lambda x.\ (\alpha\ x,\ \beta\ x))$
using *assms* **by**(*auto simp*: *pair-qbs-Mx*)

lemma *pair-qbs-MxI*:
assumes $(\lambda x.\ fst\ (\gamma\ x)) \in qbs\text{-}Mx\ X$ **and** $(\lambda x.\ snd\ (\gamma\ x)) \in qbs\text{-}Mx\ Y$
shows $\gamma \in qbs\text{-}Mx\ (X \bigotimes_Q Y)$
using *assms* **by**(*auto simp*: *pair-qbs-Mx comp-def*)

lemma
shows *fst-qbs-morphism*: $fst \in X \bigotimes_Q Y \to_Q X$
and *snd-qbs-morphism*: $snd \in X \bigotimes_Q Y \to_Q Y$
by(*auto intro*!: *pair-qbs-morphismI simp*: *comp-def*)

lemma *qbs-morphism-pair-iff*:
$f \in X \to_Q Y \bigotimes_Q Z \longleftrightarrow fst \circ f \in X \to_Q Y \wedge snd \circ f \in X \to_Q Z$
by(*auto intro*!: *qbs-morphism-comp fst-qbs-morphism snd-qbs-morphism*)
(*auto dest*: *qbs-morphism-Mx intro*!: *qbs-morphismI simp*: *pair-qbs-Mx comp-assoc*[*symmetric*])

lemma *qbs-morphism-Pair*:
assumes $f \in Z \to_Q X$
and $g \in Z \to_Q Y$
shows $(\lambda z.\ (f\ z,\ g\ z)) \in Z \to_Q X \bigotimes_Q Y$
unfolding *qbs-morphism-pair-iff*
using *assms* **by** (*auto simp*: *comp-def*)

lemma *qbs-morphism-curry*: $curry \in exp\text{-}qbs\ (X \bigotimes_Q Y)\ Z \to_Q exp\text{-}qbs\ X\ (exp\text{-}qbs\ Y\ Z)$
by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx exp-qbs-Mx comp-def*)

corollary *curry-preserves-morphisms*:
assumes $(\lambda xy.\ f\ (fst\ xy)\ (snd\ xy)) \in X \bigotimes_Q Y \to_Q Z$
shows $f \in X \to_Q Y \Rightarrow_Q Z$
using *qbs-morphism-space*[*OF qbs-morphism-curry assms*] **by** (*auto simp*: *curry-def*)

lemma *qbs-morphism-eval*:
$(\lambda fx.\ (fst\ fx)\ (snd\ fx)) \in (X \Rightarrow_Q Y) \bigotimes_Q X \to_Q Y$
by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx exp-qbs-Mx comp-def*)

corollary *qbs-morphism-app*:
assumes $f \in X \to_Q (Y \Rightarrow_Q Z)\ g \in X \to_Q Y$
shows $(\lambda x.\ (f\ x)\ (g\ x)) \in X \to_Q Z$
by(*rule qbs-morphism-cong'*[**where** f=$(\lambda fx.\ (fst\ fx)\ (snd\ fx)) \circ (\lambda x.\ (f\ x,\ g\ x))$,*OF*

- *qbs-morphism-comp*[*OF qbs-morphism-Pair*[*OF assms*] *qbs-morphism-eval*]]) *auto*

ML-file ‹*qbs.ML*›

attribute-setup *qbs* = ‹
Attrib.add-del Qbs.qbs-add Qbs.qbs-del›
declaration of qbs rule

method-setup *qbs* = ‹ *Scan.lift* (*Scan.succeed* (*METHOD o Qbs.qbs-tac*))›

simproc-setup *qbs* (*x* ∈ *qbs-space X*) = ‹*K Qbs.simproc*›

declare
fst-qbs-morphism[*qbs*]
snd-qbs-morphism[*qbs*]
qbs-morphism-const[*qbs*]
qbs-morphism-ident[*qbs*]
qbs-morphism-ident'[*qbs*]
qbs-morphism-curry[*qbs*]

lemma [*qbs*]:
shows *qbs-morphism-Pair1*: *Pair* ∈ *X* →$_Q$ *Y* ⇒$_Q$ (*X* ⨂$_Q$ *Y*)
by(*auto intro*!: *qbs-morphismI simp*: *exp-qbs-Mx pair-qbs-Mx comp-def*)

lemma *qbs-morphism-case-prod*[*qbs*]: *case-prod* ∈ *exp-qbs X* (*exp-qbs Y Z*) →$_Q$ *exp-qbs* (*X* ⨂$_Q$ *Y*) *Z*
by(*fastforce intro*!: *qbs-morphismI simp*: *exp-qbs-Mx pair-qbs-Mx comp-def split-beta'*)

lemma *uncurry-preserves-morphisms*:
assumes [*qbs*]:(λ*x y*. *f* (*x*,*y*)) ∈ *X* →$_Q$ *Y* ⇒$_Q$ *Z*
shows *f* ∈ *X* ⨂$_Q$ *Y* →$_Q$ *Z*
by(*rule qbs-morphism-cong'*[**where** *f*=*case-prod* (λ*x y*. *f* (*x*,*y*))],*simp*) *qbs*

lemma *qbs-morphism-comp'*[*qbs*]:*comp* ∈ *Y* ⇒$_Q$ *Z* →$_Q$ (*X* ⇒$_Q$ *Y*) ⇒$_Q$ *X* ⇒$_Q$ *Z*
by(*auto intro*!: *qbs-morphismI simp*: *exp-qbs-Mx*)

lemma *arg-swap-morphism*:
assumes *f* ∈ *X* →$_Q$ *exp-qbs Y Z*
shows (λ*y x*. *f x y*) ∈ *Y* →$_Q$ *exp-qbs X Z*
using *assms* **by** *simp*

lemma *exp-qbs-comp-morphism*:
assumes *f* ∈ *W* →$_Q$ *exp-qbs X Y*
and *g* ∈ *W* →$_Q$ *exp-qbs Y Z*
shows (λ*w*. *g w* ∘ *f w*) ∈ *W* →$_Q$ *exp-qbs X Z*
using *assms* **by** *qbs*

lemma *arg-swap-morphism-map-qbs1*:
assumes *g* ∈ *exp-qbs W* (*exp-qbs X Y*) →$_Q$ *Z*

shows $(\lambda k.\ g\ (k \circ f)) \in$ *exp-qbs* (*map-qbs f W*) (*exp-qbs X Y*) $\to_Q$ *Z*
using *assms map-qbs-morphism-f* **by** *qbs*

lemma *qbs-morphism-map-prod*[*qbs*]: *map-prod* $\in X \Rightarrow_Q Y \to_Q (W \Rightarrow_Q Z) \Rightarrow_Q (X \bigotimes_Q W) \Rightarrow_Q (Y \bigotimes_Q Z)$
by(*auto intro*!: *qbs-morphismI simp*: *exp-qbs-Mx pair-qbs-Mx map-prod-def comp-def case-prod-beta'*)

lemma *qbs-morphism-pair-swap*:
assumes $f \in X \bigotimes_Q Y \to_Q Z$
shows $(\lambda(x,y).\ f\ (y,x)) \in Y \bigotimes_Q X \to_Q Z$
using *assms* **by** *simp*

lemma
shows *qbs-morphism-pair-assoc1*: $(\lambda((x,y),z).\ (x,(y,z))) \in (X \bigotimes_Q Y) \bigotimes_Q Z \to_Q X \bigotimes_Q (Y \bigotimes_Q Z)$
and *qbs-morphism-pair-assoc2*: $(\lambda(x,(y,z)).\ ((x,y),z)) \in X \bigotimes_Q (Y \bigotimes_Q Z) \to_Q (X \bigotimes_Q Y) \bigotimes_Q Z$
by *simp-all*

lemma *Inl-qbs-morphism*[*qbs*]: *Inl* $\in X \to_Q X \bigoplus_Q Y$
by(*auto intro*!: *qbs-morphismI bexI*[**where** *x*={}] *simp*: *copair-qbs-Mx copair-qbs-Mx-def comp-def*)

lemma *Inr-qbs-morphism*[*qbs*]: *Inr* $\in Y \to_Q X \bigoplus_Q Y$
by(*auto intro*!: *qbs-morphismI bexI*[**where** *x*=*UNIV*] *simp*: *copair-qbs-Mx copair-qbs-Mx-def comp-def*)

lemma *case-sum-qbs-morphism*[*qbs*]: *case-sum* $\in X \Rightarrow_Q Z \to_Q (Y \Rightarrow_Q Z) \Rightarrow_Q (X \bigoplus_Q Y \Rightarrow_Q Z)$
by(*auto intro*!: *qbs-morphismI qbs-Mx-indicat simp*: *copair-qbs-Mx copair-qbs-Mx-def exp-qbs-Mx case-sum-if*)

lemma *map-sum-qbs-morphism*[*qbs*]: *map-sum* $\in X \Rightarrow_Q Y \to_Q (X' \Rightarrow_Q Y') \Rightarrow_Q (X \bigoplus_Q X' \Rightarrow_Q Y \bigoplus_Q Y')$
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* $(X \Rightarrow_Q Y)$
then have *ha*[*measurable*]: $\forall (k :: real \Rightarrow real) \in$*borel-measurable borel*. $\forall a \in$*qbs-Mx X*. $(\lambda r.\ \alpha\ (k\ r)\ (a\ r)) \in$ *qbs-Mx Y*
by (*auto simp*: *exp-qbs-Mx*)
show *map-sum* $\circ\ \alpha \in$ *qbs-Mx* $((X' \Rightarrow_Q Y') \Rightarrow_Q X \bigoplus_Q X' \Rightarrow_Q Y \bigoplus_Q Y')$
unfolding *exp-qbs-Mx*
proof *safe*
fix $\beta\ b$ **and** $f\ g ::$ *real* $\Rightarrow$ *real*
assume *h*[*measurable*]: $\forall (k :: real \Rightarrow real) \in$*borel-measurable borel*. $\forall b \in$*qbs-Mx X'*. $(\lambda r.\ \beta\ (k\ r)\ (b\ r)) \in$ *qbs-Mx Y'*
f $\in$ *borel-measurable borel g* $\in$ *borel-measurable borel*
and *b*: $b \in$ *qbs-Mx* $(X \bigoplus_Q X')$

show $(\lambda r.\ (\mathit{map\text{-}sum} \circ \alpha)\ (f\ (g\ r))\ (\beta\ (g\ r))\ (b\ r)) \in \mathit{qbs\text{-}Mx}\ (Y \bigoplus_Q Y')$
proof(*rule copair-qbs-MxD*[*OF b*])
fix a
assume $a \in \mathit{qbs\text{-}Mx}\ X\ b = (\lambda r.\ \mathit{Inl}\ (a\ r))$
with *ha* **show** $(\lambda r.\ (\mathit{map\text{-}sum} \circ \alpha)\ (f\ (g\ r))\ (\beta\ (g\ r))\ (b\ r)) \in \mathit{qbs\text{-}Mx}\ (Y \bigoplus_Q Y')$
by(*auto simp*: *copair-qbs-Mx copair-qbs-Mx-def intro*!: *bexI*[**where** $x=\{\}$])
next
fix a
assume $a \in \mathit{qbs\text{-}Mx}\ X'\ b = (\lambda r.\ \mathit{Inr}\ (a\ r))$
with *h*(*1*) **show** $(\lambda r.\ (\mathit{map\text{-}sum} \circ \alpha)\ (f\ (g\ r))\ (\beta\ (g\ r))\ (b\ r)) \in \mathit{qbs\text{-}Mx}\ (Y \bigoplus_Q Y')$
by(*auto simp*: *copair-qbs-Mx copair-qbs-Mx-def intro*!: *bexI*[**where** $x=\mathit{UNIV}$])
next
fix $S\ a\ a'$
assume $S \in \mathit{sets}\ \mathit{borel}\ S \neq \{\}\ S \neq \mathit{UNIV}\ a \in \mathit{qbs\text{-}Mx}\ X\ a' \in \mathit{qbs\text{-}Mx}\ X'\ b = (\lambda r.\ \mathit{if}\ r \in S\ \mathit{then}\ \mathit{Inl}\ (a\ r)\ \mathit{else}\ \mathit{Inr}\ (a'\ r))$
with *h ha* **show** $(\lambda r.\ (\mathit{map\text{-}sum} \circ \alpha)\ (f\ (g\ r))\ (\beta\ (g\ r))\ (b\ r)) \in \mathit{qbs\text{-}Mx}\ (Y \bigoplus_Q Y')$
by *simp* (*fastforce simp*: *copair-qbs-Mx copair-qbs-Mx-def intro*!: *bexI*[**where** $x=S$])
qed
qed
qed

lemma *qbs-morphism-component-singleton*[*qbs*]:
assumes $i \in I$
shows $(\lambda x.\ x\ i) \in (\Pi_Q\ i{\in}I.\ (M\ i)) \to_Q M\ i$
by(*auto intro*!: *qbs-morphismI simp*: *comp-def assms PiQ-Mx*)

lemma *qbs-morphism-component-singleton'*:
assumes $f \in Y \to_Q (\Pi_Q\ i{\in}I.\ X\ i)\ g \in Z \to_Q Y\ i \in I$
shows $(\lambda x.\ f\ (g\ x)\ i) \in Z \to_Q X\ i$
by(*auto intro*!: *qbs-morphism-compose*[*OF assms*(*2*)] *qbs-morphism-compose*[*OF assms*(*1*)] *qbs-morphism-component-singleton assms*)

lemma *product-qbs-canonical1*:
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in Y \to_Q X\ i$
and $\bigwedge i.\ i \notin I \Longrightarrow f\ i = (\lambda y.\ \mathit{undefined})$
shows $(\lambda y\ i.\ f\ i\ y) \in Y \to_Q (\Pi_Q\ i{\in}I.\ X\ i)$
using *assms qbs-morphism-Mx*[*OF assms*(*1*)] **by**(*auto intro*!: *qbs-morphismI simp*: *PiQ-Mx comp-def*)

lemma *product-qbs-canonical2*:
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in Y \to_Q X\ i$
$\bigwedge i.\ i \notin I \Longrightarrow f\ i = (\lambda y.\ \mathit{undefined})$
$g \in Y \to_Q (\Pi_Q\ i{\in}I.\ X\ i)$
$\bigwedge i.\ i \in I \Longrightarrow f\ i = (\lambda x.\ x\ i) \circ g$
and $y \in \mathit{qbs\text{-}space}\ Y$

```
    shows g y = (λi. f i y)
proof(intro ext)
  fix i
  show g y i = f i y
  proof(cases i ∈ I)
    case True
    then show ?thesis
      using assms(4)[of i] by simp
  next
    case False
    with qbs-morphism-space[OF assms(3)] assms(2,3,5) show ?thesis
      by(auto simp: PiQ-Mx PiQ-space)
  qed
qed

lemma merge-qbs-morphism:
 merge I J ∈ (Π_Q i∈I. (M i)) ⨂_Q (Π_Q j∈J. (M j)) →_Q (Π_Q i∈I∪J. (M i))
proof(rule qbs-morphismI)
  fix α
  assume h:α ∈ qbs-Mx ((Π_Q i∈I. (M i)) ⨂_Q (Π_Q j∈J. (M j)))
  show merge I J ∘ α ∈ qbs-Mx (Π_Q i∈I∪J. (M i))
  proof −
    {
      fix i
      assume i ∈ I ∪ J
      then consider i ∈ I | i ∈ I ∧ i ∈ J | i ∉ I ∧ i ∈ J
        by auto
      hence (λr. (merge I J ∘ α) r i) ∈ qbs-Mx (M i)
        by cases (insert h, auto simp: merge-def split-beta′ pair-qbs-Mx PiQ-Mx)
    }
    thus ?thesis
      by(auto simp: PiQ-Mx) (auto simp: merge-def split-beta′)
  qed
qed

lemma ini-morphism[qbs]:
  assumes j ∈ I
  shows (λx. (j,x)) ∈ X j →_Q (Π_Q i∈I. X i)
 by(fastforce intro!: qbs-morphismI exI[where x=λr. j] simp: coPiQ-Mx-def comp-def
assms coPiQ-Mx)

lemma coPiQ-canonical1:
  assumes countable I
      and ⋀i. i ∈ I ⟹ f i ∈ X i →_Q Y
    shows (λ(i,x). f i x) ∈ (Π_Q i ∈I. X i) →_Q Y
proof(rule qbs-morphismI)
  fix α
  assume α ∈ qbs-Mx (coPiQ I X)
  then obtain β g where ha:
```

$\bigwedge i.\ i \in range\ g \Longrightarrow \beta\ i \in qbs\text{-}Mx\ (X\ i)\ \alpha = (\lambda r.\ (g\ r,\ \beta\ (g\ r)\ r))$ **and** *hg*[*measurable*]:$g \in borel \to_M count\text{-}space\ I$
by(*fastforce simp*: *coPiQ-Mx-def coPiQ-Mx*)
define f' **where** $f' \equiv (\lambda i\ r.\ f\ i\ (\beta\ i\ r))$
have $range\ g \subseteq I$
using *measurable-space*[*OF hg*] **by** *auto*
hence *1*:$(\bigwedge i.\ i \in range\ g \Longrightarrow f'\ i \in qbs\text{-}Mx\ Y)$
using *qbs-morphism-Mx*[*OF assms*(*2*) *ha*(*1*),*simplified comp-def*]
by(*auto simp*: f'*-def*)
have $(\lambda(i,\ x).\ f\ i\ x) \circ \alpha = (\lambda r.\ f'\ (g\ r)\ r)$
by(*auto simp*: *ha*(*2*) f'*-def*)
also have $... \in qbs\text{-}Mx\ Y$
by(*auto intro*!: *qbs-closed3-dest2'*[*OF assms*(*1*) *hg*,*of* f',*OF 1*])
finally show $(\lambda(i,\ x).\ f\ i\ x) \circ \alpha \in qbs\text{-}Mx\ Y$ **.**
qed

lemma *coPiQ-canonical1'*:
assumes *countable I*
and $\bigwedge i.\ i \in I \Longrightarrow (\lambda x.\ f\ (i,x)) \in X\ i \to_Q Y$
shows $f \in (\Pi_Q\ i \in I.\ X\ i) \to_Q Y$
using *coPiQ-canonical1*[**where** *f*=*curry f*] *assms* **by**(*auto simp*: *curry-def*)

lemma *None-qbs*[*qbs*]: $None \in qbs\text{-}space\ (option\text{-}qbs\ X)$
by(*simp add*: *option-qbs-space*)

lemma *Some-qbs*[*qbs*]: $Some \in X \to_Q option\text{-}qbs\ X$
proof −
have *1*: $Some = (\lambda x.\ case\ x\ of\ Inl\ y \Rightarrow Some\ y \mid Inr\ y \Rightarrow None) \circ Inl$
by *standard auto*
show *?thesis*
unfolding *option-qbs-def*
by(*rule qbs-morphism-cong'*[*OF - qbs-morphism-comp*[*OF Inl-qbs-morphism map-qbs-morphism-f*]]) (*simp add*: *1*)
qed

lemma *case-option-qbs-morphism*[*qbs*]: $case\text{-}option \in qbs\text{-}space\ (Y \Rightarrow_Q (X \Rightarrow_Q Y) \Rightarrow_Q option\text{-}qbs\ X \Rightarrow_Q Y)$
proof(*rule curry-preserves-morphisms*[*OF arg-swap-morphism*])
have $(\lambda x\ y.\ case\ x\ of\ None \Rightarrow fst\ y \mid Some\ z \Rightarrow snd\ y\ z) = (\lambda x\ y.\ case\ x\ of\ Inr\ - \Rightarrow fst\ y \mid Inl\ z \Rightarrow snd\ y\ z) \circ (\lambda z.\ case\ z\ of\ Some\ x \Rightarrow Inl\ x \mid None \Rightarrow Inr\ ())$
by *standard*+ (*simp add*: *option.case-eq-if*)
also have $... \in option\text{-}qbs\ X \to_Q Y \bigotimes_Q (X \Rightarrow_Q Y) \Rightarrow_Q Y$
unfolding *option-qbs-def* **by**(*rule qbs-morphism-comp*[*OF map-qbs-morphism-inverse-f*]) (*auto simp*: *copair-qbs-space*)
finally show $(\lambda x\ y.\ case\ x\ of\ None \Rightarrow fst\ y \mid Some\ x \Rightarrow snd\ y\ x) \in option\text{-}qbs\ X \to_Q Y \bigotimes_Q (X \Rightarrow_Q Y) \Rightarrow_Q Y$ **.**
qed

lemma *rec-option-qbs-morphism*[*qbs*]: $rec\text{-}option \in qbs\text{-}space\ (Y \Rightarrow_Q (X \Rightarrow_Q Y)$

$\Rightarrow_Q$ *option-qbs* $X \Rightarrow_Q Y$)
proof −
have [*simp*]: *rec-option* = *case-option*
by *standard*+ (*metis option.case-eq-if option.exhaust-sel option.simps*(6) *option.simps*(7))
show *?thesis* **by** *simp*
qed

lemma *bind-option-qbs-morphism*[*qbs*]: (≫=) ∈ *qbs-space* (*option-qbs* $X \Rightarrow_Q$ ($X \Rightarrow_Q$ *option-qbs* Y) $\Rightarrow_Q$ *option-qbs* Y)
by(*simp add*: *Option.bind-def*)

lemma *Let-qbs-morphism*[*qbs*]: *Let* ∈ $X \Rightarrow_Q (X \Rightarrow_Q Y) \Rightarrow_Q Y$
proof −
have [*simp*]:*Let* = $(\lambda x\ f.\ f\ x)$ **by** *standard*+ *auto*
show *?thesis* **by** *simp*
qed

end

3.3 Relation to Measurable Spaces

theory *Measure-QuasiBorel-Adjunction*
imports *QuasiBorel QBS-Morphism Lemmas-S-Finite-Measure-Monad*
begin

We construct the adjunction between **Meas** and **QBS**, where **Meas** is the category of measurable spaces and measurable functions, and **QBS** is the category of quasi-Borel spaces and morphisms.

3.3.1 The Functor R

definition *measure-to-qbs* :: *'a measure* ⇒ *'a quasi-borel* **where**
measure-to-qbs $X \equiv$ *Abs-quasi-borel* (*space* X, *borel* $\rightarrow_M X$)

declare [[*coercion measure-to-qbs*]]

lemma
shows *qbs-space-R*: *qbs-space* (*measure-to-qbs* X) = *space* X (**is** *?goal1*)
and *qbs-Mx-R*: *qbs-Mx* (*measure-to-qbs* X) = *borel* $\rightarrow_M X$ (**is** *?goal2*)
proof −
have *Rep-quasi-borel* (*measure-to-qbs* X) = (*space* X, *borel* $\rightarrow_M X$)
by(*auto intro*!: *Abs-quasi-borel-inverse is-quasi-borel-intro qbs-closed1I qbs-closed2I simp*: *measure-to-qbs-def dest*:*measurable-space*) (*rule qbs-closed3I*, *auto*)
thus *?goal1 ?goal2*
by (*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

The following lemma says that *measure-to-qbs* is a functor from **Meas** to **QBS**.

lemma *r-preserves-morphisms*:
$X \rightarrow_M Y \subseteq (\textit{measure-to-qbs}\ X) \rightarrow_Q (\textit{measure-to-qbs}\ Y)$
by(*auto intro*!: *qbs-morphismI simp*: *qbs-Mx-R*)

lemma *measurable-imp-qbs-morphism*: $f \in M \rightarrow_M N \Longrightarrow f \in M \rightarrow_Q N$
using *r-preserves-morphisms* **by** *blast*

3.3.2 The Functor L

definition *sigma-Mx* :: *'a quasi-borel* ⇒ *'a set set* **where**
sigma-Mx X ≡ {*U* ∩ *qbs-space X* |*U*. ∀ α∈*qbs-Mx X*. α −‘ *U* ∈ *sets borel*}

definition *qbs-to-measure* :: *'a quasi-borel* ⇒ *'a measure* **where**
qbs-to-measure X ≡ *Abs-measure* (*qbs-space X*, *sigma-Mx X*, λ*A*. (*if A* = {} *then 0 else if A* ∈ − *sigma-Mx X then 0 else* ∞))

lemma *measure-space-L*: *measure-space* (*qbs-space X*) (*sigma-Mx X*) (λ*A*. (*if A* = {} *then 0 else if A* ∈ − *sigma-Mx X then 0 else* ∞))
unfolding *measure-space-def*
proof *safe*

show *sigma-algebra* (*qbs-space X*) (*sigma-Mx X*)
proof(*rule sigma-algebra.intro*)
show *algebra* (*qbs-space X*) (*sigma-Mx X*)
proof
have ∀ *U* ∈ *sigma-Mx X*. *U* ⊆ *qbs-space X*
using *sigma-Mx-def subset-iff* **by** *fastforce*
thus *sigma-Mx X* ⊆ *Pow* (*qbs-space X*) **by** *auto*
next
show {} ∈ *sigma-Mx X*
unfolding *sigma-Mx-def* **by** *auto*
next
fix *A*
fix *B*
assume *A* ∈ *sigma-Mx X*
B ∈ *sigma-Mx X*
then have ∃ *Ua*. *A* = *Ua* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ *Ua* ∈ *sets borel*)
by (*simp add*: *sigma-Mx-def*)
then obtain *Ua* **where** *pa*:*A* = *Ua* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ *Ua* ∈ *sets borel*) **by** *auto*
have ∃ *Ub*. *B* = *Ub* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ *Ub* ∈ *sets borel*)
using ‹*B* ∈ *sigma-Mx X*› *sigma-Mx-def* **by** *auto*
then obtain *Ub* **where** *pb*:*B* = *Ub* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ *Ub* ∈ *sets borel*) **by** *auto*
from *pa pb* **have** [*simp*]:∀ α∈*qbs-Mx X*. α −‘ (*Ua* ∩ *Ub*) ∈ *sets borel*
by *auto*
from *this pa pb sigma-Mx-def* **have** [*simp*]:(*Ua* ∩ *Ub*) ∩ *qbs-space X* ∈ *sigma-Mx X* **by** *blast*

from *pa pb* **have** [*simp*]:$A \cap B = (Ua \cap Ub) \cap$ *qbs-space X* **by** *auto*
thus $A \cap B \in$ *sigma-Mx X* **by** *simp*
next
fix *A*
fix *B*
assume $A \in$ *sigma-Mx X*
$B \in$ *sigma-Mx X*
then have $\exists$ *Ua*. $A = Ua \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ua \in$ *sets borel*)
by (*simp add*: *sigma-Mx-def*)
then obtain *Ua* **where** *pa*:$A = Ua \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ua \in$ *sets borel*) **by** *auto*
have $\exists$ *Ub*. $B = Ub \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ub \in$ *sets borel*)
using ‹$B \in$ *sigma-Mx X*› *sigma-Mx-def* **by** *auto*
then obtain *Ub* **where** *pb*:$B = Ub \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ub \in$ *sets borel*) **by** *auto*
from *pa pb* **have** [*simp*]:$A - B = (Ua \cap -Ub) \cap$ *qbs-space X* **by** *auto*
from *pa pb* **have** $\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘$(Ua \cap -Ub) \in$ *sets borel*
by (*metis Diff-Compl double-compl sets.Diff vimage-Compl vimage-Int*)
hence *1*:$A - B \in$ *sigma-Mx X*
using *sigma-Mx-def* ‹$A - B = Ua \cap - Ub \cap$ *qbs-space X*› **by** *blast*
show $\exists C \subseteq$*sigma-Mx X*. *finite C* $\wedge$ *disjoint C* $\wedge$ $A - B = \bigcup C$
proof
show $\{A - B\} \subseteq$*sigma-Mx X* $\wedge$ *finite* $\{A-B\}$ $\wedge$ *disjoint* $\{A-B\}$ $\wedge$ $A - B = \bigcup \{A-B\}$
using *1* **by** *auto*
qed
next
fix *A*
fix *B*
assume $A \in$ *sigma-Mx X*
$B \in$ *sigma-Mx X*
then have $\exists$ *Ua*. $A = Ua \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ua \in$ *sets borel*)
by (*simp add*: *sigma-Mx-def*)
then obtain *Ua* **where** *pa*:$A = Ua \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ua \in$ *sets borel*) **by** *auto*
have $\exists$ *Ub*. $B = Ub \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ub \in$ *sets borel*)
using ‹$B \in$ *sigma-Mx X*› *sigma-Mx-def* **by** *auto*
then obtain *Ub* **where** *pb*:$B = Ub \cap$ *qbs-space X* $\wedge$ ($\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $Ub \in$ *sets borel*) **by** *auto*
from *pa pb* **have** $A \cup B = (Ua \cup Ub) \cap$ *qbs-space X* **by** *auto*
from *pa pb* **have** $\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘$(Ua \cup Ub) \in$ *sets borel* **by** *auto*
then show $A \cup B \in$ *sigma-Mx X*
unfolding *sigma-Mx-def*
using ‹$A \cup B = (Ua \cup Ub) \cap$ *qbs-space X*› **by** *blast*
next
have $\forall \alpha \in$ *qbs-Mx X*. $\alpha -$‘ $(UNIV) \in$ *sets borel*
by *simp*

thus *qbs-space X ∈ sigma-Mx X*
unfolding *sigma-Mx-def*
by *blast*
qed
next
show *sigma-algebra-axioms* (*sigma-Mx X*)
unfolding *sigma-algebra-axioms-def*
proof *safe*
fix *A* :: *nat* ⇒ -
assume *1*:*range A* ⊆ *sigma-Mx X*
then have *2*:∀ *i*. ∃ *Ui*. *A i* = *Ui* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ *Ui* ∈ *sets borel*)
unfolding *sigma-Mx-def* **by** *auto*
then have ∃ *U* :: *nat* ⇒ -. ∀ *i*. *A i* = *U i* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ (*U i*) ∈ *sets borel*)
by (*rule choice*)
from *this* **obtain** *U* **where** *pu*:∀ *i*. *A i* = *U i* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −‘ (*U i*) ∈ *sets borel*)
by *auto*
hence ∀ α∈*qbs-Mx X*. α −‘ (⋃ (*range U*)) ∈ *sets borel*
by (*simp add*: *countable-Un-Int*(*1*) *vimage-UN*)
from *pu* **have** ⋃ (*range A*) = (⋃ *i*::*nat*. (*U i* ∩ *qbs-space X*)) **by** *blast*
hence ⋃ (*range A*) = ⋃ (*range U*) ∩ *qbs-space X* **by** *auto*
thus ⋃ (*range A*) ∈ *sigma-Mx X*
using *sigma-Mx-def* ‹∀ α∈*qbs-Mx X*. α −‘ ⋃ (*range U*) ∈ *sets borel*› **by** *blast*
qed
qed
next
show *countably-additive* (*sigma-Mx X*) (λ*A*. *if A* = {} *then 0 else if A* ∈ − *sigma-Mx X then 0 else* ∞)
proof(*rule countably-additiveI*)
fix *A* :: *nat* ⇒ -
assume *h*:*range A* ⊆ *sigma-Mx X*
⋃ (*range A*) ∈ *sigma-Mx X*
consider ⋃ (*range A*) = {} | ⋃ (*range A*) ≠ {}
by *auto*
then show (∑ *i*. *if A i* = {} *then 0 else if A i* ∈ − *sigma-Mx X then 0 else* ∞) =
(*if* ⋃ (*range A*) = {} *then 0 else if* ⋃ (*range A*) ∈ − *sigma-Mx X then 0 else* (∞ :: *ennreal*))
proof *cases*
case *1*
then have ⋀*i*. *A i* = {}
by *simp*
thus *?thesis*
by(*simp add*: *1*)
next
case *2*

then obtain j **where** hj:$A\ j \neq \{\}$
by *auto*
have $(\sum i.\ if\ A\ i = \{\}\ then\ 0\ \ else\ if\ A\ i \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ \infty) = (\infty :: ennreal)$
proof −
have $hsum$:$\bigwedge N\ f.\ sum\ f\ \{..<N\} \leq (\sum n.\ (f\ n :: ennreal))$
by (*simp add*: *sum-le-suminf*)
have $hsum'$:$\bigwedge P\ f.\ (\exists j.\ j \in P \wedge f\ j = (\infty :: ennreal)) \Longrightarrow finite\ P \Longrightarrow sum\ f\ P = \infty$
by *auto*
have $h1$:$(\sum i<j+1.\ if\ A\ i = \{\}\ then\ 0\ else\ if\ A\ i \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ \infty) = (\infty :: ennreal)$
proof(*rule hsum'*)
show $\exists ja.\ ja \in \{..<j\ +\ 1\} \wedge (if\ A\ ja = \{\}\ then\ 0\ else\ if\ A\ ja \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ \infty) = (\infty :: ennreal)$
proof(*rule exI*[**where** x=j],*rule conjI*)
have $A\ j \in sigma\text{-}Mx\ X$
using $h(1)$ **by** *auto*
then show $(if\ A\ j = \{\}\ then\ 0\ else\ if\ A\ j \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ \infty) = (\infty :: ennreal)$
using hj **by** *simp*
qed *simp*
qed *simp*
have $(\sum i<j+1.\ if\ A\ i = \{\}\ then\ 0\ else\ if\ A\ i \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ \infty) \leq (\sum i.\ if\ A\ i = \{\}\ then\ 0\ else\ if\ A\ i \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ (\infty :: ennreal))$
by(*rule hsum*)
thus *?thesis*
by(*simp only*: *h1*) (*simp add*: *top.extremum-unique*)
qed
moreover have $(if\ \bigcup\ (range\ A) = \{\}\ then\ 0\ else\ if\ \bigcup\ (range\ A) \in -\ sigma\text{-}Mx\ X\ then\ 0\ else\ \infty) = (\infty :: ennreal)$
using *2* $h(2)$ **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed
qed(*simp add*: *positive-def*)

lemma
shows *space-L*: *space* (*qbs-to-measure* X) = *qbs-space* X (**is** *?goal1*)
and *sets-L*: *sets* (*qbs-to-measure* X) = *sigma-Mx* X (**is** *?goal2*)
and *emeasure-L*: *emeasure* (*qbs-to-measure* X) = $(\lambda A.\ if\ A = \{\} \vee A \notin sigma\text{-}Mx\ X\ then\ 0\ else\ \infty)$ (**is** *?goal3*)
proof −
have *Rep-measure* (*qbs-to-measure* X) = (*qbs-space* X, *sigma-Mx* X, $\lambda A.$ (*if* $A = \{\}$ *then 0 else if* $A \in -$ *sigma-Mx* X *then 0 else* ∞))
unfolding *qbs-to-measure-def* **by**(*auto intro*!: *Abs-measure-inverse simp*: *measure-space-L*)

thus *?goal1 ?goal2 ?goal3*
by(*auto simp*: *sets-def space-def emeasure-def*)
qed

lemma *qbs-Mx-sigma-Mx-contra*:
assumes *qbs-space X* = *qbs-space Y*
and *qbs-Mx X* ⊆ *qbs-Mx Y*
shows *sigma-Mx Y* ⊆ *sigma-Mx X*
using *assms* **by**(*auto simp*: *sigma-Mx-def*)

The following lemma says that *qbs-to-measure* is a functor from **QBS** to **Meas**.

lemma *l-preserves-morphisms*:
X →$_Q$ *Y* ⊆ (*qbs-to-measure X*) →$_M$ (*qbs-to-measure Y*)
proof *safe*
fix *f*
assume *h*:*f* ∈ *X* →$_Q$ *Y*
show *f* ∈ (*qbs-to-measure X*) →$_M$ (*qbs-to-measure Y*)
proof(*rule measurableI*)
fix *A*
assume *A* ∈ *sets* (*qbs-to-measure Y*)
then obtain *Ua* **where** *pa*:*A* = *Ua* ∩ *qbs-space Y* ∧ (∀ α∈*qbs-Mx Y*. α −‘ *Ua* ∈ *sets borel*)
by (*auto simp*: *sigma-Mx-def sets-L*)
have ∀ α∈*qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*
∀ α∈ *qbs-Mx X*. α −‘ (*f* −‘ (*qbs-space Y*)) = *UNIV*
using *qbs-morphism-space*[*OF h*] *qbs-morphism-Mx*[*OF h*] **by** (*auto simp*: *qbs-Mx-to-X*)
hence ∀ α∈*qbs-Mx X*. α −‘ (*f* −‘ *A*) = α −‘ (*f* −‘ *Ua*)
by (*simp add*: *pa*)
from *pa this qbs-morphism-def* **have** ∀ α∈*qbs-Mx X*. α −‘ (*f* −‘ *A*) ∈ *sets borel*
by (*simp add*: *vimage-comp* ‹∀ α∈*qbs-Mx X*. *f* ∘ α ∈ *qbs-Mx Y*›)
thus *f* −‘ *A* ∩ *space* (*qbs-to-measure X*) ∈ *sets* (*qbs-to-measure X*)
using *sigma-Mx-def* **by**(*auto simp*: *sets-L space-L*)
qed (*insert qbs-morphism-space*[*OF h*], *auto simp*: *space-L*)
qed

lemma *qbs-morphism-imp-measurable*: *f* ∈ *X* →$_Q$ *Y* ⟹ *f* ∈ *qbs-to-measure X* →$_M$ *qbs-to-measure Y*
using *l-preserves-morphisms* **by** *blast*

abbreviation *qbs-borel* (*borel*$_Q$) **where** *borel*$_Q$ ≡ *measure-to-qbs borel*
abbreviation *qbs-count-space* (*count′-space*$_Q$) **where** *qbs-count-space I* ≡ *measure-to-qbs* (*count-space I*)

lemma
shows *qbs-space-qbs-borel*[*simp*]: *qbs-space borel*$_Q$ = *UNIV*
and *qbs-space-count-space*[*simp*]: *qbs-space* (*qbs-count-space I*) = *I*
and *qbs-Mx-qbs-borel*: *qbs-Mx borel*$_Q$ = *borel-measurable borel*

and *qbs-Mx-count-space*: *qbs-Mx* (*qbs-count-space I*) = *borel* $\rightarrow_M$ *count-space I*
by(*simp-all add*: *qbs-space-R qbs-Mx-R*)

lemma
shows *qbs-space-qbs-borel'*[*qbs*]: $r \in$ *qbs-space borel*$_Q$
and *qbs-space-count-space-UNIV'*[*qbs*]: $x \in$ *qbs-space* (*qbs-count-space* (*UNIV* :: (- :: *countable*) *set*))
by *simp-all*

lemma *qbs-Mx-is-morphisms*: *qbs-Mx X* = *borel*$_Q$ $\rightarrow_Q$ *X*
proof *safe*
fix α :: *real* $\Rightarrow$ -
assume $\alpha \in$ *borel*$_Q$ $\rightarrow_Q$ *X*
have *id* $\in$ *qbs-Mx borel*$_Q$ **by** (*simp add*: *qbs-Mx-R*)
then have $\alpha \circ$ *id* $\in$ *qbs-Mx X*
using *qbs-morphism-Mx*[*OF* ‹$\alpha \in$ *borel*$_Q$ $\rightarrow_Q$ *X*›]
by *blast*
then show $\alpha \in$ *qbs-Mx X* **by** *simp*
qed(*auto intro*!: *qbs-morphismI simp*: *qbs-Mx-qbs-borel*)

lemma *exp-qbs-Mx'*: *qbs-Mx* (*exp-qbs X Y*) = {*g*. *case-prod g* $\in$ *borel*$_Q$ $\bigotimes_Q$ *X* $\rightarrow_Q$ *Y*}
by(*auto simp*: *qbs-Mx-qbs-borel comp-def qbs-Mx-is-morphisms split-beta' intro*!:*curry-preserves-morphisms*)

lemma *arg-swap-morphism'*:
assumes (λg. *f* ($\lambda w\ x$. *g x w*)) $\in$ *exp-qbs X* (*exp-qbs W Y*) $\rightarrow_Q$ *Z*
shows *f* $\in$ *exp-qbs W* (*exp-qbs X Y*) $\rightarrow_Q$ *Z*
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* (*exp-qbs W* (*exp-qbs X Y*))
then have ($\lambda((r,w),x)$. α *r w x*) $\in$ (*borel*$_Q$ $\bigotimes_Q$ *W*) $\bigotimes_Q$ *X* $\rightarrow_Q$ *Y*
by(*auto simp*: *qbs-Mx-is-morphisms dest*: *uncurry-preserves-morphisms*)
hence ($\lambda(r,w,x)$. α *r w x*) $\in$ *borel*$_Q$ $\bigotimes_Q$ *W* $\bigotimes_Q$ *X* $\rightarrow_Q$ *Y*
by(*auto intro*!: *qbs-morphism-cong'*[**where** *f*=($\lambda((r,w),x)$. α *r w x*) $\circ$ ($\lambda(x, y, z)$. ((*x*, *y*), *z*)) **and** *g*=$\lambda(r,w,x)$. α *r w x*] *qbs-morphism-comp*[*OF qbs-morphism-pair-assoc2*])
hence ($\lambda(r,x,w)$. α *r w x*) $\in$ *borel*$_Q$ $\bigotimes_Q$ *X* $\bigotimes_Q$ *W* $\rightarrow_Q$ *Y*
by(*auto intro*!: *qbs-morphism-cong'*[**where** *f*=($\lambda(r,w,x)$. α *r w x*) $\circ$ *map-prod id* ($\lambda(x,y)$. (*y*,*x*)) **and** *g*=($\lambda(r,x,w)$. α *r w x*)] *qbs-morphism-comp qbs-morphism-map-prod qbs-morphism-pair-swap*)
hence ($\lambda((r,x),w)$. α *r w x*) $\in$ (*borel*$_Q$ $\bigotimes_Q$ *X*) $\bigotimes_Q$ *W* $\rightarrow_Q$ *Y*
by(*auto intro*!: *qbs-morphism-cong'*[**where** *f*=($\lambda(r,x,w)$. α *r w x*) $\circ$ ($\lambda((x, y), z)$. (*x*, *y*, *z*)) **and** *g*=$\lambda((r,x),w)$. α *r w x*] *qbs-morphism-comp*[*OF qbs-morphism-pair-assoc1*])
hence ($\lambda r\ x\ w$. α *r w x*) $\in$ *qbs-Mx* (*exp-qbs X* (*exp-qbs W Y*))
by(*auto simp*: *qbs-Mx-is-morphisms split-beta'*)
from *qbs-morphism-Mx*[*OF assms this*] **show** *f* $\circ$ α $\in$ *qbs-Mx Z*
by(*auto simp*: *comp-def*)
qed

lemma *qbs-Mx-subset-of-measurable*: *qbs-Mx X* $\subseteq$ *borel* $\rightarrow_M$ *qbs-to-measure X*
proof
fix α
assume $\alpha \in$ *qbs-Mx X*
show $\alpha \in$ *borel* $\rightarrow_M$ *qbs-to-measure X*
proof(*rule measurableI*)
fix *x*
show α *x* $\in$ *space* (*qbs-to-measure X*)
using *qbs-Mx-to-X* ‹$\alpha \in$ *qbs-Mx X*› **by**(*simp add*: *space-L*)
next
fix *A*
assume *A* $\in$ *sets* (*qbs-to-measure X*)
then have $\alpha -$‘(*qbs-space X*) = *UNIV*
using ‹$\alpha \in$ *qbs-Mx X*› *qbs-Mx-to-X* **by**(*auto simp*: *sets-L*)
then show $\alpha -$‘ *A* $\cap$ *space borel* $\in$ *sets borel*
using ‹$\alpha \in$ *qbs-Mx X*› ‹*A* $\in$ *sets* (*qbs-to-measure X*)›
by(*auto simp add*: *sigma-Mx-def sets-L*)
qed
qed

lemma *L-max-of-measurables*:
assumes *space M* = *qbs-space X*
and *qbs-Mx X* $\subseteq$ *borel* $\rightarrow_M$ *M*
shows *sets M* $\subseteq$ *sets* (*qbs-to-measure X*)
proof
fix *U*
assume *U* $\in$ *sets M*
from *sets.sets-into-space*[*OF this*] *in-mono*[*OF assms*(*2*)] *measurable-sets-borel*[*OF - this*]
show *U* $\in$ *sets* (*qbs-to-measure X*)
using *assms*(*1*)
by(*auto intro*!: *exI*[**where** *x*=*U*] *simp*: *sigma-Mx-def sets-L*)
qed

lemma *qbs-Mx-are-measurable*[*simp*,*measurable*]:
assumes $\alpha \in$ *qbs-Mx X*
shows $\alpha \in$ *borel* $\rightarrow_M$ *qbs-to-measure X*
using *assms qbs-Mx-subset-of-measurable* **by** *auto*

lemma *measure-to-qbs-cong-sets*:
assumes *sets M* = *sets N*
shows *measure-to-qbs M* = *measure-to-qbs N*
by(*rule qbs-eqI*) (*simp add*: *qbs-Mx-R measurable-cong-sets*[*OF - assms*])

lemma *lr-sets*[*simp*]:
sets X $\subseteq$ *sets* (*qbs-to-measure* (*measure-to-qbs X*))
unfolding *sets-L*
proof *safe*

fix U
assume $U \in sets\ X$
then have $U \cap space\ X = U$ **by** *simp*
moreover have $\forall \alpha{\in}borel \rightarrow_M X.\ \alpha -` U \in sets\ borel$
using ‹$U \in sets\ X$› **by**(*auto simp add*: *measurable-def*)
ultimately show $U \in sigma\text{-}Mx\ (measure\text{-}to\text{-}qbs\ X)$
by(*auto simp add*: *sigma-Mx-def qbs-Mx-R qbs-space-R*)
qed

lemma(**in** *standard-borel*) *lr-sets-ident*[*simp*, *measurable-cong*]:
$sets\ (qbs\text{-}to\text{-}measure\ (measure\text{-}to\text{-}qbs\ M)) = sets\ M$
unfolding *sets-L*
proof *safe*
fix V
assume $V \in sigma\text{-}Mx\ (measure\text{-}to\text{-}qbs\ M)$
then obtain U **where** *H2*: $V = U \cap space\ M \bigwedge \alpha{::}real \Rightarrow$ -. $\alpha{\in}borel \rightarrow_M M \Longrightarrow \alpha -` U \in sets\ borel$
by(*auto simp*: *sigma-Mx-def qbs-Mx-R qbs-space-R*)
consider $space\ M = \{\} \mid space\ M \neq \{\}$ **by** *auto*
then show $V \in sets\ M$
proof *cases*
case *1*
then show *?thesis*
by(*simp add*: *H2*)
next
case *2*
have $from\text{-}real -` V = from\text{-}real -` (U \cap space\ M)$ **using** *H2* **by** *auto*
also have ... = $from\text{-}real -` U$ **using** *from-real-measurable'*[*OF 2*] **by**(*auto simp add*: *measurable-def*)
finally have $to\text{-}real -` from\text{-}real -` U \cap space\ M \in sets\ M$
by (*meson 2 H2*(*2*) *from-real-measurable' measurable-sets to-real-measurable*)
moreover have $to\text{-}real -` from\text{-}real -` U \cap space\ M = U \cap space\ M$
by *auto*
ultimately show *?thesis* **using** *H2* **by** *simp*
qed
qed(*insert lr-sets*, *auto simp*: *sets-L*)

corollary *sets-lr-polish-borel*[*simp*, *measurable-cong*]: $sets\ (qbs\text{-}to\text{-}measure\ qbs\text{-}borel) = sets\ (borel :: (- :: polish\text{-}space)\ measure)$
by(*auto intro*!: *standard-borel.lr-sets-ident standard-borel-ne.standard-borel*)

corollary *sets-lr-count-space*[*simp*, *measurable-cong*]: $sets\ (qbs\text{-}to\text{-}measure\ (qbs\text{-}count\text{-}space\ (UNIV :: (- :: countable)\ set))) = sets\ (count\text{-}space\ UNIV)$
by(*rule standard-borel.lr-sets-ident*) (*auto intro*!: *standard-borel-ne.standard-borel*)

lemma *map-qbs-embed-measure1*:
assumes *inj-on f* (*space M*)
shows *map-qbs f* (*measure-to-qbs M*) = *measure-to-qbs* (*embed-measure M f*)
proof(*safe intro*!: *qbs-eqI*)

fix α
show $\alpha \in$ *qbs-Mx* (*map-qbs f* (*measure-to-qbs M*)) $\Longrightarrow \alpha \in$ *qbs-Mx* (*measure-to-qbs* (*embed-measure M f*))
using *measurable-embed-measure2'*[*OF assms*] **by**(*auto intro*!: *qbs-eqI simp*: *map-qbs-Mx qbs-Mx-R*)
next
fix α
assume $\alpha \in$ *qbs-Mx* (*measure-to-qbs* (*embed-measure M f*))
hence *h*[*measurable*]:$\alpha \in$ *borel* $\rightarrow_M$ *embed-measure M f*
by(*simp add*: *qbs-Mx-R*)
have [*measurable*]:*the-inv-into* (*space M*) *f* $\in$ *embed-measure M f* $\rightarrow_M$ *M*
by (*simp add*: *assms in-sets-embed-measure measurable-def sets.sets-into-space space-embed-measure the-inv-into-into the-inv-into-vimage*)
show $\alpha \in$ *qbs-Mx* (*map-qbs f* (*measure-to-qbs M*))
using *measurable-space*[*OF h*] *f-the-inv-into-f*[*OF assms*] **by**(*auto intro*!: *exI*[**where** *x*=*the-inv-into* (*space M*) *f* $\circ$ α] *simp*: *map-qbs-Mx qbs-Mx-R space-embed-measure*)
qed

lemma *map-qbs-embed-measure2*:
assumes *inj-on f* (*qbs-space X*)
shows *sets* (*qbs-to-measure* (*map-qbs f X*)) = *sets* (*embed-measure* (*qbs-to-measure X*) *f*)
proof *safe*
fix *A*
assume *A* $\in$ *sets* (*qbs-to-measure* (*map-qbs f X*))
then obtain *U* **where** *A* = *U* $\cap$ *f* ‘ *qbs-space X* $\bigwedge\alpha$. $\alpha \in$ *qbs-Mx X* $\Longrightarrow$ ($f \circ \alpha$) −‘ *U* $\in$ *sets borel*
by(*auto simp*: *sets-L sigma-Mx-def map-qbs-space map-qbs-Mx*)
thus *A* $\in$ *sets* (*embed-measure* (*qbs-to-measure X*) *f*)
by(*auto simp*: *sets-L sets-embed-measure'*[*of - qbs-to-measure X*,*simplified space-L*,*OF assms*] *sigma-Mx-def qbs-Mx-to-X vimage-def intro*!: *exI*[**where** *x*=*f* −‘ *U* $\cap$ *qbs-space X*])
next
fix *A*
assume *A*:*A* $\in$ *sets* (*embed-measure* (*qbs-to-measure X*) *f*)
then obtain *B U* **where** *A* = *f* ‘ *B B* = *U* $\cap$ *qbs-space X* $\bigwedge\alpha$. $\alpha \in$ *qbs-Mx X* $\Longrightarrow$ α −‘ *U* $\in$ *sets borel*
by(*auto simp*: *sets-L sigma-Mx-def sets-embed-measure'*[*of - qbs-to-measure X*,*simplified space-L*,*OF assms*])
thus *A* $\in$ *sets* (*qbs-to-measure* (*map-qbs f X*))
using *measurable-sets-borel*[*OF measurable-comp*[*OF qbs-Mx-are-measurable measurable-embed-measure2'*[*of - qbs-to-measure X*,*simplified space-L*,*OF assms*]]] *A*
by(*auto simp*: *sets-L sigma-Mx-def map-qbs-space map-qbs-Mx intro*!: *exI*[**where** *x*=*f* ‘ (*U* $\cap$ *qbs-space X*)])
qed

lemma(**in** *standard-borel*) *map-qbs-embed-measure2'*:
assumes *inj-on f* (*space M*)

shows *sets* (*qbs-to-measure* (*map-qbs f* (*measure-to-qbs M*))) = *sets* (*embed-measure M f*)
by(*auto intro*!: *standard-borel.lr-sets-ident*[*OF standard-borel-embed-measure*] *assms simp*: *map-qbs-embed-measure1*[*OF assms*])

3.3.3 The Adjunction

lemma *lr-adjunction-correspondence* :
$X \to_Q$ (*measure-to-qbs* Y) = (*qbs-to-measure* X) $\to_M$ Y
proof *safe*

fix f
assume $f \in X \to_Q$ (*measure-to-qbs* Y)
show $f \in$ *qbs-to-measure* $X \to_M$ Y
proof(*rule measurableI*)
fix x
assume $x \in$ *space* (*qbs-to-measure* X)
thus $f\ x \in$ *space* Y
using *qbs-morphism-space*[*OF* ‹$f \in X \to_Q$ (*measure-to-qbs* Y)›]
by (*auto simp*: *qbs-space-R space-L*)
next
fix A
assume $A \in$ *sets* Y
have $\forall \alpha \in$ *qbs-Mx* X. $f \circ \alpha \in$ *qbs-Mx* (*measure-to-qbs* Y)
using *qbs-morphism-Mx*[*OF* ‹$f \in X \to_Q$ (*measure-to-qbs* Y)›] **by** *auto*
hence $\forall \alpha \in$ *qbs-Mx* X. $f \circ \alpha \in$ *borel* $\to_M$ Y **by** (*simp add*: *qbs-Mx-R*)
hence $\forall \alpha \in$ *qbs-Mx* X. $\alpha -`$ ($f -`$ A) $\in$ *sets borel*
using ‹$A \in$ *sets* Y› *measurable-sets-borel vimage-comp* **by** *metis*
thus $f -`$ $A \cap$ *space* (*qbs-to-measure* X) $\in$ *sets* (*qbs-to-measure* X)
using *sigma-Mx-def* **by** (*auto simp*: *space-L sets-L*)
qed

next
fix f
assume $f \in$ *qbs-to-measure* $X \to_M$ Y
show $f \in X \to_Q$ *measure-to-qbs* Y
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* X
have $f \circ \alpha \in$ *borel* $\to_M$ Y
proof(*rule measurableI*)
fix x :: *real*
from ‹$\alpha \in$ *qbs-Mx* X› *qbs-Mx-to-X* **have** $\alpha\ x \in$ *qbs-space* X **by** *auto*
hence $\alpha\ x \in$ *space* (*qbs-to-measure* X) **by** (*simp add*: *space-L*)
thus $(f \circ \alpha)\ x \in$ *space* Y
using ‹$f \in$ *qbs-to-measure* $X \to_M$ Y›
by (*metis comp-def measurable-space*)
next

fix A
assume $A \in sets\ Y$
from ‹$f \in qbs\text{-}to\text{-}measure\ X \to_M\ Y$› *measurable-sets this measurable-def*
have $f -` A \cap space\ (qbs\text{-}to\text{-}measure\ X) \in sets\ (qbs\text{-}to\text{-}measure\ X)$
by *blast*
hence $f -` A \cap qbs\text{-}space\ X \in sigma\text{-}Mx\ X$ **by** (*simp add*: *sets-L space-L*)
then have $\exists V.\ f -` A \cap qbs\text{-}space\ X = V \cap qbs\text{-}space\ X \wedge (\forall \beta \in qbs\text{-}Mx\ X.\ \beta -` V \in sets\ borel)$
by (*simp add*: *sigma-Mx-def*)
then obtain V **where** $h{:}f -` A \cap qbs\text{-}space\ X = V \cap qbs\text{-}space\ X \wedge (\forall \beta \in qbs\text{-}Mx\ X.\ \beta -` V \in sets\ borel)$ **by** *auto*
have *1*:$\alpha -` (f -` A) = \alpha -` (f -` A \cap qbs\text{-}space\ X)$
using ‹$\alpha \in qbs\text{-}Mx\ X$› *qbs-Mx-to-X* **by** *blast*
have *2*:$\alpha -` (V \cap qbs\text{-}space\ X) = \alpha -` V$
using ‹$\alpha \in qbs\text{-}Mx\ X$› *qbs-Mx-to-X* **by** *blast*
from *1 2 h* **have** $(f \circ \alpha) -` A = \alpha -` V$ **by** (*simp add*: *vimage-comp*)
from *this h* ‹$\alpha \in qbs\text{-}Mx\ X$›**show** $(f \circ \alpha) -` A \cap space\ borel \in sets\ borel$ **by** *simp*
qed
thus $f \circ \alpha \in qbs\text{-}Mx\ (measure\text{-}to\text{-}qbs\ Y)$
by(*simp add*:*qbs-Mx-R*)
qed
qed

lemma(**in** *standard-borel*) *standard-borel-r-full-faithful*:
$M \to_M\ Y = measure\text{-}to\text{-}qbs\ M \to_Q measure\text{-}to\text{-}qbs\ Y$
proof
have $measure\text{-}to\text{-}qbs\ M \to_Q measure\text{-}to\text{-}qbs\ Y \subseteq qbs\text{-}to\text{-}measure\ (measure\text{-}to\text{-}qbs\ M) \to_M qbs\text{-}to\text{-}measure\ (measure\text{-}to\text{-}qbs\ Y)$
by (*simp add*: *l-preserves-morphisms*)
also have $... = M \to_M qbs\text{-}to\text{-}measure\ (measure\text{-}to\text{-}qbs\ Y)$
using *measurable-cong-sets* **by** *auto*
also have $... \subseteq M \to_M\ Y$
by(*rule measurable-mono*[*OF lr-sets*]) (*simp-all add*: *qbs-space-R space-L*)
finally show $measure\text{-}to\text{-}qbs\ M \to_Q measure\text{-}to\text{-}qbs\ Y \subseteq M \to_M\ Y$ **.**
qed(*rule r-preserves-morphisms*)

lemma *qbs-morphism-dest*[*measurable-dest*]:
assumes $f \in X \to_Q measure\text{-}to\text{-}qbs\ Y$
shows $f \in qbs\text{-}to\text{-}measure\ X \to_M\ Y$
using *assms lr-adjunction-correspondence* **by** *auto*

lemma(**in** *standard-borel*) *qbs-morphism-dest*:
assumes $k \in measure\text{-}to\text{-}qbs\ M \to_Q measure\text{-}to\text{-}qbs\ Y$
shows $k \in M \to_M\ Y$
using *standard-borel-r-full-faithful assms* **by** *auto*

lemma *qbs-morphism-measurable-intro*:
assumes $f \in qbs\text{-}to\text{-}measure\ X \to_M\ Y$

shows $f \in X \rightarrow_Q$ *measure-to-qbs* Y
using *assms lr-adjunction-correspondence* **by** *auto*

lemma(**in** *standard-borel*) *qbs-morphism-measurable-intro*:
assumes $k \in M \rightarrow_M Y$
shows $k \in$ *measure-to-qbs* $M \rightarrow_Q$ *measure-to-qbs* Y
using *standard-borel-r-full-faithful assms* **by** *auto*

lemma *r-preserves-product* :
measure-to-qbs $(X \bigotimes_M Y) =$ *measure-to-qbs* $X \bigotimes_Q$ *measure-to-qbs* Y
by(*auto intro*!: *qbs-eqI simp*: *measurable-pair-iff pair-qbs-Mx qbs-Mx-R*)

lemma *l-product-sets*:
sets (*qbs-to-measure* $X \bigotimes_M$ *qbs-to-measure* Y) $\subseteq$ *sets* (*qbs-to-measure* ($X \bigotimes_Q$ Y))
proof(*rule sets-pair-in-sets*)
fix A B
assume h:$A \in$ *sets* (*qbs-to-measure* X) $B \in$ *sets* (*qbs-to-measure* Y)
then obtain *Ua Ub* **where** *hu*:
$A = Ua \cap$ *qbs-space* X $\forall \alpha \in$ *qbs-Mx* X. $\alpha -` Ua \in$ *sets borel*
$B = Ub \cap$ *qbs-space* Y $\forall \alpha \in$ *qbs-Mx* Y. $\alpha -` Ub \in$ *sets borel*
by(*auto simp add*: *sigma-Mx-def sets-L*)
show $A \times B \in$ *sets* (*qbs-to-measure* ($X \bigotimes_Q Y$))
proof −
have $A \times B = Ua \times Ub \cap$ *qbs-space* ($X \bigotimes_Q Y$) $\wedge$ ($\forall \alpha \in$ *qbs-Mx* ($X \bigotimes_Q Y$). $\alpha -` (Ua \times Ub) \in$ *sets borel*)
using *hu* **by**(*auto simp add*: *vimage-Times pair-qbs-space pair-qbs-Mx*)
thus *?thesis*
by(*auto simp add*: *sigma-Mx-def sets-L intro*!: *exI*[**where** x=$Ua \times Ub$])
qed
qed

corollary *qbs-borel-prod*: *qbs-borel* $\bigotimes_Q$ *qbs-borel* = (*qbs-borel* :: ('*a*::*second-countable-topology* × '*b*::*second-countable-topology*) *quasi-borel*)
by(*simp add*: *r-preserves-product*[*symmetric*] *borel-prod*)

corollary *qbs-count-space-prod*: *qbs-count-space* (*UNIV* :: ('*a* :: *countable*) *set*) $\bigotimes_Q$ *qbs-count-space* (*UNIV* :: ('*b* :: *countable*) *set*) = *qbs-count-space UNIV*
by(*auto simp*: *r-preserves-product*[*symmetric*] *count-space-prod*)

lemma *r-preserves-product'*: *measure-to-qbs* (Π_M $i \in I$. M i) = (Π_Q $i \in I$. *measure-to-qbs* (M i))
proof(*rule qbs-eqI*)
show *qbs-Mx* (*measure-to-qbs* (Pi_M I M)) = *qbs-Mx* (Π_Q $i \in I$. *measure-to-qbs* (M i))
proof *safe*
fix f :: *real* ⇒ -
assume $f \in$ *qbs-Mx* (*measure-to-qbs* (Pi_M I M))
with *measurable-space*[*of f borel* Pi_M I M] **show** $f \in$ *qbs-Mx* (Π_Q $i \in I$. *mea-*

sure-to-qbs (*M i*))
by(*auto simp*: *qbs-Mx-R PiQ-Mx space-PiM intro*!:*ext*[*of* λ*r*. *f r* -])
next
fix *f* :: *real* ⇒ -
assume *f* ∈ *qbs-Mx* (Π$_Q$ *i*∈*I*. *measure-to-qbs* (*M i*))
then have ⋀*i*. *i* ∈ *I* ⟹ (λ*r*. *f r i*) ∈ *borel* →$_M$ *M i* ⋀*i*. *i* ∉ *I* ⟹ (λ*r*. *f r i*)
= (λ*r*. *undefined*)
by (*auto simp*: *qbs-Mx-R PiQ-Mx*)
with *measurable-space*[*OF this*(*1*)] *fun-cong*[*OF this*(*2*)] **show** *f* ∈ *qbs-Mx*
(*measure-to-qbs* (*Pi*$_M$ *I M*))
by(*auto intro*!: *measurable-PiM-single*′ *simp*: *qbs-Mx-R*)
qed
qed

lemma *PiQ-qbs-borel*:
(Π$_Q$ *i*::(′*a*:: *countable*)∈*UNIV*. (*qbs-borel* :: (′*b*::*second-countable-topology quasi-borel*)))
= *qbs-borel*
by(*simp add*: *r-preserves-product*′[*symmetric*] *measure-to-qbs-cong-sets*[*OF sets-PiM-equal-borel*])

lemma *qbs-morphism-from-countable*:
fixes *X* :: ′*a quasi-borel*
assumes *countable* (*qbs-space X*)
qbs-Mx X ⊆ *borel* →$_M$ *count-space* (*qbs-space X*)
and ⋀*i*. *i* ∈ *qbs-space X* ⟹ *f i* ∈ *qbs-space Y*
shows *f* ∈ *X* →$_Q$ *Y*
proof(*rule qbs-morphismI*)
fix α
assume α ∈ *qbs-Mx X*
then have [*measurable*]: α ∈ *borel* →$_M$ *count-space* (*qbs-space X*)
using *assms*(*2*) **..**
define *k* :: ′*a* ⇒ *real* ⇒ -
where *k* ≡ (λ*i* -. *f i*)
have *f* ∘ α = (λ*r*. *k* (α *r*) *r*)
by(*auto simp add*: *k-def*)
also have ... ∈ *qbs-Mx Y*
by(*rule qbs-closed3-dest2*[*OF assms*(*1*)]) (*use assms*(*3*) *k-def* **in** *simp-all*)
finally show *f* ∘ α ∈ *qbs-Mx Y* **.**
qed

corollary *qbs-morphism-count-space*′:
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *qbs-space Y countable I*
shows *f* ∈ *qbs-count-space I* →$_Q$ *Y*
using *assms* **by**(*auto intro*!: *qbs-morphism-from-countable simp*: *qbs-Mx-R*)

corollary *qbs-morphism-count-space*:
assumes ⋀*i*. *f i* ∈ *qbs-space Y*
shows *f* ∈ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) →$_Q$ *Y*
using *assms* **by**(*auto intro*!: *qbs-morphism-from-countable simp*: *qbs-Mx-R*)

lemma [*qbs*]:
shows *not-qbs-pred*: $Not \in$ *qbs-count-space UNIV* $\rightarrow_Q$ *qbs-count-space UNIV*
and *or-qbs-pred*: $(\vee) \in$ *qbs-count-space UNIV* $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *and-qbs-pred*: $(\wedge) \in$ *qbs-count-space UNIV* $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *implies-qbs-pred*: $(\longrightarrow) \in$ *qbs-count-space UNIV* $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *iff-qbs-pred*: $(\longleftrightarrow) \in$ *qbs-count-space UNIV* $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
by(*auto intro*!: *qbs-morphism-count-space*)

lemma [*qbs*]:
shows *less-count-qbs-pred*: $(<) \in$ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *le-count-qbs-pred*: $(\leq) \in$ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *eq-count-qbs-pred*: $(=) \in$ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *plus-count-qbs-morphism*: $(+) \in$ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *minus-count-qbs-morphism*: $(-) \in$ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *mult-count-qbs-morphism*: $(*) \in$ *qbs-count-space* (*UNIV* :: (- :: *countable*) *set*) $\rightarrow_Q$ *exp-qbs* (*qbs-count-space UNIV*) (*qbs-count-space UNIV*)
and *Suc-qbs-morphism*: *Suc* $\in$ *qbs-count-space UNIV* $\rightarrow_Q$ *qbs-count-space UNIV*
by(*auto intro*!: *qbs-morphism-count-space*)

lemma *qbs-morphism-product-iff*:
$f \in X \rightarrow_Q (\Pi_Q\ i :: (- :: countable) \in UNIV.\ Y) \longleftrightarrow f \in X \rightarrow_Q$ *qbs-count-space UNIV* $\Rightarrow_Q Y$
proof
assume *h*:$f \in X \rightarrow_Q (\Pi_Q\ i \in UNIV.\ Y)$
show $f \in X \rightarrow_Q$ *qbs-count-space UNIV* $\Rightarrow_Q Y$
by(*rule arg-swap-morphism*, *rule qbs-morphism-count-space*) (*simp add*: *qbs-morphism-component-singleton h qbs-morphism-ident'*])
next
assume $f \in X \rightarrow_Q$ *qbs-count-space UNIV* $\Rightarrow_Q Y$
from *qbs-morphism-space*[*OF arg-swap-morphism*[*OF this*]]
show $f \in X \rightarrow_Q (\Pi_Q\ i \in UNIV.\ Y)$
by(*auto intro*!: *product-qbs-canonical1*[**where** $f=(\lambda i\ x.\ f\ x\ i)$])
qed

lemma *qbs-morphism-pair-countable1*:
assumes *countable* (*qbs-space X*)
qbs-Mx X $\subseteq$ *borel* $\rightarrow_M$ *count-space* (*qbs-space X*)
and $\bigwedge i.\ i \in$ *qbs-space X* $\Longrightarrow f\ i \in Y \rightarrow_Q Z$
shows $(\lambda(x,y).\ f\ x\ y) \in X \bigotimes_Q Y \rightarrow_Q Z$
by(*auto intro*!: *uncurry-preserves-morphisms qbs-morphism-from-countable*[*OF*

assms(*1*,*2*)] *assms*(*3*))

lemma *qbs-morphism-pair-countable2*:
assumes *countable* (*qbs-space Y*)
qbs-Mx Y ⊆ *borel* →$_M$ *count-space* (*qbs-space Y*)
and ⋀*i*. *i* ∈ *qbs-space Y* ⟹ (λ*x*. *f x i*) ∈ *X* →$_Q$ *Z*
shows (λ(*x*,*y*). *f x y*) ∈ *X* ⨂$_Q$ *Y* →$_Q$ *Z*
by(*auto intro*!: *qbs-morphism-pair-swap*[*of case-prod* (λ*x y*. *f y x*),*simplified*] *qbs-morphism-pair-countable1 assms*)

corollary *qbs-morphism-pair-count-space1*:
assumes ⋀*i*. *f i* ∈ *Y* →$_Q$ *Z*
shows (λ(*x*,*y*). *f x y*) ∈ *qbs-count-space* (*UNIV* :: ('*a* :: *countable*) *set*) ⨂$_Q$ *Y* →$_Q$ *Z*
by(*auto intro*!: *qbs-morphism-pair-countable1 simp*: *qbs-Mx-R assms*)

corollary *qbs-morphism-pair-count-space2*:
assumes ⋀*i*. (λ*x*. *f x i*) ∈ *X* →$_Q$ *Z*
shows (λ(*x*,*y*). *f x y*) ∈ *X* ⨂$_Q$ *qbs-count-space* (*UNIV* :: ('*a* :: *countable*) *set*) →$_Q$ *Z*
by(*auto intro*!: *qbs-morphism-pair-countable2 simp*: *qbs-Mx-R assms*)

lemma *qbs-morphism-compose-countable'*:
assumes [*qbs*]:⋀*i*. *i* ∈ *I* ⟹ (λ*x*. *f i x*) ∈ *X* →$_Q$ *Y g* ∈ *X* →$_Q$ *qbs-count-space I countable I*
shows (λ*x*. *f* (*g x*) *x*) ∈ *X* →$_Q$ *Y*
proof −
have [*qbs*]:*f* ∈ *qbs-count-space I* →$_Q$ *X* ⇒$_Q$ *Y*
by(*auto intro*!: *qbs-morphism-count-space'* *simp*: *assms*(*3*))
show *?thesis*
by *simp*
qed

lemma *qbs-morphism-compose-countable*:
assumes [*simp*]:⋀*i*::'*i*::*countable*. (λ*x*. *f i x*) ∈ *X* →$_Q$ *Y g* ∈ *X* →$_Q$ (*qbs-count-space UNIV*)
shows (λ*x*. *f* (*g x*) *x*) ∈ *X* →$_Q$ *Y*
by(*rule qbs-morphism-compose-countable'*[*of UNIV f*]) *simp-all*

lemma *qbs-morphism-op*:
assumes *case-prod f* ∈ *X* ⨂$_M$ *Y* →$_M$ *Z*
shows *f* ∈ *measure-to-qbs X* →$_Q$ *measure-to-qbs Y* ⇒$_Q$ *measure-to-qbs Z*
using *r-preserves-morphisms assms*
by(*fastforce simp*: *r-preserves-product*[*symmetric*] *intro*!: *curry-preserves-morphisms*)

lemma [*qbs*]:
shows *plus-qbs-morphism*: (+) ∈ (*qbs-borel* :: (-::{*second-countable-topology*, *topological-monoid-add*}) *quasi-borel*) →$_Q$ *qbs-borel* ⇒$_Q$ *qbs-borel*
and *plus-ereal-qbs-morphism*: (+) ∈ (*qbs-borel* :: *ereal quasi-borel*) →$_Q$ *qbs-borel*

$\Rightarrow_Q$ *qbs-borel*
and *diff-qbs-morphism*: $(-) \in$ (*qbs-borel* :: (-::{*second-countable-topology*, *real-normed-vector*}) *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *diff-ennreal-qbs-morphism*: $(-) \in$ (*qbs-borel* :: *ennreal quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *diff-ereal-qbs-morphism*: $(-) \in$ (*qbs-borel* :: *ereal quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *times-qbs-morphism*: $(*) \in$ (*qbs-borel* :: (-::{*second-countable-topology*, *real-normed-algebra*}) *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *times-ennreal-qbs-morphism*: $(*) \in$ (*qbs-borel* :: *ennreal quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *times-ereal-qbs-morphism*: $(*) \in$ (*qbs-borel* :: *ereal quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *divide-qbs-morphism*: $(/) \in$ (*qbs-borel* :: (-::{*second-countable-topology*, *real-normed-div-algebra*}) *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *divide-ennreal-qbs-morphism*: $(/) \in$ (*qbs-borel* :: *ennreal quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *divide-ereal-qbs-morphism*: $(/) \in$ (*qbs-borel* :: *ereal quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *log-qbs-morphism*: *log* $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *root-qbs-morphism*: *root* $\in$ *qbs-count-space UNIV* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *scaleR-qbs-morphism*: $(*_R) \in$ *qbs-borel* $\rightarrow_Q$ (*qbs-borel* :: (-::{*second-countable-topology*, *real-normed-vector*}) *quasi-borel*) $\Rightarrow_Q$ *qbs-borel*
and *qbs-morphism-inner*: $(\cdot) \in$ *qbs-borel* $\rightarrow_Q$ (*qbs-borel* :: (-::{*second-countable-topology*, *real-inner*}) *quasi-borel*) $\Rightarrow_Q$ *qbs-borel*
and *dist-qbs-morphism*: *dist* $\in$ (*qbs-borel* :: (-::{*second-countable-topology*, *metric-space*}) *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *powr-qbs-morphism*: (*powr*) $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ (*qbs-borel* :: *real quasi-borel*)
and *max-qbs-morphism*: (*max* :: (- :: {*second-countable-topology*, *linorder-topology*}) $\Rightarrow$ - $\Rightarrow$ -) $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *min-qbs-morphism*: (*min* :: (- :: {*second-countable-topology*, *linorder-topology*}) $\Rightarrow$ - $\Rightarrow$ -) $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *sup-qbs-morphism*: (*sup* :: (- :: {*lattice*,*second-countable-topology*, *linorder-topology*}) $\Rightarrow$ - $\Rightarrow$ -) $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *inf-qbs-morphism*: (*inf* :: (- :: {*lattice*,*second-countable-topology*, *linorder-topology*}) $\Rightarrow$ - $\Rightarrow$ -) $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
and *less-qbs-pred*: $(<) \in$ (*qbs-borel* :: - ::{*second-countable-topology*, *linorder-topology*} *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-count-space UNIV*
and *eq-qbs-pred*: $(=) \in$ (*qbs-borel* :: - ::{*second-countable-topology*, *linorder-topology*} *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-count-space UNIV*
and *le-qbs-pred*: $(\leq) \in$ (*qbs-borel* :: - ::{*second-countable-topology*, *linorder-topology*} *quasi-borel*) $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-count-space UNIV*
by(*auto intro*!: *qbs-morphism-op*)

lemma [*qbs*]:
shows *abs-real-qbs-morphism*: *abs* $\in$ (*qbs-borel* :: *real quasi-borel*) $\rightarrow_Q$ *qbs-borel*
and *abs-ereal-qbs-morphism*: *abs* $\in$ (*qbs-borel* :: *ereal quasi-borel*) $\rightarrow_Q$ *qbs-borel*
and *real-floor-qbs-morphism*: (*floor* :: *real* $\Rightarrow$ *int*) $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-count-space*

UNIV
and *real-ceiling-qbs-morphism*: (*ceiling* :: *real* ⇒ *int*) ∈ *qbs-borel* →$_Q$ *qbs-count-space*
UNIV
and *exp-qbs-morphism*: (*exp*::'*a*::{*real-normed-field*,*banach*}⇒'*a*) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *ln-qbs-morphism*: *ln* ∈ (*qbs-borel* :: *real quasi-borel*) →$_Q$ *qbs-borel*
and *sqrt-qbs-morphism*: *sqrt* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *of-real-qbs-morphism*: (*of-real* :: - ⇒ (-::*real-normed-algebra*)) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *sin-qbs-morphism*: (*sin* :: - ⇒ (-::{*real-normed-field*,*banach*})) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *cos-qbs-morphism*: (*cos* :: - ⇒ (-::{*real-normed-field*,*banach*})) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *arctan-qbs-morphism*: *arctan* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *Re-qbs-morphism*: *Re* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *Im-qbs-morphism*: *Im* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *sgn-qbs-morphism*: (*sgn*::-::*real-normed-vector* ⇒ -) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *norm-qbs-morphism*: *norm* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *invers-qbs-morphism*: (*inverse* :: - ⇒ (- ::*real-normed-div-algebra*)) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *invers-ennreal-qbs-morphism*: (*inverse* :: - ⇒ *ennreal*) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *invers-ereal-qbs-morphism*: (*inverse* :: - ⇒ *ereal*) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *uminus-qbs-morphism*: (*uminus* :: - ⇒ (-::{*second-countable-topology*, *real-normed-vector*})) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *ereal-qbs-morphism*: *ereal* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *real-of-ereal-qbs-morphism*: *real-of-ereal* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *enn2ereal-qbs-morphism*: *enn2ereal* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *e2ennreal-qbs-morphism*: *e2ennreal* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *ennreal-qbs-morphism*: *ennreal* ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *qbs-morphism-nth*: (λ*x*::*real*^'*n*. *x* $ *i*) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *qbs-morphism-product-candidate*: ⋀*i*. (λ*x*. *x i*) ∈ *qbs-borel* →$_Q$ *qbs-borel*
and *uminus-ereal-qbs-morphism*: (*uminus* :: - ⇒ *ereal*) ∈ *qbs-borel* →$_Q$ *qbs-borel*
by(*auto intro*!: *set-mp*[*OF r-preserves-morphisms*])

lemma *qbs-morphism-sum*:
fixes *f* :: '*c* ⇒ '*a* ⇒ '*b*::{*second-countable-topology*, *topological-comm-monoid-add*}
assumes ⋀*i*. *i* ∈ *S* ⟹ *f i* ∈ *X* →$_Q$ *qbs-borel*
shows (λ*x*. ∑ *i*∈*S*. *f i x*) ∈ *X* →$_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-suminf-order*:
fixes *f* :: *nat* ⇒ '*a* ⇒ '*b*::{*complete-linorder*, *second-countable-topology*, *linorder-topology*, *topological-comm-monoid-add*}
assumes ⋀*i*. *f i* ∈ *X* →$_Q$ *qbs-borel*
shows (λ*x*. ∑ *i*. *f i x*) ∈ *X* →$_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-prod*:

fixes $f :: 'c \Rightarrow 'a \Rightarrow 'b$::{*second-countable-topology*, *real-normed-field*}
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-Min*:
finite $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i \in X \to_Q$ *qbs-borel*$) \Longrightarrow (\lambda x.\ Min\ ((\lambda i.\ f\ i\ x)\ `I) ::$
$'b$::{*second-countable-topology*, *linorder-topology*}$) \in X \to_Q$ *qbs-borel*
by(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-Max*:
finite $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i \in X \to_Q$ *qbs-borel*$) \Longrightarrow (\lambda x.\ Max\ ((\lambda i.\ f\ i\ x)\ `I)$
$::\ 'b$::{*second-countable-topology*, *linorder-topology*}$) \in X \to_Q$ *qbs-borel*
by(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-Max2*:
fixes f::$- \Rightarrow - \Rightarrow 'a$::{*second-countable-topology*, *dense-linorder*, *linorder-topology*}
shows *finite* $I \Longrightarrow (\bigwedge i.\ f\ i \in X \to_Q$ *qbs-borel*$) \Longrightarrow (\lambda x.\ Max\{f\ i\ x\ |i.\ i \in I\}) \in$
$X \to_Q$ *qbs-borel*
by(*simp add*: *lr-adjunction-correspondence*)

lemma [*qbs*]:
shows *qbs-morphism-liminf*: *liminf* $\in$ (*qbs-count-space UNIV* $\Rightarrow_Q$ *qbs-borel*$) \Rightarrow_Q$
(*qbs-borel* :: $'a$ :: {*complete-linorder*, *second-countable-topology*, *linorder-topology*}
quasi-borel)
and *qbs-morphism-limsup*: *limsup* $\in$ (*qbs-count-space UNIV* $\Rightarrow_Q$ *qbs-borel*$) \Rightarrow_Q$
(*qbs-borel* :: $'a$ :: {*complete-linorder*, *second-countable-topology*, *linorder-topology*}
quasi-borel)
and *qbs-morphism-lim*: *lim* $\in$ (*qbs-count-space UNIV* $\Rightarrow_Q$ *qbs-borel*$) \Rightarrow_Q$ (*qbs-borel*
:: $'a$ :: {*complete-linorder*, *second-countable-topology*, *linorder-topology*} *quasi-borel*)
proof(*safe intro*!: *qbs-morphismI*)
fix f :: *real* $\Rightarrow$ *nat* $\Rightarrow 'a$
assume $f \in$ *qbs-Mx* (*count-space*$_Q$ *UNIV* $\Rightarrow_Q$ *borel*$_Q$)
then have [*measurable*]:$\bigwedge i.\ (\lambda r.\ f\ r\ i) \in$ *borel-measurable borel*
by(*auto simp*: *qbs-Mx-is-morphisms*) (*metis PiQ-qbs-borel measurable-product-then-coordinatewise*
qbs-Mx-is-morphisms qbs-Mx-qbs-borel qbs-morphism-product-iff)
show *liminf* $\circ f \in$ *qbs-Mx borel*$_Q$ *limsup* $\circ f \in$ *qbs-Mx borel*$_Q$ *lim* $\circ f \in$ *qbs-Mx*
borel$_Q$
by(*auto simp*: *qbs-Mx-is-morphisms lr-adjunction-correspondence comp-def*)
qed

lemma *qbs-morphism-SUP*:
fixes F :: $- \Rightarrow - \Rightarrow -$::{*complete-linorder*, *linorder-topology*, *second-countable-topology*}
assumes *countable* I $\bigwedge i.\ i \in I \Longrightarrow F\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ \bigsqcup\ i{\in}I.\ F\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-INF*:
fixes F :: $- \Rightarrow - \Rightarrow -$::{*complete-linorder*, *linorder-topology*, *second-countable-topology*}

assumes *countable I* $\bigwedge i.\ i \in I \Longrightarrow F\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ \prod_{i \in I}\ F\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-cSUP*:
fixes $F :: - \Rightarrow - \Rightarrow 'a::\{$*conditionally-complete-linorder*, *linorder-topology*, *second-countable-topology*$\}$
assumes *countable I* $\bigwedge i.\ i \in I \Longrightarrow F\ i \in X \to_Q$ *qbs-borel* $\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *bdd-above* $((\lambda i.\ F\ i\ x)\ `\ I)$
shows $(\lambda x.\ \bigsqcup_{i \in I}\ F\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence space-L*)

lemma *qbs-morphism-cINF*:
fixes $F :: - \Rightarrow - \Rightarrow 'a::\{$*conditionally-complete-linorder*, *linorder-topology*, *second-countable-topology*$\}$
assumes *countable I* $\bigwedge i.\ i \in I \Longrightarrow F\ i \in X \to_Q$ *qbs-borel* $\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *bdd-below* $((\lambda i.\ F\ i\ x)\ `\ I)$
shows $(\lambda x.\ \prod_{i \in I}\ F\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence space-L*)

lemma *qbs-morphism-lim-metric*:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b::\{$*banach*, *second-countable-topology*$\}$
assumes $\bigwedge i.\ f\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ lim\ (\lambda i.\ f\ i\ x)) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-LIMSEQ-metric*:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b ::$ *metric-space*
assumes $\bigwedge i.\ f\ i \in X \to_Q$ *qbs-borel* $\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow (\lambda i.\ f\ i\ x) \longrightarrow g\ x$
shows $g \in X \to_Q$ *qbs-borel*
using *borel-measurable-LIMSEQ-metric*[**where** *M=qbs-to-measure X*] *assms*
by(*auto simp add*: *lr-adjunction-correspondence space-L*)

lemma *power-qbs-morphism*[*qbs*]:
$(power :: (- ::\{power, real\text{-}normed\text{-}algebra\}) \Rightarrow nat \Rightarrow -) \in$ *qbs-borel* $\to_Q$ *qbs-count-space UNIV* $\Rightarrow_Q$ *qbs-borel*
by(*rule arg-swap-morphism*) (*auto intro*!: *qbs-morphism-count-space set-mp*[*OF r-preserves-morphisms*])

lemma *power-ennreal-qbs-morphism*[*qbs*]:
$(power :: ennreal \Rightarrow nat \Rightarrow -) \in$ *qbs-borel* $\to_Q$ *qbs-count-space UNIV* $\Rightarrow_Q$ *qbs-borel*
by(*rule arg-swap-morphism*) (*auto intro*!: *qbs-morphism-count-space set-mp*[*OF r-preserves-morphisms*])

lemma *qbs-morphism-compw*: $(\frown) \in (X \Rightarrow_Q X) \to_Q$ *qbs-count-space UNIV* $\Rightarrow_Q (X \Rightarrow_Q X)$
proof(*rule arg-swap-morphism*,*rule qbs-morphism-count-space*)
fix *n*

show $(\lambda y.\ y\ \widehat{\ }\widehat{\ }\ n) \in X \Rightarrow_Q X \to_Q X \Rightarrow_Q X$
by(*induction n*) *simp-all*
qed

lemma *qbs-morphism-compose-n*[*qbs*]:
assumes [*qbs*]: $f \in X \to_Q X$
shows $(\lambda n.\ f\widehat{\ }\widehat{\ }n) \in$ *qbs-count-space UNIV* $\to_Q X \Rightarrow_Q X$
proof(*intro qbs-morphism-count-space*)
fix *n*
show $f\ \widehat{\ }\widehat{\ }\ n \in X \to_Q X$
by (*induction n*) *simp-all*
qed

lemma *qbs-morphism-compose-n'*:
assumes $f \in X \to_Q X$
shows $f\widehat{\ }\widehat{\ }n \in X \to_Q X$
using *qbs-morphism-space*[*OF qbs-morphism-compose-n*[*OF assms*]] **by**(*simp add*: *exp-qbs-space qbs-space-R*)

lemma *qbs-morphism-uminus-eq-ereal*[*simp*]:
$(\lambda x.\ -\ f\ x :: ereal) \in X \to_Q$ *qbs-borel* $\longleftrightarrow f \in X \to_Q$ *qbs-borel* (**is** *?l* = *?r*)
by(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-ereal-iff*:
shows $(\lambda x.\ ereal\ (f\ x)) \in X \to_Q$ *qbs-borel*$\longleftrightarrow f \in X \to_Q$ *qbs-borel*
by(*simp add*: *borel-measurable-ereal-iff lr-adjunction-correspondence*)

lemma *qbs-morphism-ereal-sum*:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ereal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ \sum i{\in}S.\ f\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-ereal-prod*:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ereal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-extreal-suminf*:
fixes $f :: nat \Rightarrow 'a \Rightarrow ereal$
assumes $\bigwedge i.\ f\ i \in X \to_Q$ *qbs-borel*
shows $(\lambda x.\ (\sum i.\ f\ i\ x)) \in X \to_Q$ *qbs-borel*
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-ennreal-iff*:
assumes $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow 0 \le f\ x$
shows $(\lambda x.\ ennreal\ (f\ x)) \in X \to_Q$ *qbs-borel* $\longleftrightarrow f \in X \to_Q$ *qbs-borel*
using *borel-measurable-ennreal-iff*[**where** *M*=*qbs-to-measure X*] *assms*

by(*simp add*: *space-L lr-adjunction-correspondence*)

lemma *qbs-morphism-prod-ennreal*:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ennreal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in X \to_Q qbs\text{-}borel$
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in X \to_Q qbs\text{-}borel$
using *assms* **by**(*simp add*: *space-L lr-adjunction-correspondence*)

lemma *count-space-qbs-morphism*:
$f \in qbs\text{-}count\text{-}space\ (UNIV :: 'a\ set) \to_Q qbs\text{-}borel$
by(*auto intro*!: *set-mp*[*OF r-preserves-morphisms*])

declare *count-space-qbs-morphism*[**where** *'a*=- :: *countable*,*qbs*]

lemma *count-space-count-space-qbs-morphism*:
$f \in qbs\text{-}count\text{-}space\ (UNIV :: (- :: countable)\ set) \to_Q qbs\text{-}count\text{-}space\ (UNIV :: (- :: countable)\ set)$
by(*auto intro*!: *set-mp*[*OF r-preserves-morphisms*])

lemma *qbs-morphism-case-nat'*:
assumes [*qbs*]: $i = 0 \Longrightarrow f \in X \to_Q Y$
$\bigwedge j.\ i = Suc\ j \Longrightarrow (\lambda x.\ g\ x\ j) \in X \to_Q Y$
shows $(\lambda x.\ case\text{-}nat\ (f\ x)\ (g\ x)\ i) \in X \to_Q Y$
by (*cases i*) *simp-all*

lemma *qbs-morphism-case-nat*[*qbs*]:
$case\text{-}nat \in X \to_Q (qbs\text{-}count\text{-}space\ UNIV \Rightarrow_Q X) \Rightarrow_Q qbs\text{-}count\text{-}space\ UNIV \Rightarrow_Q X$
by(*rule curry-preserves-morphisms*, *rule arg-swap-morphism*) (*auto intro*!: *qbs-morphism-count-space qbs-morphism-case-nat'*)

lemma *qbs-morphism-case-nat''*:
assumes $f \in X \to_Q Y\ g \in X \to_Q (\Pi_Q\ i{\in}UNIV.\ Y)$
shows $(\lambda x.\ case\text{-}nat\ (f\ x)\ (g\ x)) \in X \to_Q (\Pi_Q\ i{\in}UNIV.\ Y)$
using *assms* **by** (*simp add*: *qbs-morphism-product-iff*)

lemma *qbs-morphism-rec-nat*[*qbs*]: $rec\text{-}nat \in X \to_Q (count\text{-}space\ UNIV \Rightarrow_Q X \Rightarrow_Q X) \Rightarrow_Q count\text{-}space\ UNIV \Rightarrow_Q X$
proof(*rule curry-preserves-morphisms*,*rule arg-swap-morphism*,*rule qbs-morphism-count-space*)
fix *n*
show $(\lambda y.\ rec\text{-}nat\ (fst\ y)\ (snd\ y)\ n) \in X \bigotimes_Q (qbs\text{-}count\text{-}space\ UNIV \Rightarrow_Q X \Rightarrow_Q X) \to_Q X$
by (*induction n*) *simp-all*
qed

lemma *qbs-morphism-Max-nat*:
fixes $P :: nat \Rightarrow 'a \Rightarrow bool$
assumes $\bigwedge i.\ P\ i \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$

shows $(\lambda x.\ Max\ \{i.\ P\ i\ x\}) \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-Min-nat*:
fixes $P :: nat \Rightarrow 'a \Rightarrow bool$
assumes $\bigwedge i.\ P\ i \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$
shows $(\lambda x.\ Min\ \{i.\ P\ i\ x\}) \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-sum-nat*:
fixes $f :: 'c \Rightarrow 'a \Rightarrow nat$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$
shows $(\lambda x.\ \sum i{\in}S.\ f\ i\ x) \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$
using *assms* **by**(*simp add*: *lr-adjunction-correspondence*)

lemma *qbs-morphism-case-enat'*:
assumes *f*[*qbs*]: $f \in X \to_Q qbs\text{-}count\text{-}space\ UNIV$ **and** [*qbs*]: $\bigwedge i.\ g\ i \in X \to_Q Y\ h \in X \to_Q Y$
shows $(\lambda x.\ case\ f\ x\ of\ enat\ i \Rightarrow g\ i\ x \mid \infty \Rightarrow h\ x) \in X \to_Q Y$
proof (*rule qbs-morphism-compose-countable*[*OF - f*])
fix *i*
show $(\lambda x.\ case\ i\ of\ enat\ i \Rightarrow g\ i\ x \mid \infty \Rightarrow h\ x) \in X \to_Q Y$
by (*cases i*) *simp-all*
qed

lemma *qbs-morphism-case-enat*[*qbs*]: *case-enat* $\in$ *qbs-space* ((*qbs-count-space UNIV* $\Rightarrow_Q$ *X*) $\Rightarrow_Q$ *X* $\Rightarrow_Q$ *qbs-count-space UNIV* $\Rightarrow_Q$ *X*)
proof −
note *qbs-morphism-case-enat'*[*qbs*]
show *?thesis*
by(*auto intro*!: *curry-preserves-morphisms*,*rule qbs-morphismI*) (*simp add*: *qbs-Mx-is-morphisms comp-def*, *qbs*, *simp-all*)
qed

lemma *qbs-morphism-restrict*[*qbs*]:
assumes *X*: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in X \to_Q (Y\ i)$
shows $(\lambda x.\ \lambda i{\in}I.\ f\ i\ x) \in X \to_Q (\Pi_Q\ i{\in}I.\ Y\ i)$
using *assms* **by**(*auto intro*!: *product-qbs-canonical1*)

lemma *If-qbs-morphism*[*qbs*]: *If* $\in$ *qbs-count-space UNIV* $\to_Q$ *X* $\Rightarrow_Q$ *X* $\Rightarrow_Q$ *X*
proof(*rule qbs-morphismI*)
show $\alpha \in qbs\text{-}Mx\ (count\text{-}space_Q\ UNIV) \Longrightarrow If \circ \alpha \in qbs\text{-}Mx\ (X \Rightarrow_Q X \Rightarrow_Q X)$ **for** α
by(*auto intro*!: *qbs-Mx-indicat*[**where** $S{=}\{r.\ \alpha\ (-\ (-\ r))\}$,*simplified*] *simp*: *qbs-Mx-count-space exp-qbs-Mx*)
qed

lemma *normal-density-qbs*[*qbs*]: *normal-density* $\in$ *qbs-borel* $\to_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*

$\Rightarrow_Q$ *qbs-borel*
proof −
have [*simp*]:*normal-density* = ($\lambda\mu$ σ *x. 1 / sqrt* (*2* ∗ *pi* ∗ σ^2) ∗ *exp* (−(*x* − μ)^2/ (*2* ∗ σ^2)))
by *standard*+ (*auto simp*: *normal-density-def*)
show *?thesis*
by *simp*
qed

lemma *erlang-density-qbs*[*qbs*]: *erlang-density* ∈ *qbs-count-space UNIV* $\rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel* $\Rightarrow_Q$ *qbs-borel*
proof −
have [*simp*]: *erlang-density* = (λ*k l x.* (*if x* < *0 then 0 else* (*l*^(*Suc k*) ∗ *x*^*k* ∗ *exp* (− *l* ∗ *x*)) / *fact k*))
by *standard*+ (*auto simp*: *erlang-density-def*)
show *?thesis*
by *simp*
qed

lemma *list-nil-qbs*[*qbs*]: [] ∈ *qbs-space* (*list-qbs X*)
by(*simp add*: *list-qbs-space*)

lemma *list-cons-qbs-morphism*: *list-cons* ∈ *X* $\rightarrow_Q$ (Π_Q *n*∈(*UNIV* :: *nat set*).Π_Q *i*∈{..<*n*}. *X*) $\Rightarrow_Q$ (Π_Q *n*∈(*UNIV* :: *nat set*).Π_Q *i*∈{..<*n*}. *X*)
proof(*intro curry-preserves-morphisms pair-qbs-morphismI*)
fix α β
assume *h*:α ∈ *qbs-Mx X*
β ∈ *qbs-Mx* (Π_Q *n*∈(*UNIV* :: *nat set*).Π_Q *i*∈{..<*n*}. *X*)
then obtain γ *f* **where** *hf*:
β = (λ*r.* (*f r*, γ (*f r*) *r*)) *f* ∈ *borel* $\rightarrow_M$ *count-space UNIV* $\bigwedge$*i. i* ∈ *range f* $\Longrightarrow$ γ *i* ∈ *qbs-Mx* (Π_Q *j*∈{..<*i*}. *X*)
by(*auto simp*: *coPiQ-Mx-def coPiQ-Mx*)
define *f*′ β′
where *f*′ ≡ (λ*r. Suc* (*f r*)) β′ ≡ (λ*i r n. if n* = *0 then* α *r else* γ (*i* − *1*) *r* (*n* − *1*))
then have (λ*r. list-cons* (*fst* (α *r*, β *r*)) (*snd* (α *r*, β *r*))) = (λ*r.* (*f*′ *r*, β′ (*f*′ *r*) *r*))
by(*auto simp*: *comp-def hf*(*1*) *ext list-cons-def*)
also have ... ∈ *qbs-Mx* (Π_Q *n*∈(*UNIV* :: *nat set*).Π_Q *i*∈{..<*n*}. *X*)
proof(*rule coPiQ-MxI*)
show *f*′ ∈ *borel* $\rightarrow_M$ *count-space UNIV*
using *hf* **by**(*simp add*: *f*′-β′-*def*(*1*))
next
fix *j*
assume *hj*:*j* ∈ *range f*′
then have *hj*′:*j* − *1* ∈ *range f*
by(*auto simp*: *f*′-β′-*def*(*1*))
show β′ *j* ∈ *qbs-Mx* (Π_Q *i*∈{..<*j*}. *X*)
proof(*rule prod-qbs-MxI*)

```
    fix i
    assume hi:i ∈ {..<j}
    then consider i = 0 | 0 < i i < j
      by auto
    then show (λr. β' j r i) ∈ qbs-Mx X
    proof cases
      case 1
      then show ?thesis by(simp add: h(1) f'-β'-def(2))
    next
      case 2
      then have i − 1 ∈ {..<j − 1} by simp
      from prod-qbs-MxD(1)[OF hf(3)[OF hj'] this] 2
      show ?thesis
        by(simp add: f'-β'-def(2))
    qed
  next
    fix i
    assume hi:i ∉ {..<j}
    then have i ≠ 0 i − Suc 0 ∉ {..<j − Suc 0}
      using f'-β'-def(1) hj by fastforce+
    with prod-qbs-MxD(2)[OF hf(3)[OF hj']]
    show (λr. β' j r i) = (λr. undefined)
      by(simp add: f'-β'-def(2))
  qed
 qed
 finally show (λr. list-cons (fst (α r, β r)) (snd (α r, β r))) ∈ qbs-Mx (Π_Q
n∈(UNIV :: nat set).Π_Q i∈{..<n}. X) .
qed

corollary cons-qbs-morphism[qbs]: Cons ∈ X →_Q (list-qbs X) ⇒_Q list-qbs X
proof(rule arg-swap-morphism)
 show (λx y. y # x) ∈ list-qbs X →_Q X ⇒_Q list-qbs X
 proof(rule qbs-morphism-cong')
  show (λl x. x # to-list l) ∘ from-list ∈ list-qbs X →_Q X ⇒_Q list-qbs X
  proof(rule qbs-morphism-comp)
   show (λl x. x # to-list l) ∈ (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X) →_Q
X ⇒_Q list-qbs X
    proof(rule curry-preserves-morphisms)
     show (λlx. snd lx # to-list (fst lx)) ∈ (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}.
X) ⨂_Q X →_Q list-qbs X
      proof(rule qbs-morphism-cong')
       show to-list ∘ (λ(l, x). from-list (x # to-list l)) ∈ (Π_Q n∈(UNIV :: nat
set).Π_Q i∈{..<n}. X) ⨂_Q X →_Q list-qbs X
        proof(rule qbs-morphism-comp)
         show (λ(l, x). from-list (x # to-list l)) ∈(Π_Q n∈(UNIV :: nat set).Π_Q
i∈{..<n}. X) ⨂_Q X →_Q (Π_Q n∈(UNIV :: nat set).Π_Q i∈{..<n}. X)
          by(rule qbs-morphism-cong'[OF - uncurry-preserves-morphisms[of λ(l,x).
list-cons x l,simplified,OF arg-swap-morphism[OF list-cons-qbs-morphism]]]) (auto
simp: pair-qbs-space to-list-from-list-ident)
```

```
      qed(simp add: list-qbs-def map-qbs-morphism-f)
    qed(auto simp: pair-qbs-space to-list-from-list-ident to-list-simp2)
  qed
 qed(auto simp: list-qbs-def to-list-from-list-ident intro!: map-qbs-morphism-inverse-f)
qed(simp add: from-list-to-list-ident)
qed

lemma rec-list-morphism':
 rec-list' ∈ qbs-space (Y ⇒Q (X ⇒Q (ΠQ n∈(UNIV :: nat set).ΠQ i∈{..<n}. X)
⇒Q Y ⇒Q Y) ⇒Q (ΠQ n∈(UNIV :: nat set).ΠQ i∈{..<n}. X) ⇒Q Y)
proof(intro curry-preserves-morphisms[OF arg-swap-morphism] coPiQ-canonical1')
 fix n
 show (λx y. rec-list' (fst y) (snd y) (n, x)) ∈ (ΠQ i∈{..<n}. X) →Q exp-qbs (Y
⨂Q exp-qbs X (exp-qbs (ΠQ n∈UNIV. ΠQ i∈{..<n}. X) (exp-qbs Y Y))) Y
 proof(induction n)
  case 0
  show ?case
  proof(rule curry-preserves-morphisms[OF qbs-morphismI])
   fix α
   assume h:α ∈ qbs-Mx ((ΠQ i∈{..<0::nat}. X) ⨂Q Y ⨂Q exp-qbs X (exp-qbs
(ΠQ n∈UNIV. ΠQ i∈{..<n::nat}. X) (exp-qbs Y Y)))
    have ⋀r. fst (α r) = (λn. undefined)
    proof −
     fix r
     have ⋀i. (λr. fst (α r) i) = (λr. undefined)
     using h by(auto simp: exp-qbs-Mx PiQ-Mx pair-qbs-Mx comp-def split-beta')
     thus fst (α r) = (λn. undefined)
      by(fastforce dest: fun-cong)
    qed
    hence (λxy. rec-list' (fst (snd xy)) (snd (snd xy)) (0, fst xy)) ∘ α = (λx. fst
(snd (α x)))
     by(auto simp: rec-list'-simp1[simplified list-nil-def] comp-def split-beta')
    also have ... ∈ qbs-Mx Y
     using h by(auto simp: pair-qbs-Mx comp-def)
    finally show (λxy. rec-list' (fst (snd xy)) (snd (snd xy)) (0, fst xy)) ∘ α ∈
qbs-Mx Y .
  qed
 next
  case ih:(Suc n)
  show ?case
  proof(rule qbs-morphismI)
   fix α
   assume h:α ∈ qbs-Mx (ΠQ i∈{..<Suc n}. X)
   define α' where α' ≡ (λr. snd (list-tail (Suc n, α r)))
   define a where a ≡ (λr. α r 0)
   then have ha:a ∈ qbs-Mx X
    using h by(auto simp: PiQ-Mx)
   have 1:α' ∈ qbs-Mx (ΠQ i∈{..<n}. X)
    using h by(fastforce simp: PiQ-Mx list-tail-def α'-def)
```

hence *2*: $\bigwedge r.\ (n,\ \alpha'\ r) \in$ *qbs-space* (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)
using *qbs-Mx-to-X*[*of* α'] **by** (*fastforce simp*: *PiQ-space coPiQ-space*)
have *3*: $\bigwedge r.$ (*Suc* n, α r) $\in$ *qbs-space* (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)
using *qbs-Mx-to-X*[*of* α] *h* **by** (*fastforce simp*: *PiQ-space coPiQ-space*)
have *4*: $\bigwedge r.$ $(n,\ \alpha'\ r)$ = *list-tail* (*Suc* n, α r)
by(*simp add*: *list-tail-def* α'*-def*)
have *5*: $\bigwedge r.$ (*Suc* n, α r) = *list-cons* (a r) (n, α' r)
unfolding *a-def* **by**(*simp add*: *list-simp5*[*OF 3*,*simplified 4*[*symmetric*],*simplified list-head-def list-cons-def list-nil-def*] *list-cons-def*) *auto*
have *6*: ($\lambda r.$ $(n,\ \alpha'\ r)$) $\in$ *qbs-Mx* (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)
using *1* **by**(*auto intro*!: *coPiQ-MxI simp*: *PiQ-space coPiQ-space*)

have ($\lambda x\ y.$ *rec-list'* (*fst* y) (*snd* y) (*Suc* n, x)) $\circ$ α = ($\lambda r\ y.$ *rec-list'* (*fst* y) (*snd* y) (*Suc* n, α r))
by *auto*
also have ... = ($\lambda r\ y.$ *snd* y (a r) (n, α' r) (*rec-list'* (*fst* y) (*snd* y) (n, α' r)))
by(*simp only*: *5 rec-list'-simp2*[*OF 2*])
also have ... $\in$ *qbs-Mx* (*exp-qbs* (Y $\bigotimes_Q$ *exp-qbs* X (*exp-qbs* (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X) (*exp-qbs* Y Y))) Y)
proof −
have (λ(r,y). *snd* y (a r) (n, α' r) (*rec-list'* (*fst* y) (*snd* y) (n, α' r))) = (λ(y,$x1$,$x2$,$x3$). y $x1$ $x2$ $x3$) $\circ$ (λ(r,y). (*snd* y, a r, (n, α' r), *rec-list'* (*fst* y) (*snd* y) (n, α' r)))
by *auto*
also have ... $\in$ *qbs-borel* $\bigotimes_Q$ (Y $\bigotimes_Q$ *exp-qbs* X (*exp-qbs* (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X) (*exp-qbs* Y Y))) $\to_Q$ Y
proof(*rule qbs-morphism-comp*[**where** Y=*exp-qbs* X (*exp-qbs* (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X) (*exp-qbs* Y Y)) $\bigotimes_Q$ X $\bigotimes_Q$ (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X) $\bigotimes_Q$ Y])
show (λ(r, y). (*snd* y, a r, (n, α' r), *rec-list'* (*fst* y) (*snd* y) (n, α' r))) $\in$ *qbs-borel* $\bigotimes_Q$ Y $\bigotimes_Q$ *exp-qbs* X (*exp-qbs* ((Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)) (*exp-qbs* Y Y)) $\to_Q$ *exp-qbs* X (*exp-qbs* ((Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)) (*exp-qbs* Y Y)) $\bigotimes_Q$ X $\bigotimes_Q$ (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X) $\bigotimes_Q$ Y
unfolding *split-beta'*
proof(*safe intro*!: *qbs-morphism-Pair*)
show ($\lambda x.$ a (*fst* x)) $\in$ *qbs-borel* $\bigotimes_Q$ Y $\bigotimes_Q$ *exp-qbs* X (*exp-qbs* ((Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)) (*exp-qbs* Y Y)) $\to_Q$ X
using *ha qbs-Mx-is-morphisms*[*of* X] *ha* **by** *auto*
next
show ($\lambda x.$ (n, α' (*fst* x))) $\in$ *qbs-borel* $\bigotimes_Q$ Y $\bigotimes_Q$ *exp-qbs* X (*exp-qbs* ((Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)) (*exp-qbs* Y Y)) $\to_Q$ (Π_Q $n{\in}(UNIV$ :: *nat set*).Π_Q $i{\in}\{..{<}n\}.$ X)
using *6* **by**(*simp add*: *qbs-Mx-is-morphisms*) (*use fst-qbs-morphism qbs-morphism-compose* **in** *blast*)
next
show ($\lambda x.$ *rec-list'* (*fst* (*snd* x)) (*snd* (*snd* x)) (n, α' (*fst* x))) $\in$ *qbs-borel*

$\bigotimes_Q$ *Y* $\bigotimes_Q$ *exp-qbs X* (*exp-qbs* ((Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)) (*exp-qbs Y Y*)) →$_Q$ *Y*
using *qbs-morphism-Mx*[*OF ih 1*, *simplified comp-def*] *uncurry-preserves-morphisms*[*of* (λ(*x*,*y*). *rec-list'* (*fst y*) (*snd y*) (*n*, α' *x*)) *qbs-borel Y* $\bigotimes_Q$ *exp-qbs X* (*exp-qbs* ((Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)) (*exp-qbs Y Y*)) *Y*] *qbs-Mx-is-morphisms*[*of* *exp-qbs* (*Y* $\bigotimes_Q$ *exp-qbs X* (*exp-qbs* ((Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)) (*exp-qbs Y Y*))) *Y*]
by(*fastforce simp*: *split-beta'*)
qed *qbs*
next
show (λ(*y*, *x1*, *x2*, *x3*). *y x1 x2 x3*) ∈ *exp-qbs X* (*exp-qbs* ((Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)) (*exp-qbs Y Y*)) $\bigotimes_Q$ *X* $\bigotimes_Q$ (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*) $\bigotimes_Q$ *Y* →$_Q$ *Y*
by *simp*
qed
finally show *?thesis*
by(*simp add*: *exp-qbs-Mx'*)
qed
finally show (λ*x y*. *rec-list'* (*fst y*) (*snd y*) (*Suc n*, *x*)) ∘ α ∈ *qbs-Mx* (*exp-qbs* (*Y* $\bigotimes_Q$ *exp-qbs X* (*exp-qbs* (Π$_Q$ *n*∈*UNIV*. Π$_Q$ *i*∈{..<*n*}. *X*) (*exp-qbs Y Y*))) *Y*)
by *blast*
qed
qed
qed *simp*

lemma *rec-list-morphism*[*qbs*]: *rec-list* ∈ *qbs-space* (*Y* ⇒$_Q$ (*X* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y*)
proof(*rule curry-preserves-morphisms*[*OF arg-swap-morphism*])
show (λ*l yf*. *rec-list* (*fst yf*) (*snd yf*) *l*) ∈ *list-qbs X* →$_Q$ *Y* $\bigotimes_Q$ (*X* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*) ⇒$_Q$ *Y*
proof(*rule qbs-morphism-cong'*[**where** *f*=(λ*l'* (*y*,*f*). *rec-list y f* (*to-list l'*)) ∘ *from-list*,*OF - qbs-morphism-comp*[**where** *Y*=(Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*)]])
show (λ*l'* (*y*,*f*). *rec-list y f* (*to-list l'*)) ∈ (Π$_Q$ *n*∈(*UNIV* :: *nat set*).Π$_Q$ *i*∈{..<*n*}. *X*) →$_Q$ *Y* $\bigotimes_Q$ (*X* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*) ⇒$_Q$ *Y*
apply(*rule arg-swap-morphism*,*simp only*: *split-beta' list-qbs-def*)
apply(*rule uncurry-preserves-morphisms*)
apply(*rule arg-swap-morphism*)
apply(*rule arg-swap-morphism'*)
apply(*rule qbs-morphism-cong'*[*OF - arg-swap-morphism-map-qbs1*[*OF arg-swap-morphism'*[*OF arg-swap-morphism*[*OF rec-list-morphism'*]]]])
apply(*auto simp*: *rec-list'-def from-list-to-list-ident*)
done
qed(*auto simp*: *from-list-to-list-ident list-qbs-def to-list-from-list-ident intro*!: *map-qbs-morphism-inverse-f*)
qed

hide-const (**open**) *list-nil list-cons list-head list-tail from-list rec-list' to-list'*

hide-fact (**open**) *list-simp1 list-simp2 list-simp3 list-simp4 list-simp5 list-simp6*

list-simp7 from-list-in-list-of′ list-cons-qbs-morphism rec-list′-simp1 to-list-from-list-ident from-list-in-list-of to-list-set to-list-simp1 to-list-simp2 list-head-def list-tail-def from-list-length list-cons-in-list-of rec-list-morphism′ rec-list′-simp2 list-decomp1 list-destruct-rule list-induct-rule from-list-to-list-ident

corollary *case-list-morphism*[*qbs*]: *case-list* ∈ *qbs-space* ((*Y* :: *'b quasi-borel*) ⇒$_Q$ ((*X* :: *'a quasi-borel*) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y*) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y*)
proof −
have [*simp*]:*case-list* = (λ*y* (*f* :: *'a* ⇒ *'a list* ⇒ *'b*) *l*. *rec-list y* (λ*x l′ y*. *f x l′*) *l*)
proof *standard*+
fix *y* :: *'b* **and** *f* :: *'a* ⇒ *'a list* ⇒ *'b* **and** *l* :: *'a list*
show (*case l of* [] ⇒ *y* | *x* # *xa* ⇒ *f x xa*) = *rec-list y* (λ*x l′ y*. *f x l′*) *l*
by (*cases l*) *auto*
qed
show *?thesis*
by *simp*
qed

lemma *fold-qbs-morphism*[*qbs*]: *fold* ∈ *qbs-space* ((*X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*)
proof −
have [*simp*]:*fold* = (λ*f l*. *rec-list id* (λ*x xs l*. *l* ∘ *f x*) *l*)
apply *standard*+
subgoal for *f l x*
by(*induction l arbitrary*: *x*) *simp-all*
done
show *?thesis*
by *simp*
qed

lemma [*qbs*]:
shows *foldr-qbs-morphism*: *foldr* ∈ *qbs-space* ((*X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *Y* ⇒$_Q$ *Y*)
and *foldl-qbs-morphism*: *foldl* ∈ *qbs-space* ((*X* ⇒$_Q$ *Y* ⇒$_Q$ *X*) ⇒$_Q$ *X* ⇒$_Q$ *list-qbs Y* ⇒$_Q$ *X*)
and *zip-qbs-morphism*: *zip* ∈ *qbs-space* (*list-qbs X* ⇒$_Q$ *list-qbs Y* ⇒$_Q$ *list-qbs* (*pair-qbs X Y*))
and *append-qbs-morphism*: *append* ∈ *qbs-space* (*list-qbs X* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*)
and *concat-qbs-morphism*: *concat* ∈ *qbs-space* (*list-qbs* (*list-qbs X*) ⇒$_Q$ *list-qbs X*)
and *drop-qbs-morphism*: *drop* ∈ *qbs-space* (*qbs-count-space UNIV* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*)
and *take-qbs-morphism*: *take* ∈ *qbs-space* (*qbs-count-space UNIV* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*)
and *rev-qbs-morphism*: *rev* ∈ *qbs-space* (*list-qbs X* ⇒$_Q$ *list-qbs X*)
by(*auto simp*: *foldr-def foldl-def zip-def append-def concat-def drop-def take-def rev-def*)

lemma [*qbs*]:
fixes $X :: 'a$ *quasi-borel* **and** $Y :: 'b$ *quasi-borel*
shows *map-qbs-morphism*: $map \in qbs\text{-}space\ ((X \Rightarrow_Q Y) \Rightarrow_Q list\text{-}qbs\ X \Rightarrow_Q list\text{-}qbs\ Y)$ (**is** *?map*)
and *fileter-qbs-morphism*: $filter \in qbs\text{-}space\ ((X \Rightarrow_Q count\text{-}space_Q\ UNIV) \Rightarrow_Q list\text{-}qbs\ X \Rightarrow_Q list\text{-}qbs\ X)$ (**is** *?filter*)
and *length-qbs-morphism*: $length \in qbs\text{-}space\ (list\text{-}qbs\ X \Rightarrow_Q qbs\text{-}count\text{-}space\ UNIV)$ (**is** *?length*)
and *tl-qbs-morphism*: $tl \in qbs\text{-}space\ (list\text{-}qbs\ X \Rightarrow_Q list\text{-}qbs\ X)$ (**is** *?tl*)
and *list-all-qbs-morphism*: $list\text{-}all \in qbs\text{-}space\ ((X \Rightarrow_Q qbs\text{-}count\text{-}space\ UNIV) \Rightarrow_Q list\text{-}qbs\ X \Rightarrow_Q qbs\text{-}count\text{-}space\ UNIV)$ (**is** *?list-all*)
and *bind-list-qbs-morphism*: $(\gg\!=) \in qbs\text{-}space\ (list\text{-}qbs\ X \Rightarrow_Q (X \Rightarrow_Q list\text{-}qbs\ Y) \Rightarrow_Q list\text{-}qbs\ Y)$ (**is** *?bind*)
proof −
have [*simp*]: $map = (\lambda f.\ rec\text{-}list\ []\ (\lambda x\ xs\ l.\ f\ x\ \#\ l))$
apply *standard+*
subgoal for *f l*
by(*induction l*) *simp-all*
done
have [*simp*]: $filter = (\lambda P.\ rec\text{-}list\ []\ (\lambda x\ xs\ l.\ if\ P\ x\ then\ x\ \#\ l\ else\ l))$
apply *standard+*
subgoal for *f l*
by(*induction l*) *simp-all*
done
have [*simp*]: $length = (\lambda l.\ foldr\ (\lambda\text{-}\ n.\ Suc\ n)\ l\ 0)$
apply *standard*
subgoal for *l*
by (*induction l*) *simp-all*
done
have [*simp*]: $tl = (\lambda l.\ case\ l\ of\ []\ \Rightarrow\ []\ |\ \text{-}\ \#\ xs\ \Rightarrow\ xs)$
by *standard* (*simp add*: *tl-def*)
have [*simp*]: $list\text{-}all = (\lambda P\ xs.\ foldr\ (\lambda x\ b.\ b \wedge P\ x)\ xs\ True)$
apply (*standard*,*standard*)
subgoal for *P xs*
by(*induction xs arbitrary*: *P*) *auto*
done
have [*simp*]: $List.bind = (\lambda xs\ f.\ concat\ (map\ f\ xs))$
by *standard+* (*simp add*: *List.bind-def*)
show *?map ?filter ?length ?tl ?list-all ?bind*
by *simp-all*
qed

lemma *list-eq-qbs-morphism*[*qbs*]:
assumes [*qbs*]: $(=) \in qbs\text{-}space\ (X \Rightarrow_Q X \Rightarrow_Q count\text{-}space\ UNIV)$
shows $(=) \in qbs\text{-}space\ (list\text{-}qbs\ X \Rightarrow_Q list\text{-}qbs\ X \Rightarrow_Q count\text{-}space\ UNIV)$
proof −
have [*simp*]:$(=) = (\lambda xs\ ys.\ length\ xs = length\ ys \wedge list\text{-}all\ (case\text{-}prod\ (=))\ (zip\ xs\ ys))$

using *Ball-set list-eq-iff-zip-eq* **by** *fastforce*
show *?thesis*
by *simp*
qed

lemma *insort-key-qbs-morphism*[*qbs*]:
shows *insort-key* ∈ *qbs-space* ((*X* ⇒$_Q$ (*borel*$_Q$ ::*'b* :: {*second-countable-topology*, *linorder-topology*} *quasi-borel*)) ⇒$_Q$ *X* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*) (**is** *?g1*)
and *insort-key* ∈ *qbs-space* ((*X* ⇒$_Q$ *count-space*$_Q$ (*UNIV* :: (- :: *countable*) *set*)) ⇒$_Q$ *X* ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*) (**is** *?g2*)
proof −
have [*simp*]:*insort-key* = (λ*f x*. *rec-list* [*x*] (λ*y ys l*. *if f x* ≤ *f y then x*#*y*#*ys else y*#*l*))
apply *standard*+
subgoal for *f x l*
by(*induction l*) *simp-all*
done
show *?g1 ?g2*
by *simp-all*
qed

lemma *sort-key-qbs-morphism*[*qbs*]:
shows *sort-key* ∈ *qbs-space* ((*X* ⇒$_Q$ (*borel*$_Q$ ::*'b* :: {*second-countable-topology*, *linorder-topology*} *quasi-borel*)) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*)
and *sort-key* ∈ *qbs-space* ((*X* ⇒$_Q$ *count-space*$_Q$ (*UNIV* :: (- :: *countable*) *set*)) ⇒$_Q$ *list-qbs X* ⇒$_Q$ *list-qbs X*)
unfolding *sort-key-def* **by** *simp-all*

lemma *sort-qbs-morphism*[*qbs*]:
shows *sort* ∈ *list-qbs* (*borel*$_Q$ ::*'b* :: {*second-countable-topology*, *linorder-topology*} *quasi-borel*) →$_Q$ *list-qbs borel*$_Q$
and *sort* ∈ *list-qbs* (*count-space*$_Q$ (*UNIV* :: (- :: *countable*) *set*)) →$_Q$ *list-qbs* (*count-space*$_Q$ *UNIV*)
by *simp-all*

3.3.4 Morphism Pred

abbreviation *qbs-pred X P* ≡ *P* ∈ *X* →$_Q$ *qbs-count-space* (*UNIV* :: *bool set*)

lemma *qbs-pred-iff-measurable-pred*:
qbs-pred X P = *Measurable.pred* (*qbs-to-measure X*) *P*
by(*simp add*: *lr-adjunction-correspondence*)

lemma(**in** *standard-borel*) *qbs-pred-iff-measurable-pred*:
qbs-pred (*measure-to-qbs M*) *P* = *Measurable.pred M P*
by(*simp add*: *qbs-pred-iff-measurable-pred measurable-cong-sets*[*OF lr-sets-ident refl*])

lemma *qbs-pred-iff-sets*:

$\{x \in space\ (qbs\text{-}to\text{-}measure\ X).\ P\ x\} \in sets\ (qbs\text{-}to\text{-}measure\ X) \longleftrightarrow qbs\text{-}pred\ X\ P$
by (*simp add*: *Measurable.pred-def lr-adjunction-correspondence space-L*)

lemma
assumes [*qbs*]:$P \in X \rightarrow_Q Y \Rightarrow_Q qbs\text{-}count\text{-}space\ UNIV\ f \in X \rightarrow_Q Y$
shows *indicator-qbs-morphism'''*: $(\lambda x.\ indicator\ \{y.\ P\ x\ y\}\ (f\ x)) \in X \rightarrow_Q$ *qbs-borel* (**is** *?g1*)
and *indicator-qbs-morphism''*: $(\lambda x.\ indicator\ \{y \in qbs\text{-}space\ Y.\ P\ x\ y\}\ (f\ x)) \in X \rightarrow_Q qbs\text{-}borel$ (**is** *?g2*)
proof −
have [*simp*]:$\{x \in qbs\text{-}space\ X.\ P\ x\ (f\ x)\} = \{x \in qbs\text{-}space\ X.\ f\ x \in qbs\text{-}space\ Y \wedge P\ x\ (f\ x)\}$
using *qbs-morphism-space*[*OF assms*(*2*)] **by** *blast*
show *?g1 ?g2*
using *qbs-morphism-app*[*OF assms*,*simplified qbs-pred-iff-sets*[*symmetric*]] *qbs-morphism-space*[*OF assms*(*2*)]
by(*auto intro*!: *borel-measurable-indicator'* *simp*: *lr-adjunction-correspondence space-L*)
qed

lemma
assumes [*qbs*]:$P \in X \rightarrow_Q Y \Rightarrow_Q qbs\text{-}count\text{-}space\ UNIV$
shows *indicator-qbs-morphism*[*qbs*]:$(\lambda x.\ indicator\ \{y \in qbs\text{-}space\ Y.\ P\ x\ y\}) \in X \rightarrow_Q Y \Rightarrow_Q qbs\text{-}borel$ (**is** *?g1*)
and *indicator-qbs-morphism'*:$(\lambda x.\ indicator\ \{y.\ P\ x\ y\}) \in X \rightarrow_Q Y \Rightarrow_Q qbs\text{-}borel$ (**is** *?g2*)
proof −
note *indicator-qbs-morphism''*[*qbs*] *indicator-qbs-morphism'''*[*qbs*]
show *?g1 ?g2*
by(*auto intro*!: *curry-preserves-morphisms*[*OF pair-qbs-morphismI*] *simp*: *qbs-Mx-is-morphisms*)
qed

lemma *indicator-qbs*[*qbs*]:
assumes *qbs-pred X P*
shows *indicator* $\{x.\ P\ x\} \in X \rightarrow_Q$ *qbs-borel*
using *assms* **by**(*auto simp*: *lr-adjunction-correspondence*)

lemma *All-qbs-pred*[*qbs*]: *qbs-pred* $(count\text{-}space_Q\ (UNIV :: ('a :: countable)\ set) \Rightarrow_Q count\text{-}space_Q\ UNIV)\ All$
proof(*rule qbs-morphismI*)
fix $a :: real \Rightarrow 'a \Rightarrow bool$
assume $a \in qbs\text{-}Mx\ (count\text{-}space_Q\ UNIV \Rightarrow_Q count\text{-}space_Q\ UNIV)$
hence [*measurable*]: $\bigwedge f\ g.\ f \in borel\text{-}measurable\ borel \Longrightarrow g \in borel \rightarrow_M count\text{-}space\ UNIV \Longrightarrow (\lambda x{::}real.\ a\ (f\ x)\ (g\ x)) \in borel \rightarrow_M count\text{-}space\ UNIV$
by(*auto simp add*: *exp-qbs-Mx qbs-Mx-R*)
show $All \circ a \in qbs\text{-}Mx\ (count\text{-}space_Q\ UNIV)$
by(*simp add*: *comp-def qbs-Mx-R*)
qed

lemma *Ex-qbs-pred*[*qbs*]: *qbs-pred* (*count-space*$_Q$ (*UNIV* :: ('*a* :: *countable*) *set*) $\Rightarrow_Q$ *count-space*$_Q$ *UNIV*) *Ex*
proof(*rule qbs-morphismI*)
fix *a* :: *real* $\Rightarrow$ '*a* $\Rightarrow$ *bool*
assume *a* $\in$ *qbs-Mx* (*count-space*$_Q$ *UNIV* $\Rightarrow_Q$ *count-space*$_Q$ *UNIV*)
hence [*measurable*]: $\bigwedge f\, g.\ f \in$ *borel-measurable borel* $\Longrightarrow$ $g \in$ *borel* $\rightarrow_M$ *count-space UNIV* $\Longrightarrow$ (λx::*real*. *a* (*f x*) (*g x*)) $\in$ *borel* $\rightarrow_M$ *count-space UNIV*
by(*auto simp add*: *exp-qbs-Mx qbs-Mx-R*)
show *Ex* $\circ$ *a* $\in$ *qbs-Mx* (*count-space*$_Q$ *UNIV*)
by(*simp add*: *comp-def qbs-Mx-R*)
qed

lemma *Ball-qbs-pred-countable*:
assumes $\bigwedge i$::'*a* :: *countable*. *i* $\in$ *I* $\Longrightarrow$ *qbs-pred X* (*P i*)
shows *qbs-pred X* (λx. $\forall\, x{\in}I$. *P i x*)
using *assms* **by**(*simp add*: *qbs-pred-iff-measurable-pred*)

lemma *Ball-qbs-pred*:
assumes *finite I* $\bigwedge i$. *i* $\in$ *I* $\Longrightarrow$ *qbs-pred X* (*P i*)
shows *qbs-pred X* (λx. $\forall\, x{\in}I$. *P i x*)
using *assms* **by**(*simp add*: *qbs-pred-iff-measurable-pred*)

lemma *Bex-qbs-pred-countable*:
assumes $\bigwedge i$::'*a* :: *countable*. *i* $\in$ *I* $\Longrightarrow$ *qbs-pred X* (*P i*)
shows *qbs-pred X* (λx. $\exists\, x{\in}I$. *P i x*)
using *assms* **by**(*simp add*: *qbs-pred-iff-measurable-pred*)

lemma *Bex-qbs-pred*:
assumes *finite I* $\bigwedge i$. *i* $\in$ *I* $\Longrightarrow$ *qbs-pred X* (*P i*)
shows *qbs-pred X* (λx. $\exists\, x{\in}I$. *P i x*)
using *assms* **by**(*simp add*: *qbs-pred-iff-measurable-pred*)

lemma *qbs-morphism-If-sub-qbs*:
assumes [*qbs*]: *qbs-pred X P*
and [*qbs*]: *f* $\in$ *sub-qbs X* {$x{\in}$*qbs-space X*. *P x*} $\rightarrow_Q$ *Y* *g* $\in$ *sub-qbs X* {$x{\in}$*qbs-space X*. $\neg$ *P x*} $\rightarrow_Q$ *Y*
shows (λx. *if P x then f x else g x*) $\in$ *X* $\rightarrow_Q$ *Y*
proof(*rule qbs-morphismI*)
fix α
assume *h*:α $\in$ *qbs-Mx X*
interpret *standard-borel-ne borel* :: *real measure* **by** *simp*
have [*measurable*]: *Measurable.pred borel* (λx. *P* (α *x*))
using *h* **by**(*simp add*: *qbs-pred-iff-measurable-pred*[*symmetric*] *qbs-Mx-is-morphisms*)
consider *qbs-space X* = {}
| {$x{\in}$*qbs-space X*. $\neg$ *P x*} = *qbs-space X*
| {$x{\in}$*qbs-space X*. *P x*} = *qbs-space X*
| {$x{\in}$*qbs-space X*. *P x*} $\neq$ {} {$x{\in}$*qbs-space X*. $\neg$ *P x*} $\neq$ {} **by** *blast*
then show (λx. *if P x then f x else g x*) $\circ$ α $\in$ *qbs-Mx Y* (**is** *?f* $\in$ -)
proof *cases*

case *1*
with *h* **show** *?thesis*
by(*simp add*: *qbs-empty-equiv*)
next
case *2*
have [*simp*]:(λx. *if P x then f x else g x*) $\circ$ α = *g* $\circ$ α
by *standard* (*use qbs-Mx-to-X*[*OF h*] *2* **in** *auto*)
show *?thesis*
using *2 qbs-morphism-Mx*[*OF assms*(*3*)] *h* **by**(*simp add*: *sub-qbs-ident*)
next
case *3*
have [*simp*]:(λx. *if P x then f x else g x*) $\circ$ α = *f* $\circ$ α
by *standard* (*use qbs-Mx-to-X*[*OF h*] *3* **in** *auto*)
show *?thesis*
using *3 qbs-morphism-Mx*[*OF assms*(*2*)] *h* **by**(*simp add*: *sub-qbs-ident*)
next
case *4*
then obtain *x0 x1* **where**
x0:*x0* $\in$ *qbs-space X P x0* **and** *x1*:*x1* $\in$ *qbs-space X* $\neg$ *P x1*
by *blast*
define *a0* **where** *a0* = (λr. *if P* (α *r*) *then* α *r else x0*)
define *a1* **where** *a1* = (λr. *if* $\neg$ *P* (α *r*) *then* α *r else x1*)
have *a0* $\in$ *qbs-Mx* (*sub-qbs X* {*x*∈*qbs-space X*. *P x*}) *a1* $\in$ *qbs-Mx* (*sub-qbs X* {*x*∈*qbs-space X*. $\neg$ *P x*})
using *x0 x1 qbs-Mx-to-X*[*OF h*] *h*
by(*auto simp*: *sub-qbs-Mx a0-def a1-def intro*!: *qbs-closed3-dest2'*[*of UNIV* λr. *P* (α *r*) λb *r*. *if b then* α *r else x0*]) (*simp-all add*: *qbs-Mx-is-morphisms*)
from *qbs-morphism-Mx*[*OF assms*(*2*) *this*(*1*)] *qbs-morphism-Mx*[*OF assms*(*3*) *this*(*2*)]
have *h0*:(λr. *f* (*a0 r*)) $\in$ *qbs-Mx Y* (λr. *g* (*a1 r*)) $\in$ *qbs-Mx Y*
by (*simp-all add*: *comp-def*)
have [*simp*]:(λx. *if P x then f x else g x*) $\circ$ α = (λr. *if P* (α *r*) *then f* (*a0 r*) *else g* (*a1 r*))
by *standard* (*auto simp*: *comp-def a0-def a1-def*)
show (λx. *if P x then f x else g x*) $\circ$ α $\in$ *qbs-Mx Y*
using *h h0* **by**(*simp add*: *qbs-Mx-is-morphisms*)
qed
qed

3.3.5 The Adjunction w.r.t. Ordering

lemma *l-mono*: *mono qbs-to-measure*
proof
fix *X Y* :: *'a quasi-borel*
show *X* $\leq$ *Y* $\Longrightarrow$ *qbs-to-measure X* $\leq$ *qbs-to-measure Y*
proof(*induction rule*: *less-eq-quasi-borel.induct*)
case (*1 X Y*)
then show *?case*
by(*simp add*: *less-eq-measure.intros*(*1*) *space-L*)

```
  next
    case (2 X Y)
    then have sigma-Mx X ⊆ sigma-Mx Y
      by(auto simp add: sigma-Mx-def)
    then consider sigma-Mx X ⊂ sigma-Mx Y | sigma-Mx X = sigma-Mx Y
      by auto
    then show ?case
      apply(cases)
       apply(rule less-eq-measure.intros(2))
        apply(simp-all add: 2 space-L sets-L)
      by(rule less-eq-measure.intros(3),simp-all add: 2 sets-L space-L emeasure-L)
  qed
qed

lemma r-mono: mono measure-to-qbs
proof
  fix M N :: 'a measure
  show M ≤ N ⟹ measure-to-qbs M ≤ measure-to-qbs N
  proof(induction rule: less-eq-measure.inducts)
    case (1 M N)
    then show ?case
      by(simp add: less-eq-quasi-borel.intros(1) qbs-space-R)
  next
    case (2 M N)
    then have (borel :: real measure) →M N ⊆ borel →M M
      by(simp add: measurable-mono)
     then consider (borel :: real measure) →M N ⊂ borel →M M | (borel :: real
measure) →M N = borel →M M
      by auto
    then show ?case
     by cases (rule less-eq-quasi-borel.intros(2),simp-all add: 2 qbs-space-R qbs-Mx-R)+
  next
    case (3 M N)
    then show ?case
      apply −
     by(rule less-eq-quasi-borel.intros(2)) (simp-all add: measurable-mono qbs-space-R
qbs-Mx-R)
  qed
qed

lemma rl-order-adjunction:
  X ≤ qbs-to-measure Y ⟷ measure-to-qbs X ≤ Y
proof
  assume 1: X ≤ qbs-to-measure Y
  then show measure-to-qbs X ≤ Y
  proof(induction rule: less-eq-measure.cases)
    case (1 M N)
    then have [simp]:qbs-space Y = space N
      by(simp add: 1(2)[symmetric] space-L)
```

```
  show ?case
    by(rule less-eq-quasi-borel.intros(1),simp add: 1 qbs-space-R)
next
  case (2 M N)
  then have [simp]:qbs-space Y = space N
    by(simp add: 2(2)[symmetric] space-L)
  show ?case
  proof(rule less-eq-quasi-borel.intros(2))
    show qbs-Mx Y ⊆ qbs-Mx (measure-to-qbs X)
      unfolding qbs-Mx-R
    proof
      fix α
      assume h:α ∈ qbs-Mx Y
      show α ∈ borel →M X
      proof(rule measurableI)
        show ⋀x. α x ∈ space X
          using qbs-Mx-to-X[OF h] by (auto simp add: 2)
      next
        fix A
        assume A ∈ sets X
        then have A ∈ sets (qbs-to-measure Y)
          using 2 by auto
        then obtain U where
          hu:A = U ∩ space N (∀α∈qbs-Mx Y. α -‘ U ∈ sets borel)
          by(auto simp add: sigma-Mx-def sets-L)
        have α -‘ A = α -‘ U
          using qbs-Mx-to-X[OF h] by(auto simp add: hu)
        thus α -‘ A ∩ space borel ∈ sets borel
          using h hu(2) by simp
      qed
    qed
  qed(auto simp: 2 qbs-space-R)
next
  case (3 M N)
  then have [simp]:qbs-space Y = space N
    by(simp add: 3(2)[symmetric] space-L)
  show ?case
  proof(rule less-eq-quasi-borel.intros(2))
    show qbs-Mx Y ⊆ qbs-Mx (measure-to-qbs X)
      unfolding qbs-Mx-R
    proof
      fix α
      assume h:α ∈ qbs-Mx Y
      show α ∈ borel →M X
      proof(rule measurableI)
        show ⋀x. α x ∈ space X
          using qbs-Mx-to-X[OF h] by(auto simp: 3)
      next
        fix A
```

```
        assume A ∈ sets X
        then have A ∈ sets (qbs-to-measure Y)
          using 3 by auto
        then obtain U where
          hu:A = U ∩ space N (∀ α∈qbs-Mx Y. α −‘ U ∈ sets borel)
          by(auto simp add: sigma-Mx-def sets-L)
        have α −‘ A = α −‘ U
          using qbs-Mx-to-X[OF h] by(auto simp add: hu)
        thus α −‘ A ∩ space borel ∈ sets borel
          using h hu(2) by simp
      qed
    qed
  qed(auto simp: 3 qbs-space-R)
 qed
next
 assume measure-to-qbs X ≤ Y
 then show X ≤ qbs-to-measure Y
 proof(induction rule: less-eq-quasi-borel.cases)
  case (1 A B)
  have [simp]: space X = qbs-space A
   by(simp add: 1(1)[symmetric] qbs-space-R)
  show ?case
   by(rule less-eq-measure.intros(1)) (simp add: 1 space-L)
 next
  case (2 A B)
  then have hmy:qbs-Mx Y ⊆ borel →M X
   using qbs-Mx-R by blast
  have [simp]: space X = qbs-space A
   by(simp add: 2(1)[symmetric] qbs-space-R)
  have sets X ⊆ sigma-Mx Y
  proof
   fix U
   assume hu:U ∈ sets X
   show U ∈ sigma-Mx Y
    unfolding sigma-Mx-def
   proof(safe intro!: exI[where x=U])
    show ⋀x. x ∈ U ⟹ x ∈ qbs-space Y
     using sets.sets-into-space[OF hu]
     by(auto simp add: 2)
   next
    fix α
    assume α ∈ qbs-Mx Y
    then have α ∈ borel →M X
     using hmy by(auto)
    thus α −‘ U ∈ sets borel
     using hu by(simp add: measurable-sets-borel)
   qed
  qed
  then consider sets X = sigma-Mx Y | sets X ⊂ sigma-Mx Y
```

```
      by auto
    then show ?case
    proof cases
      case 1
      show ?thesis
      proof(rule less-eq-measure.intros(3))
        show emeasure X ≤ emeasure (qbs-to-measure Y)
          unfolding emeasure-L
        proof(rule le-funI)
          fix U
          consider U = {} | U ∉ sigma-Mx Y | U ≠ {} ∧ U ∈ sigma-Mx Y
            by auto
          then show emeasure X U ≤ (if U = {} ∨ U ∉ sigma-Mx Y then 0 else
∞)
          proof cases
            case 1
            then show ?thesis by simp
          next
            case h:2
            then have U ∉ sigma-Mx A
            using qbs-Mx-sigma-Mx-contra[OF 2(3)[symmetric] 2(4)] 2(2) by auto
            hence U ∉ sets X
              using lr-sets 2(1) sets-L by blast
            thus ?thesis
              by(simp add: h emeasure-notin-sets)
          next
            case 3
            then show ?thesis
            by simp
        qed
      qed
    qed(simp-all add: 1 2 space-L sets-L)
    next
      case h2:2
      show ?thesis
      by(rule less-eq-measure.intros(2)) (simp add: space-L 2, simp add: h2 sets-L)
    qed
  qed
qed

end
```

4 The S-Finite Measure Monad

```
theory Monad-QuasiBorel
  imports
    Measure-QuasiBorel-Adjunction
    Kernels
```

begin

4.1 The s-Finite Measure Monad

- In the previous version:
  - A measure on $X = [X, \alpha, \mu]_\sim$
    - $\alpha \in M_X$
    - μ is an s-finite measure on $\mathbb{R}$
    - $(X, \alpha, \mu) \sim (X, \beta, \nu) \iff \alpha_*\mu = \beta_*\nu$
  - The s-finite measure monad: $\mathscr{M}(X) = \{p \mid p \text{ is a measure on } X\}$
- Current version: measures are not restricted to s-finite measures.
  - A measure on $X = [X, \alpha, \mu]_\sim$
    - $\alpha \in M_X$
    - μ is a measure on $\mathbb{R}$
    - $(X, \alpha, \mu) \sim (X, \beta, \nu) \iff \alpha_*\mu = \beta_*\nu$
  - The s-finite measure monad: $\mathscr{M}(X) = \{[X, \alpha, \mu]_\sim \mid \mu \text{ is s-finite}\}$
    The space of all measures: $\mathscr{M}_{\text{all}}(X) = \{p \mid p \text{ is a measure on } X\}$

4.1.1 Measures on Quasi-Borel spaces

locale *in-Mx* =
  fixes X :: *'a quasi-borel*
    and α :: *real* ⇒ *'a*
  assumes *in-Mx*[*simp*]:$\alpha \in$ *qbs-Mx X*
begin

lemma α-*measurable*[*measurable*]: $\alpha \in$ *borel* $\to_M$ *qbs-to-measure X*
  using *in-Mx qbs-Mx-subset-of-measurable* **by** *blast*

lemma α-*qbs-morphism*[*qbs*]: $\alpha \in$ *qbs-borel* $\to_Q$ *X*
  using *in-Mx* **by**(*simp only*: *qbs-Mx-is-morphisms*)

lemma *X-not-empty*: *qbs-space X* ≠ {}
  using *in-Mx* **by**(*auto simp*: *qbs-empty-equiv simp del*: *in-Mx*)

lemma *inverse-UNIV*[*simp*]: α −‘ (*qbs-space X*) = *UNIV*
  by *fastforce*

end

locale *qbs-meas* = *in-Mx X* α
  for X :: *'a quasi-borel* **and** α **and** μ :: *real measure* +

assumes *mu-sets*[*measurable-cong*]: *sets* μ = *sets borel*
begin

lemma *mu-not-empty*: *space* $\mu \neq \{\}$
by(*simp add*: *sets-eq-imp-space-eq*[*OF mu-sets*])

end

lemma *qbs-meas-All*:
assumes $\alpha \in$ *qbs-Mx X measure-kernel M borel k x* $\in$ *space M*
shows *qbs-meas X* α (*k x*)
proof −
interpret *measure-kernel M borel k* **by** *fact*
show *?thesis*
using *assms*(*1*,*3*) **by**(*auto simp*: *qbs-meas-def in-Mx-def qbs-meas-axioms-def kernel-sets*)
qed

locale *qbs-s-finite* = *qbs-meas* + *s-finite-measure* μ

lemma *qbs-s-finite-All*:
assumes $\alpha \in$ *qbs-Mx X s-finite-kernel M borel k x* $\in$ *space M*
shows *qbs-s-finite X* α (*k x*)
proof −
interpret *s-finite-kernel M borel k* **by** *fact*
show *?thesis*
using *assms*(*1*,*3*) *image-s-finite-measure*[*OF assms*(*3*)]
by(*auto simp*: *qbs-s-finite-def in-Mx-def kernel-sets qbs-meas-def qbs-meas-axioms-def*)
qed

locale *qbs-prob* = *in-Mx X* α + *real-distribution* μ
for *X* :: *'a quasi-borel* **and** α μ
begin

lemma *qbs-meas*: *qbs-meas X* α μ
by(*auto simp*: *qbs-meas-def qbs-meas-axioms-def in-Mx-def*)

lemma *qbs-s-finite*: *qbs-s-finite X* α μ
by(*auto simp*: *qbs-s-finite-def qbs-meas-def qbs-meas-axioms-def in-Mx-def s-finite-measure-prob*)

sublocale *qbs-s-finite* **by**(*rule qbs-s-finite*)

end

locale *pair-qbs-meas'* = *pq1*: *qbs-meas X* α μ + *pq2*: *qbs-meas Y* β ν
for *X* :: *'a quasi-borel* **and** α μ **and** *Y* :: *'b quasi-borel* **and** β ν
begin

lemma *ab-measurable*[*measurable*]: *map-prod* α $\beta \in$ *borel* $\bigotimes_M$ *borel* $\rightarrow_M$ *qbs-to-measure*

$(X \bigotimes_Q Y)$
proof −
have *map-prod* α β $\in$ *qbs-to-measure* (*measure-to-qbs* (*borel* $\bigotimes_M$ *borel*)) $\rightarrow_M$ *qbs-to-measure* $(X \bigotimes_Q Y)$
by(*auto intro*!: *set-mp*[*OF l-preserves-morphisms*] *simp*: *r-preserves-product*)
moreover have *sets* (*qbs-to-measure* (*measure-to-qbs* (*borel* $\bigotimes_M$ *borel*))) = *sets* ((*borel* $\bigotimes_M$ *borel*) :: (*real* × *real*) *measure*)
by(*auto intro*!: *standard-borel.lr-sets-ident pair-standard-borel-ne standard-borel-ne.standard-borel*)
ultimately show *?thesis* **by** *simp*
qed

end

locale *pair-qbs-meas* = *pq1*: *qbs-meas* X α μ + *pq2*: *qbs-meas* X β ν
for X :: *'a quasi-borel* **and** α μ β ν
begin

sublocale *pair-qbs-meas'* X α μ X β ν
by *standard*

end

locale *pair-qbs-s-finites* = *pq1*: *qbs-s-finite* X α μ + *pq2*: *qbs-s-finite* Y β ν
for X :: *'a quasi-borel* **and** α μ **and** Y :: *'b quasi-borel* **and** β ν
begin

sublocale *pair-qbs-meas'* X α μ Y β ν
by *standard*

end

locale *pair-qbs-s-finite* = *pq1*: *qbs-s-finite* X α μ + *pq2*: *qbs-s-finite* X β ν
for X :: *'a quasi-borel* **and** α μ **and** β ν
begin

sublocale *pair-qbs-s-finites* X α μ X β ν
by *standard*

sublocale *pair-qbs-meas* X α μ β ν
by *standard*

end

locale *pair-qbs-probs* = *pq1*: *qbs-prob* X α μ + *pq2*: *qbs-prob* Y β ν
for X :: *'a quasi-borel* **and** α μ **and** Y :: *'b quasi-borel* **and** β ν
begin

sublocale *pair-qbs-s-finites*
by *standard*

end

locale *pair-qbs-prob* = *pq1*: *qbs-prob X* α μ + *pq2*: *qbs-prob X* β ν
  for *X* :: *'a quasi-borel* **and** α μ **and** β ν
begin

sublocale *pair-qbs-s-finite X* α μ β ν
  by *standard*

sublocale *pair-qbs-probs X* α μ *X* β μ
  by *standard*

end

lemma(**in** *qbs-meas*) *qbs-probI*: *prob-space* $\mu \Longrightarrow$ *qbs-prob X* α μ
  by(*auto simp*: *qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def mu-sets*)

type-synonym *'a qbs-measure-t* = *'a quasi-borel* $*$ (*real* $\Rightarrow$ *'a*) $*$ *real measure*
definition *qbs-meas-eq* :: [*'a qbs-measure-t*, *'a qbs-measure-t*] $\Rightarrow$ *bool* **where**
  qbs-meas-eq p1 p2 $\equiv$
   (*let* (*X*, α, μ) = *p1*;
        (*Y*, β, ν) = *p2 in*
    qbs-meas X α μ $\wedge$ *qbs-meas Y* β ν $\wedge$ *X* = *Y* $\wedge$
      distr μ (*qbs-to-measure X*) α = *distr* ν (*qbs-to-measure Y*) β)

lemma *qbs-meas-eq-def2*:
 qbs-meas-eq p1 p2 =
   (*let* (*X*::*'a quasi-borel*, α, μ) = *p1*;
        (*Y*, β, ν) = *p2 in*
    qbs-meas X α μ $\wedge$ *qbs-meas Y* β ν $\wedge$ *X* = *Y* $\wedge$
      ($\forall f \in X \rightarrow_Q$ (*qbs-borel* :: *ennreal quasi-borel*). ($\int^+ x.$ *f* (α *x*) $\partial\mu$) = ($\int^+ x.$ *f* (β *x*) $\partial\nu$)))
  unfolding *qbs-meas-eq-def Let-def*
proof *safe*
  fix *X* :: *'a quasi-borel* **and** α μ β ν **and** *f* :: *'a* $\Rightarrow$ *ennreal*
  assume *qbs-meas X* α μ *qbs-meas X* β ν
    and *h*: *distr* μ (*qbs-to-measure X*) α = *distr* ν (*qbs-to-measure X*) β
    and *f* $\in$ *X* $\rightarrow_Q$ $borel_Q$
  then have [*measurable*]: *f* $\in$ *qbs-to-measure X* $\rightarrow_M$ *borel*
    by(*auto dest*: *qbs-morphism-dest*)
  interpret *qm1*: *qbs-meas X* α μ **by** *fact*
  interpret *qm2*: *qbs-meas X* β ν **by** *fact*
  show ($\int^+ x.$ *f* (α *x*) $\partial\mu$) = ($\int^+ x.$ *f* (β *x*) $\partial\nu$)
    by(*simp add*: *nn-integral-distr*[*symmetric*,*of* - - *qbs-to-measure X*] *h*)
next
  fix *X* :: *'a quasi-borel* **and** α μ β ν
  assume *qbs-meas X* α μ *qbs-meas X* β ν

and h: $\forall f \in X \to_Q borel_Q.\ (\int^+ x.\ f\ (\alpha\ x)\ \partial\mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\nu)$
interpret *qm1*: *qbs-meas* $X\ \alpha\ \mu$ **by** *fact*
interpret *qm2*: *qbs-meas* $X\ \beta\ \nu$ **by** *fact*
show *distr* μ (*qbs-to-measure* X) α = *distr* ν (*qbs-to-measure* X) β
proof(*rule measure-eqI*)
fix A
assume A[*measurable*]:$A \in$ *sets* (*distr* μ (*qbs-to-measure* X) α)
show *emeasure* (*distr* μ (*qbs-to-measure* X) α) A = *emeasure* (*distr* ν (*qbs-to-measure* X) β) A
using h[*rule-format*,*of indicator* A] A
by(*simp add*: *lr-adjunction-correspondence nn-integral-distr*[*symmetric*,*of* - - *qbs-to-measure* X])
qed *simp*
qed

lemma(**in** *qbs-meas*) *qbs-meas-eq-refl*[*simp*]: *qbs-meas-eq* (X,α,μ) (X,α,μ)
by(*simp-all add*: *qbs-meas-eq-def qbs-meas-axioms*)

lemma (**in** *pair-qbs-meas*)
shows *qbs-meas-eq-intro*:
distr μ (*qbs-to-measure* X) α = *distr* ν (*qbs-to-measure* X) $\beta \Longrightarrow$ *qbs-meas-eq* (X,α,μ) (X,β,ν)
and *qbs-meas-eq-intro2*:
$(\bigwedge f.\ f \in X \to_Q$ *qbs-borel* $\Longrightarrow (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\ \nu)) \Longrightarrow$ *qbs-meas-eq* (X,α,μ) (X,β,ν)
by(*simp add*: *pq1.qbs-meas-axioms pq2.qbs-meas-axioms qbs-meas-eq-def*)
(*simp add*: *pq1.qbs-meas-axioms pq2.qbs-meas-axioms qbs-meas-eq-def2*)

lemma *qbs-meas-eq-dest*:
assumes *qbs-meas-eq* (X,α,μ) (Y,β,ν)
shows *qbs-meas* $X\ \alpha\ \mu$ *qbs-meas* $Y\ \beta\ \nu$ $Y = X$ *distr* μ (*qbs-to-measure* X) α = *distr* ν (*qbs-to-measure* X) β
using *assms* **by**(*auto simp*: *qbs-meas-eq-def*)

lemma *qbs-meas-eq-dest2*:
assumes *qbs-meas-eq* (X,α,μ) (Y,β,ν)
shows *qbs-meas* $X\ \alpha\ \mu$ *qbs-meas* $Y\ \beta\ \nu$ $Y = X\ \bigwedge f.\ f \in X \to_Q$ *qbs-borel* $\Longrightarrow (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\ \nu)$
using *assms* **by**(*auto simp*: *qbs-meas-eq-def2*)

lemma *qbs-meas-eq-integral-eq*:
assumes *qbs-meas-eq* (X,α,μ) (Y,β,ν)
and [*measurable*]:$f \in X \to_Q$ (*qbs-borel* :: 'b::{*banach*, *second-countable-topology*} *quasi-borel*)
shows $(\int x.\ f\ (\alpha\ x)\ \partial\mu) = (\int x.\ f\ (\beta\ x)\ \partial\nu)$
proof −
interpret *pair-qbs-meas* $X\ \alpha\ \mu\ \beta\ \nu$
using *assms* **by**(*auto simp*: *qbs-meas-eq-def pair-qbs-meas-def*)
show *?thesis*

by(*simp add*: *integral-distr*[**where** *N*=*qbs-to-measure X*,*symmetric*] *qbs-meas-eq-dest*(*4*)[*OF assms*(*1*)])
qed

lemma
shows *qbs-meas-eq-symp*: *symp qbs-meas-eq*
and *qbs-meas-eq-transp*: *transp qbs-meas-eq*
by(*simp-all add*: *qbs-meas-eq-def transp-def symp-def*)

quotient-type *'a qbs-measure* = *'a qbs-measure-t* / *partial*: *qbs-meas-eq*
morphisms *rep-qbs-measure qbs-measure*
proof(*rule part-equivpI*)
let *?U* = *UNIV* :: *'a set*
let *?Uf* = *UNIV* :: (*real* ⇒ *'a*) *set*
let *?f* = (λ-. *undefined*) :: *real* ⇒ *'a*
have *qbs-meas* (*Abs-quasi-borel* (*?U*, *?Uf*)) *?f* (*return borel 0*)
unfolding *qbs-meas-def in-Mx-def qbs-meas-axioms-def*
proof *safe*
have *Rep-quasi-borel* (*Abs-quasi-borel* (*?U*,*?Uf*)) = (*?U*, *?Uf*)
using *Abs-quasi-borel-inverse* **by** (*auto simp add*: *qbs-closed1-def qbs-closed2-def qbs-closed3-def is-quasi-borel-def*)
thus (λ-. *undefined*) ∈ *qbs-Mx* (*Abs-quasi-borel* (*?U*, *?Uf*))
by(*simp add*: *qbs-Mx-def*)
qed *simp-all*
thus ∃ *x* :: *'a qbs-measure-t*. *qbs-meas-eq x x*
by(*auto simp*: *qbs-meas-eq-def intro*!: *exI*[**where** *x*=(*Abs-quasi-borel* (*?U*,*?Uf*), *?f*, *return borel 0*)])
qed(*simp-all add*: *qbs-meas-eq-symp qbs-meas-eq-transp*)

interpretation *qbs-measure* : *quot-type qbs-meas-eq Abs-qbs-measure Rep-qbs-measure*
using *Abs-qbs-measure-inverse Rep-qbs-measure*
by(*simp add*: *quot-type-def equivp-implies-part-equivp qbs-measure-equivp Rep-qbs-measure-inverse Rep-qbs-measure-inject*) *blast*

syntax
-qbs-measure :: *'a quasi-borel* ⇒ (*real* ⇒ *'a*) ⇒ *real measure* ⇒ *'a qbs-measure* (⟦-,/ -,/ -⟧$_{meas}$)
syntax-consts
-qbs-measure ⇌ *qbs-measure*
translations
⟦*X*, *α*, *μ*⟧$_{meas}$ ⇌ *CONST qbs-measure* (*X*, *α*, *μ*)

lemma *rep-qbs-measure'*: ∃ *X α μ*. *p* = ⟦*X*, *α*, *μ*⟧$_{meas}$ ∧ *qbs-meas X α μ*
by(*rule qbs-measure.abs-induct*) (*auto simp*: *qbs-meas-eq-def*)

lemma *rep-qbs-measure*:
obtains *X α μ* **where** *p* = ⟦*X*, *α*, *μ*⟧$_{meas}$ ∧ *qbs-meas X α μ*
using *that rep-qbs-measure'* **by** *blast*

definition *qbs-null-measure* :: *'a quasi-borel* ⇒ *'a qbs-measure* **where**
qbs-null-measure X ≡ ⟦*X*, *SOME a. a* ∈ *qbs-Mx X*, *null-measure borel*⟧$_{meas}$

lemma *qbs-null-measure-meas*: *qbs-space X* ≠ {} ⟹ *qbs-meas X* (*SOME a. a* ∈ *qbs-Mx X*) (*null-measure borel*)
and *qbs-null-measure-s-finite*: *qbs-space X* ≠ {} ⟹ *qbs-s-finite X* (*SOME a. a* ∈ *qbs-Mx X*) (*null-measure borel*)
by(*auto simp*: *qbs-empty-equiv qbs-s-finite-def in-Mx-def qbs-meas-def qbs-meas-axioms-def some-in-eq*
intro!: *finite-measure.s-finite-measure-finite-measure finite-measureI*)

lemma *in-Rep-qbs-measure'*:
assumes *qbs-meas-eq* (*X*,*α*,*μ*) (*X'*,*α'*,*μ'*)
shows (*X'*,*α'*,*μ'*) ∈ *Rep-qbs-measure* ⟦*X*, *α*, *μ*⟧$_{meas}$
by (*metis assms mem-Collect-eq qbs-meas.qbs-meas-eq-refl qbs-meas-eq-dest2*(*1*) *qbs-measure.abs-def qbs-measure.abs-inverse qbs-measure-def*)

lemmas(**in** *qbs-meas*) *in-Rep-qbs-measure* = *in-Rep-qbs-measure'*[*OF qbs-meas-eq-refl*]

lemma(**in** *qbs-meas*) *in-Rep-qbs-measure-dest*:
assumes (*X'*,*α'*,*μ'*) ∈ *Rep-qbs-measure* ⟦*X*, *α*, *μ*⟧$_{meas}$
shows *X'* = *X*
qbs-meas X' α' μ'
qbs-meas-eq (*X*,*α*,*μ*) (*X'*,*α'*,*μ'*)
proof −
show *h*:*X'* = *X*
by (*metis assms mem-Collect-eq qbs-meas-eq-dest*(*3*) *qbs-meas-eq-refl qbs-measure.abs-def qbs-measure.abs-inverse qbs-measure-def*)
next
show *qbs-meas X' α' μ'*
using *assms qbs-measure.Rep-qbs-measure*[*of* ⟦*X*, *α*, *μ*⟧$_{meas}$]
by(*auto simp*: *qbs-meas-eq-def*)
next
show *qbs-meas-eq* (*X*,*α*,*μ*) (*X'*,*α'*,*μ'*)
using *assms qbs-measure.Rep-qbs-measure*[*of* ⟦*X*, *α*, *μ*⟧$_{meas}$]
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq qbs-meas-eq-refl qbs-measure.abs-def qbs-measure.abs-inverse qbs-measure-def*)
qed

lemma(**in** *qbs-meas*) *in-Rep-qbs-measure-dest'*:
assumes *p* ∈ *Rep-qbs-measure* ⟦*X*, *α*, *μ*⟧$_{meas}$
obtains *α'* *μ'* **where** *p* = (*X*, *α'*, *μ'*) *qbs-meas X α' μ' qbs-meas-eq* (*X*,*α*,*μ*) (*X*,*α'*,*μ'*)
by (*metis prod-cases3 in-Rep-qbs-measure-dest assms*)

lemma *qbs-meas-eqI'*: *qbs-meas-eq* (*X*,*α*,*μ*) (*Y*,*β*,*ν*) ⟹ ⟦*X*, *α*, *μ*⟧$_{meas}$ = ⟦*Y*, *β*, *ν*⟧$_{meas}$
using *Quotient3-rel*[*OF Quotient3-qbs-measure*] **by** *blast*

lemma(**in** *pair-qbs-meas*) *qbs-meas-eqI*:
distr μ (*qbs-to-measure* X) α = *distr* ν (*qbs-to-measure* X) $\beta \Longrightarrow [\![X, \alpha, \mu]\!]_{meas}$ = $[\![X, \beta, \nu]\!]_{meas}$
by(*auto intro*!: *qbs-meas-eqI'* *qbs-meas-eq-intro*)

lemma(**in** *pair-qbs-meas*) *qbs-meas-eqI2*:
$(\bigwedge f.\ f \in X \to_Q$ *qbs-borel* $\Longrightarrow (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\ \nu)) \Longrightarrow$ $[\![X, \alpha, \mu]\!]_{meas} = [\![X, \beta, \nu]\!]_{meas}$
by(*auto intro*!: *qbs-meas-eqI'* *qbs-meas-eq-intro2*)

lemma(**in** *pair-qbs-s-finite*) *qbs-s-finite-measure-eq-inverse*:
assumes $[\![X, \alpha, \mu]\!]_{meas} = [\![X, \beta, \nu]\!]_{meas}$
shows *qbs-meas-eq* (X,α,μ) (X,β,ν)
using *Quotient3-rel*[*OF Quotient3-qbs-measure*,*of* (X,α,μ) (X,β,ν),*simplified*]
by(*simp-all add*: *assms*)

lift-definition *qbs-space-of* :: *'a qbs-measure* ⇒ *'a quasi-borel*
is *fst* **by**(*auto simp*: *qbs-meas-eq-def*)

lemma (**in** *qbs-meas*) *qbs-space-of*[*simp*]: *qbs-space-of* $[\![X, \alpha, \mu]\!]_{meas} = X$
by(*simp add*: *qbs-space-of.abs-eq*)

lemma *qbs-space-of-non-empty*: *qbs-space* (*qbs-space-of p*) ≠ {}
by (*metis in-Mx.X-not-empty qbs-meas.axioms*(*1*) *qbs-meas.qbs-space-of rep-qbs-measure*)

4.1.2 The Space of All Measures

definition *all-meas-qbs* :: *'a quasi-borel* ⇒ *'a qbs-measure quasi-borel* **where**
all-meas-qbs X ≡ *Abs-quasi-borel* ({*s. qbs-space-of s* = *X*}, {λ*r*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *measure-kernel borel borel k*})

lemma
shows *all-meas-qbs-space*: *qbs-space* (*all-meas-qbs X*) = {*s. qbs-space-of s* = *X*}
(**is** *?g1*)
and *all-meas-qbs-Mx*: *qbs-Mx* (*all-meas-qbs X*) = {λ*r*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *measure-kernel borel borel k*} (**is** *?g2*)
proof −
have {λ*r*::*real*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *measure-kernel borel borel k*} ⊆ *UNIV* → {*s. qbs-space-of s* = *X*}
by(*auto intro*!: *qbs-meas.qbs-space-of simp*: *qbs-meas-def in-Mx-def qbs-meas-axioms-def measure-kernel.kernel-sets*)
moreover have *qbs-closed1* {λ*r*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *measure-kernel borel borel k*}
proof(*safe intro*!: *qbs-closed1I*)
fix α **and** *f* :: *real* ⇒ *real* **and** *k* :: *real*⇒ *real measure*
assume *h*:*f* ∈ *borel-measurable borel* α ∈ *qbs-Mx X measure-kernel borel borel k*
then show ∃α′ *ka*. (λ*r*. $[\![X, \alpha, k\ r]\!]_{meas}$) ∘ *f* = (λ*r*. $[\![X, \alpha', ka\ r]\!]_{meas}$) ∧ α′

$\in$ *qbs-Mx X* $\wedge$ *measure-kernel borel borel ka*
by(*auto intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=$\lambda x.\ k\ (f\ x)$] *simp*: *measure-kernel.measure-kernel-comp*[*OF h*(*3*,*1*)])
qed
moreover have *qbs-closed2* $\{s.\ qbs\text{-}space\text{-}of\ s = X\}$ $\{\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas}\ |\alpha\ k.\ \alpha \in qbs\text{-}Mx\ X \wedge measure\text{-}kernel\ borel\ borel\ k\}$
proof(*safe intro*!: *qbs-closed2I*)
fix *p*
assume *h*:*X* = *qbs-space-of p*
obtain *X'* α μ **where** *p*:*p* = $[\![X',\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas X'* α μ
using *rep-qbs-measure* **by** *meson*
then interpret *qbs-meas X'* α μ
by *simp*
show $\exists \alpha\ k.\ (\lambda r.\ p) = (\lambda r.\ [\![qbs\text{-}space\text{-}of\ p,\ \alpha,\ k\ r]\!]_{meas}) \wedge \alpha \in qbs\text{-}Mx\ (qbs\text{-}space\text{-}of\ p) \wedge$
measure-kernel borel borel k
using *p* **by**(*auto simp*: *qbs-meas-axioms-def qbs-meas-def h*
intro!: *exI*[**where** *x*=α] *exI*[**where** *x*=$\lambda r.\ \mu$] *measure-kernel-const'*)
qed
moreover have *qbs-closed3* $\{\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas}\ |\alpha\ k.\ \alpha \in qbs\text{-}Mx\ X \wedge measure\text{-}kernel\ borel\ borel\ k\}$
proof(*rule qbs-closed3I*)
fix *P* :: *real* $\Rightarrow$ *nat* **and** *Fi*
assume *P*[*measurable*]:*P* $\in$ *borel* $\rightarrow_M$ *count-space UNIV*
and $\bigwedge i$::*nat*. *Fi i* $\in$ $\{\lambda r$::*real*. $[\![X,\ \alpha,\ k\ r]\!]_{meas}\ |\alpha\ k.\ \alpha \in qbs\text{-}Mx\ X \wedge measure\text{-}kernel\ borel\ borel\ k\}$
then obtain αi *ki* **where** *Fi*: $\bigwedge i.\ Fi\ i = (\lambda r.\ [\![X,\ \alpha i\ i,\ ki\ i\ r]\!]_{meas})$ $\bigwedge i.\ \alpha i\ i \in qbs\text{-}Mx\ X$
$\bigwedge i.\ measure\text{-}kernel\ borel\ borel\ (ki\ i)$
by *auto metis*
interpret *nat-real*: *standard-borel-ne count-space* (*UNIV* :: *nat set*) $\bigotimes_M$ (*borel* :: *real measure*)
by(*auto intro*!: *pair-standard-borel-ne*)
note [*simp*] = *nat-real.from-real-to-real*[*simplified space-pair-measure*, *simplified*]
define α **where** $\alpha \equiv (\lambda r.\ case\text{-}prod\ \alpha i\ (nat\text{-}real.from\text{-}real\ r))$
define *k* **where** $k \equiv (\lambda r.\ distr\ (distr\ (ki\ (P\ r)\ r)\ (count\text{-}space\ UNIV \bigotimes_M borel)\ (\lambda r'.\ (P\ r,\ r'))))\ borel\ nat\text{-}real.to\text{-}real)$
have α: $\alpha \in qbs\text{-}Mx\ X$
unfolding α*-def qbs-Mx-is-morphisms*
proof(*rule qbs-morphism-compose*[**where** *g*=*nat-real.from-real* **and** *Y*=*qbs-count-space UNIV* $\bigotimes_Q$ *qbs-borel*])
show *nat-real.from-real* $\in$ *qbs-borel* $\rightarrow_Q$ *qbs-count-space UNIV* $\bigotimes_Q$ *qbs-borel*
by(*simp add*: *r-preserves-product*[*symmetric*] *standard-borel.standard-borel-r-full-faithful*[*of borel* :: *real measure*,*simplified*,*symmetric*] *standard-borel-ne.standard-borel*)
next
show *case-prod* $\alpha i \in$ *qbs-count-space UNIV* $\bigotimes_Q$ *qbs-borel* $\rightarrow_Q$ *X*
using *Fi*(*2*) **by**(*auto intro*!: *qbs-morphism-pair-count-space1 simp*: *qbs-Mx-is-morphisms*)
qed
have *sets-ki*[*measurable-cong*]: *sets* (*ki i r*) = *sets borel* *sets* (*k r*) = *sets borel*

for *i r*
using *Fi(3)* **by**(*auto simp*: *s-finite-kernel-def measure-kernel-def k-def*)
interpret *k*: *measure-kernel borel borel k*
proof −
have *1*:*k* = (λ(*i*,*r*). *distr* (*ki i r*) *borel* (λ*r*′. *nat-real.to-real* (*i*, *r*′))) ∘ (λ*r*. (*P r*, *r*))
by *standard* (*auto simp*: *k-def distr-distr comp-def*)
have *measure-kernel borel borel* ...
unfolding *comp-def*
by(*rule measure-kernel.measure-kernel-comp*[**where** *X*=*count-space UNIV* ⨂$_M$ *borel*])
(*auto intro*!: *measure-kernel-pair-countble1 measure-kernel.distr-measure-kernel*[*OF Fi(3)*])
thus *measure-kernel borel borel k*
by(*simp add*: *1*)
qed
have (λ*r*. *Fi* (*P r*) *r*) = (λ*r*. ⟦*X*, α, *k r* ⟧$_{meas}$)
unfolding *Fi(1)*
proof
fix *r*
interpret *pq*: *pair-qbs-meas X* α*i* (*P r*) *ki* (*P r*) *r* α *k r*
by(*auto simp*: *pair-qbs-meas-def qbs-meas-def qbs-meas-axioms-def in-Mx-def sets-ki* α *Fi(2)*)
show ⟦*X*, α*i* (*P r*), *ki* (*P r*) *r*⟧$_{meas}$ = ⟦*X*, α, *k r*⟧$_{meas}$
by(*intro pq.qbs-meas-eqI*) (*simp add*: *k-def distr-distr*, *simp add*: *comp-def* α*-def*)
qed
thus (λ*r*. *Fi* (*P r*) *r*) ∈ {λ*r*. ⟦*X*, α, *k r*⟧$_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *measure-kernel borel borel k*}
by(*auto intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=*k*] *simp*: α *k.measure-kernel-axioms*)
qed
ultimately have *Rep-quasi-borel* (*all-meas-qbs X*)
= ({*s*. *qbs-space-of s* = *X*}, {λ*r*. ⟦*X*, α, *k r*⟧$_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *measure-kernel borel borel k*})
by(*auto intro*!: *Abs-quasi-borel-inverse simp*: *all-meas-qbs-def is-quasi-borel-def*)
thus *?g1 ?g2*
by(*simp-all add*: *all-meas-qbs-def qbs-space-def qbs-Mx-def*)
qed

lemma *all-meas-qbs-empty-iff*: *qbs-space X* = {} ⟷ *qbs-space* (*all-meas-qbs X*) = {}
by (*metis* (*mono-tags*, *lifting*) *qbs-space-of-non-empty all-meas-qbs-space empty-Collect-eq qbs-meas.qbs-space-of qbs-null-measure-meas*)

lemma(**in** *qbs-meas*) *in-space-all-meas*[*qbs*]: ⟦*X*, α, μ⟧$_{meas}$ ∈ *qbs-space* (*all-meas-qbs X*)
by(*simp add*: *all-meas-qbs-space*)

lemma *rep-qbs-space-all-meas*:

assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
obtains α μ **where** $s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-meas* X α μ
by (*metis* (*mono-tags*, *lifting*) *all-meas-qbs-space assms mem-Collect-eq qbs-meas.qbs-space-of rep-qbs-measure'*)

lemma *qbs-space-of-in-all-meas*: $s \in$ *qbs-space* (*all-meas-qbs* X) $\Longrightarrow$ *qbs-space-of* $s = X$
by(*simp add*: *all-meas-qbs-space*)

lemma *in-qbs-space-of-all-meas*: $s \in$ *qbs-space* (*all-meas-qbs* (*qbs-space-of* s))
by(*simp add*: *all-meas-qbs-space*)

4.1.3 *l*

lift-definition *qbs-l* :: *'a qbs-measure* $\Rightarrow$ *'a measure*
is $\lambda(X,\alpha,\mu)$. *distr* μ (*qbs-to-measure* X) α
by(*auto simp*: *qbs-meas-eq-def*)

lemma(**in** *qbs-meas*) *qbs-l*: *qbs-l* $[\![X, \alpha, \mu]\!]_{meas}$ = *distr* μ (*qbs-to-measure* X) α
by(*simp add*: *qbs-l.abs-eq*)

lemma *space-qbs-l*: *qbs-space* (*qbs-space-of* s) = *space* (*qbs-l* s)
by(*transfer*, *auto simp*: *space-L*)

lemma *space-qbs-l-ne*: *space* (*qbs-l* s) $\neq \{\}$
by *transfer* (*auto simp*: *qbs-meas-eq-def qbs-meas-def in-Mx-def space-L qbs-empty-equiv*)

lemma *qbs-l-sets*: *sets* (*qbs-to-measure* (*qbs-space-of* s)) = *sets* (*qbs-l* s)
by *transfer auto*

lemma *qbs-null-measure-in-all-meas*: *qbs-space* $X \neq \{\} \Longrightarrow$ *qbs-null-measure* $X \in$ *qbs-space* (*all-meas-qbs* X)
by(*simp add*: *qbs-meas.in-space-all-meas qbs-null-measure-def qbs-null-measure-meas*)

lemma *qbs-null-measure-null-measure*:*qbs-space* $X \neq \{\} \Longrightarrow$ *qbs-l* (*qbs-null-measure* X) = *null-measure* (*qbs-to-measure* X)
by(*auto simp*: *qbs-null-measure-def qbs-meas.qbs-l*[*OF qbs-null-measure-meas*] *null-measure-distr*)

lemma *space-qbs-l-in-all-meas*:
assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
shows *space* (*qbs-l* s) = *qbs-space* X
by (*metis assms qbs-meas.qbs-space-of rep-qbs-space-all-meas space-qbs-l*)

lemma *sets-qbs-l-all-meaures*:
assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
shows *sets* (*qbs-l* s) = *sets* (*qbs-to-measure* X)
using *assms qbs-l-sets qbs-space-of-in-all-meas* **by** *blast*

lemma *measurable-qbs-l-all-meas*:

assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
shows *qbs-l* $s \to_M M = X \to_Q$ *measure-to-qbs* M
by(*auto simp*: *measurable-cong-sets*[*OF qbs-l-sets*[*of s*,*simplified qbs-space-of-in-all-meas*[*OF assms*(*1*)],*symmetric*] *refl*]
lr-adjunction-correspondence)

lemma *measurable-qbs-l-all-meas'*:
assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
shows *qbs-l* $s \to_M M =$ *qbs-to-measure* $X \to_M M$
by(*simp add*: *measurable-qbs-l-all-meas*[*OF assms*] *lr-adjunction-correspondence*)

lemma *rep-all-meas-qbs-Mx*:
assumes $\gamma \in$ *qbs-Mx* (*all-meas-qbs* X)
obtains α k **where** $\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})$ $\alpha \in$ *qbs-Mx* X *measure-kernel borel borel* k $\bigwedge r.$ *qbs-meas* X α (k r)
proof −
obtain α k **where** $\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})$ $\alpha \in$ *qbs-Mx* X *measure-kernel borel borel* k
using *assms* **by**(*auto simp*: *all-meas-qbs-Mx*)
moreover hence $\bigwedge r.$ *qbs-meas* X α (k r)
by (*simp add*: *qbs-meas-All*)
ultimately show *?thesis*
using *that* **by** *blast*
qed

lemma *qbs-l-measure-kernel-all-meas*:
measure-kernel (*qbs-to-measure* (*all-meas-qbs* X)) (*qbs-to-measure* X) *qbs-l*
proof(*cases qbs-space* $X = \{\}$)
case *True*
with *all-meas-qbs-empty-iff*[*of* X,*simplified this*] **show** *?thesis*
by(*auto intro*!: *measure-kernel-empty-trivial simp*: *space-L*)
next
case *1*:*False*
show *?thesis*
proof
show $\bigwedge x.\ x \in$ *space* (*qbs-to-measure* (*all-meas-qbs* X)) $\Longrightarrow$ *sets* (*qbs-l* x) = *sets* (*qbs-to-measure* X)
using *qbs-l-sets* **by**(*auto simp*: *space-L all-meas-qbs-space*)
next
show *space* (*qbs-to-measure* X) $\neq \{\}$
by(*simp add*: *space-L 1*)
next
fix B
assume [*measurable*]:$B \in$ *sets* (*qbs-to-measure* X)
show ($\lambda x.$ *emeasure* (*qbs-l* x) B) $\in$ *borel-measurable* (*qbs-to-measure* (*all-meas-qbs* X))
proof(*rule qbs-morphism-dest*[*OF qbs-morphismI*])
fix γ
assume $\gamma \in$ *qbs-Mx* (*all-meas-qbs* X)

from *rep-all-meas-qbs-Mx*[*OF this*] **obtain** α k **where**
$\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})\ \alpha \in$ *qbs-Mx X measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas X* α (*k r*)
by *blast*
then show ($\lambda x.$ *emeasure* (*qbs-l x*) *B*) $\circ\ \gamma \in$ *qbs-Mx borel*$_Q$
by(*auto simp*: *comp-def qbs-meas.qbs-l qbs-Mx-R*
intro!: *measure-kernel.distr-measure-kernel measure-kernel.emeasure-measurable*)
qed
qed
qed

lemma *qbs-l-inj-all-meas*: *inj-on qbs-l* (*qbs-space* (*all-meas-qbs X*))
by *standard* (*simp add*:*all-meas-qbs-space*, *transfer*,*auto simp*: *qbs-meas-eq-def*)

lemma *qbs-l-morphism-all-meas*:
assumes [*measurable*]:$A \in$ *sets* (*qbs-to-measure X*)
shows ($\lambda s.$ *qbs-l s A*) $\in$ *all-meas-qbs X* $\rightarrow_Q$ *qbs-borel*
proof(*rule qbs-morphismI*)
fix γ
assume *h*:$\gamma \in$ *qbs-Mx* (*all-meas-qbs X*)
hence [*qbs*]: $\gamma \in$ *qbs-borel* $\rightarrow_Q$ *all-meas-qbs X*
by(*simp-all add*: *qbs-Mx-is-morphisms*)
from *rep-all-meas-qbs-Mx*[*OF h*(*1*)] **obtain** α k **where** *hk*:
$\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})\ \alpha \in$ *qbs-Mx X measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas X* α (*k r*)
by *metis*
then interpret *a*: *in-Mx X* α **by**(*simp add*: *in-Mx-def*)
have *k*[*measurable-cong*]:*sets* (*k r*) = *sets borel* **for** *r*
using *hk*(*3*) **by**(*auto simp*: *measure-kernel-def*)
show ($\lambda s.$ *emeasure* (*qbs-l s*) *A*) $\circ\ \gamma \in$ *qbs-Mx qbs-borel*
by(*auto simp*: *hk*(*1*) *qbs-meas.qbs-l*[*OF hk*(*4*)] *comp-def qbs-Mx-qbs-borel emeasure-distr sets-eq-imp-space-eq*[*OF k*]
intro!: *measurable-sets-borel*[*OF - assms*] *measure-kernel.emeasure-measurable*[*OF hk*(*3*)])
qed

lemma *qbs-l-finite-pred-all-meas*: *qbs-pred* (*all-meas-qbs X*) ($\lambda s.$ *finite-measure* (*qbs-l s*))
proof −
have *qbs-space X* $\in$ *sets* (*qbs-to-measure X*)
by (*metis sets.top space-L*)
note *qbs-l-morphism-all-meas*[*OF this*,*qbs*]
have [*simp*]:*finite-measure* (*qbs-l s*) $\longleftrightarrow$ *qbs-l s X* $\neq \infty$ **if** $s \in$ *all-meas-qbs X* **for** *s*
by(*auto intro*!: *finite-measureI dest*: *finite-measure.emeasure-finite simp*: *space-qbs-l-in-all-meas*[*OF that*])
show *?thesis*
by(*simp cong*: *qbs-morphism-cong*)
qed

lemma *qbs-l-subprob-pred-all-meas*: *qbs-pred* (*all-meas-qbs X*) (*λs. subprob-space* (*qbs-l s*))
proof −
have *qbs-space X ∈ sets* (*qbs-to-measure X*)
by (*metis sets.top space-L*)
note *qbs-l-morphism-all-meas*[*OF this*,*qbs*]
have [*simp*]:*subprob-space* (*qbs-l s*) ⟷ *qbs-l s X ≤ 1* **if** *s ∈ all-meas-qbs X* **for** *s*
by(*auto intro*!: *subprob-spaceI dest*: *subprob-space.subprob-emeasure-le-1 simp*: *space-qbs-l-ne*)
(*simp add*: *space-qbs-l-in-all-meas*[*OF that*])
show *?thesis*
by(*simp cong*: *qbs-morphism-cong*)
qed

lemma *qbs-l-prob-pred-all-meas*: *qbs-pred* (*all-meas-qbs X*) (*λs. prob-space* (*qbs-l s*))
proof −
have *qbs-space X ∈ sets* (*qbs-to-measure X*)
by (*metis sets.top space-L*)
note *qbs-l-morphism-all-meas*[*OF this*,*qbs*]
have [*simp*]:*prob-space* (*qbs-l s*) ⟷ *qbs-l s X = 1* **if** *s ∈ all-meas-qbs X* **for** *s*
by(*auto intro*!: *prob-spaceI simp*: *space-qbs-l-ne*)
(*auto simp add*: *space-qbs-l-in-all-meas*[*OF that*] *dest*: *prob-space.emeasure-space-1*)
show *?thesis*
by(*simp cong*: *qbs-morphism-cong*)
qed

4.1.4 Return

definition *return-qbs* :: *'a quasi-borel ⇒ 'a ⇒ 'a qbs-measure* **where**
return-qbs X x ≡ ⟦*X*, *λr. x*, *SOME μ. real-distribution μ*⟧$_{meas}$

lemma(**in** *real-distribution*)
assumes *x ∈ qbs-space X*
shows *return-qbs*:*return-qbs X x* = ⟦*X*, *λr. x*, *M*⟧$_{meas}$
and *return-qbs-meas*:*qbs-meas X* (*λr. x*) *M*
and *return-qbs-prob*:*qbs-prob X* (*λr. x*) *M*
and *return-qbs-s-finite*:*qbs-s-finite X* (*λr. x*) *M*
proof −
interpret *qs1*: *qbs-prob X λr. x M*
by(*auto simp*: *qbs-prob-def in-Mx-def real-distribution-axioms intro*!: *qbs-closed2-dest assms*)
show *return-qbs X x* = ⟦*X*, *λr. x*, *M*⟧$_{meas}$
unfolding *return-qbs-def*
proof(*rule someI2*)
show *real-distribution* (*return borel 0*)
by (*auto simp*: *real-distribution-def real-distribution-axioms-def prob-space-return*)

next
fix N
assume *real-distribution N*
then interpret *qs2*: *qbs-meas X* $\lambda r.\ x\ N$
by(*auto simp*: *qbs-meas-def in-Mx-def qbs-meas-axioms-def real-distribution-def real-distribution-axioms-def*)
interpret *pair-qbs-meas X* $\lambda r.\ x\ N\ \lambda r.\ x\ M$
by *standard*
show $[\![X,\ \lambda r.\ x,\ N]\!]_{meas} = [\![X,\ \lambda r.\ x,\ M]\!]_{meas}$
by(*auto intro*!: *qbs-meas-eqI measure-eqI simp*: *emeasure-distr*)
(*metis* ‹*real-distribution N*› *emeasure-space-1 prob-space.emeasure-space-1 qs2.mu-sets real-distribution.axioms*(*1*) *sets-eq-imp-space-eq space-borel space-eq-univ*)
qed
show *qbs-prob X* $(\lambda r.\ x)$ *M qbs-s-finite X* $(\lambda r.\ x)$ *M qbs-meas X* $(\lambda r.\ x)$ *M*
by(*simp-all add*: *qs1.qbs-prob-axioms qs1.qbs-s-finite-axioms qs1.qbs-meas-axioms*)
qed

lemma *return-qbs-comp*:
assumes $\alpha \in$ *qbs-Mx X*
shows (*return-qbs X* $\circ\ \alpha$) $= (\lambda r.\ [\![X,\ \alpha,\ return\ borel\ r]\!]_{meas})$
proof
fix r
interpret *pqp*: *pair-qbs-prob X* $\lambda k.\ \alpha\ r$ *return borel 0* α *return borel r*
by(*simp add*: *assms qbs-Mx-to-X*[*OF assms*] *pair-qbs-prob-def qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def prob-space-return*)
show (*return-qbs X* $\circ\ \alpha$) $r = [\![X,\ \alpha,\ return\ borel\ r]\!]_{meas}$
by(*auto simp*: *pqp.pq1.return-qbs*[*OF qbs-Mx-to-X*[*OF assms*]] *distr-return intro*!: *pqp.qbs-meas-eqI*)
qed

corollary *return-qbs-morphism-all-meas*: *return-qbs X* $\in X \to_Q$ *all-meas-qbs X*
proof(*rule qbs-morphismI*)
interpret *rr* : *real-distribution return borel 0*
by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-return*)
fix α
assume *h*:$\alpha \in$ *qbs-Mx X*
then have *1*:*return-qbs X* $\circ\ \alpha = (\lambda r.\ [\![X,\ \alpha,\ return\ borel\ r]\!]_{meas})$
by(*rule return-qbs-comp*)
show *return-qbs X* $\circ\ \alpha \in$ *qbs-Mx* (*all-meas-qbs X*)
by(*auto simp*: *1 all-meas-qbs-Mx h prob-kernel-def'*
intro!: *exI*[**where** $x{=}\alpha$] *exI*[**where** *x*=*return borel*] *prob-kernel.axioms*(*1*))
qed

4.1.5 Bind

definition *bind-qbs* :: [$'a$ *qbs-measure*, $'a \Rightarrow 'b$ *qbs-measure*] $\Rightarrow$ $'b$ *qbs-measure*
where
bind-qbs s f $\equiv$ (*let* $(X,\ \alpha,\ \mu)$ = *rep-qbs-measure s*;

Y = *qbs-space-of* (f (α *undefined*));
(β, k) = (*SOME* (β, k). $f \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas}) \wedge \beta \in$ *qbs-Mx* $Y \wedge$ *measure-kernel borel borel* k) *in*
$[\![Y,\ \beta,\ \mu \gg\!=_k\ k]\!]_{meas}$)

adhoc-overloading *Monad-Syntax.bind* $\rightleftharpoons$ *bind-qbs*

lemma(**in** *qbs-meas*)
assumes $s = [\![X,\ \alpha,\ \mu]\!]_{meas}$
$f \in X \to_Q$ *all-meas-qbs* Y
$\beta \in$ *qbs-Mx* Y
measure-kernel borel borel k
and $(f \circ \alpha) = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$
shows *bind-qbs-meas*:*qbs-meas* $Y\ \beta\ (\mu \gg\!=_k\ k)$
and *bind-qbs-all-meas*: $s \gg\!= f = [\![Y,\ \beta,\ \mu \gg\!=_k\ k]\!]_{meas}$
proof −
interpret k: *measure-kernel borel borel* k **by** *fact*
interpret *qm*: *qbs-meas* $Y\ \beta\ \mu \gg\!=_k\ k$
by(*auto simp*: *qbs-meas-def qbs-meas-axioms-def in-Mx-def assms*(*3*) *mu-sets k.sets-bind-kernel*[*OF - mu-sets*])
{
fix $X'\ \alpha'\ \mu'$
assume (X',α',μ') ∈ *Rep-qbs-measure* $[\![X,\ \alpha,\ \mu]\!]_{meas}$
then have *h*: $X' = X$ *qbs-meas* $X'\ \alpha'\ \mu'$ *qbs-meas-eq* (X,α,μ) (X',α',μ')
by(*simp-all add*: *in-Rep-qbs-measure-dest*)
then interpret *pq1*: *pair-qbs-meas* $X\ \alpha\ \mu\ \alpha'\ \mu'$
by(*auto simp*: *pair-qbs-meas-def qbs-meas-axioms*)
have [*simp*]: *qbs-space-of* (f (α' r)) = Y **for** r
using *qbs-Mx-to-X*[*OF qbs-morphism-Mx*[*OF assms*(*2*) *pq1.pq2.in-Mx*],*of r*]
by(*auto simp*: *all-meas-qbs-space*)
have (*let* Y = *qbs-space-of* (f (α' *undefined*))
in case SOME (β, k). ($\lambda r.\ f\ (\alpha'\ r)) = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas}) \wedge \beta \in$ *qbs-Mx* $Y \wedge$ *measure-kernel borel borel* k *of*
(β, k) ⇒ $[\![Y,\ \beta,\ \mu' \gg\!=_k\ k]\!]_{meas}) = [\![Y,\ \beta,\ \mu \gg\!=_k\ k]\!]_{meas}$
proof −
have (*case SOME* (β, k). ($\lambda r.\ f\ (\alpha'\ r)) = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas}) \wedge \beta \in$ *qbs-Mx* $Y \wedge$ *measure-kernel borel borel* k *of* (β, k) ⇒ $[\![Y,\ \beta,\ \mu' \gg\!=_k\ k]\!]_{meas}) = [\![Y,\ \beta,\ \mu \gg\!=_k\ k]\!]_{meas}$
proof(*rule someI2-ex*)
show ∃ *a*. *case a of* (β, k) ⇒ $(\lambda r.\ f\ (\alpha'\ r)) = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas}) \wedge \beta \in$ *qbs-Mx* $Y \wedge$ *measure-kernel borel borel* k
using *qbs-morphism-Mx*[*OF assms*(*2*) *pq1.pq2.in-Mx*]
by(*auto simp*: *comp-def all-meas-qbs-Mx*)
next
show ⋀*x*. (*case x of* (β, k) ⇒ $(\lambda r.\ f\ (\alpha'\ r)) = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas}) \wedge \beta \in$ *qbs-Mx* $Y \wedge$ *measure-kernel borel borel* k) ⟹ (*case x of* (β, k) ⇒ $[\![Y,\ \beta,\ \mu' \gg\!=_k\ k]\!]_{meas}) = [\![Y,\ \beta,\ \mu \gg\!=_k\ k]\!]_{meas}$
proof *safe*
fix $\beta'\ k'$

assume h':$(\lambda r.\ f\ (\alpha'\ r)) = (\lambda r.\ [\![Y,\ \beta',\ k'\ r]\!]_{meas})\ \beta' \in$ *qbs-Mx Y measure-kernel borel borel* k'

interpret k': *measure-kernel borel borel* k' **by** *fact*

have *qbs-meas Y* β' $(\mu' \gg\!=_k k')$

by(*auto simp*: *qbs-meas-def in-Mx-def* $h'(2)$ *qbs-meas-axioms-def* k'*.sets-bind-kernel*[*OF - pq1.pq2.mu-sets*])

then interpret *pq2*: *pair-qbs-meas Y* β' $\mu' \gg\!=_k k'$ β $\mu \gg\!=_k k$

by(*auto simp*: *pair-qbs-meas-def qm.qbs-meas-axioms*)

show $[\![Y,\ \beta',\ \mu' \gg\!=_k k']\!]_{meas} = [\![Y,\ \beta,\ \mu \gg\!=_k k]\!]_{meas}$

proof(*rule pq2.qbs-meas-eqI*)

have *distr* $(\mu' \gg\!=_k k')$ (*qbs-to-measure Y*) $\beta' = \mu' \gg\!=_k (\lambda r.\ distr\ (k'\ r)$ (*qbs-to-measure Y*) $\beta')$

by(*simp add*: k'*.distr-bind-kernel*[*OF - pq1.pq2.mu-sets*])

also have ... $= \mu' \gg\!=_k (\lambda r.\ qbs\text{-}l\ [\![Y,\ \beta',\ k'\ r]\!]_{meas})$

by(*auto intro*!: *bind-kernel-cong-All qbs-meas.qbs-l*[*symmetric*] *qbs-meas-All*[**where** k=k' **and** *M=borel*] h')

also have ... $= \mu' \gg\!=_k (\lambda r.\ qbs\text{-}l\ (f\ (\alpha'\ r)))$

by(*auto simp*: *fun-cong*[*OF* $h'(1)$])

also have ... $=$ *distr* μ' (*qbs-to-measure X*) $\alpha' \gg\!=_k (\lambda x.\ qbs\text{-}l\ (f\ x))$

by(*simp add*: *measure-kernel.bind-kernel-distr*[*OF measure-kernel.measure-kernel-comp*[*OF qbs-l-measure-kernel-all-meas set-mp*[*OF l-preserves-morphisms assms*(2)]]] *sets-eq-imp-space-eq*[*OF pq1.pq2.mu-sets*])

also have ... $=$ *distr* μ (*qbs-to-measure X*) $\alpha \gg\!=_k (\lambda x.\ qbs\text{-}l\ (f\ x))$

by(*simp add*: *qbs-meas-eq-dest*(4)[*OF* $h(3)$])

also have ... $= \mu \gg\!=_k (\lambda r.\ qbs\text{-}l\ (f\ (\alpha\ r)))$

by(*simp add*: *measure-kernel.bind-kernel-distr*[*OF measure-kernel.measure-kernel-comp*[*OF qbs-l-measure-kernel-all-meas set-mp*[*OF l-preserves-morphisms assms*(2)]]] *sets-eq-imp-space-eq*[*OF mu-sets*])

also have ... $= \mu \gg\!=_k (\lambda r.\ qbs\text{-}l\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$

by(*simp add*: *fun-cong*[*OF assms*(5),*simplified comp-def*])

also have ... $= \mu \gg\!=_k (\lambda r.\ distr\ (k\ r)$ (*qbs-to-measure Y*) $\beta)$

by(*auto intro*!: *bind-kernel-cong-All qbs-meas.qbs-l qbs-meas-All*[**where** k=k **and** *M=borel*] *k.measure-kernel-axioms*)

also have ... $=$ *distr* $(\mu \gg\!=_k k)$ (*qbs-to-measure Y*) β

by(*simp add*: *k.distr-bind-kernel*[*OF - mu-sets*])

finally show *distr* $(\mu' \gg\!=_k k')$ (*qbs-to-measure Y*) β' = *distr* $(\mu \gg\!=_k k)$ (*qbs-to-measure Y*) β **.**

qed

qed

qed

thus *?thesis* **by** *simp*

qed

}

note $* =$ *this*

show $s \gg\!= f = [\![Y,\ \beta,\ \mu \gg\!=_k k]\!]_{meas}$

unfolding *bind-qbs-def rep-qbs-measure-def qbs-measure.rep-def assms*(1)

apply(*rule someI2*)

apply(*rule in-Rep-qbs-measure*)

using $*$ **by** *auto*

show *qbs-meas Y β* (*μ* ⤜$_k$ *k*)
by (*rule qm.qbs-meas-axioms*)
qed

lemma *bind-qbs-morphism-all-meas′*:
assumes *f* ∈ *X* →$_Q$ *all-meas-qbs Y*
shows (λ*x*. *x* ⤜ *f*) ∈ *all-meas-qbs X* →$_Q$ *all-meas-qbs Y*
proof(*rule qbs-morphismI*)
fix *γ*
assume *γ* ∈ *qbs-Mx* (*all-meas-qbs X*)
from *rep-all-meas-qbs-Mx*[*OF this*] **obtain** *α k* **where** *h*:
γ = (λ*r*. ⟦*X*, *α*, *k r*⟧$_{meas}$) *α* ∈ *qbs-Mx X measure-kernel borel borel k* ⋀*r*. *qbs-meas X α* (*k r*)
by *metis*
from *rep-all-meas-qbs-Mx*[*OF qbs-morphism-Mx*[*OF assms this*(*2*)]] **obtain** *α′ k′* **where** *h′*:
f ∘ *α* = (λ*r*. ⟦*Y*, *α′*, *k′ r*⟧$_{meas}$) *α′* ∈ *qbs-Mx Y measure-kernel borel borel k′* ⋀*r*. *qbs-meas Y α′* (*k′ r*)
by *metis*
have [*simp*]:(λ*x*. *x* ⤜ *f*) ∘ *γ* = (λ*r*. ⟦*Y*, *α′*, *k r* ⤜$_k$ *k′*⟧$_{meas}$)
by *standard* (*simp add*: *h*(*1*) *qbs-meas.bind-qbs-all-meas*[*OF h*(*4*) - *assms h′*(*2*,*3*,*1*)])
show (λ*x*. *x* ⤜ *f*) ∘ *γ* ∈ *qbs-Mx* (*all-meas-qbs Y*)
using *h′*(*2*) **by**(*auto simp*: *all-meas-qbs-Mx measure-kernel.bind-kernel-measure-kernel*[*OF h*(*3*) *h′*(*3*)]
intro!: *exI*[**where** *x*=*α′*])
qed

lemma *bind-qbs-return-all-meas′*:
assumes *x* ∈ *qbs-space* (*all-meas-qbs X*)
shows *x* ⤜ *return-qbs X* = *x*
proof −
obtain *α μ* **where** *h*:*qbs-meas X α μ x* = ⟦*X*, *α*, *μ*⟧$_{meas}$
using *rep-qbs-space-all-meas*[*OF assms*] **by** *blast*
then interpret *qs*: *qbs-meas X α μ* **by** *simp*
note *return-qbs-morphism-all-meas*[*qbs*]
interpret *prob-kernel borel borel return borel*
by(*simp add*: *prob-kernel-def′*)
show *?thesis*
by(*simp add*: *qs.bind-qbs-all-meas*[*OF h*(*2*) *return-qbs-morphism-all-meas - prob-kernel.axioms*(*1*) *return-qbs-comp*]
prob-kernel-axioms bind-kernel-return′′[*OF qs.mu-sets*] *h*(*2*)[*symmetric*])
qed

lemma *bind-qbs-return-all-meas*:
assumes *f* ∈ *X* →$_Q$ *all-meas-qbs Y*
and *x* ∈ *qbs-space X*
shows *return-qbs X x* ⤜ *f* = *f x*

proof −
from *rep-qbs-space-all-meas*[*OF qbs-morphism-space*[*OF assms*]] **obtain** α μ **where** *h*:
$f\ x = [\![Y,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas Y* α μ **by** *auto*
then interpret *qs*:*qbs-meas Y* α μ **by** *simp*
interpret *sk*: *measure-kernel borel borel* $\lambda r.\ \mu$
by(*auto intro*!: *measure-kernel-const*′ *simp*: *qs.mu-sets*)
interpret *rd*: *real-distribution return borel 0*
by(*simp add*: *real-distribution-def prob-space-return real-distribution-axioms-def*)
interpret *qbs-prob X* $\lambda r.$ *x return borel 0*
by(*rule rd.return-qbs-prob*[*OF assms*(*2*)])
show *?thesis*
using *bind-qbs-all-meas*[*OF rd.return-qbs*[*OF assms*(*2*)] *assms*(*1*) *qs.in-Mx sk.measure-kernel-axioms*]
by(*simp add*: *h*(*1*) *sk.bind-kernel-return*)
qed

Associativity seems not to hold for *all-meas-qbs*.

lemma *bind-qbs-cong-all-meas*:
assumes [*qbs*]:$s \in$ *qbs-space* (*all-meas-qbs X*)
$\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow f\ x = g\ x$
and [*qbs*]:$f \in X \to_Q$ *all-meas-qbs Y*
shows $s \gg\!= f = s \gg\!= g$
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** α μ **where** *h*:
$s = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas X* α μ **by** *auto*
interpret *qbs-meas X* α μ **by** *fact*
from *rep-all-meas-qbs-Mx*[*OF qbs-morphism-Mx*[*OF assms*(*3*) *in-Mx*]] **obtain** β *k* **where** *h*′:
$f \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$ $\beta \in$ *qbs-Mx Y measure-kernel borel borel k* **by** *metis*
have *g*: $g \in X \to_Q$ *all-meas-qbs Y* $g \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$
using *qbs-Mx-to-X*[*OF in-Mx*] *assms*(*2*) *fun-cong*[*OF h*′(*1*)]
by(*auto simp*: *assms*(*2*)[*symmetric*] *cong*: *qbs-morphism-cong*)
show *?thesis*
by(*simp add*: *bind-qbs-all-meas*[*OF h*(*1*) *assms*(*3*) *h*′(*2*,*3*,*1*)] *bind-qbs-all-meas*[*OF h*(*1*) *g*(*1*) *h*′(*2*,*3*) *g*(*2*)])
qed

4.1.6 The Functorial Action

definition *distr-qbs* :: ['*a quasi-borel*, '*b quasi-borel*,'*a* ⇒ '*b*,'*a qbs-measure*] ⇒ '*b qbs-measure* **where**
distr-qbs - Y f sx ≡ *sx* $\gg\!=$ *return-qbs Y* ∘ *f*

lemma *distr-qbs-morphism-all-meas*′:
assumes $f \in X \to_Q Y$
shows *distr-qbs X Y f* ∈ *all-meas-qbs X* $\to_Q$ *all-meas-qbs Y*
unfolding *distr-qbs-def*

by(*rule bind-qbs-morphism-all-meas'*[*OF qbs-morphism-comp*[*OF assms return-qbs-morphism-all-meas*]])

lemma(**in** *qbs-meas*)
assumes $s = [\![X, \alpha, \mu]\!]_{meas}$
and $f \in X \to_Q Y$
shows *distr-qbs-meas*:*qbs-meas* Y $(f \circ \alpha)$ μ
and *distr-qbs*: *distr-qbs* X Y f $s = [\![Y, f \circ \alpha, \mu]\!]_{meas}$
by(*auto intro*!: *bind-qbs-all-meas*[*OF assms*(*1*) *qbs-morphism-comp*[*OF assms*(*2*) *return-qbs-morphism-all-meas*],*of f* ∘ α *return borel* ,*simplified bind-kernel-return''*[*OF mu-sets*]]
bind-qbs-meas[*OF assms*(*1*) *qbs-morphism-comp*[*OF assms*(*2*) *return-qbs-morphism-all-meas*],*of f* ∘ α *return borel* ,*simplified bind-kernel-return''*[*OF mu-sets*]]
simp: *distr-qbs-def return-qbs-comp*[*OF qbs-morphism-Mx*[*OF assms*(*2*) *in-Mx*],*simplified comp-assoc*[*symmetric*]] *qbs-morphism-Mx*[*OF assms*(*2*) *in-Mx*]
prob-kernel.axioms(*1*)[*of borel borel return borel*,*simplified prob-kernel-def'*])

lemma(**in** *qbs-s-finite*) *distr-qbs-s-finite*:
assumes [*qbs*]:$f \in X \to_Q Y$
shows *qbs-s-finite* Y $(f \circ \alpha)$ μ
using *qbs-s-finite-axioms* **by**(*auto simp*: *qbs-s-finite-def qbs-meas-def in-Mx-def intro*!: *qbs-morphism-Mx*[*of - X*])

lemma(**in** *qbs-prob*) *distr-qbs-prob*:
assumes [*qbs*]: $f \in X \to_Q Y$
shows *qbs-prob* Y $(f \circ \alpha)$ μ
using *qbs-prob-axioms* **by**(*auto simp*: *qbs-prob-def qbs-meas-def in-Mx-def intro*!: *qbs-morphism-Mx*[*of - X*])

lemma *distr-qbs-id-all-meas*:
assumes $s \in$ *qbs-space* (*all-meas-qbs X*)
shows *distr-qbs X X id s* = *s*
using *bind-qbs-return-all-meas'*[*OF assms*] **by**(*simp add*: *distr-qbs-def*)

lemma *distr-qbs-comp-all-meas*:
assumes $s \in$ *qbs-space* (*all-meas-qbs X*)
$f \in X \to_Q Y$
and $g \in Y \to_Q Z$
shows ((*distr-qbs Y Z g*) ∘ (*distr-qbs X Y f*)) *s* = *distr-qbs X Z* (*g* ∘ *f*) *s*
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** α μ **where** *h*:
qbs-meas X α μ *s* = $[\![X, \alpha, \mu]\!]_{meas}$ **by** *metis*
have *qbs-meas Y* (*f* ∘ α) μ *distr-qbs X Y f s* = $[\![Y, f \circ \alpha, \mu]\!]_{meas}$
by(*simp-all add*: *qbs-meas.distr-qbs-meas*[*OF h assms*(*2*)] *qbs-meas.distr-qbs*[*OF h assms*(*2*)])
from *qbs-meas.distr-qbs*[*OF this assms*(*3*)] *qbs-meas.distr-qbs*[*OF h qbs-morphism-comp*[*OF assms*(*2*,*3*)]]
show *?thesis*

by(*simp add*: *comp-assoc*)
qed

4.1.7 Join

definition *join-qbs* :: *'a qbs-measure qbs-measure* ⇒ *'a qbs-measure* **where**
join-qbs ≡ (λ*sst*. *sst* ⤜ *id*)

lemma *join-qbs-morphism-all-meas*: *join-qbs* ∈ *all-meas-qbs* (*all-meas-qbs X*) →$_Q$ *all-meas-qbs X*
by(*simp add*: *join-qbs-def bind-qbs-morphism-all-meas'*)

lemma
assumes *qbs-meas* (*all-meas-qbs X*) *β μ*
ssx = ⟦*all-meas-qbs X*, *β*, *μ*⟧$_{meas}$
α ∈ *qbs-Mx X*
measure-kernel borel borel k
and *β* =(λ*r*. ⟦*X*, *α*, *k r*⟧$_{meas}$)
shows *join-qbs-meas*: *qbs-meas X α* (*μ* ⤜$_k$ *k*)
and *join-qbs-all-meas*: *join-qbs ssx* = ⟦*X*, *α*, *μ* ⤜$_k$ *k*⟧$_{meas}$
using *qbs-meas.bind-qbs-all-meas*[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)]
qbs-meas.bind-qbs-meas[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)]
by(*auto simp*: *assms*(*5*) *join-qbs-def*)

4.1.8 Strength

definition *strength-qbs* :: [*'a quasi-borel*,*'b quasi-borel*,*'a* × *'b qbs-measure*] ⇒ (*'a* × *'b*) *qbs-measure* **where**
strength-qbs W X = (λ(*w*,*sx*). *let* (-,*α*,*μ*) = *rep-qbs-measure sx*
in ⟦*W* ⨂$_Q$ *X*, λ*r*. (*w*, *α r*), *μ*⟧$_{meas}$)

lemma(**in** *qbs-meas*)
assumes [*qbs*]:*w* ∈ *qbs-space W*
and *sx* = ⟦*X*, *α*, *μ*⟧$_{meas}$
shows *strength-qbs-meas*: *qbs-meas* (*W* ⨂$_Q$ *X*) (λ*r*. (*w*,*α r*)) *μ*
and *strength-qbs*: *strength-qbs W X* (*w*,*sx*) = ⟦*W* ⨂$_Q$ *X*, λ*r*. (*w*,*α r*), *μ*⟧$_{meas}$
proof −
interpret *qs*: *qbs-meas W* ⨂$_Q$ *X* λ*r*. (*w*,*α r*) *μ*
by(*auto simp*: *qbs-meas-def qbs-meas-axioms-def mu-sets in-Mx-def assms*(*1*) *intro*!: *pair-qbs-MxI*)
show *qbs-meas* (*W* ⨂$_Q$ *X*) (λ*r*. (*w*,*α r*)) *μ* **by** (*rule qs.qbs-meas-axioms*)
show *strength-qbs W X* (*w*,*sx*) = ⟦*W* ⨂$_Q$ *X*, λ*r*. (*w*,*α r*), *μ*⟧$_{meas}$
proof −
{
fix *X' α' μ'*
assume (*X'*,*α'*,*μ'*) ∈ *Rep-qbs-measure* ⟦*X*, *α*, *μ*⟧$_{meas}$
then have *h*: *X'* = *X qbs-meas X' α' μ' qbs-meas-eq* (*X*,*α*,*μ*) (*X'*,*α'*,*μ'*)
by(*simp-all add*: *in-Rep-qbs-measure-dest*)
then interpret *qs'*: *qbs-meas W* ⨂$_Q$ *X* λ*r*. (*w*,*α' r*) *μ'*
by(*auto simp*: *qbs-meas-def in-Mx-def assms*(*1*) *intro*!: *pair-qbs-MxI*)

interpret *pq*: *pair-qbs-meas W* $\bigotimes_Q$ *X* $\lambda r.\ (w,\alpha\ r)\ \mu\ \lambda r.\ (w,\alpha'\ r)\ \mu'$
by *standard*
have $[\![W \bigotimes_Q X, \lambda r.\ (w,\ \alpha'\ r),\ \mu']\!]_{meas} = [\![W \bigotimes_Q X, \lambda r.\ (w,\ \alpha\ r),\ \mu]\!]_{meas}$
using *h*(*3*) **by**(*auto simp*: *qbs-meas-eq-def2 h*(*1*) *intro*!: *pq.qbs-meas-eqI2*[*symmetric*])
}
note $* = this$
show *?thesis*
unfolding *strength-qbs-def rep-qbs-measure-def qbs-measure.rep-def assms*(*2*) *prod.case*
apply(*rule someI2*)
apply(*rule in-Rep-qbs-measure*)
using $*$ **by** *auto*
qed
qed

lemma(**in** *qbs-s-finite*) *strength-qbs-s-finite*: $w \in$ *qbs-space W* $\Longrightarrow$ *qbs-s-finite* (*W* $\bigotimes_Q$ *X*) $(\lambda r.\ (w,\alpha\ r))\ \mu$
using *qbs-s-finite-axioms* **by**(*auto simp*: *qbs-s-finite-def qbs-meas-def in-Mx-def intro*!: *pair-qbs-MxI*)

lemma(**in** *qbs-prob*) *strength-qbs-prob*: $w \in$ *qbs-space W* $\Longrightarrow$ *qbs-prob* (*W* $\bigotimes_Q$ *X*) $(\lambda r.\ (w,\alpha\ r))\ \mu$
using *qbs-prob-axioms* **by**(*auto simp*: *qbs-prob-def qbs-meas-def in-Mx-def intro*!: *pair-qbs-MxI*)

lemma *strength-qbs-natural-all-meas*:
assumes [*qbs*]:$f \in X \to_Q X'\ g \in Y \to_Q Y'\ x \in$ *qbs-space X sy* $\in$ *qbs-space* (*all-meas-qbs Y*)
shows (*distr-qbs* (*X* $\bigotimes_Q$ *Y*) (*X'* $\bigotimes_Q$ *Y'*) (*map-prod f g*) $\circ$ *strength-qbs X Y*) (*x*,*sy*) = (*strength-qbs X' Y'* $\circ$ *map-prod f* (*distr-qbs Y Y' g*)) (*x*,*sy*)
(**is** *?lhs* = *?rhs*)
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*4*)] **obtain** $\alpha\ \mu$
where *h*:$sy = [\![Y,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas Y* $\alpha\ \mu$ **by** *metis*
have *?lhs* = (*distr-qbs* (*X* $\bigotimes_Q$ *Y*) (*X'* $\bigotimes_Q$ *Y'*) (*map-prod f g*)) ($[\![X \bigotimes_Q Y,$ $\lambda r.\ (x,\alpha\ r),\ \mu]\!]_{meas}$)
by(*simp add*: *qbs-meas.strength-qbs*[*OF h*(*2*) *assms*(*3*) *h*(*1*)])
also have ... = $[\![X' \bigotimes_Q Y',$ *map-prod f g* $\circ$ $(\lambda r.\ (x,\ \alpha\ r)),\ \mu]\!]_{meas}$
using *assms* **by**(*simp add*: *qbs-meas.distr-qbs*[*OF qbs-meas.strength-qbs-meas*[*OF h*(*2*) *assms*(*3*) *h*(*1*)]])
also have ... = $[\![X' \bigotimes_Q Y', \lambda r.\ (f\,x,\ (g \circ \alpha)\ r),\ \mu]\!]_{meas}$ **by** (*simp add*: *comp-def*)
also have ... = *?rhs*
by(*simp add*: *qbs-meas.strength-qbs*[*OF qbs-meas.distr-qbs-meas*[*OF h*(*2*,*1*) *assms*(*2*)] *qbs-morphism-space*[*OF assms*(*1*,*3*)] *qbs-meas.distr-qbs*[*OF h*(*2*,*1*) *assms*(*2*)]])
finally show *?thesis* **.**
qed

lemma *strength-qbs-law1-all-meas*:
assumes $x \in$ *qbs-space* (*unit-quasi-borel* $\bigotimes_Q$ *all-meas-qbs X*)

shows *snd x* = (*distr-qbs* (*unit-quasi-borel* $\bigotimes_Q$ *X*) *X snd* ∘ *strength-qbs unit-quasi-borel X*) *x*
proof −
obtain α μ **where** *h*:
qbs-meas X α μ (*snd x*) = $[\![X, \alpha, \mu]\!]_{meas}$
using *rep-qbs-space-all-meas*[*of snd x X*] *assms*
by (*metis qbs-morphism-space snd-qbs-morphism*)
have [*simp*]: ((),*snd x*) = *x*
using *SigmaE assms* **by** (*auto simp*: *pair-qbs-space*)
show *?thesis*
using *qbs-meas.distr-qbs*[*OF qbs-meas.strength-qbs-meas*[*OF h*(*1*) - *h*(*2*),*of fst x unit-quasi-borel*]
qbs-meas.strength-qbs[*OF h*(*1*) - *h*(*2*)] *snd-qbs-morphism*]
by(*auto simp*: *comp-def*,*simp add*: *h*(*2*))
qed

lemma *strength-qbs-law2-all-meas*:
assumes *x* ∈ *qbs-space* ((*X* $\bigotimes_Q$ *Y*) $\bigotimes_Q$ *all-meas-qbs Z*)
shows (*strength-qbs X* (*Y* $\bigotimes_Q$ *Z*) ∘ (*map-prod id* (*strength-qbs Y Z*)) ∘ (λ((*x*,*y*),*z*). (*x*,(*y*,*z*)))) *x* =
(*distr-qbs* ((*X* $\bigotimes_Q$ *Y*) $\bigotimes_Q$ *Z*) (*X* $\bigotimes_Q$ (*Y* $\bigotimes_Q$ *Z*)) (λ((*x*,*y*),*z*). (*x*,(*y*,*z*))) ∘ *strength-qbs* (*X* $\bigotimes_Q$ *Y*) *Z*) *x*
(**is** *?lhs* = *?rhs*)
proof −
obtain α μ **where** *h*:
qbs-meas Z α μ *snd x* = $[\![Z, \alpha, \mu]\!]_{meas}$
using *rep-qbs-space-all-meas*[*of snd x Z*] *assms*
by (*metis qbs-morphism-space snd-qbs-morphism*)
then have *?lhs* = $[\![X \bigotimes_Q Y \bigotimes_Q Z, \lambda r.\ (fst\ (fst\ x), snd\ (fst\ x), \alpha\ r), \mu]\!]_{meas}$
using *assms qbs-meas.strength-qbs-meas*[*OF h*(*1*),*of snd* (*fst x*) *Y*]
by(*auto intro*!: *qbs-meas.strength-qbs simp*: *pair-qbs-space*)
also have ... = *?rhs*
using *qbs-meas.distr-qbs*[*OF qbs-meas.strength-qbs-meas*[*OF h*(*1*) - *h*(*2*),*of fst x X* $\bigotimes_Q$ *Y*]
qbs-meas.strength-qbs[*OF h*(*1*) - *h*(*2*),*of fst x X* $\bigotimes_Q$ *Y*] *qbs-morphism-pair-assoc1*] *assms*
by(*auto simp*: *comp-def pair-qbs-space*)
finally show *?thesis* **.**
qed

4.1.9 The s-Finite Measure Monad

definition *monadM-qbs* :: *'a quasi-borel* ⇒ *'a qbs-measure quasi-borel* **where**
monadM-qbs X ≡ *Abs-quasi-borel* ({$[\![X, \alpha, \mu]\!]_{meas}$ |α μ. *qbs-s-finite X* α μ}, {λ*r*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. α ∈ *qbs-Mx X* ∧ *s-finite-kernel borel borel k*})

lemma
shows *monadM-qbs-space*: *qbs-space* (*monadM-qbs X*) = {$[\![X, \alpha, \mu]\!]_{meas}$ |α μ. *qbs-s-finite X* α μ}

and *monadM-qbs-Mx*: *qbs-Mx* (*monadM-qbs X*) = {λr. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. $\alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*}
proof −
have {λr::*real*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. $\alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*}
$\subseteq$ *UNIV* $\to$ {$[\![X, \alpha, \mu]\!]_{meas}$ |α μ. *qbs-s-finite X* α μ}
proof *safe*
fix *x* α **and** *k* :: *real* $\Rightarrow$ *real measure*
assume *h*:$\alpha \in$ *qbs-Mx X s-finite-kernel borel borel k*
interpret *k*:*s-finite-kernel borel borel k* **by** *fact*
interpret *qbs-s-finite X* α *k x*
using *k.image-s-finite-measure h*(*1*)
by(*auto simp*: *qbs-s-finite-def in-Mx-def qbs-meas-def qbs-meas-axioms-def k.kernel-sets*)
show $\exists \alpha'\ \mu'$. $[\![X, \alpha, k\ x]\!]_{meas} = [\![X, \alpha', \mu']\!]_{meas} \wedge$ *qbs-s-finite X* α' μ'
by(*auto intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=*k x*] *qbs-s-finite-axioms*)
qed
moreover have *qbs-closed1* {λr. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. $\alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*}
proof(*safe intro*!: *qbs-closed1I*)
fix α **and** *f* :: *real* $\Rightarrow$ *real* **and** *k* :: *real*$\Rightarrow$ *real measure*
assume *h*:*f* $\in$ *borel-measurable borel* $\alpha \in$ *qbs-Mx X s-finite-kernel borel borel k*
then show $\exists \alpha'$ *ka*. (λr. $[\![X, \alpha, k\ r]\!]_{meas}$) $\circ$ *f* = (λr. $[\![X, \alpha', ka\ r]\!]_{meas}$) $\wedge$ $\alpha' \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel ka*
by(*auto intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=λx. *k* (*f x*)] *simp*: *s-finite-kernel.comp-measurable*[*OF h*(*3*,*1*)])
qed
moreover have *qbs-closed2* {$[\![X, \alpha, \mu]\!]_{meas}$ |α μ. *qbs-s-finite X* α μ} {λr. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. $\alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*}
proof(*safe intro*!: *qbs-closed2I*)
fix α μ
assume *qbs-s-finite X* α μ
then interpret *qbs-s-finite X* α μ **.**
show $\exists \alpha'$ *k*. (λr. $[\![X, \alpha, \mu]\!]_{meas}$) = ($\lambda r$. $[\![X, \alpha', k\ r]\!]_{meas}$) $\wedge$ $\alpha' \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*
by(*auto intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=λr. μ] *s-finite-kernel-const*$'$ *mu-sets*)
qed
moreover have *qbs-closed3* {λr. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. $\alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*}
proof(*safe intro*!: *qbs-closed3I*)
fix *P* :: *real* $\Rightarrow$ *nat* **and** *Fi* :: *nat* $\Rightarrow$ -
assume *P*[*measurable*]: *P* $\in$ *borel* $\to_M$ *count-space UNIV*
and $\forall$ *i*. *Fi i* $\in$ {λr::*real*. $[\![X, \alpha, k\ r]\!]_{meas}$ |α *k*. $\alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*}
then obtain αi *ki* **where** *Fi*: $\bigwedge i$. *Fi i* = (λr. $[\![X, \alpha i\ i, ki\ i\ r]\!]_{meas}$) $\bigwedge i$. $\alpha i\ i \in$ *qbs-Mx X* $\bigwedge i$. *s-finite-kernel borel borel* (*ki i*)
by *auto metis*
interpret *nat-real*: *standard-borel-ne count-space* (*UNIV* :: *nat set*) $\bigotimes_M$ (*borel*

:: *real measure*)
by(*auto intro*!: *pair-standard-borel-ne*)
note [*simp*] = *nat-real.from-real-to-real*[*simplified space-pair-measure*, *simplified*]
define α **where** $\alpha \equiv (\lambda r.$ *case-prod* αi (*nat-real.from-real* $r))$
define k **where** $k \equiv (\lambda r.$ *distr* (*distr* (*ki* (*P* r) r) (*count-space UNIV* $\bigotimes_M$ *borel*) $(\lambda r'.$ (*P* r, $r'))$) *borel nat-real.to-real*)
have α: $\alpha \in$ *qbs-Mx X*
unfolding α*-def qbs-Mx-is-morphisms*
proof(*rule qbs-morphism-compose*[**where** *g*=*nat-real.from-real* **and** *Y*=*qbs-count-space UNIV* $\bigotimes_Q$ *qbs-borel*])
show *nat-real.from-real* $\in$ *qbs-borel* $\to_Q$ *qbs-count-space UNIV* $\bigotimes_Q$ *qbs-borel*
by(*simp add*: *r-preserves-product*[*symmetric*] *standard-borel.standard-borel-r-full-faithful*[*of borel* :: *real measure*,*simplified*,*symmetric*] *standard-borel-ne.standard-borel*)
next
show *case-prod* $\alpha i \in$ *qbs-count-space UNIV* $\bigotimes_Q$ *qbs-borel* $\to_Q$ *X*
using *Fi*(*2*) **by**(*auto intro*!: *qbs-morphism-pair-count-space1 simp*: *qbs-Mx-is-morphisms*)
qed
have *sets-ki*[*measurable-cong*]: *sets* (*ki i r*) = *sets borel sets* (*k r*) = *sets borel* **for** *i r*
using *Fi*(*3*) **by**(*auto simp*: *s-finite-kernel-def measure-kernel-def k-def*)
interpret *k*:*s-finite-kernel borel borel k*
proof −
have *1*:*k* = $(\lambda(i,r).$ *distr* (*ki i r*) *borel* $(\lambda r'.$ *nat-real.to-real* $(i, r'))) \circ (\lambda r.$ $(P\ r, r))$
by *standard* (*auto simp*: *k-def distr-distr comp-def*)
have *s-finite-kernel borel borel* ...
unfolding *comp-def*
by(*rule s-finite-kernel.comp-measurable*[**where** *X*=*count-space UNIV* $\bigotimes_M$ *borel*])
(*auto intro*!: *s-finite-kernel-pair-countble1 s-finite-kernel.distr-s-finite-kernel*[*OF Fi*(*3*)])
thus *s-finite-kernel borel borel k* **by**(*simp add*: *1*)
qed
have $(\lambda r.$ *Fi* (*P r*) $r) = (\lambda r.$ $[\![X, \alpha, k\ r]\!]_{meas})$
unfolding *Fi*(*1*)
proof
fix r
interpret *pq*:*pair-qbs-s-finite X* αi (*P r*) *ki* (*P r*) r α *k r*
by(*auto simp*: *pair-qbs-s-finite-def qbs-s-finite-def in-Mx-def qbs-meas-def qbs-meas-axioms-def*
k.image-s-finite-measure s-finite-kernel.image-s-finite-measure[*OF Fi*(*3*)] *sets-ki* α *Fi*(*2*))
show $[\![X, \alpha i\ (P\ r), ki\ (P\ r)\ r]\!]_{meas} = [\![X, \alpha, k\ r]\!]_{meas}$
by(*intro pq.qbs-meas-eqI*) (*simp add*: *k-def distr-distr*, *simp add*: *comp-def* α*-def*)
qed
thus $\exists \alpha\ k.\ (\lambda r.$ *Fi* (*P r*) $r) = (\lambda r.$ $[\![X, \alpha, k\ r]\!]_{meas}) \wedge \alpha \in$ *qbs-Mx X* $\wedge$ *s-finite-kernel borel borel k*
by(*auto intro*!: *exI*[**where** x=α] *exI*[**where** x=*k*] *simp*: α *k.s-finite-kernel-axioms*)

qed
ultimately have *Rep-quasi-borel* (*monadM-qbs X*) = ({⟦*X*, *α*, *μ*⟧$_{meas}$ |*α μ*. *qbs-s-finite X α μ*}, {*λr*. ⟦*X*, *α*, *k r*⟧$_{meas}$ |*α k*. *α* ∈ *qbs-Mx X* ∧ *s-finite-kernel borel borel k*})
by(*auto intro*!: *Abs-quasi-borel-inverse simp*: *monadM-qbs-def is-quasi-borel-def*)
thus *qbs-space* (*monadM-qbs X*) ={⟦*X*, *α*, *μ*⟧$_{meas}$ |*α μ*. *qbs-s-finite X α μ*}
qbs-Mx (*monadM-qbs X*) = {*λr*. ⟦*X*, *α*, *k r*⟧$_{meas}$ |*α k*. *α* ∈ *qbs-Mx X* ∧ *s-finite-kernel borel borel k*}
by(*simp-all add*: *qbs-space-def qbs-Mx-def*)
qed

lemma *monadM-all-meas-space'*: *qbs-space* (*monadM-qbs X*) ⊆ *qbs-space* (*all-meas-qbs X*)
and *monadM-all-meas-space*: ⋀*p*. *p* ∈ *qbs-space* (*monadM-qbs X*) ⟹ *p* ∈ *qbs-space* (*all-meas-qbs X*)
and *monadM-all-meas-Mx*: *qbs-Mx* (*monadM-qbs X*) ⊆ *qbs-Mx* (*all-meas-qbs X*)
by(*auto simp*: *monadM-qbs-space monadM-qbs-Mx all-meas-qbs-space all-meas-qbs-Mx qbs-meas.qbs-space-of*[*OF qbs-s-finite.axioms*(*1*)] *dest*: *s-finite-kernel.axioms*(*1*))

lemma
shows *qbs-morphism-monadAD*: *f* ∈ *X* →$_Q$ *monadM-qbs Y* ⟹ *f* ∈ *X* →$_Q$ *all-meas-qbs Y*
and *qbs-morphism-monadAD'*: *g* ∈ *all-meas-qbs X* →$_Q$ *Y* ⟹ *g* ∈ *monadM-qbs X* →$_Q$ *Y*
using *monadM-all-meas-Mx* **by**(*auto intro*!: *qbs-morphismI dest*: *qbs-morphism-Mx*)

lemma *monadM-qbs-empty-iff*: *qbs-space X* = {} ⟷ *qbs-space* (*monadM-qbs X*) = {}
by (*metis* (*mono-tags*, *lifting*) *all-meas-qbs-empty-iff monadM-all-meas-space' monadM-qbs-space bot.extremum-uniqueI empty-Collect-eq qbs-null-measure-s-finite*)

lemma(**in** *qbs-s-finite*) *in-space-monadM*[*qbs*]: ⟦*X*, *α*, *μ*⟧$_{meas}$ ∈ *qbs-space* (*monadM-qbs X*)
using *qbs-s-finite-axioms* **by**(*auto simp add*: *monadM-qbs-space*)

lemma *rep-qbs-space-monadM*:
assumes *s* ∈ *qbs-space* (*monadM-qbs X*)
obtains *α μ* **where** *s* = ⟦*X*, *α*, *μ*⟧$_{meas}$ *qbs-s-finite X α μ*
using *assms monadM-qbs-space* **by** *fastforce*

lemma *rep-qbs-space-monadM-sigma-finite*:
assumes *s* ∈ *qbs-space* (*monadM-qbs X*)
obtains *α μ* **where** *s* = ⟦*X*, *α*, *μ*⟧$_{meas}$ *qbs-s-finite X α μ sigma-finite-measure μ*
proof −
obtain *α μ* **where** *s*:*s* = ⟦*X*, *α*, *μ*⟧$_{meas}$ *qbs-s-finite X α μ*
by(*metis rep-qbs-space-monadM assms*)
hence *standard-borel-ne μs-finite-measure μ*
by(*auto intro*!: *standard-borel-ne-sets*[*of borel μ*] *simp*: *qbs-s-finite-def qbs-meas-def*

qbs-meas-axioms-def)
from *exists-push-forward*[*OF this*] **obtain** μ' f **where** f:
$f \in (borel :: real\ measure) \to_M \mu$ *sets* $\mu' =$ *sets borel sigma-finite-measure* μ' *distr* μ' μ $f = \mu$
by *metis*
hence [*measurable*]: $f \in$ *borel-measurable borel*
using *s*(*2*) **by**(*auto simp*: *qbs-s-finite-def qbs-meas-def qbs-meas-axioms-def cong*: *measurable-cong-sets*)
interpret *pair-qbs-s-finite X* α μ $\alpha \circ f$ μ'
proof −
have *qbs-s-finite X* $(\alpha \circ f)$ μ'
using *s*(*2*)
by(*auto simp*: *qbs-s-finite-def in-Mx-def qbs-meas-def qbs-meas-axioms-def f*(*2*,*3*) *sigma-finite-measure.s-finite-measure*)
thus *pair-qbs-s-finite X* α μ $(\alpha \circ f)$ μ'
by(*auto simp*: *pair-qbs-s-finite-def s*(*2*))
qed
have $[\![X, \alpha, \mu]\!]_{meas} = [\![X, \alpha \circ f, \mu']\!]_{meas}$
proof −
have [*simp*]: *distr* μ (*qbs-to-measure X*) α = *distr* (*distr* μ' μ f) (*qbs-to-measure X*) α
by(*simp add*: *f*(*4*))
show *?thesis*
by(*auto intro*!: *qbs-meas-eqI simp*: *distr-distr*)
qed
with *s*(*1*) *pq2.qbs-s-finite-axioms f*(*3*) *that*
show *?thesis* **by** *metis*
qed

lemma *qbs-space-of-in*: $s \in$ *qbs-space* (*monadM-qbs X*) $\Longrightarrow$ *qbs-space-of s* = *X*
using *monadM-all-meas-space qbs-space-of-in-all-meas* **by** *blast*

lemma *qbs-l-s-finite*:
assumes $p \in$ *qbs-space* (*monadM-qbs X*)
shows *s-finite-measure* (*qbs-l p*)
proof −
obtain α μ **where** *p*: $p = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-s-finite X* α μ
using *rep-qbs-space-monadM*[*OF assms*] **by** *blast*
interpret *qbs-s-finite X* α μ **by** *fact*
show *?thesis*
by(*simp add*: *qbs-l p*(*1*) *s-finite-measure-distr*)
qed

lemma *qbs-null-measure-in-Mx*: *qbs-space X* $\neq \{\}$ $\Longrightarrow$ *qbs-null-measure X* $\in$ *qbs-space* (*monadM-qbs X*)
by(*simp add*: *qbs-s-finite.in-space-monadM*[*OF qbs-null-measure-s-finite*] *qbs-null-measure-def*)

lemma *space-qbs-l-in*:

assumes $s \in$ *qbs-space* (*monadM-qbs* X)
shows *space* (*qbs-l* s) = *qbs-space* X
using *space-qbs-l-in-all-meas assms monadM-all-meas-space* **by** *blast*

lemma *sets-qbs-l*:
assumes $s \in$ *qbs-space* (*monadM-qbs* X)
shows *sets* (*qbs-l* s) = *sets* (*qbs-to-measure* X)
using *assms qbs-l-sets qbs-space-of-in* **by** *blast*

lemma *measurable-qbs-l*:
assumes $s \in$ *qbs-space* (*monadM-qbs* X)
shows *qbs-l* $s \to_M$ M = $X \to_Q$ *measure-to-qbs* M
by(*auto simp*: *measurable-cong-sets*[*OF qbs-l-sets*[*of s*,*simplified qbs-space-of-in*[*OF assms*(*1*)],*symmetric*] *refl*] *lr-adjunction-correspondence*)

lemma *measurable-qbs-l'*:
assumes $s \in$ *qbs-space* (*monadM-qbs* X)
shows *qbs-l* $s \to_M$ M = *qbs-to-measure* $X \to_M$ M
by(*simp add*: *measurable-qbs-l*[*OF assms*] *lr-adjunction-correspondence*)

lemma *rep-qbs-Mx-monadM*:
assumes $\gamma \in$ *qbs-Mx* (*monadM-qbs* X)
obtains α k **where** $\gamma = (\lambda r.\ [\![X, \alpha, k\ r]\!]_{meas})$ $\alpha \in$ *qbs-Mx X s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite X* α (*k r*)
using *assms* **by**(*fastforce simp*: *monadM-qbs-Mx qbs-s-finite-All*)

lemma *qbs-l-measurable*[*measurable*]:*qbs-l* $\in$ *qbs-to-measure* (*monadM-qbs X*) $\to_M$ *s-finite-measure-algebra* (*qbs-to-measure X*)
proof(*rule qbs-morphism-dest*[*OF qbs-morphismI*])
fix γ
assume $\gamma \in$ *qbs-Mx* (*monadM-qbs X*)
from *rep-qbs-Mx-monadM*[*OF this*] **obtain** α k **where** h:
$\gamma = (\lambda r.\ [\![X, \alpha, k\ r]\!]_{meas})$ $\alpha \in$ *qbs-Mx X s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite X* α (*k r*)
by *metis*
show *qbs-l* $\circ$ $\gamma \in$ *qbs-Mx* (*measure-to-qbs* (*s-finite-measure-algebra* (*qbs-to-measure X*)))
by(*auto simp add*: *qbs-Mx-R comp-def h*(*1*) *qbs-meas.qbs-l*[*OF qbs-s-finite.axioms*(*1*)[*OF h*(*4*)]] *h*(*2*,*3*)
intro!: *s-finite-kernel.kernel-measurable-s-finite s-finite-kernel.distr-s-finite-kernel*[**where** *Y*=*borel*])
qed

lemma *qbs-l-measure-kernel*: *measure-kernel* (*qbs-to-measure* (*monadM-qbs X*)) (*qbs-to-measure X*) *qbs-l*
proof(*cases qbs-space X* = {})
case *True*
with *monadM-qbs-empty-iff*[*of X*,*simplified this*] **show** *?thesis*
by(*auto intro*!: *measure-kernel-empty-trivial simp*: *space-L*)

next
case *1*:*False*
show *?thesis*
proof
show $\bigwedge x.\ x \in space\ (qbs\text{-}to\text{-}measure\ (monadM\text{-}qbs\ X)) \Longrightarrow sets\ (qbs\text{-}l\ x) = sets\ (qbs\text{-}to\text{-}measure\ X)$
using *qbs-l-sets qbs-space-of-in* **by**(*auto simp*: *space-L*)
next
show $space\ (qbs\text{-}to\text{-}measure\ X) \neq \{\}$
by(*simp add*: *space-L 1*)
qed (*rule measurable-emeasure-kernel-s-finite-measure-algebra*[*OF qbs-l-measurable*])
qed

lemmas *qbs-l-inj* = *inj-on-subset*[*OF qbs-l-inj-all-meas monadM-all-meas-space'*]

lemmas *qbs-l-morphism* = *qbs-morphism-monadAD'*[*OF qbs-l-morphism-all-meas*]

lemmas *qbs-l-finite-pred* = *qbs-morphism-monadAD'*[*OF qbs-l-finite-pred-all-meas*]

lemmas *qbs-l-subprob-pred* = *qbs-morphism-monadAD'*[*OF qbs-l-subprob-pred-all-meas*]

lemmas *qbs-l-prob-pred* = *qbs-morphism-monadAD'*[*OF qbs-l-prob-pred-all-meas*]

lemma *return-qbs-morphism*[*qbs*]: $return\text{-}qbs\ X \in X \to_Q monadM\text{-}qbs\ X$
proof(*rule qbs-morphismI*)
interpret *rr* : *real-distribution return borel 0*
by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-return*)
fix α
assume *h*:$\alpha \in qbs\text{-}Mx\ X$
then have *1*:$return\text{-}qbs\ X \circ \alpha = (\lambda r.\ [\![X,\ \alpha,\ return\ borel\ r]\!]_{meas})$
by(*rule return-qbs-comp*)
show $return\text{-}qbs\ X \circ \alpha \in qbs\text{-}Mx\ (monadM\text{-}qbs\ X)$
by(*auto simp*: *1 monadM-qbs-Mx h prob-kernel-def'*
intro!: *exI*[**where** *x*=α] *exI*[**where** *x*=*return borel*] *prob-kernel.s-finite-kernel-prob-kernel*)
qed

lemma(**in** *qbs-s-finite*)
assumes $s = [\![X,\ \alpha,\ \mu]\!]_{meas}$
$f \in X \to_Q monadM\text{-}qbs\ Y$
$\beta \in qbs\text{-}Mx\ Y$
s-finite-kernel borel borel k
and $(f \circ \alpha) = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$
shows *bind-qbs-s-finite*:*qbs-s-finite Y* β $(\mu \gg\!=_k k)$
and *bind-qbs*: $s \gg\!= f = [\![Y,\ \beta,\ \mu \gg\!=_k k]\!]_{meas}$
using *bind-qbs-all-meas*[*OF assms*(*1*) *qbs-morphism-monadAD*[*OF assms*(*2*)] *assms*(*3*)
s-finite-kernel.axioms(*1*)[*OF assms*(*4*)] *assms*(*5*)]
bind-qbs-meas[*OF assms*(*1*) *qbs-morphism-monadAD*[*OF assms*(*2*)] *assms*(*3*)
s-finite-kernel.axioms(*1*)[*OF assms*(*4*)] *assms*(*5*)]
assms(*4*) *mu-sets s-finite-measure-axioms*

by(*auto intro*!: *s-finite-kernel.comp-s-finite-measure*[*of borel borel*] *simp*: *qbs-s-finite-def*)

lemma *bind-qbs-morphism'*:
assumes $f \in X \to_Q$ *monadM-qbs Y*
shows $(\lambda x.\ x \gg\!= f) \in$ *monadM-qbs X* $\to_Q$ *monadM-qbs Y*
proof(*rule qbs-morphismI*)
fix γ
assume $\gamma \in$ *qbs-Mx* (*monadM-qbs X*)
from *rep-qbs-Mx-monadM*[*OF this*] **obtain** α *k* **where** *h*:
$\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})$ $\alpha \in$ *qbs-Mx X s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite X* α (*k r*)
by *metis*
from *rep-qbs-Mx-monadM*[*OF qbs-morphism-Mx*[*OF assms this*(*2*)]] **obtain** α' k' **where** *h'*:
$f \circ \alpha = (\lambda r.\ [\![Y,\ \alpha',\ k'\ r]\!]_{meas})$ $\alpha' \in$ *qbs-Mx Y s-finite-kernel borel borel k'* $\bigwedge r.$ *qbs-s-finite Y* α' (*k' r*)
by *metis*
have [*simp*]:$(\lambda x.\ x \gg\!= f) \circ \gamma = (\lambda r.\ [\![Y,\ \alpha',\ k\ r \gg\!=_k k']\!]_{meas})$
by *standard* (*simp add*: *h*(*1*) *qbs-s-finite.bind-qbs*[*OF h*(*4*) - *assms h'*(*2*,*3*,*1*)])
show $(\lambda x.\ x \gg\!= f) \circ \gamma \in$ *qbs-Mx* (*monadM-qbs Y*)
using *h'*(*2*) **by**(*auto simp*: *s-finite-kernel.bind-kernel-s-finite-kernel*[*OF h*(*3*) *h'*(*3*)] *monadM-qbs-Mx intro*!: *exI*[**where** *x*=α'])
qed

lemmas *bind-qbs-return'* = *bind-qbs-return-all-meas'*[*OF monadM-all-meas-space*]

lemmas *bind-qbs-return* = *bind-qbs-return-all-meas*[*OF qbs-morphism-monadAD*]

lemma *bind-qbs-assoc*:
assumes $s \in$ *qbs-space* (*monadM-qbs X*)
$f \in X \to_Q$ *monadM-qbs Y*
and $g \in Y \to_Q$ *monadM-qbs Z*
shows $s \gg\!= (\lambda x.\ f\ x \gg\!= g) = (s \gg\!= f) \gg\!= g$ (**is** *?lhs* = *?rhs*)
proof −
obtain α μ **where** *h*:$s = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-s-finite X* α μ
using *rep-qbs-space-monadM*[*OF assms*(*1*)] **by** *blast*
then interpret *qs*: *qbs-s-finite X* α μ **by** *simp*
from *rep-qbs-Mx-monadM*[*OF qbs-morphism-Mx*[*OF assms*(*2*) *qs.in-Mx*]] **obtain** β *k* **where** *h'*:
$f \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$ $\beta \in$ *qbs-Mx Y s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite Y* β (*k r*)
by *metis*
from *rep-qbs-Mx-monadM*[*OF qbs-morphism-Mx*[*OF assms*(*3*) *h'*(*2*)]] **obtain** γ k' **where** *h''*:
$g \circ \beta = (\lambda r.\ [\![Z,\ \gamma,\ k'\ r]\!]_{meas})$ $\gamma \in$ *qbs-Mx Z s-finite-kernel borel borel k'* $\bigwedge r.$ *qbs-s-finite Z* γ (*k' r*)
by *metis*
have *1*:$(\lambda x.\ f\ x \gg\!= g) \circ \alpha = (\lambda r.\ [\![Z,\ \gamma,\ k\ r \gg\!=_k k']\!]_{meas})$
by *standard* (*simp add*: *qbs-s-finite.bind-qbs*[*OF h'*(*4*) *fun-cong*[*OF h'*(*1*),*simplified*]

assms(*3*) *h''*(*2*,*3*,*1*)])

have *?lhs* = $[\![Z, \gamma, \mu \gg\!=_k (\lambda r.\ k\ r \gg\!=_k k')]\!]_{meas}$
by(*rule qs.bind-qbs*[*OF h*(*1*) *qbs-morphism-compose*[*OF assms*(*2*) *bind-qbs-morphism'*[*OF assms*(*3*)]] *h''*(*2*) *s-finite-kernel.bind-kernel-s-finite-kernel*[*OF h'*(*3*) *h''*(*3*)] *1*])
also have ... = $[\![Z, \gamma, \mu \gg\!=_k k \gg\!=_k k']\!]_{meas}$
by(*simp add*: *s-finite-kernel.bind-kernel-assoc*[*OF h'*(*3*) *h''*(*3*) *qs.mu-sets*])
also have ... = *?rhs*
by(*simp add*: *qbs-s-finite.bind-qbs*[*OF qs.bind-qbs-s-finite*[*OF h*(*1*) *assms*(*2*) *h'*(*2*,*3*,*1*)] *qs.bind-qbs*[*OF h*(*1*) *assms*(*2*) *h'*(*2*,*3*,*1*)] *assms*(*3*) *h''*(*2*,*3*,*1*)])
finally show *?thesis* **.**
qed

lemma *bind-qbs-cong*:
assumes [*qbs*]:$s \in$ *qbs-space* (*monadM-qbs X*)
$\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow f\ x = g\ x$
and [*qbs*]:$f \in X \to_Q$ *monadM-qbs Y*
shows $s \gg\!= f = s \gg\!= g$
proof −
from *rep-qbs-space-monadM*[*OF assms*(*1*)] **obtain** $\alpha\ \mu$ **where** *h*:
$s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-s-finite X* $\alpha\ \mu$ **by** *auto*
interpret *qbs-s-finite X* $\alpha\ \mu$ **by** *fact*
from *rep-qbs-Mx-monadM*[*OF qbs-morphism-Mx*[*OF assms*(*3*) *in-Mx*]] **obtain** $\beta\ k$ **where** *h'*:
$f \circ \alpha = (\lambda r.\ [\![Y, \beta, k\ r]\!]_{meas})$ $\beta \in$ *qbs-Mx Y s-finite-kernel borel borel k* **by** *metis*
have *g*: $g \in X \to_Q$ *monadM-qbs Y* $g \circ \alpha = (\lambda r.\ [\![Y, \beta, k\ r]\!]_{meas})$
using *qbs-Mx-to-X*[*OF in-Mx*] *assms*(*2*) *fun-cong*[*OF h'*(*1*)]
by(*auto simp*: *assms*(*2*)[*symmetric*] *cong*: *qbs-morphism-cong*)
show *?thesis*
by(*simp add*: *bind-qbs*[*OF h*(*1*) *assms*(*3*) *h'*(*2*,*3*,*1*)] *bind-qbs*[*OF h*(*1*) *g*(*1*) *h'*(*2*,*3*) *g*(*2*)])
qed

lemma *distr-qbs-morphism'*:
assumes $f \in X \to_Q Y$
shows *distr-qbs X Y f* $\in$ *monadM-qbs X* $\to_Q$ *monadM-qbs Y*
unfolding *distr-qbs-def*
by(*rule bind-qbs-morphism'*[*OF qbs-morphism-comp*[*OF assms return-qbs-morphism*]])

We show that M is a functor i.e. M preserve identity and composition.

lemma *distr-qbs-id*:
assumes $s \in$ *qbs-space* (*monadM-qbs X*)
shows *distr-qbs X X id s* = *s*
using *bind-qbs-return'*[*OF assms*] **by**(*simp add*: *distr-qbs-def*)

lemma *distr-qbs-comp*:
assumes [*qbs*]:$s \in$ *qbs-space* (*monadM-qbs X*) $f \in X \to_Q Y$ $g \in Y \to_Q Z$
shows (*distr-qbs Y Z g*) $\circ$ (*distr-qbs X Y f*)) *s* = *distr-qbs X Z* ($g \circ f$) *s*
by(*intro distr-qbs-comp-all-meas*) (*auto simp*: *monadM-all-meas-space*)

lemma *join-qbs-morphism*[*qbs*]: *join-qbs* ∈ *monadM-qbs* (*monadM-qbs X*) →$_Q$ *monadM-qbs X*
by(*simp add*: *join-qbs-def bind-qbs-morphism'*)

lemma
assumes *qbs-s-finite* (*monadM-qbs X*) *β μ*
ssx = ⟦*monadM-qbs X*, *β*, *μ*⟧$_{meas}$
α ∈ *qbs-Mx X*
s-finite-kernel borel borel k
and *β* =(λ*r*. ⟦*X*, *α*, *k r*⟧$_{meas}$)
shows *join-qbs-s-finite*: *qbs-s-finite X α* (*μ* ⤜$_k$ *k*)
and *join-qbs*: *join-qbs ssx* = ⟦*X*, *α*, *μ* ⤜$_k$ *k*⟧$_{meas}$
using *qbs-s-finite.bind-qbs*[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)] *qbs-s-finite.bind-qbs-s-finite*[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)]
by(*auto simp*: *assms*(*5*) *join-qbs-def*)

lemma *strength-qbs-natural*:
assumes [*qbs*]:*f* ∈ *X* →$_Q$ *X'* *g* ∈ *Y* →$_Q$ *Y'* *x* ∈ *qbs-space X sy* ∈ *qbs-space* (*monadM-qbs Y*)
shows (*distr-qbs* (*X* ⨂$_Q$ *Y*) (*X'* ⨂$_Q$ *Y'*) (*map-prod f g*) ∘ *strength-qbs X Y*) (*x*,*sy*) = (*strength-qbs X' Y'* ∘ *map-prod f* (*distr-qbs Y Y' g*)) (*x*,*sy*)
by(*intro strength-qbs-natural-all-meas*) (*auto simp*: *monadM-all-meas-space*)

context
begin

interpretation *rr* : *standard-borel-ne borel* ⨂$_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

lemma *rr-from-real-to-real-id*[*simp*]: *rr.from-real* (*rr.to-real x*) = *x rr.from-real* ∘ *rr.to-real* = *id*
using *rr.from-real-to-real* **by**(*auto simp*: *comp-def space-pair-measure*)

lemma
assumes *α* ∈ *qbs-Mx X*
β ∈ *qbs-Mx* (*monadM-qbs Y*)
γ ∈ *qbs-Mx Y*
s-finite-kernel borel borel k
and *β* = (λ*r*. ⟦*Y*, *γ*, *k r*⟧$_{meas}$)
shows *strength-qbs-ab-r-s-finite*: *qbs-s-finite* (*X* ⨂$_Q$ *Y*) (*map-prod α γ* ∘ *rr.from-real*) (*distr* (*return borel r* ⨂$_M$ *k r*) *borel rr.to-real*)
and *strength-qbs-ab-r*: *strength-qbs X Y* (*α r*, *β r*) = ⟦*X* ⨂$_Q$ *Y*, *map-prod α γ* ∘ *rr.from-real*, *distr* (*return borel r* ⨂$_M$ *k r*) *borel rr.to-real*⟧$_{meas}$ (**is** *?goal2*)
proof −
interpret *k*: *s-finite-kernel borel borel k* **by** *fact*
note *1*[*measurable-cong*] = *sets-return*[*of borel r*] *k.kernel-sets*[*of r*,*simplified*]
show *qbs-s-finite* (*X* ⨂$_Q$ *Y*) (*map-prod α γ* ∘ *rr.from-real*) (*distr* (*return borel r* ⨂$_M$ *k r*) *borel rr.to-real*)

using *assms*(*1*,*3*)
by(*auto simp*: *qbs-s-finite-def qbs-meas-def in-Mx-def qbs-meas-axioms-def qbs-Mx-is-morphisms r-preserves-product*[*symmetric*]
standard-borel-ne.standard-borel
intro!: *s-finite-measure.s-finite-measure-distr*[*OF pair-measure-s-finite-measure*[*OF prob-space.s-finite-measure-prob*[*OF prob-space-return*[*of r borel*]]
k.image-s-finite-measure[*of r*]]] *qbs-morphism-comp*[**where** *Y*=*qbs-borel* $\bigotimes_Q$ *qbs-borel*] *qbs-morphism-space*[*OF qbs-morphism-space*[*OF qbs-morphism-map-prod*]]
standard-borel.qbs-morphism-measurable-intro[*of borel* :: *real measure*])
then interpret *qs*: *qbs-s-finite X* $\bigotimes_Q$ *Y map-prod* α γ $\circ$ *rr.from-real distr* (*return borel r* $\bigotimes_M$ *k r*) *borel rr.to-real* **.**
interpret *qs2*: *qbs-s-finite Y* γ *k r*
by(*auto simp*: *qbs-s-finite-def k.image-s-finite-measure in-Mx-def assms qbs-meas-def qbs-meas-axioms-def k.kernel-sets*)
interpret *pq*: *pair-qbs-s-finite X* $\bigotimes_Q$ *Y* $\lambda l.$ (α *r*, γ *l*) *k r map-prod* α γ $\circ$ *rr.from-real distr* (*return borel r* $\bigotimes_M$ *k r*) *borel rr.to-real*
by (*auto simp*: *pair-qbs-s-finite-def qs.qbs-s-finite-axioms qs2.strength-qbs-s-finite*[*OF qbs-Mx-to-X*[*OF assms*(*1*),*of r*]])
have [*measurable*]: *map-prod* α γ $\in$ *borel* $\bigotimes_M$ *borel* $\to_M$ *qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)
proof −
have *map-prod* α γ $\in$ *qbs-borel* $\bigotimes_Q$ *qbs-borel* $\to_Q$ *X* $\bigotimes_Q$ *Y*
using *assms* **by**(*auto intro*!: *qbs-morphism-map-prod simp*: *qbs-Mx-is-morphisms*)
also have ... $\subseteq$ *qbs-to-measure* (*qbs-borel* $\bigotimes_Q$ *qbs-borel*) $\to_M$ *qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)
by(*rule l-preserves-morphisms*)
also have ... = *borel* $\bigotimes_M$ *borel* $\to_M$ *qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)
using *rr.lr-sets-ident l-preserves-morphisms* **by**(*auto simp add*: *r-preserves-product*[*symmetric*])
finally show *?thesis* **.**
qed
show *?goal2*
unfolding *qs2.strength-qbs*[*OF qbs-Mx-to-X*[*OF assms*(*1*),*of r*] *fun-cong*[*OF assms*(*5*)]]
proof(*rule pq.qbs-meas-eqI*)
show *distr* (*k r*) (*qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)) ($\lambda l.$ (α *r*, γ *l*))
= *distr* (*distr* (*return borel r* $\bigotimes_M$ *k r*) *borel rr.to-real*) (*qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)) (*map-prod* α γ $\circ$ *rr.from-real*)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *distr* (*k r*) (*qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)) (*map-prod* α γ $\circ$ *Pair r*)
by(*simp add*: *comp-def*)
also have ... = *distr* (*distr* (*k r*) (*borel* $\bigotimes_M$ *borel*) (*Pair r*)) (*qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)) (*map-prod* α γ)
by(*auto intro*!: *distr-distr*[*symmetric*])
also have ... = *distr* (*return borel r* $\bigotimes_M$ *k r*) (*qbs-to-measure* (*X* $\bigotimes_Q$ *Y*)) (*map-prod* α γ)
proof −
have *return borel r* $\bigotimes_M$ *k r* = *distr* (*k r*) (*borel* $\bigotimes_M$ *borel*) ($\lambda l.$ (*r*,*l*))

```
        by(auto intro!: measure-eqI simp: sets-pair-measure-cong[OF refl 1(2)]
qs2.emeasure-pair-measure-alt'
                emeasure-distr nn-integral-return[OF - qs2.measurable-emeasure-Pair'])
      thus ?thesis by simp
    qed
    also have ... = ?rhs
      by(simp add: distr-distr comp-def)
    finally show ?thesis .
  qed
 qed
qed

lemma strength-qbs-morphism[qbs]: strength-qbs X Y ∈ X ⨂Q monadM-qbs Y
→Q monadM-qbs (X ⨂Q Y)
proof(rule pair-qbs-morphismI)
 fix α β
 assume h:α ∈ qbs-Mx X
        β ∈ qbs-Mx (monadM-qbs Y)
 from rep-qbs-Mx-monadM[OF this(2)] obtain γ k where hb:
   β = (λr. ⟦Y, γ, k r⟧meas) γ ∈ qbs-Mx Y s-finite-kernel borel borel k
   by metis
 have s-finite-kernel borel borel (λr. distr (return borel r ⨂M k r) borel rr.to-real)
   by(auto intro!: s-finite-kernel.distr-s-finite-kernel[where Y=borel ⨂M borel]
           s-finite-kernel-pair-measure[OF prob-kernel.s-finite-kernel-prob-kernel]
simp: hb prob-kernel-def')
 thus (λr. strength-qbs X Y (α r, β r)) ∈ qbs-Mx (monadM-qbs (X ⨂Q Y))
  using strength-qbs-ab-r[OF h hb(2,3,1)] strength-qbs-ab-r-s-finite[OF h hb(2,3,1)]
   by(auto simp: monadM-qbs-Mx qbs-s-finite-def qbs-meas-def in-Mx-def
       intro!: exI[where x=map-prod α γ ∘ rr.from-real] exI[where x=λr. distr
(return borel r ⨂M k r) borel rr.to-real])
qed

lemma bind-qbs-morphism[qbs]: (≫=) ∈ monadM-qbs X →Q (X ⇒Q monadM-qbs
Y) ⇒Q monadM-qbs Y
proof −
 {
  fix f s
  assume h[qbs]:f ∈ X →Q monadM-qbs Y s ∈ qbs-space (monadM-qbs X)
  from rep-qbs-space-monadM[OF this(2)] obtain α μ where h':
   s = ⟦X, α, μ⟧meas qbs-s-finite X α μ by metis
  then interpret qbs-s-finite X α μ by simp
  from rep-qbs-Mx-monadM[OF qbs-morphism-Mx[OF h(1) in-Mx]] obtain β k
    where hb:f ∘ α = (λr. ⟦Y, β, k r⟧meas) β ∈ qbs-Mx Y s-finite-kernel borel
borel k by metis
   have join-qbs (distr-qbs ((X ⇒Q monadM-qbs Y) ⨂Q X) (monadM-qbs Y)
(λfx. fst fx (snd fx)) (strength-qbs (X ⇒Q monadM-qbs Y) X (f, s))) = s ≫= f
    unfolding bind-qbs[OF h'(1) h(1) hb(2,3,1)] using hb
   by(intro join-qbs[OF qbs-s-finite.distr-qbs-s-finite[OF strength-qbs-s-finite[OF
h(1)]] qbs-meas.distr-qbs[OF strength-qbs-meas[OF h(1) h'(1)] strength-qbs[OF h(1)
```

$h'(1)]]]$)
(*auto simp*: *comp-def*)
}
thus *?thesis*
by(*auto intro*!: *arg-swap-morphism*[*OF curry-preserves-morphisms*[*OF qbs-morphism-cong'*[*of - join-qbs* ∘ (*distr-qbs* (*exp-qbs X* (*monadM-qbs Y*) ⨂$_Q$ *X*) (*monadM-qbs Y*) (λ*fx*. (*fst fx*) (*snd fx*))) ∘ (*strength-qbs* (*exp-qbs X* (*monadM-qbs Y*)) *X*)]]] *qbs-morphism-comp distr-qbs-morphism' strength-qbs-morphism join-qbs-morphism qbs-morphism-eval simp*: *pair-qbs-space*)
qed

lemma *strength-qbs-law1*:
x ∈ *qbs-space* (*unit-quasi-borel* ⨂$_Q$ *monadM-qbs X*)
⟹ *snd x* = (*distr-qbs* (*unit-quasi-borel* ⨂$_Q$ *X*) *X snd* ∘ *strength-qbs unit-quasi-borel X*) *x*
by(*intro strength-qbs-law1-all-meas*) (*auto simp*: *pair-qbs-space monadM-all-meas-space*)

lemma *strength-qbs-law2*:
x ∈ *qbs-space* ((*X* ⨂$_Q$ *Y*) ⨂$_Q$ *monadM-qbs Z*)
⟹ (*strength-qbs X* (*Y* ⨂$_Q$ *Z*) ∘ (*map-prod id* (*strength-qbs Y Z*)) ∘ (λ((*x*,*y*),*z*). (*x*,(*y*,*z*)))) *x* =
(*distr-qbs* ((*X* ⨂$_Q$ *Y*) ⨂$_Q$ *Z*) (*X* ⨂$_Q$ (*Y* ⨂$_Q$ *Z*)) (λ((*x*,*y*),*z*). (*x*,(*y*,*z*))) ∘ *strength-qbs* (*X* ⨂$_Q$ *Y*) *Z*) *x*
by(*intro strength-qbs-law2-all-meas*) (*auto simp*: *pair-qbs-space monadM-all-meas-space*)

lemma *strength-qbs-law3*:
assumes *x* ∈ *qbs-space* (*X* ⨂$_Q$ *Y*)
shows *return-qbs* (*X* ⨂$_Q$ *Y*) *x* = (*strength-qbs X Y* ∘ (*map-prod id* (*return-qbs Y*))) *x*
proof −
interpret *qp*: *qbs-prob Y* λ*r*. *snd x return borel 0*
using *assms* **by**(*auto simp*: *prob-space-return pair-qbs-space qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def*)
show *?thesis*
using *qp.strength-qbs*[*OF - qp.return-qbs*[*of snd x Y*],*of fst x X*] *qp.return-qbs*[*OF assms*] *assms*
by(*auto simp*: *pair-qbs-space*)
qed

lemma *strength-qbs-law4*:
assumes *x* ∈ *qbs-space* (*X* ⨂$_Q$ *monadM-qbs* (*monadM-qbs Y*))
shows (*strength-qbs X Y* ∘ *map-prod id join-qbs*) *x* = (*join-qbs* ∘ *distr-qbs* (*X* ⨂$_Q$ *monadM-qbs Y*) (*monadM-qbs* (*X* ⨂$_Q$ *Y*)) (*strength-qbs X Y*) ∘ *strength-qbs X* (*monadM-qbs Y*)) *x*
(**is** *?lhs* = *?rhs*)
proof −
have [*qbs*]:*fst x* ∈ *qbs-space X snd x* ∈ *qbs-space* (*monadM-qbs* (*monadM-qbs Y*))
using *assms* **by**(*auto simp*: *pair-qbs-space*)
from *assms rep-qbs-space-monadM*[*of snd x monadM-qbs Y*] **obtain** β μ

where *h*:*qbs-s-finite* (*monadM-qbs Y*) β μ *snd x* = $[\![$*monadM-qbs Y*, β, $\mu]\!]_{meas}$
by (*auto simp*: *pair-qbs-space*) *metis*
with *rep-qbs-Mx-monadM*[*of* β *Y*] **obtain** γ *k*
where *h'*: $\gamma \in$ *qbs-Mx Y s-finite-kernel borel borel k* β = ($\lambda r.$ $[\![Y, \gamma, k\ r]\!]_{meas}$)
and *h''*: $\bigwedge r.$ *qbs-s-finite Y* γ (*k r*)
by (*metis in-Mx.in-Mx qbs-meas-def qbs-s-finite-def*)
have *?lhs* = $[\![X \bigotimes_Q Y, \lambda r.$ (*fst x*, γ *r*), $\mu \gg=_k k]\!]_{meas}$
using *qbs-meas.strength-qbs*[*OF qbs-s-finite.axioms*(*1*)[*OF join-qbs-s-finite*[*OF h h'*]] - *join-qbs*[*OF h h'*],*of fst x X*] *assms*
by(*auto simp*: *pair-qbs-space*)
also have ... = *?rhs*
using *qbs-meas.strength-qbs*[**where** *W*=*X*,*OF qbs-s-finite.axioms*(*1*)[*OF h''*] - *fun-cong*[*OF h'*(*3*)]]
by(*auto intro*!: *qbs-meas.strength-qbs*[**where** *w*=*fst x* **and** *sx* =*snd x*,*simplified*] *join-qbs*[*symmetric*,**where** β=*strength-qbs X Y* $\circ$ ($\lambda r.$ (*fst x*, β *r*))]
qbs-meas.distr-qbs qbs-s-finite.distr-qbs-s-finite[**where** *X*=(*X* $\bigotimes_Q$ *monadM-qbs Y*)] *qbs-s-finite.strength-qbs-s-finite h*
qbs-s-finite.axioms(*1*) *pair-qbs-MxI h'*(*1*,*2*) *simp*: *comp-def*)
finally show *?thesis* **.**
qed

lemma *distr-qbs-morphism*[*qbs*]: *distr-qbs X Y* $\in$ (*X* $\Rightarrow_Q$ *Y*) $\rightarrow_Q$ (*monadM-qbs X* $\Rightarrow_Q$ *monadM-qbs Y*)
proof −
have [*simp*]: *distr-qbs X Y* = (λf *sx*. *sx* $\gg=$ *return-qbs Y* $\circ$ *f*)
by *standard*+ (*auto simp*: *distr-qbs-def*)
show *?thesis*
by *simp*
qed

lemma
assumes $\alpha \in$ *qbs-Mx X* $\beta \in$ *qbs-Mx Y*
shows *return-qbs-pair-Mx*: *return-qbs* (*X* $\bigotimes_Q$ *Y*) (α *r*, β *k*) = $[\![X \bigotimes_Q Y$,*map-prod* α β $\circ$ *rr.from-real*, *distr* (*return borel r* $\bigotimes_M$ *return borel k*) *borel rr.to-real*$]\!]_{meas}$
and *return-qbs-pair-Mx-prob*: *qbs-prob* (*X* $\bigotimes_Q$ *Y*) (*map-prod* α β $\circ$ *rr.from-real*) (*distr* (*return borel r* $\bigotimes_M$ *return borel k*) *borel rr.to-real*)
proof −
note [*measurable-cong*] = *sets-return*[*of borel*]
interpret *qp*: *qbs-prob X* $\bigotimes_Q$ *Y map-prod* α β $\circ$ *rr.from-real distr* (*return borel r* $\bigotimes_M$ *return borel k*) *borel rr.to-real*
using *qbs-closed1-dest*[*OF assms*(*1*)] *qbs-closed1-dest*[*OF assms*(*2*)]
by(*auto intro*!: *prob-space.prob-space-distr prob-space-pair simp*: *comp-def prob-space-return pair-qbs-Mx qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def*)
show *qbs-prob* (*X* $\bigotimes_Q$ *Y*) (*map-prod* α β $\circ$ *rr.from-real*) (*distr* (*return borel r* $\bigotimes_M$ *return borel k*) *borel rr.to-real*)
by *standard*
show *return-qbs* (*X* $\bigotimes_Q$ *Y*) (α *r*, β *k*) = $[\![X \bigotimes_Q Y$,*map-prod* α β $\circ$ *rr.from-real*, *distr* (*return borel r* $\bigotimes_M$ *return borel k*) *borel rr.to-real*$]\!]_{meas}$ (**is** *?lhs* = *?rhs*)
proof −

have *?lhs* = (*strength-qbs X Y* ∘ *map-prod id* (*return-qbs Y*)) (*α r*, *β k*)
by(*rule strength-qbs-law3*[*of* (*α r*, *β k*) *X Y*], *insert assms*) (*auto simp*: *qbs-Mx-to-X pair-qbs-space*)
also have ... = *strength-qbs X Y* (*α r*, ⟦*Y*, *β*, *return borel k*⟧$_{meas}$)
using *fun-cong*[*OF return-qbs-comp*[*OF assms*(*2*)]] **by** *simp*
also have ... = *?rhs*
by(*rule strength-qbs-ab-r*[*OF assms*(*1*) - *assms*(*2*)])
(*auto intro*!: *qbs-closed2-dest qbs-s-finite.in-space-monadM*
s-finite-measure.s-finite-kernel-const[*of return borel k*,*simplified s-finite-kernel-cong-sets*[*OF - sets-return*]]
prob-space.s-finite-measure-prob
simp: *qbs-s-finite-def in-Mx-def qbs-meas-def qbs-meas-axioms-def assms*(*2*) *prob-space-return*)
finally show *?thesis* **.**
qed
qed

lemma *bind-bind-return-distr*:
assumes *s-finite-measure μ*
and *s-finite-measure ν*
and [*measurable-cong*]: *sets μ* = *sets borel sets ν* = *sets borel*
shows *μ* ⤜$_k$ (*λr*. *ν* ⤜$_k$ (*λl*. *distr* (*return borel r* ⨂$_M$ *return borel l*) *borel rr.to-real*))
= *distr* (*μ* ⨂$_M$ *ν*) *borel rr.to-real*
(**is** *?lhs* = *?rhs*)
proof −
interpret *rd1*: *s-finite-measure μ* **by** *fact*
interpret *rd2*: *s-finite-measure ν* **by** *fact*
have *ne*: *space μ* ≠ {} *space ν* ≠ {}
by(*auto simp*: *sets-eq-imp-space-eq assms*(*3*,*4*))

have *?lhs* = *μ* ⤜$_k$ (*λr*. *ν* ⤜$_k$ (*λl*. *distr* (*return* (*borel* ⨂$_M$ *borel*) (*r*,*l*)) *borel rr.to-real*))
by(*simp add*: *pair-measure-return*)
also have ... = *μ* ⤜$_k$ (*λr*. *ν* ⤜$_k$ (*λl*. *distr* (*return* (*μ* ⨂$_M$ *ν*) (*r*, *l*)) *borel rr.to-real*))
proof −
have *return* (*borel* ⨂$_M$ *borel*) = *return* (*μ* ⨂$_M$ *ν*)
by(*auto intro*!: *return-sets-cong sets-pair-measure-cong simp*: *assms*(*3*,*4*))
thus *?thesis* **by** *simp*
qed
also have ... = *μ* ⤜$_k$ (*λr*. *distr* (*ν* ⤜$_k$ (*λl*. (*return* (*μ* ⨂$_M$ *ν*) (*r*, *l*)))) *borel rr.to-real*)
by(*auto intro*!: *bind-kernel-cong-All measure-kernel.distr-bind-kernel*[*of ν μ* ⨂$_M$ *ν*,*symmetric*] *simp*: *ne measure-kernel-def space-pair-measure*)
also have ... = *distr* (*μ* ⤜$_k$ (*λr*. *ν* ⤜$_k$ (*λl*. *return* (*μ* ⨂$_M$ *ν*) (*r*, *l*)))) *borel rr.to-real*
by(*auto intro*!: *measure-kernel.distr-bind-kernel*[*of μ μ* ⨂$_M$ *ν*,*symmetric*] *s-finite-kernel.axioms*(*1*)

s-finite-kernel.bind-kernel-s-finite-kernel'[**where** $Y{=}\nu$] *s-finite-measure.s-finite-kernel-const*[*OF assms*(*2*)]
prob-kernel.s-finite-kernel-prob-kernel[*of* $\mu \bigotimes_M \nu$] *simp*: *ne prob-kernel-def'*)
also have ... = *?rhs*
by(*simp add*: *pair-measure-eq-bind-s-finite*[*OF assms*(*1*,*2*),*symmetric*])
finally show *?thesis* **.**
qed

end

context
begin
interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

lemma *from-real-rr-qbs-morphism*[*qbs*]: *rr.from-real* ∈ *qbs-borel* →$_Q$ *qbs-borel* $\bigotimes_Q$ *qbs-borel*
by (*metis borel-prod qbs-Mx-R qbs-Mx-is-morphisms qbs-borel-prod rr.from-real-measurable*)

end

context *pair-qbs-s-finites*
begin

interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

sublocale *qbs-s-finite X* $\bigotimes_Q$ *Y map-prod* α β ∘ *rr.from-real distr* ($\mu \bigotimes_M \nu$) *borel rr.to-real*
by(*auto simp*: *qbs-s-finite-def in-Mx-def qbs-meas-def qbs-meas-axioms-def*
qbs-Mx-is-morphisms pq1.s-finite-measure-axioms pq2.s-finite-measure-axioms
intro!: *s-finite-measure.s-finite-measure-distr*[*OF pair-measure-s-finite-measure*])

lemma *qbs-bind-bind-return-qp*:
⟦*Y*,β,ν⟧$_{meas}$ ⤜ (λ*y*. ⟦*X*,α,μ⟧$_{meas}$ ⤜ (λ*x*. *return-qbs* (*X* $\bigotimes_Q$ *Y*) (*x*,*y*))) = ⟦*X* $\bigotimes_Q$ *Y*, *map-prod* α β ∘ *rr.from-real*, *distr* ($\mu \bigotimes_M \nu$) *borel rr.to-real*⟧$_{meas}$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ⟦*X* $\bigotimes_Q$ *Y*, *map-prod* α β ∘ *rr.from-real*, ν ⤜$_k$ (λ*l*. μ ⤜$_k$ (λ*r*. *distr* (*return borel r* $\bigotimes_M$ *return borel l*) *borel rr.to-real*))⟧$_{meas}$
by(*auto intro*!: *pq2.bind-qbs*[*OF refl*] *s-finite-kernel.bind-kernel-s-finite-kernel'*[**where** *Y*=μ]
s-finite-measure.s-finite-kernel-const s-finite-kernel.distr-s-finite-kernel[**where** *Y*=*borel* $\bigotimes_M$ *borel*]
prob-kernel.s-finite-kernel-prob-kernel[*of borel* $\bigotimes_M$ μ]
simp: *sets-eq-imp-space-eq*[*OF pq1.mu-sets*] *pq1.s-finite-measure-axioms split-beta' pair-measure-return*[*of - snd -*] *prob-kernel-def'*)
(*auto intro*!: *pq1.bind-qbs prob-kernel.s-finite-kernel-prob-kernel*

simp: *comp-def return-qbs-pair-Mx qbs-Mx-is-morphisms prob-kernel-def'*)
also have ... = *?rhs*
proof −
have $\nu \gg\!=_k (\lambda l.\ \mu \gg\!=_k (\lambda r.\ distr\ (return\ borel\ r \bigotimes_M return\ borel\ l)\ borel\ rr.to\text{-}real)) = distr\ (\mu \bigotimes_M \nu)\ borel\ rr.to\text{-}real$
by(*auto simp*: *bind-bind-return-distr*[*OF pq1.s-finite-measure-axioms pq2.s-finite-measure-axioms pq1.mu-sets pq2.mu-sets*,*symmetric*] *pq1.s-finite-measure-axioms pq2.s-finite-measure-axioms prob-kernel-def' intro*!: *bind-kernel-rotate*[**where** *Z*=*borel*] *prob-kernel.s-finite-kernel-prob-kernel*)
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **.**
qed

lemma *qbs-bind-bind-return-pq*:
$[\![X,\alpha,\mu]\!]_{meas} \gg\!= (\lambda x.\ [\![Y,\beta,\nu]\!]_{meas} \gg\!= (\lambda y.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y))) = [\![X \bigotimes_Q Y,\ map\text{-}prod\ \alpha\ \beta \circ rr.from\text{-}real,\ distr\ (\mu \bigotimes_M \nu)\ borel\ rr.to\text{-}real]\!]_{meas}$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $[\![X \bigotimes_Q Y,\ map\text{-}prod\ \alpha\ \beta \circ rr.from\text{-}real,\ \mu \gg\!=_k (\lambda r.\ \nu \gg\!=_k (\lambda l.\ distr\ (return\ borel\ r \bigotimes_M return\ borel\ l)\ borel\ rr.to\text{-}real))]\!]_{meas}$
by(*auto intro*!: *pq1.bind-qbs*[*OF refl*]*s-finite-kernel.bind-kernel-s-finite-kernel'*[**where** *Y*=ν]
s-finite-measure.s-finite-kernel-const s-finite-kernel.distr-s-finite-kernel[**where** *Y*=*borel* $\bigotimes_M$ *borel*]
prob-kernel.s-finite-kernel-prob-kernel[*of borel* $\bigotimes_M$ ν]
simp: *sets-eq-imp-space-eq*[*OF pq2.mu-sets*] *pq2.s-finite-measure-axioms split-beta' pair-measure-return*[*of - fst -*] *prob-kernel-def'*)
(*auto intro*!: *pq2.bind-qbs prob-kernel.s-finite-kernel-prob-kernel*
simp: *comp-def return-qbs-pair-Mx qbs-Mx-is-morphisms prob-kernel-def'*)
also have ... = *?rhs*
by(*simp add*: *bind-bind-return-distr*[*OF pq1.s-finite-measure-axioms pq2.s-finite-measure-axioms pq1.mu-sets pq2.mu-sets*])
finally show *?thesis* **.**
qed

end

lemma *bind-qbs-return-rotate*:
assumes $p \in qbs\text{-}space\ (monadM\text{-}qbs\ X)$
and $q \in qbs\text{-}space\ (monadM\text{-}qbs\ Y)$
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y))) = p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y)))$
proof −
from *rep-qbs-space-monadM*[*OF assms*(*1*)] *rep-qbs-space-monadM*[*OF assms*(*2*)]
obtain $\alpha\ \mu\ \beta\ \nu$ **where** *h*: $p = [\![X,\ \alpha,\ \mu]\!]_{meas}$ $q = [\![Y,\ \beta,\ \nu]\!]_{meas}$ *qbs-s-finite X* $\alpha\ \mu$ *qbs-s-finite Y* $\beta\ \nu$
by *metis*
then interpret *pair-qbs-s-finites X* $\alpha\ \mu$ *Y* $\beta\ \nu$
by(*simp add*: *pair-qbs-s-finites-def*)

show *?thesis*
by(*simp add*: *h*(*1*,*2*) *qbs-bind-bind-return-pq qbs-bind-bind-return-qp*)
qed

lemma *qbs-bind-bind-return1*:
assumes [*qbs*]: $f \in X \bigotimes_Q Y \to_Q$ *monadM-qbs Z*
$p \in$ *qbs-space* (*monadM-qbs X*)
$q \in$ *qbs-space* (*monadM-qbs Y*)
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) = (q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y)))) \gg\!= f$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* $= q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y) \gg\!= f))$
by(*auto intro*!: *bind-qbs-cong*[*OF assms*(*3*),**where** *Y*=*Z*] *bind-qbs-cong*[*OF assms*(*2*),**where** *Y*=*Z*]
simp: *bind-qbs-return*[*OF assms*(*1*),*simplified pair-qbs-space*])
also have ... $= q \gg\!= (\lambda y.\ (p \gg\!= (\lambda x.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y))) \gg\!= f)$
by(*auto intro*!: *bind-qbs-cong*[*OF assms*(*3*),**where** *Y*=*Z*] *bind-qbs-assoc*[*OF assms*(*2*) - *assms*(*1*)] *simp*:)
also have ... = *?rhs*
by(*simp add*: *bind-qbs-assoc*[*OF assms*(*3*) - *assms*(*1*)])
finally show *?thesis* **.**
qed

lemma *qbs-bind-bind-return2*:
assumes [*qbs*]:$f \in X \bigotimes_Q Y \to_Q$ *monadM-qbs Z*
$p \in$ *qbs-space* (*monadM-qbs X*) $q \in$ *qbs-space* (*monadM-qbs Y*)
shows $p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ (x,y))) = (p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y)))) \gg\!= f$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* $= p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y) \gg\!= f))$
by(*auto intro*!: *bind-qbs-cong*[*OF assms*(*2*),**where** *Y*=*Z*] *bind-qbs-cong*[*OF assms*(*3*),**where** *Y*=*Z*] *simp*: *bind-qbs-return*[*OF assms*(*1*),*simplified pair-qbs-space*])
also have ... $= p \gg\!= (\lambda x.\ (q \gg\!= (\lambda y.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y))) \gg\!= f)$
by(*auto intro*!: *bind-qbs-cong*[*OF assms*(*2*),**where** *Y*=*Z*] *bind-qbs-assoc*[*OF assms*(*3*) - *assms*(*1*)])
also have ... = *?rhs*
by(*simp add*: *bind-qbs-assoc*[*OF assms*(*2*) - *assms*(*1*)])
finally show *?thesis* **.**
qed

corollary *bind-qbs-rotate*:
assumes $f \in X \bigotimes_Q Y \to_Q$ *monadM-qbs Z*
$p \in$ *qbs-space* (*monadM-qbs X*)
and $q \in$ *qbs-space* (*monadM-qbs Y*)
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) = p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ (x,y)))$
by(*simp add*: *qbs-bind-bind-return1*[*OF assms*] *qbs-bind-bind-return2*[*OF assms*] *bind-qbs-return-rotate assms*)

context *pair-qbs-s-finites*
begin

interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

lemma
assumes [*qbs*]:$f \in X \bigotimes_Q Y \to_Q Z$
shows *qbs-bind-bind-return*:$[\![X,\alpha,\mu]\!]_{meas} \gg\!= (\lambda x.\ [\![Y,\beta,\nu]\!]_{meas} \gg\!= (\lambda y.\ return\text{-}qbs\ Z\ (f\ (x,y)))) = [\![Z,\ f \circ (map\text{-}prod\ \alpha\ \beta \circ rr.from\text{-}real),\ distr\ (\mu \bigotimes_M \nu)\ borel\ rr.to\text{-}real]\!]_{meas}$ (**is** *?lhs* = *?rhs*)
and *qbs-bind-bind-return-s-finite*: *qbs-s-finite Z* (*f* ∘ (*map-prod α β* ∘ *rr.from-real*)) (*distr* (*μ* $\bigotimes_M$ *ν*) *borel rr.to-real*)
proof −
show *qbs-s-finite Z* (*f* ∘ (*map-prod α β* ∘ *rr.from-real*)) (*distr* (*μ* $\bigotimes_M$ *ν*) *borel rr.to-real*)
using *qbs-s-finite-axioms* **by**(*auto simp*: *qbs-s-finite-def qbs-meas-def in-Mx-def qbs-Mx-is-morphisms*)
have *?lhs* = $[\![X,\alpha,\mu]\!]_{meas} \gg\!= (\lambda x.\ [\![Y,\beta,\nu]\!]_{meas} \gg\!= (\lambda y.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y))) \gg\!= return\text{-}qbs\ Z \circ f$
by(*auto simp*: *comp-def intro*!: *qbs-bind-bind-return2*[*of return-qbs Z* ∘ *f* - - *Z*,*simplified comp-def*])
also have ... = $[\![X \bigotimes_Q Y,\ map\text{-}prod\ \alpha\ \beta \circ rr.from\text{-}real,\ distr\ (\mu \bigotimes_M \nu)\ borel\ rr.to\text{-}real]\!]_{meas} \gg\!= return\text{-}qbs\ Z \circ f$
by(*simp add*: *qbs-bind-bind-return-pq*)
also have ... = *?rhs*
by(*rule distr-qbs*[*OF refl assms*,*simplified distr-qbs-def*])
finally show *?lhs* = *?rhs* **.**
qed

end

4.1.10 The Probability Monad

definition *monadP-qbs X* ≡ *sub-qbs* (*monadM-qbs X*) {*s*. *prob-space* (*qbs-l s*)}

lemma *monadP-qbs-def2*: *monadP-qbs X* = *sub-qbs* (*all-meas-qbs X*) {*s*. *prob-space* (*qbs-l s*)}
unfolding *monadP-qbs-def*
proof(*safe intro*!: *qbs-eqI*)
fix *γ*
assume *h*:*γ* ∈ *qbs-Mx* (*sub-qbs* (*all-meas-qbs X*) {*s*. *prob-space* (*qbs-l s*)})
then obtain *α k* **where** *h'*:*γ* = (*λr*. $[\![X,\ \alpha,\ k\ r]\!]_{meas}$) *α* ∈ *qbs-Mx X measure-kernel borel borel k* ⋀*r*. *qbs-meas X α* (*k r*)
using *rep-all-meas-qbs-Mx*[*of* - *X*] **by**(*auto simp*: *sub-qbs-Mx*)
then interpret *qbs-meas X α k r* **for** *r*
by *simp*
have ⋀*r*. *prob-space* (*k r*)

using h **by**(*auto simp*: *sub-qbs-Mx qbs-l qbs-meas-def h'(1) intro*!: *prob-space-distrD*[*of* α - *qbs-to-measure X*])
then interpret *prob-kernel borel borel k*
by(*auto simp*: *prob-kernel-def prob-kernel-axioms-def h'(3)*)
show $\gamma \in$ *qbs-Mx* (*sub-qbs* (*monadM-qbs X*) {*s*. *prob-space* (*qbs-l s*)})
using *h* **by**(*auto simp*: *sub-qbs-Mx monadM-qbs-Mx h'(1) intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=*k*] *s-finite-kernel-axioms*)
qed(*use monadM-all-meas-Mx* **in** *auto simp*: *sub-qbs-Mx*)

lemma
shows *qbs-space-monadPM*: $s \in$ *qbs-space* (*monadP-qbs X*) $\Longrightarrow$ $s \in$ *qbs-space* (*monadM-qbs X*)
and *qbs-Mx-monadPM*: $f \in$ *qbs-Mx* (*monadP-qbs X*) $\Longrightarrow$ $f \in$ *qbs-Mx* (*monadM-qbs X*)
by(*simp-all add*: *monadP-qbs-def sub-qbs-space sub-qbs-Mx*)

lemma *monadP-qbs-space*: *qbs-space* (*monadP-qbs X*) = {*s*. *qbs-space-of s* = *X* $\wedge$ *prob-space* (*qbs-l s*)}
by(*auto simp*: *monadP-qbs-def2 sub-qbs-space all-meas-qbs-space*)

lemma *rep-qbs-space-monadP*:
assumes $s \in$ *qbs-space* (*monadP-qbs X*)
obtains α μ **where** $s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-prob X* α μ
proof −
obtain α μ **where** *h*:$s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-s-finite X* α μ
using *assms rep-qbs-space-monadM*[*of s X*] **by**(*auto simp*: *monadP-qbs-def sub-qbs-space*)
interpret *qbs-s-finite X* α μ **by** *fact*
have *prob-space* μ
by(*rule prob-space-distrD*[*of* α - *qbs-to-measure X*]) (*insert assms*, *auto simp*: *qbs-l*[*symmetric*] *h*(*1*)[*symmetric*] *monadP-qbs-space*)
thus *?thesis*
by (*simp add*: *h*(*1*) *in-Mx-axioms mu-sets qbs-prob.intro real-distribution-axioms-def real-distribution-def that*)
qed

lemma *qbs-l-prob-space*:
s $\in$ *qbs-space* (*monadP-qbs X*) $\Longrightarrow$ *prob-space* (*qbs-l s*)
by(*auto simp*: *monadP-qbs-space*)

lemma *monadP-qbs-empty-iff*:
(*qbs-space X* = {}) = (*qbs-space* (*monadP-qbs X*) = {})
proof
show *qbs-space X* = {} $\Longrightarrow$ *qbs-space* (*monadP-qbs X*) = {}
using *qbs-space-of-non-empty* **by**(*auto simp add*: *monadP-qbs-space*)
next
assume *qbs-space* (*monadP-qbs X*) = {}
then have *h*:$\bigwedge s$. *qbs-space-of s* = *X* $\Longrightarrow$ $\neg$ *prob-space* (*qbs-l s*)
by(*simp add*: *monadP-qbs-space*)

show *qbs-space X* = {}
proof(*rule ccontr*)
assume *qbs-space X* ≠ {}
then obtain *a* **where** *a*:*a* ∈ *qbs-Mx X* **by** (*auto simp*: *qbs-empty-equiv*)
then interpret *qbs-prob X a return borel 0*
by(*auto simp*: *qbs-prob-def in-Mx-def real-distribution-axioms-def real-distribution-def prob-space-return*)
have *qbs-space-of* ⟦*X*, *a*, *return borel 0*⟧$_{meas}$ = *X prob-space* (*qbs-l* ⟦*X*, *a*, *return borel 0*⟧$_{meas}$)
by(*auto simp*: *qbs-l intro*!: *prob-space-distr*)
with *h* **show** *False* **by** *simp*
qed
qed

lemma *in-space-monadP-qbs-pred*: *qbs-pred* (*monadM-qbs X*) (λ*s*. *s* ∈ *monadP-qbs X*)
by(*rule qbs-morphism-cong'*[**where** *f*=λ*s*. *prob-space* (*qbs-l s*)],*auto simp*: *qbs-l-prob-pred*) (*auto simp*: *monadP-qbs-def sub-qbs-space*)

lemma(**in** *qbs-prob*) *in-space-monadP*[*qbs*]: ⟦*X*, α, μ⟧$_{meas}$ ∈ *qbs-space* (*monadP-qbs X*)
by(*auto simp*: *monadP-qbs-space qbs-l prob-space-distr*)

lemma *qbs-morphism-monadPD*: *f* ∈ *X* →$_Q$ *monadP-qbs Y* ⟹ *f* ∈ *X* →$_Q$ *monadM-qbs Y*
unfolding *monadP-qbs-def* **by**(*rule qbs-morphism-subD*)

lemma *qbs-morphism-monadPD'*: *f* ∈ *monadM-qbs X* →$_Q$ *Y* ⟹ *f* ∈ *monadP-qbs X* →$_Q$ *Y*
unfolding *monadP-qbs-def* **by**(*rule qbs-morphism-subI2*)

lemma *qbs-morphism-monadPI*:
assumes ⋀*x*. *x* ∈ *qbs-space X* ⟹ *prob-space* (*qbs-l* (*f x*)) *f* ∈ *X* →$_Q$ *monadM-qbs Y*
shows *f* ∈ *X* →$_Q$ *monadP-qbs Y*
using *assms* **by**(*auto simp*: *monadP-qbs-def intro*!:*qbs-morphism-subI1*)

lemma *qbs-morphism-monadPI'*:
assumes ⋀*x*. *x* ∈ *qbs-space X* ⟹ *f x* ∈ *qbs-space* (*monadP-qbs Y*) *f* ∈ *X* →$_Q$ *monadM-qbs Y*
shows *f* ∈ *X* →$_Q$ *monadP-qbs Y*
using *assms* **by**(*auto intro*!: *qbs-morphism-monadPI simp*: *monadP-qbs-space*)

lemma *qbs-morphism-monadPI''*:
assumes *f* ∈ *monadM-qbs X* →$_Q$ *monadM-qbs Y* ⋀*s*. *s* ∈ *qbs-space* (*monadP-qbs X*) ⟹ *f s* ∈ *qbs-space* (*monadP-qbs Y*)
shows *f* ∈ *monadP-qbs X* →$_Q$ *monadP-qbs Y*
unfolding *monadP-qbs-def* **using** *assms*
by(*auto intro*!: *qbs-morphism-subsubI simp*: *monadP-qbs-space qbs-space-of-in*)

lemma *monadP-qbs-Mx*: *qbs-Mx* (*monadP-qbs X*) = {$\lambda r.$ $[\![X,\ \alpha,\ k\ r]\!]_{meas}$ |α *k*. α $\in$ *qbs-Mx X* $\wedge$ *k* $\in$ *borel* $\rightarrow_M$ *prob-algebra borel*}
proof *safe*
fix γ
assume *h*:γ $\in$ *qbs-Mx* (*monadP-qbs X*)
then obtain α *k* **where** *h1*:
γ = ($\lambda r.$ $[\![X,\ \alpha,\ k\ r]\!]_{meas}$) α $\in$ *qbs-Mx X s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite X* α (*k r*)
using *rep-qbs-Mx-monadM*[*of* γ *X*] **by**(*auto simp add*: *monadP-qbs-def sub-qbs-Mx*)
interpret *s-finite-kernel borel borel k* **by** *fact*
interpret *qs*: *qbs-s-finite X* α *k r* **for** *r*
by *fact*
have $\bigwedge r.$ *prob-space* (*k r*)
using *h* **by**(*auto intro*!: *prob-space-distrD*[*of* α - *qbs-to-measure X*] *simp*: *h1*(*1*) *monadP-qbs-def sub-qbs-Mx qs.qbs-l*)
hence *prob-kernel borel borel k*
by(*auto simp*: *prob-kernel-def prob-kernel-axioms-def measure-kernel-axioms*)
with *h1*(*1*,*2*) **show** $\exists \alpha$ *k*. γ = ($\lambda r.$ $[\![X,\ \alpha,\ k\ r]\!]_{meas}$) $\wedge$ α $\in$ *qbs-Mx X* $\wedge$ *k* $\in$ *borel* $\rightarrow_M$ *prob-algebra borel*
by(*auto intro*!: *exI*[**where** *x*=α] *exI*[**where** *x*=*k*] *simp*: *prob-kernel-def'*)
next
fix α **and** *k* :: *real* $\Rightarrow$ *real measure*
assume *h*:α $\in$ *qbs-Mx X k* $\in$ *borel* $\rightarrow_M$ *prob-algebra borel*
then interpret *pk*: *prob-kernel borel borel k*
by(*simp add*: *prob-kernel-def'*[*symmetric*])
have *qp*: *qbs-prob X* α (*k r*) **for** *r*
using *h* **by**(*auto simp*: *qbs-prob-def in-Mx-def pk.kernel-sets pk.prob-spaces real-distribution-axioms-def real-distribution-def*)
show ($\lambda r.$ $[\![X,\ \alpha,\ k\ r]\!]_{meas}$) $\in$ *qbs-Mx* (*monadP-qbs X*)
using *h*(*1*) *qp*
by(*auto simp*: *monadP-qbs-def sub-qbs-Mx monadM-qbs-space qbs-meas.qbs-l*[*OF qbs-prob.qbs-meas*[*OF qp*]] *monadM-qbs-Mx qbs-prob-def real-distribution-def*
intro!: *exI*[**where** *x*=α] *exI*[**where** *x*=*k*] *h pk.s-finite-kernel-axioms prob-space.prob-space-distr*)
qed

lemma *rep-qbs-Mx-monadP*:
assumes γ $\in$ *qbs-Mx* (*monadP-qbs X*)
obtains α *k* **where** γ = ($\lambda r.$ $[\![X,\ \alpha,\ k\ r]\!]_{meas}$) α $\in$ *qbs-Mx X k* $\in$ *borel* $\rightarrow_M$ *prob-algebra borel* $\bigwedge r.$ *qbs-prob X* α (*k r*)
proof −
have $\bigwedge \alpha$ *r k*. α $\in$ *qbs-Mx X* $\Longrightarrow$ *k* $\in$ *borel* $\rightarrow_M$ *prob-algebra borel* $\Longrightarrow$ *qbs-prob X* α (*k r*)
by(*auto simp*: *qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def prob-kernel-def'*[*symmetric*] *prob-kernel-def prob-kernel-axioms-def measure-kernel-def*)
thus *?thesis*
using *assms that* **by**(*fastforce simp*: *monadP-qbs-Mx*)

qed

lemma *qbs-l-monadP-le1*:*s* ∈ *qbs-space* (*monadP-qbs X*) ⟹ *qbs-l s A* ≤ *1*
by(*auto simp*: *monadP-qbs-space intro*!: *prob-space.emeasure-le-1*)

lemma *qbs-l-inj-P*: *inj-on qbs-l* (*qbs-space* (*monadP-qbs X*))
by(*auto intro*!: *inj-on-subset*[*OF qbs-l-inj*] *simp*: *monadP-qbs-def sub-qbs-space*)

lemma *qbs-l-measurable-prob*[*measurable*]:*qbs-l* ∈ *qbs-to-measure* (*monadP-qbs X*) →$_M$ *prob-algebra* (*qbs-to-measure X*)
proof(*rule qbs-morphism-dest*[*OF qbs-morphismI*])
fix *γ*
assume *γ* ∈ *qbs-Mx* (*monadP-qbs X*)
from *rep-qbs-Mx-monadP*[*OF this*] **obtain** *α k* **where** *h*[*measurable*]:
γ = (*λr*. ⟦*X*, *α*, *k r*⟧$_{meas}$) *α* ∈ *qbs-Mx X k* ∈ *borel* →$_M$ *prob-algebra borel* ⋀*r*. *qbs-prob X α* (*k r*)
by *metis*
show *qbs-l* ∘ *γ* ∈ *qbs-Mx* (*measure-to-qbs* (*prob-algebra* (*qbs-to-measure X*)))
by(*auto simp*: *qbs-Mx-R comp-def h*(*1*) *qbs-meas.qbs-l*[*OF qbs-prob.qbs-meas*[*OF h*(*4*)]])
qed

lemma *return-qbs-morphismP*: *return-qbs X* ∈ *X* →$_Q$ *monadP-qbs X*
proof(*rule qbs-morphismI*)
interpret *rr* : *real-distribution return borel 0*
by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-return*)
fix *α*
assume *h*:*α* ∈ *qbs-Mx X*
then have *1*:*return-qbs X* ∘ *α* = (*λr*. ⟦*X*, *α*, *return borel r*⟧$_{meas}$)
by(*rule return-qbs-comp*)
show *return-qbs X* ∘ *α* ∈ *qbs-Mx* (*monadP-qbs X*)
by(*auto simp*: *1 monadP-qbs-Mx h intro*!: *exI*[**where** *x*=*α*] *exI*[**where** *x*=*return borel*])
qed

lemma(**in** *qbs-prob*)
assumes *s* = ⟦*X*, *α*, *μ*⟧$_{meas}$
f ∈ *X* →$_Q$ *monadP-qbs Y*
β ∈ *qbs-Mx Y*
and *g*[*measurable*]:*g* ∈ *borel* →$_M$ *prob-algebra borel*
and (*f* ∘ *α*) = (*λr*. ⟦*Y*, *β*, *g r*⟧$_{meas}$)
shows *bind-qbs-prob*:*qbs-prob Y β* (*μ* ⤜ *g*)
and *bind-qbs*′: *s* ⤜ *f* = ⟦*Y*, *β*, *μ* ⤜ *g*⟧$_{meas}$
proof −
interpret *prob-kernel borel borel g*
using *assms*(*4*) **by**(*simp add*: *prob-kernel-def*′)
have *prob-space* (*μ* ⤜ *g*)
by(*auto intro*!: *prob-space-bind*′[*OF - g*] *simp*: *space-prob-algebra prob-space-axioms*)
thus *qbs-prob Y β* (*μ* ⤜ *g*) *s* ⤜ *f* = ⟦*Y*, *β*, *μ* ⤜ *g*⟧$_{meas}$

using *bind-qbs-meas*[*OF assms*(*1*) *qbs-morphism-monadAD*[*OF qbs-morphism-monadPD*[*OF assms*(*2*)]] *assms*(*3*) *prob-kernel.axioms*(*1*) *assms*(*5*)]
by(*auto simp*: *bind-qbs*[*OF assms*(*1*) *qbs-morphism-monadPD*[*OF assms*(*2*)] *assms*(*3*) *s-finite-kernel-axioms assms*(*5*)]
bind-kernel-bind[*of g μ borel*] *prob-kernel-def' intro*!: *qbs-meas.qbs-probI*)
qed

lemma *bind-qbs-morphism'P*:
assumes $f \in X \to_Q$ *monadP-qbs Y*
shows $(\lambda x.\ x \gg= f) \in$ *monadP-qbs X* $\to_Q$ *monadP-qbs Y*
proof(*safe intro*!: *qbs-morphism-monadPI'*)
fix *x*
assume $x \in$ *qbs-space* (*monadP-qbs X*)
from *rep-qbs-space-monadP*[*OF this*] **obtain** $\alpha\ \mu$ **where** *h*:$x = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-prob X α μ*
by *metis*
then interpret *qbs-prob X α μ* **by** *simp*
from *rep-qbs-Mx-monadP*[*OF qbs-morphism-Mx*[*OF assms in-Mx*]] **obtain** $\beta\ g$ **where** *h'*[*measurable*]:
$f \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ g\ r]\!]_{meas})\ \beta \in$ *qbs-Mx Y g* $\in$ *borel* $\to_M$ *prob-algebra borel*
by *metis*
show $x \gg= f \in$ *qbs-space* (*monadP-qbs Y*)
using *sets-bind*[*of μ g*] *measurable-space*[*OF h'*(*3*),*simplified space-prob-algebra*]
by(*auto simp*: *qbs-prob.bind-qbs'*[*OF h*(*2*,*1*) *assms h'*(*2*,*3*,*1*)] *qbs-prob-def in-Mx-def h'*(*2*) *real-distribution-def real-distribution-axioms-def intro*!: *qbs-prob.in-space-monadP prob-space-bind*[**where** *S*=*borel*] *measurable-prob-algebraD*)
qed(*auto intro*!: *qbs-morphism-monadPD' bind-qbs-morphism'*[*OF qbs-morphism-monadPD*[*OF assms*]])

lemma *distr-qbs-morphismP'*:
assumes $f \in X \to_Q Y$
shows *distr-qbs X Y f* $\in$ *monadP-qbs X* $\to_Q$ *monadP-qbs Y*
unfolding *distr-qbs-def*
by(*rule bind-qbs-morphism'P*[*OF qbs-morphism-comp*[*OF assms return-qbs-morphismP*]])

lemma *join-qbs-morphismP*: *join-qbs* $\in$ *monadP-qbs* (*monadP-qbs X*) $\to_Q$ *monadP-qbs X*
by(*simp add*: *join-qbs-def bind-qbs-morphism'P*[*OF qbs-morphism-ident*])

lemma
assumes *qbs-prob* (*monadP-qbs X*) *β μ*
ssx = $[\![$*monadP-qbs X*, *β*, *μ*$]\!]_{meas}$
α $\in$ *qbs-Mx X*
g $\in$ *borel* $\to_M$ *prob-algebra borel*
and $\beta = (\lambda r.\ [\![X,\ \alpha,\ g\ r]\!]_{meas})$
shows *qbs-prob-join-qbs-s-finite*: *qbs-prob X α* $(\mu \gg= g)$
and *qbs-prob-join-qbs*: *join-qbs ssx* = $[\![X,\ \alpha,\ \mu \gg= g]\!]_{meas}$
using *qbs-prob.bind-qbs'*[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)] *qbs-prob.bind-qbs-prob*[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)]

by(*auto simp*: *assms*(*5*) *join-qbs-def*)

context
begin

interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

lemma *strength-qbs-ab-r-prob*:
assumes $\alpha \in$ *qbs-Mx X*
$\beta \in$ *qbs-Mx* (*monadP-qbs Y*)
$\gamma \in$ *qbs-Mx Y*
and [*measurable*]:*g* ∈ *borel* →$_M$ *prob-algebra borel*
and $\beta = (\lambda r.\ [\![Y,\ \gamma,\ g\ r]\!]_{meas})$
shows *qbs-prob* (*X* $\bigotimes_Q$ *Y*) (*map-prod* α γ ∘ *rr.from-real*) (*distr* (*return borel r* $\bigotimes_M$ *g r*) *borel rr.to-real*)
using *measurable-space*[*OF assms*(*4*),*of r*] *sets-return*[*of borel r*]
by(*auto intro*!: *qbs-meas.qbs-probI qbs-s-finite.axioms*(*1*) *strength-qbs-ab-r-s-finite*[*OF assms*(*1*) *qbs-Mx-monadPM*[*OF assms*(*2*)]
assms(*3*) *prob-kernel.s-finite-kernel-prob-kernel assms*(*5*),*simplified prob-kernel-def'*,*OF assms*(*4*)]
prob-space.prob-space-distr prob-space-pair prob-space-return simp: *space-prob-algebra simp del*: *sets-return*)

lemma *strength-qbs-morphismP*: *strength-qbs X Y* ∈ *X* $\bigotimes_Q$ *monadP-qbs Y* →$_Q$ *monadP-qbs* (*X* $\bigotimes_Q$ *Y*)
proof(*rule pair-qbs-morphismI*)
fix α β
assume *h*:$\alpha \in$ *qbs-Mx X*
$\beta \in$ *qbs-Mx* (*monadP-qbs Y*)
from *rep-qbs-Mx-monadP*[*OF this*(*2*)] **obtain** γ *g* **where** *hb*[*measurable*]:
$\beta = (\lambda r.\ [\![Y,\ \gamma,\ g\ r]\!]_{meas})$ $\gamma \in$ *qbs-Mx Y g* ∈ *borel* →$_M$ *prob-algebra borel*
by *metis*
show ($\lambda r.$ *strength-qbs X Y* (α *r*, β *r*)) ∈ *qbs-Mx* (*monadP-qbs* (*X* $\bigotimes_Q$ *Y*))
using *strength-qbs-ab-r-prob*[*OF h hb*(*2*,*3*,*1*)] *strength-qbs-ab-r*[*OF h*(*1*) *qbs-Mx-monadPM*[*OF h*(*2*)] *hb*(*2*) *prob-kernel.s-finite-kernel-prob-kernel hb*(*1*),*simplified prob-kernel-def'*,*OF hb*(*3*)]
by(*auto simp*: *monadP-qbs-Mx qbs-prob-def in-Mx-def intro*!: *exI*[**where** *x*=*map-prod* α γ ∘ *rr.from-real*] *exI*[**where** *x*=$\lambda r.$ *distr* (*return borel r* $\bigotimes_M$ *g r*) *borel rr.to-real*])
qed

end

lemma *bind-qbs-morphismP*: (≫=) ∈ *monadP-qbs X* →$_Q$ (*X* ⇒$_Q$ *monadP-qbs Y*) ⇒$_Q$ *monadP-qbs Y*
proof −
{
fix *f s*
assume *h*:*f* ∈ *X* →$_Q$ *monadP-qbs Y s* ∈ *qbs-space* (*monadP-qbs X*)

from *rep-qbs-space-monadP*[*OF this*(*2*)] **obtain** α μ **where** h':
$s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-prob X* α μ **by** *metis*
then interpret *qbs-prob X* α μ **by** *simp*
from *rep-qbs-Mx-monadP*[*OF qbs-morphism-Mx*[*OF h*(*1*) *in-Mx*]] **obtain** β *g*
where *hb*[*measurable*]:*f* $\circ$ α = ($\lambda r.$ $[\![Y, \beta, g\ r]\!]_{meas}$) β $\in$ *qbs-Mx Y g* $\in$ *borel* $\rightarrow_M$ *prob-algebra borel* **by** *metis*
have *join-qbs* (*distr-qbs* ((*X* $\Rightarrow_Q$ *monadP-qbs Y*) $\bigotimes_Q$ *X*) (*monadP-qbs Y*) (λ*fx. fst fx* (*snd fx*)) (*strength-qbs* (*X* $\Rightarrow_Q$ *monadP-qbs Y*) *X* (*f*, *s*))) = *s* $\gg\!=$ *f*
using *qbs-prob-join-qbs*[*OF qbs-prob.distr-qbs-prob*[*OF strength-qbs-prob*[*OF h*(*1*)] *qbs-morphism-eval*] *qbs-meas.distr-qbs*[*OF strength-qbs-meas*[*OF h*(*1*) *h'*(*1*)] *strength-qbs*[*OF h*(*1*) *h'*(*1*)] *qbs-morphism-eval*] *hb*(*2*,*3*)] *hb*(*1*)
by(*simp add*: *bind-qbs*[*OF h'*(*1*) *qbs-morphism-monadPD*[*OF h*(*1*)] *hb*(*2*) *prob-kernel.s-finite-kernel-prob-kernel hb*(*1*),*simplified prob-kernel-def'*] *comp-def bind-kernel-bind*[*of g* μ *borel*,*OF measurable-prob-algebraD*])
}
thus *?thesis*
by(*auto intro*!: *arg-swap-morphism*[*OF curry-preserves-morphisms* [*OF qbs-morphism-cong'*[*of* - *join-qbs* $\circ$ (*distr-qbs* (*exp-qbs X* (*monadP-qbs Y*) $\bigotimes_Q$ *X*) (*monadP-qbs Y*) (λ*fx.* (*fst fx*) (*snd fx*))) $\circ$ (*strength-qbs* (*exp-qbs X* (*monadP-qbs Y*)) *X*)]]] *qbs-morphism-comp distr-qbs-morphismP' strength-qbs-morphismP join-qbs-morphismP qbs-morphism-eval simp*: *pair-qbs-space*)
qed

corollary *strength-qbs-law1P*:
assumes *x* $\in$ *qbs-space* (*unit-quasi-borel* $\bigotimes_Q$ *monadP-qbs X*)
shows *snd x* = (*distr-qbs* (*unit-quasi-borel* $\bigotimes_Q$ *X*) *X snd* $\circ$ *strength-qbs unit-quasi-borel X*) *x*
by(*rule strength-qbs-law1*, *insert assms*) (*auto simp*: *pair-qbs-space qbs-space-monadPM*)

corollary *strength-qbs-law2P*:
assumes *x* $\in$ *qbs-space* ((*X* $\bigotimes_Q$ *Y*) $\bigotimes_Q$ *monadP-qbs Z*)
shows (*strength-qbs X* (*Y* $\bigotimes_Q$ *Z*) $\circ$ (*map-prod id* (*strength-qbs Y Z*)) $\circ$ (λ((*x*,*y*),*z*). (*x*,(*y*,*z*)))) *x* =
(*distr-qbs* ((*X* $\bigotimes_Q$ *Y*) $\bigotimes_Q$ *Z*) (*X* $\bigotimes_Q$ (*Y* $\bigotimes_Q$ *Z*)) (λ((*x*,*y*),*z*). (*x*,(*y*,*z*))) $\circ$ *strength-qbs* (*X* $\bigotimes_Q$ *Y*) *Z*) *x*
by(*rule strength-qbs-law2*, *insert assms*) (*auto simp*: *pair-qbs-space qbs-space-monadPM*)

lemma *strength-qbs-law4P*:
assumes *x* $\in$ *qbs-space* (*X* $\bigotimes_Q$ *monadP-qbs* (*monadP-qbs Y*))
shows (*strength-qbs X Y* $\circ$ *map-prod id join-qbs*) *x* = (*join-qbs* $\circ$ *distr-qbs* (*X* $\bigotimes_Q$ *monadP-qbs Y*) (*monadP-qbs* (*X* $\bigotimes_Q$ *Y*)) (*strength-qbs X Y*) $\circ$ *strength-qbs X* (*monadP-qbs Y*)) *x*
(**is** *?lhs* = *?rhs*)
proof −
from *assms rep-qbs-space-monadP*[*of snd x monadP-qbs Y*] **obtain** β μ
where *h*:*qbs-prob* (*monadP-qbs Y*) β μ *snd x* = $[\![$*monadP-qbs Y*, β, $\mu]\!]_{meas}$
by (*auto simp*: *pair-qbs-space*) *metis*
then interpret *qp*: *qbs-prob monadP-qbs Y* β μ **by** *simp*
from *rep-qbs-Mx-monadP*[*OF qp.in-Mx*] **obtain** γ *g*

where h': $\gamma \in$ *qbs-Mx* Y $g \in$ *borel* $\rightarrow_M$ *prob-algebra borel* $\beta = (\lambda r.\ [\![Y,\ \gamma,\ g\ r]\!]_{meas})$
and h'': $\bigwedge r.$ *qbs-prob* Y γ $(g\ r)$
by *metis*
have *?lhs* $= [\![X \bigotimes_Q Y,\ \lambda r.\ (fst\ x,\ \gamma\ r),\ \mu \gg\!= g]\!]_{meas}$
using *qbs-meas.strength-qbs*[*OF qbs-prob.qbs-meas*[*OF qbs-prob-join-qbs-s-finite*[*OF h h'*]] - *qbs-prob-join-qbs*[*OF h h'*],*of fst x X*] *assms*
by (*auto simp*: *pair-qbs-space*)
also have ... $=$ *?rhs*
using *qbs-prob-join-qbs*[*OF qbs-prob.distr-qbs-prob*[*OF qp.strength-qbs-prob*[*of fst x X*] *strength-qbs-morphismP*] *qbs-meas.distr-qbs*[*OF qp.strength-qbs-meas*[*OF - h(2)*,*of fst x X*] *qp.strength-qbs*[*OF - h(2)*] *strength-qbs-morphismP*] *pair-qbs-MxI h'(2)*,*of* $\lambda r.$ (*fst x*, γ *r*)] *assms*
qbs-meas.strength-qbs[*OF qbs-prob.qbs-meas*[*OF h''*] - *fun-cong*[*OF h'(3)*]]
by(*fastforce simp*: *pair-qbs-space* *h'*)
finally show *?thesis* **.**
qed

lemma *distr-qbs-morphismP*: *distr-qbs* X $Y \in X \Rightarrow_Q Y \rightarrow_Q$ *monadP-qbs* $X \Rightarrow_Q$ *monadP-qbs* Y
proof −
note [*qbs*] = *bind-qbs-morphismP return-qbs-morphismP*
have [*simp*]: *distr-qbs* X $Y = (\lambda f\ sx.\ sx \gg\!= return\text{-}qbs\ Y \circ f)$
by *standard*+ (*auto simp*: *distr-qbs-def*)
show *?thesis*
by *simp*
qed

lemma *bind-qbs-return-rotateP*:
assumes $p \in$ *qbs-space* (*monadP-qbs* X)
and $q \in$ *qbs-space* (*monadP-qbs* Y)
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y))) = p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y)))$
by(*auto intro*!: *bind-qbs-return-rotate qbs-space-monadPM assms*)

lemma *qbs-bind-bind-return1P*:
assumes $f \in X \bigotimes_Q Y \rightarrow_Q$ *monadP-qbs* Z
$p \in$ *qbs-space* (*monadP-qbs* X)
$q \in$ *qbs-space* (*monadP-qbs* Y)
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) = (q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y)))) \gg\!= f$
by(*auto intro*!: *qbs-bind-bind-return1 assms qbs-space-monadPM qbs-morphism-monadPD*)

corollary *qbs-bind-bind-return1P'*:
assumes [*qbs*]:$f \in$ *qbs-space* $(X \Rightarrow_Q Y \Rightarrow_Q$ *monadP-qbs* $Z)$
$p \in$ *qbs-space* (*monadP-qbs* X)
$q \in$ *qbs-space* (*monadP-qbs* Y)
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ x\ y)) = (q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ return\text{-}qbs\ (X \bigotimes_Q Y)\ (x,y)))) \gg\!= (case\text{-}prod\ f)$

by(*auto intro*!: *qbs-bind-bind-return1P*[**where** *f*=*case-prod f* **and** *Z*=*Z*,*simplified*])

lemma *qbs-bind-bind-return2P*:
assumes $f \in X \bigotimes_Q Y \to_Q$ *monadP-qbs Z*
$p \in$ *qbs-space* (*monadP-qbs X*) $q \in$ *qbs-space* (*monadP-qbs Y*)
shows $p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ (x,y))) = (p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y)))) \gg\!= f$
by(*auto intro*!: *qbs-bind-bind-return2 assms qbs-space-monadPM qbs-morphism-monadPD*)

corollary *qbs-bind-bind-return2P'*:
assumes [*qbs*]:$f \in$ *qbs-space* $(X \Rightarrow_Q Y \Rightarrow_Q$ *monadP-qbs Z*)
$p \in$ *qbs-space* (*monadP-qbs X*)
$q \in$ *qbs-space* (*monadP-qbs Y*)
shows $p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ x\ y)) = (p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.$ *return-qbs* $(X \bigotimes_Q Y)\ (x,y)))) \gg\!= ($*case-prod f*$)$
by(*auto intro*!: *qbs-bind-bind-return2P*[**where** *f*=*case-prod f* **and** *Z*=*Z*,*simplified*])

corollary *bind-qbs-rotateP*:
assumes $f \in X \bigotimes_Q Y \to_Q$ *monadP-qbs Z*
$p \in$ *qbs-space* (*monadP-qbs X*)
and $q \in$ *qbs-space* (*monadP-qbs Y*)
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) = p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ (x,y)))$
by(*auto intro*!: *bind-qbs-rotate assms qbs-space-monadPM qbs-morphism-monadPD*)

context *pair-qbs-probs*
begin

interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

sublocale *qbs-prob* $X \bigotimes_Q Y$ *map-prod* $\alpha\ \beta \circ$ *rr.from-real distr* $(\mu \bigotimes_M \nu)$ *borel rr.to-real*
by(*auto simp*: *qbs-prob-def in-Mx-def real-distribution-def qbs-Mx-is-morphisms real-distribution-axioms-def pq1.prob-space-axioms pq2.prob-space-axioms intro*!: *prob-space.prob-space-distr prob-space-pair*)

lemma *qbs-bind-bind-return-prob*:
assumes [*qbs*]:$f \in X \bigotimes_Q Y \to_Q Z$
shows *qbs-prob Z* (*f* ∘ (*map-prod* $\alpha\ \beta \circ$ *rr.from-real*)) (*distr* $(\mu \bigotimes_M \nu)$ *borel rr.to-real*)
using *qbs-prob-axioms* **by**(*auto simp*: *qbs-prob-def in-Mx-def qbs-Mx-is-morphisms*)

end

4.1.11 Almost Everywhere

lift-definition *qbs-almost-everywhere* :: [*'a qbs-measure*, *'a* ⇒ *bool*] ⇒ *bool*
is $\lambda(X,\alpha,\mu)$. *almost-everywhere* (*distr* μ (*qbs-to-measure X*) α)
by(*auto simp*: *qbs-meas-eq-def*) *metis*

syntax
-qbs-almost-everywhere :: *pttrn* ⇒ *'a* ⇒ *bool* ⇒ *bool* (*AE*$_Q$ - *in* -. - [*0*,*0*,*10*] *10*)

syntax-consts
-qbs-almost-everywhere ⇌ *qbs-almost-everywhere*

translations
AE$_Q$ *x in p. P* ⇌ *CONST qbs-almost-everywhere p* (λ*x. P*)

lemma *AEq-qbs-l*: (*AE*$_Q$ *x in p. P x*) = (*AE x in qbs-l p. P x*)
by *transfer fastforce*

lemma(**in** *qbs-meas*) *AEq-def*:
(*AE*$_Q$ *x in* ⟦*X*, *α*, *μ*⟧$_{meas}$. *P x*) = (*AE x in* (*distr μ* (*qbs-to-measure X*) *α*). *P x*)
by(*simp add*: *qbs-almost-everywhere.abs-eq*)

lemma(**in** *qbs-meas*) *AEq-AE*: (*AE*$_Q$ *x in* ⟦*X*, *α*, *μ*⟧$_{meas}$. *P x*) ⟹ (*AE x in μ*. *P* (*α x*))
by(*auto simp*: *AEq-def intro*!:*AE-distrD*[*of α*])

lemma(**in** *qbs-meas*) *AEq-AE-iff*:
assumes [*qbs*]:*qbs-pred X P*
shows (*AE*$_Q$ *x in* ⟦*X*, *α*, *μ*⟧$_{meas}$. *P x*) ⟷ (*AE x in μ*. *P* (*α x*))
by(*auto simp*: *AEq-AE AEq-def qbs-pred-iff-sets intro*!: *AE-distr-iff*[*THEN iffD2*])

lemma *AEq-qbs-pred*[*qbs*]: *qbs-almost-everywhere* ∈ *monadM-qbs X* →$_Q$ (*X* ⇒$_Q$ *qbs-count-space UNIV*) ⇒$_Q$ *qbs-count-space UNIV*
proof(*rule curry-preserves-morphisms*[*OF pair-qbs-morphismI*])
fix *γ β*
assume *h*:*γ* ∈ *qbs-Mx* (*monadM-qbs X*) *β* ∈ *qbs-Mx* (*X* ⇒$_Q$ *qbs-count-space* (*UNIV* :: *bool set*))
from *rep-qbs-Mx-monadM*[*OF h*(*1*)] **obtain** *α k* **where** *hk*:
γ = (λ*r*. ⟦*X*, *α*, *k r*⟧$_{meas}$) *α* ∈ *qbs-Mx X s-finite-kernel borel borel k* ⋀*r*. *qbs-s-finite X α* (*k r*)
by *metis*
interpret *s*:*standard-borel-ne borel* :: *real measure* **by** *simp*
interpret *s2*: *standard-borel-ne borel* ⨂$_M$ *borel* :: (*real* × *real*) *measure* **by**(*simp add*: *borel-prod*)
have [*measurable*]:*Measurable.pred* (*borel* ⨂$_M$ *borel*) (λ(*x*, *y*). *β x* (*α y*))
using *h*(*2*) *hk*(*2*) **by**(*simp add*: *s2.qbs-pred-iff-measurable-pred*[*symmetric*] *r-preserves-product qbs-Mx-is-morphisms*)
show (λ*r*. *qbs-almost-everywhere* (*fst* (*γ r*, *β r*)) (*snd* (*γ r*, *β r*))) ∈ *qbs-Mx* (*qbs-count-space UNIV*)
using *h*(*2*) *hk*(*2*) **by**(*simp add*: *hk*(*1*) *qbs-Mx-is-morphisms qbs-meas.AEq-AE-iff*[*OF hk*(*4*)[*THEN qbs-s-finite.axioms*(*1*)]])
(*auto simp add*: *s.qbs-pred-iff-measurable-pred intro*!: *s-finite-kernel.AE-pred*[*OF hk*(*3*)])

qed

lemma *AEq-I2'*[*simp*]:
assumes $p \in$ *qbs-space* (*all-meas-qbs X*) $\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *P x*
shows AE_Q *x in p. P x*
by(*auto simp*: *space-qbs-l-in-all-meas*[*OF assms*(*1*)] *assms*(*2*) *AEq-qbs-l*)

lemma *AEq-I2*[*simp*]:
assumes $p \in$ *qbs-space* (*monadM-qbs X*) $\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *P x*
shows AE_Q *x in p. P x*
by(*auto simp*: *space-qbs-l-in*[*OF assms*(*1*)] *assms*(*2*) *AEq-qbs-l*)

lemma *AEq-mp*[*elim*!]:
assumes AE_Q *x in s. P x* AE_Q *x in s. P x* $\longrightarrow$ *Q x*
shows AE_Q *x in s. Q x*
using *assms* **by**(*auto simp*: *AEq-qbs-l*)

lemma
shows *AEq-iffI*: AE_Q *x in s. P x* $\Longrightarrow$ AE_Q *x in s. P x* $\longleftrightarrow$ *Q x* $\Longrightarrow$ AE_Q *x in s. Q x*
and *AEq-disjI1*: AE_Q *x in s. P x* $\Longrightarrow$ AE_Q *x in s. P x* $\vee$ *Q x*
and *AEq-disjI2*: AE_Q *x in s. Q x* $\Longrightarrow$ AE_Q *x in s. P x* $\vee$ *Q x*
and *AEq-conjI*: AE_Q *x in s. P x* $\Longrightarrow$ AE_Q *x in s. Q x* $\Longrightarrow$ AE_Q *x in s. P x* $\wedge$ *Q x*
and *AEq-conj-iff*[*simp*]: (AE_Q *x in s. P x* $\wedge$ *Q x*) $\longleftrightarrow$ (AE_Q *x in s. P x*) $\wedge$ (AE_Q *x in s. Q x*)
by(*auto simp*: *AEq-qbs-l*)

lemma *AEq-symmetric*:
assumes AE_Q *x in s. P x* = *Q x*
shows AE_Q *x in s. Q x* = *P x*
using *assms* **by**(*auto simp*: *AEq-qbs-l*)

lemma *AEq-impI*: (*P* $\Longrightarrow$ AE_Q *x in M. Q x*) $\Longrightarrow$ AE_Q *x in M. P* $\longrightarrow$ *Q x*
by(*auto simp*: *AEq-qbs-l AE-impI*)

lemma
shows *AEq-Ball-mp-all-meas*:
s $\in$ *qbs-space* (*all-meas-qbs X*) $\Longrightarrow$ ($\bigwedge x.\ x \in$*qbs-space X* $\Longrightarrow$ *P x*) $\Longrightarrow$ AE_Q *x in s. P x* $\longrightarrow$ *Q x* $\Longrightarrow$ AE_Q *x in s. Q x*
and *AEq-Ball-mp*:
s $\in$ *qbs-space* (*monadM-qbs X*) $\Longrightarrow$ ($\bigwedge x.\ x \in$*qbs-space X* $\Longrightarrow$ *P x*) $\Longrightarrow$ AE_Q *x in s. P x* $\longrightarrow$ *Q x* $\Longrightarrow$ AE_Q *x in s. Q x*
and *AEq-cong-all-meas*:
s $\in$ *qbs-space* (*all-meas-qbs X*) $\Longrightarrow$ ($\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *P x* $\longleftrightarrow$ *Q x*) $\Longrightarrow$ (AE_Q *x in s. P x*) $\longleftrightarrow$ (AE_Q *x in s. Q x*)
and *AEq-cong*:
s $\in$ *qbs-space* (*monadM-qbs X*) $\Longrightarrow$ ($\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *P x* $\longleftrightarrow$ *Q x*) $\Longrightarrow$ (AE_Q *x in s. P x*) $\longleftrightarrow$ (AE_Q *x in s. Q x*)

by *auto*

lemma
shows *AEq-cong-simp-all-meas*: $s \in$ *qbs-space* (*all-meas-qbs* X) $\Longrightarrow$ ($\bigwedge x.\ x \in$ *qbs-space* X =*simp*=> $P\ x = Q\ x$) $\Longrightarrow$ (AE_Q x *in s.* $P\ x$) $\longleftrightarrow$ (AE_Q x *in s.* $Q\ x$)
and *AEq-cong-simp*: $s \in$ *qbs-space* (*monadM-qbs* X) $\Longrightarrow$ ($\bigwedge x.\ x \in$ *qbs-space* X =*simp*=> $P\ x = Q\ x$) $\Longrightarrow$ (AE_Q x *in s.* $P\ x$) $\longleftrightarrow$ (AE_Q x *in s.* $Q\ x$)
by (*auto simp*: *simp-implies-def*)

lemma *AEq-all-countable*: (AE_Q x *in s.* $\forall i.\ P\ i\ x$) $\longleftrightarrow$ ($\forall i{::}'i{::}$*countable.* AE_Q x *in s.* $P\ i\ x$)
by(*simp add*: *AEq-qbs-l AE-all-countable*)

lemma *AEq-ball-countable*: *countable* $X \Longrightarrow$ (AE_Q x *in s.* $\forall y{\in}X.\ P\ x\ y$) $\longleftrightarrow$ ($\forall y{\in}X.$ AE_Q x *in s.* $P\ x\ y$)
by(*simp add*: *AEq-qbs-l AE-ball-countable*)

lemma *AEq-ball-countable'*: ($\bigwedge N.\ N \in I \Longrightarrow AE_Q$ x *in s.* $P\ N\ x$) $\Longrightarrow$ *countable* $I \Longrightarrow AE_Q$ x *in s.* $\forall N \in I.\ P\ N\ x$
unfolding *AEq-ball-countable* **by** *simp*

lemma *AEq-pairwise*: *countable* $F \Longrightarrow$ *pairwise* ($\lambda A\ B.\ AE_Q$ x *in s.* $R\ x\ A\ B$) F $\longleftrightarrow$ (AE_Q x *in s.* *pairwise* ($R\ x$) F)
unfolding *pairwise-alt* **by** (*simp add*: *AEq-ball-countable*)

lemma *AEq-finite-all*: *finite* $S \Longrightarrow$ (AE_Q x *in s.* $\forall i{\in}S.\ P\ i\ x$) $\longleftrightarrow$ ($\forall i{\in}S.\ AE_Q$ x *in s.* $P\ i\ x$)
by(*simp add*: *AEq-qbs-l AE-finite-all*)

lemma *AEq-finite-allI*:*finite* $S \Longrightarrow$ ($\bigwedge s.\ s \in S \Longrightarrow AE_Q$ x *in* $M.\ Q\ s\ x$) $\Longrightarrow AE_Q$ x *in* $M.\ \forall s{\in}S.\ Q\ s\ x$
by(*simp add*: *AEq-qbs-l AE-finite-all*)

4.1.12 Integral

lift-definition *qbs-nn-integral* :: [$'a$ *qbs-measure*, $'a \Rightarrow$ *ennreal*] $\Rightarrow$ *ennreal*
is $\lambda(X,\alpha,\mu)\ f.(\int^+ x.\ f\ x\ \partial distr\ \mu\ (qbs\text{-}to\text{-}measure\ X)\ \alpha)$
by(*auto simp*: *qbs-meas-eq-def*)

lift-definition *qbs-integral* :: [$'a$ *qbs-measure*, $'a \Rightarrow$ ($'b$ :: {*banach*,*second-countable-topology*})] $\Rightarrow$ $'b$
is $\lambda(X,\alpha,\mu)\ f.$ *if* $f \in X \to_Q$ *qbs-borel then* ($\int x.\ f\ (\alpha\ x)\ \partial\mu$) *else 0*
by(*fastforce dest*: *qbs-meas-eq-integral-eq qbs-meas-eq-dest*(*3*))

syntax
-qbs-nn-integral :: *pttrn* $\Rightarrow$ *ennreal* $\Rightarrow$ $'a$ *qbs-measure* $\Rightarrow$ *ennreal* ($\int^+{}_Q$((*2* -./ -)/ ∂-) [*60*,*61*] *110*)

syntax-consts

-qbs-nn-integral $\rightleftharpoons$ *qbs-nn-integral*

translations
$\int^+_Q x.\ f\ \partial p \rightleftharpoons CONST\ qbs\text{-}nn\text{-}integral\ p\ (\lambda x.\ f)$

syntax
-qbs-integral :: *pttrn* $\Rightarrow$ - $\Rightarrow$ *'a qbs-measure* $\Rightarrow$ - ($\int_Q$((*2* -./ -)/ ∂-) [*60*,*61*] *110*)

syntax-consts
-qbs-integral $\rightleftharpoons$ *qbs-integral*

translations
$\int_Q x.\ f\ \partial p \rightleftharpoons CONST\ qbs\text{-}integral\ p\ (\lambda x.\ f)$

lemma(**in** *qbs-meas*)
shows *qbs-nn-integral-def*: $f \in X \to_Q qbs\text{-}borel \Longrightarrow (\int^+_Q x.\ f\ x\ \partial[\![X,\ \alpha,\ \mu]\!]_{meas}) = (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu)$
and *qbs-nn-integral-def2*:$(\int^+_Q x.\ f\ x\ \partial[\![X,\ \alpha,\ \mu]\!]_{meas}) = (\int^+ x.\ f\ x\ \partial(distr\ \mu\ (qbs\text{-}to\text{-}measure\ X)\ \alpha))$
by(*simp-all add*: *qbs-nn-integral.abs-eq nn-integral-distr lr-adjunction-correspondence*)

lemma(**in** *qbs-meas*) *qbs-integral-def*:
$f \in X \to_Q qbs\text{-}borel \Longrightarrow (\int_Q x.\ f\ x\ \partial[\![X,\ \alpha,\ \mu]\!]_{meas}) = (\int x.\ f\ (\alpha\ x)\ \partial\ \mu)$
by(*simp add*: *qbs-integral.abs-eq*)

lemma(**in** *qbs-meas*) *qbs-integral-def2*: $(\int_Q x.\ f\ x\ \partial[\![X,\ \alpha,\ \mu]\!]_{meas}) = (\int x.\ f\ x\ \partial(distr\ \mu\ (qbs\text{-}to\text{-}measure\ X)\ \alpha))$
proof −
consider $f \in X \to_Q qbs\text{-}borel \mid f \notin X \to_Q qbs\text{-}borel$ **by** *auto*
thus *?thesis*
proof *cases*
case *h*:*2*
then have ¬ *integrable* (*qbs-l* $[\![X,\ \alpha,\ \mu]\!]_{meas}$) *f*
by (*metis borel-measurable-integrable measurable-distr-eq1 qbs-l qbs-morphism-measurable-intro*)
thus *?thesis*
using *h* **by**(*simp add*: *qbs-l qbs-integral.abs-eq not-integrable-integral-eq*)
qed(*simp add*: *qbs-integral.abs-eq integral-distr*)
qed

lemma *qbs-measure-eqI-all-meas*:
assumes [*qbs*]:$p \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)\ q \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)$
and $\bigwedge f.\ f \in X \to_Q qbs\text{-}borel \Longrightarrow (\int^+_Q x.\ f\ x\ \partial p) = (\int^+_Q x.\ f\ x\ \partial q)$
shows $p = q$
proof −
obtain $\alpha\ \mu\ \beta\ \nu$ **where** *h*:$p = [\![X,\ \alpha,\ \mu]\!]_{meas}\ q = [\![X,\ \beta,\ \nu]\!]_{meas}\ qbs\text{-}meas\ X\ \alpha\ \mu\ qbs\text{-}meas\ X\ \beta\ \nu$
by (*meson assms*(*1*) *assms*(*2*) *rep-qbs-space-all-meas*)
then interpret *pq*:*pair-qbs-meas X* $\alpha\ \mu\ \beta\ \nu$
by(*auto simp*: *pair-qbs-meas-def*)

show *?thesis*
using *assms*(*3*) **by**(*auto simp*: *h*(*1*,*2*) *pq.pq1*.*qbs-nn-integral-def pq.pq2*.*qbs-nn-integral-def intro*!: *pq.qbs-meas-eqI2*)
qed

lemma *qbs-measure-eqI*:
assumes [*qbs*]:*p* ∈ *qbs-space* (*monadM-qbs X*) *q* ∈ *qbs-space* (*monadM-qbs X*)
and ⋀*f*. *f* ∈ *X* →$_Q$ *qbs-borel* ⟹ (∫^+_Q *x*. *f x* ∂*p*) = (∫^+_Q *x*. *f x* ∂*q*)
shows *p* = *q*
by(*auto intro*!: *assms qbs-measure-eqI-all-meas monadM-all-meas-space*)

lemma *qbs-nn-integral-def2-l*: *qbs-nn-integral s f* = *integral*^{N} (*qbs-l s*) *f*
by *transfer auto*

lemma *qbs-integral-def2-l*: *qbs-integral s f* = *integral*^{L} (*qbs-l s*) *f*
by (*metis qbs-meas.qbs-integral-def2 qbs-meas.qbs-l rep-qbs-measure'*)

definition *qbs-integrable* :: *'a qbs-measure* ⇒ (*'a* ⇒ *'b*::{*second-countable-topology*,*real-normed-vector*}) ⇒ *bool*
where *qbs-integrable-iff-integrable*: *qbs-integrable p f* ⟷ *integrable* (*qbs-l p*) *f*

lemma(**in** *qbs-meas*) *qbs-integrable-def*:
fixes *f* :: *'a* ⇒ *'b*::{*second-countable-topology*,*banach*}
shows *qbs-integrable* ⟦*X*, *α*, *μ*⟧$_{meas}$ *f* ⟷ *f* ∈ *X* →$_Q$ *qbs-borel* ∧ *integrable μ* (λ*x*. *f* (*α x*))
proof −
have *qbs-integrable* ⟦*X*, *α*, *μ*⟧$_{meas}$ *f* ⟷ *f* ∈ *X* →$_Q$ *qbs-borel* ∧ *integrable* (*distr μ* (*qbs-to-measure X*) *α*) *f*
by(*auto simp*: *lr-adjunction-correspondence qbs-integrable-iff-integrable qbs-l*)
also have ... ⟷ *f* ∈ *X* →$_Q$ *qbs-borel* ∧ *integrable μ* (λ*x*. *f* (*α x*))
by(*auto simp*: *integrable-distr-eq*)
finally show *?thesis* **.**
qed

lemma *qbs-integrable-morphism-dest-all-meas*:
assumes *s* ∈ *qbs-space* (*all-meas-qbs X*)
and *qbs-integrable s f*
shows *f* ∈ *X* →$_Q$ *qbs-borel*
using *assms*(*2*)
by(*auto simp add*: *measurable-qbs-l-all-meas*[*OF assms*(*1*),*symmetric*] *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-morphism-dest*:
assumes *s* ∈ *qbs-space* (*monadM-qbs X*)
and *qbs-integrable s f*
shows *f* ∈ *X* →$_Q$ *qbs-borel*
by(*auto intro*!: *qbs-integrable-morphism-dest-all-meas assms monadM-all-meas-space*)

lemma *qbs-integrable-morphismP*:
assumes *s* ∈ *qbs-space* (*monadP-qbs X*)

and *qbs-integrable s f*
shows $f \in X \to_Q$ *qbs-borel*
by(*auto intro*!: *qbs-integrable-morphism-dest assms qbs-space-monadPM*)

lemma(**in** *qbs-s-finite*) *qbs-integrable-measurable*[*simp*]:
assumes *qbs-integrable* $[\![X,\alpha,\mu]\!]_{meas}$ *f*
shows $f \in$ *qbs-to-measure X* $\to_M$ *borel*
by(*auto intro*!: *qbs-integrable-morphism-dest assms simp*: *lr-adjunction-correspondence*[*symmetric*])

corollary(**in** *qbs-meas*) *qbs-integrable-distr*: *qbs-integrable* $[\![X,\ \alpha,\ \mu]\!]_{meas}$ *f* = *integrable* (*distr* μ (*qbs-to-measure X*) α) *f*
by(*simp add*: *qbs-integrable-iff-integrable qbs-l*)

lemma *qbs-integrable-morphism*[*qbs*]: *qbs-integrable* $\in$ *monadM-qbs X* $\to_Q$ (*X* $\Rightarrow_Q$ (*qbs-borel* :: (*'a* :: {*banach*, *second-countable-topology*}) *quasi-borel*)) $\Rightarrow_Q$ *qbs-count-space UNIV*
proof(*rule curry-preserves-morphisms*[*OF pair-qbs-morphismI*])
fix γ β
assume *h*:$\gamma \in$ *qbs-Mx* (*monadM-qbs X*) $\beta \in$ *qbs-Mx* (*X* $\Rightarrow_Q$ (*qbs-borel* :: *'a quasi-borel*))
from *rep-qbs-Mx-monadM*[*OF this*(*1*)] **obtain** α *k*
where *hk*:γ = ($\lambda r.$ $[\![X,\ \alpha,\ k\ r]\!]_{meas}$) $\alpha \in$ *qbs-Mx X s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite X* α (*k r*)
by *metis*
then interpret *qbs-s-finite X* α *k r* **for** *r*
by *simp*
interpret *standard-borel-ne borel* :: *real measure* **by** *simp*
have [*measurable*]: β *r* $\in$ *qbs-to-measure X* $\to_M$ *borel* **for** *r*
using *h*(*2*) **by**(*simp add*: *qbs-Mx-is-morphisms lr-adjunction-correspondence*[*symmetric*])
have *1*: *borel-measurable* (*borel* $\bigotimes_M$ *borel*) = (*qbs-borel* $\bigotimes_Q$ *qbs-borel* $\to_Q$ *qbs-borel* :: (*real* $\times$ *real* $\Rightarrow$ *'a*) *set*)
by (*metis borel-prod pair-standard-borel qbs-borel-prod standard-borel.standard-borel-r-full-faithful standard-borel-axioms*)
show ($\lambda r.$ *qbs-integrable* (*fst* (γ *r*, β *r*)) (*snd* (γ *r*, β *r*))) $\in$ *qbs-Mx* (*qbs-count-space UNIV*)
by(*auto simp*: *fun-cong*[*OF hk*(*1*)] *qbs-integrable-distr integrable-distr-eq qbs-Mx-is-morphisms qbs-pred-iff-measurable-pred*
intro!: *s-finite-kernel.integrable-measurable-pred*[*OF hk*(*3*)])
(*use h*(*2*) **in** *simp add*: *1 qbs-Mx-is-morphisms split-beta'*)
qed

lemma(**in** *qbs-meas*) *qbs-integrable-iff-integrable*:
assumes *f* $\in$ *qbs-to-measure X* $\to_M$ (*borel* :: - ::{*second-countable-topology*,*banach*} *measure*)
shows *qbs-integrable* $[\![X,\ \alpha,\ \mu]\!]_{meas}$ *f* = *integrable* μ ($\lambda x.$ *f* (α *x*))
by(*auto intro*!: *integrable-distr-eq*[*of* α μ *qbs-to-measure X f*] *simp*: *assms qbs-integrable-distr*)

lemma *qbs-integrable-iff-bounded-all-meas*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*second-countable-topology*,*banach*}

assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
shows *qbs-integrable* $s\ f \longleftrightarrow f \in X \to_Q$ *qbs-borel* $\wedge$ $(\int^+_Q x.$ *ennreal* (*norm* (f x)) $\partial s) < \infty$
(**is** *?lhs* = *?rhs*)
proof −
from *rep-qbs-space-all-meas*[*OF assms*] **obtain** α μ **where** *hs*:
qbs-meas X α μ $s = [\![X, \alpha, \mu]\!]_{meas}$
by *metis*
then interpret *qs*: *qbs-meas* X α μ **by** *simp*
have *?lhs* = *integrable* (*distr* μ (*qbs-to-measure* X) α) f
by (*simp add*: *hs*(*2*) *qbs-integrable-iff-integrable qs.qbs-l*)
also have ... = ($f \in$ *borel-measurable* (*distr* μ (*qbs-to-measure* X) α) $\wedge$ (($\int^+ x.$ *ennreal* (*norm* (f x)) ∂(*distr* μ (*qbs-to-measure* X) α)) $< \infty$))
by(*rule integrable-iff-bounded*)
also have ... = *?rhs*
by(*auto simp add*: *hs*(*2*) *qs.qbs-nn-integral-def2 lr-adjunction-correspondence*)
finally show *?thesis* **.**
qed

lemmas *qbs-integrable-iff-bounded* = *qbs-integrable-iff-bounded-all-meas*[*OF monadM-all-meas-space*]

lemma *not-qbs-integrable-qbs-integral*: ¬ *qbs-integrable* $s\ f \Longrightarrow$ *qbs-integral* $s\ f$ = *0*
by(*simp add*: *qbs-integral-def2-l qbs-integrable-iff-integrable not-integrable-integral-eq*)

lemma *qbs-integrable-cong-AE-all-meas*:
assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
AE_Q x *in* s. $f\ x = g\ x$
and *qbs-integrable* $s\ f\ g \in X \to_Q$ *qbs-borel*
shows *qbs-integrable* $s\ g$
using *assms*(*2*,*4*)
by(*auto intro*!: *qbs-integrable-iff-integrable*[*THEN iffD2*] *Bochner-Integration.integrable-cong-AE*[*of g - f*,*THEN iffD2*]
qbs-integrable-iff-integrable[*THEN iffD1*,*OF assms*(*3*)] *qbs-integrable-morphism-dest-all-meas*[*OF assms*(*1*),*of f*]
simp: *AEq-qbs-l measurable-qbs-l-all-meas*[*OF assms*(*1*)])

lemmas *qbs-integrable-cong-AE* = *qbs-integrable-cong-AE-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integrable-cong-all-meas*:
assumes $s \in$ *qbs-space* (*all-meas-qbs* X)
$\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x = g\ x$
and *qbs-integrable* $s\ f$
shows *qbs-integrable* $s\ g$
by(*auto intro*!: *qbs-integrable-iff-integrable*[*THEN iffD2*] *Bochner-Integration.integrable-cong*[*OF refl*,*of - g f*,*THEN iffD2*]
qbs-integrable-iff-integrable[*THEN iffD1*,*OF assms*(*3*)]
simp: *space-qbs-l-in-all-meas*[*OF assms*(*1*)] *assms*(*2*))

lemmas *qbs-integrable-cong* = *qbs-integrable-cong-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integrable-zero*[*simp*, *intro*]: *qbs-integrable s* (λ*x*. *0*)
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-const*:
assumes *s* ∈ *qbs-space* (*monadP-qbs X*)
shows *qbs-integrable s* (λ*x*. *c*)
using *assms* **by**(*auto intro*!: *qbs-integrable-iff-integrable*[*THEN iffD2*] *finite-measure.integrable-const*
simp: *monadP-qbs-space prob-space-def*)

lemma *qbs-integrable-add*[*simp*,*intro*!]:
assumes *qbs-integrable s f*
and *qbs-integrable s g*
shows *qbs-integrable s* (λ*x*. *f x* + *g x*)
by(*rule qbs-integrable-iff-integrable*[*THEN iffD2*,*OF Bochner-Integration.integrable-add*[*OF qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*(*1*)] *qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*(*2*)]]])

lemma *qbs-integrable-diff*[*simp*,*intro*!]:
assumes *qbs-integrable s f*
and *qbs-integrable s g*
shows *qbs-integrable s* (λ*x*. *f x* − *g x*)
by(*rule qbs-integrable-iff-integrable*[*THEN iffD2*,*OF Bochner-Integration.integrable-diff*[*OF qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*(*1*)] *qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*(*2*)]]])

lemma *qbs-integrable-sum*[*simp*, *intro*!]: (⋀*i*. *i* ∈ *I* ⟹ *qbs-integrable s* (*f i*)) ⟹ *qbs-integrable s* (λ*x*. ∑ *i*∈*I*. *f i x*)
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-scaleR-left*[*simp*, *intro*!]: *qbs-integrable s f* ⟹ *qbs-integrable s* (λ*x*. *f x* ∗*R* (*c* :: *'a* :: {*second-countable-topology*,*banach*}))
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-scaleR-right*[*simp*, *intro*!]: *qbs-integrable s f* ⟹ *qbs-integrable s* (λ*x*. *c* ∗*R* (*f x* :: *'a* :: {*second-countable-topology*,*banach*}))
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-mult-iff*:
fixes *f* :: *'a* ⇒ *real*
shows (*qbs-integrable s* (λ*x*. *c* ∗ *f x*)) = (*c* = *0* ∨ *qbs-integrable s f*)
using *qbs-integrable-iff-integrable*[*of s* λ*x*. *c* ∗ *f x*] *integrable-mult-left-iff*[*of - c f*] *qbs-integrable-iff-integrable*[*of s f*]
by *simp*

lemma
fixes *c* :: -::{*real-normed-algebra*,*second-countable-topology*}

assumes *qbs-integrable s f*
shows *qbs-integrable-mult-right*:*qbs-integrable s* (λ*x*. *c* ∗ *f x*)
and *qbs-integrable-mult-left*: *qbs-integrable s* (λ*x*. *f x* ∗ *c*)
using *assms* **by**(*auto simp*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-divide-zero*[*simp*, *intro*!]:
fixes *c* :: -::{*real-normed-field*, *field*, *second-countable-topology*}
shows *qbs-integrable s f* ⟹ *qbs-integrable s* (λ*x*. *f x* / *c*)
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-inner-left*[*simp*, *intro*!]:
qbs-integrable s f ⟹ *qbs-integrable s* (λ*x*. *f x* · *c*)
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-inner-right*[*simp*, *intro*!]:
qbs-integrable s f ⟹ *qbs-integrable s* (λ*x*. *c* · *f x*)
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-minus*[*simp*, *intro*!]:
qbs-integrable s f ⟹ *qbs-integrable s* (λ*x*. − *f x*)
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma [*simp*, *intro*]:
assumes *qbs-integrable s f*
shows *qbs-integrable-Re*: *qbs-integrable s* (λ*x*. *Re* (*f x*))
and *qbs-integrable-Im*: *qbs-integrable s* (λ*x*. *Im* (*f x*))
and *qbs-integrable-cnj*: *qbs-integrable s* (λ*x*. *cnj* (*f x*))
using *assms* **by**(*simp-all add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-of-real*[*simp*, *intro*!]: *qbs-integrable s f* ⟹ *qbs-integrable s* (λ*x*. *of-real* (*f x*))
by(*simp-all add*: *qbs-integrable-iff-integrable*)

lemma [*simp*, *intro*]:
assumes *qbs-integrable s f*
shows *qbs-integrable-fst*: *qbs-integrable s* (λ*x*. *fst* (*f x*))
and *qbs-integrable-snd*: *qbs-integrable s* (λ*x*. *snd* (*f x*))
using *assms* **by**(*simp-all add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-norm*:
fixes *f* :: *'a* ⇒ *'b*::{*second-countable-topology*,*banach*}
assumes *qbs-integrable s f*
shows *qbs-integrable s* (λ*x*. *norm* (*f x*))
by(*rule qbs-integrable-iff-integrable*[*THEN iffD2*,*OF integrable-norm*[*OF qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*]]])

lemma *qbs-integrable-abs*:
fixes *f* :: - ⇒ *real*
assumes *qbs-integrable s f*

shows *qbs-integrable s* (λ*x*. |*f x*|)
by(*rule qbs-integrable-iff-integrable*[*THEN iffD2*,*OF integrable-abs*[*OF qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*]]])

lemma *qbs-integrable-sq*:
fixes *c* :: -::{*real-normed-field*,*second-countable-topology*}
assumes *qbs-integrable s* (λ*x*. *c*) *qbs-integrable s f*
and *qbs-integrable s* (λ*x*. (*f x*)^2)
shows *qbs-integrable s* (λ*x*. (*f x* − *c*)^2)
by(*simp add*: *comm-ring-1-class.power2-diff*,*rule qbs-integrable-diff*,*rule qbs-integrable-add*)
(*simp-all add*: *comm-semiring-1-class.semiring-normalization-rules*(*16*)[*of 2*]
assms qbs-integrable-mult-right power2-eq-square[*of c*])

lemma *qbs-nn-integral-eq-integral-AEq*:
assumes *qbs-integrable s f AE*$_Q$ *x in s*. *0* ≤ *f x*
shows (∫^+_Q *x*. *ennreal* (*f x*) ∂*s*) = *ennreal* (∫$_Q$ *x*. *f x* ∂*s*)
using *nn-integral-eq-integral*[*OF qbs-integrable-iff-integrable*[*THEN iffD1*,*OF assms*(*1*)]]
qbs-integrable-morphism-dest-all-meas[*OF in-qbs-space-of-all-meas assms*(*1*)]
assms(*2*)
by(*simp add*: *qbs-integral-def2-l qbs-nn-integral-def2-l AEq-qbs-l*)

lemma *qbs-nn-integral-eq-integral-all-meas*:
assumes *s* ∈ *qbs-space* (*all-meas-qbs X*) *qbs-integrable s f*
and ⋀*x*. *x* ∈ *qbs-space X* ⟹ *0* ≤ *f x*
shows (∫^+_Q *x*. *ennreal* (*f x*) ∂*s*) = *ennreal* (∫$_Q$ *x*. *f x* ∂*s*)
using *qbs-nn-integral-eq-integral-AEq*[*OF assms*(*2*) *AEq-I2'*[*OF assms*(*1*,*3*)]] **by**
simp

lemmas *qbs-nn-integral-eq-integral* = *qbs-nn-integral-eq-integral-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-nn-integral-cong-AEq-all-meas*:
assumes *s* ∈ *qbs-space* (*all-meas-qbs X*) *AE*$_Q$ *x in s*. *f x* = *g x*
shows *qbs-nn-integral s f* = *qbs-nn-integral s g*
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** α μ
where *hs*: *s* = ⟦*X*, α, μ⟧$_{meas}$ *qbs-meas X* α μ **by** *metis*
then interpret *qbs-meas X* α μ **by** *simp*
show *?thesis*
using *assms*(*2*) **by**(*auto simp*: *qbs-nn-integral-def2 hs*(*1*) *AEq-def intro*!: *nn-integral-cong-AE*)
qed

lemmas *qbs-nn-integral-cong-AEq* = *qbs-nn-integral-cong-AEq-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-nn-integral-cong-all-meas*:
assumes *s* ∈ *qbs-space* (*all-meas-qbs X*) ⋀*x*. *x* ∈ *qbs-space X* ⟹ *f x* = *g x*
shows *qbs-nn-integral s f* = *qbs-nn-integral s g*
using *qbs-nn-integral-cong-AEq-all-meas*[*OF assms*(*1*) *AEq-I2'*[*OF assms*]] **by**

simp

lemmas *qbs-nn-integral-cong* = *qbs-nn-integral-cong-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-nn-integral-const*:
$(\int^+_Q\ x.\ c\ \partial s) = c * qbs\text{-}l\ s\ (qbs\text{-}space\ (qbs\text{-}space\text{-}of\ s))$
by(*simp add*: *qbs-nn-integral-def2-l space-qbs-l*)

lemma *qbs-nn-integral-const-prob*:
assumes $s \in qbs\text{-}space\ (monadP\text{-}qbs\ X)$
shows $(\int^+_Q\ x.\ c\ \partial s) = c$
using *assms* **by**(*simp add*: *qbs-nn-integral-const prob-space.emeasure-space-1 qbs-l-prob-space space-qbs-l*)

lemma *qbs-nn-integral-add-all-meas*:
assumes $s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)$
and [*qbs*]:$f \in X \to_Q qbs\text{-}borel\ g \in X \to_Q qbs\text{-}borel$
shows $(\int^+_Q\ x.\ f\ x + g\ x\ \partial s) = (\int^+_Q\ x.\ f\ x\ \partial s) + (\int^+_Q\ x.\ g\ x\ \partial s)$
by(*auto simp*: *qbs-nn-integral-def2-l measurable-qbs-l-all-meas*[*OF assms*(*1*)] *intro*!: *nn-integral-add measurable-qbs-l*)

lemmas *qbs-nn-integral-add* = *qbs-nn-integral-add-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-nn-integral-cmult-all-meas*:
assumes $s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)$ **and** [*qbs*]:$f \in X \to_Q qbs\text{-}borel$
shows $(\int^+_Q\ x.\ c * f\ x\ \partial s) = c * (\int^+_Q\ x.\ f\ x\ \partial s)$
by(*auto simp*: *qbs-nn-integral-def2-l measurable-qbs-l-all-meas*[*OF assms*(*1*)] *intro*!: *nn-integral-cmult*)

lemmas *qbs-nn-integral-cmult* = *qbs-nn-integral-cmult-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-cong-AEq-all-meas*:
assumes [*qbs*]:$s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)\ f \in X \to_Q qbs\text{-}borel\ g \in X \to_Q qbs\text{-}borel$
and $AE_Q\ x\ in\ s.\ f\ x = g\ x$
shows *qbs-integral s f* = *qbs-integral s g*
using *assms*(*4*) **by**(*auto simp*: *qbs-integral-def2-l AEq-qbs-l measurable-qbs-l-all-meas*[*OF assms*(*1*)] *intro*!: *integral-cong-AE*)

lemmas *qbs-integral-cong-AEq* = *qbs-integral-cong-AEq-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-cong-all-meas*:
assumes $s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)\ \bigwedge x.\ x \in qbs\text{-}space\ X \Longrightarrow f\ x = g\ x$
shows *qbs-integral s f* = *qbs-integral s g*
by(*auto simp*: *qbs-integral-def2-l space-qbs-l-in-all-meas*[*OF assms*(*1*)] *assms*(*2*) *intro*!: *Bochner-Integration.integral-cong*)

lemmas *qbs-integral-cong* = *qbs-integral-cong-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-nonneg-AEq*:
fixes $f :: - \Rightarrow real$
shows $AE_Q\ x\ in\ s.\ 0 \leq f\ x \Longrightarrow 0 \leq qbs\text{-}integral\ s\ f$
by(*auto simp*: *qbs-integral-def2-l AEq-qbs-l intro*!: *integral-nonneg-AE*)

lemma *qbs-integral-nonneg-all-meas*:
fixes $f :: - \Rightarrow real$
assumes $s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)\ \bigwedge x.\ x \in qbs\text{-}space\ X \Longrightarrow 0 \leq f\ x$
shows $0 \leq qbs\text{-}integral\ s\ f$
by(*auto simp*: *qbs-integral-def2-l space-qbs-l-in-all-meas*[*OF assms*(*1*)] *assms*(*2*) *intro*!: *Bochner-Integration.integral-nonneg*)

lemmas *qbs-integral-nonneg* = *qbs-integral-nonneg-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-mono-AEq*:
fixes $f :: - \Rightarrow real$
assumes *qbs-integrable s f qbs-integrable s g* $AE_Q\ x\ in\ s.\ f\ x \leq g\ x$
shows *qbs-integral s f* $\leq$ *qbs-integral s g*
using *assms* **by**(*auto simp*: *qbs-integral-def2-l AEq-qbs-l qbs-integrable-iff-integrable intro*!: *integral-mono-AE*)

lemma *qbs-integral-mono-all-meas*:
fixes $f :: - \Rightarrow real$
assumes $s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)$
and *qbs-integrable s f qbs-integrable s g* $\bigwedge x.\ x \in qbs\text{-}space\ X \Longrightarrow f\ x \leq g\ x$
shows *qbs-integral s f* $\leq$ *qbs-integral s g*
by(*auto simp*: *qbs-integral-def2-l space-qbs-l-in-all-meas*[*OF assms*(*1*)] *qbs-integrable-iff-integrable*[*symmetric*] *assms intro*!: *integral-mono*)

lemmas *qbs-integral-mono* = *qbs-integral-mono-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-const-prob*:
assumes $s \in qbs\text{-}space\ (monadP\text{-}qbs\ X)$
shows $(\int_Q x.\ c\ \partial s) = c$
using *assms* **by**(*simp add*: *qbs-integral-def2-l monadP-qbs-space prob-space.prob-space*)

lemma
assumes *qbs-integrable s f qbs-integrable s g*
shows *qbs-integral-add*:$(\int_Q x.\ f\ x + g\ x\ \partial s) = (\int_Q x.\ f\ x\ \partial s) + (\int_Q x.\ g\ x\ \partial s)$
and *qbs-integral-diff*: $(\int_Q x.\ f\ x - g\ x\ \partial s) = (\int_Q x.\ f\ x\ \partial s) - (\int_Q x.\ g\ x\ \partial s)$
using *assms* **by**(*auto simp*: *qbs-integral-def2-l qbs-integrable-iff-integrable*[*symmetric*] *intro*!: *Bochner-Integration.integral-add Bochner-Integration.integral-diff*)

lemma [*simp*]:
fixes $c :: -::\{real\text{-}normed\text{-}field, second\text{-}countable\text{-}topology\}$
shows *qbs-integral-mult-right-zero*:$(\int_Q x.\ c * f\ x\ \partial s) = c * (\int_Q x.\ f\ x\ \partial s)$
and *qbs-integral-mult-left-zero*:$(\int_Q x.\ f\ x * c\ \partial s) = (\int_Q x.\ f\ x\ \partial s) * c$
and *qbs-integral-divide-zero*: $(\int_Q x.\ f\ x\ /\ c\ \partial s) = (\int_Q x.\ f\ x\ \partial s)\ /\ c$
by(*auto simp*: *qbs-integral-def2-l*)

lemma *qbs-integral-minus*[*simp*]: $(\int_Q x.\ -f\ x\ \partial s) = -(\int_Q x.\ f\ x\ \partial s)$
by(*auto simp*: *qbs-integral-def2-l*)

lemma [*simp*]:
shows *qbs-integral-scaleR-right*:$(\int_Q x.\ c *_R f\ x\ \partial s) = c *_R (\int_Q x.\ f\ x\ \partial s)$
and *qbs-integral-scaleR-left*: $(\int_Q x.\ f\ x *_R c\ \partial s) = (\int_Q x.\ f\ x\ \partial s) *_R c$
by(*auto simp*: *qbs-integral-def2-l*)

lemma [*simp*]:
shows *qbs-integral-inner-left*: *qbs-integrable s f* $\Longrightarrow (\int_Q x.\ f\ x \cdot c\ \partial s) = (\int_Q x.\ f\ x\ \partial s) \cdot c$
and *qbs-integral-inner-right*: *qbs-integrable s f* $\Longrightarrow$ $(\int_Q x.\ c \cdot f\ x\ \partial s) = c \cdot (\int_Q x.\ f\ x\ \partial s)$
by(*auto simp*: *qbs-integral-def2-l qbs-integrable-iff-integrable*)

lemma *integral-complex-of-real*[*simp*]: $(\int_Q x.\ complex\text{-}of\text{-}real\ (f\ x)\ \partial s) = of\text{-}real\ (\int_Q x.\ f\ x\ \partial s)$
by(*simp add*: *qbs-integral-def2-l*)

lemma *integral-cnj*[*simp*]: $(\int_Q x.\ cnj\ (f\ x)\ \partial s) = cnj\ (\int_Q x.\ f\ x\ \partial s)$
by(*simp add*: *qbs-integral-def2-l*)

lemma [*simp*]:
assumes *qbs-integrable s f*
shows *qbs-integral-Im*: $(\int_Q x.\ Im\ (f\ x)\ \partial s) = Im\ (\int_Q x.\ f\ x\ \partial s)$
and *qbs-integral-Re*: $(\int_Q x.\ Re\ (f\ x)\ \partial s) = Re\ (\int_Q x.\ f\ x\ \partial s)$
using *assms* **by**(*auto simp*: *qbs-integral-def2-l qbs-integrable-iff-integrable*)

lemma *qbs-integral-of-real*[*simp*]:*qbs-integrable s f* $\Longrightarrow (\int_Q x.\ of\text{-}real\ (f\ x)\ \partial s) = of\text{-}real\ (\int_Q x.\ f\ x\ \partial s)$
by(*auto simp*: *qbs-integral-def2-l qbs-integrable-iff-integrable*)

lemma [*simp*]:
assumes *qbs-integrable s f*
shows *qbs-integral-fst*: $(\int_Q x.\ fst\ (f\ x)\ \partial s) = fst\ (\int_Q x.\ f\ x\ \partial s)$
and *qbs-integral-snd*: $(\int_Q x.\ snd\ (f\ x)\ \partial s) = snd\ (\int_Q x.\ f\ x\ \partial s)$
using *assms* **by**(*auto simp*: *qbs-integral-def2-l qbs-integrable-iff-integrable*)

lemma *real-qbs-integral-def*:
assumes *qbs-integrable s f*
shows *qbs-integral s f* $= enn2real\ (\int^+_Q x.\ ennreal\ (f\ x)\ \partial s) - enn2real\ (\int^+_Q x.\ ennreal\ (-f\ x)\ \partial s)$
using *qbs-integrable-morphism-dest-all-meas*[*OF in-qbs-space-of-all-meas assms*] *assms*
by(*auto simp*: *qbs-integral-def2-l qbs-nn-integral-def2-l qbs-integrable-iff-integrable*[*symmetric*] *intro*!: *real-lebesgue-integral-def*)

lemma *Markov-inequality-qbs-prob*:

qbs-integrable s f $\Longrightarrow$ AE_Q *x in s.* $0 \leq f\ x \Longrightarrow 0 < c \Longrightarrow \mathcal{P}(x$ *in qbs-l s.* $c \leq f\ x)$ $\leq (\int_Q\ x.\ f\ x\ \partial s)\ /\ c$
by(*auto simp*: *qbs-integral-def2-l AEq-qbs-l qbs-integrable-iff-integrable intro*!: *integral-Markov-inequality-measure*[**where** *A*={}])

lemma *Chebyshev-inequality-qbs-prob*:
assumes $s \in$ *qbs-space* (*monadP-qbs X*)
and $f \in X \to_Q$ *qbs-borel qbs-integrable s* $(\lambda x.\ (f\ x)^2)$
and $0 < e$
shows $\mathcal{P}(x$ *in qbs-l s.* $e \leq |f\ x - (\int_Q\ x.\ f\ x\ \partial s)|) \leq (\int_Q\ x.\ (f\ x - (\int_Q\ x.\ f\ x\ \partial s))^2\ \partial s)\ /\ e^2$
using *prob-space.Chebyshev-inequality*[*OF qbs-l-prob-space*[*OF assms*(*1*)] - - *assms*(*4*),*of f*] *assms*(*2*,*3*)
by(*simp add*: *qbs-integral-def2-l qbs-integrable-iff-integrable lr-adjunction-correspondence measurable-qbs-l'*[*OF qbs-space-monadPM*[*OF assms*(*1*)]])

lemma *qbs-l-return-qbs*:
assumes $x \in$ *qbs-space X*
shows *qbs-l* (*return-qbs X x*) = *return* (*qbs-to-measure X*) *x*
proof −
interpret *qp*: *qbs-prob X* $\lambda r.$ *x return borel 0*
by(*auto simp*: *qbs-prob-def prob-space-return assms in-Mx-def real-distribution-def real-distribution-axioms-def*)
show *?thesis*
by(*simp add*: *qp.return-qbs*[*OF assms*] *distr-return qp.qbs-l*)
qed

lemma *qbs-l-bind-qbs-all-meas*:
assumes [*qbs*]: $s \in$ *qbs-space* (*all-meas-qbs X*) $f \in X \to_Q$ *all-meas-qbs Y*
shows *qbs-l* $(s \gg\!= f)$ = *qbs-l s* $\gg\!=_k$ *qbs-l* $\circ$ *f* (**is** *?lhs* = *?rhs*)
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** $\alpha\ \mu$
where *hs*: $s = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas X* $\alpha\ \mu$ **by** *metis*
then interpret *qs*: *qbs-meas X* $\alpha\ \mu$ **by** *simp*
from *rep-all-meas-qbs-Mx*[*OF qbs-morphism-Mx*[*OF assms*(*2*) *qs.in-Mx*]] **obtain** $\beta\ k$ **where**
hk: $f \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})\ \beta \in$ *qbs-Mx Y measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas Y* β (*k r*)
by *metis*
then interpret *sk*: *measure-kernel borel borel k* **by** *simp*
interpret *im*: *in-Mx Y* β **using** *hk*(*2*) **by**(*simp add*: *in-Mx-def*)

have *?lhs* = *distr* $(\mu \gg\!=_k k)$ (*qbs-to-measure Y*) β
by(*simp add*: *qs.bind-qbs-all-meas*[*OF hs*(*1*) *assms*(*2*) *hk*(*2*,*3*,*1*)] *qbs-meas.qbs-l*[*OF qs.bind-qbs-meas*[*OF hs*(*1*) *assms*(*2*) *hk*(*2*,*3*,*1*)]])
also have ... = $\mu \gg\!=_k$ $(\lambda x.$ *distr* (*k x*) (*qbs-to-measure Y*) $\beta)$
by(*auto intro*!: *sk.distr-bind-kernel simp*: *qs.mu-sets*)
also have ... = $\mu \gg\!=_k$ $(\lambda r.$ *qbs-l* $((f \circ \alpha)\ r))$
by(*simp add*: *qbs-meas.qbs-l*[*OF hk*(*4*)] *hk*(*1*))

also have ... = $\mu \gg\!=_k (\lambda r.\ (\lambda x.\ qbs\text{-}l\ (f\ x))\ (\alpha\ r))$ **by** *simp*
also have ... = *distr* μ (*qbs-to-measure* X) $\alpha \gg\!=_k (\lambda x.\ qbs\text{-}l\ (f\ x))$
using *l-preserves-morphisms*[*of* X *all-meas-qbs* Y] *assms*(*2*)
by(*auto intro*!: *measure-kernel.bind-kernel-distr*[*OF measure-kernel.measure-kernel-comp*[*OF qbs-l-measure-kernel-all-meas*],*symmetric*]
simp: *sets-eq-imp-space-eq*[*OF qs.mu-sets*])
also have ... = *?rhs*
by(*simp add*: *hs*(*1*) *qs.qbs-l comp-def*)
finally show *?thesis* **.**
qed

lemma *qbs-l-bind-qbs*:
assumes $s \in qbs\text{-}space\ (monadM\text{-}qbs\ X)$ $f \in X \to_Q monadM\text{-}qbs\ Y$
shows $qbs\text{-}l\ (s \gg\!= f) = qbs\text{-}l\ s \gg\!=_k qbs\text{-}l \circ f$
by(*auto intro*!: *qbs-l-bind-qbs-all-meas assms qbs-morphism-monadAD monadM-all-meas-space*)

lemma *qbs-l-bind-qbsP*:
assumes [*qbs*]: $s \in qbs\text{-}space\ (monadP\text{-}qbs\ X)$ $f \in X \to_Q monadP\text{-}qbs\ Y$
shows $qbs\text{-}l\ (s \gg\!= f) = qbs\text{-}l\ s \gg\!= qbs\text{-}l \circ f$
proof −
have $qbs\text{-}l\ (s \gg\!= f) = qbs\text{-}l\ s \gg\!=_k qbs\text{-}l \circ f$
by(*auto intro*!: *qbs-l-bind-qbs*[**where** $X{=}X$ **and** $Y{=}Y$] *qbs-space-monadPM qbs-morphism-monadPD*)
also have ... = $qbs\text{-}l\ s \gg\!= qbs\text{-}l \circ f$
using *qbs-l-measurable-prob qbs-morphism-imp-measurable*[*OF assms*(*2*)]
by(*auto intro*!: *bind-kernel-bind*[**where** $N{=}qbs\text{-}to\text{-}measure\ Y$] *measurable-prob-algebraD simp*: *measurable-qbs-l'*[*OF qbs-space-monadPM*[*OF assms*(*1*)]])
finally show *?thesis* **.**
qed

lemma *qbs-integrable-return*[*simp*, *intro*]:
fixes $f :: 'a \Rightarrow 'b$::{*second-countable-topology*,*banach*}
assumes $x \in qbs\text{-}space\ X$ $f \in X \to_Q qbs\text{-}borel$
shows $qbs\text{-}integrable\ (return\text{-}qbs\ X\ x)\ f$
using *assms* **by**(*auto simp*: *qbs-integrable-iff-integrable qbs-l-return-qbs*[*OF assms*(*1*)] *nn-integral-return space-L intro*!: *integrableI-bounded*)

lemma *qbs-integrable-bind-return-all-meas*:
fixes $f :: 'a \Rightarrow 'b$::{*second-countable-topology*,*banach*}
assumes [*qbs*]:$s \in qbs\text{-}space\ (all\text{-}meas\text{-}qbs\ X)$ $f \in Y \to_Q qbs\text{-}borel$ $g \in X \to_Q Y$
shows $qbs\text{-}integrable\ (s \gg\!= (\lambda x.\ return\text{-}qbs\ Y\ (g\ x)))\ f = qbs\text{-}integrable\ s\ (f \circ g)$ (**is** *?lhs* = *?rhs*)
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** $\alpha\ \mu$
where *hs*: $s = [\![X, \alpha, \mu]\!]_{meas}$ $qbs\text{-}meas\ X\ \alpha\ \mu$ **by** *metis*
then interpret *qs*: $qbs\text{-}meas\ X\ \alpha\ \mu$ **by** *simp*

have *1*:$return\text{-}qbs\ Y \circ (g \circ \alpha) = (\lambda r.\ [\![Y, g \circ \alpha, return\ borel\ r]\!]_{meas})$
by(*auto intro*!: *return-qbs-comp qbs-morphism-Mx*[*OF assms*(*3*)])

have *hb*: *qbs-meas Y* $(g \circ \alpha)$ μ *s* $\gg\!=$ $(\lambda x.$ *return-qbs Y* $(g\ x)) = [\![Y,\ g \circ \alpha,\ \mu]\!]_{meas}$
using *qbs-meas.bind-qbs-all-meas*[*OF hs*(*2*,*1*) *qbs-morphism-comp*[*OF assms*(*3*) *return-qbs-morphism-all-meas*] *qbs-morphism-Mx*[*OF assms*(*3*)] *prob-kernel.axioms*(*1*) *1*[*simplified comp-assoc*[*symmetric*]]]
qbs-meas.bind-qbs-meas[*OF hs*(*2*,*1*) *qbs-morphism-comp*[*OF assms*(*3*) *return-qbs-morphism-all-meas*] *qbs-morphism-Mx*[*OF assms*(*3*)] *prob-kernel.axioms*(*1*) *1*[*simplified comp-assoc*[*symmetric*]]]
by(*auto simp*: *prob-kernel-def' comp-def bind-kernel-return''*[*OF qs.mu-sets*])
have *?lhs* = *integrable* μ $(f \circ (g \circ \alpha))$
by(*auto intro*!: *qbs-meas.qbs-integrable-iff-integrable*[*OF hb*(*1*),*simplified comp-def*] *simp*: *hb*(*2*) *comp-def lr-adjunction-correspondence*[*symmetric*])
also have ... = *?rhs*
by(*auto simp*: *hs*(*1*) *comp-def lr-adjunction-correspondence*[*symmetric*] *intro*!: *qs.qbs-integrable-iff-integrable*[*symmetric*])
finally show *?thesis* **.**
qed

lemma *qbs-integrable-bind-return*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*second-countable-topology*,*banach*}
assumes *s* $\in$ *qbs-space* (*monadM-qbs X*) *f* $\in$ *Y* $\rightarrow_Q$ *qbs-borel g* $\in$ *X* $\rightarrow_Q$ *Y*
shows *qbs-integrable* (*s* $\gg\!=$ $(\lambda x.$ *return-qbs Y* $(g\ x))$) *f* = *qbs-integrable s* $(f \circ g)$
by(*intro qbs-integrable-bind-return-all-meas*[*of s X*] *assms monadM-all-meas-space*)

lemma *qbs-nn-integral-morphism*[*qbs*]: *qbs-nn-integral* $\in$ *monadM-qbs X* $\rightarrow_Q$ (*X* $\Rightarrow_Q$ *qbs-borel*) $\Rightarrow_Q$ *qbs-borel*
proof(*rule curry-preserves-morphisms*[*OF pair-qbs-morphismI*])
fix α β
assume *h*:α $\in$ *qbs-Mx* (*monadM-qbs X*) β $\in$ *qbs-Mx* (*X* $\Rightarrow_Q$ (*qbs-borel* :: *ennreal quasi-borel*))
from *rep-qbs-Mx-monadM*[*OF h*(*1*)] **obtain** *a k*
where *ak*: $\alpha = (\lambda r.\ [\![X,\ a,\ k\ r]\!]_{meas})$ *a* $\in$ *qbs-Mx X s-finite-kernel borel borel k* $\bigwedge r.$ *qbs-s-finite X a* (*k r*)
by *metis*
interpret *qbs-s-finite X a k r* **for** *r* **by** *fact*
have *1*:*borel-measurable* ((*borel* :: *real measure*) $\bigotimes_M$ (*borel* :: *real measure*)) = *qbs-borel* $\bigotimes_Q$ *qbs-borel* $\rightarrow_Q$ (*qbs-borel* :: *ennreal quasi-borel*)
by (*metis borel-prod qbs-borel-prod standard-borel.standard-borel-r-full-faithful standard-borel-ne-borel standard-borel-ne-def*)
show $(\lambda r.$ *qbs-nn-integral* (*fst* $(\alpha\ r,\ \beta\ r)$) (*snd* $(\alpha\ r,\ \beta\ r)$)) $\in$ *qbs-Mx qbs-borel*
unfolding *qbs-Mx-qbs-borel*
by(*rule measurable-cong*[**where** *f*=$\lambda r.\ \int^+ x.\ \beta\ r\ (a\ x)\ \partial k\ r$,*THEN iffD1*])
(*use h ak*(*2*) **in** *auto simp*: *qbs-nn-integral-def qbs-Mx-is-morphisms ak*(*1*) *1 intro*!: *s-finite-kernel.nn-integral-measurable-f*[*OF ak*(*3*)])
qed

lemma *qbs-nn-integral-morphism'*:
assumes [*qbs*,*measurable*]:*f* $\in$ *X* $\rightarrow_Q$ *qbs-borel*
shows $(\lambda x.$ *qbs-nn-integral x f*) $\in$ *all-meas-qbs X* $\rightarrow_Q$ *qbs-borel*

proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* (*all-meas-qbs X*)
from *rep-all-meas-qbs-Mx*[*OF this*] **obtain** *a k*
where *ak*: $\alpha = (\lambda r.\ [\![X,\ a,\ k\ r]\!]_{meas})$ $a \in$ *qbs-Mx X measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas X a* (*k r*)
by *blast*
interpret *qbs-meas X a k r* **for** *r* **by** *fact*
show $(\lambda x.$ *qbs-nn-integral x f*$) \circ \alpha \in$ *qbs-Mx borel*$_Q$
by(*auto simp*: *comp-def ak*(*1*) *qbs-nn-integral-def qbs-Mx-qbs-borel measurable-compose*[*OF* α*-measurable*]
intro!: *measure-kernel.nn-integral-measurable-kernel*[*OF ak*(*3*)])
qed

lemma *qbs-nn-integral-return*:
assumes $f \in X \to_Q$ *qbs-borel*
and $x \in$ *qbs-space X*
shows *qbs-nn-integral* (*return-qbs X x*) *f* = *f x*
using *assms* **by**(*auto intro*!: *nn-integral-return simp*: *qbs-nn-integral-def2-l qbs-l-return-qbs space-L lr-adjunction-correspondence*)

lemma *qbs-nn-integral-bind-all-meas*:
assumes [*qbs*]:*s* $\in$ *qbs-space* (*all-meas-qbs X*)
f $\in X \to_Q$ *all-meas-qbs Y g* $\in Y \to_Q$ *qbs-borel*
shows *qbs-nn-integral* (*s* $\gg\!=$ *f*) *g* = *qbs-nn-integral s* $(\lambda y.$ (*qbs-nn-integral* (*f y*) *g*)) (**is** *?lhs* = *?rhs*)
proof −
note [*qbs*] = *qbs-morphism-compose*[*OF assms*(*2*) *qbs-nn-integral-morphism'*[*OF assms*(*3*)]]
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** α μ
where *hs*: $s = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas X* α μ **by** *metis*
then interpret *qs*: *qbs-meas X* α μ **by** *simp*
from *rep-all-meas-qbs-Mx*[*OF qbs-morphism-Mx*[*OF assms*(*2*) *qs.in-Mx*]] **obtain** β *k*
where *hk*: $f \circ \alpha = (\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$ $\beta \in$ *qbs-Mx Y measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas Y* β (*k r*)
by *metis*
interpret *hk*: *qbs-meas Y* β *k r* **for** *r* **by** *fact*
interpret *sf*: *qbs-meas Y* β $\mu \gg\!=_k$ *k*
by(*rule qs.bind-qbs-meas*[*OF hs*(*1*) *assms*(*2*) *hk*(*2,3,1*)])
note *sf* = *qs.bind-qbs-all-meas*[*OF hs*(*1*) *assms*(*2*) *hk*(*2,3,1*)]
have *?lhs* = $(\int^+ x.\ g\ (\beta\ x)\ \partial(\mu \gg\!=_k k))$
by(*simp add*: *sf sf.qbs-nn-integral-def*)
also have ... = $(\int^+ r.\ (\int^+ y.\ g\ (\beta\ y)\ \partial(k\ r))\ \partial\mu)$
using *assms*(*3*) *hk*(*2*)
by(*auto intro*!: *measure-kernel.nn-integral-bind-kernel*[*OF hk*(*3*)] *qs.mu-sets simp*: *lr-adjunction-correspondence*)
also have ... = *?rhs*
using *fun-cong*[*OF hk*(*1*)] **by**(*auto simp*: *hs*(*1*) *qs.qbs-nn-integral-def hk.qbs-nn-integral-def*[*symmetric*]

intro!: *nn-integral-cong*)
finally show *?thesis* **.**
qed

lemmas *qbs-nn-integral-bind* = *qbs-nn-integral-bind-all-meas*[*OF monadM-all-meas-space qbs-morphism-monadAD*]

lemma *qbs-nn-integral-bind-return-all-meas*:
assumes [*qbs*]:$s \in$ *qbs-space* (*all-meas-qbs* Y) $f \in Z \to_Q$ *qbs-borel* $g \in Y \to_Q Z$
shows *qbs-nn-integral* ($s \gg\!=$ (λy. *return-qbs* Z (g y))) f = *qbs-nn-integral* s ($f \circ g$)
proof −
note *return-qbs-morphism-all-meas*[*qbs*]
show *?thesis*
by(*auto simp*: *qbs-nn-integral-bind-all-meas*[*OF assms*(*1*) - *assms*(*2*)] *qbs-nn-integral-return intro*!: *qbs-nn-integral-cong-all-meas*[*OF assms*(*1*)])
qed

lemmas *qbs-nn-integral-bind-return* = *qbs-nn-integral-bind-return-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-morphism*[*qbs*]:
qbs-integral ∈ *monadM-qbs* $X \to_Q$ ($X \Rightarrow_Q$ *qbs-borel*) $\Rightarrow_Q$ (*qbs-borel* :: ('*b* :: {*second-countable-topology*,*banach*}) *quasi-borel*)
proof(*rule curry-preserves-morphisms*[*OF pair-qbs-morphismI*])
fix α **and** γ :: - ⇒ - ⇒ '*b*
assume *h*:$\alpha \in$ *qbs-Mx* (*monadM-qbs* X) $\gamma \in$ *qbs-Mx* ($X \Rightarrow_Q$ *qbs-borel*)
from *rep-qbs-Mx-monadM*[*OF this*(*1*)] **obtain** β k
where *hk*: $\alpha = (\lambda r.\ [\![X, \beta, k\ r]\!]_{meas})$ $\beta \in$ *qbs-Mx* X *s-finite-kernel borel borel* k $\bigwedge r$. *qbs-s-finite* X β (k r)
by *metis*
interpret *qbs-s-finite* X β k r **for** r **by** *fact*
have *1*:*borel-measurable* ((*borel* :: *real measure*) $\bigotimes_M$ (*borel* :: *real measure*)) = *qbs-borel* $\bigotimes_Q$ *qbs-borel* $\to_Q$ (*qbs-borel* :: (- :: {*second-countable-topology*,*banach*}) *quasi-borel*)
by (*metis borel-prod qbs-borel-prod standard-borel.standard-borel-r-full-faithful standard-borel-ne-borel standard-borel-ne-def*)
show (λr. *qbs-integral* (*fst* (α r,γ r)) (*snd* (α r,γ r))) ∈ *qbs-Mx borel*$_Q$
unfolding *qbs-Mx-R*
by(*rule measurable-cong*[**where** f=λr. $\int$ x. γ r (β x) ∂k r,*THEN iffD1*], *insert h hk*(*2*))
(*auto simp*: *qbs-integral-def qbs-Mx-is-morphisms hk*(*1*) *1 intro*!: *s-finite-kernel.integral-measurable-f*[*OF hk*(*3*)])
qed

lemma *qbs-integral-morphism'*:
assumes [*qbs*,*measurable*]:$f \in X \to_Q$ *qbs-borel*
shows (λx. *qbs-integral* x f) ∈ *all-meas-qbs* $X \to_Q$ *qbs-borel*
proof(*rule qbs-morphismI*)

fix α
assume $\alpha \in$ *qbs-Mx* (*all-meas-qbs X*)
from *rep-all-meas-qbs-Mx*[*OF this*] **obtain** *a k*
where *ak*: $\alpha = (\lambda r.\ [\![X,\ a,\ k\ r]\!]_{meas})$ $a \in$ *qbs-Mx X measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas X a* (*k r*)
by *blast*
interpret *qbs-meas X a k r* **for** *r* **by** *fact*
show $(\lambda x.$ *qbs-integral x f*$) \circ \alpha \in$ *qbs-Mx borel*$_Q$
by(*auto simp*: *comp-def ak*(*1*) *qbs-integral-def qbs-Mx-qbs-borel measurable-compose*[*OF α-measurable*]
intro!: *measure-kernel.integral-measurable-kernel*[*OF ak*(*3*)])
qed

lemma *qbs-integral-return*:
assumes [*qbs*]:*f* $\in X \to_Q$ *qbs-borel x* $\in$ *qbs-space X*
shows *qbs-integral* (*return-qbs X x*) *f* = *f x*
by(*auto simp*: *qbs-integral-def2-l qbs-l-return-qbs lr-adjunction-correspondence*[*symmetric*] *space-L integral-return*)

lemma
assumes [*qbs*]: *s* $\in$ *qbs-space* (*all-meas-qbs X*) *f* $\in X \to_Q$ *all-meas-qbs Y g* $\in Y \to_Q$ *qbs-borel*
and *qbs-integrable s* $(\lambda x. \int_Q y.$ *norm* (*g y*) *∂f x*) AE_Q *x in s. qbs-integrable* (*f x*) *g*
shows *qbs-integrable-bind-all-meas*: *qbs-integrable* (*s* $\gg\!=$ *f*) *g* (**is** *?goal1*)
and *qbs-integral-bind-all-meas*:$(\int_Q y.\ g\ y\ \partial(s \gg\!= f)) = (\int_Q x. \int_Q y.\ g\ y\ \partial f\ x\ \partial s)$ (**is** *?lhs* = *?rhs*)
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** α μ
where *hs*: *s* = $[\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas X* α μ **by** *metis*
then interpret *qs*: *qbs-meas X* α μ **by** *simp*
from *rep-all-meas-qbs-Mx*[*OF qbs-morphism-Mx*[*OF assms*(*2*) *qs.in-Mx*]] **obtain** β *k*
where *hk*: *f* $\circ$ α = $(\lambda r.\ [\![Y,\ \beta,\ k\ r]\!]_{meas})$ $\beta \in$ *qbs-Mx Y measure-kernel borel borel k* $\bigwedge r.$ *qbs-meas Y* β (*k r*)
by *metis*
interpret *hk*: *qbs-meas Y* β *k r* **for** *r* **by** *fact*
note *sf* = *qs.bind-qbs-all-meas*[*OF hs*(*1*) *assms*(*2*) *hk*(*2*,*3*,*1*)]
have *g*[*measurable*]: $\bigwedge h\ M.\ h \in M \to_M$ *qbs-to-measure Y* $\Longrightarrow$ $(\lambda x.\ g\ (h\ x)) \in M \to_M$ *borel*
using *assms*(*3*) **by**(*auto simp*: *lr-adjunction-correspondence*)
interpret *qs2*: *qbs-meas Y* β μ $\gg\!=_k$ *k*
by(*rule qs.bind-qbs-meas*[*OF hs*(*1*) *assms*(*2*) *hk*(*2*,*3*,*1*)])
show *?goal1*
by(*auto simp*: *sf qs2.qbs-integrable-def intro*!: *measure-kernel.integrable-bind-kernel*[*OF hk*(*3*) *qs.mu-sets*])
(*insert qs.AEq-AE*[*OF assms*(*5*)[*simplified hs*(*1*)],*simplified fun-cong*[*OF hk*(*1*),*simplified*] *hk.qbs-integrable-def*] *assms*(*4*)[*simplified hs*(*1*) *qs.qbs-integrable-def fun-cong*[*OF hk*(*1*),*simplified*]],*auto simp*: *hs*(*1*) *qs.qbs-integrable-def hk.qbs-integral-def*)

have *?lhs* = $(\int r.\ g\ (\beta\ r)\ \partial(\mu \gg\!=_k\ k))$
by(*simp add*: *sf qs2.qbs-integral-def*)
also have ... = $(\int r.\ (\int l.\ g\ (\beta\ l)\partial k\ r)\ \partial\mu)$
using *qs.AEq-AE*[*OF assms*(*5*)[*simplified hs*(*1*)],*simplified fun-cong*[*OF hk*(*1*),*simplified*] *hk.qbs-integrable-def*] *assms*(*4*)[*simplified hs*(*1*) *qs.qbs-integrable-def fun-cong*[*OF hk*(*1*),*simplified*]]
by(*auto intro*!: *measure-kernel.integral-bind-kernel*[*OF hk*(*3*) *qs.mu-sets*] *simp*: *hk.qbs-integral-def*)
also have ... = $(\int r.\ (\int_Q y.\ g\ y\partial[\![Y,\ \beta,\ k\ r]\!]_{meas})\ \partial\mu)$
by(*auto intro*!: *Bochner-Integration.integral-cong simp*: *hk.qbs-integral-def*)
also have ... = *?rhs*
using *qbs-morphism-compose*[*OF assms*(*2*) *qbs-integral-morphism'*[*OF assms*(*3*)]]
by(*auto simp*: *hs*(*1*) *qs.qbs-integral-def fun-cong*[*OF hk*(*1*),*simplified comp-def*])
finally show *?lhs* = *?rhs* **.**
qed

lemmas *qbs-integrable-bind* = *qbs-integrable-bind-all-meas*[*OF monadM-all-meas-space qbs-morphism-monadAD*]
lemmas *qbs-integral-bind* = *qbs-integral-bind-all-meas*[*OF monadM-all-meas-space qbs-morphism-monadAD*]

lemma *qbs-integral-bind-return-all-meas*:
assumes [*qbs*]:$s \in$ *qbs-space* (*all-meas-qbs Y*) $f \in Z \to_Q$ *qbs-borel* $g \in Y \to_Q Z$
shows *qbs-integral* $(s \gg\!=\ (\lambda y.$ *return-qbs Z* $(g\ y)))\ f =$ *qbs-integral s* $(f \circ g)$
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** $\alpha\ \mu$
where *hs*: $s = [\![Y,\ \alpha,\ \mu]\!]_{meas}$ *qbs-meas Y* $\alpha\ \mu$ **by** *metis*
then interpret *qs*: *qbs-meas Y* $\alpha\ \mu$ **by** *simp*
note [*qbs*] = *qbs-morphism-compose*[*OF assms*(*3*) *return-qbs-morphism-all-meas*]
have *hb*: *qbs-meas Z* $(g \circ \alpha)\ \mu\ s \gg\!=\ (\lambda y.$ *return-qbs Z* $(g\ y)) = [\![Z,\ g \circ \alpha,\ \mu]\!]_{meas}$
using *qs.bind-qbs-meas*[*OF hs*(*1*) *- qbs-morphism-Mx*[*OF assms*(*3*) *qs.in-Mx*] *prob-kernel.axioms*(*1*) *return-qbs-comp*[*OF qbs-morphism-Mx*[*OF assms*(*3*) *qs.in-Mx*],*simplified comp-assoc*[*symmetric*] *comp-def*[*of - g*]],*simplified prob-kernel-def'*]
by(*auto simp*: *qs.bind-qbs-all-meas*[*OF hs*(*1*) *- qbs-morphism-Mx*[*OF assms*(*3*) *qs.in-Mx*] *prob-kernel.axioms*(*1*) *return-qbs-comp*[*OF qbs-morphism-Mx*[*OF assms*(*3*) *qs.in-Mx*],*simplified comp-assoc*[*symmetric*] *comp-def*[*of - g*]],*simplified prob-kernel-def'*] *bind-kernel-return''*[*OF qs.mu-sets*])
show *?thesis*
by(*simp add*: *hb*(*2*) *qbs-meas.qbs-integral-def*[*OF hb*(*1*)] *qs.qbs-integral-def*[*simplified hs*(*1*)[*symmetric*]])
qed

lemmas *qbs-integral-bind-return* = *qbs-integral-bind-return-all-meas*[*OF monadM-all-meas-space*]

4.1.13 Binary Product Measures

definition *qbs-pair-measure* :: ['*a qbs-measure*, '*b qbs-measure*] ⇒ ('*a* × '*b*) *qbs-measure* (**infix** $\bigotimes_{Qmes}$ *80*) **where**
qbs-pair-measure-def':*qbs-pair-measure p q* ≡ (*p* ≫= (λ*x*. *q* ≫= (λ*y*. *return-qbs*

(*qbs-space-of p* $\bigotimes_Q$ *qbs-space-of q*) (*x*, *y*))))

context *pair-qbs-s-finites*
begin

interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

lemma
shows *qbs-pair-measure*: $[\![X, \alpha, \mu]\!]_{meas} \bigotimes_{Qmes} [\![Y, \beta, \nu]\!]_{meas} = [\![X \bigotimes_Q Y,$ *map-prod* α β ∘ *rr.from-real*, *distr* ($\mu \bigotimes_M \nu$) *borel rr.to-real*$]\!]_{meas}$
and *qbs-pair-measure-s-finite*: *qbs-s-finite* ($X \bigotimes_Q Y$) (*map-prod* α β ∘ *rr.from-real*) (*distr* ($\mu \bigotimes_M \nu$) *borel rr.to-real*)
by(*simp-all add*: *qbs-pair-measure-def' pq1.qbs-l pq2.qbs-l qbs-bind-bind-return-pq qbs-s-finite-axioms*)

lemma *qbs-l-qbs-pair-measure*:
qbs-l ($[\![X, \alpha, \mu]\!]_{meas} \bigotimes_{Qmes} [\![Y, \beta, \nu]\!]_{meas}$) = *distr* ($\mu \bigotimes_M \nu$) (*qbs-to-measure* ($X \bigotimes_Q Y$)) (*map-prod* α β)
by(*simp add*: *qbs-pair-measure qbs-l distr-distr comp-assoc*)

lemma *qbs-nn-integral-pair-measure*:
assumes [*qbs*]:*f* ∈ $X \bigotimes_Q Y \to_Q$ *qbs-borel*
shows ($\int^+_Q z.\ f\ z\ \partial([\![X, \alpha, \mu]\!]_{meas} \bigotimes_{Qmes} [\![Y, \beta, \nu]\!]_{meas})$) = ($\int^+ z.$ (*f* ∘ *map-prod* α β) *z* $\partial(\mu \bigotimes_M \nu)$)
using *assms* **by**(*simp add*: *qbs-nn-integral-def2 qbs-pair-measure distr-distr comp-assoc nn-integral-distr lr-adjunction-correspondence*)

lemma *qbs-integral-pair-measure*:
assumes [*qbs*]:*f* ∈ $X \bigotimes_Q Y \to_Q$ *qbs-borel*
shows ($\int_Q z.\ f\ z\ \partial([\![X, \alpha, \mu]\!]_{meas} \bigotimes_{Qmes} [\![Y, \beta, \nu]\!]_{meas})$) = ($\int z.$ (*f* ∘ *map-prod* α β) *z* $\partial(\mu \bigotimes_M \nu)$)
using *assms* **by**(*simp add*: *qbs-integral-def2 qbs-pair-measure distr-distr comp-assoc integral-distr lr-adjunction-correspondence*)

lemma *qbs-pair-measure-integrable-eq*:
fixes *f* :: - ⇒ -::{*second-countable-topology*,*banach*}
shows *qbs-integrable* ($[\![X, \alpha, \mu]\!]_{meas} \bigotimes_{Qmes} [\![Y, \beta, \nu]\!]_{meas}$) *f* ⟷ *f* ∈ $X \bigotimes_Q Y \to_Q$ *qbs-borel* ∧ *integrable* ($\mu \bigotimes_M \nu$) (*f* ∘ (*map-prod* α β)) (**is** *?h* ⟷ *?h1* ∧ *?h2*)
proof *safe*
assume *h*: *?h*
show *?h1*
by(*auto intro*!: *qbs-integrable-morphism-dest*[*OF - h*] *simp*: *qbs-pair-measure-def'*)
have *1*:*integrable* (*distr* ($\mu \bigotimes_M \nu$) *borel* (*to-real-on* (*borel* $\bigotimes_M$ *borel*))) (*f* ∘ (*map-prod* α β ∘ *from-real-into* (*borel* $\bigotimes_M$ *borel*)))
using *h*[*simplified qbs-pair-measure*] **by**(*simp add*: *qbs-integrable-def*[*of f*] *comp-def*[*of f*])

have *integrable* $(\mu \bigotimes_M \nu)$ $(\lambda x.\ (f \circ (\textit{map-prod}\ \alpha\ \beta \circ \textit{from-real-into}\ (\textit{borel} \bigotimes_M \textit{borel}))))\ (\textit{to-real-on}\ (\textit{borel} \bigotimes_M \textit{borel})\ x))$
by(*intro integrable-distr*[*OF - 1*]) *simp*
thus *?h2*
by(*simp add*: *comp-def*)
next
assume *h*: *?h1 ?h2*
then show *?h*
by(*simp add*: *qbs-pair-measure qbs-integrable-def*) (*simp add*: *lr-adjunction-correspondence integrable-distr-eq*[*of rr.to-real* $\mu \bigotimes_M \nu$ *borel* $\lambda x.\ f\ (\textit{map-prod}\ \alpha\ \beta\ (\textit{rr.from-real}\ x))$] *comp-def*)
qed

end

lemmas(**in** *pair-qbs-probs*) *qbs-pair-measure-prob* = *qbs-prob-axioms*

context
fixes *X Y p q*
assumes *p*[*qbs*]:$p \in \textit{qbs-space}\ (\textit{monadM-qbs}\ X)$ **and** *q*[*qbs*]:$q \in \textit{qbs-space}\ (\textit{monadM-qbs}\ Y)$
begin

lemma *qbs-pair-measure-def*: $p \bigotimes_{Qmes} q = p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ \textit{return-qbs}\ (X \bigotimes_Q Y)\ (x,y)))$
by(*simp add*: *qbs-space-of-in*[*OF p*] *qbs-space-of-in*[*OF q*] *qbs-pair-measure-def'*)

lemma *qbs-pair-measure-def2*: $p \bigotimes_{Qmes} q = q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ \textit{return-qbs}\ (X \bigotimes_Q Y)\ (x,y)))$
by(*simp add*: *bind-qbs-return-rotate qbs-pair-measure-def*)

lemma
assumes $f \in X \bigotimes_Q Y \to_Q \textit{monadM-qbs}\ Z$
shows *qbs-pair-bind-bind-return1'*:$q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) = p \bigotimes_{Qmes} q \gg\!= f$
and *qbs-pair-bind-bind-return2'*:$p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ (x,y))) = p \bigotimes_{Qmes} q \gg\!= f$
by(*simp-all add*: *qbs-bind-bind-return1*[*OF assms*] *qbs-bind-bind-return2*[*OF assms*] *bind-qbs-return-rotate qbs-pair-measure-def*)

lemma
assumes [*qbs*]:$f \in X \to_Q \textit{exp-qbs}\ Y\ (\textit{monadM-qbs}\ Z)$
shows *qbs-pair-bind-bind-return1''*: $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ x\ y)) = p \bigotimes_{Qmes} q \gg\!= (\lambda x.\ f\ (\textit{fst}\ x)\ (\textit{snd}\ x))$
and *qbs-pair-bind-bind-return2''*: $p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ f\ x\ y)) = p \bigotimes_{Qmes} q \gg\!= (\lambda x.\ f\ (\textit{fst}\ x)\ (\textit{snd}\ x))$
by(*auto intro*!: *qbs-pair-bind-bind-return1'*[**where** $f{=}\lambda x.\ f\ (\textit{fst}\ x)\ (\textit{snd}\ x)$,*simplified*] *qbs-pair-bind-bind-return2'*[**where** $f{=}\lambda x.\ f\ (\textit{fst}\ x)\ (\textit{snd}\ x)$,*simplified*] *uncurry-preserves-morphisms*) *qbs*

lemma *qbs-nn-integral-Fubini-fst*:
assumes [*qbs*]:$f \in X \bigotimes_Q Y \to_Q$ *qbs-borel*
shows $(\int^+_Q x.\ \int^+_Q y.\ f\ (x,y)\ \partial q\ \partial p) = (\int^+_Q z.\ f\ z\ \partial(p \bigotimes_{Qmes} q))$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+_Q x.\ \int^+_Q y.$ *qbs-nn-integral* (*return-qbs* $(X \bigotimes_Q Y)\ (x,\ y))\ f\ \partial q\ \partial p)$
by(*auto intro*!: *qbs-nn-integral-cong p q simp*: *qbs-nn-integral-return*)
also have ... = *?rhs*
by(*auto intro*!: *qbs-nn-integral-cong*[*OF p*] *simp*:*qbs-nn-integral-bind*[*OF q - assms*] *qbs-nn-integral-bind*[*OF p - assms*] *qbs-pair-measure-def*)
finally show *?thesis* .
qed

lemma *qbs-nn-integral-Fubini-snd*:
assumes [*qbs*]:$f \in X \bigotimes_Q Y \to_Q$ *qbs-borel*
shows $(\int^+_Q y.\ \int^+_Q x.\ f\ (x,y)\ \partial p\ \partial q) = (\int^+_Q z.\ f\ z\ \partial(p \bigotimes_{Qmes} q))$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+_Q y.\ \int^+_Q x.$ *qbs-nn-integral* (*return-qbs* $(X \bigotimes_Q Y)\ (x,\ y))\ f\ \partial p\ \partial q)$
by(*auto intro*!: *qbs-nn-integral-cong p q simp*: *qbs-nn-integral-return*)
also have ... = *?rhs*
by(*auto intro*!: *qbs-nn-integral-cong*[*OF q*] *simp*:*qbs-nn-integral-bind*[*OF q - assms*] *qbs-nn-integral-bind*[*OF p - assms*] *qbs-pair-measure-def2*)
finally show *?thesis* .
qed

lemma *qbs-ennintegral-indep-mult*:
assumes [*qbs*]: $f \in X \to_Q$ *qbs-borel* $g \in Y \to_Q$ *qbs-borel*
shows $(\int^+_Q z.\ f\ (fst\ z) * g\ (snd\ z)\ \partial(p \bigotimes_{Qmes} q)) = (\int^+_Q x.\ f\ x\ \partial p) * (\int^+_Q y.\ g\ y\ \partial q)$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+_Q x.\ \int^+_Q y\ .f\ x * g\ y\ \partial q\ \partial p)$
using *qbs-nn-integral-Fubini-fst*[**where** $f{=}\lambda z.\ f\ (fst\ z) * g\ (snd\ z)$] **by** *simp*
also have ... = $(\int^+_Q x.\ f\ x * \int^+_Q y\ .\ g\ y\ \partial q\ \partial p)$
by(*simp add*: *qbs-nn-integral-cmult*[*OF q*])
also have ... = *?rhs*
by(*simp add*: *qbs-nn-integral-cmult*[*OF p*] *ab-semigroup-mult-class.mult.commute*[**where** *b*=*qbs-nn-integral q g*])
finally show *?thesis* .
qed

end

lemma *qbs-l-qbs-pair-measure*:
assumes *standard-borel M standard-borel N*
defines $X \equiv$ *measure-to-qbs M* **and** $Y \equiv$ *measure-to-qbs N*

assumes [*qbs*]: $p \in$ *qbs-space* (*monadM-qbs* X) $q \in$ *qbs-space* (*monadM-qbs* Y)
shows *qbs-l* ($p \bigotimes_{Qmes} q$) = *qbs-l* $p \bigotimes_M$ *qbs-l* q
proof −
obtain α μ β ν
where *hp*: $p = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-s-finite* X α μ
and *hq*: $q = [\![Y, \beta, \nu]\!]_{meas}$ *qbs-s-finite* Y β ν
using *rep-qbs-space-monadM assms*(5,6) **by** *meson*
have *1*:*sets* (*qbs-to-measure* ($X \bigotimes_Q Y$)) = *sets* ($M \bigotimes_M N$)
by(*auto simp*: *r-preserves-product*[*symmetric*] *X-def Y-def intro*!: *standard-borel.lr-sets-ident pair-standard-borel assms*)
have *qbs-l* ($p \bigotimes_{Qmes} q$) = *qbs-l* $p \gg\!=_k$ *qbs-l* $\circ$ ($\lambda x.\ q \gg\!= (\lambda y.$ *return-qbs* ($X \bigotimes_Q Y$) (x,y)))
by(*auto simp*: *qbs-pair-measure-def*[*of p X q Y*] *intro*!: *qbs-l-bind-qbs*[*of - X - X* $\bigotimes_Q$ *Y*])
also have ... = *qbs-l* $p \gg\!=_k$ ($\lambda x.$ *qbs-l* ($q \gg\!= (\lambda y.$ *return-qbs* ($X \bigotimes_Q Y$) (x, y))))
by(*simp add*: *comp-def*)
also have ... = *qbs-l* $p \gg\!=_k$ ($\lambda x.$ *qbs-l* $q \gg\!=_k$ *qbs-l* $\circ$ ($\lambda y.$ *return-qbs* ($X \bigotimes_Q Y$) (x, y)))
by(*auto intro*!: *bind-kernel-cong-All qbs-l-bind-qbs*[*of - Y - X* $\bigotimes_Q$ *Y*] *simp*: *space-qbs-l-in*[*OF assms*(5)])
also have ... = *qbs-l* $p \gg\!=_k$ ($\lambda x.$ *qbs-l* $q \gg\!=_k$ ($\lambda y.$ *return* (*qbs-to-measure* ($X \bigotimes_Q Y$)) (x, y)))
by(*auto simp*: *comp-def space-qbs-l-in*[*OF assms*(6)] *space-qbs-l-in*[*OF assms*(5)] *qbs-l-return-qbs intro*!: *bind-kernel-cong-All*)
also have ... = *qbs-l* $p \gg\!=_k$ ($\lambda x.$ *qbs-l* $q \gg\!=_k$ ($\lambda y.$ *return* ($M \bigotimes_M N$) (x, y)))
by(*simp add*: *return-cong*[*OF 1*])
also have ... = *qbs-l* $p \gg\!=_k$ ($\lambda x.$ *qbs-l* $q \gg\!=_k$ ($\lambda y.$ *return* (*qbs-l* $p \bigotimes_M$ *qbs-l* q) (x, y)))
by(*auto cong*: *return-cong sets-pair-measure-cong simp*: *sets-qbs-l*[*OF assms*(5)] *standard-borel.lr-sets-ident*[*OF assms*(1)] *sets-qbs-l*[*OF assms*(6)] *standard-borel.lr-sets-ident*[*OF assms*(2)] *X-def Y-def*)
also have ... = *qbs-l* $p \bigotimes_M$ *qbs-l* q
by(*auto intro*!: *pair-measure-eq-bind-s-finite*[*symmetric*] *qbs-l-s-finite assms*)
finally show *?thesis* **.**
qed

lemma *qbs-pair-measure-morphism*[*qbs*]: *qbs-pair-measure* $\in$ *monadM-qbs* $X \to_Q$ *monadM-qbs* $Y \Rightarrow_Q$ *monadM-qbs* ($X \bigotimes_Q Y$)
by(*rule curry-preserves-morphisms*[*OF qbs-morphism-cong'*[**where** f=(λ(p,q). (p $\gg\!=$ ($\lambda x.$ $q \gg\!=$ ($\lambda y.$ *return-qbs* ($X \bigotimes_Q Y$) (x, y)))))]])
(*auto simp*: *pair-qbs-space qbs-pair-measure-def*)

lemma *qbs-pair-measure-morphismP*: *qbs-pair-measure* $\in$ *monadP-qbs* $X \to_Q$ *monadP-qbs* $Y \Rightarrow_Q$ *monadP-qbs* ($X \bigotimes_Q Y$)
proof −
note [*qbs*] = *return-qbs-morphismP bind-qbs-morphismP*
show *?thesis*
by(*rule curry-preserves-morphisms*[*OF qbs-morphism-cong'*[**where** f=(λ(p,q).

$(p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ return\text{-}qbs\ (X \bigotimes_Q\ Y)\ (x,\ y)))))]])$
(*auto simp*: *pair-qbs-space qbs-pair-measure-def*[*OF qbs-space-monadPM qbs-space-monadPM*])
qed

lemma *qbs-nn-integral-indep1*:
assumes [*qbs*]:$p \in$ *qbs-space* (*monadM-qbs X*) $q \in$ *qbs-space* (*monadP-qbs X*) $f \in X \to_Q$ *qbs-borel*
shows $(\int^+_Q\ z.\ f\ (fst\ z)\ \partial(p \bigotimes_{Qmes}\ q)) = (\int^+_Q\ x.\ f\ x\ \partial p)$
proof −
obtain $Y\ \beta\ \nu$ **where** *hq*:
$q = [\![Y,\ \beta,\ \nu]\!]_{meas}$ *qbs-prob* $Y\ \beta\ \nu$
using *rep-qbs-space-monadP*[*OF assms*(*2*)] **by** *blast*
then interpret *qbs-prob* $Y\ \beta\ \nu$ **by** *simp*
show *?thesis*
by(*simp add*: *qbs-nn-integral-const-prob*[*OF in-space-monadP*] *qbs-nn-integral-Fubini-snd*[*OF assms*(*1*) *in-space-monadM*,*symmetric*] *hq*(*1*))
qed

lemma *qbs-nn-integral-indep2*:
assumes [*qbs*]:$q \in$ *qbs-space* (*monadM-qbs Y*) $p \in$ *qbs-space* (*monadP-qbs X*) $f \in Y \to_Q$ *qbs-borel*
shows $(\int^+_Q\ z.\ f\ (snd\ z)\ \partial(p \bigotimes_{Qmes}\ q)) = (\int^+_Q\ y.\ f\ y\ \partial q)$
proof −
obtain $X\ \alpha\ \mu$ **where** *hp*:
$p = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-prob* $X\ \alpha\ \mu$
using *rep-qbs-space-monadP*[*OF assms*(*2*)] **by** *metis*
then interpret *qbs-prob* $X\ \alpha\ \mu$ **by** *simp*
show *?thesis*
by(*simp add*: *qbs-nn-integral-const-prob*[*OF in-space-monadP*] *qbs-nn-integral-Fubini-snd*[*OF in-space-monadM assms*(*1*),*symmetric*] *hp*(*1*))
qed

context
begin

interpretation *rr* : *standard-borel-ne borel* $\bigotimes_M$ *borel* :: (*real* × *real*) *measure*
by(*auto intro*!: *pair-standard-borel-ne*)

lemma *qbs-integrable-pair-swap*:
fixes f :: - ⇒ -::{*second-countable-topology*,*banach*}
assumes $p \in$ *qbs-space* (*monadM-qbs X*) $q \in$ *qbs-space* (*monadM-qbs Y*)
and *qbs-integrable* $(p \bigotimes_{Qmes}\ q)\ f$
shows *qbs-integrable* $(q \bigotimes_{Qmes}\ p)\ (\lambda(x,y).\ f\ (y,x))$
proof −
obtain $\alpha\ \mu\ \beta\ \nu$
where *hp*: $p = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-s-finite* $X\ \alpha\ \mu$
and *hq*: $q = [\![Y,\ \beta,\ \nu]\!]_{meas}$ *qbs-s-finite* $Y\ \beta\ \nu$
using *rep-qbs-space-monadM assms* **by** *meson*
interpret *p1*: *pair-qbs-s-finites* $X\ \alpha\ \mu\ Y\ \beta\ \nu$

by(*simp add*: *pair-qbs-s-finites-def hq hp*)
interpret *p2*: *pair-qbs-s-finites* $Y\ \beta\ \nu\ X\ \alpha\ \mu$
by(*simp add*: *pair-qbs-s-finites-def hq hp*)
have $*$:$((\lambda(y,\ x).\ f\ (x,\ y)) \circ$ *map-prod* $\beta\ \alpha) = (\lambda(y,x).\ (\lambda(a,\ b).\ f\ (\alpha\ a,\ \beta\ b))$ $(x,y))$
by *auto*
show *?thesis*
using *assms*(*3*) *integrable-product-swap-s-finite*[*OF p1.pq1.s-finite-measure-axioms p1.pq2.s-finite-measure-axioms*,*of f* $\circ$ *map-prod* $\alpha\ \beta$]
by(*auto simp*: *p1.qbs-pair-measure-integrable-eq p2.qbs-pair-measure-integrable-eq hp hq* $*$)
qed

lemma *qbs-integrable-pair1′*:
fixes *f* :: - $\Rightarrow$ -::{*second-countable-topology*,*banach*}
assumes [*qbs*]:$p \in$ *qbs-space* (*monadM-qbs X*)
$q \in$ *qbs-space* (*monadM-qbs Y*)
$f \in X \bigotimes_Q Y \to_Q$ *qbs-borel*
qbs-integrable p $(\lambda x.\ \int_Q y.$ *norm* $(f\ (x,y))\ \partial q)$
and AE_Q *x in p. qbs-integrable q* $(\lambda y.\ f\ (x,y))$
shows *qbs-integrable* $(p \bigotimes_{Qmes} q)\ f$
proof −
obtain $\alpha\ \mu\ \beta\ \nu$
where *hp*: $p = [\![X,\ \alpha,\ \mu]\!]_{meas}$ *qbs-s-finite X* $\alpha\ \mu$
and *hq*: $q = [\![Y,\ \beta,\ \nu]\!]_{meas}$ *qbs-s-finite Y* $\beta\ \nu$
using *rep-qbs-space-monadM assms*(*1*,*2*) **by** *meson*
then interpret *pqs*: *pair-qbs-s-finites X* $\alpha\ \mu\ Y\ \beta\ \nu$
by(*simp add*: *pair-qbs-s-finites-def*)
have [*measurable*]: $f \in$ *borel-measurable* (*qbs-to-measure* $(X \bigotimes_Q Y)$)
by(*simp add*: *lr-adjunction-correspondence*[*symmetric*])
show *?thesis*
using *assms*(*4*) *pqs.pq1.AEq-AE*[*OF assms*(*5*)[*simplified hp*(*1*)]]
by(*auto simp*: *pqs.qbs-integrable-def pqs.qbs-pair-measure hp*(*1*) *hq*(*1*) *integrable-distr-eq pqs.pq2.qbs-integrable-def pqs.pq1.qbs-integrable-def pqs.pq2.qbs-integral-def intro*!: *s-finite-measure.Fubini-integrable′ pqs.pq2.s-finite-measure-axioms*)
qed

lemma
assumes $p \in$ *qbs-space* (*monadM-qbs X*) $q \in$ *qbs-space* (*monadM-qbs Y*)
assumes *qbs-integrable* $(p \bigotimes_{Qmes} q)\ f$
shows *qbs-integrable-pair1D1′*: *qbs-integrable p* $(\lambda x.\ \int_Q y.\ f\ (x,y)\ \partial q)$ (**is** *?g1*)
and *qbs-integrable-pair1D1-norm′*: *qbs-integrable p* $(\lambda x.\ \int_Q y.$ *norm* $(f\ (x,y))$ $\partial q)$ (**is** *?g2*)
and *qbs-integrable-pair1D2′*: AE_Q *x in p. qbs-integrable q* $(\lambda y.\ f\ (x,y))$ (**is** *?g3*)
and *qbs-integrable-pair2D1′*: *qbs-integrable q* $(\lambda y.\ \int_Q x.\ f\ (x,y)\ \partial p)$ (**is** *?g4*)
and *qbs-integrable-pair2D1-norm′*: *qbs-integrable q* $(\lambda y.\ \int_Q x.$ *norm* $(f\ (x,y))$

∂p) (**is** *?g5*)
  and *qbs-integrable-pair2D2'*: AE_Q *y in q. qbs-integrable p* ($\lambda x.$ *f* (*x*,*y*)) (**is** *?g6*)
  and *qbs-integral-Fubini-fst'*: ($\int_Q$ *x.* $\int_Q$ *y. f* (*x*,*y*) ∂q ∂p) = ($\int_Q$ *z. f z* ∂(*p* $\bigotimes_{Qmes}$ *q*)) (**is** *?g7*)
  and *qbs-integral-Fubini-snd'*: ($\int_Q$ *y.* $\int_Q$ *x. f* (*x*,*y*) ∂p ∂q) = ($\int_Q$ *z. f z* ∂(*p* $\bigotimes_{Qmes}$ *q*)) (**is** *?g8*)
proof −
  obtain α μ β ν
    where *hp*: *p* = $[\![X, \alpha, \mu]\!]_{meas}$ *qbs-s-finite X* α μ
      and *hq*: *q* = $[\![Y, \beta, \nu]\!]_{meas}$ *qbs-s-finite Y* β ν
    using *rep-qbs-space-monadM assms* **by** *meson*
  then interpret *pqs*: *pair-qbs-s-finites X* α μ *Y* β ν
    by(*simp add*: *pair-qbs-s-finites-def*)
  have [*qbs*]:*p* ∈ *qbs-space* (*monadM-qbs X*) *q* ∈ *qbs-space* (*monadM-qbs Y*)
    by(*simp-all add*: *hp hq*)
  note *qbs-pair-measure-morphism*[*qbs*]
  have *f*[*qbs*]:*f* ∈ *X* $\bigotimes_Q$ *Y* →$_Q$ *qbs-borel*
    by(*auto intro*!: *qbs-integrable-morphism-dest*[*OF - assms*(*3*)])
  have [*measurable*]: *f* ∈ *borel-measurable* (*qbs-to-measure* (*X* $\bigotimes_Q$ *Y*))
    by(*simp add*: *lr-adjunction-correspondence*[*symmetric*])
  show *?g1 ?g2 ?g4 ?g5*
    using *assms*
      by(*auto simp*: *hp*(*1*) *hq*(*1*) *pqs.qbs-integrable-def pqs.qbs-pair-measure integrable-distr-eq pqs.pq1.qbs-integrable-def pqs.pq2.qbs-integrable-def pqs.pq2.qbs-integral-def pqs.pq1.qbs-integral-def intro*!: *Bochner-Integration.integrable-cong*[**where** *g*=$\lambda r.$ $\int_Q$ *y. f* (α *r*, *y*) $\partial$$[\![Y, \beta, \nu]\!]_{meas}$ **and** *f*=$\lambda x.$ $\int$ *y. f* (α *x*, β *y*) $\partial\nu$ **and** *N0*=μ,*THEN iffD1*] *Bochner-Integration.integrable-cong*[**where** *g*=$\lambda r.$ $\int_Q$ *x. f* (*x*, β *r*) $\partial$$[\![X, \alpha, \mu]\!]_{meas}$ **and** *f*=$\lambda y.$ $\int$ *x. f* (α *x*, β *y*) $\partial\mu$ **and** *N0*=ν,*THEN iffD1*])
    (*auto intro*!: *pqs.pq2.integrable-fst''*[*of* μ] *integrable-snd-s-finite*[*OF pqs.pq1.s-finite-measure-axioms pqs.pq2.s-finite-measure-axioms*] *simp*: *map-prod-def split-beta'*)
  show *?g3 ?g6*
    using *assms*
    by(*auto simp*: *hp*(*1*) *pqs.pq1.AEq-AE-iff hq*(*1*) *pqs.pq2.AEq-AE-iff pqs.qbs-integrable-def pqs.qbs-pair-measure integrable-distr-eq*)
      (*auto simp*: *pqs.pq1.qbs-integrable-def pqs.pq2.qbs-integrable-def map-prod-def split-beta' intro*!: *pqs.pq2.AE-integrable-fst'' AE-integrable-snd-s-finite*[*OF pqs.pq1.s-finite-measure-axioms pqs.pq2.s-finite-measure-axioms*])
  show *?g7 ?g8*
    using *assms*
    by(*auto simp*: *hp*(*1*) *hq*(*1*) *pqs.qbs-integrable-def pqs.qbs-pair-measure pqs.qbs-integral-def pqs.pq1.qbs-integral-def pqs.pq2.qbs-integral-def integral-distr integrable-distr-eq*)
      (*auto simp*: *map-prod-def split-beta' intro*!: *pqs.pq2.integral-fst'''*[*of* μ $\lambda x.$ *f* (α (*fst x*),β (*snd x*)),*simplified*] *integral-snd-s-finite*[*OF pqs.pq1.s-finite-measure-axioms pqs.pq2.s-finite-measure-axioms*,*of* λx *y. f* (α *x*, β *y*),*simplified split-beta'*])
qed

end

lemma
  assumes *h*:*p* ∈ *qbs-space* (*monadM-qbs X*) *q* ∈ *qbs-space* (*monadM-qbs Y*)
qbs-integrable (*p* ⨂$_{Qmes}$ *q*) (*case-prod f*)
 shows *qbs-integrable-pair1D1*: *qbs-integrable p* (λ*x*. ∫$_Q$ *y*. *f x y* ∂*q*)
  and *qbs-integrable-pair1D1-norm*: *qbs-integrable p* (λ*x*. ∫$_Q$ *y*. *norm* (*f x y*) ∂*q*)
  and *qbs-integrable-pair1D2*: *AE*$_Q$ *x in p*. *qbs-integrable q* (λ*y*. *f x y*)
  and *qbs-integrable-pair2D1*: *qbs-integrable q* (λ*y*. ∫$_Q$ *x*. *f x y* ∂*p*)
  and *qbs-integrable-pair2D1-norm*: *qbs-integrable q* (λ*y*. ∫$_Q$ *x*. *norm* (*f x y*) ∂*p*)
  and *qbs-integrable-pair2D2*: *AE*$_Q$ *y in q*. *qbs-integrable p* (λ*x*. *f x y*)
  and *qbs-integral-Fubini-fst*: (∫$_Q$ *x*. ∫$_Q$ *y*. *f x y* ∂*q* ∂*p*) = (∫$_Q$ (*x*,*y*). *f x y* ∂(*p*
⨂$_{Qmes}$ *q*)) (**is** *?g7*)
  and *qbs-integral-Fubini-snd*: (∫$_Q$ *y*. ∫$_Q$ *x*. *f x y* ∂*p* ∂*q*) = (∫$_Q$ (*x*,*y*). *f x y* ∂(*p*
⨂$_{Qmes}$ *q*)) (**is** *?g8*)
 using *qbs-integrable-pair1D1'*[*OF h*] *qbs-integrable-pair1D1-norm'*[*OF h*] *qbs-integrable-pair1D2'*[*OF
h*] *qbs-integral-Fubini-fst'*[*OF h*]
   qbs-integrable-pair2D1'[*OF h*] *qbs-integrable-pair2D1-norm'*[*OF h*] *qbs-integrable-pair2D2'*[*OF
h*] *qbs-integral-Fubini-snd'*[*OF h*]
 by *simp-all*

lemma *qbs-integrable-pair2'*:
 fixes *f* :: - ⇒ -::{*second-countable-topology*,*banach*}
 assumes *p* ∈ *qbs-space* (*monadM-qbs X*)
      q ∈ *qbs-space* (*monadM-qbs Y*)
      f ∈ *X* ⨂$_Q$ *Y* →$_Q$ *qbs-borel*
      qbs-integrable q (λ*y*. ∫$_Q$ *x*. *norm* (*f* (*x*,*y*)) ∂*p*)
   and *AE*$_Q$ *y in q*. *qbs-integrable p* (λ*x*. *f* (*x*,*y*))
  shows *qbs-integrable* (*p* ⨂$_{Qmes}$ *q*) *f*
 using *qbs-integrable-pair-swap*[*OF assms*(*2*,*1*) *qbs-integrable-pair1'*[*OF assms*(*2*,*1*)
qbs-morphism-pair-swap[*OF assms*(*3*)]]] *assms*(*4*,*5*)
 by *simp*

lemma *qbs-integrable-indep-mult*:
 fixes *f* :: - ⇒ -::{*real-normed-div-algebra*,*second-countable-topology*, *banach*}
 assumes [*qbs*]:*p* ∈ *qbs-space* (*monadM-qbs X*) *q* ∈ *qbs-space* (*monadM-qbs Y*)
  and *qbs-integrable p f qbs-integrable q g*
 shows *qbs-integrable* (*p* ⨂$_{Qmes}$ *q*) (λ*x*. *f* (*fst x*) ∗ *g* (*snd x*))
proof −
 obtain α μ β ν
  where *hp*: *p* = ⟦*X*, α, μ⟧$_{meas}$ *qbs-s-finite X* α μ
   and *hq*: *q* = ⟦*Y*, β, ν⟧$_{meas}$ *qbs-s-finite Y* β ν
  using *assms rep-qbs-space-monadM* **by** *meson*
 then interpret *pqs*: *pair-qbs-s-finites X* α μ *Y* β ν
  by(*simp add*: *pair-qbs-s-finites-def*)
 have [*qbs*]:*f* ∈ *X* →$_Q$ *qbs-borel g* ∈ *Y* →$_Q$ *qbs-borel*
  by(*auto intro*!: *qbs-integrable-morphism-dest assms simp*: *hp hq*)
 show *?thesis*
  by(*auto intro*!: *qbs-integrable-pair1'*[*of - X - Y*] *qbs-integrable-mult-left qbs-integrable-norm
assms*(*3*) *AEq-I2*[*of - X*]
      simp: *norm-mult qbs-integrable-mult-right*[*OF assms*(*4*)])

qed

lemma *qbs-integrable-indep1*:
fixes f :: - ⇒ -::{*real-normed-div-algebra*,*second-countable-topology*, *banach*}
assumes $p \in$ *qbs-space* (*monadM-qbs* X) $q \in$ *qbs-space* (*monadP-qbs* Y) *qbs-integrable* p f
shows *qbs-integrable* ($p \bigotimes_{Qmes} q$) ($\lambda x.\ f$ (*fst* x))
using *qbs-integrable-indep-mult*[*OF assms*(*1*) *qbs-space-monadPM*[*OF assms*(*2*)] *assms*(*3*) *qbs-integrable-const*[*OF assms*(*2*),*of 1*]]
by *simp*

lemma *qbs-integral-indep1*:
fixes f :: - ⇒ -::{*real-normed-div-algebra*,*second-countable-topology*,*banach*}
assumes $p \in$ *qbs-space* (*monadM-qbs* X) $q \in$ *qbs-space* (*monadP-qbs* Y) *qbs-integrable* p f
shows ($\int_Q$ z. f (*fst* z) ∂($p \bigotimes_{Qmes} q$)) = ($\int_Q$ x. $f\ x\ \partial p$)
using *qbs-integral-Fubini-snd'*[*OF assms*(*1*) *qbs-space-monadPM*[*OF assms*(*2*)] *qbs-integrable-indep1*[*OF assms*]]
by(*simp add*: *qbs-integral-const-prob*[*OF assms*(*2*)])

lemma *qbs-integrable-indep2*:
fixes g :: - ⇒ -::{*real-normed-div-algebra*,*second-countable-topology*, *banach*}
assumes $p \in$ *qbs-space* (*monadP-qbs* X) $q \in$ *qbs-space* (*monadM-qbs* Y) *qbs-integrable* q g
shows *qbs-integrable* ($p \bigotimes_{Qmes} q$) ($\lambda x.\ g$ (*snd* x))
using *qbs-integrable-pair-swap*[*OF assms*(*2*) *qbs-space-monadPM*[*OF assms*(*1*)] *qbs-integrable-indep1*[*OF assms*(*2*,*1*,*3*)]]
by(*simp add*: *split-beta'*)

lemma *qbs-integral-indep2*:
fixes g :: - ⇒ -::{*real-normed-div-algebra*,*second-countable-topology*}
assumes $p \in$ *qbs-space* (*monadP-qbs* X) $q \in$ *qbs-space* (*monadM-qbs* Y) *qbs-integrable* q g
shows ($\int_Q$ z. g (*snd* z) ∂($p \bigotimes_{Qmes} q$)) = ($\int_Q$ y. $g\ y\ \partial q$)
using *qbs-integral-Fubini-fst'*[*OF qbs-space-monadPM*[*OF assms*(*1*)] *assms*(*2*) *qbs-integrable-indep2*[*OF assms*]]
by(*simp add*: *qbs-integral-const-prob*[*OF assms*(*1*)])

lemma *qbs-integral-indep-mult1*:
fixes f **and** g:: - ⇒ -::{*real-normed-field*,*second-countable-topology*, *banach*}
assumes $p \in$ *qbs-space* (*monadP-qbs* X) $q \in$ *qbs-space* (*monadP-qbs* Y)
and *qbs-integrable p f qbs-integrable q g*
shows ($\int_Q$ z. f (*fst* z) $*$ g (*snd* z) ∂($p \bigotimes_{Qmes} q$)) = ($\int_Q$ x. $f\ x\ \partial p$) $*$ ($\int_Q$ y. $g\ y\ \partial q$)
using *qbs-integrable-indep-mult*[*OF qbs-space-monadPM*[*OF assms*(*1*)] *qbs-space-monadPM*[*OF assms*(*2*)] *qbs-integrable-indep-mult*[*OF qbs-space-monadPM*[*OF assms*(*1*)] *qbs-space-monadPM*[*OF assms*(*2*)] *assms*(*3*,*4*)]]
by *simp*

lemma *qbs-integral-indep-mult2*:
fixes *f* **and** *g*:: - ⇒ -::{*real-normed-field*,*second-countable-topology*}
assumes *p* ∈ *qbs-space* (*monadP-qbs X*) *q* ∈ *qbs-space* (*monadP-qbs Y*)
and *qbs-integrable p f qbs-integrable q g*
shows (∫ *Q z*. *g* (*snd z*) ∗ *f* (*fst z*) ∂(*p* ⨂ *Qmes q*)) = (∫ *Q y*. *g y* ∂*q*) ∗ (∫ *Q x*. *f x* ∂*p*)
using *qbs-integral-indep-mult1*[*OF assms*] **by**(*simp add*: *mult.commute*)

4.1.14 The Inverse Function of *l*

definition *qbs-l-inverse* :: *'a measure* ⇒ *'a qbs-measure* **where**
qbs-l-inverse M ≡ ⟦*measure-to-qbs M*, *from-real-into M*, *distr M borel* (*to-real-on M*)⟧$_{meas}$

context *standard-borel-ne*
begin

lemma *qbs-l-inverse-def2*:
assumes [*measurable-cong*]: *sets μ* = *sets M*
shows *qbs-l-inverse μ* = ⟦*measure-to-qbs M*, *from-real*, *distr μ borel to-real*⟧$_{meas}$
proof −
interpret *s*: *standard-borel-ne μ*
using *assms standard-borel-ne-axioms standard-borel-ne-sets* **by** *blast*
have [*measurable*]: *s.from-real* ∈ *borel* →$_M$ *M*
using *assms*(*1*) *measurable-cong-sets s.from-real-measurable* **by** *blast*
interpret *p*: *pair-qbs-meas measure-to-qbs M s.from-real distr μ borel s.to-real from-real distr μ borel to-real*
by(*auto simp*: *pair-qbs-meas-def qbs-meas-def in-Mx-def qbs-Mx-R qbs-meas-axioms-def*)
have *distr μ* (*qbs-to-measure* (*measure-to-qbs M*)) (*s.from-real* ∘ *s.to-real*) = *distr μ* (*qbs-to-measure* (*measure-to-qbs M*)) (*from-real* ∘ *to-real*)
by(*auto intro*!: *distr-cong simp*: *sets-eq-imp-space-eq*[*OF assms*(*1*)])
thus *?thesis*
by(*auto simp*: *distr-distr qbs-l-inverse-def intro*!: *p.qbs-meas-eqI cong*: *measure-to-qbs-cong-sets*[*OF assms*(*1*)])
qed

lemma
assumes [*measurable-cong*]:*sets μ* = *sets M*
shows *qbs-l-inverse-meas*: *qbs-meas* (*measure-to-qbs M*) *from-real* (*distr μ borel to-real*)
and *qbs-l-inverse-s-finite*: *s-finite-measure μ* ⟹ *qbs-s-finite* (*measure-to-qbs M*) *from-real* (*distr μ borel to-real*)
and *qbs-l-inverse-qbs-prob*: *prob-space μ* ⟹ *qbs-prob* (*measure-to-qbs M*) *from-real* (*distr μ borel to-real*)
by(*auto simp*: *qbs-s-finite-def qbs-prob-def in-Mx-def qbs-meas-axioms-def qbs-meas-def real-distribution-def real-distribution-axioms-def qbs-Mx-R*
intro!: *s-finite-measure.s-finite-measure-distr prob-space.prob-space-distr*)

corollary
assumes *sets* μ = *sets M*
shows *qbs-l-inverse-in-space-all-meas*:*qbs-l-inverse* $\mu \in$ *qbs-space* (*all-meas-qbs M*)
and *qbs-l-inverse-in-space-monadM*: *s-finite-measure* $\mu \Longrightarrow$ *qbs-l-inverse* $\mu \in$ *qbs-space* (*monadM-qbs M*)
and *qbs-l-inverse-in-space-monadP*: *prob-space* $\mu \Longrightarrow$ *qbs-l-inverse* $\mu \in$ *qbs-space* (*monadP-qbs M*)
by(*auto simp*: *qbs-l-inverse-def2*[*OF assms*(*1*)] *assms*
intro!: *qbs-meas.in-space-all-meas*[*OF qbs-l-inverse-meas*] *qbs-s-finite.in-space-monadM*[*OF qbs-l-inverse-s-finite*] *qbs-prob.in-space-monadP*[*OF qbs-l-inverse-qbs-prob*])

lemma *qbs-l-qbs-l-inverse*:
assumes [*measurable-cong*]: *sets* μ = *sets M*
shows *qbs-l* (*qbs-l-inverse* μ) = μ
proof −
interpret *qbs-meas measure-to-qbs M from-real distr* μ *borel to-real*
by(*auto intro*!: *qbs-l-inverse-meas assms*)
show *?thesis*
using *distr-id'*[*OF assms*(*1*),*simplified sets-eq-imp-space-eq*[*OF assms*(*1*)]]
by(*auto simp*: *qbs-l qbs-l-inverse-def2*[*OF assms*] *distr-distr cong*: *distr-cong*)
qed

lemma *qbs-l-inverse-qbs-l-all-meas*:
assumes *s* ∈ *qbs-space* (*all-meas-qbs* (*measure-to-qbs M*))
shows *qbs-l-inverse* (*qbs-l s*) = *s*
proof −
from *rep-qbs-space-all-meas*[*OF assms*] **obtain** α μ **where** *h*:
s = ⟦*measure-to-qbs M*, α, μ⟧$_{meas}$ *qbs-meas* (*measure-to-qbs M*) α μ
by *metis*
then interpret *qm*:*qbs-meas measure-to-qbs M* α μ **by** *simp*
interpret *s*: *standard-borel-ne distr* μ *M* α
by(*rule standard-borel-ne-sets*[*of M*]) (*auto simp*: *standard-borel-ne-axioms*)
have *s.from-real* ∘ (*s.to-real* ∘ α) = α
by *standard* (*metis o-apply qbs-Mx-to-X qbs-space-R qm.in-Mx s.from-real-to-real space-distr*)
hence [*simp*]:*distr* μ (*qbs-to-measure* (*measure-to-qbs M*)) (*s.from-real* ∘ (*s.to-real* ∘ α)) = *distr* μ *M* α
distr μ (*qbs-to-measure* (*measure-to-qbs M*)) α = *distr* μ *M* α
by(*simp-all cong*: *distr-cong*)
have [*measurable*]: *s.from-real* ∈ *borel* →$_M$ *M* α ∈ μ →$_M$ *M*
using *qm.*α*-measurable*[*simplified measurable-cong-sets*[*OF refl lr-sets-ident*]]
by(*auto simp*: *s.from-real-measurable*[*simplified measurable-cong-sets*[*OF refl sets-distr*]])
interpret *pqs*:*pair-qbs-meas measure-to-qbs M s.from-real distr* μ *borel* (*s.to-real* ∘ α) α μ
using *h* **by**(*auto simp*: *pair-qbs-meas-def qbs-meas-def in-Mx-def qbs-Mx-R qbs-meas-axioms-def*)
show *?thesis*

by(*auto simp add*: *qm.qbs-l h qbs-l-inverse-def distr-distr cong*: *measure-to-qbs-cong-sets intro*!: *pqs.qbs-meas-eqI*)
qed

lemmas *qbs-l-inverse-qbs-l* = *qbs-l-inverse-qbs-l-all-meas*[*OF monadM-all-meas-space*]

lemmas *qbs-l-inverse-qbs-l-monadP* = *qbs-l-inverse-qbs-l*[*OF qbs-space-monadPM*]

lemma *qbs-l-inverse-morphism-kernel*:
assumes *measure-kernel N M k*
shows (λ*x*. *qbs-l-inverse* (*k x*)) ∈ *measure-to-qbs N* →$_Q$ *all-meas-qbs* (*measure-to-qbs M*)
proof −
interpret *ms*:*measure-kernel N M k* **by** *fact*
have ⟦*measure-to-qbs M*, *from-real*, *distr* (*k x*) *borel to-real*⟧$_{meas}$ = *qbs-l-inverse* (*k x*) **if** *x*:*x* ∈ *space N* **for** *x*
proof −
note *ms.kernel-sets*[*OF x*,*simp*, *measurable-cong*]
then interpret *skx*: *standard-borel-ne k x*
using *standard-borel-ne-axioms standard-borel-ne-sets* **by** *blast*
interpret *pqs*: *pair-qbs-meas measure-to-qbs M from-real distr* (*k x*) *borel to-real skx.from-real distr* (*k x*) *borel skx.to-real*
using *skx.from-real-measurable*[*simplified measurable-cong-sets*[*OF refl ms.kernel-sets*[*OF x*]]]
by(*auto simp*: *pair-qbs-meas-def qbs-meas-def qbs-meas-axioms-def in-Mx-def qbs-Mx-R*)
have *distr* (*k x*) (*qbs-to-measure* (*measure-to-qbs M*)) (*from-real* ∘ *to-real*)
= *distr* (*k x*) (*qbs-to-measure* (*measure-to-qbs M*)) (*skx.from-real* ∘ *skx.to-real*)
by(*auto intro*!: *distr-cong simp*: *ms.kernel-space*[*OF x*])
thus *?thesis*
by(*auto simp*: *qbs-l-inverse-def distr-distr cong*: *measure-to-qbs-cong-sets intro*!: *pqs.qbs-meas-eqI*)
qed
moreover have (λ*x*. ⟦*measure-to-qbs M*, *from-real*, *distr* (*k x*) *borel to-real*⟧$_{meas}$) ∈ *measure-to-qbs N* →$_Q$ *all-meas-qbs* (*measure-to-qbs M*)
proof(*rule qbs-morphismI*)
fix α :: *real* ⇒ -
assume α ∈ *qbs-Mx* (*measure-to-qbs N*)
then have [*measurable*]: α ∈ *borel* →$_M$ *N*
by(*simp add*: *qbs-Mx-R*)
show (λ*x*. ⟦*measure-to-qbs M*, *from-real*, *distr* (*k x*) *borel to-real*⟧$_{meas}$) ∘ α ∈ *qbs-Mx* (*all-meas-qbs* (*measure-to-qbs M*))
by(*auto simp*: *all-meas-qbs-Mx qbs-Mx-R intro*!: *exI*[**where** *x*=*from-real*] *exI*[**where** *x*=λ*x*. *distr* (*k* (α *x*)) *borel to-real*] *measure-kernel.distr-measure-kernel*[*OF ms.measure-kernel-comp*])
qed
ultimately show *?thesis*
by(*rule qbs-morphism-cong'*[*of measure-to-qbs N*,*simplified qbs-space-R*])

qed

lemma *qbs-l-inverse-morphism-s-finite*:
assumes *s-finite-kernel N M k*
shows $(\lambda x.\ qbs\text{-}l\text{-}inverse\ (k\ x)) \in measure\text{-}to\text{-}qbs\ N \to_Q monadM\text{-}qbs\ (measure\text{-}to\text{-}qbs\ M)$
proof −
interpret *sfin*: *s-finite-kernel N M k* **by** *fact*
have $[\![measure\text{-}to\text{-}qbs\ M, from\text{-}real, distr\ (k\ x)\ borel\ to\text{-}real]\!]_{meas} = qbs\text{-}l\text{-}inverse\ (k\ x)$ **if** *x*:$x \in space\ N$ **for** *x*
proof −
note *sfin.kernel-sets*[*OF x*,*simp*, *measurable-cong*]
then interpret *skx*: *standard-borel-ne k x*
using *standard-borel-ne-axioms standard-borel-ne-sets* **by** *blast*
interpret *pqs*: *pair-qbs-s-finite measure-to-qbs M from-real distr* (*k x*) *borel to-real skx.from-real distr* (*k x*) *borel skx.to-real*
using *skx.from-real-measurable*[*simplified measurable-cong-sets*[*OF refl sfin.kernel-sets*[*OF x*]]]
by(*auto simp*: *pair-qbs-s-finite-def qbs-s-finite-def in-Mx-def qbs-Mx-R qbs-meas-def qbs-meas-axioms-def sfin.image-s-finite-measure*[*OF x*] *intro*!: *s-finite-measure.s-finite-measure-distr*)
have *distr* (*k x*) (*qbs-to-measure* (*measure-to-qbs M*)) (*from-real* ∘ *to-real*)
= *distr* (*k x*) (*qbs-to-measure* (*measure-to-qbs M*)) (*skx.from-real* ∘ *skx.to-real*)
by(*auto intro*!: *distr-cong simp*: *sfin.kernel-space*[*OF x*])
thus *?thesis*
by(*auto simp*: *qbs-l-inverse-def distr-distr cong*: *measure-to-qbs-cong-sets intro*!: *pqs.qbs-meas-eqI*)
qed
moreover have $(\lambda x.\ [\![measure\text{-}to\text{-}qbs\ M, from\text{-}real, distr\ (k\ x)\ borel\ to\text{-}real]\!]_{meas}) \in measure\text{-}to\text{-}qbs\ N \to_Q monadM\text{-}qbs\ (measure\text{-}to\text{-}qbs\ M)$
proof(*rule qbs-morphismI*)
fix α :: *real* ⇒ -
assume $\alpha \in qbs\text{-}Mx\ (measure\text{-}to\text{-}qbs\ N)$
then have [*measurable*]: $\alpha \in borel \to_M N$
by(*simp add*: *qbs-Mx-R*)
show $(\lambda x.\ [\![measure\text{-}to\text{-}qbs\ M, from\text{-}real, distr\ (k\ x)\ borel\ to\text{-}real]\!]_{meas}) \circ \alpha \in qbs\text{-}Mx\ (monadM\text{-}qbs\ (measure\text{-}to\text{-}qbs\ M))$
by(*auto simp*: *monadM-qbs-Mx qbs-Mx-R intro*!: *exI*[**where** *x*=*from-real*] *exI*[**where** $x{=}\lambda x.\ distr\ (k\ (\alpha\ x))\ borel\ to\text{-}real$] *s-finite-kernel.comp-measurable*[*OF sfin.distr-s-finite-kernel*])
qed
ultimately show *?thesis*
by(*rule qbs-morphism-cong'*[*of measure-to-qbs N*,*simplified qbs-space-R*])
qed

lemma *qbs-l-inverse-qbs-morphism-prob*:
assumes [*measurable*]:$k \in N \to_M prob\text{-}algebra\ M$
shows $(\lambda x.\ qbs\text{-}l\text{-}inverse\ (k\ x)) \in measure\text{-}to\text{-}qbs\ N \to_Q monadP\text{-}qbs\ (measure\text{-}to\text{-}qbs\ M)$

proof(*safe intro*!: *qbs-morphism-monadPI'* *qbs-l-inverse-morphism-s-finite prob-kernel.s-finite-kernel-prob-ker*
fix *x*
assume *x* ∈ *qbs-space* (*measure-to-qbs N*)
then have *x* ∈ *space N* **by**(*simp add*: *qbs-space-R*)
from *measurable-space*[*OF assms this*]
have [*measurable-cong*, *simp*]: *sets* (*k x*) = *sets M* **and** *p*:*prob-space* (*k x*)
by(*auto simp*: *space-prob-algebra*)
then interpret *s*: *standard-borel-ne k x*
using *standard-borel-ne-axioms standard-borel-ne-sets* **by** *blast*
show *qbs-l-inverse* (*k x*) ∈ *qbs-space* (*monadP-qbs* (*measure-to-qbs M*))
using *s.qbs-l-inverse-in-space-monadP*[*OF refl p*] **by**(*simp cong*: *measure-to-qbs-cong-sets*)
qed(*simp add*: *prob-kernel-def'*)

lemma *qbs-l-inverse-return*:
assumes *x* ∈ *space M*
shows *qbs-l-inverse* (*return M x*) = *return-qbs* (*measure-to-qbs M*) *x*
proof −
interpret *s*: *standard-borel-ne return M x*
by(*rule standard-borel-ne-sets*[*of M*]) (*auto simp*: *standard-borel-ne-axioms*)
show *?thesis*
using *s.qbs-l-inverse-in-space-monadP*[*OF refl prob-space-return*[*OF assms*]]
by(*auto intro*!: *inj-onD*[*OF qbs-l-inj-P*[*of measure-to-qbs M*]] *return-cong qbs-l-inverse-in-space-monadP qbs-morphism-space*[*OF return-qbs-morphismP*[*of measure-to-qbs M*]] *assms*
simp: *s.qbs-l-qbs-l-inverse*[*OF refl*] *qbs-l-return-qbs*[*of - M*,*simplified qbs-space-R*,*OF assms*] *qbs-space-R cong*: *measure-to-qbs-cong-sets*)
qed

lemma *qbs-l-inverse-bind-kernel*:
assumes *standard-borel-ne N measure-kernel M N k*
shows *qbs-l-inverse* (*M* ⤜$_k$ *k*) = *qbs-l-inverse M* ⤜ (λ*x*. *qbs-l-inverse* (*k x*))
(**is** *?lhs* = *?rhs*)
proof −
interpret *ms*: *measure-kernel M N k* **by** *fact*
interpret *s*: *standard-borel-ne N* **by** *fact*
have *sets*[*simp*,*measurable-cong*]:*sets* (*M* ⤜$_k$ *k*) = *sets N*
by(*auto intro*!: *sets-bind-kernel*[*OF - space-ne*] *simp*: *ms.kernel-sets*)
then interpret *s2*: *standard-borel-ne M* ⤜$_k$ *k*
using *s.standard-borel-ne-axioms standard-borel-ne-sets* **by** *blast*
have *eq1*:*distr* (*M* ⤜$_k$ *k*) (*qbs-to-measure* (*measure-to-qbs N*)) (*s2.from-real* ∘ *s2.to-real*)
= *distr* (*M* ⤜$_k$ *k*) (*qbs-to-measure* (*measure-to-qbs N*)) (*s.from-real* ∘ *s.to-real*)
by(*auto intro*!: *distr-cong cong*: *sets-eq-imp-space-eq*)
have [*measurable*]: *s2.from-real* ∈ *borel* →$_M$ *N*
using *measurable-cong-sets s2.from-real-measurable sets* **by** *blast*
have *comp1*:(λ*x*. *qbs-l-inverse* (*k x*)) ∘ *from-real* = (λ*r*. ⟦*measure-to-qbs N*, *s.from-real*, *distr* (*k* (*from-real r*)) *borel s.to-real*⟧$_{meas}$)
proof
fix *r*

have *setskfr*[*measurable-cong*, *simp*]: *sets* (*k* (*from-real r*)) = *sets N*
by(*auto intro*!: *ms.kernel-sets measurable-space*[*OF from-real-measurable*])
then interpret *s3*: *standard-borel-ne k* (*from-real r*)
using *s.standard-borel-ne-axioms standard-borel-ne-sets* **by** *blast*
have [*measurable*]: *s3.from-real* ∈ *borel* →$_M$ *N*
using *measurable-cong-sets s3.from-real-measurable setskfr* **by** *blast*
have *distr* (*k* (*from-real r*)) (*qbs-to-measure* (*measure-to-qbs N*)) (*s3.from-real* ∘ *s3.to-real*)
= *distr* (*k* (*from-real r*)) (*qbs-to-measure* (*measure-to-qbs N*)) (*s.from-real* ∘ *s.to-real*)
by(*auto intro*!: *distr-cong simp*: *sets-eq-imp-space-eq*[*OF setskfr*])
thus ((λ*x*. *qbs-l-inverse* (*k x*)) ∘ *from-real*) *r* = ⟦*measure-to-qbs N*, *s.from-real*, *distr* (*k* (*from-real r*)) *borel s.to-real*⟧$_{meas}$
by(*auto simp*: *pair-qbs-meas-def qbs-meas-def qbs-l-inverse-def qbs-meas-eq-def qbs-s-finite-def in-Mx-def qbs-meas-axioms-def qbs-Mx-R distr-distr measurable-space*[*OF from-real-measurable*] *cong*: *measure-to-qbs-cong-sets intro*!: *s-finite-measure.s-finite-measure-distr pair-qbs-meas.qbs-meas-eqI*)
qed
have *?lhs* = ⟦*measure-to-qbs* (*M* ⤜$_k$ *k*), *s2.from-real*, *distr* (*M* ⤜$_k$ *k*) *borel s2.to-real*⟧$_{meas}$
by(*simp add*: *qbs-l-inverse-def*)
also have ... = ⟦*measure-to-qbs N*, *s.from-real*, *distr* (*M* ⤜$_k$ *k*) *borel s.to-real*⟧$_{meas}$
by(*auto cong*: *measure-to-qbs-cong-sets intro*!: *pair-qbs-meas.qbs-meas-eqI simp*: *pair-qbs-meas-def qbs-meas-def qbs-meas-axioms-def in-Mx-def qbs-Mx-R distr-distr eq1*)
also have ... = ⟦*measure-to-qbs N*, *s.from-real*, *M* ⤜$_k$ (λ*x*. *distr* (*k x*) *borel s.to-real*)⟧$_{meas}$
by(*simp add*: *ms.distr-bind-kernel*[*OF space-ne refl*])
also have ... = ⟦*measure-to-qbs N*, *s.from-real*, *distr M borel to-real* ⤜$_k$ (λ*r*. *distr* (*k* (*from-real r*)) *borel s.to-real*)⟧$_{meas}$
proof −
have *M* ⤜$_k$ (λ*x*. *distr* (*k x*) *borel s.to-real*) = *M* ⤜$_k$ (λ*x*. *distr* (*k* (*from-real* (*to-real x*))) *borel s.to-real*)
by(*auto intro*!: *bind-kernel-cong-All*)
also have ... = *distr M borel to-real* ⤜$_k$ (λ*r*. *distr* (*k* (*from-real r*)) *borel s.to-real*)
by(*auto intro*!: *measure-kernel.bind-kernel-distr*[*symmetric*,**where** *Y*=*borel*] *space-ne measure-kernel.distr-measure-kernel*[**where** *Y*=*N*] *ms.measure-kernel-comp*)
finally show *?thesis* **by** *simp*
qed
also have ... = *?rhs*
by(*intro qbs-meas.bind-qbs-all-meas*[*OF qbs-l-inverse-meas*[*OF refl*],*symmetric*]
s.qbs-l-inverse-morphism-kernel[*OF assms*(*2*)] *comp1*
measure-kernel.distr-measure-kernel[*OF ms.measure-kernel-comp*])
(*auto simp*: *qbs-Mx-R qbs-l-inverse-def*)
finally show *?thesis* **.**
qed

lemma *qbs-l-inverse-bind*:

assumes *standard-borel-ne N s-finite-measure M k* ∈ *M* →$_M$ *prob-algebra N*
shows *qbs-l-inverse* (*M* ⤜ *k*) = *qbs-l-inverse M* ⤜ (λ*x*. *qbs-l-inverse* (*k x*))
by(*auto simp*: *bind-kernel-bind*[*OF measurable-prob-algebraD*[*OF assms*(*3*)],*symmetric*] *prob-kernel-def'*
intro!: *qbs-l-inverse-bind-kernel assms prob-kernel.axioms*(*1*))

end

4.1.15 PMF and SPMF

definition *qbs-pmf* ≡ (λ*p*. *qbs-l-inverse* (*measure-pmf p*))
definition *qbs-spmf* ≡ (λ*p*. *qbs-l-inverse* (*measure-spmf p*))

declare [[*coercion qbs-pmf*]]

lemma *qbs-pmf-qbsP*:
fixes *p* :: (- :: *countable*) *pmf*
shows *qbs-pmf p* ∈ *qbs-space* (*monadP-qbs* (*count-space*$_Q$ *UNIV*))
by(*auto simp*: *qbs-pmf-def measure-to-qbs-cong-sets*[*of count-space UNIV measure-pmf p*,*simplified*]
intro!: *standard-borel-ne.qbs-l-inverse-in-space-monadP measure-pmf.prob-space-axioms*)

lemma *qbs-pmf-qbs*[*qbs*]:
fixes *p* :: (- :: *countable*) *pmf*
shows *qbs-pmf p* ∈ *qbs-space* (*monadM-qbs* (*count-space*$_Q$ *UNIV*))
by (*simp add*: *qbs-pmf-qbsP qbs-space-monadPM*)

lemma *qbs-spmf-qbs*[*qbs*]:
fixes *q* :: (- :: *countable*) *spmf*
shows *qbs-spmf q* ∈ *qbs-space* (*monadM-qbs* (*count-space*$_Q$ *UNIV*))
by(*auto simp*: *qbs-spmf-def measure-to-qbs-cong-sets*[*of count-space UNIV measure-spmf q*,*simplified*]
intro!: *standard-borel-ne.qbs-l-inverse-in-space-monadM subprob-space.s-finite-measure-subprob*)

lemma [*simp*]:
fixes *p* :: (- :: *countable*) *pmf* **and** *q* :: (- :: *countable*) *spmf*
shows *qbs-l-qbs-pmf*: *qbs-l* (*qbs-pmf p*) = *measure-pmf p*
and *qbs-l-qbs-spmf*: *qbs-l* (*qbs-spmf q*) = *measure-spmf q*
by(*auto simp*: *qbs-pmf-def qbs-spmf-def intro*!: *standard-borel-ne.qbs-l-qbs-l-inverse subprob-space.s-finite-measure-subprob measure-pmf.subprob-space-axioms*)

lemma *qbs-pmf-return-pmf*:
fixes *x* :: - :: *countable*
shows *qbs-pmf* (*return-pmf x*) = *return-qbs* (*count-space*$_Q$ *UNIV*) *x*
proof −
note *return-qbs-morphismP*[*qbs*]
show *?thesis*
by(*auto intro*!: *inj-onD*[*OF qbs-l-inj-P*[**where** *X*=*count-space*$_Q$ *UNIV*]] *return-cong qbs-pmf-qbsP*

simp: *qbs-l-return-qbs return-pmf.rep-eq*)
qed

lemma *qbs-pmf-bind-pmf*:
fixes p :: ('*a* :: *countable*) *pmf* **and** f :: '*a* ⇒ ('*b* :: *countable*) *pmf*
shows *qbs-pmf* (p ⤜ f) = *qbs-pmf* p ⤜ (λ*x*. *qbs-pmf* (f *x*))
by(*auto simp*: *measure-pmf-bind qbs-pmf-def space-prob-algebra measure-pmf.prob-space-axioms*
intro!: *standard-borel-ne.qbs-l-inverse-bind*[**where** *N*=*count-space UNIV*]
prob-space.s-finite-measure-prob)

lemma *qbs-pair-pmf*:
fixes p :: ('*a* :: *countable*) *pmf* **and** q :: ('*b* :: *countable*) *pmf*
shows *qbs-pmf* p ⨂$_{Qmes}$ *qbs-pmf* q = *qbs-pmf* (*pair-pmf* p q)
proof(*rule inj-onD*[*OF qbs-l-inj-P*[*of count-space*$_Q$ *UNIV* ⨂$_Q$ *count-space*$_Q$ *UNIV*]])
show *qbs-l* (*qbs-pmf* p ⨂$_{Qmes}$ *qbs-pmf* q) = *qbs-l* (*qbs-pmf* (*pair-pmf* p q))
by(*simp add*: *measure-pair-pmf qbs-l-qbs-pair-measure*[*OF standard-borel-ne.standard-borel*
standard-borel-ne.standard-borel,*of count-space UNIV count-space UNIV*])
next
note [*qbs*] = *qbs-pmf-qbsP qbs-pair-measure-morphismP*
show *qbs-pmf* p ⨂$_{Qmes}$ *qbs-pmf* q ∈ *qbs-space* (*monadP-qbs* (*count-space*$_Q$ *UNIV*
⨂$_Q$ *count-space*$_Q$ *UNIV*))
qbs-pmf (*pair-pmf* p q) ∈ *qbs-space* (*monadP-qbs* (*count-space*$_Q$ *UNIV* ⨂$_Q$
count-space$_Q$ *UNIV*))
by *auto* (*simp add*: *qbs-count-space-prod*)
qed

4.1.16 Density

lift-definition *density-qbs* :: ['*a qbs-measure*, '*a* ⇒ *ennreal*] ⇒ '*a qbs-measure*
is λ(*X*,α,μ) *f*. *if f* ∈ *X* →$_Q$ *qbs-borel then* (*X*, α, *density* μ (*f* ∘ α)) *else* (*X*, *SOME*
a. *a* ∈ *qbs-Mx X*, *null-measure borel*)
proof *safe*
fix *X Y* :: '*a quasi-borel*
fix α β μ ν **and** *f* :: - ⇒ *ennreal*
assume *1*:*qbs-meas-eq* (*X*, α, μ) (*Y*, β, ν)
then interpret *qs*: *pair-qbs-meas X* α μ β ν
using *qbs-meas-eq-dest*[*OF 1*] **by**(*simp add*: *pair-qbs-meas-def*)
have [*simp*]:(*SOME a*. *a* ∈ *qbs-Mx X*) ∈ *qbs-Mx X* (*SOME a*. *a* ∈ *qbs-Mx Y*) ∈
qbs-Mx X
using *qs.pq1.in-Mx* **by**(*simp-all only*: *some-in-eq qbs-meas-eq-dest*[*OF 1*]) *blast*+
{
assume *f* ∈ *X* →$_Q$ *qbs-borel*
then have *qbs-meas-eq* (*X*, α, *density* μ (*f* ∘ α)) (*Y*, β, *density* ν (*f* ∘ β))
by(*auto simp*: *qbs-meas-eq-def lr-adjunction-correspondence density-distr*[*symmetric*]
comp-def
qbs-meas-eq-dest[*OF 1*] *qbs-meas-def in-Mx-def qbs-meas-axioms-def*
qs.pq1.mu-sets qs.pq2.mu-sets AE-distr-iff)
}
moreover have *f* ∈ *X* →$_Q$ *qbs-borel* ⟹ *f* ∉ *Y* →$_Q$ *qbs-borel* ⟹ *qbs-meas-eq*

$(X, \alpha, \textit{density}\ \mu\ (f \circ \alpha))\ (Y, (\textit{SOME}\ a.\ a \in \textit{qbs-Mx}\ Y), \textit{null-measure borel})$
$f \notin X \to_Q \textit{qbs-borel} \Longrightarrow f \in Y \to_Q \textit{qbs-borel} \Longrightarrow \textit{qbs-meas-eq}\ (X, (\textit{SOME}\ a.\ a \in \textit{qbs-Mx}\ X), \textit{null-measure borel})\ (Y, \beta, \textit{density}\ \nu\ (f \circ \beta))$
$f \notin X \to_Q \textit{qbs-borel} \Longrightarrow f \notin Y \to_Q \textit{qbs-borel} \Longrightarrow \textit{qbs-meas-eq}\ (X, (\textit{SOME}\ a.\ a \in \textit{qbs-Mx}\ X), \textit{null-measure borel})\ (Y, (\textit{SOME}\ a.\ a \in \textit{qbs-Mx}\ Y), \textit{null-measure borel})$
by(*auto simp*: *qbs-meas-eq-dest*[*OF 1*] *qbs-meas-eq-def qbs-meas-def in-Mx-def qbs-meas-axioms-def distr-return null-measure-distr*)
ultimately show $\textit{qbs-meas-eq}\ (\textit{if}\ f \in X \to_Q \textit{borel}_Q\ \textit{then}\ (X, \alpha, \textit{density}\ \mu\ (f \circ \alpha))\ \textit{else}\ (X, \textit{SOME}\ aa.\ aa \in \textit{qbs-Mx}\ X, \textit{null-measure borel}))$
$(\textit{if}\ f \in Y \to_Q \textit{borel}_Q\ \textit{then}\ (Y, \beta, \textit{density}\ \nu\ (f \circ \beta))\ \textit{else}\ (Y, \textit{SOME}\ a.\ a \in \textit{qbs-Mx}\ Y, \textit{null-measure borel}))$
by *auto*
qed

lemma(**in** *qbs-meas*) *density-qbs*:
shows $f \in X \to_Q \textit{qbs-borel} \Longrightarrow \textit{density-qbs}\ [\![X,\alpha,\ \mu]\!]_{meas}\ f = [\![X,\ \alpha,\ \textit{density}\ \mu\ (f \circ \alpha)]\!]_{meas}$
by (*simp add*: *density-qbs.abs-eq*)

lemma (**in** *qbs-meas*) *density-qbs-meas*: $\textit{qbs-meas}\ X\ \alpha\ (\textit{density}\ \mu\ (f \circ \alpha))$
by (*simp-all add*: *in-Mx-axioms mu-sets qbs-meas.intro qbs-meas-axioms-def*)

lemma(**in** *qbs-s-finite*) *density-qbs-s-finite*:
$f \in X \to_Q \textit{qbs-borel} \Longrightarrow \textit{qbs-s-finite}\ X\ \alpha\ (\textit{density}\ \mu\ (f \circ \alpha))$
by(*auto simp add*: *qbs-s-finite-def qbs-meas-def in-Mx-def qbs-meas-axioms-def mu-sets intro*!: *s-finite-measure-density*)

lemma *density-qbs-density-qbs-eq-all-meas*:
assumes [*qbs*]:$s \in \textit{qbs-space}\ (\textit{all-meas-qbs}\ X)\ f \in X \to_Q \textit{qbs-borel}\ g \in X \to_Q \textit{qbs-borel}$
shows $\textit{density-qbs}\ (\textit{density-qbs}\ s\ f)\ g = \textit{density-qbs}\ s\ (\lambda x.\ f\ x * g\ x)$
proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)] **obtain** $\alpha\ \mu$
where *hs*: $s = [\![X,\ \alpha,\ \mu]\!]_{meas}\ \textit{qbs-meas}\ X\ \alpha\ \mu$ **by** *metis*
then interpret $\textit{qbs-meas}\ X\ \alpha\ \mu$ **by** *simp*
have $(\lambda x.\ f\ (\alpha\ x) * g\ (\alpha\ x)) = (\lambda x.\ f\ x * g\ x) \circ \alpha$ **by** *auto*
with *assms*(*2*,*3*) **show** *?thesis*
by(*simp add*: *hs*(*1*) *density-qbs*[*OF assms*(*2*)] *qbs-meas.density-qbs*[*OF density-qbs-meas assms*(*3*)]
density-qbs lr-adjunction-correspondence density-density-eq)
qed

lemmas *density-qbs-density-qbs-eq* = *density-qbs-density-qbs-eq-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-l-density-qbs-all-meas*:
assumes [*qbs*,*measurable*]:$s \in \textit{qbs-space}\ (\textit{all-meas-qbs}\ X)\ f \in X \to_Q \textit{qbs-borel}$
shows $\textit{qbs-l}\ (\textit{density-qbs}\ s\ f) = \textit{density}\ (\textit{qbs-l}\ s)\ f$

proof −
from *rep-qbs-space-all-meas*[*OF assms*(*1*)]
obtain α μ **where** *s*: $s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-meas X* α μ
by *metis*
then interpret *qbs-meas X* α μ **by** *simp*
have $(\lambda x.\ f\ (\alpha\ x)) = f \circ \alpha$ **by** *auto*
thus *?thesis*
by(*simp add*: *s*(*1*) *qbs-l density-qbs qbs-meas.qbs-l*[*OF density-qbs-meas*] *density-distr*)
qed

lemmas *qbs-l-density-qbs* = *qbs-l-density-qbs-all-meas*[*OF monadM-all-meas-space*]

corollary *qbs-l-density-qbs-indicator-all-meas*:
assumes [*qbs*,*measurable*]:*s* ∈ *qbs-space* (*all-meas-qbs X*) *qbs-pred X P*
shows *qbs-l* (*density-qbs s* (*indicator* {*x*∈*qbs-space X*. *P x*})) (*qbs-space X*) = *qbs-l s* {*x*∈*qbs-space X*. *P x*}
proof −
have *1*[*measurable*]: {*x* ∈ *qbs-space X*. *P x*} ∈ *sets* (*qbs-to-measure X*)
by (*metis* (*no-types*, *lifting*) *Collect-cong assms*(*2*) *qbs-pred-iff-sets space-L*)
have *2*[*qbs*]: *indicator* {*x* ∈ *qbs-space X*. *P x*} ∈ *X* →$_Q$ *qbs-borel*
by(*rule indicator-qbs-morphism''*) *qbs*
show *?thesis*
using *assms*(*2*)
by(*auto simp*: *qbs-l-density-qbs-all-meas*[*of* - *X*]
emeasure-density[*of indicator* {*x*∈*space* (*qbs-to-measure X*). *P x*} *qbs-l s*,*OF* - *sets.top*,*simplified measurable-qbs-l-all-meas'*[*OF assms*(*1*)],*OF borel-measurable-indicator*[*OF predE*],*simplified space-L space-qbs-l-in-all-meas*[*OF assms*(*1*)]]
qbs-pred-iff-measurable-pred nn-set-integral-space[*of qbs-l s*,*simplified space-qbs-l-in-all-meas*[*OF assms*(*1*)]]
nn-integral-indicator[*of* - *qbs-l s*,*simplified sets-qbs-l-all-meaures*[*OF assms*(*1*)]])
qed

lemmas *qbs-l-density-qbs-indicator* = *qbs-l-density-qbs-indicator-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-nn-integral-density-qbs-all-meas*:
assumes [*qbs*,*measurable*]:*s* ∈ *qbs-space* (*all-meas-qbs X*) *f* ∈ *X* →$_Q$ *qbs-borel g* ∈ *X* →$_Q$ *qbs-borel*
shows $(\int^+_Q x.\ g\ x\ \partial(\textit{density-qbs}\ s\ f)) = (\int^+_Q x.\ f\ x * g\ x\ \partial s)$
by(*auto simp*: *qbs-nn-integral-def2-l qbs-l-density-qbs-all-meas*[*of* - *X*] *measurable-qbs-l-all-meas'*[*OF assms*(*1*)] *intro*!: *nn-integral-density*)

lemmas *qbs-nn-integral-density-qbs* = *qbs-nn-integral-density-qbs-all-meas*[*OF monadM-all-meas-space*]

lemma *qbs-integral-density-qbs-all-meas*:

fixes $g :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*} **and** $f :: 'a \Rightarrow$ *real*
assumes [*qbs*,*measurable*]:$s \in$ *qbs-space* (*all-meas-qbs* X) $f \in X \to_Q$ *qbs-borel* $g \in X \to_Q$ *qbs-borel*
and $AE_Q\ x$ *in* $s.\ f\ x \geq 0$
shows $(\int_Q\ x.\ g\ x\ \partial(\textit{density-qbs}\ s\ f)) = (\int_Q\ x.\ f\ x *_R\ g\ x\ \partial s)$
using *assms*(*4*)
by(*auto simp*: *qbs-integral-def2-l qbs-l-density-qbs-all-meas*[*of* - X] *measurable-qbs-l-all-meas'*[*OF assms*(*1*)]
AEq-qbs-l intro!: *integral-density*)

lemmas *qbs-integral-density-qbs* = *qbs-integral-density-qbs-all-meas*[*OF monadM-all-meas-space*]

lemma *density-qbs-morphism*[*qbs*]: *density-qbs* $\in$ *monadM-qbs* $X \to_Q$ ($X \Rightarrow_Q$ *qbs-borel*) $\Rightarrow_Q$ *monadM-qbs* X
proof(*rule curry-preserves-morphisms*[*OF pair-qbs-morphismI*])
fix γ **and** β :: - $\Rightarrow$ - $\Rightarrow$ *ennreal*
assume h:$\gamma \in$ *qbs-Mx* (*monadM-qbs* X) $\beta \in$ *qbs-Mx* ($X \Rightarrow_Q$ *qbs-borel*)
hence [*qbs*]: $\gamma \in$ *qbs-borel* $\to_Q$ *monadM-qbs* X $\beta \in$ *qbs-borel* $\to_Q X \Rightarrow_Q$ *qbs-borel*
by(*simp-all add*: *qbs-Mx-is-morphisms*)
from *rep-qbs-Mx-monadM*[*OF h*(*1*)] **obtain** α k **where** *hk*:
$\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})\ \alpha \in$ *qbs-Mx* X *s-finite-kernel borel borel* k $\bigwedge r.$ *qbs-s-finite* $X\ \alpha\ (k\ r)$
by *metis*
interpret *qs*: *qbs-s-finite* $X\ \alpha\ k\ r$ **for** r **by** *fact*
have [*measurable*]: $(\lambda(x,\ y).\ \beta\ x\ (\alpha\ y)) \in$ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
proof −
have $(\lambda(x,\ y).\ \beta\ x\ (\alpha\ y)) \in$ *qbs-borel* $\bigotimes_Q$ *qbs-borel* $\to_Q$ *qbs-borel*
by *simp*
thus *?thesis*
by(*simp add*: *lr-adjunction-correspondence qbs-borel-prod borel-prod*)
qed
have [*simp*]:*density-qbs* $(\gamma\ r)\ (\beta\ r) = [\![X,\ \alpha,\ \textit{density}\ (k\ r)\ (\beta\ r \circ \alpha)]\!]_{meas}$ **for** r
using *hk*(*4*) **by**(*auto simp add*: *hk*(*1*) *density-qbs.abs-eq*)
show $(\lambda r.\ \textit{density-qbs}\ (\textit{fst}\ (\gamma\ r, \beta\ r))\ (\textit{snd}\ (\gamma\ r, \beta\ r))) \in$ *qbs-Mx* (*monadM-qbs* X)
by(*auto simp*: *monadM-qbs-Mx comp-def intro*!: *exI*[**where** x=α] *exI*[**where** x=$\lambda r.$ *density* $(k\ r)\ (\beta\ r \circ \alpha)$] *s-finite-kernel.density-s-finite-kernel*[*OF hk*(*3*)])
qed

lemma *density-qbs-morphism'*:
assumes [*qbs*,*measurable*]: $f \in X \Rightarrow_Q$ *qbs-borel*
shows $(\lambda p.$ *density-qbs* $p\ f) \in$ *all-meas-qbs* $X \Rightarrow_Q$ *all-meas-qbs* X
proof(*rule qbs-morphismI*)
fix γ
assume $\gamma \in$ *qbs-Mx* (*all-meas-qbs* X)
from *rep-all-meas-qbs-Mx*[*OF this*] **obtain** α k **where** *ak*:
$\gamma = (\lambda r.\ [\![X,\ \alpha,\ k\ r]\!]_{meas})\ \alpha \in$ *qbs-Mx* X *measure-kernel borel borel* k $\bigwedge r.$ *qbs-meas* $X\ \alpha\ (k\ r)$
by *blast*

interpret *qbs-meas X α k r* **for** *r* **by** *fact*
show (λ*p*. *density-qbs p f*) ∘ *γ* ∈ *qbs-Mx* (*all-meas-qbs X*)
by(*auto simp*: *all-meas-qbs-Mx density-qbs comp-def ak*(*1*,*3*)
intro!: *exI*[**where** *x*=*α*] *exI*[**where** *x*=λ*x*. *density* (*k x*) (λ*r*. *f* (*α r*))]
measure-kernel.density-measure-kernel′ measurable-compose[*OF α-measurable*])
qed

lemma *density-qbs-cong-AE-all-meas*:
assumes [*qbs*]: *s* ∈ *qbs-space* (*all-meas-qbs X*) *f* ∈ *X* →*Q qbs-borel g* ∈ *X* →*Q* *qbs-borel*
and *AE*$_Q$ *x in s*. *f x* = *g x*
shows *density-qbs s f* = *density-qbs s g*
proof(*rule inj-onD*[*OF qbs-l-inj-all-meas*[*of X*]])
from *assms*(*4*) **show** *qbs-l* (*density-qbs s f*) = *qbs-l* (*density-qbs s g*)
by(*auto simp*: *qbs-l-density-qbs-all-meas*[*of - X*] *measurable-qbs-l-all-meas′*[*OF assms*(*1*)]
AEq-qbs-l lr-adjunction-correspondence[*symmetric*] *intro*!: *density-cong*)
qed(*auto intro*!: *qbs-morphism-space*[*OF density-qbs-morphism′*[*OF assms*(*2*)]] *qbs-morphism-space*[*OF density-qbs-morphism′*[*OF assms*(*3*)]])

lemmas *density-qbs-cong-AE* = *density-qbs-cong-AE-all-meas*[*OF monadM-all-meas-space*]

corollary *density-qbs-cong-all-meas*:
assumes [*qbs*]: *s* ∈ *qbs-space* (*all-meas-qbs X*) *f* ∈ *X* →*Q qbs-borel*
and ⋀*x*. *x* ∈ *qbs-space X* ⟹ *f x* = *g x*
shows *density-qbs s f* = *density-qbs s g*
by(*auto intro*!: *density-qbs-cong-AE-all-meas*[*of - X*] *AEq-I2′*[*of - X*] *cong*: *assms*(*3*) *qbs-morphism-cong*[**where** *g*=*f*])

lemmas *density-qbs-cong* = *density-qbs-cong-all-meas*[*OF monadM-all-meas-space*]

lemma *density-qbs-1*[*simp*]: *density-qbs s* (λ*x*. *1*) = *s*
proof −
obtain *X* **where** *s*[*qbs*]: *s* ∈ *qbs-space* (*all-meas-qbs X*)
using *in-qbs-space-of-all-meas* **by** *blast*
show *?thesis*
by(*auto intro*!: *inj-onD*[*OF qbs-l-inj-all-meas - - s*] *qbs-morphism-space*[*OF density-qbs-morphism′*]
simp: *qbs-l-density-qbs-all-meas*[*of - X*] *density-1*)
qed

lemma *pair-density-qbs*:
assumes [*qbs*]: *p* ∈ *qbs-space* (*monadM-qbs X*) *q* ∈ *qbs-space* (*monadM-qbs Y*)
and [*qbs*]: *f* ∈ *X* →*Q qbs-borel g* ∈ *Y* →*Q qbs-borel*
shows *density-qbs p f* ⨂$_{Qmes}$ *density-qbs q g* = *density-qbs* (*p* ⨂$_{Qmes}$ *q*) (λ(*x*,*y*). *f x* ∗ *g y*)
proof(*safe intro*!: *qbs-measure-eqI*[*of - X* ⨂$_Q$ *Y*])
fix *h* :: - ⇒ *ennreal*
assume *h*[*qbs*]:*h* ∈ *X* ⨂$_Q$ *Y* →*Q borel*$_Q$

show $(\int^+_Q z.\ h\ z\ \partial(\textit{density-qbs}\ p\ f \bigotimes_{Qmes} \textit{density-qbs}\ q\ g)) = (\int^+_Q z.\ h\ z\ \partial(\textit{density-qbs}\ (p \bigotimes_{Qmes} q)\ (\lambda(x, y).\ f\ x * g\ y)))$ (**is** *?lhs = ?rhs*)
proof −
have *?lhs* $= (\int^+_Q x.\ \int^+_Q y.\ h\ (x, y)\ \partial\textit{density-qbs}\ q\ g\ \partial\textit{density-qbs}\ p\ f)$
by(*simp add*: *qbs-nn-integral-Fubini-fst*[*of - X - Y*])
also have ... $= (\int^+_Q x.\ \int^+_Q y.\ g\ y * h\ (x, y)\ \partial q\ \partial\textit{density-qbs}\ p\ f)$
by(*auto intro*!: *qbs-nn-integral-cong*[*of - X*] *simp*: *qbs-nn-integral-density-qbs*[*of - Y*])
also have ... = *?rhs*
by(*auto simp add*: *qbs-nn-integral-density-qbs*[*of - X*] *qbs-nn-integral-density-qbs*[*of - X* $\bigotimes_Q$ *Y*] *split-beta' qbs-nn-integral-Fubini-fst*[*of - X - Y*,*symmetric*] *qbs-nn-integral-cmult*[*of - Y*] *mult.assoc intro*!: *qbs-nn-integral-cong*[*of - X*])
finally show *?thesis* **.**
qed
qed *simp-all*

4.1.17 Normalization

definition *normalize-qbs* :: *'a qbs-measure* ⇒ *'a qbs-measure* **where**
normalize-qbs s ≡ (*let X* = *qbs-space-of s*;
r = *qbs-l s* (*qbs-space X*) *in*
if r ≠ *0* ∧ *r* ≠ ∞ *then density-qbs s* (λ*x*. *1* / *r*)
else qbs-null-measure X)

lemma
assumes *s* ∈ *qbs-space* (*all-meas-qbs X*)
shows *normalize-qbs-all-meas*: *qbs-l s* (*qbs-space X*) ≠ *0* ⟹ *qbs-l s* (*qbs-space X*) ≠ ∞ ⟹ *normalize-qbs s* = *density-qbs s* (λ*x*. *1* / *emeasure* (*qbs-l s*) (*qbs-space X*))
and *normalize-qbs0-all-meas*: *qbs-l s* (*qbs-space X*) = *0* ⟹ *normalize-qbs s* = *qbs-null-measure X*
and *normalize-qbsinfty-all-meas*: *qbs-l s* (*qbs-space X*) = ∞ ⟹ *normalize-qbs s* = *qbs-null-measure X*
by(*auto simp*: *qbs-space-of-in-all-meas*[*OF assms*(*1*)] *normalize-qbs-def*)

lemma
assumes *s* ∈ *qbs-space* (*monadM-qbs X*)
shows *normalize-qbs*: *qbs-l s* (*qbs-space X*) ≠ *0* ⟹ *qbs-l s* (*qbs-space X*) ≠ ∞ ⟹ *normalize-qbs s* = *density-qbs s* (λ*x*. *1* / *emeasure* (*qbs-l s*) (*qbs-space X*))
and *normalize-qbs0*: *qbs-l s* (*qbs-space X*) = *0* ⟹ *normalize-qbs s* = *qbs-null-measure X*
and *normalize-qbsinfty*: *qbs-l s* (*qbs-space X*) = ∞ ⟹ *normalize-qbs s* = *qbs-null-measure X*
by(*auto simp*: *qbs-space-of-in*[*OF assms*(*1*)] *normalize-qbs-def*)

lemma *normalize-qbs-prob-all-meas*:
assumes *s* ∈ *qbs-space* (*all-meas-qbs X*) *qbs-l s* (*qbs-space X*) ≠ *0 qbs-l s* (*qbs-space X*) ≠ ∞
shows *normalize-qbs s* ∈ *qbs-space* (*monadP-qbs X*)

unfolding *normalize-qbs-all-meas*[*OF assms*]
proof −
obtain α μ
where *hs*: $s = [\![X, \alpha, \mu]\!]_{meas}$ *qbs-meas X* α μ
using *rep-qbs-space-all-meas assms*(*1*) **by** *meson*
interpret *qs*: *qbs-meas X* α μ **by** *fact*
have *1 / emeasure* μ (*space* μ) ∗ *emeasure* μ (*space* μ) = *1*
using *assms*(*2*,*3*) **by**(*auto simp*: *hs*(*1*) *qs.qbs-l emeasure-distr*[*of - - qbs-to-measure X*,*OF - sets.top*,*simplified space-L*] *divide-eq-1-ennreal ennreal-divide-times*)
hence ∗: *prob-space* (*density* μ (λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*)))
by(*auto intro*!: *prob-spaceI simp*: *emeasure-density hs*(*1*) *qs.qbs-l emeasure-distr*[*of - - qbs-to-measure X*,*OF - sets.top*,*simplified space-L*])
have *density-qbs s* (λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*)) = *density-qbs* $[\![X, \alpha, \mu]\!]_{meas}$ (λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*))
by(*simp add*: *hs*)
also have ... ∈ *qbs-space* (*monadP-qbs X*)
using ∗
by(*auto simp*: *qs.density-qbs monadP-qbs-space*
qbs-meas.qbs-l[*OF qs.density-qbs-meas*,*of* λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*),*simplified*]
qbs-meas.qbs-space-of[*OF qs.density-qbs-meas*,*of* λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*),*simplified*]
intro!: *prob-space.prob-space-distr*)
finally show *density-qbs s* (λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*)) ∈ *qbs-space* (*monadP-qbs X*) **.**
qed

lemmas *normalize-qbs-prob* = *normalize-qbs-prob-all-meas*[*OF monadM-all-meas-space*]

lemma *normalize-qbs-morphism*[*qbs*]: *normalize-qbs* ∈ *monadM-qbs X* →$_Q$ *monadM-qbs X*
proof −
have (*if emeasure* (*qbs-l s*) (*qbs-space X*) ≠ *0* ∧ *emeasure* (*qbs-l s*) (*qbs-space X*) ≠ ∞ *then density-qbs s* (λx. *1 / emeasure* (*qbs-l s*) (*qbs-space X*)) *else qbs-null-measure X*) = *normalize-qbs s* (**is** *?f s* = -) **if** *s*:*s* ∈ *qbs-space* (*monadM-qbs X*) **for** *s*
by(*simp add*: *qbs-space-of-in*[*OF s*] *normalize-qbs-def*)
moreover have (λs. *?f s*) ∈ *monadM-qbs X* →$_Q$ *monadM-qbs X*
proof(*cases qbs-space X* = {})
case *True*
then show *?thesis*
by(*simp add*: *qbs-morphism-from-empty monadM-qbs-empty-iff*[*of X*])
next
case *X*:*False*
have [*qbs*]:(λs. *emeasure* (*qbs-l s*) (*qbs-space X*)) ∈ *monadM-qbs X* →$_Q$ *qbs-borel*
by(*rule qbs-l-morphism*[*OF sets.top*[*of qbs-to-measure X*,*simplified space-L*]])
have [*qbs*]: *qbs-null-measure X* ∈ *qbs-space* (*monadM-qbs X*)
by(*auto intro*!: *qbs-null-measure-in-Mx X*)
have [*qbs*]: (λs *x*. *1 / emeasure* (*qbs-l s*) (*qbs-space X*)) ∈ *monadM-qbs X* →$_Q$ *X* ⇒$_Q$ *qbs-borel*

```
    by(rule arg-swap-morphism) simp
  show ?thesis
    by qbs
 qed
 ultimately show ?thesis
  by(simp cong: qbs-morphism-cong)
qed

lemma normalize-qbs-morphismP:
 assumes [qbs]:s ∈ X →Q monadM-qbs Y
    and ⋀x. x ∈ qbs-space X ⟹ qbs-l (s x) (qbs-space Y) ≠ 0
    and ⋀x. x ∈ qbs-space X ⟹ qbs-l (s x) (qbs-space Y) ≠ ∞
  shows (λx. normalize-qbs (s x)) ∈ X →Q monadP-qbs Y
 by(rule qbs-morphism-monadPI'[OF normalize-qbs-prob]) (use assms(2,3) qbs-morphism-space[OF
assms(1)] in auto)

lemma normalize-qbs-monadP-ident:
 assumes s ∈ qbs-space (monadP-qbs X)
 shows normalize-qbs s = s
 using normalize-qbs[OF qbs-space-monadPM[OF assms]] prob-space.emeasure-space-1[OF
qbs-l-prob-space[OF assms]]
 by(auto simp: space-qbs-l-in[OF qbs-space-monadPM[OF assms]] intro!: inj-onD[OF
qbs-l-inj-P - - assms])

corollary normalize-qbs-idenpotent: normalize-qbs (normalize-qbs s) = normal-
ize-qbs s
proof −
 obtain X where s[qbs]: s ∈ qbs-space (all-meas-qbs X)
  using in-qbs-space-of-all-meas by blast
 then have X: qbs-space X ≠ {}
  using all-meas-qbs-empty-iff by blast
 then obtain x where x:x ∈ qbs-space X by auto
 consider qbs-l s (qbs-space X) = 0 | qbs-l s (qbs-space X) = ⊤ | qbs-l s (qbs-space
X) ≠ 0  qbs-l s (qbs-space X) ≠ ⊤
  by auto
 then show ?thesis
 proof cases
  case 1
  then show ?thesis
   using normalize-qbs0[OF qbs-null-measure-in-Mx[OF X]]
   by(simp add: normalize-qbs0-all-meas[OF s] qbs-null-measure-null-measure[OF
X])
 next
  case 2
  then show ?thesis
   using normalize-qbs0[OF qbs-null-measure-in-Mx[OF X]]
   by(simp add: normalize-qbsinfty-all-meas[OF s] qbs-null-measure-null-measure[OF
X])
 next
```

case *3*
have *normalize-qbs s ∈ qbs-space (monadP-qbs X)*
by(*intro normalize-qbs-prob-all-meas*) (*auto simp*: *3 x*)
then show *?thesis*
by(*simp add*: *normalize-qbs-monadP-ident*)
qed
qed

4.1.18 Product Measures

definition *PiQ-measure* :: [*'a set*, *'a ⇒ 'b qbs-measure*] ⇒ (*'a ⇒ 'b*) *qbs-measure*
where
PiQ-measure ≡ (λ*I si*. *if* (∀ *i*∈*I*. ∃ *Mi*. *standard-borel-ne Mi* ∧ *si i* ∈ *qbs-space* (*monadM-qbs* (*measure-to-qbs Mi*)))
then if countable I ∧ (∀ *i*∈*I*. *prob-space* (*qbs-l* (*si i*))) *then* *qbs-l-inverse* (Π$_M$ *i*∈*I*. *qbs-l* (*si i*))
else if finite I ∧ (∀ *i*∈*I*. *sigma-finite-measure* (*qbs-l* (*si i*))) *then qbs-l-inverse* (Π$_M$ *i*∈*I*. *qbs-l* (*si i*))
else qbs-null-measure (Π$_Q$ *i*∈*I*. *qbs-space-of* (*si i*))
else qbs-null-measure (Π$_Q$ *i*∈*I*. *qbs-space-of* (*si i*)))

syntax
-PiQ-measure :: *pttrn ⇒ 'i set ⇒ 'a qbs-measure ⇒ ('i => 'a) qbs-measure* ((*3*Π$_{Qmeas}$ *-*∈*-./ -*) *10*)
syntax-consts
-PiQ-measure ⇌ *PiQ-measure*
translations
Π$_{Qmeas}$ *x*∈*I*. *X* == *CONST PiQ-measure I* (λ*x*. *X*)

context
fixes *I* **and** *Mi*
assumes *standard-borel-ne*:⋀*i*. *i* ∈ *I* ⟹ *standard-borel-ne* (*Mi i*)
begin

context
assumes *countableI*:*countable I*
begin

interpretation *sb*:*standard-borel-ne* Π$_M$ *i*∈*I*. (*borel* :: *real measure*)
by (*simp add*: *countableI product-standard-borel-ne*)

interpretation *sbM*: *standard-borel-ne* Π$_M$ *i*∈*I*. *Mi i*
by (*simp add*: *countableI standard-borel-ne product-standard-borel-ne*)

lemma
assumes ⋀*i*. *i* ∈ *I* ⟹ *si i* ∈ *qbs-space* (*monadP-qbs* (*measure-to-qbs* (*Mi i*)))
and ⋀*i*. *i* ∈ *I* ⟹ *si i* = ⟦*measure-to-qbs* (*Mi i*), α *i*, μ *i*⟧$_{meas}$ ⋀*i*. *i* ∈ *I* ⟹ *qbs-prob* (*measure-to-qbs* (*Mi i*)) (α *i*) (μ *i*)
shows *PiQ-measure-prob-eq*: (Π$_{Qmeas}$ *i*∈*I*. *si i*) = ⟦*measure-to-qbs* (Π$_M$ *i*∈*I*.

Mi i), *sbM.from-real*, *distr* (Π_M *i∈I*. *qbs-l* (*si i*)) *borel sbM.to-real*⟧$_{meas}$ (**is** - = *?rhs*)
and *PiQ-measure-qbs-prob*: *qbs-prob* (*measure-to-qbs* (Π_M *i∈I*. *Mi i*)) *sbM.from-real* (*distr* (Π_M *i∈I*. *qbs-l* (*si i*)) *borel sbM.to-real*) (**is** *?qbsprob*)
proof −
have [*measurable-cong*,*simp*]: *prob-space* (Π_M *i∈I*. *qbs-l* (*si i*)) *sets* (Π_M *i∈I*. *qbs-l* (*si i*)) = *sets* (Π_M *i∈I*. *Mi i*)
using *sets-qbs-l*[*OF assms*(*1*)[*THEN qbs-space-monadPM*]] *standard-borel.lr-sets-ident*[*OF standard-borel-ne.standard-borel*[*OF standard-borel-ne*]]
by(*auto cong*: *sets-PiM-cong intro*!: *prob-space-PiM qbs-l-prob-space assms*(*1*))
show *?qbsprob*
by(*auto simp*: *pair-qbs-s-finite-def intro*!: *qbs-prob.qbs-s-finite sbM.qbs-l-inverse-qbs-prob*)
have (Π_{Qmeas} *i∈I*. *si i*) = *qbs-l-inverse* (Π_M *i∈I*. *qbs-l* (*si i*))
using *countableI assms*(*1*)[*THEN qbs-space-monadPM*] *qbs-l-prob-space*[*OF assms*(*1*)] *standard-borel-ne* **by**(*auto simp*: *PiQ-measure-def*)
also have ... = *?rhs*
by(*auto intro*!: *sbM.qbs-l-inverse-def2 prob-space.s-finite-measure-prob cong*: *sets-PiM-cong*[*OF refl*])
finally show (Π_{Qmeas} *i∈I*. *si i*) = *?rhs* **.**
qed

lemma *qbs-l-PiQ-measure-prob*:
assumes ⋀*i*. *i* ∈ *I* ⟹ *si i* ∈ *qbs-space* (*monadP-qbs* (*measure-to-qbs* (*Mi i*)))
shows *qbs-l* (Π_{Qmeas} *i∈I*. *si i*) = (Π_M *i∈I*. *qbs-l* (*si i*))
proof −
have *qbs-l* (Π_{Qmeas} *i∈I*. *si i*) = *qbs-l* (*qbs-l-inverse* (Π_M *i∈I*. *qbs-l* (*si i*)))
using *countableI assms*(*1*)[*THEN qbs-space-monadPM*] *qbs-l-prob-space*[*OF assms*(*1*)] *standard-borel-ne* **by**(*auto simp*: *PiQ-measure-def*)
also have ... = (Π_M *i∈I*. *qbs-l* (*si i*))
using *sets-qbs-l*[*OF assms*(*1*)[*THEN qbs-space-monadPM*]] *standard-borel.lr-sets-ident*[*OF standard-borel-ne.standard-borel*[*OF standard-borel-ne*]]
by(*auto intro*!: *sbM.qbs-l-qbs-l-inverse prob-space-PiM qbs-l-prob-space*[*OF assms*(*1*)] *cong*: *sets-PiM-cong*)
finally show *?thesis* **.**
qed

end

context
assumes *finI*: *finite I*
begin

interpretation *sb*:*standard-borel-ne* Π_M *i∈I*. (*borel* :: *real measure*)
by (*simp add*: *finI product-standard-borel-ne countable-finite*)

interpretation *sbM*: *standard-borel-ne* Π_M *i∈I*. *Mi i*
by (*simp add*: *countable-finite finI standard-borel-ne product-standard-borel-ne*)

lemma *qbs-l-PiQ-measure*:

assumes $\bigwedge i.\ i \in I \Longrightarrow si\ i \in qbs\text{-}space\ (monadM\text{-}qbs\ (measure\text{-}to\text{-}qbs\ (Mi\ i)))$
and $\bigwedge i.\ i \in I \Longrightarrow sigma\text{-}finite\text{-}measure\ (qbs\text{-}l\ (si\ i))$
shows $qbs\text{-}l\ (\Pi_{Qmeas}\ i{\in}I.\ si\ i) = (\Pi_M\ i{\in}I.\ qbs\text{-}l\ (si\ i))$
proof −
have [*simp*]: *s-finite-measure* $(\Pi_M\ i{\in}I.\ qbs\text{-}l\ (si\ i))$
proof −
have $(\Pi_M\ i{\in}I.\ qbs\text{-}l\ (si\ i)) = (\Pi_M\ i{\in}I.\ if\ i \in I\ then\ qbs\text{-}l\ (si\ i)\ else\ null\text{-}measure\ (count\text{-}space\ UNIV))$
by(*simp cong*: *PiM-cong*)
also have *s-finite-measure* ...
by(*auto intro*!: *sigma-finite-measure.s-finite-measure product-sigma-finite.sigma-finite finI simp*: *product-sigma-finite-def assms*(*2*)) (*auto intro*!: *finite-measure.sigma-finite-measure finite-measureI*)
finally show *?thesis* **.**
qed
have $qbs\text{-}l\ (\Pi_{Qmeas}\ i{\in}I.\ si\ i) = qbs\text{-}l\ (qbs\text{-}l\text{-}inverse\ (\Pi_M\ i{\in}I.\ qbs\text{-}l\ (si\ i)))$
using *finI assms*(*1*) *assms*(*2*) *standard-borel-ne* **by**(*fastforce simp*: *PiQ-measure-def*)
also have $... = (\Pi_M\ i{\in}I.\ qbs\text{-}l\ (si\ i))$
using *sets-qbs-l*[*OF assms*(*1*)] *standard-borel.lr-sets-ident*[*OF standard-borel-ne.standard-borel*[*OF standard-borel-ne*]]
by(*auto intro*!: *sbM.qbs-l-qbs-l-inverse prob-space-PiM cong*: *sets-PiM-cong*)
finally show *?thesis* **.**
qed

end

end

4.2 Measures

4.2.1 The Lebesgue Measure

definition *lborel-qbs* $(lborel_Q)$ **where** *lborel-qbs* ≡ *qbs-l-inverse lborel*

lemma *lborel-qbs-qbs*[*qbs*]: *lborel-qbs* ∈ *qbs-space* (*monadM-qbs qbs-borel*)
by(*auto simp*: *lborel-qbs-def measure-to-qbs-cong-sets*[*OF sets-lborel*,*symmetric*] *intro*!: *standard-borel-ne.qbs-l-inverse-in-space-monadM lborel.s-finite-measure-axioms*)

lemma *qbs-l-lborel-qbs*[*simp*]: $qbs\text{-}l\ lborel_Q = lborel$
by(*auto intro*!: *standard-borel-ne.qbs-l-qbs-l-inverse lborel.s-finite-measure-axioms simp*: *lborel-qbs-def*)

corollary
shows *qbs-integral-lborel*: $(\int_Q x.\ f\ x\ \partial lborel\text{-}qbs) = (\int x.\ f\ x\ \partial lborel)$
and *qbs-nn-integral-lborel*: $(\int^+_Q x.\ f\ x\ \partial lborel\text{-}qbs) = (\int^+ x.\ f\ x\ \partial lborel)$
by(*simp-all add*: *qbs-integral-def2-l qbs-nn-integral-def2-l*)

lemma(**in** *standard-borel-ne*) *measure-with-args-morphism*:

assumes *s-finite-kernel X M k*
shows *qbs-l-inverse ∘ k ∈ measure-to-qbs X →Q monadM-qbs (measure-to-qbs M)*
proof(*safe intro*!: *qbs-morphismI*)
fix *α* :: *real ⇒ -*
assume *α ∈ qbs-Mx (measure-to-qbs X)*
then have *h*[*measurable*]:*α ∈ borel →M X*
by(*simp add*: *qbs-Mx-R*)
interpret *s*:*s-finite-kernel X M k* **by** *fact*
have *1*: *⋀r. sets (k (α r)) = sets M ⋀r. s-finite-measure (k (α r))*
using *measurable-space*[*OF h*] *s.kernel-sets* **by**(*auto intro*!: *s.image-s-finite-measure*)
show *qbs-l-inverse ∘ k ∘ α ∈ qbs-Mx (monadM-qbs (measure-to-qbs M))*
by(*auto intro*!: *exI*[**where** *x*=*from-real*] *exI*[**where** *x*=(*λr. distr (k (α r)) borel to-real*)] *s-finite-kernel.comp-measurable*[*OF s-finite-kernel.distr-s-finite-kernel*[*OF assms*]]
simp: *monadM-qbs-Mx qbs-Mx-R qbs-l-inverse-def2*[*OF 1*(*1*)] *comp-def*)
qed

lemma(**in** *standard-borel-ne*) *measure-with-args-morphismP*:
assumes [*measurable*]:*μ ∈ X →M prob-algebra M*
shows *qbs-l-inverse ∘ μ ∈ measure-to-qbs X →Q monadP-qbs (measure-to-qbs M)*
using *measurable-space*[*OF assms*]
by(*intro qbs-morphism-monadPI'*[*OF - measure-with-args-morphism*])
(*auto simp*: *qbs-space-R space-prob-algebra prob-kernel-def'*
intro!: *qbs-l-inverse-in-space-monadP prob-kernel.s-finite-kernel-prob-kernel*)

4.2.2 Counting Measure

abbreviation *counting-measure-qbs A ≡ qbs-l-inverse (count-space A)*

lemma *qbs-nn-integral-count-space-nat*:
fixes *f* :: *nat ⇒ ennreal*
shows (*∫*^+_Q *i. f i ∂counting-measure-qbs UNIV*) = (*∑ i. f i*)
by(*simp add*: *standard-borel-ne.qbs-l-qbs-l-inverse*[*OF - refl*] *qbs-nn-integral-def2-l nn-integral-count-space-nat*)

4.2.3 Normal Distribution

lemma *qbs-normal-distribution-qbs*: (*λμ σ. density-qbs lborel*$_Q$ (*normal-density μ σ*)) ∈ *qbs-borel ⇒Q qbs-borel ⇒Q monadM-qbs qbs-borel*
by *simp*

lemma *qbs-l-qbs-normal-distribution*[*simp*]: *qbs-l* (*density-qbs lborel*$_Q$ (*normal-density μ σ*)) = *density lborel* (*normal-density μ σ*)
by(*auto simp*: *qbs-l-density-qbs*[*of - qbs-borel*])

lemma *qbs-normal-distribution-P*: *σ > 0 ⟹ density-qbs lborel*$_Q$ (*normal-density μ σ*) ∈ *qbs-space* (*monadP-qbs qbs-borel*)

by(*auto simp*: *monadP-qbs-def sub-qbs-space prob-space-normal-density*)

lemma *qbs-normal-distribution-integral*:
($\int_Q$ *x*. *f x* ∂ (*density-qbs lborel*$_Q$ (*normal-density* μ σ))) = ($\int$ *x*. *f x* ∂ (*density lborel* (λ*x*. *ennreal* (*normal-density* μ σ *x*))))
by(*auto simp*: *qbs-integral-def2-l*)

lemma *qbs-normal-distribution-expectation*:
assumes [*measurable*]:*f* ∈ *borel-measurable borel* **and** [*arith*]: σ > *0*
shows ($\int_Q$ *x*. *f x* ∂ (*density-qbs lborel*$_Q$ (*normal-density* μ σ))) = ($\int$ *x*. *normal-density* μ σ *x* ∗ *f x* ∂ *lborel*)
by(*simp add*: *qbs-normal-distribution-integral integral-real-density integral-density*)

lemma *qbs-normal-posterior*:
assumes [*arith*]: σ > *0* σ′ > *0*
shows *normalize-qbs* (*density-qbs* (*density-qbs lborel*$_Q$ (*normal-density* μ σ)) (*normal-density* μ′ σ′)) = *density-qbs lborel*$_Q$ (*normal-density* ((μ ∗ σ′^{2} + μ′ ∗ σ^2) / (σ^2 + σ′^{2})) (σ ∗ σ′ / *sqrt* (σ^2 + σ′^{2}))) (**is** *?lhs* = *?rhs*)
proof −
have *0*: σ ∗ σ′ / *sqrt* (σ^2 + σ′^{2}) > *0* *sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ′^{2})) > *0*
by (*simp-all add*: *power2-eq-square sum-squares-gt-zero-iff*)
have *1*:*qbs-l* (*density-qbs lborel*$_Q$ (λ*x*. *ennreal* (*1* / *sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ′^{2})) ∗ *exp* (− ((μ − μ′)^2 / (*2* ∗ σ^2 + *2* ∗ σ′^{2}))) ∗ *normal-density* ((μ ∗ σ′^{2} + μ′ ∗ σ^2) / (σ^2 + σ′^{2})) (σ ∗ σ′ / *sqrt* (σ^2 + σ′^{2})) *x*))) *UNIV* = *ennreal* (*exp* (− ((μ − μ′)^2 / (*2* ∗ σ^2 + *2* ∗ σ′^{2}))) / *sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ′^{2})))
using *prob-space.emeasure-space-1*[*OF prob-space-normal-density*[*OF 0*(*1*)]]
by(*auto simp add*: *qbs-l-density-qbs*[*of - qbs-borel*] *emeasure-density ennreal-mult′ nn-integral-cmult simp del*: *times-divide-eq-left*) (*simp add*: *ennreal-mult′*[*symmetric*])
have *?lhs* = *normalize-qbs* (*density-qbs lborel*$_Q$ (λ*x*. *ennreal* (*1* / *sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ′^{2})) ∗ *exp* (− ((μ − μ′)^2 / (*2* ∗ σ^2 + *2* ∗ σ′^{2}))) ∗ *normal-density* ((μ ∗ σ′^{2} + μ′ ∗ σ^2) / (σ^2 + σ′^{2})) (σ ∗ σ′ / *sqrt* (σ^2 + σ′^{2})) *x*)))
by(*simp add*: *density-qbs-density-qbs-eq*[*of - qbs-borel*] *ennreal-mult′*[*symmetric*] *normal-density-times del*: *times-divide-eq-left*)
also have ... = *density-qbs* (*density-qbs lborel*$_Q$ (λ*x*. *ennreal* (*1* / *sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ′^{2})) ∗ *exp* (− ((μ − μ′)^2 / (*2* ∗ σ^2 + *2* ∗ σ′^{2}))) ∗ *normal-density* ((μ ∗ σ′^{2} + μ′ ∗ σ^2) / (σ^2 + σ′^{2})) (σ ∗ σ′ / *sqrt* (σ^2 + σ′^{2})) *x*))) (λ*x*. *1* / *emeasure* (*qbs-l* (*density-qbs lborel*$_Q$ (λ*x*. *ennreal* (*1* / *sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ′^{2})) ∗ *exp* (− ((μ − μ′)^2 / (*2* ∗ σ^2 + *2* ∗ σ′^{2}))) ∗ *normal-density* ((μ ∗ σ′^{2} + μ′ ∗ σ^2) / (σ^2 + σ′^{2})) (σ ∗ σ′ / *sqrt* (σ^2 + σ′^{2})) *x*))) (*qbs-space borel*$_Q$))
by(*rule normalize-qbs*) (*simp-all add*: *1 del*: *times-divide-eq-left*)
also have ... = *?rhs*
by(*simp add*: *1 density-qbs-density-qbs-eq*[*of - qbs-borel*] *del*: *times-divide-eq-left*, *auto intro*!: *density-qbs-cong*[*of - qbs-borel*])
(*insert 0*, *auto simp*: *ennreal-1*[*symmetric*] *ennreal-mult′*[*symmetric*] *divide-ennreal normal-density-def simp del*: *ennreal-1*)
finally show *?thesis* **.**
qed

4.2.4 Uniform Distribution

definition *uniform-qbs* :: *'a qbs-measure ⇒ 'a set ⇒ 'a qbs-measure* **where**
uniform-qbs ≡ (λs A. qbs-l-inverse (uniform-measure (qbs-l s) A))

lemma(**in** *standard-borel-ne*) *qbs-l-uniform-qbs'*:
assumes *sets μ = sets M μ A ≠ 0*
shows *qbs-l (uniform-qbs (qbs-l-inverse μ) A) = uniform-measure μ A* (**is** *?lhs = ?rhs*)
proof −
have *?lhs = qbs-l (qbs-l-inverse (uniform-measure μ A))*
by(*simp add*: *qbs-l-qbs-l-inverse*[*OF assms*(*1*)] *uniform-qbs-def*)
also have *... = ?rhs*
by(*rule qbs-l-qbs-l-inverse*) (*simp add*: *assms*)
finally show *?thesis* **.**
qed

corollary(**in** *standard-borel-ne*) *qbs-l-uniform-qbs*:
assumes *s ∈ qbs-space (monadM-qbs (measure-to-qbs M)) qbs-l s A ≠ 0*
shows *qbs-l (uniform-qbs s A) = uniform-measure (qbs-l s) A*
by(*simp add*: *qbs-l-uniform-qbs'*[*OF sets-qbs-l*[*OF assms*(*1*),*simplified lr-sets-ident*] *assms*(*2*),*symmetric*] *qbs-l-inverse-qbs-l assms*)

lemma *interval-uniform-qbs*: *(λa b. uniform-qbs lborel$_Q$ {a<..<b::real}) ∈ borel$_Q$ ⇒$_Q$ borel$_Q$ ⇒$_Q$ monadM-qbs borel$_Q$*
proof(*rule curry-preserves-morphisms*)
have *(λxy. uniform-qbs lborel$_Q$ {fst xy<..<snd xy::real}) = qbs-l-inverse ∘ (λxy. uniform-measure lborel {fst xy<..<snd xy})*
by(*auto simp*: *uniform-qbs-def*)
also have *... ∈ measure-to-qbs (borel ⨂$_M$ borel) →$_Q$ monadM-qbs borel$_Q$*
proof(*safe intro*!: *standard-borel-ne.measure-with-args-morphism measure-kernel.s-finite-kernel-finite-bounde*
show *measure-kernel (borel ⨂$_M$ borel) borel (λxy. uniform-measure lborel {fst xy<..<snd xy :: real})*
proof
fix *B :: real set*
assume [*measurable*]:*B ∈ sets borel*
have [*simp*]:*emeasure lborel ({fst x<..<snd x} ∩ B) / emeasure lborel {fst x<..<snd x} = (if fst x ≤ snd x then emeasure lborel ({fst x<..<snd x} ∩ B) / ennreal (snd x − fst x) else 0)* **for** *x*
by *auto*
show *(λx. emeasure (uniform-measure lborel {fst x<..<snd x}) B) ∈ borel-measurable (borel ⨂$_M$ borel)*
by (*simp*, *measurable*) *auto*
qed *auto*
next
show *(a, b) ∈ space (borel ⨂$_M$ borel) ⟹ emeasure (uniform-measure lborel {fst (a, b)<..<snd (a, b)}) (space borel) < ∞* **for** *a b :: real*
by(*cases a ≤ b*) (*use ennreal-divide-eq-top-iff top.not-eq-extremum* **in** *auto*)
qed *simp*
finally show *(λxy. uniform-qbs lborel$_Q$ {fst xy<..<snd xy::real}) ∈ borel$_Q$ ⨂$_Q$*

$borel_Q \to_Q monadM\text{-}qbs\ borel_Q$
by (*simp add*: *borel-prod qbs-borel-prod*)
qed

context
fixes $a\ b :: real$
assumes [*arith*]:$a < b$
begin

lemma *qbs-uniform-distribution-expectation*:
assumes $f \in qbs\text{-}borel \to_Q qbs\text{-}borel$
shows $(\int^+_Q x.\ f\ x\ \partial uniform\text{-}qbs\ lborel_Q\ \{a<..<b\}) = (\int^+ x \in \{a<..<b\}.\ f\ x\ \partial lborel) / (b - a)$
proof −
have [*measurable*]: $f \in borel\text{-}measurable\ borel$
using *assms qbs-Mx-is-morphisms qbs-Mx-qbs-borel* **by** *blast*
show *?thesis*
by(*auto simp*: *qbs-nn-integral-def2-l standard-borel-ne.qbs-l-uniform-qbs*[*of borel lborel-qbs*] *nn-integral-uniform-measure*)
qed

end

4.2.5 Bernoulli Distribution

abbreviation *qbs-bernoulli* :: *real* ⇒ *bool qbs-measure* **where**
$qbs\text{-}bernoulli \equiv (\lambda x.\ qbs\text{-}pmf\ (bernoulli\text{-}pmf\ x))$

lemma *bernoulli-measurable*:
$(\lambda x.\ measure\text{-}pmf\ (bernoulli\text{-}pmf\ x)) \in borel \to_M prob\text{-}algebra\ (count\text{-}space\ UNIV)$
proof(*rule measurable-prob-algebra-generated*[**where** Ω=*UNIV* **and** *G*=*UNIV*])
fix $A :: bool\ set$
have $A \subseteq \{True,False\}$
by *auto*
then consider $A = \{\} \mid A = \{True\} \mid A = \{False\} \mid A = \{False, True\}$
by *auto*
thus $(\lambda a.\ emeasure\ (measure\text{-}pmf\ (bernoulli\text{-}pmf\ a))\ A) \in borel\text{-}measurable\ borel$
by(*cases*,*simp-all add*: *emeasure-measure-pmf-finite bernoulli-pmf.rep-eq UNIV-bool*[*symmetric*])
qed (*auto simp add*: *sets-borel-eq-count-space Int-stable-def measure-pmf.prob-space-axioms*)

lemma *qbs-bernoulli-morphism*: $qbs\text{-}bernoulli \in qbs\text{-}borel \to_Q monadP\text{-}qbs\ (qbs\text{-}count\text{-}space\ UNIV)$
using *standard-borel-ne.measure-with-args-morphismP*[*OF - bernoulli-measurable*]
by (*simp add*: *qbs-pmf-def comp-def*)

lemma *qbs-bernoulli-expectation*:
assumes [*simp*]: $0 \le p\ p \le 1$
shows $(\int_Q x.\ f\ x\ \partial qbs\text{-}bernoulli\ p) = f\ True * p + f\ False * (1 - p)$
by(*simp add*: *qbs-integral-def2-l*)

end

5 Examples

5.1 Montecarlo Approximation

theory *Montecarlo*
imports *Monad-QuasiBorel*
begin

declare [[*coercion qbs-l*]]

abbreviation *real-quasi-borel* :: *real quasi-borel* ($\mathbb{R}_Q$) **where**
real-quasi-borel ≡ *qbs-borel*
abbreviation *nat-quasi-borel* :: *nat quasi-borel* ($\mathbb{N}_Q$) **where**
nat-quasi-borel ≡ *qbs-count-space UNIV*

primrec *montecarlo* :: *'a qbs-measure* ⇒ (*'a* ⇒ *real*) ⇒ *nat* ⇒ *real qbs-measure*
where
montecarlo - - 0 = *return-qbs* $\mathbb{R}_Q$ *0* |
montecarlo d h (*Suc n*) = *do* { *m* ← *montecarlo d h n*;
x ← *d*;
return-qbs $\mathbb{R}_Q$ ((*h x* + *m* ∗ *real n*) / *real* (*Suc n*))}

declare
bind-qbs-morphismP[*qbs*]
return-qbs-morphismP[*qbs*]
qbs-pair-measure-morphismP[*qbs*]
qbs-space-monadPM[*qbs*]

lemma *montecarlo-qbs-morphism*[*qbs*]: *montecarlo* ∈ *qbs-space* (*monadP-qbs X* ⇒$_Q$ (*X* ⇒$_Q$ $\mathbb{R}_Q$) ⇒$_Q$ $\mathbb{N}_Q$ ⇒$_Q$ *monadP-qbs* $\mathbb{R}_Q$)
by(*simp add*: *montecarlo-def*)

lemma *qbs-integrable-indep-mult2*:
fixes *f* :: *-* ⇒ *real*
assumes *p* ∈ *qbs-space* (*monadM-qbs X*) *q* ∈ *qbs-space* (*monadM-qbs Y*)
and *qbs-integrable p f*
and *qbs-integrable q g*
shows *qbs-integrable* (*p* $\bigotimes_{Qmes}$ *q*) (λ*x*. *g* (*snd x*) ∗ *f* (*fst x*))
using *qbs-integrable-indep-mult*[*OF assms*] **by** (*simp add*: *mult.commute*)

lemma *montecarlo-integrable*:
assumes [*qbs*]:*p* ∈ *qbs-space* (*monadP-qbs X*) *h* ∈ *X* →$_Q$ $\mathbb{R}_Q$ *qbs-integrable p h qbs-integrable p* (λ*x*. *h x* ∗ *h x*)
shows *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x*) *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x* ∗ *x*)

proof −
note *qbs-space-monadPM*[*qbs*]
have *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x*) ∧ *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x* ∗ *x*)
proof(*induction n*)
case *0*
then show *?case*
by *simp*
next
case (*Suc n*)
hence *ih*[*intro*,*simp*]:*qbs-integrable* (*montecarlo p h n*) (λ*x*. *x*) *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x* ∗ *x*)
by *simp-all*
have *qbs-integrable* (*montecarlo p h n* ⨂$_{Qmes}$ *p*) (λ*x*. (*h* (*snd x*) + *fst x* ∗ *real n*) ∗ (*h* (*snd x*) + *fst x* ∗ *real n*))
proof(*subst qbs-integrable-cong*[*OF qbs-space-monadPM*[**where** *X*=ℝ$_Q$ ⨂$_Q$ *X*]])
show *qbs-integrable* (*montecarlo p h n* ⨂$_{Qmes}$ *p*) (λ*x*. *h* (*snd x*) ∗ *h* (*snd x*) + *fst x* ∗ *h* (*snd x*) ∗ *2* ∗ *real n* + *fst x* ∗ *fst x* ∗ *real n* ∗ *real n*)
by(*auto intro*!: *qbs-integrable-indep-mult*[*OF qbs-space-monadPM*[*of* - ℝ$_Q$] *qbs-space-monadPM*[*of* - *X*]] *qbs-integrable-indep2*[*OF* - *qbs-space-monadPM*,*of* - ℝ$_Q$ - *X*] *assms qbs-integrable-mult-left qbs-integrable-indep1*[*OF qbs-space-monadPM*,*of* - ℝ$_Q$ - *X*])
qed (*auto simp*: *distrib-right distrib-left*)
thus *?case*
by(*auto simp*: *qbs-bind-bind-return2P'*[*of* - ℝ$_Q$ *X* ℝ$_Q$] *qbs-pair-measure-def*[*OF qbs-space-monadPM qbs-space-monadPM*,*symmetric*] *split-beta'* *qbs-integrable-bind-return*[*OF qbs-space-monadPM*,*of* - ℝ$_Q$ ⨂$_Q$ *X* - ℝ$_Q$] *comp-def*
intro!: *qbs-integrable-indep2*[*OF* - *qbs-space-monadPM*,*of* - ℝ$_Q$ - *X*] *assms qbs-integrable-mult-left qbs-integrable-indep1*[*OF qbs-space-monadPM*,*of* - ℝ$_Q$ - *X*])
qed
thus *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x*) *qbs-integrable* (*montecarlo p h n*) (λ*x*. *x* ∗ *x*)
by *simp-all*
qed

lemma
fixes *n* :: *nat*
assumes [*qbs*,*simp*]:*p* ∈ *qbs-space* (*monadP-qbs X*) *h* ∈ *X* →$_Q$ ℝ$_Q$ *qbs-integrable p h qbs-integrable p* (λ*x*. *h x* ∗ *h x*)
and *e*:*e* > *0*
and (∫$_Q$ *x*. *h x* ∂*p*) = *μ* (∫$_Q$ *x*. (*h x* − *μ*)^2 ∂*p*) = *σ*^{2}
and *n*:*n* > *0*
shows *P*(*y in montecarlo p h n*. |*y* − *μ*| ≥ *e*) ≤ *σ*^{2} / (*real n* ∗ *e*^{2}) (**is** *?P* ≤ -)
proof −
have *monte-p*: ⋀*n*. *montecarlo p h n* ∈ *monadP-qbs* ℝ$_Q$
by *simp*
note [*intro*!, *simp*] =
montecarlo-integrable[*OF assms*(*1*−*4*)] *qbs-integrable-indep1*[**where** *X*=ℝ$_Q$

and $Y{=}X$]
qbs-integrable-indep2[**where** $X{=}\mathbb{R}_Q$ **and** $Y{=}X$] *qbs-integrable-sq qbs-integrable-mult-left qbs-integrable-mult-right*
qbs-integrable-const[*OF assms*(*1*)] *qbs-integrable-const*[*OF monte-p*]
qbs-integrable-indep-mult2[**where** $X{=}\mathbb{R}_Q$ **and** $Y{=}X$,*OF qbs-space-monadPM qbs-space-monadPM*]
have [*intro*!, *simp*]:*qbs-integrable p* ($\lambda x.\ (h\ x)^2$) $\bigwedge n.$ *qbs-integrable* (*montecarlo p h n*) *power2*
by(*simp-all add*: *power2-eq-square*)
have *exp*:($\int_Q$ *y*. *y* ∂(*montecarlo p h n*)) = μ (**is** *?e n*) **and** *var*:($\int_Q$ *y*. $(y - \mu)^2$ ∂(*montecarlo p h n*)) = σ^2 / *n* (**is** *?v n*)
proof −
have *?e n* ∧ *?v n*
proof(*rule nat-induct-non-zero*[*OF n*])
fix *n* :: *nat*
assume *n*[*arith*]:*0* < *n*
assume *?e n* ∧ *?v n*
hence *ih*: *?e n ?v n*
by *simp-all*
have *?e* (*Suc n*)
proof −
have ($\int_Q$ *y*. *y* ∂*montecarlo p h* (*Suc n*)) = ($\int_Q$ *x*. *h* (*snd x*) + *fst x* ∗ *real n* ∂(*montecarlo p h n* $\bigotimes_{Qmes}$ *p*)) / (*1* + *real n*)
by(*auto simp*: *qbs-bind-bind-return2P'*[*of* - $\mathbb{R}_Q$ *X* $\mathbb{R}_Q$] *split-beta' qbs-pair-measure-def*[*OF qbs-space-monadPM qbs-space-monadPM*,*symmetric*] *qbs-integral-bind-return*[*OF qbs-space-monadPM*,*of* - $\mathbb{R}_Q$ $\bigotimes_Q$ *X* - $\mathbb{R}_Q$] *comp-def*)
also have ... = (($\int_Q$ *x*. *h* (*snd x*) ∂(*montecarlo p h n* $\bigotimes_{Qmes}$ *p*)) + ($\int_Q$ *x*. *fst x* ∗ *real n* ∂(*montecarlo p h n* $\bigotimes_{Qmes}$ *p*))) / (*1* + *real n*)
by(*auto intro*!: *qbs-integral-add simp del*: *qbs-integral-mult-left-zero*)
also have ... = (μ + ($\int_Q$ *x*. *fst x*∂(*montecarlo p h n* $\bigotimes_{Qmes}$ *p*)) ∗ *real n*) / (*1* + *real n*)
by(*subst qbs-integral-indep2*[**where** $X{=}\mathbb{R}_Q$ **and** $Y{=}X$,*OF* - *qbs-space-monadPM*]) (*auto simp*: *assms*(*6*))
also have ... = (μ ∗ (*1* + *real n*)) / (*1* + *real n*)
by(*subst qbs-integral-indep1*[**where** $X{=}\mathbb{R}_Q$ **and** $Y{=}X$,*OF qbs-space-monadPM*]) (*auto simp*: *ih distrib-left*)
also have ... = μ
by *simp*
finally show *?e* (*Suc n*) **.**
qed
moreover have *?v* (*Suc n*)
proof −
have ($\int_Q$ *y*. $(y - \mu)^2$ ∂*montecarlo p h* (*Suc n*)) =($\int_Q$ *x*. ((*h* (*snd x*) + *fst x* ∗ *real n*) / *real* (*Suc n*) − μ)^2 ∂(*montecarlo p h n* $\bigotimes_{Qmes}$ *p*))
by(*auto simp*: *qbs-bind-bind-return2P'*[*of* - $\mathbb{R}_Q$ *X* $\mathbb{R}_Q$] *split-beta' qbs-pair-measure-def*[*OF qbs-space-monadPM qbs-space-monadPM*,*symmetric*] *qbs-integral-bind-return*[*OF qbs-space-monadPM*,*of* - $\mathbb{R}_Q$ $\bigotimes_Q$ *X* - $\mathbb{R}_Q$] *comp-def*)
also have ... = ($\int_Q$ *x*. ((*h* (*snd x*) + *fst x* ∗ *real n*) / *real* (*Suc n*) − (*Suc n*) ∗ μ / *Suc n*)^2 ∂(*montecarlo p h n* $\bigotimes_{Qmes}$ *p*))

by *simp*

also have ... = $(\int_Q x.\ ((h\ (snd\ x) + fst\ x * real\ n - (Suc\ n) * \mu)\ /\ Suc\ n)^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by(*simp only*: *diff-divide-distrib*[*symmetric*])

also have ... = $(\int_Q x.\ ((h\ (snd\ x) - \mu + (fst\ x * real\ n - real\ n * \mu))\ /\ Suc\ n)^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by (*simp add*: *add-diff-add distrib-left mult.commute*)

also have ... = $(\int_Q x.\ (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu) + n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by(*auto simp*: *add-divide-distrib*[*symmetric*] *pair-qbs-space mult.commute*[*of* - *real n*]) (*simp add*: *right-diff-distrib*)

also have ... = $(\int_Q x.\ (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu))^2 + (n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))^2 + 2 * (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu)) * (n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by(*simp add*: *power2-sum*)

also have ... = $(\int_Q x.\ 1\ /\ (real\ (Suc\ n))^2 * ((h\ (snd\ x) - \mu))^2 + (n\ /\ real\ (Suc\ n))^2 * ((fst\ x\ - \mu))^2 + 2 * (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu)) * (n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by(*simp only*: *power-mult-distrib*) (*simp add*: *power2-eq-square*)

also have ... = $(\int_Q x.\ 1\ /\ (real\ (Suc\ n))^2 * ((h\ (snd\ x) - \mu))^2 + (n\ /\ real\ (Suc\ n))^2 * ((fst\ x\ - \mu))^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p)) + (\int_Q x.\ 2 * (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu)) * (n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by(*rule qbs-integral-add*) *auto*

also have ... = $(\int_Q x.\ 1\ /\ (real\ (Suc\ n))^2 * ((h\ (snd\ x) - \mu))^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p)) + (\int_Q x.\ (n\ /\ real\ (Suc\ n))^2 * ((fst\ x\ - \mu))^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p)) + (\int_Q x.\ 2 * (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu)) * (n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p))$

by(*subst qbs-integral-add*) *auto*

also have ... = $1\ /\ (real\ (Suc\ n))^2 * \sigma^2 + (n\ /\ real\ (Suc\ n))^2 * (\sigma^2\ /\ n)$

proof −

have *1*: $(\int_Q x.\ ((h\ (snd\ x) - \mu))^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p)) = (\int_Q x.\ (h\ x - \mu)^2\ \partial p)$

by(*subst qbs-integral-indep2*[*of* - $\mathbb{R}_Q$ - *X*]) *auto*

have *2*: $(\int_Q x.\ ((fst\ x\ - \mu))^2\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p)) = (\int_Q y.\ (y - \mu)^2\ \partial montecarlo\ p\ h\ n)$

by(*subst qbs-integral-indep1*[*of* - $\mathbb{R}_Q$ - *X*]) *auto*

have *3*: $(\int_Q x.\ 2 * (1\ /\ real\ (Suc\ n) * (h\ (snd\ x) - \mu)) * (n\ /\ real\ (Suc\ n) * (fst\ x\ - \mu))\ \partial(montecarlo\ p\ h\ n \bigotimes_{Qmes} p)) = 0$ (**is** *?l* = -)

by(*subst qbs-integral-indep-mult2*[*of* - $\mathbb{R}_Q$ - *X*])

(*auto simp*: *qbs-integral-diff*[*OF montecarlo-integrable*(*1*)[*OF assms*(*1*−*4*)]] *qbs-integrable-const*[*of* - $\mathbb{R}_Q$]] *qbs-integral-const-prob*[*of* - $\mathbb{R}_Q$] *ih*)

show *?thesis*

unfolding *3* **by**(*simp add*: *1 2 ih assms*(*7*))

qed

also have ... = $1\ /\ (real\ (Suc\ n))^2 * \sigma^2 + real\ n\ /\ (real\ (Suc\ n))^2 * \sigma^2$

by(*auto simp*: *power2-eq-square*)

also have ... = $(1 + real\ n) * \sigma^2\ /\ (real\ (Suc\ n))^2$

by (*simp add*: *add-divide-distrib ring-class.ring-distribs*(*2*))

also have ... = σ^2 / *real* (*Suc n*)
by(*auto simp*: *power2-eq-square*)
finally show *?thesis* **.**
qed
ultimately show *?e* (*Suc n*) ∧ *?v* (*Suc n*)
by *blast*
qed(*simp add*: *assms*(6,7) *qbs-integral-bind-return*[**where** *Y*=*X*,*OF qbs-space-monadPM*] *bind-qbs-return*[**where** *Y*=$\mathbb{R}_Q$] *comp-def*)
thus *?e n ?v n* **by** *simp-all*
qed
have *?P* ≤ (∫ $_Q$ *x*. (*x* − *μ*)^2 ∂*montecarlo p h n*) / *e*^{2}
unfolding *exp*[*symmetric*] **by**(*rule Chebyshev-inequality-qbs-prob*[*of montecarlo p h n qbs-borel* λ*x*. *x*]) (*auto simp*: *power2-eq-square e*)
also have ... = σ^2 / (*real n* ∗ *e*^{2})
by(*simp add*: *var*)
finally show *?thesis* **.**
qed

end

5.2 Query

theory *Query*
imports *Monad-QuasiBorel*
begin

declare [[*coercion qbs-l*]]
abbreviation *qbs-real* :: *real quasi-borel* ($\mathbb{R}_Q$) **where** $\mathbb{R}_Q$ ≡ *qbs-borel*
abbreviation *qbs-ennreal* :: *ennreal quasi-borel* ($\mathbb{R}_{Q\geq 0}$) **where** $\mathbb{R}_{Q\geq 0}$ ≡ *qbs-borel*
abbreviation *qbs-nat* :: *nat quasi-borel* ($\mathbb{N}_Q$) **where** $\mathbb{N}_Q$ ≡ *qbs-count-space UNIV*
abbreviation *qbs-bool* :: *bool quasi-borel* ($\mathbb{B}_Q$) **where** $\mathbb{B}_Q$ ≡ *count-space*$_Q$ *UNIV*

definition *query* :: ['*a qbs-measure*, '*a* ⇒ *ennreal*] ⇒ '*a qbs-measure* **where**
query ≡ (λ*s f*. *normalize-qbs* (*density-qbs s f*))

lemma *query-qbs-morphism*[*qbs*]: *query* ∈ *monadM-qbs X* →$_Q$ (*X* ⇒$_Q$ *qbs-borel*) ⇒$_Q$ *monadM-qbs X*
by(*simp add*: *query-def*)

definition *condition* ≡ (λ*s P*. *query s* (λ*x*. *if P x then 1 else 0*))

lemma *condition-qbs-morphism*[*qbs*]: *condition* ∈ *monadM-qbs X* ⇒$_Q$ (*X* ⇒$_Q$ $\mathbb{B}_Q$) ⇒$_Q$ *monadM-qbs X*
by(*simp add*: *condition-def*)

lemma *condition-morphismP*:

assumes $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow \mathcal{P}(y$ *in qbs-l* $(s\ x).\ P\ x\ y) \neq 0$
and [*qbs*]: $s \in X \to_Q$ *monadP-qbs* $Y\ P \in X \to_Q Y \Rightarrow_Q$ *qbs-count-space UNIV*
shows $(\lambda x.$ *condition* $(s\ x)\ (P\ x)) \in X \to_Q$ *monadP-qbs* Y
proof(*rule qbs-morphism-cong'*[**where** $f{=}\lambda x.$ *normalize-qbs* (*density-qbs* $(s\ x)$ (*indicator* $\{y{\in}$*qbs-space* $Y.\ P\ x\ y\}$))])
fix x
assume x[*qbs*]:$x \in$ *qbs-space* X
have *density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\}) =$ *density-qbs* $(s\ x)$ $(\lambda y.$ *if* $P\ x\ y$ *then 1 else 0*)
by(*auto intro*!: *density-qbs-cong*[*OF qbs-space-monadPM*[*OF qbs-morphism-space*[*OF assms*(*2*) *x*]]] *indicator-qbs-morphism''*)
thus *normalize-qbs* (*density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\}$)) = *condition* $(s\ x)\ (P\ x)$
unfolding *condition-def query-def* **by** *simp*
next
show $(\lambda x.$ *normalize-qbs* (*density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\}$))) $\in X \to_Q$ *monadP-qbs* Y
proof(*rule normalize-qbs-morphismP*[*of* $\lambda x.$ *density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\}$)])
show $(\lambda x.$ *density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\})) \in X \to_Q$ *monadM-qbs* Y
using *qbs-morphism-monadPD*[*OF assms*(*2*)] **by** *simp*
next
fix x
assume x:$x \in$ *qbs-space* X
show *emeasure* (*qbs-l* (*density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\}$))) (*qbs-space* Y) $\neq 0$
emeasure (*qbs-l* (*density-qbs* $(s\ x)$ (*indicator* $\{y \in$ *qbs-space* $Y.\ P\ x\ y\}$))) (*qbs-space* Y) $\neq \infty$
using *assms*(*1*)[*OF x*] *qbs-l-monadP-le1*[*OF qbs-morphism-space*[*OF assms*(*2*) *x*]]
by(*auto simp add*: *qbs-l-density-qbs-indicator*[*OF qbs-space-monadPM*[*OF qbs-morphism-space*[*OF assms*(*2*) *x*]] *qbs-morphism-space*[*OF assms*(*3*) *x*]] *measure-def space-qbs-l-in*[*OF qbs-space-monadPM*[*OF qbs-morphism-space*[*OF assms*(*2*) *x*]]])
qed
qed

lemma *query-Bayes*:
assumes [*qbs*]: $s \in$ *qbs-space* (*monadP-qbs* X) *qbs-pred* $X\ P$ *qbs-pred* $X\ Q$
shows $\mathcal{P}(x$ *in condition s P.* $Q\ x) = \mathcal{P}(x$ *in s.* $Q\ x \mid P\ x)$ (**is** *?lhs* = *?pq*)
proof −
have X: *qbs-space* $X \neq \{\}$
using *assms*(*1*) **by**(*simp only*: *monadP-qbs-empty-iff*[*of X*]) *blast*
note s[*qbs*] = *qbs-space-monadPM*[*OF assms*(*1*)]
have *density-eq*: *density-qbs s* $(\lambda x.$ *if P x then 1 else 0*) = *density-qbs s* (*indicator* $\{x{\in}$*qbs-space* $X.\ P\ x\}$)
by(*auto intro*!: *density-qbs-cong*[*of - X*] *indicator-qbs-morphism''*)
consider *emeasure* (*qbs-l* (*density-qbs s* $(\lambda x.$ *if* $P\ x$ *then 1 else 0*))) (*qbs-space*

X) = *0* | *emeasure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*) ≠ *0* **by** *auto*
then show *?thesis*
proof *cases*
case *1*
have *2*:*normalize-qbs* (*density-qbs s* (λ*x*. *if P x then 1 else 0*)) = *qbs-null-measure X*
by(*rule normalize-qbs0*) (*auto simp*: *1*)
have *P*(ω *in qbs-l s*. *P* ω) = *measure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*)
by(*simp add*: *space-qbs-l-in*[*OF s*] *measure-def density-eq qbs-l-density-qbs-indicator*[*OF s*])
also have ... = *0*
by(*simp add*: *measure-def 1*)
finally show *?thesis*
by(*auto simp*: *condition-def query-def cond-prob-def 2 1 qbs-null-measure-null-measure*[*OF X*])
next
case *1*[*simp*]:*2*
from *rep-qbs-space-monadP*[*OF assms*(*1*)]
obtain α μ **where** *hs*: *s* = ⟦*X*, α, μ⟧$_{meas}$ *qbs-prob X* α μ **by** *auto*
then interpret *qp*: *qbs-prob X* α μ **by** *simp*
have [*measurable*]:*Measurable.pred* (*qbs-to-measure X*) *P Measurable.pred* (*qbs-to-measure X*) *Q*
using *assms*(*2*,*3*) **by**(*simp-all add*: *lr-adjunction-correspondence*)
have *2*[*simp*]: *emeasure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*) ≠ ⊤
by(*simp add*: *hs*(*1*) *qp.density-qbs qbs-meas.qbs-l*[*OF qp.density-qbs-meas*] *emeasure-distr emeasure-distr*[**where** *N*=*qbs-to-measure X*,*OF - sets.top*,*simplified space-L*] *emeasure-density*,*rule order.strict-implies-not-eq*[*OF order.strict-trans1*[*OF qp.nn-integral-le-const*[*of 1*] *ennreal-one-less-top*]]) *auto*
have *3*: *measure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*) > *0*
using *2 emeasure-eq-ennreal-measure zero-less-measure-iff* **by** *fastforce*
have *query s* (λ*x*. *if P x then 1 else 0*) = *density-qbs* (*density-qbs s* (λ*x*. *if P x then 1 else 0*)) (λ*x*. *1* / *emeasure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*))
unfolding *query-def* **by**(*rule normalize-qbs*) *auto*
also have ... = *density-qbs s* (λ*x*. (*if P x then 1 else 0*) ∗ (*1* / *emeasure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*)))
by(*simp add*: *density-qbs-density-qbs-eq*[*OF qbs-space-monadPM*[*OF assms*(*1*)]])
finally have *query*:*query s* (λ*x*. *if P x then 1 else 0*) = ... **.**
have *?lhs* = *measure* (*density* (*qbs-l s*) (λ*x*. (*if P x then 1 else 0*) ∗ (*1* / *emeasure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*)))) {*x* ∈ *space* (*qbs-l s*). *Q x*}
by(*simp add*: *condition-def query qbs-l-density-qbs*[*OF qbs-space-monadPM*[*OF assms*(*1*)]])
also have ... = *measure* (*density* μ (λ*x*. (*if P* (α *x*) *then 1 else 0*) ∗ (*1* / *emeasure* (*qbs-l* (*density-qbs s* (λ*x*. *if P x then 1 else 0*))) (*qbs-space X*)))) {*y*. α *y*

$\in$ *space* (*qbs-to-measure X*) $\wedge$ *Q* (α *y*)}
by(*simp add*: *hs*(*1*) *qp.qbs-l density-distr measure-def emeasure-distr*)
also have ... = *measure* (*density* μ (λx. *indicator* {*r*. *P* (α *r*)} *x* $*$ (*1* / *emeasure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*)))) {*y*. *Q* (α *y*)}
proof −
have [*simp*]:(*if P* (α *r*) *then 1 else 0*) = *indicator* {*r*. *P* (α *r*)} *r* **for** *r*
by *auto*
thus *?thesis* **by**(*simp add*: *space-L*)
qed
also have ... = *enn2real* (*1* / *emeasure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*)) $*$ *measure* μ {*r*. *P* (α *r*) $\wedge$ *Q* (α *r*)}
proof −
have *n-inf*: *1* / *emeasure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*) $\neq \infty$
using *1* **by**(*auto simp*: *ennreal-divide-eq-top-iff*)
show *?thesis*
by(*simp add*: *measure-density-times*[*OF - - n-inf*] *Collect-conj-eq*)
qed
also have ... = (*1* / *measure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*)) $*$ *qp.prob* {*r*. *P* (α *r*) $\wedge$ *Q* (α *r*)}
proof −
have *1* / *emeasure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*) = *ennreal* (*1* / *measure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*))
by(*auto simp add*: *emeasure-eq-ennreal-measure*[*OF 2*] *ennreal-1*[*symmetric*] *simp del*: *ennreal-1 intro*!: *divide-ennreal*) (*simp-all add*: *3*)
thus *?thesis* **by** *simp*
qed **also have** ... = *?pq*
proof −
have *qp*:$\mathcal{P}$(*x in s*. *Q x* $\wedge$ *P x*) = *qp.prob* {*r*. *P* (α *r*) $\wedge$ *Q* (α *r*)}
by(*auto simp*: *hs*(*1*) *qp.qbs-l measure-def emeasure-distr*, *simp add*: *space-L*) *meson*
note *sets* = *sets-qbs-l*[*OF qbs-space-monadPM*[*OF assms*(*1*)],*measurable-cong*]
have [*simp*]: *density* (*qbs-l s*) (λx. *if P x then 1 else 0*) = *density* (*qbs-l s*) (*indicator* {*x*$\in$*space* (*qbs-to-measure X*). *P x*})
by(*auto intro*!: *density-cong*) (*auto simp*: *indicator-def space-L sets-eq-imp-space-eq*[*OF sets*])
have *p*: $\mathcal{P}$(*x in s*. *P x*) = *measure* (*qbs-l* (*density-qbs s* (λx. *if P x then 1 else 0*))) (*qbs-space X*)
by(*auto simp*: *qbs-l-density-qbs*[*OF qbs-space-monadPM*[*OF assms*(*1*),*qbs*]]) (*auto simp*: *measure-restricted*[*of* {*x* $\in$ *space* (*qbs-to-measure X*). *P x*} *qbs-l s*,*simplified sets*,*OF - sets.top*,*simplified*,*simplified space-L*] *space-L sets-eq-imp-space-eq*[*OF sets*])
thus *?thesis*
by(*simp add*: *qp p cond-prob-def*)
qed
finally show *?thesis* **.**
qed
qed

lemma *qbs-pmf-cond-pmf*:
fixes $p :: 'a :: countable\ pmf$
assumes $set\text{-}pmf\ p \cap \{x.\ P\ x\} \neq \{\}$
shows $condition\ (qbs\text{-}pmf\ p)\ P = qbs\text{-}pmf\ (cond\text{-}pmf\ p\ \{x.\ P\ x\})$
proof(*rule inj-onD*[*OF qbs-l-inj*[*of count-space UNIV*]])
note *count-space-count-space-qbs-morphism*[*of P*,*qbs*]
show *g1*:$condition\ (qbs\text{-}pmf\ p)\ P \in qbs\text{-}space\ (monadM\text{-}qbs\ (count\text{-}space_Q\ UNIV))$
$qbs\text{-}pmf\ (cond\text{-}pmf\ p\ \{x.\ P\ x\}) \in qbs\text{-}space\ (monadM\text{-}qbs\ (count\text{-}space_Q\ UNIV))$
by *auto*
show $qbs\text{-}l\ (condition\ (qbs\text{-}pmf\ p)\ P) = qbs\text{-}l\ (qbs\text{-}pmf\ (cond\text{-}pmf\ p\ \{x.\ P\ x\}))$
proof(*safe intro*!: *measure-eqI-countable*)
fix *a*
have $condition\ (qbs\text{-}pmf\ p)\ P = normalize\text{-}qbs\ (density\text{-}qbs\ (qbs\text{-}pmf\ p)\ (\lambda x.\ if\ P\ x\ then\ 1\ else\ 0))$
by(*auto simp*: *condition-def query-def*)
also have $... = density\text{-}qbs\ (density\text{-}qbs\ (qbs\text{-}pmf\ p)\ (\lambda x.\ if\ P\ x\ then\ 1\ else\ 0))\ (\lambda x.\ 1\ /\ emeasure\ (qbs\text{-}l\ (density\text{-}qbs\ (qbs\text{-}pmf\ p)\ (\lambda x.\ if\ P\ x\ then\ 1\ else\ 0)))\ (qbs\text{-}space\ (count\text{-}space_Q\ UNIV)))$
proof −
have *1*:$(\int^+ x.\ ennreal\ (pmf\ p\ x) * (if\ P\ x\ then\ 1\ else\ 0)\ \partial count\text{-}space\ UNIV) = (\int^+ x{\in}\{x.\ P\ x\}.\ ennreal\ (pmf\ p\ x)\ \partial count\text{-}space\ UNIV)$
by(*auto intro*!: *nn-integral-cong*)
have $... > 0$
using *assms*(*1*) **by**(*force intro*!: *nn-integral-less*[*of* $\lambda x.\ 0$,*simplified*] *simp*: *AE-count-space set-pmf-eq*′ *indicator-def*)
hence *2*:$(\int^+ x{\in}\{x.\ P\ x\}.\ ennreal\ (pmf\ p\ x)\ \partial count\text{-}space\ UNIV) \neq 0$
by *auto*
have *3*:$(\int^+ x{\in}\{x.\ P\ x\}.\ ennreal\ (pmf\ p\ x)\ \partial count\text{-}space\ UNIV) \neq \top$
proof −
have $(\int^+ x{\in}\{x.\ P\ x\}.\ ennreal\ (pmf\ p\ x)\ \partial count\text{-}space\ UNIV) \leq (\int^+ x.\ ennreal\ (pmf\ p\ x)\ \partial count\text{-}space\ UNIV)$
by(*auto intro*!: *nn-integral-mono simp*: *indicator-def*)
also have $... = 1$
by (*simp add*: *nn-integral-pmf-eq-1*)
finally show *?thesis*
using *ennreal-one-neq-top neq-top-trans* **by** *fastforce*
qed
show *?thesis*
by(*rule normalize-qbs*) (*auto simp*: *qbs-l-density-qbs*[*of - count-space UNIV*] *emeasure-density nn-integral-measure-pmf 1 2 3*)
qed
also have $... = density\text{-}qbs\ (qbs\text{-}pmf\ p)\ (\lambda x.\ (if\ P\ x\ then\ 1\ else\ 0) * (1\ /\ (\int^+ x.\ ennreal\ (pmf\ p\ x) * (if\ P\ x\ then\ 1\ else\ 0)\ \partial count\text{-}space\ UNIV)))$
by(*simp add*: *density-qbs-density-qbs-eq*[*of - count-space UNIV*] *qbs-l-density-qbs*[*of - count-space UNIV*] *emeasure-density nn-integral-measure-pmf*)
also have $... = density\text{-}qbs\ (qbs\text{-}pmf\ p)\ (\lambda x.\ (if\ P\ x\ then\ 1\ else\ 0) * (1\ /\ (emeasure\ (measure\text{-}pmf\ p)\ (Collect\ P))))$
proof −
have [*simp*]: $(\int^+ x.\ ennreal\ (pmf\ p\ x) * (if\ P\ x\ then\ 1\ else\ 0)\ \partial count\text{-}space$

UNIV) = *emeasure* (*measure-pmf p*) (*Collect P*) (**is** *?l* = *?r*)
proof −
have *?l* = (∫⁺ *x*. *ennreal* (*pmf p x*) ∗ (*if P x then 1 else 0*) ∂*count-space* {*x*. *P x*})
by(*rule nn-integral-count-space-eq*) *auto*
also have ... = *?r*
by(*auto simp*: *nn-integral-pmf*[*symmetric*] *intro*!: *nn-integral-cong*)
finally show *?thesis* **.**
qed
show *?thesis* **by** *simp*
qed
finally show *emeasure* (*qbs-l* (*condition* (*qbs-pmf p*) *P*)) {*a*} = *emeasure* (*qbs-l* (*qbs-pmf* (*cond-pmf p* {*x*. *P x*}))) {*a*}
by(*simp add*: *ennreal-divide-times qbs-l-density-qbs*[*of - count-space UNIV*] *emeasure-density cond-pmf.rep-eq*[*OF assms*(*1*)])
qed(*auto simp*: *sets-qbs-l*[*OF g1*(*1*)])
qed

5.2.1 `twoUs`

Example from Section 2 in [3].

definition *Uniform* ≡ (λ*a b*::*real*. *uniform-qbs lborel-qbs* {*a*<..<*b*})

lemma *Uniform-qbs*[*qbs*]: *Uniform* ∈ $\mathbb{R}_Q$ ⇒$_Q$ $\mathbb{R}_Q$ ⇒$_Q$ *monadM-qbs* $\mathbb{R}_Q$
unfolding *Uniform-def* **by** (*rule interval-uniform-qbs*)

definition *twoUs* :: (*real* × *real*) *qbs-measure* **where**
twoUs ≡ *do* {
let u1 = *Uniform 0 1*;
let u2 = *Uniform 0 1*;
let y = *u1* ⨂$_{Qmes}$ *u2*;
condition y (λ(*x*,*y*). *x* < *0.5* ∨ *y* > *0.5*)
}

lemma *twoUs-qbs*: *twoUs* ∈ *monadM-qbs* ($\mathbb{R}_Q$ ⨂$_Q$ $\mathbb{R}_Q$)
by(*simp add*: *twoUs-def*)

interpretation *rr*: *standard-borel-ne borel* ⨂$_M$ *borel* :: (*real* × *real*) *measure*
by(*simp add*: *borel-prod*)

lemma *qbs-l-Uniform*[*simp*]: *a* < *b* ⟹ *qbs-l* (*Uniform a b*) = *uniform-measure lborel* {*a*<..<*b*}
by(*simp add*: *standard-borel-ne.qbs-l-uniform-qbs*[*of borel lborel-qbs*] *Uniform-def*)

lemma *Uniform-qbsP*:
assumes [*arith*]: *a* < *b*
shows *Uniform a b* ∈ *monadP-qbs* $\mathbb{R}_Q$
by(*auto simp*: *monadP-qbs-def sub-qbs-space intro*!: *prob-space-uniform-measure*)

interpretation *UniformP-pair*: *pair-prob-space uniform-measure lborel* {*0*<..<*1*::*real*} *uniform-measure lborel* {*0*<..<*1*::*real*}
by(*auto simp*: *pair-prob-space-def pair-sigma-finite-def intro*!: *prob-space-imp-sigma-finite prob-space-uniform-measure*)

lemma *qbs-l-Uniform-pair*: $a < b \Longrightarrow$ *qbs-l* (*Uniform a b* $\bigotimes_{Qmes}$ *Uniform a b*) = *uniform-measure lborel* {*a*<..<*b*} $\bigotimes_M$ *uniform-measure lborel* {*a*<..<*b*}
by(*auto intro*!: *qbs-l-qbs-pair-measure*[*of borel borel*] *standard-borel-ne.standard-borel simp*: *qbs-l-Uniform*[*symmetric*] *simp del*: *qbs-l-Uniform*)

lemma *Uniform-pair-qbs*[*qbs*]:
assumes $a < b$
shows *Uniform a b* $\bigotimes_{Qmes}$ *Uniform a b* $\in$ *qbs-space* (*monadP-qbs* ($\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$))
proof −
note [*qbs*] = *qbs-pair-measure-morphismP Uniform-qbsP*[*OF assms*]
show *?thesis*
by *simp*
qed

lemma *twoUs-prob1*: $\mathcal{P}$(*z in Uniform 0 1* $\bigotimes_{Qmes}$ *Uniform 0 1*. *fst z* < *0.5* ∨ *snd z* > *0.5*) = *3 / 4*
proof −
have [*simp*]:{*z* ∈ *space* (*uniform-measure lborel* {*0*<..<*1*::*real*} $\bigotimes_M$ *uniform-measure lborel* {*0*<..<*1*::*real*}). *fst z* ∗ *2* < *1* ∨ *1* < *snd z* ∗ *2*} = *UNIV* × {*1/2*<..} ∪ {..<*1/2*} × *UNIV*
by(*auto simp*: *space-pair-measure*)
have *1*:*UniformP-pair.prob* (*UNIV* × {*1 / 2*<..}) = *1 / 2*
proof −
have [*simp*]:{*0*<..<*1*} ∩ {*1 / 2*<..} = {*1/2*<..<*1*::*real*} **by** *auto*
thus *?thesis*
by(*auto simp*: *UniformP-pair.M1.measure-times*)
qed
have *2*:*UniformP-pair.prob* ({..<*1 / 2*} × *UNIV* − *UNIV* × {*1 / 2*<..}) = *1 / 4*
proof −
have [*simp*]: {..<*1/2*::*real*} × *UNIV* − *UNIV* × {*1/2*::*real*<..} = {..<*1/2*} × {..*1/2*} {*0*<..<*1*} ∩ {..<*1/2*} = {*0*<..<*1/2*::*real*} {*0*<..<*1*} ∩ {..*1/2*::*real*} = {*0*<..*1/2*}
by *auto*
show *?thesis*
by(*auto simp*: *UniformP-pair.M1.measure-times*)
qed
show *?thesis*
by(*auto simp*: *qbs-l-Uniform-pair UniformP-pair.P.finite-measure-Union′ 1 2*)
qed

lemma *twoUs-prob2*:$\mathcal{P}$(*z in Uniform 0 1* $\bigotimes_{Qmes}$ *Uniform 0 1*. *1/2* < *fst z* ∧ (*fst z* < *1/2* ∨ *snd z* > *1/2*)) = *1 / 4*

proof −
have [*simp*]:{*z* ∈ *space* (*uniform-measure lborel* {*0*<..<*1*::*real*} ⨂$_M$ *uniform-measure lborel* {*0*<..<*1*::*real*}). *1* < *fst z* ∗ *2* ∧ (*fst z* ∗ *2* < *1* ∨ *1* < *snd z* ∗ *2*)} = {*1/2*<..} × {*1/2*<..}
by(*auto simp*: *space-pair-measure*)
have [*simp*]: {*0*<..<*1*::*real*} ∩ {*1/2*<..} = {*1/2*<..<*1*} **by** *auto*
show *?thesis*
by(*auto simp*: *qbs-l-Uniform-pair UniformP-pair.M1.measure-times*)
qed

lemma *twoUs-qbs-prob*: *twoUs* ∈ *qbs-space* (*monadP-qbs* (ℝ$_Q$ ⨂$_Q$ ℝ$_Q$))
proof −
have $\mathcal{P}$(*z in Uniform 0 1* ⨂$_{Qmes}$ *Uniform 0 1*. *fst z* < *0.5* ∨ *snd z* > *0.5*) ≠ *0*
unfolding *twoUs-prob1* **by** *simp*
note *qbs-morphism-space*[*OF condition-morphismP*[*of qbs-borel* λ*x*. *Uniform 0 1* ⨂$_{Qmes}$ *Uniform 0 1* λ*x z*. *fst z* < *0.5* ∨ *snd z* > *0.5* ℝ$_Q$ ⨂$_Q$ ℝ$_Q$,*OF this*],*simplified*,*qbs*]
note *Uniform-pair-qbs*[*of 0 1*,*simplified*,*qbs*]
show *?thesis*
by(*simp add*: *twoUs-def split-beta′*)
qed

lemma $\mathcal{P}$((*x*,*y*) *in twoUs*. *1/2* < *x*) = *1* / *3*
proof −
have $\mathcal{P}$((*x*,*y*) *in twoUs*. *1/2* < *x*) = $\mathcal{P}$(*z in twoUs*. *1/2* < *fst z*)
by (*simp add*: *split-beta′*)
also have ... = $\mathcal{P}$(*z in Uniform 0 1* ⨂$_{Qmes}$ *Uniform 0 1*. *1/2* < *fst z* | *fst z* < *0.5* ∨ *snd z* > *0.5*)
by(*simp add*: *twoUs-def split-beta′*,*rule query-Bayes*[*OF Uniform-pair-qbs*[*of 0 1*,*simplified*,*qbs*]]) *auto*
also have ... = $\mathcal{P}$(*z in Uniform 0 1* ⨂$_{Qmes}$ *Uniform 0 1*. *1/2* < *fst z* ∧ (*fst z* < *1/2* ∨ *snd z* > *1/2*)) / $\mathcal{P}$(*z in Uniform 0 1* ⨂$_{Qmes}$ *Uniform 0 1*. *fst z* < *0.5* ∨ *snd z* > *0.5*)
by(*simp add*: *cond-prob-def*)
also have ... = *1* / *3*
by(*simp only*: *twoUs-prob2 twoUs-prob1*) *simp*
finally show *?thesis* **.**
qed

5.2.2 Two Dice

Example from Adrian [2, Sect. 2.3].

abbreviation *die* ≡ *qbs-pmf* (*pmf-of-set* {*Suc 0..6*})

lemma *die-qbs*[*qbs*]: *die* ∈ *monadM-qbs* ℕ$_Q$
by *simp*

definition *two-dice* :: *nat qbs-measure* **where**
two-dice ≡ *do* {

let die1 = die;
let die2 = die;
let twodice = die1 $\bigotimes_{Qmes}$ *die2*;
(*x*,*y*) ← *condition twodice*
(λ(*x*,*y*). *x* = *4* ∨ *y* = *4*);
return-qbs $\mathbb{N}_Q$ (*x* + *y*)
}

lemma *two-dice-qbs*: *two-dice* ∈ *monadM-qbs* $\mathbb{N}_Q$
by(*simp add*: *two-dice-def*)

lemma *prob-die2*: $\mathcal{P}$(*x in qbs-l* (*die* $\bigotimes_{Qmes}$ *die*). *P x*) = *real* (*card* ({*x*. *P x*} ∩ ({*1..6*} × {*1..6*}))) / *36* (**is** *?P* = *?rhs*)
proof −
have *?P* = *measure-pmf.prob* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {*x*. *P x*}
by(*auto simp*: *qbs-pair-pmf*)
also have ... = *measure-pmf.prob* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) ({*x*. *P x*} ∩ *set-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})))
by(*rule measure-Int-set-pmf*[*symmetric*])
also have ... = *measure-pmf.prob* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) ({*x*. *P x*} ∩ ({*Suc 0..6*} × {*Suc 0..6*}))
by *simp*
also have ... = (∑ *z*∈{*x*. *P x*} ∩ ({*Suc 0..6*} × {*Suc 0..6*}). *pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) *z*)
by(*simp add*: *measure-measure-pmf-finite*)
also have ... = (∑ *z*∈{*x*. *P x*} ∩ ({*Suc 0..6*} × {*Suc 0..6*}). *1 / 36*)
by(*rule Finite-Cartesian-Product.sum-cong-aux*) (*auto simp*: *pmf-pair*)
also have ... = *?rhs*
by *auto*
finally show *?thesis* **.**
qed

lemma *dice-prob1*: $\mathcal{P}$(*z in qbs-l* (*die* $\bigotimes_{Qmes}$ *die*). *fst z = 4* ∨ *snd z = 4*) = *11 / 36*
proof −
have *1*:*Restr* {*z*. *fst z = 4* ∨ *snd z = 4*} {*Suc 0..6*::*nat*} = {*Suc 0..Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))} × {*Suc* (*Suc* (*Suc* (*Suc 0*)))} ∪ {*Suc* (*Suc* (*Suc* (*Suc 0*)))} × {*Suc 0..*(*Suc* (*Suc* (*Suc 0*)))} ∪ {*Suc* (*Suc* (*Suc* (*Suc 0*)))} × {*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))..*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))}
by *fastforce*
have *card* ... = *card* ({*Suc 0..Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))} × {*Suc* (*Suc* (*Suc* (*Suc 0*)))} ∪ {*Suc* (*Suc* (*Suc* (*Suc 0*)))} × {*Suc 0..*(*Suc* (*Suc* (*Suc 0*)))}) + *card* ({*Suc* (*Suc* (*Suc* (*Suc 0*)))} × {*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))..*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))})
by(*rule card-Un-disjnt*) (*auto simp*: *disjnt-def*)
also have ... = *card* ({*Suc 0..Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))} × {*Suc* (*Suc* (*Suc* (*Suc 0*)))}) + *card* ({*Suc* (*Suc* (*Suc* (*Suc 0*)))} × {*Suc 0..*(*Suc* (*Suc* (*Suc*

0))))}) + *card* ({*Suc* (*Suc* (*Suc* (*Suc* *0*)))} × {*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))..*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))})
proof −
have *card* ({*Suc* *0*..*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))} × {*Suc* (*Suc* (*Suc* (*Suc* *0*)))} ∪ {*Suc* (*Suc* (*Suc* (*Suc* *0*)))} × {*Suc* *0*..(*Suc* (*Suc* (*Suc* *0*)))}) = *card* ({*Suc* *0*..*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))} × {*Suc* (*Suc* (*Suc* (*Suc* *0*)))}) + *card* ({*Suc* (*Suc* (*Suc* (*Suc* *0*)))} × {*Suc* *0*..(*Suc* (*Suc* (*Suc* *0*)))})
by(*rule card-Un-disjnt*) (*auto simp*: *disjnt-def*)
thus *?thesis* **by** *simp*
qed
also have ... = *11* **by** *auto*
finally show *?thesis*
by(*auto simp*: *prob-die2 1*)
qed

lemma *dice-program-prob*:$\mathcal{P}$(*x in two-dice*. *P x*) = *2* ∗ (∑ *n*∈{*5*,*6*,*7*,*9*,*10*}. *of-bool* (*P n*) / *11*) + *of-bool* (*P 8*) / *11* (**is** *?P* = *?rp*)
proof −
have *0*: (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*}) = {*5*,*6*,*7*,*8*,*9*,*10*}
proof *safe*
show *5* ∈ (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*})
by(*auto intro*!: *bexI*[**where** *x*=(*1*,*4*)])
show *6* ∈ (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*})
by(*auto intro*!: *bexI*[**where** *x*=(*2*,*4*)])
show *7* ∈ (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*})
by(*auto intro*!: *bexI*[**where** *x*=(*3*,*4*)])
show *8* ∈ (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*})
by(*auto intro*!: *bexI*[**where** *x*=(*4*,*4*)])
show *9* ∈ (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*})
by(*auto intro*!: *bexI*[**where** *x*=(*5*,*4*)])
show *10* ∈ (⋃ *x*∈{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*}. {*fst x* + *snd x*})
by(*auto intro*!: *bexI*[**where** *x*=(*6*,*4*)])
qed *auto*

have *1*:{*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {*x*. *fst x* = *4* ∨ *snd x* = *4*} ≠ {}
proof −
have (*1*,*4*) ∈ {*Suc* *0*..*6*} × {*Suc* *0*..*6*} ∩ {*x*. *fst x* = *4* ∨ *snd x* = *4*}
by *auto*
thus *?thesis* **by** *blast*
qed
hence *2*: *set-pmf* (*pair-pmf* (*pmf-of-set* {*Suc* *0*..*6*}) (*pmf-of-set* {*Suc* *0*..*6*})) ∩ {(*x*, *y*). *x* = *4* ∨ *y* = *4*} ≠ {}

by(*auto simp*: *split-beta′*)
have *ceq*:*condition* (*die* $\bigotimes_{Qmes}$ *die*) (λ(*x*,*y*). *x* = *4* ∨ *y* = *4*) = *qbs-pmf* (*cond-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*,*y*). *x* = *4* ∨ *y* = *4*})
by(*auto simp*: *split-beta′ qbs-pair-pmf 1 intro*!: *qbs-pmf-cond-pmf*)
have *two-dice* = *condition* (*die* $\bigotimes_{Qmes}$ *die*) (λ(*x*,*y*). *x* = *4* ∨ *y* = *4*) ⤜ (λ(*x*,*y*). *return-qbs* $\mathbb{N}_Q$ (*x* + *y*))
by(*simp add*: *two-dice-def*)
also have ... = *qbs-pmf* (*cond-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*,*y*). *x* = *4* ∨ *y* = *4*}) ⤜ (λ*z*. *qbs-pmf* (*return-pmf* (*fst z* + *snd z*)))
by(*simp add*: *ceq*) (*simp add*: *qbs-pmf-return-pmf split-beta′*)
also have ... = *qbs-pmf* (*cond-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*,*y*). *x* = *4* ∨ *y* = *4*} ⤜ (λ*z*. *return-pmf* (*fst z* + *snd z*)))
by(*rule qbs-pmf-bind-pmf*[*symmetric*])
finally have *two-dice-eq*:*two-dice* = *qbs-pmf* (*cond-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*,*y*). *x* = *4* ∨ *y* = *4*} ⤜ (λ*z*. *return-pmf* (*fst z* + *snd z*))) **.**

have *3*:*measure-pmf.prob* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*, *y*). *x* = *4* ∨ *y* = *4*} = *11* / *36*
using *dice-prob1* **by**(*auto simp*: *split-beta′ qbs-pair-pmf*)

have *?P* = *measure-pmf.prob* (*cond-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*, *y*). *x* = *4* ∨ *y* = *4*} ⤜ (λ*z*. *return-pmf* (*fst z* + *snd z*))) {*x*. *P x*} (**is** - = *measure-pmf.prob ?bind* -)
by(*simp add*: *two-dice-eq*)
also have ... = *measure-pmf.prob ?bind* ({*x*. *P x*} ∩ *set-pmf ?bind*)
by(*rule measure-Int-set-pmf*[*symmetric*])
also have ... = *sum* (*pmf ?bind*) ({*x*. *P x*} ∩ *set-pmf ?bind*)
by(*rule measure-measure-pmf-finite*) (*auto simp*: *set-cond-pmf*[*OF 2*])
also have ... = *sum* (*pmf ?bind*) ({*x*. *P x*} ∩ {*5*, *6*, *7*, *8*, *9*, *10*})
by(*auto simp*: *set-cond-pmf*[*OF 2*] *0*)
also have ... = (∑ *n*∈{*n*. *P n*}∩{*5*, *6*, *7*, *8*, *9*, *10*}. *measure-pmf.expectation* (*cond-pmf* (*pair-pmf* (*pmf-of-set* {*Suc 0..6*}) (*pmf-of-set* {*Suc 0..6*})) {(*x*, *y*). *x* = *4* ∨ *y* = *4*}) (λ*x*. *indicat-real* {*n*} (*fst x* + *snd x*))) (**is** - = (∑ -∈-. *measure-pmf.expectation ?cond* -))
by(*simp add*: *pmf-bind*)
also have ... = (∑ *n*∈{*n*. *P n*}∩{*5*, *6*, *7*, *8*, *9*, *10*}. (∑ *m*∈{(*1*,*4*),(*2*,*4*),(*3*,*4*),(*4*,*4*),(*5*,*4*),(*6*,*4*),(*4*,*1*),(*4*,*2*),
indicat-real {*n*} (*fst m* + *snd m*) ∗ *pmf ?cond m*))
proof(*intro Finite-Cartesian-Product.sum-cong-aux integral-measure-pmf-real*)
fix *n m*
assume *h*:*n* ∈ {*n*. *P n*}∩{*5*, *6*, *7*, *8*, *9*, *10*} *m* ∈ *set-pmf ?cond indicat-real* {*n*} (*fst m* + *snd m*) ≠ *0*
then have *nm*:*fst m* + *snd m* = *n*
by(*auto simp*: *indicator-def*)
have *m*: *fst m* ≠ *0 snd m* ≠ *0 fst m* = *4* ∨ *snd m* = *4*
using *h*(*2*) **by**(*auto simp*: *set-cond-pmf*[*OF 2*])
show *m* ∈ {(*1*, *4*), (*2*, *4*), (*3*, *4*), (*4*,*4*), (*5*, *4*), (*6*, *4*), (*4*, *1*), (*4*, *2*), (*4*, *3*),

```
(4, 5), (4, 6)}
      using h(1) nm m by(auto, metis prod.collapse)+
  qed simp
  also have ... = (∑ n∈{n. P n}∩{5, 6, 7, 8, 9, 10}. (∑ m∈{(1,4),(2,4),(3,4),(4,4),(5,4),(6,4),(4,1),(4,2),(
indicat-real {n} (fst m + snd m) * 1 / 11))
  proof(rule Finite-Cartesian-Product.sum-cong-aux[OF Finite-Cartesian-Product.sum-cong-aux])
    fix n m
    assume h:n ∈ {n. P n}∩{5, 6, 7, 8, 9, 10} m ∈ {(1,4),(2,4),(3,4),(4,4),(5,4),(6,4),(4,1),(4,2),(4,3),(4,
    have pmf ?cond m = 1 / 11
      using h(2) by(auto simp add: pmf-cond[OF 2] 3 pmf-pair)
    thus  indicat-real {n} (fst m + snd m) * pmf ?cond m = indicat-real {n} (fst
m + snd m) * 1 / 11
      by simp
  qed
  also have ... = ?rp
    by fastforce
  finally show ?thesis .
qed

corollary
 P(x in two-dice. x = 5)  = 2 / 11
 P(x in two-dice. x = 6)  = 2 / 11
 P(x in two-dice. x = 7)  = 2 / 11
 P(x in two-dice. x = 8)  = 1 / 11
 P(x in two-dice. x = 9)  = 2 / 11
 P(x in two-dice. x = 10) = 2 / 11

  unfolding dice-program-prob by simp-all
```

5.2.3 Gaussian Mean Learning

Example from Sato et al. Section 8. 2 in [3].

```
definition Gauss ≡ (λμ σ. density-qbs lborel_Q (normal-density μ σ))

lemma Gauss-qbs[qbs]: Gauss ∈ ℝ_Q ⇒_Q ℝ_Q ⇒_Q monadM-qbs ℝ_Q
  by(simp add: Gauss-def)

primrec GaussLearn' :: [real, real qbs-measure, real list]
                                  ⇒ real qbs-measure where
  GaussLearn' - p [] = p
| GaussLearn' σ p (y#ls) = query (GaussLearn' σ p ls)
                                 (normal-density y σ)

lemma GaussLearn'-qbs[qbs]:GaussLearn' ∈ ℝ_Q ⇒_Q monadM-qbs ℝ_Q ⇒_Q list-qbs
ℝ_Q ⇒_Q monadM-qbs ℝ_Q
  by(simp add: GaussLearn'-def)

context
  fixes σ :: real
```

assumes [*arith*]: $\sigma > 0$
begin

abbreviation *GaussLearn* $\equiv$ *GaussLearn'* σ

lemma *GaussLearn-qbs*[*qbs*]: *GaussLearn* $\in$ *qbs-space* (*monadM-qbs* $\mathbb{R}_Q \Rightarrow_Q$ *list-qbs* $\mathbb{R}_Q \Rightarrow_Q$ *monadM-qbs* $\mathbb{R}_Q$)
by *simp*

definition *Total* :: *real list* $\Rightarrow$ *real* **where** *Total* = (λ*l*. *foldr* (+) *l* 0)

lemma *Total-simp*: *Total* [] = 0 *Total* (*y*#*ls*) = *y* + *Total ls*
by(*simp-all add*: *Total-def*)

lemma *Total-qbs*[*qbs*]: *Total* $\in$ *list-qbs* $\mathbb{R}_Q \to_Q \mathbb{R}_Q$
by(*simp add*: *Total-def*)

lemma *GaussLearn-Total*:
assumes [*arith*]: $\xi > 0$ *n* = *length L*
shows *GaussLearn* (*Gauss* δ ξ) *L* = *Gauss* ((*Total L*$*\xi^2$+δ*σ^2)/(*n*$*\xi^2$+σ^2)) (*sqrt* ((ξ^2*σ^2)/(*n*$*\xi^2$+σ^2)))
using *assms*(*2*)
proof(*induction L arbitrary*: *n*)
case *Nil*
then show *?case*
by(*simp add*: *Total-def*)
next
case *ih*:(*Cons a L*)
then obtain *n'* **where** *n'*:*n* = *Suc n' n'* = *length L*
by *auto*
have *1*:$\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$) > 0
by(*auto intro*!: *divide-pos-pos add-nonneg-pos*)
have *sigma*:(*sqrt* ($\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$)) * σ / *sqrt* ($\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$) + σ^2)) = (*sqrt* ($\xi^2 * \sigma^2$ / (*real n* * $\xi^2 + \sigma^2$)))
proof(*rule power2-eq-imp-eq*)
show (*sqrt* ($\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$)) * σ / *sqrt* ($\xi^2 * \sigma^2$ / (*real n'* * ξ^2 + σ^2) + σ^2))^2 = (*sqrt* ($\xi^2 * \sigma^2$ / (*real n* * $\xi^2 + \sigma^2$)))^2 (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$)) * (σ^2 / ($\xi^2 * \sigma^2$ / (*real n'* * ξ^2 + σ^2) + σ^2))
by (*simp add*: *power-divide power-mult-distrib*)
also have ... = $\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$) * (σ^2 / ((ξ^2 / (*real n'* * ξ^2 + σ^2) + *1*) * σ^2))
by (*simp add*: *distrib-left mult.commute*)
also have ... = $\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$) * (*1* / (ξ^2 / (*real n'* * $\xi^2 + \sigma^2$) + *1*))
by *simp*
also have ... = $\xi^2 * \sigma^2$ / (*real n'* * $\xi^2 + \sigma^2$) * (*1* / ((ξ^2 + (*real n'* * ξ^2 +

$\sigma^2)) / (real\ n' * \xi^2 + \sigma^2)))$
by(*simp only*: *add-divide-distrib*[*of* ξ^2]) *auto*
also have ... = $\xi^2 * \sigma^2 / (real\ n' * \xi^2 + \sigma^2) * ((real\ n' * \xi^2 + \sigma^2) / (\xi^2 + (real\ n' * \xi^2 + \sigma^2)))$
by *simp*
also have ... = $\xi^2 * \sigma^2 / (\xi^2 + (real\ n' * \xi^2 + \sigma^2))$
using *1* **by** *force*
also have ... = *?rhs*
by(*simp add*: *n'*(*1*) *distrib-right*)
finally show *?thesis* **.**
qed
qed *simp-all*
have *mu*: $((Total\ L * \xi^2 + \delta * \sigma^2) * \sigma^2 / (real\ n' * \xi^2 + \sigma^2) + a * (\xi^2 * \sigma^2) / (real\ n' * \xi^2 + \sigma^2)) / (\xi^2 * \sigma^2 / (real\ n' * \xi^2 + \sigma^2) + \sigma^2) = ((a + Total\ L) * \xi^2 + \delta * \sigma^2) / (real\ n * \xi^2 + \sigma^2)$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(((Total\ L * \xi^2 + \delta * \sigma^2) * \sigma^2 + a * (\xi^2 * \sigma^2))/ (real\ n' * \xi^2 + \sigma^2)) / (\xi^2 * \sigma^2 / (real\ n' * \xi^2 + \sigma^2) + \sigma^2)$
by(*simp add*: *add-divide-distrib*)
also have ... = $(((Total\ L * \xi^2 + \delta * \sigma^2) + a * \xi^2) * \sigma^2 / (real\ n' * \xi^2 + \sigma^2)) / (\xi^2 * \sigma^2 / (real\ n' * \xi^2 + \sigma^2) + \sigma^2)$
by (*simp add*: *distrib-left mult.commute*)
also have ... = $(((Total\ L * \xi^2 + \delta * \sigma^2) + a * \xi^2) * \sigma^2 / (real\ n' * \xi^2 + \sigma^2)) / ((\xi^2 * \sigma^2 + (real\ n' * \xi^2 + \sigma^2) * \sigma^2) / (real\ n' * \xi^2 + \sigma^2))$
by (*simp add*: *add-divide-distrib*)
also have ... = $(((Total\ L * \xi^2 + \delta * \sigma^2) + a * \xi^2) * \sigma^2) / (\xi^2 * \sigma^2 + (real\ n' * \xi^2 + \sigma^2) * \sigma^2)$
using *1* **by** *auto*
also have ... = $(((Total\ L * \xi^2 + \delta * \sigma^2) + a * \xi^2) * \sigma^2) / ((\xi^2 + (real\ n' * \xi^2 + \sigma^2)) * \sigma^2)$
by(*simp only*: *distrib-right*)
also have ... = $((Total\ L * \xi^2 + \delta * \sigma^2) + a * \xi^2) / (\xi^2 + (real\ n' * \xi^2 + \sigma^2))$
by *simp*
also have ... = $((Total\ L * \xi^2 + \delta * \sigma^2) + a * \xi^2) / (real\ n * \xi^2 + \sigma^2)$
by(*simp add*: *n'*(*1*) *distrib-right*)
also have ... = *?rhs*
by (*simp add*: *distrib-right*)
finally show *?thesis* **.**
qed
show *?case*
by(*simp add*: *ih*(*1*)[*OF n'*(*2*)]) (*simp add*: *query-def qbs-normal-posterior*[*OF real-sqrt-gt-zero*[*OF 1*]] *Gauss-def Total-simp sigma mu*)
qed

lemma *GaussLearn-KL-divergence-lem1*:
fixes *a* :: *real*
assumes [*arith*]: *a* > *0 b* > *0 c* > *0 d* > *0*
shows $(\lambda n.\ ln\ ((b * (n * d + c)) / (d * (n * b + a)))) \longrightarrow 0$
proof −

have (λn::*nat*. *ln* ((*b* ∗ (*Suc n* ∗ *d* + *c*)) / (*d* ∗ (*Suc n* ∗ *b* + *a*)))) = (λn. *ln* ((*b* ∗ (*d* + *c* / *Suc n*)) / (*d* ∗ (*b* + *a* / *Suc n*))))
proof
fix *n*
show *ln* (*b* ∗ (*real* (*Suc n*) ∗ *d* + *c*) / (*d* ∗ (*real* (*Suc n*) ∗ *b* + *a*))) = *ln* (*b* ∗ (*d* + *c* / *real* (*Suc n*)) / (*d* ∗ (*b* + *a* / *real* (*Suc n*)))) (**is** *ln ?l* = *ln ?r*)
proof −
have *?l* = *b* ∗ (*d* + *c* / *real* (*Suc n*)) / (*d* ∗ (*b* + *a* / *real* (*Suc n*))) ∗ (*Suc n* / *Suc n*)
unfolding *times-divide-times-eq distrib-left distrib-right* **by** (*simp add*: *mult.assoc mult.commute*)
also have ... = *?r* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed
also have ... ⟶ *0*
apply(*rule tendsto-eq-intros*(*33*)[*of* - *1*])
apply(*rule Topological-Spaces.tendsto-eq-intros*(*25*)[*of* - *b* ∗ *d* - - *b* ∗ *d*,*OF LIMSEQ-Suc*[*OF Topological-Spaces.tendsto-eq-intros*(*18*)[*of* - *b* - - *d*]] *LIMSEQ-Suc*[*OF Topological-Spaces.tendsto-eq-intros*(*18*)[*of* - *d* - - *b*]]])
apply(*intro Topological-Spaces.tendsto-eq-intros* | *auto*)+
done
finally show *?thesis*
by(*rule LIMSEQ-imp-Suc*)
qed

lemma *GaussLearn-KL-divergence-lem1′*:
fixes *b* :: *real*
assumes [*arith*]: *b* > *0 d* > *0 s* > *0*
shows (λn. *ln* (*sqrt* (*b*^{2} ∗ *s*^{2} / (*real n* ∗ *b*^{2} + *s*^{2})) / *sqrt* (*d*^{2} ∗ *s*^{2} / (*real n* ∗ *d*^{2} + *s*^{2})))) ⟶ *0* (**is** *?f* ⟶ *0*)
proof −
have *?f* = (λn. *ln* (*sqrt* ((*b*^{2} ∗ (*n* ∗ *d*^{2} + *s*^{2}))/ (*d*^{2} ∗ (*n* ∗ *b*^{2} + *s*^{2})))))
by(*simp add*: *real-sqrt-divide real-sqrt-mult mult.commute*)
also have ... = (λn. *ln* ((*b*^{2} ∗ (*n* ∗ *d*^{2} + *s*^{2}) / (*d*^{2} ∗ (*n* ∗ *b*^{2} + *s*^{2})))) / *2*)
by (*standard*, *rule ln-sqrt*) (*auto intro*!: *divide-pos-pos mult-pos-pos add-nonneg-pos*)
also have ... ⟶ *0*
using *GaussLearn-KL-divergence-lem1* **by** *auto*
finally show *?thesis* **.**
qed

lemma *GaussLearn-KL-divergence-lem2*:
fixes *s* :: *real*
assumes [*arith*]: *s* > *0 b* > *0 d* > *0*
shows (λn. ((*d* ∗ *s*) / (*n* ∗ *d* + *s*)) / (*2* ∗ ((*b* ∗ *s*) / (*n* ∗ *b* + *s*)))) ⟶ *1* / *2*
proof −
have (λn::*nat*. ((*d* ∗ *s*) / (*Suc n* ∗ *d* + *s*)) / (*2* ∗ ((*b* ∗ *s*) / (*Suc n* ∗ *b* + *s*)))) = (λn. (*d* ∗ *b* + *d* ∗ *s* / *Suc n*) / (*2* ∗ *b* ∗ *d* + *2* ∗ *b* ∗ *s* / *Suc n*))
proof

fix n
show $d * s / (real\ (Suc\ n) * d + s) / (2 * (b * s / (real\ (Suc\ n) * b + s))) = (d * b + d * s / real\ (Suc\ n)) / (2 * b * d + 2 * b * s / real\ (Suc\ n))$ (**is** $?l = ?r$)
proof −
have $?l = d * (Suc\ n * b + s) / ((2 * b) * (Suc\ n * d + s))$
by(*simp add*: *divide-divide-times-eq*)
also have $... = d * (b + s / Suc\ n) / ((2 * b) * (d + s / Suc\ n)) * (Suc\ n / Suc\ n)$
proof −
have *1*:$(2 * b * d * real\ (Suc\ n) + 2 * b * (s / real\ (Suc\ n)) * real\ (Suc\ n)) = (2 * b) * (Suc\ n * d + s)$
unfolding *distrib-left distrib-right* **by**(*simp add*: *mult.assoc mult.commute*)
show *?thesis*
unfolding *times-divide-times-eq distrib-left distrib-right 1*
by (*simp add*: *mult.assoc mult.commute*)
qed
also have $... = ?r$
by(*auto simp*: *distrib-right distrib-left mult.commute*)
finally show *?thesis* **.**
qed
qed
also have $... \longrightarrow 1 / 2$
by(*rule Topological-Spaces.tendsto-eq-intros*(*25*)[*of - d * b - - 2 * b * d*,*OF LIMSEQ-Suc LIMSEQ-Suc*]) (*intro Topological-Spaces.tendsto-eq-intros* | *auto*)+
finally show *?thesis*
by(*rule LIMSEQ-imp-Suc*)
qed

lemma *GaussLearn-KL-divergence-lem2'*:
fixes s :: *real*
assumes [*arith*]: $s > 0\ b > 0\ d > 0$
shows $(\lambda n.\ ((d\hat{}2 * s\hat{}2) / (n * d\hat{}2 + s\hat{}2)) / (2 * ((b\hat{}2 * s\hat{}2) / (n * b\hat{}2 + s\hat{}2))) - 1 / 2) \longrightarrow 0$
using *GaussLearn-KL-divergence-lem2*[*of s^2 b^2 d^2*]
by(*rule LIM-zero*) *auto*

lemma *GaussLearn-KL-divergence-lem3*:
fixes $a\ b\ c\ d\ s\ K\ L$ :: *real*
assumes [*arith*]: $b > 0\ d > 0\ s > 0$
shows $((K * d + c * s) / (n * d + s) - (L * b + a * s) / (n * b + s))\hat{}2 / (2 * ((b * s) / (n * b + s))) = ((((((K - L) * d * b * real\ n + c * s * b * real\ n + K * d * s + c * s * s) - a * s * d * real\ n - L * b * s - a * s * s))^2 / (d * d * b * (real\ n * real\ n * real\ n) + s * s * b * real\ n + 2 * d * s * b * (real\ n * real\ n) + d * d * (real\ n * real\ n) * s + s * s * s + 2 * d * s * s * real\ n))) / (2 * (b * s))$ (**is** $?lhs = ?rhs$)
proof −
have *0*:$real\ n * d + s > 0\ real\ n * b + s > 0$
by(*auto intro*!: *add-nonneg-pos*)

hence *1*:*real n * d + s ≠ 0 real n * b + s ≠ 0* **by** *simp-all*
have *?lhs* = (((*K * d + c * s*) * (*n * b + s*) − (*L * b + a * s*) * (*n * d + s*)) / ((*n * d + s*) * (*n * b + s*)))^2 / (*2* * (*b * s* / (*n * b + s*)))
unfolding *diff-frac-eq*[*OF 1*] **by** *simp*
also have ... = (((((*K * d + c * s*) * (*n * b + s*) − (*L * b + a * s*) * (*n * d + s*)))^2 / ((*n * d + s*)^*2* * (*n * b + s*)))) / (*2* * (*b * s*))
by(*auto simp*: *power2-eq-square*)
also have ... = (((((*K * d * (n * b*) + *c * s * (n * b*) + *K * d * s + c * s * s*) − ((*L * b * (n * d*) + *a * s * (n * d*) + *L * b * s + a * s * s*))))^2 / ((*n * d*)^*2* * (*n * b*) + *s*^*2* * (*n * b*) + *2* * (*n * d*) * *s * (n * b*) + (*n * d*)^*2* * *s* + *s*^*2* * *s* + *2* * (*n * d*) * *s * s*))) / (*2* * (*b * s*))
by(*simp add*: *power2-sum distrib-left distrib-right is-num-normalize*(*1*))
also have ... = (((((*K * d * b * real n + c * s * s * b * real n + K * d * s + c * s * s*) − ((*L * b * d * real n + a * s * s * d * real n + L * b * s + a * s * s*))))^2 / (*d * d * b * (real n * real n * real n*) + *s * s * s * b *real n + 2 * d * s * s * b * (real n * real n*) + *d * d * (real n * real n*) * *s* + *s * s * s * s* + *2 * d * s * s * s * real n*))) / (*2* * (*b * s*))
by (*simp add*: *mult.commute mult.left-commute power2-eq-square*)
also have ... = ((((((*K* − *L*) * *d * b * real n + c * s * s * b * real n + K * d * s* + *c * s * s * s*) − ((*a * s * s * d * real n + L * b * s + a * s * s*))))^2 / (*d * d * b * (real n * real n * real n*) + *s * s * s * b * real n + 2 * d * s * s * b * (real n * real n*) + *d * d * (real n * real n*) * *s* + *s * s * s * s* + *2 * d * s * s * s * real n*))) / (*2* * (*b * s*))
proof −
have *1*:*K * d * b * real n + c * s * s * b * real n + K * d * s + c * s * s * s* − (*L * b * d * real n + a * s * s * d * real n + L * b * s + a * s * s*) = (*K* − *L*) * *d * b * real n + c * s * s * b * real n + K * d * s + c * s * s * s* − (*a * s * s * d * real n + L * b * s + a * s * s*)
by (*simp add*: *left-diff-distrib*)
show *?thesis*
unfolding *1* **..**
qed
also have ... = *?rhs*
by (*simp add*: *diff-diff-eq*)
finally show *?thesis* **.**
qed

lemma *GaussLearn-KL-divergence-lem4*:
fixes *a b c d s K L* :: *real*
assumes [*arith*]: *b > 0 d > 0 s > 0*
shows (λ*n*. (|*c * s * s * b * real n*| + |*K * (real n) * d * s*| + |*c * s * s * s*| + |*a * s * d * real n*| + |*L * (real n) * b * s*| + |*a * s * s * s*|)^2 / (*d * d * b * (real n * real n * real n*) + *s * s * s * b * real n + 2 * d * s * s * b * (real n * real n*) + *d * d * (real n * real n*) * *s* + *s * s * s * s* + *2 * d * s * s * s * real n*) / (*2* * (*b * s*))) ⟶ *0* (**is** (λ*n*. *?f n*) ⟶ *0*)
proof −
have *t1*: (λ*n*. *x* / (*real n * real n*)) ⟶ *0* **for** *x*
proof −
have (λ*n*. *x* / (*real n * real n*)) = (λ*n*. *x* / (*real n*) * (*1* / *real n*))
by *simp*

```
    also have ... ⟶ 0
      by (intro Topological-Spaces.tendsto-eq-intros | auto)+
    finally show ?thesis .
  qed
  have t4: (λn. x / (real n * real n * real n)) ⟶ 0 for x
  proof −
    have (λn. x / (real n * real n * real n)) = (λn. x / (real n) * (1 / real n) * (1
/ real n))
      by simp
    also have ... ⟶ 0
      by (intro Topological-Spaces.tendsto-eq-intros | auto)+
    finally show ?thesis .
  qed
  have t2[tendsto-intros]: (λn. x / (sqrt n)) ⟶ 0 for x
    by(rule power-tendsto-0-iff[of 2,THEN iffD1],simp-all add: power2-eq-square)
(intro Topological-Spaces.tendsto-eq-intros | auto)+
  have t3: (λn. x / (sqrt n * real n)) ⟶ 0 for x
  proof −
    have (λn. x / (sqrt n * real n)) = (λn. x / sqrt n * (1 / n)) by simp
    also have ... ⟶ 0
      by (intro Topological-Spaces.tendsto-eq-intros | auto)+
    finally show ?thesis .
  qed

  have (λn. ?f (Suc n)) = (λn. ((|(c * s * b) / sqrt (real (Suc n))| + |(K * d *
s) / sqrt (real (Suc n))| + |(c * s * s) / (sqrt (Suc n) * real (Suc n))| + |(a * s *
d) / sqrt (real (Suc n))| + |(L * b * s) / sqrt (real (Suc n))| + |(a * s * s) / (sqrt
(Suc n) * real (Suc n))|)^2 / ((d * d * b + (s * s * b) / (real (Suc n) * real (Suc
n)) + (2 * d * s * b) / real (Suc n) + (d * d * s) / real (Suc n) + (s * s * s)
/ (real (Suc n) * real (Suc n) * real (Suc n)) + (2 * d * s * s) / (real (Suc n) *
real (Suc n))))) / (2 * (b * s))) (is - = (λn. ?g (Suc n)))
  proof
    fix n
    show ?f (Suc n) = ?g (Suc n)  (is ?lhs = ?rhs)
    proof −
      have ?lhs = (|c * s * b * real (Suc n)| + |K * d * s * real (Suc n)| + |c * s
* s| + |a * s * d * real (Suc n)| + |L * b * s * real (Suc n)| + |a * s * s|)^2 / (d
* d * b * (real (Suc n) * real (Suc n) * real (Suc n)) + s * s * b * real (Suc n) +
2 * d * s * b * (real (Suc n) * real (Suc n)) + d * d * (real (Suc n) * real (Suc
n)) * s + s * s * s + 2 * d * s * s * real (Suc n)) / (2 * (b * s))
      proof −
        have 1:K * real (Suc n) * d * s = K * d * s * real (Suc n) L * real (Suc
n) * b * s = L * b * s * real (Suc n)
          by auto
        show ?thesis
          unfolding 1 ..
      qed
      also have ... = ((|c * s * b / sqrt (real (Suc n))| + |K * d * s / sqrt (real
(Suc n))| + |(c * s * s) / (sqrt (Suc n) * real (Suc n))| + |a * s * d / sqrt (real
```

$(Suc\ n))| + |L * b * s\ /\ sqrt\ (real\ (Suc\ n))| + |(a * s * s)\ /\ (sqrt\ (Suc\ n) * real\ (Suc\ n))|) * (sqrt\ (Suc\ n) * real\ (Suc\ n))\ )^2\ /\ (d * d * b * (real\ (Suc\ n) * real\ (Suc\ n) * real\ (Suc\ n)) + s * s * b * real\ (Suc\ n) + 2 * d * s * b * (real\ (Suc\ n) * real\ (Suc\ n)) + d * d * (real\ (Suc\ n) * real\ (Suc\ n)) * s + s * s * s + 2 * d * s * s * real\ (Suc\ n))\ /\ (2 * (b * s))$

by(*simp add*: *distrib-right left-diff-distrib mult.assoc*[*symmetric*] *abs-mult*[*of - real* (*Suc n*)] *del*: *of-nat-Suc*)

also have ... = $((|c * s * b\ /\ sqrt\ (real\ (Suc\ n))| + |K * d * s\ /\ sqrt\ (real\ (Suc\ n))| + |(c * s * s)\ /\ (sqrt\ (Suc\ n) * real\ (Suc\ n))| + |a * s * d\ /\ sqrt\ (real\ (Suc\ n))| + |L * b * s\ /\ sqrt\ (real\ (Suc\ n))| + |(a * s * s)\ /\ (sqrt\ (Suc\ n) * real\ (Suc\ n))|)\hat{}2 * (real\ (Suc\ n) * real\ (Suc\ n) * real\ (Suc\ n)))\ /\ (d * d * b * (real\ (Suc\ n) * real\ (Suc\ n) * real\ (Suc\ n)) + s * s * b * real\ (Suc\ n) + 2 * d * s * b * (real\ (Suc\ n) * real\ (Suc\ n)) + d * d * (real\ (Suc\ n) * real\ (Suc\ n)) * s + s * s * s + 2 * d * s * s * real\ (Suc\ n))\ /\ (2 * (b * s))$

by(*simp add*: *power2-eq-square*)

also have ... = $((|c * s * b\ /\ sqrt\ (real\ (Suc\ n))| + |K * d * s\ /\ sqrt\ (real\ (Suc\ n))| + |(c * s * s)\ /\ (sqrt\ (Suc\ n) * real\ (Suc\ n))| + |a * s * d\ /\ sqrt\ (real\ (Suc\ n))| + |L * b * s\ /\ sqrt\ (real\ (Suc\ n))| + |(a * s * s)\ /\ (sqrt\ (Suc\ n) * real\ (Suc\ n))|)\hat{}2\ /\ ((d * d * b * (real\ (Suc\ n) * real\ (Suc\ n) * real\ (Suc\ n)) + s * s * b * real\ (Suc\ n) + 2 * d * s * b * (real\ (Suc\ n) * real\ (Suc\ n)) + d * d * (real\ (Suc\ n) * real\ (Suc\ n)) * s + s * s * s + 2 * d * s * s * s * real\ (Suc\ n))\ /\ (real\ (Suc\ n) * real\ (Suc\ n) * real\ (Suc\ n))))\ /\ (2 * (b * s))$

by *simp*

also have ... = *?rhs*

by(*simp add*: *add-divide-distrib*)

finally show *?thesis* **.**

qed

qed

also have ... $\longrightarrow$ *0*

apply(*rule LIMSEQ-Suc*)

apply(*rule Topological-Spaces.tendsto-eq-intros*(*25*)[*of - 0 - - 2 * (b * s)*,*OF Topological-Spaces.tendsto-eq-intros*(*25*)[*of - 0 - - d * d * b*]])

apply(*intro lim-const-over-n t1 t2 t3 t4 tendsto-diff*[*of - 0 - - 0*,*simplified*] *tendsto-add-zero tendsto-add*[*of - d * d * b - - 0*,*simplified*] | *auto*)+

done

finally show *?thesis*

by(*rule LIMSEQ-imp-Suc*)

qed

lemma *GaussLearn-KL-divergence-lem5*:

fixes *a b c d K* :: *real*

assumes [*arith*]: $b > 0\ d > 0\ s > 0\ K > 0\ |f\ l| < K * length\ l$

shows $|(c * s * b * real\ (length\ l) + f\ l * d * s + c * s * s - a * s * d * real\ (length\ l) - f\ l * b * s - a * s * s)^2\ /\ (d * d * b * (real\ (length\ l) * real\ (length\ l) * real\ (length\ l)) + s * s * b * real\ (length\ l) + 2 * d * s * b * (real\ (length\ l) * real\ (length\ l)) + d * d * (real\ (length\ l) * real\ (length\ l)) * s + s * s * s + 2 * d * s * s * real\ (length\ l))\ /\ (2 * (b * s))| \leq\ |(|c * s * b * real\ (length\ l)| + |K * real\ (length\ l) * d * s| + |c * s * s| + |a * s * d * real\ (length\ l)| + |- K * real\ (length\ l) * b * s| + |a * s * s|)^2\ /\ (d * d * b * (real\ (length\ l) * real\ (length\ l) *$

*real (length l)) + s * s * b * real (length l) + 2 * d * s * b * (real (length l) * real (length l)) + d * d * (real (length l) * real (length l)) * s + s * s * s + 2 * d * s * s * real (length l)) / (2 * (b * s))|* (**is** *|(?l)^2 / ?c1 / ?c2| ≤ |(?r)^2 / - / -|*)
proof −
  have *?l^2 / ?c1 / ?c2 ≤ ?r^2 / ?c1 / ?c2*
   proof(*rule divide-right-mono*[*OF divide-right-mono*[*OF abs-le-square-iff*[*THEN iffD1*]]])
    show *|?l| ≤ |?r|*
    proof −
     have *|?l| ≤ |c * s * b * real (length l)| + |f l * d * s| + |c * s * s| + |a * s * d * real (length l)| + |f l * b * s| + |a * s * s|*
       by *linarith*
     also have *... ≤ |?r|*
      by (*auto simp*: *mult.assoc abs-mult*) (*auto intro*!: *add-mono*)
     finally show *?thesis* **.**
    qed
  qed *auto*
   thus *?thesis*
    by *fastforce*
qed

lemma *GaussLearn-KL-divergence-lem6*:
  fixes *a e b c d K* :: *real* **and** *f* :: *'a list* ⇒ *real*
  assumes [*arith*]:*e > 0 b > 0 d > 0 s > 0*
  shows ∃ *N*. ∀ *l*. *length l ≥ N* ⟶ *|f l| < K * length l* ⟶ *|((f l * d + c * s) / (length l * d + s) − (f l * b + a * s) / (length l * b + s))^2 / (2 * ((b * s) / (length l * b + s))) | < e*
proof(*cases K > 0*)
  case *K*[*arith*]:*True*
  from *GaussLearn-KL-divergence-lem4*[*OF assms*(*2*−),*of c K a − K*] *assms*(*1*)
obtain *N* **where** *N*:
  ⋀*n*. *n ≥ N* ⟹ *|(|c * s * b * real n| + |K * real n * d * s| + |c * s * s| + |a * s * d * real n| + |− K * real n * b * s| + |a * s * s|)^2 / (d * d * b * (real n * real n * real n) + s * s * b * real n + 2 * d * s * b * (real n * real n) + d * d * (real n * real n) * s + s * s * s + 2 * d * s * s * real n) / (2 * (b * s))| < e*
   by(*fastforce simp*: *LIMSEQ-def*)
  show *?thesis*
  proof(*safe intro*!: *exI*[**where** *x*=*N*])
   fix *l* :: *'a list*
   assume *l*:*N ≤ length l |f l| < K * real (length l)*
   show *|((f l * d + c * s) / (real (length l) * d + s) − (f l * b + a * s) / (real (length l) * b + s))^2 / (2 * (b * s / (real (length l) * b + s)))| < e* (**is** *?l < -*)
   proof −
    have *?l = |(c * s * b * real (length l) + f l * d * s + c * s * s − a * s * d * real (length l) − f l * b * s − a * s * s)^2 / (d * d * b * (real (length l) * real (length l) * real (length l)) + s * s * b * real (length l) + 2 * d * s * b * (real (length l) * real (length l)) + d * d * (real (length l) * real (length l)) * s + s * s * s + 2 * d * s * s * real (length l)) / (2 * (b * s))|*
      unfolding *GaussLearn-KL-divergence-lem3*[*OF assms*(*2*−)] **by** *simp*

also have ... ≤ |(|*c* * *s* * *b* * *real* (*length l*)| + |*K* * *real* (*length l*) * *d* * *s*| + |*c* * *s* * *s*| + |*a* * *s* * *d* * *real* (*length l*)| + |− *K* * *real* (*length l*) * *b* * *s*| + |*a* * *s* * *s*|)² / (*d* * *d* * *b* * (*real* (*length l*) * *real* (*length l*) * *real* (*length l*)) + *s* * *s* * *b* * *real* (*length l*) + *2* * *d* * *s* * *b* * (*real* (*length l*) * *real* (*length l*)) + *d* * *d* * (*real* (*length l*) * *real* (*length l*)) * *s* + *s* * *s* * *s* * *s* + *2* * *d* * *s* * *s* * *real* (*length l*)) / (*2* * (*b* * *s*))|
by(*rule GaussLearn-KL-divergence-lem5*) (*use l* **in** *auto*)
also have ... < *e*
by(*rule N*) *fact*
finally show *?thesis* **.**
qed
qed
next
case *False*
then show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *abs-ge-zero add-le-cancel-left add-nonneg-nonneg diff-add-cancel diff-ge-0-iff-ge linorder-not-less of-nat-0-le-iff zero-less-mult-iff*)
qed

lemma *GaussLearn-KL-divergence*:
fixes *a b c d e K* :: *real*
assumes [*arith*]:*e* > *0 b* > *0 d* > *0*
shows ∃ *N*. ∀ *L*. *length L* > *N* ⟶ |*Total L* / *length L*| < *K* ⟶ *KL-divergence* (*exp 1*) (*GaussLearn* (*Gauss a b*) *L*) (*GaussLearn* (*Gauss c d*) *L*) < *e*
proof −
have *h*:*σ*^*2* > *0 b*^*2*>*0 d*^*2*>*0*
by *auto*
from *GaussLearn-KL-divergence-lem6*[*of e* / *3*,*OF* - *h*(*2*,*3*,*1*)] **obtain** *N1* **where** *N1*:
⋀*l*. *N1* ≤ *length l* ⟹ |*Total l*| < *K* * *real* (*length l*) ⟹ |((*Total l* * *d*² + *c* * *σ*²) / (*real* (*length l*) * *d*² + *σ*²) − (*Total l* * *b*² + *a* * *σ*²) / (*real* (*length l*) * *b*² + *σ*²))² / (*2* *(*b*² * *σ*² / (*real* (*length l*) * *b*² + *σ*²)))| < *e* / *3*
by *fastforce*
from *GaussLearn-KL-divergence-lem1'*[*OF assms*(*2*,*3*) ‹*σ* > *0*›]
have ⋀*e*. *e* > *0* ⟹ ∃ *N*. ∀ *n*. *n* ≥ *N* ⟶ |*ln* (*sqrt* (*b*² * *σ*² / (*real n* * *b*² + *σ*²)) / *sqrt* (*d*² * *σ*² / (*real n* * *d*² + *σ*²)))| < *e*
by(*auto simp*: *LIMSEQ-def*)
from *this*[*of e* / *3*] **obtain** *N2* **where** *N2*:
⋀*n*. *n* ≥ *N2* ⟹ |*ln* (*sqrt* (*b*² * *σ*² / (*real n* * *b*² + *σ*²)) / *sqrt* (*d*² * *σ*² / (*real n* * *d*² + *σ*²)))| < *e* / *3*
by *auto*
from *GaussLearn-KL-divergence-lem2'*[*OF* ‹*σ* > *0*› *assms*(*2*,*3*)]
have ⋀*e*. *e* > *0* ⟹ ∃ *N*. ∀ *n*. *n* ≥ *N* ⟶ |*d*² * *σ*² / (*real n* * *d*² + *σ*²) / (*2* * (*b*² * *σ*² / (*real n* * *b*² + *σ*²))) − *1* / *2*| < *e*
by(*auto simp*: *LIMSEQ-def*)
from *this*[*of e* / *3*] **obtain** *N3* **where** *N3*:
⋀*n*. *n* ≥ *N3* ⟹ |*d*² * *σ*² / (*real n* * *d*² + *σ*²) / (*2* * (*b*² * *σ*² / (*real n* * *b*² + *σ*²))) − *1* / *2*| < *e* / *3*

by *auto*
define N **where** $N = max\ (max\ N1\ N2)\ (max\ N3\ 1)$
have N: $N \geq N1\ N \geq N2\ N \geq N3\ N \geq 1$
by(*auto simp*: *N-def*)
show *?thesis*
proof(*safe intro*!: *exI*[**where** x=N])
fix L :: *real list*
assume l:$N < length\ L\ |local.Total\ L\ /\ real\ (length\ L)| < K$
then have l': $N \leq length\ L\ |Total\ L| < K * real\ (length\ L)$
using *order.strict-trans1*[*OF N*(*4*) *l*(*1*)] **by**(*auto intro*!: *pos-divide-less-eq*[*THEN iffD1*])
show *KL-divergence* (*exp 1*) (*GaussLearn* (*Gauss a b*) *L*) (*GaussLearn* (*Gauss c d*) *L*) < *e* (**is** *?lhs* < -)
proof −
have h': $sqrt\ (b^2 * \sigma^2\ /\ (real\ (length\ L) * b^2 + \sigma^2)) > 0\ sqrt\ (d^2 * \sigma^2\ /\ (real\ (length\ L) * d^2 + \sigma^2)) > 0$
by(*auto intro*!: *divide-pos-pos add-nonneg-pos*)
have *?lhs* ≤ |*?lhs*|
by *auto*
also have ... = |*KL-divergence* (*exp 1*) (*Gauss* ((*Total L* * b^2 + *a* * σ^2) / (*real* (*length L*) * b^2 + σ^2)) (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)))) (*Gauss* ((*Total L* * d^2 + *c* * σ^2) / (*real* (*length L*) * d^2 + σ^2)) (*sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2))))|
by(*simp add*: *GaussLearn-Total*[*OF assms*(*2*) *refl*] *GaussLearn-Total*[*OF assms*(*3*) *refl*])
also have ... = |*ln* (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)) / *sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2))) + ((*sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2)))^2 + ((*Total L* * d^2 + *c* * σ^2) / (*real* (*length L*) * d^2 + σ^2) − (*Total L* * b^2 + *a* * σ^2) / (*real* (*length L*) * b^2 + σ^2))^2) / (*2* * (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)))^2) − *1* / *2*|
by(*simp add*: *KL-normal-density*[*OF h'*] *Gauss-def*)
also have ... = |*ln* (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)) / *sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2))) + (*sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2)))^2 / (*2* * (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)))^2) + ((*Total L* * d^2 + *c* * σ^2) / (*real* (*length L*) * d^2 + σ^2) − (*Total L* * b^2 + *a* * σ^2) / (*real* (*length L*) * b^2 + σ^2))^2 / (*2* * (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)))^2) − *1* / *2*|
unfolding *add-divide-distrib* **by** *auto*
also have ... = |*ln* (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)) / *sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2))) + (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2)) / (*2* * (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2))) + ((*Total L* * d^2 + *c* * σ^2) / (*real* (*length L*) * d^2 + σ^2) − (*Total L* * b^2 + *a* * σ^2) / (*real* (*length L*) * b^2 + σ^2))^2 / (*2* * (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2))) − *1* / *2*|
using h' **by** *auto*
also have ... ≤ |*ln* (*sqrt* (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)) / *sqrt* (d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2))) + ((d^2 * σ^2 / (*real* (*length L*) * d^2 + σ^2)) / (*2* * (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2))) − *1* / *2*) + ((*Total L* * d^2 + *c* * σ^2) / (*real* (*length L*) * d^2 + σ^2) − (*Total L* * b^2 + *a* * σ^2) / (*real* (*length L*) * b^2 + σ^2))^2 / (*2* * (b^2 * σ^2 / (*real* (*length L*) * b^2 + σ^2)))|
by *auto*

also have ... $\leq |ln\ (sqrt\ (b^2 * \sigma^2\ /\ (real\ (length\ L) * b^2 + \sigma^2))\ /\ sqrt\ (d^2 * \sigma^2\ /\ (real\ (length\ L) * d^2 + \sigma^2)))| + |(d^2 * \sigma^2\ /\ (real\ (length\ L) * d^2 + \sigma^2))\ /\ (2 * (b^2 * \sigma^2\ /\ (real\ (length\ L) * b^2 + \sigma^2))) - 1\ /\ 2| + |((Total\ L * d^2 + c * \sigma^2)\ /\ (real\ (length\ L) * d^2 + \sigma^2) - (Total\ L * b^2 + a * \sigma^2)\ /\ (real\ (length\ L) * b^2 + \sigma^2))^2\ /\ (2 * (b^2 * \sigma^2\ /\ (real\ (length\ L) * b^2 + \sigma^2)))|$
by *linarith*
also have ... $< e$
using *N1*[*OF order.trans*[*OF N*(*1*) *l'*(*1*)] *l'*(*2*)] *N2*[*OF order.trans*[*OF N*(*2*) *l'*(*1*)]] *N3*[*OF order.trans*[*OF N*(*3*) *l'*(*1*)]] **by** *auto*
finally show *?thesis* **.**
qed
qed
qed

end

5.2.4 Continuous Distributions

The following (highr-order) program receives a non-negative function f and returns the distribution whose density function is (noramlized) f if f is integrable w.r.t. the Lebesgue measure.

definition *dens-to-dist* :: [*'a* :: *euclidean-space* ⇒ *real*] ⇒ *'a qbs-measure* **where**
dens-to-dist ≡ (λ*f*. *do* {
query lborel$_Q$ *f*
})

lemma *dens-to-dist-qbs*[*qbs*]: *dens-to-dist* ∈ (*borel*$_Q$ ⇒$_Q$ $\mathbb{R}_Q$) →$_Q$ *monadM-qbs borel*$_Q$
by(*simp add*: *dens-to-dist-def*)

context
fixes *f* :: *'a* :: *euclidean-space* ⇒ *real*
assumes *f-qbs*[*qbs*]: *f* ∈ *qbs-borel* →$_Q$ $\mathbb{R}_Q$
and *f-le0*:⋀*x*. *f x* ≥ *0*
and *f-int-ne0*:*qbs-l* (*density-qbs lborel-qbs f*) *UNIV* ≠ *0*
and *f-integrable*: *qbs-integrable lborel-qbs f*
begin

lemma *f-integrable'*[*measurable*]: *integrable lborel f*
using *f-integrable* **by**(*simp add*: *qbs-integrable-iff-integrable*)

lemma *f-int-neinfty*:
qbs-l (*density-qbs lborel-qbs f*) *UNIV* ≠ ∞
using *f-integrable' f-le0*
by(*auto simp*: *qbs-l-density-qbs*[*of - qbs-borel*] *emeasure-density integrable-iff-bounded*)

lemma *dens-to-dist*: *dens-to-dist f* = *density-qbs lborel-qbs* (λ*x*. *ennreal* (*1* / *measure* (*qbs-l* (*density-qbs lborel-qbs f*)) *UNIV* ∗ *f x*))
proof −

have [*simp*]:*ennreal* (*f x*) ∗ (*1* / *emeasure* (*qbs-l* (*density-qbs lborel*$_Q$ (λ*x*. *ennreal* (*f x*)))) *UNIV*) = *ennreal* (*f x* / *measure* (*qbs-l* (*density-qbs lborel*$_Q$ (λ*x*. *ennreal* (*f x*)))) *UNIV*) **for** *x*
by (*metis divide-ennreal emeasure-eq-ennreal-measure ennreal-0 ennreal-times-divide f-int-ne0 f-int-neinfty f-le0 infinity-ennreal-def mult.comm-neutral zero-less-measure-iff*)
show *?thesis*
by(*auto simp*: *dens-to-dist-def query-def normalize-qbs*[*of - qbs-borel*,*simplified qbs-space-qbs-borel*,*OF - f-int-ne0 f-int-neinfty*] *density-qbs-density-qbs-eq*[*of - qbs-borel*])
qed

corollary *qbs-l-dens-to-dist*: *qbs-l* (*dens-to-dist f*) = *density lborel* (λ*x*. *ennreal* (*1* / *measure* (*qbs-l* (*density-qbs lborel-qbs f*)) *UNIV* ∗ *f x*))
by(*simp add*: *dens-to-dist qbs-l-density-qbs*[*of - qbs-borel*])

corollary *qbs-integral-dens-to-dist*:
assumes [*qbs*]: *g* ∈ *qbs-borel* →$_Q$ $\mathbb{R}_Q$
shows (∫$_Q$ *x*. *g x* ∂*dens-to-dist f*) = (∫$_Q$ *x*. *1* / *measure* (*qbs-l* (*density-qbs lborel-qbs f*)) *UNIV* ∗ *f x* ∗ *g x* ∂*lborel*$_Q$)
using *f-le0* **by**(*simp add*: *qbs-integral-density-qbs*[*of - qbs-borel - g* ,*OF - - - AEq-I2*[*of - qbs-borel*]] *dens-to-dist*)

lemma *dens-to-dist-prob*[*qbs*]:*dens-to-dist f* ∈ *qbs-space* (*monadP-qbs borel*$_Q$)
using *f-int-neinfty f-int-ne0* **by**(*auto simp*: *dens-to-dist-def query-def intro*!: *normalize-qbs-prob*)

end

5.2.5 Normal Distribution

context
fixes μ σ :: *real*
assumes *sigma-pos*[*arith*]: σ > *0*
begin

We use an unnormalized density function.

definition *normal-f* ≡ (λ*x*. *exp* (−(*x* − μ)^2/ (*2* ∗ σ^2)))

lemma *nc-normal-f*: *qbs-l* (*density-qbs lborel-qbs normal-f*) *UNIV* = *ennreal* (*sqrt* (*2* ∗ *pi* ∗ σ^2))
proof −
have *qbs-l* (*density-qbs lborel-qbs normal-f*) *UNIV* = (∫$^+$ *x*. *ennreal* (*exp* (− ((*x* − μ)^2 / (*2* ∗ σ^2)))) ∂*lborel*)
by(*auto simp*: *qbs-l-density-qbs*[*of - qbs-borel*] *normal-f-def emeasure-density*)
also have ... = *ennreal* (*sqrt* (*2* ∗ *pi* ∗ σ^2)) ∗ (∫$^+$ *x*. *normal-density* μ σ *x* ∂*lborel*)
by(*auto simp*: *nn-integral-cmult*[*symmetric*] *normal-density-def ennreal-mult'*[*symmetric*] *intro*!: *nn-integral-cong*)
also have ... = *ennreal* (*sqrt* (*2* ∗ *pi* ∗ σ^2))
using *prob-space.emeasure-space-1*[*OF prob-space-normal-density*]

by(*simp add*: *emeasure-density*)
finally show *?thesis* **.**
qed

corollary *measure-qbs-l-dens-to-dist-normal-f*: *measure* (*qbs-l* (*density-qbs lborel-qbs normal-f*)) *UNIV* = *sqrt* (*2* ∗ *pi* ∗ σ^2)
by(*simp add*: *measure-def nc-normal-f*)

lemma *normal-f*:
shows *normal-f* ∈ *qbs-borel* →$_Q$ $\mathbb{R}_Q$
and ⋀*x*. *normal-f x* ≥ *0*
and *qbs-l* (*density-qbs lborel-qbs normal-f*) *UNIV* ≠ *0*
and *qbs-integrable lborel-qbs normal-f*
using *nc-normal-f* **by**(*auto simp*: *qbs-integrable-iff-integrable integrable-iff-bounded qbs-l-density-qbs*[*of - qbs-borel*] *normal-f-def emeasure-density*)

lemma *qbs-l-densto-dist-normal-f*: *qbs-l* (*dens-to-dist normal-f*) = *density lborel* (*normal-density* μ σ)
by(*simp add*: *qbs-l-dens-to-dist*[*OF normal-f*] *measure-qbs-l-dens-to-dist-normal-f normal-density-def*) (*simp add*: *normal-f-def*)

end

5.2.6 Half Normal Distribution

context
fixes μ σ :: *real*
assumes *sigma-pos*[*arith*]:σ > *0*
begin

definition *hnormal-f* ≡ (λ*x*. *if x* ≤ μ *then 0 else normal-density* μ σ *x*)

lemma *nc-hnormal-f*: *qbs-l* (*density-qbs lborel-qbs hnormal-f*) *UNIV* = *ennreal* (*1 / 2*)
proof −
have *qbs-l* (*density-qbs lborel-qbs hnormal-f*) *UNIV* = (∫$^+$ *x*. *ennreal* (*if x* ≤ μ *then 0 else normal-density* μ σ *x*) ∂*lborel*)
by(*auto simp*: *qbs-l-density-qbs*[*of - qbs-borel*] *hnormal-f-def emeasure-density*)
also have ... = (∫$^+$ *x*∈{μ<..}. *normal-density* μ σ *x* ∂*lborel*)
by(*auto intro*!: *nn-integral-cong*)
also have ... = *1 / 2* ∗ (∫$^+$ *x*. *normal-density* μ σ *x* ∂*lborel*)
proof −
have *1*:(∫$^+$ *x*. *normal-density* μ σ *x* ∂*lborel*) = (∫$^+$ *x*∈{μ<..}. *normal-density* μ σ *x* ∂*lborel*) + (∫$^+$ *x*∈{..μ}. *normal-density* μ σ *x* ∂*lborel*)
by(*auto simp*: *nn-integral-add*[*symmetric*] *intro*!: *nn-integral-cong*) (*simp add*: *indicator-def*)
have *2*: (∫$^+$ *x*∈{μ<..}. *normal-density* μ σ *x* ∂*lborel*) = (∫$^+$ *x*∈{..μ}. *normal-density* μ σ *x* ∂*lborel*) (**is** *?l* = *?r*)

proof −
have *?l* = (∫⁺ *x*. *ennreal* (*normal-density* μ σ *x* ∗ *indicator* {μ<..} *x*) ∂*lborel*)
by(*auto intro*!: *nn-integral-cong simp add*: *indicator-mult-ennreal mult.commute*)
also have ... = *ennreal* (∫ *x*. *normal-density* μ σ *x* ∗ *indicator* {μ<..} *x* ∂*lborel*)
by(*auto intro*!: *nn-integral-eq-integral integrable-real-mult-indicator*)
also have ... = *ennreal* (∫ *x*. *normal-density* μ σ *x* ∗ *indicator* {μ<..} *x* ∂*lebesgue*)
by(*simp add*: *integral-completion*)
also have ... = *ennreal* (∫ *x*. (*if x* ∈ {μ<..} *then normal-density* μ σ *x else 0*) ∂*lebesgue*)
by (*meson indicator-times-eq-if*(*2*))
also have ... = *ennreal* (∫ *x*. *normal-density* μ σ *x* ∂*lebesgue-on* {μ<..})
by(*rule ennreal-cong*, *rule Lebesgue-Measure.integral-restrict-UNIV*) *simp*
also have ... = *ennreal* (*integral* {μ<..} (*normal-density* μ σ))
by(*rule ennreal-cong*, *rule lebesgue-integral-eq-integral*) (*auto simp*: *integrable-restrict-space integrable-completion intro*!: *integrable-mult-indicator*[**where** *'b=real*,*simplified*])
also have ... = *ennreal* (*integral* {..<μ} (λ*x*. *normal-density* μ σ (− *x* + *2* ∗ μ)))
proof −
have *integral* {μ<..} (*normal-density* μ σ) = *integral* {..<μ} (λ*x*. |− *1*| ∗$_R$ *normal-density* μ σ (− *x* + *2* ∗ μ))
proof(*rule conjunct2*[*OF has-absolute-integral-change-of-variables-1'*[**where** *g*=λ*x*. − *x* + *2* ∗ μ **and** *S*={..<μ} **and** *g'*=λ*x*. − *1* **and** *f*=*normal-density* μ σ **and** *b*=*integral* {μ<..} (*normal-density* μ σ),*THEN iffD2*],*symmetric*])
fix *x* :: *real*
show ((λ*x*. − *x* + *2* ∗ μ) *has-real-derivative* − *1*) (*at x within* {..<μ})
by(*rule derivative-eq-intros*(*35*)[*of* - − *1* - - *0*]) (*auto simp add*: *Deriv.field-differentiable-minus*)
next
show *inj-on* (λ*x*. − *x* + *2* ∗ μ) {..<μ}
by(*auto simp*: *inj-on-def*)
next
have *1*: (λ*x*. − *x* + *2* ∗ μ) ‘ {..<μ} = {μ<..}
by(*auto simp*: *image-def intro*!: *bexI*[**where** *x*=*2* ∗ μ − -])
have [*simp*]: *normal-density* μ σ *absolutely-integrable-on* {μ<..}
by(*auto simp*: *absolutely-integrable-measurable comp-def integrable-restrict-space integrable-completion intro*!: *integrable-mult-indicator*[**where** *'b=real*,*simplified*] *measurable-restrict-space1 measurable-completion*)
show *normal-density* μ σ *absolutely-integrable-on* (λ*x*. − *x* + *2* ∗ μ) ‘ {..<μ} ∧ *integral* ((λ*x*. − *x* + *2* ∗ μ) ‘ {..<μ}) (*normal-density* μ σ) = *integral* {μ<..} (*normal-density* μ σ)
unfolding *1* **by** *simp*
qed *auto*
thus *?thesis* **by** *simp*
qed
also have ... = *ennreal* (*integral* {..<μ} (*normal-density* μ σ))
proof −

have (λx. *normal-density* μ σ ($-$ x + 2 $*$ μ)) = *normal-density* μ σ
by *standard* (*auto simp*: *normal-density-def power2-commute*)
thus *?thesis* **by** *simp*
qed
also have ... = *ennreal* ($\int$ x. *normal-density* μ σ x ∂*lebesgue-on* {..<μ})
by(*rule ennreal-cong*, *rule lebesgue-integral-eq-integral*[*symmetric*]) (*auto simp*: *integrable-restrict-space integrable-completion intro*!: *integrable-mult-indicator*[**where** *'b=real*,*simplified*])
also have ... = *ennreal* ($\int$ x. (*if* x $\in$ {..<μ} *then normal-density* μ σ x *else* 0) ∂*lebesgue*)
by(*rule ennreal-cong*, *rule Lebesgue-Measure.integral-restrict-UNIV*[*symmetric*]) *simp*
also have ... = *ennreal* ($\int$ x. *normal-density* μ σ x $*$ *indicator* {..<μ} x ∂*lebesgue*)
by (*meson indicator-times-eq-if*(2)[*symmetric*])
also have ... = *ennreal* ($\int$ x. *normal-density* μ σ x $*$ *indicator* {..<μ} x ∂*lborel*)
by(*simp add*: *integral-completion*)
also have ... = ($\int^+$ x. *ennreal* (*normal-density* μ σ x $*$ *indicator* {..<μ} x) ∂*lborel*)
by(*auto intro*!: *nn-integral-eq-integral*[*symmetric*] *integrable-real-mult-indicator*)
also have ... = *?r*
using *AE-lborel-singleton* **by**(*fastforce intro*!: *nn-integral-cong-AE simp*: *indicator-def*)
finally show *?thesis* **.**
qed
show *?thesis*
by(*simp add*: *1 2*) (*metis* (*no-types*, *lifting*) *ennreal-divide-times mult-2 mult-2-right mult-divide-eq-ennreal one-add-one top-neq-numeral zero-neq-numeral*)
qed
also have ... = *ennreal* (*1 / 2*)
using *prob-space.emeasure-space-1*[*OF prob-space-normal-density*]
by(*simp add*: *emeasure-density divide-ennreal-def*)
finally show *?thesis* **.**
qed

corollary *measure-qbs-l-dens-to-dist-hnormal-f*: *measure* (*qbs-l* (*density-qbs lborel-qbs hnormal-f*)) *UNIV* = *1 / 2*
by(*simp add*: *measure-def nc-hnormal-f del*: *ennreal-half*)

lemma *hnormal-f*:
shows *hnormal-f* $\in$ *qbs-borel* $\to_Q$ $\mathbb{R}_Q$
and $\bigwedge x$. *hnormal-f* x $\geq$ *0*
and *qbs-l* (*density-qbs lborel-qbs hnormal-f*) *UNIV* $\neq$ *0*
and *qbs-integrable lborel-qbs hnormal-f*
using *nc-hnormal-f* **by**(*auto simp*: *qbs-integrable-iff-integrable integrable-iff-bounded qbs-l-density-qbs*[*of - qbs-borel*] *hnormal-f-def emeasure-density simp del*: *ennreal-half*)

lemma *qbs-l* (*dens-to-dist local.hnormal-f*) = *density lborel* (λx. *ennreal* (2 $*$ (*if* x

$\leq \mu$ *then 0 else normal-density* μ σ x)))
by(*simp add*: *qbs-l-dens-to-dist*[*OF hnormal-f*] *measure-qbs-l-dens-to-dist-hnormal-f*)
(*simp add*: *hnormal-f-def*)

end

5.2.7 Erlang Distribution

context
fixes k :: *nat* **and** l :: *real*
assumes *l-pos*[*arith*]: $l > 0$
begin

definition *erlang-f* $\equiv (\lambda x.$ *if* $x < 0$ *then 0 else* $x \widehat{} k *$ *exp* $(- l * x))$

lemma *nc-erlang-f*: *qbs-l* (*density-qbs lborel-qbs erlang-f*) *UNIV* = *ennreal* (*fact k* / $l \widehat{}$(*Suc k*))
proof −
have *qbs-l* (*density-qbs lborel-qbs erlang-f*) *UNIV* = $(\int^+ x.$ *ennreal* (*if* $x < 0$ *then 0 else* $x \widehat{} k *$ *exp* $(- l * x))$ ∂*lborel*)
by(*auto simp*: *qbs-l-density-qbs*[*of - qbs-borel*] *erlang-f-def emeasure-density*)
also have ... = *ennreal* (*fact k* / $l \widehat{}$(*Suc k*)) * $(\int^+ x.$ *erlang-density k l x* ∂*lborel*)
by(*auto simp*: *nn-integral-cmult*[*symmetric*] *ennreal-mult'*[*symmetric*] *erlang-density-def intro*!: *nn-integral-cong*)
also have ... = *ennreal* (*fact k* / $l \widehat{}$(*Suc k*))
using *prob-space.emeasure-space-1*[*OF prob-space-erlang-density*]
by(*simp add*: *emeasure-density*)
finally show *?thesis* **.**
qed

corollary *measure-qbs-l-dens-to-dist-erlang-f*: *measure* (*qbs-l* (*density-qbs lborel-qbs erlang-f*)) *UNIV* = *fact k* / $l \widehat{}$(*Suc k*)
by(*simp add*: *measure-def nc-erlang-f*)

lemma *erlang-f*:
shows *erlang-f* $\in$ *qbs-borel* $\to_Q \mathbb{R}_Q$
and $\bigwedge x.$ *erlang-f x* ≥ 0
and *qbs-l* (*density-qbs lborel-qbs erlang-f*) *UNIV* $\neq 0$
and *qbs-integrable lborel-qbs erlang-f*
using *nc-erlang-f* **by**(*auto simp*: *qbs-integrable-iff-integrable integrable-iff-bounded qbs-l-density-qbs*[*of - qbs-borel*] *erlang-f-def emeasure-density*)

lemma *qbs-l* (*dens-to-dist erlang-f*) = *density lborel* (*erlang-density k l*)
proof −
have [*simp*]: $l * l \widehat{} k *$ (*if* $x < 0$ *then 0 else* $x \widehat{} k *$ *exp* $(- l * x))$ / *fact k* = (*if* $x < 0$ *then 0 else* $l \widehat{}$ *Suc k* * $x \widehat{} k *$ *exp* $(- l * x)$ / *fact k*) **for** x
by *auto*
show *?thesis*
by(*simp add*: *qbs-l-dens-to-dist*[*OF erlang-f*] *measure-qbs-l-dens-to-dist-erlang-f*

erlang-density-def) (*simp add*: *erlang-f-def*)
qed

end

5.2.8 Uniform Distribution on $(0,1) \times (0,1)$.

definition *uniform-f* ≡ *indicat-real* ({*0*<..<*1*::*real*}×{*0*<..<*1*::*real*})

lemma
shows *uniform-f-qbs'*[*qbs*]: *uniform-f* ∈ *qbs-borel* →$_Q$ $\mathbb{R}_Q$
and *uniform-f-qbs*[*qbs*]: *uniform-f* ∈ $\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$ →$_Q$ $\mathbb{R}_Q$
proof −
have *uniform-f* ∈ $\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$ →$_Q$ $\mathbb{R}_Q$
by(*auto simp*: *uniform-f-def r-preserves-product*[*symmetric*] *intro*!: *rr.qbs-morphism-measurable-intro*)
thus *uniform-f* ∈ $\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$ →$_Q$ $\mathbb{R}_Q$ *uniform-f* ∈ *qbs-borel* →$_Q$ $\mathbb{R}_Q$
by(*simp-all add*: *qbs-borel-prod*)
qed

lemma *uniform-f-measurable*[*measurable*]: *uniform-f* ∈ *borel-measurable borel*
by (*metis borel-prod rr.standard-borel-axioms standard-borel.standard-borel-r-full-faithful uniform-f-qbs'*)

lemma *nc-uniform-f*: *qbs-l* (*density-qbs lborel-qbs uniform-f*) *UNIV* = *1*
proof −
have *qbs-l* (*density-qbs lborel-qbs uniform-f*) *UNIV* = (∫$^+$ *z*. *ennreal* (*uniform-f z*) *∂lborel*)
by(*auto simp*: *qbs-l-density-qbs*[*of - qbs-borel*] *emeasure-density*)
also have ... = (∫$^+$ *z*. *indicator* {*0*<..<*1*::*real*} (*fst z*) ∗ *indicator* {*0*<..<*1*::*real*} (*snd z*) *∂*(*lborel* $\bigotimes_M$ *lborel*))
by(*auto simp*: *lborel-prod intro*!: *nn-integral-cong*) (*auto simp*: *indicator-def uniform-f-def*)
also have ... = *1*
by(*auto simp*: *lborel.nn-integral-fst*[*symmetric*] *nn-integral-cmult*)
finally show *?thesis* **.**
qed

corollary *measure-qbs-l-dens-to-dist-uniform-f*: *measure* (*qbs-l* (*density-qbs lborel-qbs uniform-f*)) *UNIV* = *1*
by(*simp add*: *measure-def nc-uniform-f*)

lemma *uniform-f*:
shows *uniform-f* ∈ *qbs-borel* →$_Q$ $\mathbb{R}_Q$
and ⋀*x*. *uniform-f x* ≥ *0*
and *qbs-l* (*density-qbs lborel-qbs uniform-f*) *UNIV* ≠ *0*
and *qbs-integrable lborel-qbs uniform-f*
using *nc-uniform-f* **by**(*auto simp*: *qbs-integrable-iff-integrable integrable-iff-bounded qbs-l-density-qbs*[*of - qbs-borel*] *emeasure-density*) (*auto simp*: *uniform-f-def*)

lemma *qbs-l-dens-to-dist-uniform-f*:*qbs-l* (*dens-to-dist uniform-f*) = *density lborel* (λ*x*. *ennreal* (*uniform-f x*))
by(*simp add*: *qbs-l-dens-to-dist*[*OF uniform-f*,*simplified measure-qbs-l-dens-to-dist-uniform-f*])

lemma *dens-to-dist uniform-f* = *Uniform 0 1* $\bigotimes_{Qmes}$ *Uniform 0 1*
proof −
note *qbs-pair-measure-morphismP*[*qbs*] *Uniform-qbsP*[*qbs*]
have [*simp*]:*sets* (*borel* :: (*real* × *real*) *measure*) = *sets* (*borel* $\bigotimes_M$ *borel*)
by(*metis borel-prod*)
show *?thesis*
proof(*safe intro*!: *inj-onD*[*OF qbs-l-inj*[*of* $\mathbb{R}_Q$ $\bigotimes_Q$ $\mathbb{R}_Q$]] *qbs-space-monadPM measure-eqI*)

fix *A* :: (*real* × *real*) *set*
assume *A* ∈ *sets* (*qbs-l* (*dens-to-dist uniform-f*))
then have [*measurable*]: *A* ∈ *sets* (*borel* $\bigotimes_M$ *borel*)
by(*auto simp*: *qbs-l-dens-to-dist-uniform-f*)
show *emeasure* (*qbs-l* (*dens-to-dist uniform-f*)) *A* = *emeasure* (*qbs-l* (*Uniform 0 1* $\bigotimes_{Qmes}$ *Uniform 0 1*)) *A* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∫$^+$*x*∈*A*. *ennreal* (*uniform-f x*) ∂(*lborel* $\bigotimes_M$ *lborel*))
by(*simp add*: *emeasure-density lborel-prod qbs-l-dens-to-dist-uniform-f*)
also have ... = (∫$^+$*x*. *indicator A x* ∗ *indicator* {*0*<..<*1*} (*fst x*) ∗ *indicator* {*0*<..<*1*} (*snd x*) ∂(*lborel* $\bigotimes_M$ *lborel*))
by(*auto intro*!: *nn-integral-cong*) (*auto simp*: *indicator-def uniform-f-def*)
also have ... = (∫$^+$ *x*∈{*0*<..<*1*}. (∫$^+$*y*∈{*0*<..<*1*}. *indicator A* (*x*, *y*) ∂*lborel*) ∂*lborel*)
by(*auto simp add*: *lborel.nn-integral-fst*[*symmetric*] *intro*!: *nn-integral-cong*) (*auto simp*: *indicator-def*)
also have ... = (∫$^+$ *x*. (∫$^+$*y*. *indicator A* (*x*, *y*) ∂*uniform-measure lborel* {*0*<..<*1*}) ∂*uniform-measure lborel* {*0*<..<*1*})
by(*auto simp*: *nn-integral-uniform-measure divide-ennreal-def*)
also have ... = *?rhs*
by(*auto simp*: *UniformP-pair.M1.emeasure-pair-measure′ qbs-l-Uniform-pair*)
finally show *?thesis* **.**
qed
next
show *dens-to-dist uniform-f* ∈ *qbs-space* (*monadP-qbs* ($\mathbb{R}_Q$ $\bigotimes_Q$ $\mathbb{R}_Q$))
by(*simp add*: *dens-to-dist-prob*[*OF uniform-f*] *qbs-borel-prod*)
qed (*auto simp*: *qbs-l-dens-to-dist-uniform-f qbs-l-Uniform-pair*, *qbs*, *simp*)
qed

5.2.9 If then else

definition *gt* :: (*real* ⇒ *real*) ⇒ *real* ⇒ *bool qbs-measure* **where**
gt ≡ (λ*f r*. *do* {
x ← *dens-to-dist* (*normal-f 0 1*);
if f x > *r*
then return-qbs $\mathbb{B}_Q$ *True*

else return-qbs $\mathbb{B}_Q$ *False*
})

declare *normal-f*(*1*)[*of 1 0*,*simplified*]

lemma *gt-qbs*[*qbs*]: *gt* ∈ *qbs-space* ((**R**$_Q$ ⇒$_Q$ **R**$_Q$) ⇒$_Q$ **R**$_Q$ ⇒$_Q$ *monadP-qbs* $\mathbb{B}_Q$)
proof −
note [*qbs*] = *dens-to-dist-prob*[*OF normal-f*[*of 1 0*,*simplified*]] *bind-qbs-morphismP return-qbs-morphismP*
show *?thesis*
by(*simp add*: *gt-def*)
qed

lemma
assumes [*qbs*]: *f* ∈ **R**$_Q$ →$_Q$ **R**$_Q$
shows $\mathcal{P}$(*b in gt f r*. *b* = *True*) = $\mathcal{P}$(*x in std-normal-distribution*. *f x* > *r*) (**is** *?P1* = *?P2*)
proof −
note [*qbs*] = *dens-to-dist-prob*[*OF normal-f*[*of 1 0*,*simplified*]] *bind-qbs-morphismP return-qbs-morphismP*
have *1*[*simp*]: *space* (*qbs-l* (*gt f r*)) = *UNIV*
by(*simp add*: *space-qbs-l-in*[*OF qbs-space-monadPM*,*of* - $\mathbb{B}_Q$])
have *?P1* = (∫ *b*. *indicat-real* {*True*} *b* ∂*qbs-l* (*gt f r*))
by *simp* (*metis* (*full-types*) *Collect-cong singleton-conv2*)
also have ... = (∫$_Q$ *b*. *indicat-real* {*True*} *b* ∂(*gt f r*))
by(*simp add*: *qbs-integral-def2-l*)
also have ... = (∫$_Q$ *b*. *indicat-real* {*True*} *b* ∂(*dens-to-dist* (*normal-f 0 1*) ⤜ (λ*x*. *return-qbs* $\mathbb{B}_Q$ (*f x* > *r*))))
proof −
have [*simp*]:*gt f r* = *dens-to-dist* (*normal-f 0 1*) ⤜ (λ*x*. *return-qbs* $\mathbb{B}_Q$ (*f x* > *r*))
by(*auto simp*: *gt-def intro*!: *bind-qbs-cong*[*of* - **R**$_Q$ - - $\mathbb{B}_Q$] *qbs-space-monadPM qbs-morphism-monadPD*)
show *?thesis* **by** *simp*
qed
also have ... = (∫$_Q$ *x*. (*indicat-real* {*True*} ∘ (λ*x*. *f x* > *r*)) *x* ∂*dens-to-dist* (*normal-f 0 1*))
by(*rule qbs-integral-bind-return*[*of* - **R**$_Q$]) (*auto intro*!: *qbs-space-monadPM*)
also have ... = (∫$_Q$ *x*. *indicat-real* {*x*. *f x* > *r*} *x* ∂*dens-to-dist* (*normal-f 0 1*))
by(*auto intro*!: *qbs-integral-cong*[*of* - **R**$_Q$] *qbs-space-monadPM simp*: *indicator-def*)
also have ... = (∫ *x*. *indicat-real* {*x*. *f x* > *r*} *x* ∂*dens-to-dist* (*normal-f 0 1*))
by(*simp add*: *qbs-integral-def2-l*)
also have ... = *?P2*
by(*simp add*: *qbs-l-densto-dist-normal-f*[*of 1 0*])
finally show *?thesis* **.**
qed

Examples from Staton [5, Sect. 2.2].

5.2.10 Weekend

Example from Staton [5, Sect. 2.2.1].

This example is formalized in Coq by Affeldt et al. [1].

```
definition weekend :: bool qbs-measure where
weekend ≡ do {
          let x = qbs-bernoulli (2 / 7);
              f = (λx. let r = if x then 3 else 10 in pmf (poisson-pmf r) 4)
          in query x f
           }

lemma weekend-qbs[qbs]:weekend ∈ qbs-space (monadM-qbs 𝔹_Q)
  by(simp add: weekend-def)

lemma weekend-nc:
  defines N ≡ 2 / 7 * pmf (poisson-pmf 3) 4  + 5 / 7 *  pmf (poisson-pmf 10)
4
 shows qbs-l (density-qbs (bernoulli-pmf (2/7)) (λx. (pmf (poisson-pmf (if x then
3 else 10)) 4))) UNIV = N
proof −
  have [simp]:fact 4 = 4 * fact 3
    by (simp add: fact-numeral)
  show ?thesis
   by(simp add: qbs-l-density-qbs[of - 𝔹_Q] emeasure-density ennreal-plus[symmetric]
ennreal-mult'[symmetric] N-def del: ennreal-plus)
qed

lemma qbs-l-weekend:
  defines N ≡ 2 / 7 * pmf (poisson-pmf 3) 4  + 5 / 7 *  pmf (poisson-pmf 10)
4
  shows qbs-l weekend = qbs-l (density-qbs (qbs-bernoulli (2 / 7)) (λx. ennreal
(let r = if x then 3 else 10 in r ^ 4 * exp (− r) / (fact 4 * N)))) (is ?lhs = ?rhs)
proof −
  have [simp]: N > 0
    by(auto simp: N-def intro!: add-pos-pos)
  have ?lhs = qbs-l (density-qbs (density-qbs (qbs-bernoulli (2 / 7)) (λx. ennreal
(let r = if x then 3 else 10 in r ^ 4 * exp (− r) / fact 4))) (λx. 1 /  ennreal N))
   using normalize-qbs[of density-qbs (qbs-bernoulli (2/7)) (λx. (pmf (poisson-pmf
(if x then 3 else 10)) 4)) 𝔹_Q,simplified] weekend-nc
    by(simp add: weekend-def query-def N-def Let-def)
  also have ... = ?rhs
    by(simp add: density-qbs-density-qbs-eq[of - 𝔹_Q] ennreal-mult'[symmetric] en-
nreal-1[symmetric] divide-ennreal del: ennreal-1) (metis (mono-tags, opaque-lifting)
divide-divide-eq-left)
  finally show ?thesis .
qed

lemma
```

defines $N \equiv 2 / 7 * pmf\ (poisson\text{-}pmf\ 3)\ 4\ +\ 5 / 7 *\ pmf\ (poisson\text{-}pmf\ 10)\ 4$
shows $\mathcal{P}(b\ in\ weekend.\ b = True) = 2 / 7 * (3\hat{}4 * exp\ (-\ 3)) / fact\ 4 * 1 / N$
by *simp* (*simp add*: *qbs-l-weekend measure-def qbs-l-density-qbs*[*of* - $\mathbb{B}_Q$] *emeasure-density emeasure-measure-pmf-finite ennreal-mult'*[*symmetric*] *N-def*)

5.2.11 Whattime

Example from Staton [5, Sect. 2.2.3]

f is given as a parameter.

definition *whattime* :: (*real* ⇒ *real*) ⇒ *real qbs-measure* **where**
whattime ≡ (λ*f*. *do* {
let T = Uniform 0 24 in
query T (λ*t*. *let r = f t in*
exponential-density r (*1 / 60*))
})

lemma *whattime-qbs*[*qbs*]: *whattime* ∈ ($\mathbb{R}_Q$ ⇒$_Q$ $\mathbb{R}_Q$) ⇒$_Q$ *monadM-qbs* $\mathbb{R}_Q$
by(*simp add*: *whattime-def*)

lemma *qbs-l-whattime-sub*:
assumes [*qbs*]: *f* ∈ $\mathbb{R}_Q$ ⇒$_Q$ $\mathbb{R}_Q$
shows *qbs-l* (*density-qbs* (*Uniform 0 24*) (λ*x*. *exponential-density* (*f x*) (*1 / 60*)))
= *density lborel* (λ*x*. *indicator* {*0*<..<*24*} *x / 24 * exponential-density* (*f x*) (*1 / 60*))
proof −
have [*measurable*]:*f* ∈ *borel-measurable borel*
by (*simp add*: *standard-borel.standard-borel-r-full-faithful standard-borel-ne.standard-borel*)
have [*measurable*]: (λ*x*. (*exponential-density* (*f x*) (*1 / 60*))) ∈ *borel-measurable borel*
by(*simp add*: *exponential-density-def*)
have *1*[*measurable*]: (λ*x*. *ennreal* (*exponential-density* (*f x*) (*1 / 60*))) ∈ *borel-measurable* (*uniform-measure lborel* {*0*<..<*24*})
by(*simp add*: *measurable-cong-sets*[*OF sets-uniform-measure*])
show *?thesis*
by(*auto simp*: *qbs-l-density-qbs*[*of* - *qbs-borel*] *emeasure-density emeasure-density*[*OF 1*] *nn-integral-uniform-measure nn-integral-divide*[*symmetric*] *ennreal-mult' divide-ennreal*[*symmetric*] *intro*!: *measure-eqI nn-integral-cong simp del*: *times-divide-eq-left*)
(*simp add*: *ennreal-indicator ennreal-times-divide mult.commute mult.left-commute*)
qed

lemma
assumes [*qbs*]: *f* ∈ $\mathbb{R}_Q$ ⇒$_Q$ $\mathbb{R}_Q$ **and** [*measurable*]:*U* ∈ *sets borel*
and ⋀*r*. *f r* ≥ *0*
defines *N* ≡ (∫ *t*∈{*0*<..<*24*}. (*f t * exp* (− *1/ 60 * f t*)) ∂*lborel*)
defines *N'* ≡ (∫$^+$ *t*∈{*0*<..<*24*}. (*f t * exp* (− *1/ 60 * f t*)) ∂*lborel*)
assumes *N'* ≠ *0* **and** *N'* ≠ ∞
shows $\mathcal{P}$(*t in whattime f*. *t* ∈ *U*) = (∫ *t*∈{*0*<..<*24*}∩*U*. (*f t * exp* (− *1/ 60 **

f t)) *∂lborel*) / *N*
proof −
have *1*: *space* (*whattime f*) = *UNIV*
by (*rule space-qbs-l-in*[*of whattime f* $\mathbb{R}_Q$,*simplified qbs-space-qbs-borel*]) *simp*
have [*measurable*]: *f* ∈ *borel-measurable borel*
by (*simp add*: *standard-borel.standard-borel-r-full-faithful standard-borel-ne.standard-borel*)
have [*measurable*]: (λ*x*. *exponential-density* (*f x*) (*1* / *60*)) ∈ *borel-measurable borel*
by(*simp add*: *measurable-cong-sets*[*OF sets-uniform-measure*] *exponential-density-def*)
have [*measurable*]: (λ*x*. *ennreal* (*exponential-density* (*f x*) (*1* / *60*))) ∈ *borel-measurable* (*uniform-measure lborel* {*0*<..<*24*})
by(*simp add*: *measurable-cong-sets*[*OF sets-uniform-measure*])
have *qbs-ld*: *qbs-l* (*density-qbs* (*Uniform 0 24*) (λ*x*. *exponential-density* (*f x*) (*1* / *60*))) *UNIV* = (∫$^+$*x*∈{*0*<..<*24*}. *ennreal* (*f x* ∗ *exp* (− *1*/ *60* ∗ *f x*) / *24*) *∂lborel*)
by(*auto simp*: *qbs-l-whattime-sub emeasure-density intro*!: *nn-integral-cong*,*auto simp*: *ennreal-indicator*[*symmetric*] *ennreal-mult''*[*symmetric*] *exponential-density-def*) (*simp add*: *mult.commute*)
have *int*: *integrable lborel* (λ*x*. *f x* ∗ *exp* (− *1*/ *60* ∗ *f x*) ∗ *indicat-real* {*0*<..<*24*} *x*)
using *assms*(*3*,*7*) **by**(*simp add*: *N'-def integrable-iff-bounded ennreal-mult'' ennreal-indicator top.not-eq-extremum*)

have *ge*: (∫ *x*∈{*0*<..<*24*}. (*f x* ∗ *exp* (− (*f x* / *60*)) / *24*)*∂lborel*) > *0*
proof −
have (∫ *x*∈{*0*<..<*24*}. (*f x* ∗ *exp* (− (*f x* / *60*))) *∂lborel*) > *0* (**is** *?l* > *0*)
proof −
have *ennreal ?l* = (∫$^+$*x*. (*indicator* {*0*<..<*24*} *x* ∗ (*f x* ∗ *exp* (− (*f x* / *60*)))) *∂lborel*)
unfolding *set-lebesgue-integral-def* **by**(*simp*,*rule nn-integral-eq-integral*[*symmetric*]) (*insert int assms*(*3*),*auto simp*: *mult.commute*)
also have ... = (∫$^+$*x*∈{*0*<..<*24*}. *ennreal* (*f x* ∗ *exp* (− *1*/ *60* ∗ *f x*)) *∂lborel*)
by (*simp add*: *indicator-mult-ennreal mult.commute*)
also have ... > *0*
using *assms*(*6*) *not-gr-zero N'-def* **by** *blast*
finally show *?thesis*
using *ennreal-less-zero-iff* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
have *ge2*: (∫ *x*∈{*0*<..<*24*}∩ *U*. (*exponential-density* (*f x*) (*1* / *60*)) *∂lborel*) ≥ *0*
using *assms*(*3*) **by**(*auto intro*!: *integral-nonneg-AE simp*: *set-lebesgue-integral-def*)

have (∫$^+$*x*∈{*0*<..<*24*}. *ennreal* (*f x* ∗ *exp* (− *1*/ *60* ∗ *f x*) / *24*) *∂lborel*) ≠ *0* ∧ (∫$^+$*x*∈{*0*<..<*24*}. *ennreal* (*f x* ∗ *exp* (− *1*/ *60* ∗ *f x*) / *24*) *∂lborel*) ≠ ∞
proof −
have (∫$^+$*x*∈{*0*<..<*24*}. *ennreal* (*f x* ∗ *exp* (− *1*/ *60* ∗ *f x*) / *24*) *∂lborel*) = (∫$^+$*x*. *ennreal* (*f x* ∗ *exp* (− *1*/ *60* ∗ *f x*)) ∗ *indicator* {*0*<..<*24*} *x* / *24* *∂lborel*)

by(*rule nn-integral-cong*, *insert assms*(*3*)) (*auto simp*: *divide-ennreal*[*symmetric*] *ennreal-times-divide mult.commute*)
also have ... = ($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− *1* / *60* * *f x*)) *∂lborel*) / *24*
by(*simp add*: *nn-integral-divide*)
finally show *?thesis*
using *assms*(*5*,*6*,*7*) **by** (*simp add*: *ennreal-divide-eq-top-iff*)
qed
hence *normalize-qbs* (*density-qbs* (*Uniform 0 24*) (λ*x*. (*exponential-density* (*f x*) (*1* / *60*)))) = *density-qbs* (*density-qbs* (*Uniform 0 24*) (λ*x*. *ennreal* (*exponential-density* (*f x*) (*1* / *60*)))) (λ*x*. *1* / ($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− *1* / *60* * *f x*) / *24*) *∂lborel*))
using *normalize-qbs*[*of density-qbs* (*Uniform 0 24*) (λ*x*. *exponential-density* (*f x*) (*1* / *60*)) *qbs-borel*,*simplified*] **by**(*simp add*: *qbs-ld*)
also have ... = *density-qbs* (*Uniform 0 24*) (λ*x*. *ennreal* (*exponential-density* (*f x*) (*1* / *60*)) / ($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− (*f x* / *60*)) / *24*) *∂lborel*))
by(*simp add*: *density-qbs-density-qbs-eq*[*of* - *qbs-borel*] *ennreal-times-divide*)

finally have $\mathcal{P}$(*x in whattime f*. *x* ∈ *U*) = *measure* (*density* (*qbs-l* (*Uniform 0 24*)) (λ*x*. *ennreal* (*exponential-density* (*f x*) (*1* / *60*)) / ($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− (*f x* / *60*)) / *24*) *∂lborel*))) *U*
unfolding *1* **by** (*simp add*: *whattime-def query-def qbs-l-density-qbs*[*of* - *qbs-borel*])
also have ... = *enn2real* (($\int^+ x \in \{0<..<24\}$. (*ennreal* (*exponential-density* (*f x*) (*1* / *60*)) / ($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− (*f x* / *60*)) / *24*)*∂lborel*) * *indicator U x*) *∂lborel*) / *24*)
by(*simp add*: *measure-def emeasure-density nn-integral-uniform-measure*)
also have ... = *enn2real* (($\int^+ x \in \{0<..<24\}$. (*ennreal* (*exponential-density* (*f x*) (*1* / *60*)) * *indicator U x*) *∂lborel*) / ($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− (*f x* / *60*)) / *24*)*∂lborel*) / *24*)
by(*simp add*: *ennreal-divide-times ennreal-times-divide nn-integral-divide*)
also have ... = *enn2real* (*ennreal* ($\int x \in \{0<..<24\} \cap U$. (*exponential-density* (*f x*) (*1* / *60*)) *∂lborel*) / *ennreal* ($\int x \in \{0<..<24\}$. (*f x* * *exp* (− (*f x* / *60*)) / *24*)*∂lborel*) / *ennreal 24*)
proof −
have *1*:($\int^+ x \in \{0<..<24\}$. *ennreal* (*f x* * *exp* (− (*f x* / *60*)) / *24*)*∂lborel*) = *ennreal* ($\int x \in \{0<..<24\}$. (*f x* * *exp* (− (*f x* / *60*)) / *24*)*∂lborel*) (**is** *?l* = *?r*)
proof −
have *?l* = ($\int^+ x$. *ennreal* (*f x* * *exp* (− (*f x* / *60*)) / *24* * *indicat-real* {*0*<..<*24*} *x*) *∂lborel*)
by (*simp add*: *nn-integral-set-ennreal*)
also have ... = *ennreal* ($\int x$. (*f x* * *exp* (− (*f x* / *60*)) / *24* * *indicat-real* {*0*<..<*24*} *x*)*∂lborel*)
by(*rule nn-integral-eq-integral*) (*use int assms*(*3*) **in** *auto*)
also have ... = *?r*
by(*auto simp*: *set-lebesgue-integral-def intro*!: *Bochner-Integration.integral-cong ennreal-cong*)
finally show *?thesis* **.**
qed
have *2*:($\int^+ x \in \{0<..<24\}$. (*ennreal* (*exponential-density* (*f x*) (*1* / *60*)) *

*indicator U x) ∂lborel) = ennreal (∫ x∈{0<..<24}∩ U. (exponential-density (f x) (1 / 60)) ∂lborel) (***is** *?l = ?r)*
proof −
have *?l = (∫⁺x. ennreal (f x ∗ exp (− (f x / 60)) ∗ indicat-real {0<..<24} x ∗ indicator U x) ∂lborel)*
by *(auto intro*!*: nn-integral-cong simp: exponential-density-def indicator-def)*
also have *... = ennreal (∫ x. (f x ∗ exp (− (f x / 60)) ∗ indicat-real {0<..<24} x ∗ indicator U x)∂lborel)*
by*(rule nn-integral-eq-integral) (use integrable-real-mult-indicator[OF - int] assms(3) **in** auto)*
also have *... = ?r*
by*(auto simp: set-lebesgue-integral-def indicator-def exponential-density-def intro*!*: Bochner-Integration.integral-cong ennreal-cong)*
finally show *?thesis* **.**
qed
show *?thesis*
by*(simp add: 1 2)*
qed
also have *... = enn2real (ennreal ((∫ x∈{0<..<24}∩ U. (exponential-density (f x) (1 / 60)) ∂lborel) / (∫ x∈{0<..<24}. (f x ∗ exp (− (f x / 60)) / 24)∂lborel) / 24))*
by*(simp only: divide-ennreal[OF ge2 ge] divide-ennreal[OF divide-nonneg-pos[OF ge2 ge]])*
also have *... = (∫ x∈{0<..<24}∩ U. (exponential-density (f x) (1 / 60)) ∂lborel) / (∫ x∈{0<..<24}. (f x ∗ exp (− (f x / 60)) / 24)∂lborel) / 24*
by*(rule enn2real-ennreal) (use ge ge2 **in** auto)*
also have *... = (∫ x∈{0<..<24}∩U. (f x ∗ exp (− 1/ 60 ∗ f x)) ∂lborel) / N*
by*(auto simp: N-def exponential-density-def)*
finally show *?thesis* **.**
qed

5.2.12 Distributions on Functions

definition *a-times-x :: (real ⇒ real) qbs-measure* **where**
a-times-x ≡ do {
a ← Uniform (−2) 2;
return-qbs (ℝ_Q ⇒_Q ℝ_Q) (λx. a ∗ x)
}

lemma *a-times-x-qbs[qbs]: a-times-x ∈ monadM-qbs (ℝ_Q ⇒_Q ℝ_Q)*
by*(simp add: a-times-x-def)*

lemma *a-times-x-qbsP: a-times-x ∈ monadP-qbs (ℝ_Q ⇒_Q ℝ_Q)*
proof −
note *[qbs] = Uniform-qbsP[of −2 2,simplified] return-qbs-morphismP bind-qbs-morphismP*
show *?thesis*
by*(simp add: a-times-x-def)*
qed

definition *a-times-x′* :: (*real* ⇒ *real*) *qbs-measure* **where**
a-times-x′ ≡ *do* {
condition a-times-x (λ*f*. *f* 1 ≥ 0)
}

lemma *a-times-x′-qbs*[*qbs*]: *a-times-x′* ∈ *monadM-qbs* (ℝ$_Q$ ⇒$_Q$ ℝ$_Q$)
by(*simp add*: *a-times-x′-def*)

lemma *prob-a-times-x*:
assumes [*measurable*]: *Measurable.pred borel P*
shows 𝒫(*f in a-times-x*. *P* (*f r*)) = 𝒫(*a in Uniform* (−2) 2. *P* (*a* ∗ *r*)) (**is** *?lhs* = *?rhs*)
proof −
have [*qbs*]: *qbs-pred qbs-borel P*
using *r-preserves-morphisms* **by** *fastforce*
have *?lhs* = *measure a-times-x* ({*f*. *P* (*f r*)} ∩ *space a-times-x*)
by (*simp add*: *Collect-conj-eq inf-sup-aci*(*1*))
also have ... = (∫ $_Q$ *f*. *indicat-real* {*f*. *P* (*f r*)} *f* ∂*a-times-x*)
by(*simp add*: *qbs-integral-def2-l*)
also have ... = *qbs-integral* (*Uniform* (− 2) 2) (*indicat-real* {*f*. *P* (*f r*)} ∘ (∗))
unfolding *a-times-x-def* **by**(*rule qbs-integral-bind-return*[*of* - *qbs-borel*]) *auto*
also have ... = (∫ $_Q$ *a*. *indicat-real* {*a*. *P* (*a* ∗ *r*)} *a* ∂*Uniform* (− 2) 2)
by(*auto simp*: *comp-def indicator-def*)
also have ... = *?rhs*
by (*simp add*: *qbs-integral-def2-l*)
finally show *?thesis* **.**
qed

lemma 𝒫(*f in a-times-x′*. *f* 1 ≥ 1) = 1 / 2 (**is** *?P* = -)
proof −
have *?P* = 𝒫(*f in a-times-x*. *f* 1 ≥ 1 | *f* 1 ≥ 0)
by(*simp add*: *query-Bayes*[*OF a-times-x-qbsP*] *a-times-x′-def*)
also have ... = 𝒫(*f in a-times-x*. *f* 1 ≥ 1) / 𝒫(*f in a-times-x*. *f* 1 ≥ 0)
by(*auto simp add*: *cond-prob-def*) (*meson dual-order.trans linordered-nonzero-semiring-class.zero-le-one*)
also have ... = 1 / 2
proof −
have [*simp*]: {−2<..<2::*real*} ∩ *Collect* ((≤) 1) = {1..<2} {−2<..<2::*real*} ∩ *Collect* ((≤) 0) = {0..<2}
by *auto*
show *?thesis*
by(*auto simp*: *prob-a-times-x*)
qed
finally show *?thesis* **.**
qed

Almost everywhere, integrable, and integrations are also interpreted as programs.

lemma (λ*g f x*. *if* (*AE*$_Q$ *y in g x*. *f x y* ≠ ∞) *then* (∫^+_Q *y*. *f x y* ∂(*g x*)) *else* 0)
∈ (ℝ$_Q$ ⇒$_Q$ *monadM-qbs* ℝ$_Q$) ⇒$_Q$ (ℝ$_Q$ ⇒$_Q$ ℝ$_Q$ ⇒$_Q$ ℝ$_{Q≥0}$) ⇒$_Q$ ℝ$_Q$ ⇒$_Q$

$\mathbb{R}_{Q \geq 0}$
by *simp*

lemma ($\lambda g\ f\ x.$ *if qbs-integrable* $(g\ x)$ $(f\ x)$ *then Some* $(\int_Q\ y.\ f\ x\ y\ \partial(g\ x))$ *else None*)
$\in (\mathbb{R}_Q \Rightarrow_Q$ *monadM-qbs* $\mathbb{R}_Q) \Rightarrow_Q (\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q) \Rightarrow_Q \mathbb{R}_Q \Rightarrow_Q$ *option-qbs* $\mathbb{R}_Q$
by *simp*

end

References

[1] R. Affeldt, C. Cohen, and A. Saito. Semantics of probabilistic programs using s-finite kernels in Coq. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, page 316, New York, NY, USA, 2023. Association for Computing Machinery.

[2] A. Sampson. Probabilistic programming. http://adriansampson.net/doc/ppl.html. Accessed: January 25. 2023.

[3] T. Sato, A. Aguirre, G. Barthe, M. Gaboardi, D. Garg, and J. Hsu. Formal verification of higher-order probabilistic programs: reasoning about approximation, convergence, bayesian inference, and optimization. *Proceedings of the ACM on Programming Languages*, 3(POPL):130, Jan 2019.

[4] S. Staton. Commutative semantics for probabilistic programming. In H. Yang, editor, *Programming Languages and Systems*, pages 855–879, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[5] S. Staton. *Probabilistic Programs as Measures*, page 4374. Cambridge University Press, 2020.

[6] H. Yang. Semantics of higher-order probabilistic programs with continuous distributions. https://alfa.di.uminho.pt/~nevrenato/probprogschool_slides/Hongseok.pdf. Accessed: February 8. 2023.