Quasi-Borel Spaces

Michikazu Hirata, Yasuhiko Minamide, Tetsuya Sato

March 17, 2025

Abstract

The notion of quasi-Borel spaces was introduced by Heunen et al. [1]. The theory provides a suitable denotational model for higher-order probabilistic programming languages with continuous distributions.

This entry is a formalization of the theory of quasi-Borel spaces, including construction of quasi-Borel spaces (product, coproduct, function spaces), the adjunction between the category of measurable spaces and the category of quasi-Borel spaces, and the probability monad on quasi-Borel spaces. This entry also contains the formalization of the Bayesian regression presented in the work of Heunen et al.

This work is a part of the work by same authors, *Program Logic for Higher-Order Probabilistic Programs in Isabelle/HOL*, which will be published in proceedings of the 16th International Symposium on Functional and Logic Programming (FLOPS 2022).

Contents

1 Standard Borel Spaces

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

A standard Borel space is the Borel space associated with a Polish space. Here, we define standard Borel spaces in another, but equivallent, way. See [1] Proposition 5.

abbreviation *real-borel* ≡ *borel* :: *real measure*
abbreviation *nat-borel* ≡ *borel* :: *nat measure*
abbreviation *ennreal-borel* ≡ *borel* :: *ennreal measure*
abbreviation *bool-borel* ≡ *borel* :: *bool measure*

1.1 Definition

locale *standard-borel* =
  fixes M :: *'a measure*
  assumes *exist-fg*: $\exists f \in M \rightarrow_M$ *real-borel*. $\exists g \in$ *real-borel* $\rightarrow_M M$.
    $\forall x \in$ *space* M. $(g \circ f)\ x = x$
begin

abbreviation *fg* ≡ (*SOME k*. (*fst k*) $\in M \rightarrow_M$ *real-borel* ∧
  (*snd k*) $\in$ *real-borel* $\rightarrow_M M$ ∧
  ($\forall x \in$ *space* M. ((*snd k*) ∘ (*fst k*)) $x = x$))

definition f ≡ (*fst fg*)
definition g ≡ (*snd fg*)

lemma
  shows *f-meas*[*simp*,*measurable*] : $f \in M \rightarrow_M$ *real-borel*
   and *g-meas*[*simp*,*measurable*] : $g \in$ *real-borel* $\rightarrow_M M$
   and *gf-comp-id*[*simp*]: $\bigwedge x.\ x \in$ *space* $M \Longrightarrow (g \circ f)\ x = x$
    $\bigwedge x.\ x \in$ *space* $M \Longrightarrow g\ (f\ x) = x$
proof −
  obtain f' g' **where** h:
   $f' \in M \rightarrow_M$ *real-borel* $g' \in$ *real-borel* $\rightarrow_M M$ $\forall x \in$ *space* M. $(g' \circ f')\ x = x$
   using *exist-fg* **by** *blast*
  have $f \in$ *borel-measurable* M ∧ $g \in$ *real-borel* $\rightarrow_M M$ ∧ ($\forall x\in$*space* M. $(g \circ f)$ $x = x$)
   unfolding *f-def g-def*
   by(*rule someI2*[**where** *a*=(f',g')]) (*use h* **in** *auto*)
  thus $f \in$ *borel-measurable* M $g \in$ *real-borel* $\rightarrow_M M$
    $\bigwedge x.\ x \in$ *space* $M \Longrightarrow (g \circ f)\ x = x$ $\bigwedge x.\ x \in$ *space* $M \Longrightarrow g\ (f\ x) = x$
   by *auto*
qed

lemma *standard-borel-sets*[*simp*]:
  assumes *sets* M = *sets* Y
  shows *standard-borel* Y
  unfolding *standard-borel-def*
  using *measurable-cong-sets*[*OF assms refl*,*of real-borel*] *measurable-cong-sets*[*OF refl assms*,*of real-borel*] *sets-eq-imp-space-eq*[*OF assms*] *exist-fg*
  by *simp*

lemma *f-inj*:
  inj-on f (*space M*)
  by *standard* (*use gf-comp-id*(*2*) **in** *fastforce*)

lemma *singleton-sets*:
  assumes $x \in$ *space M*
    shows $\{x\} \in$ *sets M*
proof −
  let *?y* = *f x*
  let *?U* = *f* −‘ {*?y*}
  have *?U* ∩ *space M* ∈ *sets M*
    using *borel-measurable-vimage f-meas* **by** *blast*
  moreover have *?U* ∩ *space M* = {*x*}
    using *assms f-inj* **by**(*auto simp*:*inj-on-def*)
  ultimately show *?thesis*
    by *simp*
qed

lemma *countable-space-discrete*:
  assumes *countable* (*space M*)
  shows *sets M* = *sets* (*count-space* (*space M*))
proof
  show *sets* (*count-space* (*space M*)) ⊆ *sets M*
  proof *auto*
    fix *U*
    assume *1*:*U* ⊆ *space M*
    then have *2*:*countable U*
      using *assms countable-subset* **by** *auto*
    have *3*:*U* = ($\bigcup x \in U$. {*x*}) **by** *auto*
    moreover have ... ∈ *sets M*
      by(*rule sets.countable-UN*′′[*of U* λ*x*. {*x*}]) (*use 1 2 singleton-sets* **in** *auto*)
    ultimately show *U* ∈ *sets M*
      by *simp*
  qed
qed (*simp add*: *sets.sets-into-space subsetI*)

end

lemma *standard-borelI*:
  assumes $f \in Y \rightarrow_M$ *real-borel*
        $g \in$ *real-borel* $\rightarrow_M Y$
      and $\bigwedge y.\ y \in$ *space* $Y \Longrightarrow (g \circ f)\ y = y$
    shows *standard-borel Y*
  unfolding *standard-borel-def*
  by (*intro bexI*[*OF* - *assms*(*1*)] *bexI*[*OF* - *assms*(*2*)]) (*auto dest*: *assms*(*3*))

locale *standard-borel-space-UNIV* = *standard-borel* +
  assumes *space-UNIV*:*space M* = *UNIV*

begin

lemma *gf-comp-id′*[*simp*]:
  $g \circ f = id$ $g\ (f\ x) = x$
  using *space-UNIV gf-comp-id*
  by(*simp-all add*: *id-def comp-def*)

lemma *f-inj′*:
  inj f
  using *f-inj* **by**(*simp add*: *space-UNIV*)

lemma *g-surj′*:
  surj g
  using *gf-comp-id′*(*2*) *surjI* **by** *blast*

end

lemma *standard-borel-space-UNIVI*:
  assumes $f \in Y \to_M$ *real-borel*
    $g \in$ *real-borel* $\to_M Y$
    $(g \circ f) = id$
   and *space* $Y = UNIV$
  shows *standard-borel-space-UNIV Y*
  using *assms*
 by(*auto intro*!: *standard-borelI simp*: *standard-borel-space-UNIV-def standard-borel-space-UNIV-axioms-def*)

lemma *standard-borel-space-UNIVI′*:
  assumes *standard-borel Y*
   and *space* $Y = UNIV$
  shows *standard-borel-space-UNIV Y*
 using *assms* **by**(*simp add*: *standard-borel-space-UNIV-def standard-borel-space-UNIV-axioms-def*)

1.2 ℝ, ℕ, Boolean, $[0, \infty]$

ℝ is a standard Borel space.

interpretation *real* : *standard-borel-space-UNIV real-borel*
  by(*auto intro*!: *standard-borel-space-UNIVI*)

A non-empty Borel subspace of ℝ is also a standard Borel space.

lemma *real-standard-borel-subset*:
  assumes $U \in$ *sets real-borel*
   and $U \neq \{\}$
  shows *standard-borel* (*restrict-space real-borel U*)
proof −
  have *std1*: $id \in$ (*restrict-space real-borel U*) $\to_M$ *real-borel*
   by (*simp add*: *measurable-restrict-space1*)
  obtain *x* **where** *hx* : $x \in U$
   using *assms*(*2*) **by** *auto*
  define *g* :: *real* ⇒ *real*

where $g \equiv (\lambda r.\ \mathit{if}\ r \in U\ \mathit{then}\ r\ \mathit{else}\ x)$
have $g \in$ *real-borel* $\rightarrow_M$ *real-borel*
unfolding *g-def* **by**(*rule borel-measurable-continuous-on-if*) (*simp-all add*: *assms*(*1*))
hence *std2*: $g \in$ *real-borel* $\rightarrow_M$ (*restrict-space real-borel U*)
by(*auto intro*!: *measurable-restrict-space2 simp*: *g-def hx*)
have *std3*: $\forall y \in$ *space* (*restrict-space real-borel U*). $(g \circ id)\ y = y$
by(*simp add*: *g-def space-restrict-space*)
show *?thesis*
using *std1 std2 std3 standard-borel-def* **by** *blast*
qed

A non-empty measurable subset of a standard Borel space is also a standard Borel space.

lemma(**in** *standard-borel*) *standard-borel-subset*:
assumes $U \in$ *sets M*
$U \neq \{\}$
shows *standard-borel* (*restrict-space M U*)
proof −
let *?ginvU* = *g* −' *U*
have *hgu1*:*?ginvU* ∈ *sets real-borel*
using *assms*(*1*) *g-meas measurable-sets-borel* **by** *blast*
have *hgu2*:*f* ' *U* ⊆ *?ginvU*
using *gf-comp-id sets.sets-into-space*[*OF assms*(*1*)] **by** *fastforce*
hence *hgu3*:*?ginvU* ≠ {}
using *assms*(*2*) **by** *blast*
interpret *r-borel-set*: *standard-borel restrict-space real-borel ?ginvU*
by(*rule real-standard-borel-subset*[*OF hgu1 hgu3*])

have *std1*: *r-borel-set.f* ∘ *f* ∈ (*restrict-space M U*) $\rightarrow_M$ *real-borel*
using *sets.sets-into-space*[*OF assms*(*1*)]
by(*auto intro*!: *measurable-comp*[**where** *N*=*restrict-space real-borel ?ginvU*] *measurable-restrict-space3*)
have *std2*: *g* ∘ *r-borel-set.g* ∈ *real-borel* $\rightarrow_M$ (*restrict-space M U*)
by(*auto intro*!: *measurable-comp*[**where** *N*=*restrict-space real-borel ?ginvU*] *measurable-restrict-space3*[*OF g-meas*])
have *std3*: ∀ *x*∈ *space* (*restrict-space M U*). ((*g* ∘ *r-borel-set.g*) ∘ (*r-borel-set.f* ∘ *f*)) *x* = *x*
by (*simp add*: *space-restrict-space*)
show *?thesis*
using *std1 std2 std3 standard-borel-def* **by** *blast*
qed

ℕ is a standard Borel space.

interpretation *nat* : *standard-borel-space-UNIV nat-borel*
proof −
define *n-to-r* :: *nat* ⇒ *real*
where *n-to-r* ≡ (λ*n*. *of-real n*)
define *r-to-n* :: *real* ⇒ *nat*

```
    where r-to-n ≡ (λr. nat ⌊r⌋)

  have n-to-r-measurable: n-to-r ∈ nat-borel →M real-borel
   using borel-measurable-count-space measurable-cong-sets sets-borel-eq-count-space
    by blast
  have r-to-n-measurable: r-to-n ∈ real-borel →M nat-borel
    by(simp add: r-to-n-def)
  have n-to-r-to-n-id: r-to-n ∘ n-to-r = id
    by(simp add: n-to-r-def r-to-n-def comp-def id-def)
  show standard-borel-space-UNIV nat-borel
   using standard-borel-space-UNIVI[OF n-to-r-measurable r-to-n-measurable n-to-r-to-n-id]
    by simp
qed
```

For a countable space X, X is a standard Borel space iff X is a discrete space.

```
lemma countable-standard-iff:
  assumes space X ≠ {}
      and countable (space X)
  shows standard-borel X ⟷ sets X = sets (count-space (space X))
proof
  show standard-borel X ⟹ sets X = sets (count-space (space X))
    using standard-borel.countable-space-discrete assms by simp
next
  assume h[measurable-cong]: sets X = sets (count-space (space X))
  show standard-borel X
 proof(rule standard-borelI[where f=nat.f ∘ to-nat-on (space X) and g=from-nat-into
(space X) ∘ nat.g])
    show nat.f ∘ to-nat-on (space X) ∈ borel-measurable X
      by simp
  next
    have [simp]: from-nat-into (space X) ∈ UNIV → (space X)
      using from-nat-into[OF assms(1)] by simp
    hence [measurable]: from-nat-into (space X) ∈ nat-borel →M X
     using measurable-count-space-eq1[of - - X] measurable-cong-sets[OF sets-borel-eq-count-space]
      by blast
    show from-nat-into (space X) ∘ nat.g ∈ real-borel →M X
      by simp
  next
    fix x
    assume x ∈ space X
    then show (from-nat-into (space X) ∘ nat.g ∘ (nat.f ∘ to-nat-on (space X)))
x = x
      using from-nat-into-to-nat-on[OF assms(2)] by simp
  qed
qed
```

𝔹 is a standard Borel space.

```
lemma to-bool-measurable:
```

```
  assumes f −‘ {True} ∩ space M ∈ sets M
  shows f ∈ M →M bool-borel
proof(rule measurableI)
  fix A
  assume h:A ∈ sets bool-borel
  have h2: f −‘ {False} ∩ space M ∈ sets M
  proof −
    have − {False} = {True}
      by auto
    thus ?thesis
      by(simp add: vimage-sets-compl-iff[where A={False}] assms)
  qed
  have A ⊆ {True,False}
    by auto
  then consider A = {} | A = {True} | A = {False} | A = {True,False}
    by auto
  thus f −‘ A ∩ space M ∈ sets M
  proof cases
    case 1
    then show ?thesis
      by simp
  next
    case 2
    then show ?thesis
      by(simp add: assms)
  next
    case 3
    then show ?thesis
      by(simp add: h2)
  next
    case 4
    then have f −‘ A = f −‘ {True} ∪ f −‘ {False}
      by auto
    thus ?thesis
      using assms h2
      by (metis Int-Un-distrib2 sets.Un)
  qed
qed simp

interpretation bool : standard-borel-space-UNIV bool-borel
  using countable-standard-iff[of bool-borel]
  by(auto intro!: standard-borel-space-UNIVI' simp: sets-borel-eq-count-space)
```

$[0, \infty]$ (the set of extended non-negative real numbers) is a standard Borel space.

```
interpretation ennreal : standard-borel-space-UNIV ennreal-borel
proof −
  define preal-to-real :: ennreal ⇒ real
    where preal-to-real ≡ (λr. if r = ∞ then −1
```

else enn2real r)
  define *real-to-preal* :: *real* ⇒ *ennreal*
    where *real-to-preal* ≡ (λ*r*. *if r* = −*1 then* ∞
        else ennreal r)
  have *preal-to-real-measurable*: *preal-to-real* ∈ *ennreal-borel* $\rightarrow_M$ *real-borel*
    unfolding *preal-to-real-def* **by** *simp*
  have *real-to-preal-measurable*: *real-to-preal* ∈ *real-borel* $\rightarrow_M$ *ennreal-borel*
    unfolding *real-to-preal-def* **by** *simp*
  have *preal-real-preal-id*: *real-to-preal* ∘ *preal-to-real* = *id*
  proof
    fix *r* :: *ennreal*
    show (*real-to-preal* ∘ *preal-to-real*) *r* = *id r*
      using *ennreal-enn2real-if*[*of r*] *ennreal-neg*
      by(*auto simp add*: *real-to-preal-def preal-to-real-def*)
  qed
  show *standard-borel-space-UNIV ennreal-borel*
   using *standard-borel-space-UNIVI*[*OF preal-to-real-measurable real-to-preal-measurable preal-real-preal-id*]
    by *simp*
qed

1.3 $\mathbb{R} \times \mathbb{R}$

definition *real-to-01open* :: *real* ⇒ *real* **where**
real-to-01open r ≡ *arctan r* / *pi* + *1* / *2*

definition *real-to-01open-inverse* :: *real* ⇒ *real* **where**
real-to-01open-inverse r ≡ *tan* (*pi* ∗ *r* − (*pi* / *2*))

lemma *real-to-01open-inverse-correct*:
 real-to-01open-inverse ∘ *real-to-01open* = *id*
 by(*auto simp add*: *real-to-01open-def real-to-01open-inverse-def distrib-left tan-arctan*)

lemma *real-to-01open-inverse-correct′*:
  assumes *0* < *r r* < *1*
  shows *real-to-01open* (*real-to-01open-inverse r*) = *r*
  unfolding *real-to-01open-def real-to-01open-inverse-def*
proof −
  have *arctan* (*tan* (*pi* ∗ *r* − *pi* / *2*)) = *pi* ∗ *r* − *pi* / *2*
    using *arctan-unique*[*of pi* ∗ *r* − *pi* / *2*] *assms*
    by *simp*
  hence *arctan* (*tan* (*pi* ∗ *r* − *pi* / *2*)) / *pi* + *1* / *2* = ((*pi* ∗ *r*) − *pi* / *2*)/ *pi* + *1*/*2*
    by *simp*
  also have ... = *r* − *1*/*2* + *1*/*2*
   by (*metis* (*no-types*, *opaque-lifting*) *divide-inverse mult.left-neutral nonzero-mult-div-cancel-left pi-neq-zero right-diff-distrib*)
  finally show *arctan* (*tan* (*pi* ∗ *r* − *pi* / *2*)) / *pi* + *1* / *2* = *r*
    by *simp*

```
qed

lemma real-to-01open-01 :
 0 < real-to-01open r ∧ real-to-01open r < 1
proof
  have − pi / 2 < arctan r by(simp add: arctan-lbound)
  hence 0 < arctan r + pi / 2 by simp
  hence 0 < (1 / pi) ∗ (arctan r + pi / 2) by simp
  thus 0 < real-to-01open r
    by (simp add: add-divide-distrib real-to-01open-def)
next
  have arctan r < pi / 2 using arctan-ubound by simp
  hence arctan r + pi / 2 < pi by simp
  hence (1 / pi) ∗ (arctan r + pi / 2) < 1 by simp
  thus real-to-01open r < 1
    by(simp add: real-to-01open-def add-divide-distrib)
qed

lemma real-to-01open-continuous:
 continuous-on UNIV real-to-01open
proof −
  have continuous-on UNIV ((λx. x / pi + 1 / 2) ∘ arctan)
  proof (rule continuous-on-compose)
    show continuous-on UNIV arctan
      by (simp add: continuous-on-arctan)
  next
    show continuous-on (range arctan) (λx. x / pi + 1 / 2)
      by(auto intro!: continuous-on-add continuous-on-divide)
  qed
  thus ?thesis
    by(simp add: real-to-01open-def)
qed

lemma real-to-01open-inverse-continuous:
 continuous-on {0<..<1} real-to-01open-inverse
  unfolding real-to-01open-inverse-def
proof(rule Transcendental.continuous-on-tan)
  have [simp]: (λx. pi ∗ x − pi / 2) = (λx. x − pi/2) ∘ (λx. pi ∗ x)
    by auto
  have continuous-on {0<..<1} ...
  proof(rule continuous-on-compose)
    show continuous-on {0<..<1} ((∗) pi)
      by simp
  next
    show continuous-on ((∗) pi ‘ {0<..<1}) (λx. x − pi / 2)
      using continuous-on-diff[of (∗) pi ‘ {0<..<1} λx. x]
      by simp
  qed
  thus continuous-on {0<..<1} (λx. pi ∗ x − pi / 2) by simp
```

next
  have ∀ r∈{0<..<1::real}. −(pi/2) < pi ∗ r − pi / 2 ∧ pi ∗ r − pi / 2 < pi/2
    by *simp*
  thus ∀ r∈{0<..<1::real}. cos (pi ∗ r − pi / 2) ≠ 0
    using *cos-gt-zero-pi* **by** *fastforce*
qed

lemma *real-to-01open-inverse-measurable*:
 real-to-01open-inverse ∈ *restrict-space real-borel* {0<..<1} $\rightarrow_M$ *real-borel*
  using *borel-measurable-continuous-on-restrict real-to-01open-inverse-continuous*
  by *simp*

fun *r01-binary-expansion''* :: *real* ⇒ *nat* ⇒ *nat* × *real* × *real* **where**
r01-binary-expansion'' r 0 = (*if* 1/2 ≤ *r then* (1,1 ,1/2)
        else (0,1/2, 0)) |
r01-binary-expansion'' r (*Suc n*) = (*let* (-,*ur*,*lr*) = *r01-binary-expansion'' r n*;
        k = (*ur* + *lr*)/2 *in*
        (*if k* ≤ *r then* (1,*ur*,*k*)
          else (0,*k*,*lr*)))

a_n where $r = 0.a_0a_1a_2....$ for $0 < r < 1$.

definition *r01-binary-expansion'* :: *real* ⇒ *nat* ⇒ *nat* **where**
r01-binary-expansion' r n ≡ *fst* (*r01-binary-expansion'' r n*)

$a_n = 0$ or 1.

lemma *real01-binary-expansion'-0or1*:
  r01-binary-expansion' r n ∈ {0,1}
  by (*cases n*) (*simp-all add*: *r01-binary-expansion'-def split-beta' Let-def*)

definition *r01-binary-sum* :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *real* **where**
r01-binary-sum a n ≡ (∑ *i*=0..*n*. *real* (*a i*) ∗ ((1/2) ^(*Suc i*)))

definition *r01-binary-sum-lim* :: (*nat* ⇒ *nat*) ⇒ *real* **where**
r01-binary-sum-lim ≡ *lim* ∘ *r01-binary-sum*

definition *r01-binary-expression* :: *real* ⇒ *nat* ⇒ *real* **where**
r01-binary-expression ≡ *r01-binary-sum* ∘ *r01-binary-expansion'*

lemma *r01-binary-expansion-lr-r-ur*:
  assumes 0 < *r r* < 1
  shows (*snd* (*snd* (*r01-binary-expansion'' r n*))) ≤ *r* ∧
      r < (*fst* (*snd* (*r01-binary-expansion'' r n*)))
  using *assms* **by** (*induction n*) (*simp-all add*:*split-beta' Let-def*)

0 ≤ lr ∧ lr < ur ∧ ur ≤ 1.

lemma *r01-binary-expansion-lr-ur-nn*:
  shows 0 ≤ *snd* (*snd* (*r01-binary-expansion'' r n*)) ∧

snd (snd (r01-binary-expansion'' r n)) < fst (snd (r01-binary-expansion'' r n)) ∧
fst (snd (r01-binary-expansion'' r n)) ≤ 1
by (*induction n*) (*simp-all add:split-beta' Let-def*)

lemma *r01-binary-expansion-diff*:
shows (*fst* (*snd* (*r01-binary-expansion'' r n*))) − (*snd* (*snd* (*r01-binary-expansion'' r n*))) = (*1/2*)^(*Suc n*)
proof(*induction n*)
case (*Suc n'*)
then show *?case*
proof(*cases r01-binary-expansion'' r n'*)
case *1*:(*fields a ur lr*)
assume *fst* (*snd* (*r01-binary-expansion'' r n'*)) − *snd* (*snd* (*r01-binary-expansion'' r n'*)) = (*1* / *2*) ^ (*Suc n'*)
then have *2*:*ur* − *lr* = (*1/2*)^(*Suc n'*) **by** (*simp add*: *1*)
show *?thesis*
proof −
have [*simp*]:*ur* ∗ *4* − (*ur* ∗ *4* + *lr* ∗ *4*) / *2* = (*ur* − *lr*) ∗ *2*
by(*simp add*: *division-ring-class.add-divide-distrib*)
have *ur* ∗ *4* − (*ur* ∗ *4* + *lr* ∗ *4*) / *2* = (*1* / *2*) ^ *n'*
by(*simp add*: *2*)
moreover have (*ur* ∗ *4* + *lr* ∗ *4*) / *2* − *lr* ∗ *4* = (*1* / *2*) ^ *n'*
by(*simp add*: *division-ring-class.add-divide-distrib ring-class.right-diff-distrib*[*symmetric*] *2*)
ultimately show *?thesis*
by(*simp add*: *1 Let-def*)
qed
qed
qed *simp*

lrn = *Sn*.

lemma *r01-binary-expression-eq-lr*:
snd (*snd* (*r01-binary-expansion'' r n*)) = *r01-binary-expression r n*
proof(*induction n*)
case *0*
then show *?case*
by(*simp add*: *r01-binary-expression-def r01-binary-sum-def r01-binary-expansion'-def*)
next
case *1*:(*Suc n'*)
show *?case*
proof (*cases r01-binary-expansion'' r n'*)
case *2*:(*fields a ur lr*)
then have *ih*:*lr* = ($\sum$ *i* = *0..n'*. *real* (*fst* (*r01-binary-expansion'' r i*)) ∗ (*1* / *2*) ^ *i* / *2*)
using *1* **by**(*simp add*: *r01-binary-expression-def r01-binary-sum-def r01-binary-expansion'-def*)
have *3*:(*ur* + *lr*) / *2* = *lr* + (*1/2*)^(*Suc* (*Suc n'*))
using *r01-binary-expansion-diff*[*of r n'*] *2* **by** *simp*
show *?thesis*

by(*simp add*: *r01-binary-expression-def r01-binary-sum-def r01-binary-expansion′-def 2 Let-def 3*) *fact*
qed
qed

lemma *r01-binary-expression′-sum-range*:
$\exists$ *k*::*nat*. (*snd* (*snd* (*r01-binary-expansion″ r n*))) = *real k*/*2*^(*Suc n*) $\wedge$
k < *2*^(*Suc n*) $\wedge$
((*r01-binary-expansion′ r n*) = *0* $\longrightarrow$ *even k*) $\wedge$
((*r01-binary-expansion′ r n*) = *1* $\longrightarrow$ *odd k*)
proof −
have [*simp*]:(*snd* (*snd* (*r01-binary-expansion″ r n*))) = ($\sum$ *i*=*0*..*n*. *real* (*r01-binary-expansion′ r i*) ∗ ((*1*/*2*)^(*Suc i*)))
using *r01-binary-expression-eq-lr*[*of r n*] **by**(*simp add*: *r01-binary-expression-def r01-binary-sum-def*)
have $\exists$ *k*::*nat*. ($\sum$ *i*=*0*..*n*. *real* (*r01-binary-expansion′ r i*) ∗ ((*1*/*2*)^(*Suc i*))) = *real k*/*2*^(*Suc n*) $\wedge$
k < *2*^(*Suc n*) $\wedge$
((*r01-binary-expansion′ r n*) = *0* $\longrightarrow$ *even k*) $\wedge$
((*r01-binary-expansion′ r n*) = *1* $\longrightarrow$ *odd k*)
proof(*induction n*)
case *0*
consider *r01-binary-expansion′ r 0* = *0* | *r01-binary-expansion′ r 0* = *1*
using *real01-binary-expansion′-0or1*[*of r 0*] **by** *auto*
then show *?case*
by *cases auto*
next
case (*Suc n′*)
then obtain *k* :: *nat* **where** *ih*:
($\sum$ *i* = *0*..*n′*. *real* (*r01-binary-expansion′ r i*) ∗ (*1* / *2*) ^ *Suc i*) = *real k* / *2*^(*Suc n′*) $\wedge$ *k* < *2*^(*Suc n′*)
by *auto*
have ($\sum$ *i* = *0*..*Suc n′*. *real* (*r01-binary-expansion′ r i*) ∗ (*1* / *2*) ^ *Suc i*) = ($\sum$ *i* = *0*..*n′*. *real* (*r01-binary-expansion′ r i*) ∗ (*1* / *2*) ^ *Suc i*) + *real* (*r01-binary-expansion′ r* (*Suc n′*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n′*)
by *simp*
also have ... = *real k* / *2*^(*Suc n′*) + (*real* (*r01-binary-expansion′ r* (*Suc n′*)))/ *2*^ *Suc* (*Suc n′*)
proof −
have $\bigwedge$*r ra n*. (*r*::*real*) ∗ (*1* / *ra*) ^ *n* = *r* / *ra* ^ *n*
by (*simp add*: *power-one-over*)
then show *?thesis*
using *ih* **by** *presburger*
qed
also have ... = (*2*∗*real k*) / *2*^(*Suc* (*Suc n′*)) + (*real* (*r01-binary-expansion′ r* (*Suc n′*)))/ *2*^ *Suc* (*Suc n′*)
by *simp*
also have ... = (*2*∗(*real k*) + *real* (*r01-binary-expansion′ r* (*Suc n′*)))/*2* ^ *Suc* (*Suc n′*)

by (*simp add*: *add-divide-distrib*)
also have ... = (*real* (*2*∗*k* + *r01-binary-expansion′ r* (*Suc n′*)))/*2* ^ *Suc* (*Suc n′*)
by *simp*
finally have ($\sum$ *i* = *0*..*Suc n′*. *real* (*r01-binary-expansion′ r i*) ∗ (*1* / *2*) ^ *Suc i*) = *real* (*2* ∗ *k* + *r01-binary-expansion′ r* (*Suc n′*)) / *2* ^ *Suc* (*Suc n′*) **.**
moreover have *2* ∗ *k* + *r01-binary-expansion′ r* (*Suc n′*) < *2*^*Suc* (*Suc n′*)
proof −
have *k* + *1* ≤ *2*^*Suc n′*
using *ih* **by** *simp*
hence *2*∗*k* + *2* ≤ *2*^*Suc* (*Suc n′*)
by *simp*
thus *?thesis*
using *real01-binary-expansion′-0or1*[*of r Suc n′*]
by *auto*
qed
moreover have *r01-binary-expansion′ r* (*Suc n′*) = *0* ⟶ *even* (*2* ∗ *k* + *r01-binary-expansion′ r* (*Suc n′*))
by *simp*
moreover have *r01-binary-expansion′ r* (*Suc n′*) = *1* ⟶ *odd* (*2* ∗ *k* + *r01-binary-expansion′ r* (*Suc n′*))
by *simp*
ultimately show *?case* **by** *fastforce*
qed
thus *?thesis*
by *simp*
qed

$an = bn \leftrightarrow Sn = S'n.$

lemma *r01-binary-expansion′-expression-eq*:
r01-binary-expansion′ r1 = *r01-binary-expansion′ r2* ⟷
r01-binary-expression r1 = *r01-binary-expression r2*
proof
assume *r01-binary-expansion′ r1* = *r01-binary-expansion′ r2*
then show *r01-binary-expression r1* = *r01-binary-expression r2*
by(*simp add*: *r01-binary-expression-def*)
next
assume *r01-binary-expression r1* = *r01-binary-expression r2*
then have *1*:⋀*n*. *r01-binary-sum* (*r01-binary-expansion′ r1*) *n* = *r01-binary-sum* (*r01-binary-expansion′ r2*) *n*
by(*simp add*: *r01-binary-expression-def*)
show *r01-binary-expansion′ r1* = *r01-binary-expansion′ r2*
proof
fix *n*
show *r01-binary-expansion′ r1 n* = *r01-binary-expansion′ r2 n*
proof(*cases n*)
case *0*
then show *?thesis*
using *1*[*of 0*] **by**(*simp add*: *r01-binary-sum-def*)

```
    next
      fix n'
      case (Suc n')
    have r01-binary-sum (r01-binary-expansion' r1) n − r01-binary-sum (r01-binary-expansion'
r1) n' = r01-binary-sum (r01-binary-expansion' r2) n − r01-binary-sum (r01-binary-expansion'
r2) n'
        by(simp add: 1)
      thus ?thesis
        using ‹n = Suc n'› by(simp add: r01-binary-sum-def)
    qed
  qed
qed

lemma power2-e:
 ⋀e::real. 0 < e ⟹ ∃ n::nat. real-of-rat (1/2)^n < e
  by (simp add: real-arch-pow-inv)

lemma r01-binary-expression-converges-to-r:
  assumes 0 < r
     and  r < 1
   shows LIMSEQ (r01-binary-expression r)  r
proof
  fix e :: real
  assume 0 < e
  then obtain k :: nat where hk:real-of-rat (1/2)^k < e
    using power2-e by auto
  show ∀F x in sequentially. dist (r01-binary-expression r x) r < e
  proof(rule eventually-sequentiallyI[of k])
    fix m
    assume k ≤ m
    have | r − r01-binary-expression r m | < e
    proof (cases r01-binary-expansion'' r m)
      case 1:(fields a ur lr)
      then have |r − r01-binary-expression r m| = |r − lr|
        by (metis r01-binary-expression-eq-lr snd-conv)
      also have ... = r − lr
        using r01-binary-expansion-lr-r-ur[OF assms] 1
        by (metis abs-of-nonneg diff-ge-0-iff-ge snd-conv)
      also have ... < e
      proof −
        have r − lr ≤ ur − lr
          using r01-binary-expansion-lr-r-ur[of r] assms 1
          by (metis diff-right-mono fst-conv less-imp-le snd-conv)
        also have ... = (1/2)^(Suc m)
          using r01-binary-expansion-diff[of r m]
          by(simp add: 1)
        also have ... ≤ (1/2)^(Suc k)
          using ‹k ≤ m› by simp
        also have ... < (1/2)^k by simp
```

        finally show *?thesis*
          using *hk* **by** (*simp add*: *of-rat-divide*)
      qed
      finally show *?thesis* **.**
    qed
    then show *dist* (*r01-binary-expression r m*) *r* < *e*
      by (*simp add*: *dist-real-def*)
  qed
qed

lemma *r01-binary-expression-correct*:
  assumes *0* < *r*
    and *r* < *1*
  shows $r = (\sum n.\ real\ (r01\text{-}binary\text{-}expansion'\ r\ n) * (1/2)\hat{}(Suc\ n))$
proof −
  have $(\lambda n.\ (\lambda n.\ \sum i{<}n.\ real\ (r01\text{-}binary\text{-}expansion'\ r\ i) * (1\ /\ 2)\ \hat{}\ Suc\ i)\ (Suc\ n)) = r01\text{-}binary\text{-}expression\ r$
  proof −
    have $\bigwedge n.\ \{..{<}Suc\ n\} = \{0..n\}$ **by** *auto*
    thus *?thesis*
      by(*auto simp add*: *r01-binary-expression-def r01-binary-sum-def*)
  qed
  hence $LIMSEQ\ (\lambda n.\ \sum i{<}n.\ real\ (r01\text{-}binary\text{-}expansion'\ r\ i) * (1\ /\ 2)\ \hat{}\ Suc\ i)\ r$
    using *r01-binary-expression-converges-to-r*[*OF assms*] *LIMSEQ-imp-Suc*[*of* $\lambda n.\ \sum i{<}n.\ real\ (r01\text{-}binary\text{-}expansion'\ r\ i) * (1\ /\ 2)\ \hat{}\ Suc\ i\ r$]
    by *simp*
  thus *?thesis*
    using *suminf-eq-lim*[*of* $\lambda n.\ real\ (r01\text{-}binary\text{-}expansion'\ r\ n) * (1/2)\hat{}(Suc\ n)$] *assms limI*[*of* $(\lambda n.\ \sum i{<}n.\ real\ (r01\text{-}binary\text{-}expansion'\ r\ i) * (1\ /\ 2)\ \hat{}\ Suc\ i)\ r$]
    by *simp*
qed

S0 ≤ *S1* ≤ *S2* ≤ ....

lemma *binary-sum-incseq*:
  incseq (*r01-binary-sum a*)
  by(*simp add*: *incseq-Suc-iff r01-binary-sum-def*)

lemma *r01-eq-iff*:
  assumes *0* < *r1 r1* < *1*
        0 < *r2 r2* < *1*
    shows $r1 = r2 \longleftrightarrow r01\text{-}binary\text{-}expansion'\ r1 = r01\text{-}binary\text{-}expansion'\ r2$
proof *auto*
  assume $r01\text{-}binary\text{-}expansion'\ r1 = r01\text{-}binary\text{-}expansion'\ r2$
  then have *1*:*r01-binary-expression r1* = *r01-binary-expression r2*
    using *r01-binary-expansion′-expression-eq*[*of r1 r2*] **by** *simp*
  have *r1* = *lim* (*r01-binary-expression r1*)
    using *limI*[*of - r1*] *r01-binary-expression-converges-to-r*[*of r1*] *assms*(*1*,*2*)
    by *simp*

```
  also have ... = lim (r01-binary-expression r2)
    by (simp add: 1)
  also have ... = r2
    using limI[of - r2] r01-binary-expression-converges-to-r[of r2] assms(3,4)
    by simp
  finally show r1 = r2 .
qed

lemma power-half-summable:
 summable (λn. ((1::real) / 2) ^ Suc n)
  using power-half-series summable-def by blast


lemma binary-expression-summable:
   assumes ⋀n. a n ∈ {0,1 :: nat}
   shows summable (λn. real (a n) ∗ (1/2)^(Suc n))
proof −
  have summable (λn::nat. |real (a n) ∗ ((1::real) / (2::real)) ^ Suc n|)
  proof(rule summable-rabs-comparison-test[of λn. real (a n) ∗ (1/2)^(Suc n) λn.
(1/2)^(Suc n)])
    have ⋀n. |real (a n) ∗ (1 / 2) ^ Suc n| ≤ (1 / 2)^(Suc n)
    proof −
      fix n
      have |real (a n) ∗ (1 / 2) ^ Suc n| = real (a n) ∗ (1 / 2) ^ Suc n
        using assms by simp
      also have ... ≤ (1 / 2) ^ Suc n
      proof −
        consider a n = 0 | a n = 1
          using assms by (meson insertE singleton-iff)
        then show ?thesis
          by(cases,auto)
      qed
      finally show |real (a n) ∗ (1 / 2) ^ Suc n| ≤ (1 / 2)^(Suc n) .
    qed
    thus ∃N. ∀n≥N. |real (a n) ∗ (1 / 2) ^ Suc n| ≤ (1 / 2) ^ Suc n
      by simp
  next
    show summable (λn. ((1::real) / 2) ^ Suc n)
      using power-half-summable by simp
  qed
  thus ?thesis by simp
qed

lemma binary-expression-gteq0:
  assumes ⋀n. a n ∈ {0,1 :: nat}
  shows 0 ≤ (∑n. real (a (n + k)) ∗ (1 / 2) ^ Suc (n + k))
proof −
  have (∑n. 0) ≤ (∑n. real (a (n + k)) ∗ (1 / 2) ^ Suc (n + k))
    using binary-expression-summable[of a] summable-iff-shift[of λn. real (a n) ∗
```

(*1* / *2*) ^ *Suc n k*] *suminf-le*[*of* λ*n*. *0* λ*n*. *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*)] *assms*
by *simp*
thus *?thesis* **by** *simp*
qed

lemma *binary-expression-leeq1*:
assumes ⋀*n*. *a n* ∈ {*0*,*1* :: *nat*}
shows (∑*n*. *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*)) ≤ *1*
proof −
have (∑*n*. *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*)) ≤ (∑*n*. (*1*/*2*)^(*Suc n*))
proof(*rule suminf-le*)
fix *n*
have *1*:*real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*) ≤ (*1* / *2*) ^ *Suc* (*n* + *k*)
using *assms*[*of n*+*k*] **by** *auto*
have *2*:((*1*::*real*) / *2*) ^ *Suc* (*n* + *k*) ≤ (*1* / *2*) ^ *Suc n*
by *simp*
show *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*) ≤ (*1* / *2*) ^ *Suc n*
by(*rule order.trans*[*OF 1 2*])
next
show *summable* (λ*n*. *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*))
using *binary-expression-summable*[*of a*] *summable-iff-shift*[*of* λ*n*. *real* (*a n*) ∗ (*1* / *2*) ^ *Suc n k*] *assms*
by *simp*
next
show *summable* (λ*n*. ((*1*::*real*) / *2*) ^ *Suc n*)
using *power-half-summable* **by** *simp*
qed
thus *?thesis*
using *power-half-series sums-unique* **by** *fastforce*
qed

lemma *binary-expression-less-than*:
assumes ⋀*n*. *a n* ∈ {*0*,*1* :: *nat*}
shows (∑*n*. *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*)) ≤ (∑*n*. (*1* / *2*) ^ *Suc* (*n* + *k*))
proof(*rule suminf-le*)
fix *n*
show *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*) ≤ (*1* / *2*) ^ *Suc* (*n* + *k*)
using *assms*[*of n* + *k*] **by** *auto*
next
show *summable* (λ*n*. *real* (*a* (*n* + *k*)) ∗ (*1* / *2*) ^ *Suc* (*n* + *k*))
using *summable-iff-shift*[*of* λ*n*. *real* (*a n*) ∗ (*1* / *2*) ^ *Suc n k*] *binary-expression-summable*[*of a*] *assms*
by *simp*
next
show *summable* (λ*n*. ((*1*::*real*) / *2*) ^ *Suc* (*n* + *k*))
using *power-half-summable summable-iff-shift*[*of* λ*n*. ((*1*::*real*) / *2*) ^ *Suc n k*]
by *simp*

qed

lemma *lim-sum-ai*:
assumes $\bigwedge n.\ a\ n \in \{0{,}1 :: nat\}$
shows $lim\ (\lambda n.\ (\sum i{=}0..n.\ real\ (a\ i) * (1/2)\hat{}(Suc\ i))) = (\sum n{::}nat.\ real\ (a\ n) * (1/2)\hat{}(Suc\ n))$
proof −
have $\bigwedge n{::}nat.\ \{0..n\} = \{..n\}$ **by** *auto*
hence $LIMSEQ\ (\lambda n.\ \sum i{=}0..n.\ real\ (a\ i) * (1\ /\ 2)\ \hat{}\ Suc\ i)\ (\sum n.\ real\ (a\ n) * (1\ /\ 2)\ \hat{}\ Suc\ n)$
using *summable-LIMSEQ'*[*of* $\lambda n.\ real\ (a\ n) * (1/2)\hat{}(Suc\ n)$] *binary-expression-summable*[*of* *a*] *assms*
by *simp*
thus $lim\ (\lambda n.\ (\sum i{=}0..n.\ real\ (a\ i) * (1/2)\hat{}(Suc\ i))) = (\sum n.\ real\ (a\ n) * (1\ /\ 2)\ \hat{}\ Suc\ n)$
using *limI* **by** *simp*
qed

lemma *half-1-minus-sum*:
$1 - (\sum i{<}k.\ ((1{::}real)\ /\ 2)\ \hat{}\ Suc\ i) = (1/2)\hat{}k$
by(*induction k*) *auto*

lemma *half-sum*:
$(\sum n.\ ((1{::}real)\ /\ 2)\ \hat{}\ (Suc\ (n + k))) = (1/2)\hat{}k$
using *suminf-split-initial-segment*[*of* $\lambda n.\ ((1{::}real)\ /\ 2)\ \hat{}(Suc\ n)\ k$] *half-1-minus-sum*[*of* *k*] *power-half-series* *sums-unique*[*of* $\lambda n.\ (1\ /\ 2)\ \hat{}\ Suc\ n\ 1$] *power-half-summable*
by *fastforce*

lemma *ai-exists0-less-than-sum*:
assumes $\bigwedge n.\ a\ n \in \{0{,}1\}$
$i \geq m$
and $a\ i = 0$
shows $(\sum n{::}nat.\ real\ (a\ (n + m)) * (1/2)\hat{}(Suc\ (n + m))) < (1\ /\ 2)\ \hat{}\ m$
proof −
have $(\sum n{::}nat.\ real\ (a\ (n + m)) * (1/2)\hat{}(Suc\ (n + m))) = (\sum n{<}i{-}m.\ real\ (a\ (n + m)) * (1/2)\hat{}(Suc\ (n + m))) + (\sum n{::}nat.\ real\ (a\ (n + i)) * (1/2)\hat{}(Suc\ (n + i)))$
using *suminf-split-initial-segment*[*of* $\lambda n.\ real\ (a\ (n + m)) * (1/2)\hat{}(Suc\ (n + m))\ i{-}m$] *assms*(*1*) *binary-expression-summable*[*of* *a*] *summable-iff-shift*[*of* $\lambda n.\ real\ (a\ n) * (1\ /\ 2)\ \hat{}\ Suc\ n\ m$] *assms*(*2*)
by *simp*
also have ... $< (1\ /\ 2)\ \hat{}\ m$
proof −
have $(\sum n.\ real\ (a\ (n + i)) * (1\ /\ 2)\ \hat{}\ Suc\ (n + i)) \leq (1\ /\ 2)\ \hat{}\ Suc\ i$
proof −
have $(\sum n{::}nat.\ real\ (a\ (n + i)) * (1/2)\hat{}(Suc\ (n + i))) = (\sum n{::}nat.\ real\ (a\ (Suc\ n + i)) * (1/2)\hat{}(Suc\ (Suc\ n + i)))$
using *suminf-split-head*[*of* $\lambda n.\ real\ (a\ (n + i)) * (1/2)\hat{}(Suc\ (n + i))$] *assms*(*1*,*3*) *binary-expression-summable*[*of* *a*] *summable-iff-shift*[*of* $\lambda n.\ real\ (a\ n)$

∗ (1 / 2) ^ Suc n i]
  by *simp*
 also have ... = ($\sum$ n::nat. real (a (n + Suc i)) ∗ (1/2)^(Suc n + Suc i))
  by *simp*
 also have ... ≤ ($\sum$ n::nat. (1/2)^(Suc n + Suc i))
  using *binary-expression-less-than*[*of a Suc i*] *assms*(1)
  by *simp*
 also have ... = (1/2)^(Suc i)
  using *half-sum*[*of Suc i*] **by** *simp*
 finally show *?thesis* **.**
qed
moreover have ($\sum$ n<i − m. real (a (n + m)) ∗ (1 / 2) ^ Suc (n + m)) ≤ (1/2)^m − (1/2)^i
proof −
 have ($\sum$ n<i − m. real (a (n + m)) ∗ (1 / 2) ^ Suc (n + m)) ≤ ($\sum$ n<i − m. (1 / 2) ^ Suc (n + m))
 proof −
  have real (a i) ∗ (1 / 2) ^ Suc i ≤ (1 / 2) ^ Suc i **for** i
   using *assms*(1)[*of i*] **by** *auto*
  thus *?thesis*
   by (*simp add*: *sum-mono*)
 qed
 also have ... = ($\sum$ n. (1 / 2) ^ Suc (n + m)) − ($\sum$ n. (1 / 2) ^ Suc (n + (i − m) + m))
  using *suminf-split-initial-segment*[*of* λn. (1 / 2) ^ Suc (n + m) i−m] *power-half-summable summable-iff-shift*[*of* λn. ((1::real) / 2) ^ Suc n m]
  by *fastforce*
 also have ... = ($\sum$ n. (1 / 2) ^ Suc (n + m)) − ($\sum$ n. (1 / 2) ^ Suc (n + i))
  using *assms*(2) **by** *simp*
 also have ... = (1/2)^m − (1/2)^i
  using *half-sum* **by** *fastforce*
 finally show *?thesis* **.**
qed
ultimately have ($\sum$ n<i − m. real (a (n + m)) ∗ (1 / 2) ^ Suc (n + m)) + ($\sum$ n. real (a (n + i)) ∗ (1 / 2) ^ Suc (n + i)) ≤ (1 / 2) ^ Suc i + (1 / 2) ^ m − (1 / 2) ^ i
 by *linarith*
also have ... < (1 / 2) ^ m
 by *simp*
finally show *?thesis* **.**
qed
finally show *?thesis* **.**
qed

lemma *ai-exists0-less-than1*:
assumes ⋀n. a n ∈ {0,1}
 and ∃i. a i = 0
shows ($\sum$ n::nat. real (a n) ∗ (1/2)^(Suc n)) < 1

using *ai-exists0-less-than-sum*[*of a 0*] *assms*
by *auto*

lemma *ai-1-gt*:
assumes $\bigwedge n.\ a\ n \in \{0,1\}$
and $a\ i = 1$
shows $(1/2)\hat{}(Suc\ i) \le (\sum n{::}nat.\ real\ (a\ (n+i)) * (1/2)\hat{}(Suc\ (n+i)))$
proof −
have $1{:}(\sum n{::}nat.\ real\ (a\ (n+i)) * (1/2)\hat{}(Suc\ (n+i))) = (1\ /\ 2)\ \hat{}\ Suc\ (0 + i) + (\sum n.\ real\ (a\ (Suc\ n + i)) * (1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n + i))$
using *suminf-split-head*[*of* $\lambda n.\ real\ (a\ (n+i)) * (1/2)\hat{}(Suc\ (n+i))$] *binary-expression-summable*[*of a*] *summable-iff-shift*[*of* $\lambda n.\ real\ (a\ n) * (1\ /\ 2)\ \hat{}\ Suc\ n\ i$] *assms*
by *simp*
show *?thesis*
using *1 binary-expression-gteq0*[*of a Suc i*] *assms*(*1*)
by *simp*
qed

lemma *ai-exists1-gt0*:
assumes $\bigwedge n.\ a\ n \in \{0,1\}$
and $\exists i.\ a\ i = 1$
shows $0 < (\sum n{::}nat.\ real\ (a\ n) * (1/2)\hat{}(Suc\ n))$
proof −
obtain *k* **where** *h1*: $a\ k = 1$
using *assms*(*2*) **by** *auto*
have $(1/2)\hat{}(Suc\ k) = (\sum n{::}nat.\ (if\ n = k\ then\ (1/2)\hat{}(Suc\ k)\ else\ (0{::}real)))$
proof −
have $(\lambda n.\ if\ n \in \{k\}\ then\ (1\ /\ 2)\ \hat{}\ Suc\ k\ else\ (0{::}real)) = (\lambda n.\ if\ n = k\ then\ (1/2)\hat{}(Suc\ k)\ else\ 0)$
by *simp*
moreover have $(\lambda n.\ if\ n \in \{k\}\ then\ (1\ /\ 2)\ \hat{}\ Suc\ k\ else\ (0{::}real))$ *sums* $(\sum r{\in}\{k\}.\ (1\ /\ 2)\ \hat{}\ Suc\ k)$
using *sums-If-finite-set*[*of* $\{k\}\ \lambda n.\ ((1{::}real)/2)\hat{}(Suc\ k)$] **by** *simp*
ultimately have $(\lambda n.\ if\ n = k\ then\ (1\ /\ 2)\ \hat{}\ Suc\ k\ else\ (0{::}real))$ *sums* $(1/2)\hat{}(Suc\ k)$
by *simp*
thus *?thesis*
using *sums-unique*[*of* $\lambda n.\ if\ n = k\ then\ (1\ /\ 2)\ \hat{}\ Suc\ k\ else\ (0{::}real)$ $(1/2)\hat{}(Suc\ k)$]
by *simp*
qed
also have $(\sum n{::}nat.\ (if\ n = k\ then\ (1/2)\hat{}(Suc\ k)\ else\ 0)) \le (\sum n{::}nat.\ real\ (a\ n) * (1/2)\hat{}(Suc\ n))$
proof(*rule suminf-le*)
show $\bigwedge n.\ (if\ n = k\ then\ (1\ /\ 2)\ \hat{}\ Suc\ k\ else\ 0) \le real\ (a\ n) * (1\ /\ 2)\ \hat{}\ Suc\ n$
proof −
fix *n*
show $(if\ n = k\ then\ (1\ /\ 2)\ \hat{}\ Suc\ k\ else\ 0) \le real\ (a\ n) * (1\ /\ 2)\ \hat{}\ Suc\ n$

    by(*cases n = k*; *simp add*: *h1*)
  qed
next
  show *summable* ($\lambda n.$ *if n = k then* (*1* / *2*) ^ *Suc k else* (*0*::*real*))
    using *summable-single*[*of k* $\lambda n.$ ((*1*::*real*) / *2*) ^ *Suc k*]
    by *simp*
next
  show *summable* ($\lambda n.$ *real* (*a n*) $*$ (*1* / *2*) ^ *Suc n*)
    using *binary-expression-summable*[*of a*] *assms*(*1*)
    by *simp*
qed
finally have (*1* / *2*) ^ *Suc k* $\leq$ ($\sum$ *n. real* (*a n*) $*$ (*1* / *2*) ^ *Suc n*) **.**
moreover have *0* < ((*1*::*real*) / *2*) ^ *Suc k* **by** *simp*
ultimately show *?thesis* **by** *linarith*
qed

lemma *r01-binary-expression-ex0*:
  assumes *0* < *r r* < *1*
  shows $\exists$ *i. r01-binary-expansion′ r i* = *0*
proof (*rule ccontr*)
  assume ¬ ($\exists$ *i. r01-binary-expansion′ r i* = *0*)
  then have $\bigwedge$*i. r01-binary-expansion′ r i* = *1*
    using *real01-binary-expansion′-0or1*[*of r*] **by** *blast*
  hence *1*:*r01-binary-expression r* = ($\lambda n.$ $\sum$ *i*=*0*..*n.* ((*1*/*2*)^(*Suc i*)))
    by(*auto simp*: *r01-binary-expression-def r01-binary-sum-def*)
  have *LIMSEQ* (*r01-binary-expression r*) *1*
  proof −
    have *LIMSEQ* ($\lambda n.$ $\sum$ *i*=*0*..*n.* (((*1*::*real*)/*2*)^(*Suc i*))) *1*
      using *power-half-series sums-def′*[*of* $\lambda n.$ ((*1*::*real*)/*2*)^(*Suc n*) *1*]
      by *simp*
    thus *?thesis*
      using *1* **by** *simp*
  qed
  moreover have *LIMSEQ* (*r01-binary-expression r*) *r*
    using *r01-binary-expression-converges-to-r*[*of r*] *assms*
    by *simp*
  ultimately have *r* = *1*
    using *LIMSEQ-unique* **by** *auto*
  thus *False*
    using *assms* **by** *simp*
qed

lemma *r01-binary-expression-ex1*:
  assumes *0* < *r r* < *1*
  shows $\exists$ *i. r01-binary-expansion′ r i* = *1*
proof (*rule ccontr*)
  assume ¬ ($\exists$ *i. r01-binary-expansion′ r i* = *1*)
  then have $\bigwedge$*i. r01-binary-expansion′ r i* = *0*

```
    using real01-binary-expansion'-0or1[of r] by blast
  hence 1:r01-binary-expression r = (λn. ∑ i=0..n. 0)
    by(auto simp add: r01-binary-expression-def r01-binary-sum-def)
  hence LIMSEQ (r01-binary-expression r) 0
    by simp
  moreover have LIMSEQ (r01-binary-expression r) r
    using r01-binary-expression-converges-to-r[of r] assms
    by simp
  ultimately have r = 0
    using LIMSEQ-unique by auto
  thus False
    using assms by simp
qed

lemma r01-binary-expansion'-gt1:
  1 ≤ r ⟷ (∀ n. r01-binary-expansion' r n = 1)
proof auto
  fix n
  assume h:1 ≤ r
  show r01-binary-expansion' r n = Suc 0
    unfolding r01-binary-expansion'-def
  proof(cases n)
    case 0
    then show fst (r01-binary-expansion'' r n) = Suc 0
      using h by simp
  next
    case 2:(Suc n')
    show fst (r01-binary-expansion'' r n) = Suc 0
    proof(cases r01-binary-expansion'' r n')
      case 3:(fields a ur lr)
      then have (ur + lr) / 2 ≤ 1
        using r01-binary-expansion-lr-ur-nn[of r Suc n']
        by (cases ((ur + lr) / 2) ≤ r) (auto simp: Let-def)
      thus fst (r01-binary-expansion'' r n) = Suc 0
        using h by(simp add: 2 3 Let-def)
    qed
  qed
next
  assume h:∀ n. r01-binary-expansion' r n = Suc 0
  show 1 ≤ r
  proof(rule ccontr)
    assume ¬ 1 ≤ r
    then consider r ≤ 0 | 0 < r ∧ r < 1
      by linarith
    then show False
    proof cases
      case 1
      then have r01-binary-expansion' r 0 = 0
        by(simp add: r01-binary-expansion'-def)
```

```
      then show ?thesis
        using h by simp
    next
      case 2
      then have ∃ i. r01-binary-expansion' r i = 0
        using r01-binary-expression-ex0[of r] by simp
      then show ?thesis
        using h by simp
    qed
  qed
qed

lemma r01-binary-expansion'-lt0:
 r ≤ 0 ⟷ (∀ n. r01-binary-expansion' r n = 0)
proof auto
  fix n
  assume h:r ≤ 0
  show r01-binary-expansion' r n = 0
  proof(cases n)
    case 0
    then show ?thesis
      using h by(simp add: r01-binary-expansion'-def)
  next
    case hn:(Suc n')
    then show ?thesis
      unfolding r01-binary-expansion'-def
    proof(cases r01-binary-expansion'' r n')
      case 1:(fields a ur lr)
      then have 0 < ((ur + lr) / 2)
        using r01-binary-expansion-lr-ur-nn[of r n']
        by simp
      hence r < ...
        using h by linarith
      then show fst (r01-binary-expansion'' r n) = 0
        by(simp add: 1 hn Let-def)
    qed
  qed
next
  assume h:∀ n. r01-binary-expansion' r n = 0
  show r ≤ 0
  proof(rule ccontr)
    assume ¬ r ≤ 0
    then consider 0 < r ∧ r < 1 | 1 ≤ r by linarith
    thus False
    proof cases
      case 1
      then have ∃ i. r01-binary-expansion' r i = 1
        using r01-binary-expression-ex1[of r] by simp
      then show ?thesis
```

using *h* **by** *simp*
next
case *2*
then show *?thesis*
using *r01-binary-expansion′-gt1*[*of r*] *h* **by** *simp*
qed
qed
qed

The sequence 111111 . . . does not appear in $r = 0.a_1a_2 \ldots$.

lemma *r01-binary-expression-ex0-strong*:
assumes *0 < r r < 1*
shows ∃ *i*≥*n. r01-binary-expansion′ r i = 0*
proof(*cases r01-binary-expansion′′ r n*)
case *1*:(*fields a ur lr*)
show *?thesis*
proof(*rule ccontr*)
assume ¬ (∃ *i*≥*n. r01-binary-expansion′ r i = 0*)
then have *h*:∀ *i*≥*n. r01-binary-expansion′ r i = 1*
using *real01-binary-expansion′-0or1*[*of r*] **by** *blast*

have *r =* (∑ *i=0..n. real* (*r01-binary-expansion′ r i*) ∗ ((*1/2*)^(*Suc i*))) + (∑ *i::nat. real* (*r01-binary-expansion′ r* (*i +* (*Suc n*))) ∗ ((*1/2*)^(*Suc* (*i +* (*Suc n*)))))
proof −
have *r =* (∑ *l. real* (*r01-binary-expansion′ r l*) ∗ (*1 / 2*) ^ *Suc l*)
using *r01-binary-expression-correct*[*of r*] *assms* **by** *simp*
also have ... = (∑ *l. real* (*r01-binary-expansion′ r* (*l + Suc n*)) ∗ (*1 / 2*) ^ *Suc* (*l + Suc n*)) + (∑ *i<Suc n. real* (*r01-binary-expansion′ r i*) ∗ (*1 / 2*) ^ *Suc i*)
apply(*rule suminf-split-initial-segment*)
apply(*rule binary-expression-summable*)
using *real01-binary-expansion′-0or1*[*of r*] **by** *simp*
also have ... = (∑ *i=0..n. real* (*r01-binary-expansion′ r i*) ∗ ((*1/2*)^(*Suc i*))) + (∑ *i::nat. real* (*r01-binary-expansion′ r* (*i +* (*Suc n*))) ∗ ((*1/2*)^(*Suc* (*i +* (*Suc n*)))))
proof −
have ⋀*n.* {..<*Suc n*} = {*0..n*} **by** *auto*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **.**
qed
also have ... = (∑ *i=0..n. real* (*r01-binary-expansion′ r i*) ∗ ((*1/2*)^(*Suc i*))) + (∑ *i::nat.* ((*1/2*)^(*Suc* (*i +* (*Suc n*)))))
using *h* **by** *simp*
also have ... = (∑ *i=0..n. real* (*r01-binary-expansion′ r i*) ∗ ((*1/2*)^(*Suc i*))) + (*1/2*)^(*Suc n*)
using *half-sum*[*of Suc n*] **by** *simp*
also have ... = *lr +* (*1/2*)^(*Suc n*)

```
    using 1 r01-binary-expression-eq-lr[of r n]
    by(simp add: r01-binary-expression-def r01-binary-sum-def)
  also have ... = ur
    using r01-binary-expansion-diff[of r n]
    by(simp add: 1)
  finally have r = ur .
  moreover have r < ur
    using r01-binary-expansion-lr-r-ur[of r n] assms 1
    by simp
  ultimately show False
    by simp
 qed
qed
```

A binary expression is well-formed when $111\ldots$ does not appear in the tail of the sequence

```
definition biexp01-well-formed :: (nat ⇒ nat) ⇒ bool where
biexp01-well-formed a ≡ (∀n. a n ∈ {0,1}) ∧ (∀n. ∃m≥n. a m = 0)

lemma biexp01-well-formedE:
 assumes biexp01-well-formed a
 shows (∀n. a n ∈ {0,1}) ∧ (∀n. ∃m≥n. a m = 0)
 using assms by(simp add: biexp01-well-formed-def)

lemma biexp01-well-formedI:
 assumes ⋀n. a n ∈ {0,1}
    and ⋀n. ∃m≥n. a m = 0
   shows biexp01-well-formed a
 using assms by(simp add: biexp01-well-formed-def)

lemma r01-binary-expansion-well-formed:
 assumes 0 < r r < 1
 shows biexp01-well-formed (r01-binary-expansion' r)
 using r01-binary-expression-ex0-strong[of r] assms real01-binary-expansion'-0or1[of
r]
 by(simp add: biexp01-well-formed-def)

lemma biexp01-well-formed-comb:
 assumes biexp01-well-formed a
    and biexp01-well-formed b
   shows biexp01-well-formed (λn. if even n then a (n div 2)
                                  else b ((n−1) div 2))
proof(rule biexp01-well-formedI)
 show ⋀n. (if even n then a (n div 2) else b ((n − 1) div 2)) ∈ {0, 1}
   using assms biexp01-well-formedE by simp
next
 fix n
 obtain m where 1:m≥n ∧ a m = 0
   using assms biexp01-well-formedE by blast
```

```
  then have a ((2*m) div 2) = 0 by simp
  hence (if even (2*m) then a (2*m div 2) else b ((2*m − 1) div 2)) = 0
    by simp
  moreover have 2*m ≥ n using 1 by simp
  ultimately show ∃m≥n. (if even m then a (m div 2) else b ((m − 1) div 2))
= 0
    by auto
qed
```

```
lemma nat-complete-induction:
  assumes P (0 :: nat)
      and ⋀n. (⋀m. m ≤ n ⟹ P m) ⟹ P (Suc n)
    shows P n
proof(cases n)
  case 0
  then show ?thesis
    using assms(1) by simp
next
  case h:(Suc n')
  have P (Suc n')
  proof(rule assms(2))
    show ⋀m. m ≤ n' ⟹ P m
    proof(induction n')
      case 0
      then show ?case
        using assms(1) by simp
    next
      case (Suc n'')
      then show ?case
        by (metis assms(2) le-SucE)
    qed
  qed
  thus ?thesis
    using h by simp
qed
```

(∑m. real (a m) * (1 / 2) ^ Suc m) n = a n.

```
lemma biexp01-well-formed-an:
  assumes biexp01-well-formed a
  shows r01-binary-expansion' (∑m. real (a m) * (1 / 2) ^ Suc m) n = a n
proof(rule nat-complete-induction[of - n])
  show r01-binary-expansion' (∑m. real (a m) * (1 / 2) ^ Suc m) 0 = a 0
  proof (auto simp add: r01-binary-expansion'-def)
    assume h:1 ≤ (∑m. real (a m) * (1 / 2) ^ m / 2) * 2
    show Suc 0 = a 0
    proof(rule ccontr)
      assume Suc 0 ≠ a 0
```

then have *a 0 = 0*
using *assms(1) biexp01-well-formedE[of a]* **by** *auto*
hence $(\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ (Suc\ m)) = (\sum m.\ real\ (a\ (Suc\ m)) * (1\ /\ 2)\ \hat{}\ (Suc\ (Suc\ m)))$
using *suminf-split-head[of λm. real (a m) ∗ (1 / 2) ^ (Suc m)] binary-expression-summable[of a] assms biexp01-well-formedE*
by *simp*
also have ... < *1/2*
using *ai-exists0-less-than-sum[of a 1] assms biexp01-well-formedE[of a]*
by *auto*
finally have $(\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ m\ /\ 2) < 1/2$
by *simp*
thus *False*
using *h* **by** *simp*
qed
next
assume $h{:}\neg\ 1 \le (\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ m\ /\ 2) * 2$
show *a 0 = 0*
proof(*rule ccontr*)
assume $a\ 0 \neq 0$
then have *a 0 = 1*
using *assms(1) biexp01-well-formedE[of a]*
by (*meson insertE singletonD*)
hence $1/2 \le (\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ (Suc\ m))$
using *ai-1-gt[of a 0] assms(1) biexp01-well-formedE[of a]*
by *auto*
thus *False*
using *h* **by** *simp*
qed
qed
next
fix *n :: nat*
assume $ih{:}(\bigwedge m.\ m \le n \Longrightarrow r01\text{-}binary\text{-}expansion'\ (\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ Suc\ m)\ m = a\ m)$
show $r01\text{-}binary\text{-}expansion'\ (\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ Suc\ m)\ (Suc\ n) = a\ (Suc\ n)$
proof(*cases r01-binary-expansion''* $(\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ Suc\ m)\ n$)
case *h:(fields bn ur lr)*
then have $hlr{:}lr = (\sum k{=}0..n.\ real\ (a\ k) * (1\ /\ 2)\ \hat{}\ Suc\ k)$
using *r01-binary-expression-eq-lr[of* $\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ Suc\ m\ n$*] ih*
by(*simp add: r01-binary-expression-def r01-binary-sum-def*)
have $hlr2{:}(ur + lr)\ /\ 2 = lr + (1/2)\hat{}(Suc\ (Suc\ n))$
proof −
have $(ur + lr)\ /\ 2 = lr + (1/2)\hat{}(Suc\ (Suc\ n))$
using *r01-binary-expansion-diff[of* $\sum m.\ real\ (a\ m) * (1\ /\ 2)\ \hat{}\ Suc\ m\ n$*] h* **by** *simp*
show *?thesis*
by (*simp add:* ‹$(ur + lr)\ /\ 2 = lr + (1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n)$› *of-rat-add of-rat-divide of-rat-power*)

qed
show *?thesis*
using *h*
proof(*auto simp add*: *r01-binary-expansion'-def Let-def*)
assume *h1*: (*ur* + *lr*) ≤ (∑*m*. *real* (*a m*) ∗ (*1* / *2*) ^ *m* / *2*) ∗ *2*
show *Suc 0* = *a* (*Suc n*)
proof(*rule ccontr*)
assume *Suc 0* ≠ *a* (*Suc n*)
then have *a* (*Suc n*) = *0*
using *assms*(*1*) *biexp01-well-formedE*[*of a*] **by** *auto*
have (∑*m*. *real* (*a m*) ∗ (*1* / *2*) ^ *m* / *2*) < (∑*k*=*0*..*n*. *real* (*a k*) ∗ (*1* / *2*) ^ *Suc k*) + (*1*/*2*)^(*Suc* (*Suc n*))
proof −
have (∑*m*. *real* (*a m*) ∗ (*1* / *2*) ^ (*Suc m*)) = (∑*k*=*0*..*n*. *real* (*a k*) ∗ (*1* / *2*) ^ *Suc k*) + (∑*m*. *real* (*a* (*m*+*Suc n*)) ∗ (*1* / *2*) ^ *Suc* (*m* + *Suc n*))
proof −
have {*0*..*n*} = {..<*Suc n*} **by** *auto*
thus *?thesis*
using *suminf-split-initial-segment*[*of* λ*m*. *real* (*a m*) ∗ (*1* / *2*) ^ (*Suc m*) *Suc n*] *binary-expression-summable*[*of a*] *assms*(*1*) *biexp01-well-formedE*[*of a*]
by *simp*
qed
also have ... = (∑*k*=*0*..*n*. *real* (*a k*) ∗ (*1* / *2*) ^ *Suc k*) + (∑*m*. *real* (*a* (*Suc m* + *Suc n*)) ∗ (*1* / *2*) ^ *Suc* (*Suc m* + *Suc n*))
using *suminf-split-head*[*of* λ*m*. *real* (*a* (*m* + *Suc n*)) ∗ (*1* / *2*) ^ (*Suc* (*m* + *Suc n*))] *binary-expression-summable*[*of a*] *assms*(*1*) *biexp01-well-formedE*[*of a*] *Series.summable-iff-shift*[*of* λ*m*. *real* (*a m*) ∗ (*1* / *2*) ^ (*Suc m*) *Suc n*] ‹*a* (*Suc n*) = *0*›
by *simp*
also have ... = (∑*k*=*0*..*n*. *real* (*a k*) ∗ (*1* / *2*) ^ *Suc k*) + (∑*m*. *real* (*a* (*m* + *Suc* (*Suc n*))) ∗ (*1* / *2*) ^ *Suc* (*m* + *Suc* (*Suc n*)))
by *simp*
also have ... < (∑*k*=*0*..*n*. *real* (*a k*) ∗ (*1* / *2*) ^ *Suc k*) + (*1*/*2*)^*Suc* (*Suc n*)
using *ai-exists0-less-than-sum*[*of a Suc* (*Suc n*)] *assms*(*1*) *biexp01-well-formedE*[*of a*]
by *auto*
finally show *?thesis* **by** *simp*
qed
thus *False*
using *h1 hlr2 hlr* **by** *simp*
qed
next
assume *h2*:¬ *ur* + *lr* ≤ (∑*m*. *real* (*a m*) ∗ (*1* / *2*) ^ *m* / *2*) ∗ *2*
show *a* (*Suc n*) = *0*
proof(*rule ccontr*)
assume *a* (*Suc n*) ≠ *0*
then have *a* (*Suc n*) = *1*
using *biexp01-well-formedE*[*OF assms*(*1*)]

```
          by (meson insertE singletonD)
        have (∑k=0..n. real (a k) ∗ (1 / 2) ^ Suc k) + (1/2)^(Suc (Suc n)) ≤
(∑m. real (a m) ∗ (1 / 2) ^ m / 2)
        proof −
          have (∑m. real (a m) ∗ (1 / 2) ^(Suc m)) = (∑k=0..n. real (a k) ∗
(1 / 2) ^ Suc k) + (∑m. real (a (m+Suc n)) ∗ (1 / 2) ^ Suc (m + Suc n))
          proof −
            have {0..n} = {..<Suc n} by auto
            thus ?thesis
              using suminf-split-initial-segment[of λm. real (a m) ∗ (1 / 2) ^(Suc
m) Suc n] binary-expression-summable[of a] assms(1) biexp01-well-formedE[of a]
              by simp
          qed
          also have ... = (∑k=0..n. real (a k) ∗ (1 / 2) ^ Suc k) + (∑m. real (a
(Suc m + Suc n)) ∗ (1 / 2) ^ Suc (Suc m + Suc n)) + (1 / 2) ^ Suc (Suc n)
            using suminf-split-head[of λm. real (a (m + Suc n)) ∗ (1 / 2) ^(Suc (m
+ Suc n))] binary-expression-summable[of a] assms(1) biexp01-well-formedE[of a]
Series.summable-iff-shift[of λm. real (a m) ∗ (1 / 2) ^(Suc m) Suc n] ‹a (Suc n)
= 1›
            by simp
          also have ... = (∑k=0..n. real (a k) ∗ (1 / 2) ^ Suc k) + (∑m. real (a
(m + Suc (Suc n))) ∗ (1 / 2) ^ Suc (m + (Suc (Suc n)))) + (1 / 2) ^ Suc (Suc
n)
            by simp
          also have ... ≥ (∑k=0..n. real (a k) ∗ (1 / 2) ^ Suc k) + (1 / 2) ^ Suc
(Suc n)
          using binary-expression-gteq0[of a Suc (Suc n)] assms(1) biexp01-well-formedE[of
a] by simp
          finally show ?thesis by simp
        qed
        thus False
          using h2 hlr2 hlr by simp
      qed
    qed
  qed
qed


lemma f01-borel-measurable:
  assumes f −‘ {0::real} ∈ sets real-borel
          f −‘ {1} ∈ sets borel
      and ⋀r::real. f r ∈ {0,1}
    shows f ∈ borel-measurable real-borel
proof(rule measurableI)
  fix U :: real set
  assume U ∈ sets borel
  consider 1 ∈ U ∧ 0 ∈ U | 1 ∈ U ∧ 0 ∉ U | 1 ∉ U ∧ 0 ∈ U | 1 ∉ U ∧ 0 ∉ U
    by auto
  then show f −‘ U ∩ space real-borel ∈ sets borel
```

proof *cases*
  case *1*
  then have $f -` U = UNIV$
    using *assms*(*3*) **by** *auto*
  then show *?thesis* **by** *simp*
next
  case *2*
  then have $f -` U = f -` \{1\}$
    using *assms*(*3*) **by** *fastforce*
  then show *?thesis*
    using *assms*(*2*) **by** *simp*
next
  case *3*
  then have $f -` U = f -` \{0\}$
    using *assms*(*3*) **by** *fastforce*
  then show *?thesis*
    using *assms*(*1*) **by** *simp*
next
  case *4*
  then have $f -` U = \{\}$
    using *assms*(*3*) **by** (*metis all-not-in-conv insert-iff vimage-eq*)
  then show *?thesis* **by** *simp*
qed
qed *simp*

lemma *r01-binary-expansion′-measurable*:
($\lambda r.$ *real* (*r01-binary-expansion′ r n*)) $\in$ *borel-measurable* (*borel* :: *real measure*)
proof −
  have ($\lambda r.$ *real* (*r01-binary-expansion′ r n*)) −‘$\{0\}$ $\in$ *sets borel* $\wedge$ ($\lambda r.$ *real* (*r01-binary-expansion′ r n*)) −‘$\{1\}$ $\in$ *sets borel*
 proof −
   let *?A* = $\{..0::real\} \cup (\bigcup i \in \{l::nat.\ l < 2\hat{\ }(Suc\ n) \wedge even\ l\}\ .\ \{i/2\hat{\ }(Suc\ n)..<(Suc\ i)/2\hat{\ }(Suc\ n)\})$
   let *?B* = $\{1::real..\} \cup (\bigcup i \in \{l::nat.\ l < 2\hat{\ }(Suc\ n) \wedge odd\ l\}\ .\ \{i/2\hat{\ }(Suc\ n)..<(Suc\ i)/2\hat{\ }(Suc\ n)\})$
  have *?A* $\in$ *sets borel* **by** *simp*
  have *?B* $\in$ *sets borel* **by** *simp*
  have *hE*:*?A* $\cap$ *?B* = $\{\}$
  proof *auto*
    fix *r* :: *real*
    fix *l* :: *nat*
    assume *h*: $r \leq 0$
        odd l
        real l / $(2 * 2 \,\hat{}\, n) \leq r$
    then have $0 < l$ **by**(*cases l*; *auto*)
    hence $0 <$ *real l* / $(2 * 2 \,\hat{}\, n)$ **by** *simp*
    thus *False*
      using *h* **by** *simp*

```
next
  fix r :: real
  fix l :: nat
  assume h: l < 2 * 2 ^ n
          even l
          1 ≤ r
          r < (1 + real l) / (2 * 2 ^ n)
  then have 1 + real l ≤ 2 * 2 ^ n
    by (simp add: nat-less-real-le)
  moreover have 1 + real l ≠ 2 * 2 ^ n
    using h by auto
  ultimately have 1 + real l < 2 * 2 ^ n by simp
  hence (1 + real l) / (2 * 2 ^ n) < 1 by simp
  thus False using h by linarith
next
  fix r :: real
  fix l1 l2 :: nat
  assume h: even l1 odd l2
          real l1 / (2 * 2 ^ n) ≤ r r < (1 + real l1) / (2 * 2 ^ n)
          real l2 / (2 * 2 ^ n) ≤ r r < (1 + real l2) / (2 * 2 ^ n)
  then consider l1 < l2 | l2 < l1  by fastforce
  thus False
  proof cases
    case 1
    then have (1 + real l1) / (2 * 2 ^ n) ≤ real l2 / (2 * 2 ^ n)
      by (simp add: frac-le)
    then show ?thesis
      using h by simp
  next
    case 2
    then have (1 + real l2) / (2 * 2 ^ n) ≤ real l1 / (2 * 2 ^ n)
      by (simp add: frac-le)
    then show ?thesis
      using h by simp
  qed
qed
have hU:?A ∪ ?B = UNIV
proof
  show ?A ∪ ?B ⊆ UNIV by simp
next
  show UNIV ⊆ ?A ∪ ?B
  proof
    fix r :: real
    consider r ≤ 0 | 0 < r ∧ r < 1 | 1 ≤ r by linarith
    then show r ∈ ?A ∪ ?B
    proof cases
      case 1
      then show ?thesis by simp
    next
```

```
      case 2
      show ?thesis
      proof(cases r01-binary-expansion'' r n)
        case hc:(fields a ur lr)
        then have hlu:lr ≤ r ∧ r < ur
          using 2 r01-binary-expansion-lr-r-ur[of r n] by simp
        obtain k :: nat where hk:
         lr = real k / 2 ^ Suc n ∧ k < 2 ^ Suc n
          using r01-binary-expression'-sum-range[of r n] hc
          by auto
        hence ur = real (Suc k) / 2^Suc n
          using r01-binary-expansion-diff[of r n] hc
          by (simp add: add-divide-distrib power-one-over)
        thus ?thesis
          using hlu hk by auto
      qed
    next
      case 3
      then show ?thesis by simp
    qed
  qed
qed
have hi1:− ?A = ?B
proof −
  have ?B ⊆ − ?A
    using hE by blast
  moreover have −?A ⊆ ?B
  proof −
    have −(?A ∪ ?B) = {}
      using hU by simp
    hence (−?A) ∩ (−?B) = {} by simp
    thus ?thesis
      by blast
  qed
  ultimately show ?thesis
    by blast
qed
have hi2: ?A = −?B
  using hi1 by blast

let ?U0 = (λr. real (r01-binary-expansion' r n)) −‘{0}
let ?U1 = (λr. real (r01-binary-expansion' r n)) −‘{1}

have hU':?U0 ∪ ?U1 = UNIV
proof −
  have ?U0 ∪ ?U1 = (λr. real (r01-binary-expansion' r n)) −‘{0,1}
    by auto
  thus ?thesis
    using real01-binary-expansion'-0or1[of - n] by auto
```

qed
have *hE′*:*?U0* ∩ *?U1* = {}
  by *auto*

have *hiu1*:− *?U0* = *?U1*
  using *hE′ hU′* **by** *fastforce*

have *hiu2*:− *?U1* = *?U0*
  using *hE′ hU′* **by** *fastforce*

have *?U0* ⊆ *?A*
proof
  fix *r*
  assume *r* ∈ *?U0*
  then have *h1*:*r01-binary-expansion′ r n* = *0*
    by *simp*
  then consider *r* ≤ *0* | *0* < *r* ∧ *r* < *1*
    using *r01-binary-expansion′-gt1*[*of r*] **by** *fastforce*
  thus *r* ∈ *?A*
  proof *cases*
    case *1*
    then show *?thesis* **by** *simp*
  next
    case *2*
    then have *3*:(*snd* (*snd* (*r01-binary-expansion′′ r n*))) ≤ *r* ∧
          r < (*fst* (*snd* (*r01-binary-expansion′′ r n*)))
      using *r01-binary-expansion-lr-r-ur*[*of r n*] **by** *simp*
    obtain *k* **where** *4*:
      (*snd* (*snd* (*r01-binary-expansion′′ r n*))) =
       real k / *2* ^ *Suc n* ∧
       k < *2* ^ *Suc n* ∧ *even k*
      using *r01-binary-expression′-sum-range*[*of r n*] *h1*
      by *auto*
    have (*fst* (*snd* (*r01-binary-expansion′′ r n*))) = *real* (*Suc k*) / *2* ^ *Suc n*
    proof −
     have (*fst* (*snd* (*r01-binary-expansion′′ r n*))) = (*snd* (*snd* (*r01-binary-expansion′′*
r n*))) + (*1*/*2*)^*Suc n*
        using *r01-binary-expansion-diff*[*of r n*] **by** *linarith*
      thus *?thesis*
        using *4*
        by (*simp add*: *add-divide-distrib power-one-over*)
    qed
    thus *?thesis*
      using *3 4* **by** *auto*
  qed
qed

have *?U1* ⊆ *?B*
proof

```
    fix r
    assume r ∈ ?U1
    then have h1:r01-binary-expansion′ r n = 1
      by simp
    then consider 1 ≤ r | 0 < r ∧ r < 1
      using r01-binary-expansion′-lt0[of r] by fastforce
    thus r ∈ ?B
    proof cases
      case 1
      then show ?thesis by simp
    next
      case 2
      then have 3:(snd (snd (r01-binary-expansion′′ r n))) ≤ r ∧
                   r < (fst (snd (r01-binary-expansion′′ r n)))
        using r01-binary-expansion-lr-r-ur[of r n] by simp
      obtain k where 4:
        (snd (snd (r01-binary-expansion′′ r n))) =
         real k / 2 ^ Suc n ∧
         k < 2 ^ Suc n ∧ odd k
        using StandardBorel.r01-binary-expression′-sum-range[of r n] h1
        by auto
      have (fst (snd (r01-binary-expansion′′ r n))) = real (Suc k) / 2 ^ Suc n
      proof −
       have (fst (snd (r01-binary-expansion′′ r n))) = (snd (snd (r01-binary-expansion′′
r n))) + (1/2)^Suc n
          using r01-binary-expansion-diff[of r n] by simp
        thus ?thesis
          using 4
          by (simp add: add-divide-distrib power-one-over)
      qed
      thus ?thesis
        using 3 4 by auto
    qed
  qed

  have ?U0 = ?A
  proof
    show ?U0 ⊆ ?A by fact
  next
    show ?A ⊆ ?U0
      using ‹?U1 ⊆ ?B› Compl-subset-Compl-iff[of ?U0 ?A] hi1 hiu1
      by blast
  qed

  have ?U1 = ?B
    using ‹?U0 = ?A› hi1 hiu1 by auto
  show ?thesis
    using ‹?U0 = ?A› ‹?U1 = ?B› ‹?A ∈ sets borel› ‹?B ∈ sets borel›
    by simp
```

qed
thus *?thesis*
using *f01-borel-measurable*[*of* (*λr. real* (*r01-binary-expansion′ r n*))] *real01-binary-expansion′-0or1*[*of - n*]
by *simp*
qed

definition *r01-to-r01-r01-fst′* :: *real* ⇒ *nat* ⇒ *nat* **where**
r01-to-r01-r01-fst′ r n ≡ *r01-binary-expansion′ r* (*2∗n*)

lemma *r01-to-r01-r01-fst′in01*:
⋀*n. r01-to-r01-r01-fst′ r n* ∈ {*0*,*1*}
using *real01-binary-expansion′-0or1* **by** (*simp add*: *r01-to-r01-r01-fst′-def*)

definition *r01-to-r01-r01-fst-sum* :: *real* ⇒ *nat* ⇒ *real* **where**
r01-to-r01-r01-fst-sum ≡ *r01-binary-sum* ∘ *r01-to-r01-r01-fst′*

definition *r01-to-r01-r01-fst* :: *real* ⇒ *real* **where**
r01-to-r01-r01-fst = *lim* ∘ *r01-to-r01-r01-fst-sum*

lemma *r01-to-r01-r01-fst-def′*:
r01-to-r01-r01-fst r = (∑ *n. real* (*r01-binary-expansion′ r* (*2∗n*)) ∗ (*1/2*)^(*n+1*))
proof −
have *r01-to-r01-r01-fst-sum r* = (*λn.* ∑ *i=0..n. real* (*r01-binary-expansion′ r* (*2∗i*)) ∗ (*1/2*)^(*i+1*))
by(*auto simp add*: *r01-to-r01-r01-fst-sum-def r01-binary-sum-def r01-to-r01-r01-fst′-def*)
thus *?thesis*
using *lim-sum-ai real01-binary-expansion′-0or1*
by(*simp add*: *r01-to-r01-r01-fst-def*)
qed

lemma *r01-to-r01-r01-fst-measurable*:
r01-to-r01-r01-fst ∈ *borel-measurable borel*
unfolding *r01-to-r01-r01-fst-def′*
using *r01-binary-expansion′-measurable* **by** *auto*

definition *r01-to-r01-r01-snd′* :: *real* ⇒ *nat* ⇒ *nat* **where**
r01-to-r01-r01-snd′ r n = *r01-binary-expansion′ r* (*2∗n* + *1*)

lemma *r01-to-r01-r01-snd′in01*:
⋀*n. r01-to-r01-r01-snd′ r n* ∈ {*0*,*1*}
using *real01-binary-expansion′-0or1* **by** (*simp add*: *r01-to-r01-r01-snd′-def*)

definition *r01-to-r01-r01-snd-sum* :: *real* ⇒ *nat* ⇒ *real* **where**

r01-to-r01-r01-snd-sum ≡ *r01-binary-sum* ∘ *r01-to-r01-r01-snd′*

definition *r01-to-r01-r01-snd* :: *real* ⇒ *real* **where**
r01-to-r01-r01-snd = *lim* ∘ *r01-to-r01-r01-snd-sum*

lemma *r01-to-r01-r01-snd-def′*:
r01-to-r01-r01-snd r = (∑ *n*. *real* (*r01-binary-expansion′ r* (*2*∗*n* + *1*)) ∗ (*1*/*2*)^(*n*+*1*))
proof −
have *r01-to-r01-r01-snd-sum r* = (λ*n*. ∑ *i*=*0*..*n*. *real* (*r01-binary-expansion′ r* (*2*∗*i* + *1*)) ∗ (*1*/*2*)^(*i*+*1*))
by(*auto simp add*: *r01-to-r01-r01-snd-sum-def r01-binary-sum-def r01-to-r01-r01-snd′-def*)
thus *?thesis*
using *lim-sum-ai real01-binary-expansion′-0or1*
by(*simp add*: *r01-to-r01-r01-snd-def*)
qed

lemma *r01-to-r01-r01-snd-measurable*:
r01-to-r01-r01-snd ∈ *borel-measurable borel*
unfolding *r01-to-r01-r01-snd-def′*
using *r01-binary-expansion′-measurable* **by** *auto*

definition *r01-to-r01-r01* :: *real* ⇒ *real* × *real* **where**
r01-to-r01-r01 r = (*r01-to-r01-r01-fst r*,*r01-to-r01-r01-snd r*)

lemma *r01-to-r01-r01-image*:
r01-to-r01-r01 r ∈ {*0*..*1*}×{*0*..*1*}
using *r01-to-r01-r01-fst-def′*[*of r*] *r01-to-r01-r01-snd-def′*[*of r*] *real01-binary-expansion′-0or1* *binary-expression-gteq0*[*of* λ*n*. *r01-binary-expansion′ r* (*2*∗*n*) *0*] *binary-expression-leeq1*[*of* λ*n*. *r01-binary-expansion′ r* (*2*∗*n*) *0*] *binary-expression-gteq0*[*of* λ*n*. *r01-binary-expansion′ r* (*2*∗*n*+*1*) *0*] *binary-expression-leeq1*[*of* λ*n*. *r01-binary-expansion′ r* (*2*∗*n*+*1*) *0*]
by(*simp add*: *r01-to-r01-r01-def*)

lemma *r01-to-r01-r01-measurable*:
r01-to-r01-r01 ∈ *real-borel* $\rightarrow_M$ *real-borel* $\bigotimes_M$ *real-borel*
unfolding *r01-to-r01-r01-def*
using *borel-measurable-Pair*[*of r01-to-r01-r01-fst borel r01-to-r01-r01-snd*] *r01-to-r01-r01-fst-measurable r01-to-r01-r01-snd-measurable*
by(*simp add*: *borel-prod*)

lemma *r01-to-r01-r01-3over4*:
r01-to-r01-r01 (*3*/*4*) = (*1*/*2*,*1*/*2*)
proof −
have *h0*:*r01-binary-expansion′* (*3*/*4*) *0* = *1*
by (*simp add*: *r01-binary-expansion′-def*)
have *h1*:*r01-binary-expansion′* (*3*/*4*) *1* = *1*
by (*simp add*: *r01-binary-expansion′-def Let-def of-rat-divide*)
have *hn*:⋀*n*. *n*>*1*⟹ *r01-binary-expansion′* (*3*/*4*) *n* = *0*
proof −

fix *n* :: *nat*
assume *h*:*1* < *n*
show *r01-binary-expansion'* (*3* / *4*) *n* = *0*
proof(*rule ccontr*)
assume *r01-binary-expansion'* (*3* / *4*) *n* ≠ *0*
have *3*/*4* < (∑ *i*=*0*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*))
proof −
have (∑ *i*=*0*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)) = *real* (*r01-binary-expansion'* (*3*/*4*) *0*) ∗ (*1*/*2*)^(*Suc* *0*) + (∑ *i*=(*Suc* *0*)..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*))
by(*rule sum.atLeast-Suc-atMost*) (*simp add*: *h*)
also have ... = *real* (*r01-binary-expansion'* (*3*/*4*) *0*) ∗ (*1*/*2*)^(*Suc* *0*) + (*real* (*r01-binary-expansion'* (*3*/*4*) *1*) ∗ (*1*/*2*)^(*Suc* *1*) + (∑ *i*=(*Suc* (*Suc* *0*))..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)))
using *sum.atLeast-Suc-atMost*[*OF order.strict-implies-order*[*OF h*],*of* λ*i*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)]
by *simp*
also have ... = *3*/*4* + (∑ *i*=*2*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*))
using *h0 h1* **by**(*simp add*: *numeral-2-eq-2*)
also have ... > *3*/*4*
proof −
have (∑ *i*=*2*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)) = (∑ *i*=*2*..*n*−*1*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)) + *real* (*r01-binary-expansion'* (*3*/*4*) *n*) ∗ (*1*/*2*)^*Suc n*
by (*metis* (*no-types*, *lifting*) *h One-nat-def Suc-pred less-2-cases-iff less-imp-add-positive order-less-irrefl plus-1-eq-Suc sum.cl-ivl-Suc zero-less-Suc*)
hence *real* (*r01-binary-expansion'* (*3*/*4*) *n*) ∗ (*1*/*2*)^*Suc n* ≤ (∑ *i*=*2*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*))
using *ordered-comm-monoid-add-class.sum-nonneg*[*of* {*2*..*n*−*1*} λ*i*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)]
by *simp*
moreover have *0* < *real* (*r01-binary-expansion'* (*3*/*4*) *n*) ∗ (*1*/*2*)^*Suc n*
using ‹*r01-binary-expansion'* (*3* / *4*) *n* ≠ *0*› **by** *simp*
ultimately have *0* < (∑ *i*=*2*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*))
by *simp*
thus *?thesis* **by** *simp*
qed
finally show *3* / *4* < (∑ *i* = *0*..*n*. *real* (*r01-binary-expansion'* (*3* / *4*) *i*) ∗ (*1* / *2*) ^ *Suc i*) **.**
qed
moreover have (∑ *i*=*0*..*n*. *real* (*r01-binary-expansion'* (*3*/*4*) *i*) ∗ (*1*/*2*)^(*Suc* *i*)) ≤ *3*/*4*
using *r01-binary-expansion-lr-r-ur*[*of 3*/*4 n*] *r01-binary-expression-eq-lr*[*of 3*/*4 n*]
by(*simp add*: *r01-binary-expression-def r01-binary-sum-def*)
ultimately show *False* **by** *simp*

qed
qed
show *?thesis*
proof
have *fst* (*r01-to-r01-r01* (*3* / *4*)) = ($\sum$ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *n*)) ∗ (*1* / *2*) ^ *Suc n*)
by(*simp add*: *r01-to-r01-r01-def r01-to-r01-r01-fst-def'*)
also have ... = *1*/*2* + ($\sum$ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *Suc n*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n*))
using *suminf-split-head*[*of* λ*n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *n*)) ∗ (*1* / *2*) ^ *Suc n*] *binary-expression-summable*[*of* λ*n*. *r01-binary-expansion'* (*3*/*4*) (*2*∗*n*)] *real01-binary-expansion'-0or1*[*of 3*/*4*] *h0*
by *simp*
also have ... = *1*/*2*
proof −
have ∀ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *Suc n*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n*) = *0*
using *hn* **by** *simp*
hence ($\sum$ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *Suc n*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n*)) = *0*
by *simp*
thus *?thesis*
by *simp*
qed
finally show *fst* (*r01-to-r01-r01* (*3* / *4*)) = *fst* (*1* / *2*, *1* / *2*)
by *simp*
next
have *snd* (*r01-to-r01-r01* (*3* / *4*)) = ($\sum$ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *n* + *1*)) ∗ (*1* / *2*) ^ *Suc n*)
by(*simp add*: *r01-to-r01-r01-def r01-to-r01-r01-snd-def'*)
also have ... = *1*/*2* + ($\sum$ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *Suc n* + *1*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n*))
using *suminf-split-head*[*of* λ*n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *n* + *1*)) ∗ (*1* / *2*) ^ *Suc n*] *binary-expression-summable*[*of* λ*n*. *r01-binary-expansion'* (*3*/*4*) (*2*∗*n* + *1*)] *real01-binary-expansion'-0or1*[*of 3*/*4*] *h1*
by *simp*
also have ... = *1*/*2*
proof −
have ∀ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *Suc n* + *1*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n*) = *0*
using *hn* **by** *simp*
hence ($\sum$ *n*. *real* (*r01-binary-expansion'* (*3* / *4*) (*2* ∗ *Suc n* + *1*)) ∗ (*1* / *2*) ^ *Suc* (*Suc n*)) = *0*
by *simp*
thus *?thesis*
by *simp*
qed
finally show *snd* (*r01-to-r01-r01* (*3* / *4*)) = *snd* (*1* / *2*, *1* / *2*)
by *simp*

qed
qed

definition *r01-r01-to-r01′* :: *real* × *real* ⇒ *nat* ⇒ *nat* **where**
r01-r01-to-r01′ rs ≡ (*λn. if even n then r01-binary-expansion′* (*fst rs*) (*n div 2*)
else r01-binary-expansion′ (*snd rs*) ((*n*−*1*) *div 2*))

lemma *r01-r01-to-r01′in01*:
⋀*n. r01-r01-to-r01′ rs n* ∈ {*0*,*1*}
using *real01-binary-expansion′-0or1* **by** (*simp add*: *r01-r01-to-r01′-def*)

lemma *r01-r01-to-r01′-well-formed*:
assumes *0* < *r1 r1* < *1*
and *0* < *r2 r2* < *1*
shows *biexp01-well-formed* (*r01-r01-to-r01′* (*r1*,*r2*))
using *biexp01-well-formed-comb*[*of r01-binary-expansion′* (*fst* (*r1*,*r2*)) *r01-binary-expansion′* (*snd* (*r1*,*r2*))] *r01-binary-expansion-well-formed*[*of r1*] *r01-binary-expansion-well-formed*[*of r2*] *assms*
by (*auto simp add*: *r01-r01-to-r01′-def*)

definition *r01-r01-to-r01-sum* :: *real* × *real* ⇒ *nat* ⇒ *real* **where**
r01-r01-to-r01-sum ≡ *r01-binary-sum* ∘ *r01-r01-to-r01′*

definition *r01-r01-to-r01* :: *real* × *real* ⇒ *real* **where**
r01-r01-to-r01 ≡ *lim* ∘ *r01-r01-to-r01-sum*

lemma *r01-r01-to-r01-def′*:
r01-r01-to-r01 (*r1*,*r2*) = (∑ *n. real* (*r01-r01-to-r01′* (*r1*,*r2*) *n*) ∗ (*1*/*2*)^(*n*+*1*))
proof −
have *r01-r01-to-r01-sum* (*r1*,*r2*) = (*λn.* (∑ *i* = *0*..*n. real* (*r01-r01-to-r01′* (*r1*,*r2*) *i*) ∗ (*1* / *2*) ^ *Suc i*))
by(*auto simp add*: *r01-r01-to-r01-sum-def r01-binary-sum-def*)
thus *?thesis*
using *lim-sum-ai*[*of λn. r01-r01-to-r01′* (*r1*,*r2*) *n*] *r01-r01-to-r01′in01*
by(*simp add*: *r01-r01-to-r01-def*)
qed

lemma *r01-r01-to-r01-measurable*:
r01-r01-to-r01 ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *real-borel*
proof −
have *r01-r01-to-r01* = (*λx.* ∑ *n. real* (*r01-r01-to-r01′ x n*) ∗ (*1*/*2*)^(*n*+*1*))
using *r01-r01-to-r01-def′* **by** *auto*
also have ... ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *real-borel*
proof(*rule borel-measurable-suminf*)
fix *n* :: *nat*
have (*λx. real* (*r01-r01-to-r01′ x n*) ∗ (*1* / *2*) ^ (*n* + *1*)) = (*λr. r* ∗ (*1*/*2*)^(*n*+*1*)) ∘ (*λx. real* (*r01-r01-to-r01′ x n*))

by *auto*
also have ... ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
proof(*rule measurable-comp*[*of - - borel*])
have (*λx. real* (*r01-r01-to-r01′ x n*))
= (*λx. if even n then real* (*r01-binary-expansion′* (*fst x*) (*n div 2*)) *else real* (*r01-binary-expansion′* (*snd x*) ((*n* − *1*) *div 2*)))
by (*auto simp add*: *r01-r01-to-r01′-def*)
also have ... ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
using *r01-binary-expansion′-measurable* **by** *simp*
finally show (*λx. real* (*r01-r01-to-r01′ x n*)) ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*) **.**
next
show (*λr::real. r* ∗ (*1* / *2*) ^(*n* + *1*)) ∈ *borel-measurable borel*
by *simp*
qed
finally show (*λx. real* (*r01-r01-to-r01′ x n*) ∗ (*1* / *2*) ^(*n* + *1*)) ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*) **.**
qed
finally show *?thesis* **.**
qed

lemma *r01-r01-to-r01-image*:
assumes *0* < *r1 r1* < *1*
shows *r01-r01-to-r01* (*r1*,*r2*) ∈ {*0*<..<*1*}
proof −
obtain *i* **where** *r01-binary-expansion′ r1 i* = *1*
using *r01-binary-expression-ex1*[*of r1*] *assms*(*1*,*2*)
by *auto*
hence *hi*:*r01-r01-to-r01′* (*r1*,*r2*) (*2*∗*i*) = *1*
by(*simp add*: *r01-r01-to-r01′-def*)
obtain *j* **where** *r01-binary-expansion′ r1 j* = *0*
using *r01-binary-expression-ex0*[*of r1*] *assms*(*1*,*2*)
by *auto*
hence *hj*:*r01-r01-to-r01′* (*r1*,*r2*) (*2*∗*j*) = *0*
by(*simp add*: *r01-r01-to-r01′-def*)
show *?thesis*
using *ai-exists1-gt0*[*of r01-r01-to-r01′* (*r1*,*r2*)] *ai-exists0-less-than1*[*of r01-r01-to-r01′* (*r1*,*r2*)] *r01-r01-to-r01′in01*[*of* (*r1*,*r2*)] *r01-r01-to-r01-def′*[*of r1 r2*] *hi hj*
by *auto*
qed

lemma *r01-r01-to-r01-image′*:
assumes *0* < *r2 r2* < *1*
shows *r01-r01-to-r01* (*r1*,*r2*) ∈ {*0*<..<*1*}
proof −
obtain *i* **where** *r01-binary-expansion′ r2 i* = *1*
using *r01-binary-expression-ex1*[*of r2*] *assms*(*1*,*2*)
by *auto*
hence *hi*:*r01-r01-to-r01′* (*r1*,*r2*) (*2*∗*i* + *1*) = *1*

by(*simp add*: *r01-r01-to-r01′-def*)
obtain *j* **where** *r01-binary-expansion′ r2 j = 0*
using *r01-binary-expression-ex0*[*of r2*] *assms*(*1*,*2*)
by *auto*
hence *hj*:*r01-r01-to-r01′* (*r1*,*r2*) (*2*∗*j* + *1*) = *0*
by(*simp add*: *r01-r01-to-r01′-def*)
show *?thesis*
using *ai-exists1-gt0*[*of r01-r01-to-r01′* (*r1*,*r2*)] *ai-exists0-less-than1*[*of r01-r01-to-r01′* (*r1*,*r2*)] *r01-r01-to-r01′in01*[*of* (*r1*,*r2*)] *r01-r01-to-r01-def′*[*of r1 r2*] *hi hj*
by *auto*
qed

lemma *r01-r01-to-r01-binary-nth*:
assumes *0* < *r1 r1* < *1*
and *0* < *r2 r2* < *1*
shows *r01-binary-expansion′ r1 n* = *r01-binary-expansion′* (*r01-r01-to-r01* (*r1*,*r2*)) (*2*∗*n*) ∧
r01-binary-expansion′ r2 n = *r01-binary-expansion′* (*r01-r01-to-r01* (*r1*,*r2*)) (*2*∗*n* + *1*)
proof −
have ⋀*n*. *r01-binary-expansion′* (*r01-r01-to-r01* (*r1*,*r2*)) *n* = *r01-r01-to-r01′* (*r1*,*r2*) *n*
using *r01-r01-to-r01-def′*[*of r1 r2*] *biexp01-well-formed-an*[*of r01-r01-to-r01′* (*r1*,*r2*)] *r01-r01-to-r01′-well-formed*[*of r1 r2*] *assms*
by *simp*
thus *?thesis*
by(*simp add*: *r01-r01-to-r01′-def*)
qed

lemma *r01-r01--r01--r01-r01-id*:
assumes *0* < *r1 r1* < *1*
0 < *r2 r2* < *1*
shows (*r01-to-r01-r01* ∘ *r01-r01-to-r01*) (*r1*,*r2*) = (*r1*,*r2*)
proof
show *fst* ((*r01-to-r01-r01* ∘ *r01-r01-to-r01*) (*r1*, *r2*)) = *fst* (*r1*, *r2*)
proof −
have *fst* ((*r01-to-r01-r01* ∘ *r01-r01-to-r01*) (*r1*, *r2*)) = *r01-to-r01-r01-fst* (*r01-r01-to-r01* (*r1*,*r2*))
by(*simp add*: *r01-to-r01-r01-def*)
also have ... = (∑*n*. *real* (*r01-binary-expansion′* (*r01-r01-to-r01* (*r1*, *r2*)) (*2* ∗ *n*)) ∗ (*1* / *2*) ^ (*n* + *1*))
using *r01-to-r01-r01-fst-def′*[*of r01-r01-to-r01* (*r1*,*r2*)] **by** *simp*
also have ... = (∑*n*. *real* (*r01-binary-expansion′ r1 n*) ∗ (*1* / *2*) ^ (*n* + *1*))
using *r01-r01-to-r01-binary-nth*[*of r1 r2*] *assms* **by** *simp*
also have ... = *r1*
using *r01-binary-expression-correct*[*of r1*] *assms*(*1*,*2*)
by *simp*
finally show *?thesis* **by** *simp*

qed
next
show *snd* ((*r01-to-r01-r01* ∘ *r01-r01-to-r01*) (*r1*, *r2*)) = *snd* (*r1*, *r2*)
proof −
have *snd* ((*r01-to-r01-r01* ∘ *r01-r01-to-r01*) (*r1*, *r2*)) = *r01-to-r01-r01-snd* (*r01-r01-to-r01* (*r1*,*r2*))
by(*simp add*: *r01-to-r01-r01-def*)
also have ... = ($\sum$ *n*. *real* (*r01-binary-expansion′* (*r01-r01-to-r01* (*r1*, *r2*)) (*2* ∗ *n* + *1*)) ∗ (*1* / *2*) ^ (*n* + *1*))
using *r01-to-r01-r01-snd-def′*[*of r01-r01-to-r01* (*r1*,*r2*)] **by** *simp*
also have ... = ($\sum$ *n*. *real* (*r01-binary-expansion′ r2 n*) ∗ (*1* / *2*) ^ (*n* + *1*))
using *r01-r01-to-r01-binary-nth*[*of r1 r2*] *assms* **by** *simp*
also have ... = *r2*
using *r01-binary-expression-correct*[*of r2*] *assms*(*3*,*4*)
by *simp*
finally show *?thesis* **by** *simp*
qed
qed

We first show that $M \bigotimes_M N$ is a standard Borel space for standard Borel spaces M and N.

lemma *pair-measurable*[*measurable*]:
assumes *f* ∈ *X* $\rightarrow_M$ *Y*
and *g* ∈ *X′* $\rightarrow_M$ *Y′*
shows *map-prod f g* ∈ *X* $\bigotimes_M$ *X′* $\rightarrow_M$ *Y* $\bigotimes_M$ *Y′*
using *assms* **by**(*auto simp add*: *measurable-pair-iff*)

lemma *pair-standard-borel-standard*:
assumes *standard-borel M*
and *standard-borel N*
shows *standard-borel* (*M* $\bigotimes_M$ *N*)
proof −
— First, define the measurable function $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$.
define *rr-to-r* :: *real* × *real* ⇒ *real*
where *rr-to-r* ≡ *real-to-01open-inverse* ∘ *r01-r01-to-r01* ∘ (λ(*x*,*y*). (*real-to-01open x*, *real-to-01open y*))
— $\mathbb{R} \times \mathbb{R} \to (0,1) \times (0,1) \to (0,1) \to \mathbb{R}$.
have *1*[*measurable*]: *rr-to-r* ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *real-borel*
proof −
have (λ(*x*,*y*). (*real-to-01open x*, *real-to-01open y*)) ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *real-borel* $\bigotimes_M$ *real-borel*
using *borel-measurable-continuous-onI*[*OF real-to-01open-continuous*]
by *simp*
from *measurable-restrict-space2*[*OF - this*,*of* {*0*<..<*1*}×{*0*<..<*1*}]
have [*measurable*]:(λ(*x*,*y*). (*real-to-01open x*, *real-to-01open y*)) ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *restrict-space* (*real-borel* $\bigotimes_M$ *real-borel*) ({*0*<..<*1*}×{*0*<..<*1*})
by(*simp add*: *split-beta′ real-to-01open-01*)
have [*measurable*]: *r01-r01-to-r01* ∈ *restrict-space* (*real-borel* $\bigotimes_M$ *real-borel*) ({*0*<..<*1*}×{*0*<..<*1*}) $\rightarrow_M$ *restrict-space real-borel* {*0*<..<*1*}

using *r01-r01-to-r01-image'* **by**(*auto intro*!: *measurable-restrict-space3*[*OF r01-r01-to-r01-measurable*])
thus *?thesis*
using *borel-measurable-continuous-on-restrict*[*OF real-to-01open-inverse-continuous*]
by(*simp add*: *rr-to-r-def*)
qed
— Next, define the measurable function $\mathbb{R} \to \mathbb{R} \times \mathbb{R}$.
define *r-to-01* :: *real* ⇒ *real*
where *r-to-01* ≡ (λ*r*. *if r* ∈ *real-to-01open* −' (*r01-to-r01-r01* −' ({*0*<..<*1*}×{*0*<..<*1*})) *then real-to-01open r else 3/4*)
define *r01-to-r01-r01'* :: *real* ⇒ *real* × *real*
where *r01-to-r01-r01'* ≡ (λ*r*. *if r* ∈ *r01-to-r01-r01* −' ({*0*<..<*1*}×{*0*<..<*1*}) *then r01-to-r01-r01 r else* (*1/2*,*1/2*))
define *r-to-rr* :: *real* ⇒ *real* × *real*
where *r-to-rr* ≡ (λ(*x*,*y*). (*real-to-01open-inverse x*, *real-to-01open-inverse y*)) ∘ *r01-to-r01-r01'* ∘ *r-to-01*
— $\mathbb{R} \to (0,1) \to (0,1) \times (0,1) \to \mathbb{R} \times \mathbb{R}$.
have *2*[*measurable*]: *r-to-rr* ∈ *real-borel* $\to_M$ *real-borel* $\bigotimes_M$ *real-borel*
proof −
have *1*: {*0*<..<*1*}×{*0*<..<*1*} ∈ *sets* (*restrict-space* (*real-borel* $\bigotimes_M$ *real-borel*) ({*0*..*1*}×{*0*..*1*}))
by(*auto simp*: *sets-restrict-space-iff*)
have *2*[*measurable*]: *real-to-01open* ∈ *real-borel* $\to_M$ *restrict-space real-borel* {*0*<..<*1*}
using *measurable-restrict-space2*[*OF - borel-measurable-continuous-onI*[*OF real-to-01open-continuous*] ,*of* {*0*<..<*1*}]
by(*simp add*: *real-to-01open-01*)
have *3*: *real-to-01open* −' *space* (*restrict-space real-borel* {*0*<..<*1*}) = *UNIV*
using *real-to-01open-01* **by** *auto*
have *r01-to-r01-r01* ∈ *restrict-space real-borel* {*0*<..<*1*} $\to_M$ *restrict-space* (*real-borel* $\bigotimes_M$ *real-borel*) ({*0*..*1*}×{*0*..*1*})
using *r01-to-r01-r01-image measurable-restrict-space3*[*OF r01-to-r01-r01-measurable*] **by** *simp*
note *4* = *measurable-sets*[*OF this 1*]
note *5* = *measurable-sets*[*OF 2 4*,*simplified vimage-Int 3*,*simplified*]
have [*measurable*]:*r-to-01* ∈ *real-borel* $\to_M$ *restrict-space real-borel* {*0*<..<*1*}
unfolding *r-to-01-def*
by(*rule measurable-If-set*) (*auto intro*!: *measurable-restrict-space2 simp*: *5*)
have [*measurable*]: *r01-to-r01-r01'* ∈ *restrict-space real-borel* {*0*<..<*1*} $\to_M$ *restrict-space* (*real-borel* $\bigotimes_M$ *real-borel*) ({*0*<..<*1*}×{*0*<..<*1*})
using *4 r01-to-r01-r01-measurable*
by(*auto intro*!: *measurable-restrict-space3 simp*: *r01-to-r01-r01'-def*)
have [*measurable*]: (λ(*x*,*y*). (*real-to-01open-inverse x*, *real-to-01open-inverse y*)) ∈ *restrict-space* (*real-borel* $\bigotimes_M$ *real-borel*) ({*0*<..<*1*}×{*0*<..<*1*}) $\to_M$ *real-borel* $\bigotimes_M$ *real-borel*
using *borel-measurable-continuous-on-restrict*[*OF continuous-on-Pair*[*OF continuous-on-compose*[*of* {*0*<..<*1*::*real*}×{*0*<..<*1*::*real*},*OF continuous-on-fst*[*OF continuous-on-id'*],*simplified*,*OF real-to-01open-inverse-continuous*] *continuous-on-compose*[*of* {*0*<..<*1*::*real*}×{*0*<..<*1*::*real*},*OF continuous-on-snd*[*OF continuous-on-id'*],*simplified*,*OF*

real-to-01open-inverse-continuous]]]
    by(*simp add*: *split-beta′ borel-prod*)
  show *?thesis*
    by(*simp add*: *r-to-rr-def*)
 qed
 have *3*: $\bigwedge x.$ *r-to-rr* (*rr-to-r x*) = *x*
  using *r01-to-r01-r01-image r01-r01-to-r01-image r01-r01--r01--r01-r01-id real-to-01open-01 real-to-01open-inverse-correct′ fun-cong*[*OF real-to-01open-inverse-correct*]
  by(*auto simp add*: *r01-to-r01-r01′-def r-to-01-def comp-def split-beta′ r-to-rr-def rr-to-r-def*)

 interpret *s1*: *standard-borel M* **by** *fact*
 interpret *s2*: *standard-borel N* **by** *fact*
 show *?thesis*
   by(*auto intro*!: *standard-borelI*[**where** *f*=*rr-to-r* ∘ *map-prod s1.f s2.f* **and** *g*=*map-prod s1.g s2.g* ∘ *r-to-rr*] *simp*: *3 space-pair-measure*)
qed

lemma *pair-standard-borel-spaceUNIV*:
 assumes *standard-borel-space-UNIV M*
   and *standard-borel-space-UNIV N*
  shows *standard-borel-space-UNIV* ($M \bigotimes_M N$)
 apply(*rule standard-borel-space-UNIVI′*)
 using *assms pair-standard-borel-standard*[*of M N*]
 by(*auto simp add*: *standard-borel-space-UNIV-def standard-borel-space-UNIV-axioms-def space-pair-measure*)

locale *pair-standard-borel* = *s1*: *standard-borel M* + *s2*: *standard-borel N*
 for *M* :: *′a measure* **and** *N* :: *′b measure*
begin

sublocale *standard-borel* $M \bigotimes_M N$
 by(*auto intro*!: *pair-standard-borel-standard*)

end

locale *pair-standard-borel-space-UNIV* = *s1*: *standard-borel-space-UNIV M* + *s2*: *standard-borel-space-UNIV N*
 for *M* :: *′a measure* **and** *N* :: *′b measure*
begin

sublocale *pair-standard-borel M N*
 by *standard*

sublocale *standard-borel-space-UNIV* $M \bigotimes_M N$
 by(*auto intro*!: *pair-standard-borel-spaceUNIV*
  simp: *s1.standard-borel-space-UNIV-axioms s2.standard-borel-space-UNIV-axioms*)

end

$\mathbb{R} \times \mathbb{R}$ is a standard Borel space.

interpretation *real-real* : *pair-standard-borel-space-UNIV real-borel real-borel*
by(*auto intro*!: *pair-standard-borel-spaceUNIV simp*: *real.standard-borel-space-UNIV-axioms pair-standard-borel-space-UNIV-def*)

1.4 $\mathbb{N} \times \mathbb{R}$

$\mathbb{N} \times \mathbb{R}$ is a standard Borel space.

interpretation *nat-real*: *pair-standard-borel-space-UNIV nat-borel real-borel*
by(*auto intro*!: *pair-standard-borel-spaceUNIV*
simp: *real.standard-borel-space-UNIV-axioms nat.standard-borel-space-UNIV-axioms pair-standard-borel-space-UNIV-def*)

end

2 Quasi-Borel Spaces

theory *QuasiBorel*
imports *StandardBorel*
begin

2.1 Definitions

We formalize quasi-Borel spaces introduced by Heunen et al. [1].

2.1.1 Quasi-Borel Spaces

definition *qbs-closed1* :: $(real \Rightarrow 'a)\ set \Rightarrow bool$
where *qbs-closed1 Mx* $\equiv (\forall a \in Mx.\ \forall f \in real\text{-}borel \rightarrow_M real\text{-}borel.\ a \circ f \in Mx)$

definition *qbs-closed2* :: $['a\ set, (real \Rightarrow 'a)\ set] \Rightarrow bool$
where *qbs-closed2 X Mx* $\equiv (\forall x \in X.\ (\lambda r.\ x) \in Mx)$

definition *qbs-closed3* :: $(real \Rightarrow 'a)\ set \Rightarrow bool$
where *qbs-closed3 Mx* $\equiv (\forall P{::}real \Rightarrow nat.\ \forall Fi{::}nat \Rightarrow real \Rightarrow 'a.$
$(\forall i.\ P -` \{i\} \in sets\ real\text{-}borel)$
$\longrightarrow (\forall i.\ Fi\ i \in Mx)$
$\longrightarrow (\lambda r.\ Fi\ (P\ r)\ r) \in Mx)$

lemma *separate-measurable*:
fixes $P :: real \Rightarrow nat$
assumes $\bigwedge i.\ P -` \{i\} \in sets\ real\text{-}borel$
shows $P \in real\text{-}borel \rightarrow_M nat\text{-}borel$
proof −
have $P \in real\text{-}borel \rightarrow_M count\text{-}space\ UNIV$

by (*auto simp add*: *assms measurable-count-space-eq-countable*)
thus *?thesis*
using *measurable-cong-sets sets-borel-eq-count-space* **by** *blast*
qed

lemma *measurable-separate*:
fixes P :: *real* $\Rightarrow$ *nat*
assumes $P \in$ *real-borel* $\rightarrow_M$ *nat-borel*
shows P −‘ $\{i\} \in$ *sets real-borel*
by(*rule measurable-sets-borel*[*OF assms borel-singleton*[*OF sets.empty-sets*,*of i*]])

definition *is-quasi-borel X Mx* $\longleftrightarrow$ *Mx* $\subseteq$ *UNIV* $\rightarrow$ *X* $\wedge$ *qbs-closed1 Mx* $\wedge$ *qbs-closed2 X Mx* $\wedge$ *qbs-closed3 Mx*

lemma *is-quasi-borel-intro*[*simp*]:
assumes *Mx* $\subseteq$ *UNIV* $\rightarrow$ *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*) $\in$ $\{(X$::*′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* $\Rightarrow$ *′a set* **where**
qbs-space X $\equiv$ *fst* (*Rep-quasi-borel X*)

definition *qbs-Mx* :: *′a quasi-borel* $\Rightarrow$ (*real* $\Rightarrow$ *′a*) *set* **where**
qbs-Mx X $\equiv$ *snd* (*Rep-quasi-borel X*)

lemma *qbs-decomp* :
(*qbs-space X*,*qbs-Mx X*) $\in$ $\{(X$::*′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*[*dest*]:
assumes $\alpha \in$ *qbs-Mx X*
shows $\alpha \in$ *UNIV* $\rightarrow$ *qbs-space X*
α *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* $\Longrightarrow f \in$ *real-borel* $\rightarrow_M$ *real-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 $f \in$ *real-borel* $\rightarrow_M$ *real-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-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 $\bigwedge(P :: real \Rightarrow nat)\ Fi.\ (\bigwedge i.\ P -` \{i\} \in sets\ real\text{-}borel) \Longrightarrow (\bigwedge i.\ Fi\ i \in Mx)$
$\Longrightarrow (\lambda r.\ Fi\ (P\ r)\ r) \in Mx$
shows *qbs-closed3 Mx*
using *assms* **by**(*auto simp*: *qbs-closed3-def*)

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

lemma *qbs-closed3-dest*[*simp*]:
fixes $P::real \Rightarrow nat$ **and** $Fi :: nat \Rightarrow real \Rightarrow -$
assumes $\bigwedge i.\ P -` \{i\} \in sets\ real\text{-}borel$
and $\bigwedge i.\ Fi\ i \in$ *qbs-Mx X*
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in$ *qbs-Mx X*
using *assms qbs-decomp*[*of X*] **by** (*auto simp add*: *is-quasi-borel-def qbs-closed3-def*)

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

lemma *qbs-closed3-dest2*:
assumes *countable I*
and [*measurable*]: $P \in$ *real-borel* $\rightarrow_M$ *count-space I*
and $\bigwedge i.\ i \in I \Longrightarrow Fi\ i \in$ *qbs-Mx X*
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in$ *qbs-Mx X*
proof −

have *0*:$I \neq \{\}$
using *measurable-empty-iff*[*of count-space I P real-borel*] *assms*(*2*)
by *fastforce*
define P' **where** $P' \equiv$ *to-nat-on I* $\circ$ *P*
define Fi' **where** $Fi' \equiv Fi \circ$ (*from-nat-into I*)
have *1*:$P' \in$ *real-borel* $\rightarrow_M$ *nat-borel*
by(*simp add*: P'*-def*)
have *2*:$\bigwedge i.\ Fi'\ i \in$ *qbs-Mx X*
using *assms*(*3*) *from-nat-into*[*OF 0*] **by**(*simp add*: Fi'*-def*)
have $(\lambda r.\ Fi'\ (P'\ r)\ r) \in$ *qbs-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*]: $P \in$ *real-borel* $\rightarrow_M$ *count-space I*
and $\bigwedge i.\ i \in$ *range P* $\Longrightarrow$ *Fi i* $\in$ *qbs-Mx X*
shows $(\lambda r.\ Fi\ (P\ r)\ r) \in$ *qbs-Mx X*
proof −
have *0*:*range P* $\cap$ *I* = *range P*
using *measurable-space*[*OF assms*(*2*)] **by** *auto*
have *1*:$P \in$ *real-borel* $\rightarrow_M$ *count-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-space-Mx*:
qbs-space X = $\{\alpha\ x\ |x\ \alpha.\ \alpha \in$ *qbs-Mx X*$\}$
proof *auto*
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

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*])

2.1.2 Morphism of Quasi-Borel Spaces

definition *qbs-morphism* :: [*'a quasi-borel*, *'b quasi-borel*] ⇒ (*'a* ⇒ *'b*) *set* (**infixr** ‹$\to_Q$› *60*) **where**
$X \to_Q Y \equiv \{f \in$ *qbs-space* $X \to$ *qbs-space* Y. $\forall\,\alpha \in$ *qbs-Mx* X. $f \circ \alpha \in$ *qbs-Mx* $Y\}$

lemma *qbs-morphismI*:
assumes $\bigwedge\alpha.\ \alpha \in$ *qbs-Mx* $X \Longrightarrow f \circ \alpha \in$ *qbs-Mx* Y
shows $f \in X \to_Q Y$
proof −
have $f \in$ *qbs-space* $X \to$ *qbs-space* Y
proof
fix x
assume $x \in$ *qbs-space* X
then have $(\lambda r.\ x) \in$ *qbs-Mx* X
by *simp*
hence $f \circ (\lambda r.\ x) \in$ *qbs-Mx* Y
using *assms* **by** *blast*
thus $f\,x \in$ *qbs-space* Y
by *auto*
qed
thus *?thesis*
using *assms* **by**(*simp add*: *qbs-morphism-def*)
qed

lemma *qbs-morphismE*[*dest*]:
assumes $f \in X \to_Q Y$
shows $f \in$ *qbs-space* $X \to$ *qbs-space* Y
$\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\,x \in$ *qbs-space* Y
$\bigwedge\alpha.\ \alpha \in$ *qbs-Mx* $X \Longrightarrow f \circ \alpha \in$ *qbs-Mx* Y
using *assms* **by**(*auto simp add*: *qbs-morphism-def*)

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

lemma *qbs-morphism-ident'*[*simp*]:
$(\lambda x.\ x) \in X \to_Q X$
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-cong*:
assumes $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x = g\ x$
and $f \in X \rightarrow_Q Y$
shows $g \in X \rightarrow_Q Y$
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*] **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-const*:
assumes $y \in$ *qbs-space* Y
shows $(\lambda\text{-}.\ y) \in X \rightarrow_Q Y$
using *assms* **by** (*auto intro*: *qbs-morphismI*)

2.1.3 Empty Space

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

lemma *eqb-correct*: *Rep-quasi-borel empty-quasi-borel* = ({}, {})
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*)

lemma *eqb-space*[*simp*]: *qbs-space empty-quasi-borel* = {}
by(*simp add*: *qbs-space-def eqb-correct*)

lemma *eqb-Mx*[*simp*]: *qbs-Mx empty-quasi-borel* = {}
by(*simp add*: *qbs-Mx-def eqb-correct*)

lemma *qbs-empty-equiv* :*qbs-space* X = {} $\longleftrightarrow$ *qbs-Mx* X = {}
proof(*auto*)
fix x
assume *qbs-Mx* X = {}
and *h*:$x \in$ *qbs-space* X

have $(\lambda r.\ x) \in$ *qbs-Mx X*
using *h* **by** *simp*
thus *False* **using** ‹*qbs-Mx X* = {}› **by** *simp*
qed

lemma *empty-quasi-borel-iff*:
qbs-space X = {} $\longleftrightarrow$ *X* = *empty-quasi-borel*
by(*auto intro*!: *qbs-eqI*)

2.1.4 Unit Space

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

lemma *uqb-correct*: *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*)

lemma *uqb-space*[*simp*]: *qbs-space unit-quasi-borel* = {()}
by(*simp add*: *qbs-space-def UNIV-unit uqb-correct*)

lemma *uqb-Mx*[*simp*]: *qbs-Mx unit-quasi-borel* = $\{\lambda r.\ ()\}$
by(*auto simp add*: *qbs-Mx-def uqb-correct*)

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

definition *to-unit-quasi-borel* :: *'a* ⇒ *unit* (‹$!_Q$›) **where**
to-unit-quasi-borel ≡ (λ-.())

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

2.1.5 Subspaces

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

lemma *sub-qbs-closed*:
qbs-closed1 {*f* ∈ *UNIV* → *qbs-space X* ∩ *U*. *f* ∈ *qbs-Mx X*}
qbs-closed2 (*qbs-space X* ∩ *U*) {*f* ∈ *UNIV* → *qbs-space X* ∩ *U*. *f* ∈ *qbs-Mx X*}
qbs-closed3 {*f* ∈ *UNIV* → *qbs-space X* ∩ *U*. *f* ∈ *qbs-Mx X*}
unfolding *qbs-closed1-def qbs-closed2-def qbs-closed3-def* **by** *auto*

lemma *sub-qbs-correct*[*simp*]: *Rep-quasi-borel* (*sub-qbs X U*) = (*qbs-space X* ∩ *U*,{*f* ∈ *UNIV* → *qbs-space X* ∩ *U*. *f* ∈ *qbs-Mx X*})

by(*simp add*: *Abs-quasi-borel-inverse sub-qbs-def sub-qbs-closed*)

lemma *sub-qbs-space*[*simp*]: *qbs-space* (*sub-qbs X U*) = *qbs-space X* ∩ *U*
by(*simp add*: *qbs-space-def*)

lemma *sub-qbs-Mx*[*simp*]: *qbs-Mx* (*sub-qbs X U*) = {*f* ∈ *UNIV* → *qbs-space X* ∩ *U*. *f* ∈ *qbs-Mx X*}
by(*simp add*: *qbs-Mx-def*)

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*

2.1.6 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*),{*β*. ∃*α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*})

lemma *map-qbs-f*:
{*β*. ∃*α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*} ⊆ *UNIV* → *f* ‘ (*qbs-space X*)
by *fastforce*

lemma *map-qbs-closed1*:
qbs-closed1 {*β*. ∃*α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*}
unfolding *qbs-closed1-def*
using *qbs-closed1-dest* **by**(*fastforce simp*: *comp-def*)

lemma *map-qbs-closed2*:
qbs-closed2 (*f* ‘ (*qbs-space X*)) {*β*. ∃*α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*}
unfolding *qbs-closed2-def* **by** *fastforce*

lemma *map-qbs-closed3*:
qbs-closed3 {*β*. ∃*α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*}
proof(*auto simp add*: *qbs-closed3-def*)
fix *P Fi*
assume *h*:∀ *i*::*nat*. *P* −‘ {*i*} ∈ *sets real-borel*
∀ *i*::*nat*. ∃*α*∈*qbs-Mx X*. *Fi i* = *f* ∘ *α*
then obtain *αi*
where *ha*: ∀ *i*::*nat*. *αi i* ∈ *qbs-Mx X* ∧ *Fi i* = *f* ∘ (*αi i*)
by *metis*
hence *1*:(*λr*. *αi* (*P r*) *r*) ∈ *qbs-Mx X*
using *h*(*1*) **by** *fastforce*
show ∃*α*∈*qbs-Mx X*. (*λr*. *Fi* (*P r*) *r*) = *f* ∘ *α*
by(*auto intro*!: *bexI*[**where** *x*=(*λr*. *αi* (*P r*) *r*)] *simp add*: *1 ha comp-def*)
qed

lemma *map-qbs-correct*[*simp*]:

Rep-quasi-borel (*map-qbs f X*) = (*f* ‘ (*qbs-space X*),{*β*. ∃ *α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*})
unfolding *map-qbs-def*
by(*simp add*: *Abs-quasi-borel-inverse map-qbs-f map-qbs-closed1 map-qbs-closed2 map-qbs-closed3*)

lemma *map-qbs-space*[*simp*]:
qbs-space (*map-qbs f X*) = *f* ‘ (*qbs-space X*)
by(*simp add*: *qbs-space-def*)

lemma *map-qbs-Mx*[*simp*]:
qbs-Mx (*map-qbs f X*) = {*β*. ∃ *α*∈ *qbs-Mx X*. *β* = *f* ∘ *α*}
by(*simp add*: *qbs-Mx-def*)

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* ⟹ (*λr*. *x*) ∈ *generating-Mx X Mx*
| *Comp* : *f* ∈ *real-borel* $\rightarrow_M$ *real-borel* ⟹ *α* ∈ *generating-Mx X Mx* ⟹ *α* ∘ *f* ∈ *generating-Mx X Mx*
| *Part* : (⋀*i*. *Fi i* ∈ *generating-Mx X Mx*) ⟹ *P* ∈ *real-borel* $\rightarrow_M$ *nat-borel* ⟹ (*λr*. *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 *auto*
fix α
assume $\alpha \in$ *generating-Mx* (*qbs-space X*) (*qbs-Mx X*)
then show $\alpha \in$ *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

2.1.7 Ordering of Quasi-Borel Spaces

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*:
$X \leq Y \longleftrightarrow$ (*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 ⟷ ($X \leq Y \wedge \neg\, Y \leq X$)

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(*fastforce intro*!: *Abs-quasi-borel-inverse*
simp: *inf-quasi-borel-def is-quasi-borel-def qbs-closed1-def qbs-closed2-def qbs-closed3-def*)

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 auto
      apply(rule less-eq-quasi-borel.intros(2))
       apply(simp add: 1)
      by(rule less-eq-quasi-borel.cases[OF h(2)]) (auto simp: 1)
  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
       apply auto[1]
      by(simp add: less-eq-quasi-borel.intros(1))
  qed
qed
end

end
```

2.2 Relation to Measurable Spaces

```
theory Measure-QuasiBorel-Adjunction
  imports QuasiBorel
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.

2.2.1 The Functor R

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

lemma *R-Mx-correct*: *real-borel* $\to_M$ *X* ⊆ *UNIV* → *space X*
  by (*simp add*: *measurable-space subsetI*)

lemma *R-qbs-closed1*: *qbs-closed1* (*real-borel* $\to_M$ *X*)
  by (*simp add*: *qbs-closed1-def*)

lemma *R-qbs-closed2*: *qbs-closed2* (*space X*) (*real-borel* $\to_M$ *X*)
  by (*simp add*: *qbs-closed2-def*)

lemma *R-qbs-closed3*: *qbs-closed3* (*real-borel* $\to_M$ (*X* :: *'a measure*))
proof(*rule qbs-closed3I*)
  fix *P*::*real* ⇒ *nat*
  fix *Fi*::*nat* ⇒ *real* ⇒ *'a*
  assume *h*:⋀*i*. *P* −' {*i*} ∈ *sets real-borel*
          ⋀*i*. *Fi i* ∈ *real-borel* $\to_M$ *X*
  show (λ*r*. *Fi* (*P r*) *r*) ∈ *real-borel* $\to_M$ *X*
  proof(*rule measurableI*)
    fix *x*
    assume *x* ∈ *space real-borel*
    then show *Fi* (*P x*) *x* ∈ *space X*
      using *h*(*2*) *measurable-space*[*of Fi* (*P x*) *real-borel X x*]
      by *auto*
  next
    fix *A*
    assume *h'*:*A* ∈ *sets X*
    have (λ*r*. *Fi* (*P r*) *r*) −' *A* = (⋃*i*::*nat*. ((*Fi i* −' *A*) ∩ (*P* −' {*i*})))
      by *auto*
    also have ... ∈ *sets real-borel*
      using *sets.Int measurable-sets*[*OF h*(*2*) *h'*] *h*(*1*)
      by(*auto intro*!: *countable-Un-Int*(*1*))
    finally show (λ*r*. *Fi* (*P r*) *r*) −' *A* ∩ *space real-borel* ∈ *sets real-borel*
      by *simp*
  qed
qed

lemma *R-correct*[*simp*]: *Rep-quasi-borel* (*measure-to-qbs X*) = (*space X*, *real-borel* $\to_M$ *X*)
  unfolding *measure-to-qbs-def*
 by (*rule Abs-quasi-borel-inverse*) (*simp add*: *R-Mx-correct R-qbs-closed1 R-qbs-closed2 R-qbs-closed3*)

lemma *qbs-space-R*[*simp*]: *qbs-space* (*measure-to-qbs X*) = *space X*
  by (*simp add*: *qbs-space-def*)

lemma *qbs-Mx-R*[*simp*]: *qbs-Mx* (*measure-to-qbs X*) = *real-borel* $\rightarrow_M$ *X*
  by (*simp add*: *qbs-Mx-def*)

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

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

2.2.2 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 real-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 *auto*

  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 real-borel*)
        by (*simp add*: *sigma-Mx-def*)
      then obtain *Ua* **where** *pa*:*A* = *Ua* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −' *Ua* ∈ *sets real-borel*) **by** *auto*
    have ∃ *Ub*. *B* = *Ub* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −' *Ub* ∈ *sets real-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 real-borel*) **by** *auto*
from *pa pb* **have** [*simp*]:∀ α∈*qbs-Mx X*. α −' (*Ua* ∩ *Ub*) ∈ *sets real-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* ∩ *B* = (*Ua* ∩ *Ub*) ∩ *qbs-space X* **by** *auto*
thus *A* ∩ *B* ∈ *sigma-Mx X* **by** *simp*
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 real-borel*)
by (*simp add*: *sigma-Mx-def*)
then obtain *Ua* **where** *pa*:*A* = *Ua* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −' *Ua* ∈ *sets real-borel*) **by** *auto*
have ∃ *Ub*. *B* = *Ub* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −' *Ub* ∈ *sets real-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 real-borel*) **by** *auto*
from *pa pb* **have** [*simp*]:*A* − *B* = (*Ua* ∩ −*Ub*) ∩ *qbs-space X* **by** *auto*
from *pa pb* **have** ∀ α∈*qbs-Mx X*. α −'(*Ua* ∩ −*Ub*) ∈ *sets real-borel*
by (*metis Diff-Compl double-compl sets.Diff vimage-Compl vimage-Int*)
hence *1*:*A* − *B* ∈ *sigma-Mx X*
using *sigma-Mx-def* ‹*A* − *B* = *Ua* ∩ − *Ub* ∩ *qbs-space X*› **by** *blast*
show ∃ *C*⊆*sigma-Mx X*. *finite C* ∧ *disjoint C* ∧ *A* − *B* = ⋃ *C*
proof
show {*A* − *B*} ⊆*sigma-Mx X* ∧ *finite* {*A*−*B*} ∧ *disjoint* {*A*−*B*} ∧ *A* − *B* = ⋃ {*A*−*B*}
using *1* **by** *auto*
qed
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 real-borel*)
by (*simp add*: *sigma-Mx-def*)
then obtain *Ua* **where** *pa*:*A* = *Ua* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −' *Ua* ∈ *sets real-borel*) **by** *auto*
have ∃ *Ub*. *B* = *Ub* ∩ *qbs-space X* ∧ (∀ α∈*qbs-Mx X*. α −' *Ub* ∈ *sets real-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 real-borel*) **by** *auto*
from *pa pb* **have** *A* ∪ *B* = (*Ua* ∪ *Ub*) ∩ *qbs-space X* **by** *auto*
from *pa pb* **have** ∀ α∈*qbs-Mx X*. α −'(*Ua* ∪ *Ub*) ∈ *sets real-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*. α −' (*UNIV*) $\in$ *sets real-borel*
by *simp*
thus *qbs-space X* $\in$ *sigma-Mx X*
unfolding *sigma-Mx-def*
by *blast*
qed
next
show *sigma-algebra-axioms* (*sigma-Mx X*)
unfolding *sigma-algebra-axioms-def*
proof(*auto*)
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 real-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 real-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 real-borel*)
by *auto*
hence ∀ α∈*qbs-Mx X*. α −' (⋃ (*range U*)) ∈ *sets real-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 real-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 $\bigwedge i.\ A\ i = \{\}$
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-Mx X then 0 else* ∞) = (∞ :: *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-Mx X then 0 else* ∞) = (∞ :: *ennreal*)
proof(*rule hsum'*)
show $\exists ja.\ ja \in \{..<j + 1\} \wedge$ (*if* $A\ ja = \{\}$ *then 0 else if* $A\ ja \in -$ *sigma-Mx X then 0 else* ∞) = (∞ :: *ennreal*)
proof(*rule exI*[**where** *x*=*j*],*rule conjI*)
have $A\ j \in$ *sigma-Mx X*
using *h*(*1*) **by** *auto*
then show (*if* $A\ j = \{\}$ *then 0 else if* $A\ j \in -$ *sigma-Mx X then 0 else* ∞) = (∞ :: *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-Mx X then 0 else* ∞) $\leq$ ($\sum i.$ *if* $A\ i = \{\}$ *then 0 else if* $A\ i \in -$ *sigma-Mx X then 0 else* (∞ :: *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-Mx X then 0 else* ∞) = (∞ :: *ennreal*)
using *2 h*(*2*) **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed
qed(*simp add*: *positive-def*)

lemma *L-correct*[*simp*]:*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*)

lemma *space-L*[*simp*]: *space* (*qbs-to-measure X*) = *qbs-space X*
by (*simp add*: *space-def*)

lemma *sets-L*[*simp*]: *sets* (*qbs-to-measure X*) = *sigma-Mx X*
by (*simp add*: *sets-def*)

lemma *emeasure-L*[*simp*]: *emeasure* (*qbs-to-measure X*) = (λA. *if* $A = \{\} \vee A \notin$ *sigma-Mx X then 0 else* ∞)
by(*auto simp*: *emeasure-def*)

lemma *qbs-Mx-sigma-Mx-contra*:
assumes *qbs-space X* = *qbs-space Y*
and *qbs-Mx X* $\subseteq$ *qbs-Mx Y*
shows *sigma-Mx Y* $\subseteq$ *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*:
$X \to_Q Y \subseteq$ (*qbs-to-measure X*) $\to_M$ (*qbs-to-measure Y*)
proof(*auto*)
fix *f*
assume *h*:$f \in X \to_Q Y$
show $f \in$ (*qbs-to-measure X*) $\to_M$ (*qbs-to-measure Y*)
proof(*rule measurableI*)
fix *x*
assume $x \in$ *space* (*qbs-to-measure X*)
then show *f x* $\in$ *space* (*qbs-to-measure Y*)
using *h* **by** *auto*
next
fix *A*
assume $A \in$ *sets* (*qbs-to-measure Y*)
then obtain *Ua* **where** *pa*:*A* = *Ua* $\cap$ *qbs-space Y* $\wedge$ ($\forall \alpha \in$*qbs-Mx Y*. α -‘ *Ua* $\in$ *sets real-borel*)
by (*auto simp*: *sigma-Mx-def*)
have $\forall \alpha \in$*qbs-Mx X*. $f \circ \alpha \in$ *qbs-Mx Y*
$\forall \alpha \in$ *qbs-Mx X*. α -‘ (*f* -‘ (*qbs-space Y*)) = *UNIV*
using *h* **by** *auto*
hence $\forall \alpha \in$*qbs-Mx X*. α -‘ (*f* -‘ *A*) = α -‘ (*f* -‘ *Ua*)
by (*simp add*: *pa*)
from *pa this qbs-morphism-def* **have** $\forall \alpha \in$*qbs-Mx X*. α -‘ (*f* -‘ *A*) $\in$ *sets real-borel*
by (*simp add*: *vimage-comp* ‹$\forall \alpha \in$*qbs-Mx X*. $f \circ \alpha \in$ *qbs-Mx Y*›)
thus *f* -‘ *A* $\cap$ *space* (*qbs-to-measure X*) $\in$ *sets* (*qbs-to-measure X*)
using *sigma-Mx-def* **by** *auto*
qed
qed

abbreviation *qbs-borel* ≡ *measure-to-qbs borel*

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

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-borel*
abbreviation *ennreal-quasi-borel* :: *ennreal quasi-borel* (‹$\mathbb{R}_{Q\geq 0}$›) **where**
ennreal-quasi-borel ≡ *qbs-borel*
abbreviation *bool-quasi-borel* :: *bool quasi-borel* (‹$\mathbb{B}_Q$›) **where**
bool-quasi-borel ≡ *qbs-borel*

lemma *qbs-Mx-is-morphisms*:
qbs-Mx X = *real-quasi-borel* $\rightarrow_Q$ *X*
proof(*auto*)
  fix *α*
  assume *α* ∈ *qbs-Mx X*
  then have *α* ∈ *UNIV* → *qbs-space X* ∧ (∀ *f* ∈ *real-borel* $\rightarrow_M$ *real-borel*. *α* ∘ *f* ∈ *qbs-Mx X*)
    by *fastforce*
  thus *α* ∈ *real-quasi-borel* $\rightarrow_Q$ *X*
    by(*simp add*: *qbs-morphism-def*)
next
  fix *α*
  assume *α* ∈ *real-quasi-borel* $\rightarrow_Q$ *X*
  have *id* ∈ *qbs-Mx real-quasi-borel* **by** *simp*
  then have *α* ∘ *id* ∈ *qbs-Mx X*
    using ‹*α* ∈ *real-quasi-borel* $\rightarrow_Q$ *X*› *qbs-morphism-def*[*of real-quasi-borel X*]
    by *blast*
  then show *α* ∈ *qbs-Mx X* **by** *simp*
qed

lemma *qbs-Mx-subset-of-measurable*:
  qbs-Mx X ⊆ *real-borel* $\rightarrow_M$ *qbs-to-measure X*
proof
  fix *α*
  assume *α* ∈ *qbs-Mx X*
  show *α* ∈ *real-borel* $\rightarrow_M$ *qbs-to-measure X*
  proof(*rule measurableI*)
    fix *x*
    show *α x* ∈ *space* (*qbs-to-measure X*)
      using *qbs-decomp* ‹*α* ∈ *qbs-Mx X*› **by** *auto*
  next
    fix *A*
    assume *A* ∈ *sets* (*qbs-to-measure X*)

```
    then have α −‘(qbs-space X) = UNIV
      using ‹α ∈ qbs-Mx X› qbs-decomp by auto
    then show α −‘ A ∩ space real-borel ∈ sets real-borel
      using ‹α ∈ qbs-Mx X› ‹A ∈ sets (qbs-to-measure X)›
      by(auto simp add: sigma-Mx-def)
  qed
qed

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


lemma qbs-Mx-are-measurable[simp,measurable]:
  assumes α ∈ qbs-Mx X
  shows α ∈ real-borel →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: measurable-cong-sets[OF - assms])

lemma lr-sets[simp,measurable-cong]:
 sets X ⊆ sets (qbs-to-measure (measure-to-qbs X))
proof auto
  fix U
  assume U ∈ sets X
  then have U ∩ space X = U by simp
  moreover have ∀α∈real-borel →M X. α −‘ U ∈ sets real-borel
    using ‹U ∈ sets X› by(auto simp add: measurable-def)
  ultimately show U ∈ sigma-Mx (measure-to-qbs X)
    by(auto simp add: sigma-Mx-def)
qed

lemma(in standard-borel) standard-borel-lr-sets-ident[simp, measurable-cong]:
 sets (qbs-to-measure (measure-to-qbs M)) = sets M
proof auto
  fix V
```

assume *V* ∈ *sigma-Mx* (*measure-to-qbs M*)
then obtain *U* **where** *H2*: *V* = *U* ∩ *space M* ∧ (∀ *α*∈*real-borel* $\rightarrow_M$ *M*. *α* −‘ *U* ∈ *sets real-borel*)
by(*auto simp*: *sigma-Mx-def*)
hence *g* −‘ *V* = *g* −‘ (*U* ∩ *space M*) **by** *auto*
have ... = *g* −‘ *U* **using** *g-meas* **by**(*auto simp add*: *measurable-def*)
hence *f* −‘ *g* −‘ *U* ∩ *space M* ∈ *sets M*
by (*meson f-meas g-meas measurable-sets H2*)
moreover have *f* −‘ *g* −‘ *U* ∩ *space M* = *U* ∩ *space M*
by *auto*
ultimately show *V* ∈ *sets M* **using** *H2* **by** *simp*
next
fix *U*
assume *U* ∈ *sets M*
then show *U* ∈ *sigma-Mx* (*measure-to-qbs M*)
using *lr-sets* **by** *auto*
qed

2.2.3 The Adjunction

lemma *lr-adjunction-correspondence* :
X $\rightarrow_Q$ (*measure-to-qbs Y*) = (*qbs-to-measure X*) $\rightarrow_M$ *Y*
proof(*auto*)

fix *f*
assume *f* ∈ *X* $\rightarrow_Q$ (*measure-to-qbs Y*)
show *f* ∈ *qbs-to-measure X* $\rightarrow_M$ *Y*
proof(*rule measurableI*)
fix *x*
assume *x* ∈ *space* (*qbs-to-measure X*)
hence *x* ∈ *qbs-space X* **by** *simp*
thus *f x* ∈ *space Y*
using ‹*f* ∈ *X* $\rightarrow_Q$ (*measure-to-qbs Y*)› *qbs-morphismE*[*of f X measure-to-qbs Y*]
by *auto*
next
fix *A*
assume *A* ∈ *sets Y*
have ∀ *α* ∈ *qbs-Mx X*. *f* ∘ *α* ∈ *qbs-Mx* (*measure-to-qbs Y*)
using ‹*f* ∈ *X* $\rightarrow_Q$ (*measure-to-qbs Y*)› **by** *auto*
hence ∀ *α* ∈ *qbs-Mx X*. *f* ∘ *α* ∈ *real-borel* $\rightarrow_M$ *Y* **by** *simp*
hence ∀ *α* ∈ *qbs-Mx X*. *α* −‘ (*f* −‘ *A*) ∈ *sets real-borel*
using ‹*A*∈ *sets Y*› *measurable-sets-borel vimage-comp* **by** *metis*
thus *f* −‘ *A* ∩ *space* (*qbs-to-measure X*) ∈ *sets* (*qbs-to-measure X*)
using *sigma-Mx-def* **by** *auto*
qed

next

fix *f*
assume *f* ∈ *qbs-to-measure X* $\rightarrow_M$ *Y*
show *f* ∈ *X* $\rightarrow_Q$ *measure-to-qbs Y*
proof(*rule qbs-morphismI*,*simp*)
fix α
assume α ∈ *qbs-Mx X*
show *f* ∘ α ∈ *real-borel* $\rightarrow_M$ *Y*
proof(*rule measurableI*)
fix *x*
assume *x* ∈ *space real-borel*
from *this* ‹α ∈ *qbs-Mx X* ›*qbs-decomp* **have** α *x* ∈ *qbs-space X* **by** *auto*
hence α *x* ∈ *space* (*qbs-to-measure X*) **by** *simp*
thus (*f* ∘ α) *x* ∈ *space Y*
using ‹*f* ∈ *qbs-to-measure X* $\rightarrow_M$ *Y*›
by (*metis comp-def measurable-space*)
next
fix *A*
assume *A* ∈ *sets Y*
from ‹*f* ∈ *qbs-to-measure X* $\rightarrow_M$ *Y*› *measurable-sets this measurable-def*
have *f* −‘ *A* ∩ *space* (*qbs-to-measure X*) ∈ *sets* (*qbs-to-measure X*)
by *blast*
hence *f* −‘ *A* ∩ *qbs-space X* ∈ *sigma-Mx X* **by** *simp*
then have ∃ *V*. *f* −‘ *A* ∩ *qbs-space X* = *V* ∩ *qbs-space X* ∧ (∀β∈ *qbs-Mx X*. β −‘ *V* ∈ *sets real-borel*)
by (*simp add*:*sigma-Mx-def*)
then obtain *V* **where** *h*:*f* −‘ *A* ∩ *qbs-space X* = *V* ∩ *qbs-space X* ∧ (∀β∈ *qbs-Mx X*. β −‘ *V* ∈ *sets real-borel*) **by** *auto*
have *1*:α −‘ (*f* −‘ *A*) = α −‘ (*f* −‘ *A* ∩ *qbs-space X*)
using ‹α ∈ *qbs-Mx X*› **by** *blast*
have *2*:α −‘ (*V* ∩ *qbs-space X*) = α −‘ *V*
using ‹α ∈ *qbs-Mx X*› **by** *blast*
from *1 2 h* **have** (*f* ∘ α) −‘ *A* = α −‘ *V* **by** (*simp add*: *vimage-comp*)
from *this h* ‹α ∈ *qbs-Mx X* ›**show** (*f* ∘ α) −‘ *A* ∩ *space real-borel* ∈ *sets real-borel* **by** *simp*
qed
qed
qed

lemma(**in** *standard-borel*) *standard-borel-r-full-faithful*:
M $\rightarrow_M$ *Y* = *measure-to-qbs M* $\rightarrow_Q$ *measure-to-qbs Y*
proof(*standard*;*standard*)
fix *h*
assume *h* ∈ *M* $\rightarrow_M$ *Y*
then show *h* ∈ *measure-to-qbs M* $\rightarrow_Q$ *measure-to-qbs Y*
using *r-preserves-morphisms* **by** *auto*
next
fix *h*
assume *h*:*h* ∈ *measure-to-qbs M* $\rightarrow_Q$ *measure-to-qbs Y*
show *h* ∈ *M* $\rightarrow_M$ *Y*

```
  proof(rule measurableI)
    fix x
    assume x ∈ space M
    then show h x ∈ space Y
      using h by auto
  next
    fix U
    assume U ∈ sets Y
    have h ∘ g ∈ real-borel →M Y
      using ‹h ∈ measure-to-qbs M →Q measure-to-qbs Y›
      by(simp add: qbs-morphism-def)
    hence (h ∘ g) −‘ U ∈ sets real-borel
      by (simp add: ‹U ∈ sets Y› measurable-sets-borel)
    hence f −‘ ((h ∘ g) −‘ U) ∩ space M ∈ sets M
      using f-meas in-borel-measurable-borel by blast
    moreover have f −‘ ((h ∘ g) −‘ U) ∩ space M = h −‘ U ∩ space M
      using f-meas by auto
    ultimately show h −‘ U ∩ space M ∈ sets M by simp
  qed
qed

lemma qbs-morphism-dest[dest]:
  assumes f ∈ X →Q measure-to-qbs Y
  shows f ∈ qbs-to-measure X →M Y
  using assms lr-adjunction-correspondence by auto

lemma(in standard-borel) qbs-morphism-dest[dest]:
  assumes k ∈ measure-to-qbs M →Q measure-to-qbs Y
  shows k ∈ M →M Y
  using standard-borel-r-full-faithful assms by auto

lemma qbs-morphism-measurable-intro[intro!]:
  assumes f ∈ qbs-to-measure X →M Y
  shows f ∈ X →Q measure-to-qbs Y
  using assms lr-adjunction-correspondence by auto

lemma(in standard-borel) qbs-morphism-measurable-intro[intro!]:
  assumes k ∈ M →M Y
  shows k ∈ measure-to-qbs M →Q measure-to-qbs Y
  using standard-borel-r-full-faithful assms by auto
```

We can use the measurability prover when we reason about morphisms.

```
lemma
  assumes f ∈ ℝQ →Q ℝQ
  shows (λx. 2 ∗ f x + (f x)^2) ∈ ℝQ →Q ℝQ
  using assms by auto

lemma
  assumes f ∈ X →Q ℝQ
```

and $\alpha \in$ *qbs-Mx X*
shows $(\lambda x.\ 2 * f\ (\alpha\ x) + (f\ (\alpha\ x))\hat{}2) \in \mathbb{R}_Q \to_Q \mathbb{R}_Q$
using *assms* **by** *auto*

lemma *qbs-morphisn-from-countable*:
fixes X :: *'a quasi-borel*
assumes *countable* (*qbs-space X*)
qbs-Mx X $\subseteq$ *real-borel* $\to_M$ *count-space* (*qbs-space X*)
and $\bigwedge i.\ i \in$ *qbs-space X* $\Longrightarrow f\ i \in$ *qbs-space Y*
shows $f \in X \to_Q Y$
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx X*
then have [*measurable*]: $\alpha \in$ *real-borel* $\to_M$ *count-space* (*qbs-space X*)
using *assms*(*2*) **..**
define k :: *'a* $\Rightarrow$ *real* $\Rightarrow$ -
where $k \equiv (\lambda i$ -. $f\ i)$
have $f \circ \alpha = (\lambda r.\ k\ (\alpha\ r)\ r)$
by(*auto simp add*: *k-def*)
also have ... $\in$ *qbs-Mx Y*
by(*rule qbs-closed3-dest2*[*OF assms*(*1*)]) (*use assms*(*3*) *k-def* **in** *simp-all*)
finally show $f \circ \alpha \in$ *qbs-Mx Y* **.**
qed

corollary *nat-qbs-morphism*:
assumes $\bigwedge n.\ f\ n \in$ *qbs-space Y*
shows $f \in \mathbb{N}_Q \to_Q Y$
using *assms measurable-cong-sets*[*OF refl sets-borel-eq-count-space*,*of real-borel*]
by(*auto intro*!: *qbs-morphisn-from-countable*)

corollary *bool-qbs-morphism*:
assumes $\bigwedge b.\ f\ b \in$ *qbs-space Y*
shows $f \in \mathbb{B}_Q \to_Q Y$
using *assms measurable-cong-sets*[*OF refl sets-borel-eq-count-space*,*of real-borel*]
by(*auto intro*!: *qbs-morphisn-from-countable*)

2.2.4 The Adjunction w.r.t. Ordering

lemma *l-mono*:
mono qbs-to-measure
apply *standard*
subgoal for *X Y*
proof(*induction rule*: *less-eq-quasi-borel.induct*)
case (*1 X Y*)
then show *?case*
by(*simp add*: *less-eq-measure.intros*(*1*))
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*)
by(*rule less-eq-measure.intros*(*3*),*simp-all add*: *2*)
qed
done

lemma *r-mono*:
mono measure-to-qbs
apply *standard*
subgoal for *M N*
proof(*induction rule*: *less-eq-measure.inducts*)
case (*1 M N*)
then show *?case*
by(*simp add*: *less-eq-quasi-borel.intros*(*1*))
next
case (*2 M N*)
then have *real-borel* $\rightarrow_M$ *N* ⊆ *real-borel* $\rightarrow_M$ *M*
by(*simp add*: *measurable-mono*)
then consider *real-borel* $\rightarrow_M$ *N* ⊂ *real-borel* $\rightarrow_M$ *M* | *real-borel* $\rightarrow_M$ *N* = *real-borel* $\rightarrow_M$ *M*
by *auto*
then show *?case*
by *cases* (*rule less-eq-quasi-borel.intros*(*2*),*simp-all add*: *2*)+
next
case (*3 M N*)
then show *?case*
apply −
by(*rule less-eq-quasi-borel.intros*(*2*)) (*simp-all add*: *measurable-mono*)
qed
done

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*])
show *?case*
by(*rule less-eq-quasi-borel.intros*(*1*),*simp add*: *1*)
next

```
  case (2 M N)
  then have [simp]:qbs-space Y = space N
    by(simp add: 2(2)[symmetric])
  show ?case
  proof(rule less-eq-quasi-borel.intros(2),simp add:2,auto)
    fix α
    assume h:α ∈ qbs-Mx Y
    show α ∈ real-borel →M X
    proof(rule measurableI,simp-all)
      show ⋀x. α x ∈ space X
        using 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 real-borel)
        by(auto simp add: sigma-Mx-def)
      have α -` A = α -` U
        using h qbs-decomp[of Y]
        by(auto simp add: hu)
      thus α -` A ∈ sets borel
        using h hu(2) by simp
    qed
  qed
next
  case (3 M N)
  then have [simp]:qbs-space Y = space N
    by(simp add: 3(2)[symmetric])
  show ?case
  proof(rule less-eq-quasi-borel.intros(2),simp add: 3,auto)
    fix α
    assume h:α ∈ qbs-Mx Y
    show α ∈ real-borel →M X
    proof(rule measurableI,simp-all)
      show ⋀x. α x ∈ space X
        using 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 real-borel)
        by(auto simp add: sigma-Mx-def)
      have α -` A = α -` U
```

```
          using h qbs-decomp[of Y]
          by(auto simp add: hu)
        thus α −‘ A ∈ sets borel
          using h hu(2) by simp
      qed
    qed
  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])
    show ?case
      by(rule less-eq-measure.intros(1)) (simp add: 1)
  next
    case (2 A B)
    then have hmy:qbs-Mx Y ⊆ real-borel →M X
      by auto
    have [simp]: space X = qbs-space A
      by(simp add: 2(1)[symmetric])
    have sets X ⊆ sigma-Mx Y
    proof
      fix U
      assume hu:U ∈ sets X
      show U ∈ sigma-Mx Y
      proof(simp add: sigma-Mx-def,rule exI[where x=U],auto)
        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 ∈ real-borel →M X
          using hmy by(auto)
        thus α −‘ U ∈ sets real-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
        apply(rule less-eq-measure.intros(3),simp-all add: 1 2)
      proof(rule le-funI)
        fix U
```

```
        consider U = {} | U ∉ sigma-Mx B | U ≠ {} ∧ U ∈ sigma-Mx B
          by auto
        then show emeasure X U ≤ (if U = {} ∨ U ∉ sigma-Mx B 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)]
            by auto
          hence U ∉ sets X
            using lr-sets 2(1) by auto
          thus ?thesis
            by(simp add: h emeasure-notin-sets)
        next
          case 3
          then show ?thesis
            by simp
        qed
      qed
    next
      case h2:2
      show ?thesis
        by(rule less-eq-measure.intros(2)) (simp add: 2,simp add: h2)
    qed
  qed
qed

end
```

2.3 Product Spaces

```
theory Binary-Product-QuasiBorel
  imports Measure-QuasiBorel-Adjunction
begin
```

2.3.1 Binary Product Spaces

```
definition pair-qbs-Mx :: ['a quasi-borel, 'b quasi-borel] ⇒ (real => 'a × 'b) set
where
pair-qbs-Mx X Y ≡ {f. fst ∘ f ∈ qbs-Mx X ∧ snd ∘ f ∈ qbs-Mx Y}

definition pair-qbs :: ['a quasi-borel, 'b quasi-borel] ⇒ ('a × 'b) quasi-borel (infixr
‹⨂Q› 80) where
pair-qbs X Y = Abs-quasi-borel (qbs-space X × qbs-space Y, pair-qbs-Mx X Y)


lemma pair-qbs-f[simp]: pair-qbs-Mx X Y ⊆ UNIV → qbs-space X × qbs-space Y
  unfolding pair-qbs-Mx-def
```

by (*auto simp*: *mem-Times-iff*[*of* - *qbs-space X qbs-space Y*]; *fastforce*)

lemma *pair-qbs-closed1*: *qbs-closed1* (*pair-qbs-Mx* (*X*::′*a quasi-borel*) (*Y*::′*b quasi-borel*))
unfolding *pair-qbs-Mx-def qbs-closed1-def*
by (*metis* (*no-types*, *lifting*) *comp-assoc mem-Collect-eq qbs-closed1-dest*)

lemma *pair-qbs-closed2*: *qbs-closed2* (*qbs-space X* × *qbs-space Y*) (*pair-qbs-Mx X Y*)
unfolding *qbs-closed2-def pair-qbs-Mx-def*
by *auto*

lemma *pair-qbs-closed3*: *qbs-closed3* (*pair-qbs-Mx* (*X*::′*a quasi-borel*) (*Y*::′*b quasi-borel*))
proof(*auto simp add*: *qbs-closed3-def pair-qbs-Mx-def*)
fix *P* :: *real* ⇒ *nat*
fix *Fi* :: *nat* ⇒ *real* ⇒ ′*a* × ′*b*
define *Fj* :: *nat* ⇒ *real* ⇒ ′*a* **where** *Fj* ≡ λ*j*.(*fst* ∘ *Fi j*)
assume ∀ *i*. *fst* ∘ *Fi i* ∈ *qbs-Mx X* ∧ *snd* ∘ *Fi i* ∈ *qbs-Mx Y*
then have ∀ *i*. *Fj i* ∈ *qbs-Mx X* **by** (*simp add*: *Fj-def*)
moreover assume ∀ *i*. *P* −‘ {*i*} ∈ *sets real-borel*
ultimately have (λ*r*. *Fj* (*P r*) *r*) ∈ *qbs-Mx X*
by *auto*
moreover have *fst* ∘ (λ*r*. *Fi* (*P r*) *r*) = (λ*r*. *Fj* (*P r*) *r*) **by** (*auto simp add*: *Fj-def*)
ultimately show *fst* ∘ (λ*r*. *Fi* (*P r*) *r*) ∈ *qbs-Mx X* **by** *simp*
next
fix *P* :: *real* ⇒ *nat*
fix *Fi* :: *nat* ⇒ *real* ⇒ ′*a* × ′*b*
define *Fj* :: *nat* ⇒ *real* ⇒ ′*b* **where** *Fj* ≡ λ*j*.(*snd* ∘ *Fi j*)
assume ∀ *i*. *fst* ∘ *Fi i* ∈ *qbs-Mx X* ∧ *snd* ∘ *Fi i* ∈ *qbs-Mx Y*
then have ∀ *i*. *Fj i* ∈ *qbs-Mx Y* **by** (*simp add*: *Fj-def*)
moreover assume ∀ *i*. *P* −‘ {*i*} ∈ *sets real-borel*
ultimately have (λ*r*. *Fj* (*P r*) *r*) ∈ *qbs-Mx Y*
by *auto*
moreover have *snd* ∘ (λ*r*. *Fi* (*P r*) *r*) = (λ*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

lemma *pair-qbs-correct*: *Rep-quasi-borel* ($X \bigotimes_Q Y$) = (*qbs-space X* × *qbs-space Y*, *pair-qbs-Mx X Y*)
unfolding *pair-qbs-def*
by(*auto intro*!: *Abs-quasi-borel-inverse pair-qbs-f simp*: *pair-qbs-closed3 pair-qbs-closed2 pair-qbs-closed1*)

lemma *pair-qbs-space*[*simp*]: *qbs-space* ($X \bigotimes_Q Y$) = *qbs-space X* × *qbs-space Y*
by (*simp add*: *qbs-space-def pair-qbs-correct*)

lemma *pair-qbs-Mx*[*simp*]: *qbs-Mx* ($X \bigotimes_Q Y$) = *pair-qbs-Mx X Y*
by (*simp add*: *qbs-Mx-def pair-qbs-correct*)

lemma *pair-qbs-morphismI*:
assumes $\bigwedge\alpha\ \beta.\ \alpha \in$ *qbs-Mx* $X \Longrightarrow \beta \in$ *qbs-Mx* Y
$\Longrightarrow f \circ (\lambda r.\ (\alpha\ r,\ \beta\ r)) \in$ *qbs-Mx* Z
shows $f \in (X \bigotimes_Q Y) \to_Q Z$
proof(*rule qbs-morphismI*)
fix α
assume *1*:$\alpha \in$ *qbs-Mx* $(X \bigotimes_Q Y)$
have $f \circ \alpha = f \circ (\lambda r.\ ((fst \circ \alpha)\ r,\ (snd \circ \alpha)\ r))$
by *auto*
also have ... $\in$ *qbs-Mx* Z
using *1 assms*[*of fst* $\circ$ α *snd* $\circ$ α]
by(*simp add*: *pair-qbs-Mx-def*)
finally show $f \circ \alpha \in$ *qbs-Mx* Z **.**
qed

lemma *fst-qbs-morphism*:
fst $\in X \bigotimes_Q Y \to_Q X$
by(*auto simp add*: *qbs-morphism-def pair-qbs-Mx-def*)

lemma *snd-qbs-morphism*:
snd $\in X \bigotimes_Q Y \to_Q Y$
by(*auto simp add*: *qbs-morphism-def pair-qbs-Mx-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-morphismI qbs-morphism-comp*[*OF - fst-qbs-morphism*,*of f X Y Z*]*qbs-morphism-comp*[*OF - snd-qbs-morphism*,*of f X Y Z*]
simp: *pair-qbs-Mx-def comp-assoc*[*symmetric*])

lemma *qbs-morphism-Pair1*:
assumes $x \in$ *qbs-space* X
shows *Pair* $x \in Y \to_Q X \bigotimes_Q Y$
using *assms*
by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx-def comp-def*)

lemma *qbs-morphism-Pair1'*:
assumes $x \in$ *qbs-space* X
and $f \in X \bigotimes_Q Y \to_Q Z$
shows $(\lambda y.\ f\ (x,y)) \in Y \to_Q Z$
using *qbs-morphism-comp*[*OF qbs-morphism-Pair1* [*OF assms*(*1*)] *assms*(*2*)]
by(*simp add*: *comp-def*)

lemma *qbs-morphism-Pair2*:
assumes $y \in$ *qbs-space* Y
shows $(\lambda x.\ (x,y)) \in X \to_Q X \bigotimes_Q Y$
using *assms*

by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx-def comp-def*)

lemma *qbs-morphism-Pair2'*:
assumes $y \in$ *qbs-space* Y
and $f \in X \bigotimes_Q Y \to_Q Z$
shows $(\lambda x.\ f\ (x,y)) \in X \to_Q Z$
using *qbs-morphism-comp*[*OF qbs-morphism-Pair2*[*OF assms*(*1*)] *assms*(*2*)]
by(*simp add*: *comp-def*)

lemma *qbs-morphism-fst''*:
assumes $f \in X \to_Q Y$
shows $(\lambda k.\ f\ (fst\ k)) \in X \bigotimes_Q Z \to_Q Y$
using *qbs-morphism-comp*[*OF fst-qbs-morphism assms*,*of Z*]
by(*simp add*: *comp-def*)

lemma *qbs-morphism-snd''*:
assumes $f \in X \to_Q Y$
shows $(\lambda k.\ f\ (snd\ k)) \in Z \bigotimes_Q X \to_Q Y$
using *qbs-morphism-comp*[*OF snd-qbs-morphism assms*,*of Z*]
by(*simp add*: *comp-def*)

lemma *qbs-morphism-tuple*:
assumes $f \in Z \to_Q X$
and $g \in Z \to_Q Y$
shows $(\lambda z.\ (f\ z,\ g\ z)) \in Z \to_Q X \bigotimes_Q Y$
proof(*rule qbs-morphismI*,*simp*)
fix α
assume $h{:}\alpha \in$ *qbs-Mx* Z
then have $(\lambda z.\ (f\ z,\ g\ z)) \circ \alpha \in UNIV \to$ *qbs-space* $X \times$ *qbs-space* Y
using *assms qbs-morphismE*(*2*)[*OF assms*(*1*)] *qbs-morphismE*(*2*)[*OF assms*(*2*)]
by *fastforce*
moreover have $fst \circ ((\lambda z.\ (f\ z,\ g\ z)) \circ \alpha) = f \circ \alpha$ **by** *auto*
moreover have ... $\in$ *qbs-Mx* X
using *assms*(*1*) *h* **by** *auto*
moreover have $snd \circ ((\lambda z.\ (f\ z,\ g\ z)) \circ \alpha) = g \circ \alpha$ **by** *auto*
moreover have ... $\in$ *qbs-Mx* Y
using *assms*(*2*) *h* **by** *auto*
ultimately show $(\lambda z.\ (f\ z,\ g\ z)) \circ \alpha \in$ *pair-qbs-Mx* X Y
by (*simp add*: *pair-qbs-Mx-def*)
qed

lemma *qbs-morphism-map-prod*:
assumes $f \in X \to_Q Y$
and $g \in X' \to_Q Y'$
shows *map-prod f g* $\in X \bigotimes_Q X' \to_Q Y \bigotimes_Q Y'$
proof(*rule pair-qbs-morphismI*)
fix α β
assume $h{:}\alpha \in$ *qbs-Mx* X
$\beta \in$ *qbs-Mx* X'

have [*simp*]: *fst* ∘ (*map-prod f g* ∘ (λ*r*. (α *r*, β *r*))) = *f* ∘ α **by** *auto*
have [*simp*]: *snd* ∘ (*map-prod f g* ∘ (λ*r*. (α *r*, β *r*))) = *g* ∘ β **by** *auto*
show *map-prod f g* ∘ (λ*r*. (α *r*, β *r*)) ∈ *qbs-Mx* (*Y* ⨂$_Q$ *Y′*)
using *h assms* **by**(*auto simp*: *pair-qbs-Mx-def*)
qed

lemma *qbs-morphism-pair-swap′*:
(λ(*x*,*y*). (*y*,*x*)) ∈ (*X*::′*a quasi-borel*) ⨂$_Q$ (*Y*::′*b quasi-borel*) →$_Q$ *Y* ⨂$_Q$ *X*
by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx-def split-beta′ comp-def*)

lemma *qbs-morphism-pair-swap*:
assumes *f* ∈ *X* ⨂$_Q$ *Y* →$_Q$ *Z*
shows (λ(*x*,*y*). *f* (*y*,*x*)) ∈ *Y* ⨂$_Q$ *X* →$_Q$ *Z*
proof −
have (λ(*x*,*y*). *f* (*y*,*x*)) = *f* ∘ (λ(*x*,*y*). (*y*,*x*)) **by** *auto*
thus *?thesis*
using *qbs-morphism-comp*[*of* (λ(*x*,*y*). (*y*,*x*)) *Y* ⨂$_Q$ *X* - *f*] *qbs-morphism-pair-swap′* *assms*
by *auto*
qed

lemma *qbs-morphism-pair-assoc1*:
(λ((*x*,*y*),*z*). (*x*,(*y*,*z*))) ∈ (*X* ⨂$_Q$ *Y*) ⨂$_Q$ *Z* →$_Q$ *X* ⨂$_Q$ (*Y* ⨂$_Q$ *Z*)
by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx-def split-beta′ comp-def*)

lemma *qbs-morphism-pair-assoc2*:
(λ(*x*,(*y*,*z*)). ((*x*,*y*),*z*)) ∈ *X* ⨂$_Q$ (*Y* ⨂$_Q$ *Z*) →$_Q$ (*X* ⨂$_Q$ *Y*) ⨂$_Q$ *Z*
by(*auto intro*!: *qbs-morphismI simp*: *pair-qbs-Mx-def split-beta′ comp-def*)

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

lemma *pair-qbs-snd*:
assumes *qbs-space X* ≠ {}
shows *map-qbs snd* (*X* ⨂$_Q$ *Y*) = *Y*

proof(*rule qbs-eqI*)
  show *qbs-Mx* (*map-qbs snd* ($X \bigotimes_Q Y$)) = *qbs-Mx Y*
  proof *auto*
    fix αy
    assume *hy*:$\alpha y \in$ *qbs-Mx Y*
    obtain αx **where** *hx*:$\alpha x \in$ *qbs-Mx X*
      using *qbs-empty-equiv*[*of X*] *assms*
      by *auto*
    show $\exists \alpha \in$*pair-qbs-Mx X Y*. αy = *snd* $\circ$ α
      by(*auto intro*!: *exI*[**where** *x*=λr. (αx *r*, αy *r*)] *simp*: *pair-qbs-Mx-def hx hy comp-def*)
  qed (*simp add*: *pair-qbs-Mx-def*)
qed

The following lemma corresponds to [1] Proposition 19(1).

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-def*)

lemma *l-product-sets*[*simp*,*measurable-cong*]:
  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*,*simp*)
  fix *A B*
  assume *h*:$A \in$ *sigma-Mx X*
         $B \in$ *sigma-Mx Y*
  then obtain *Ua Ub* **where** *hu*:
   A = *Ua* $\cap$ *qbs-space X* $\forall \alpha \in$*qbs-Mx X*. α −' *Ua* $\in$ *sets real-borel*
   B = *Ub* $\cap$ *qbs-space Y* $\forall \alpha \in$*qbs-Mx Y*. α −' *Ub* $\in$ *sets real-borel*
    by(*auto simp add*: *sigma-Mx-def*)
  show $A \times B \in$ *sigma-Mx* ($X \bigotimes_Q Y$)
  proof(*simp add*: *sigma-Mx-def*, *rule exI*[**where** *x*=*Ua* $\times$ *Ub*])
    show $A \times B$ = *Ua* $\times$ *Ub* $\cap$ *qbs-space X* $\times$ *qbs-space Y* $\wedge$
    ($\forall \alpha \in$*pair-qbs-Mx X Y*. α −' (*Ua* $\times$ *Ub*) $\in$ *sets real-borel*)
      using *hu* **by**(*auto simp add*: *pair-qbs-Mx-def vimage-Times*)
  qed
qed

lemma(**in** *pair-standard-borel*) *l-r-r-sets*[*simp*,*measurable-cong*]:
 sets (*qbs-to-measure* (*measure-to-qbs M* $\bigotimes_Q$ *measure-to-qbs N*)) = *sets* ($M \bigotimes_M N$)
  by(*simp only*: *r-preserves-product*[*symmetric*]) (*rule standard-borel-lr-sets-ident*)

end

2.3.2 Product Spaces

theory *Product-QuasiBorel*

imports *Binary-Product-QuasiBorel*

begin

definition *prod-qbs-Mx* :: [*'a set*, *'a* ⇒ *'b quasi-borel*] ⇒ (*real* ⇒ *'a* ⇒ *'b*) *set*
where
prod-qbs-Mx I M ≡ { α | α. ∀ *i*. (*i* ∈ *I* ⟶ (λ*r*. α *r i*) ∈ *qbs-Mx* (*M i*)) ∧ (*i* ∉ *I* ⟶ (λ*r*. α *r i*) = (λ*r*. *undefined*))}

lemma *prod-qbs-MxI*:
  assumes ⋀*i*. *i* ∈ *I* ⟹ (λ*r*. α *r i*) ∈ *qbs-Mx* (*M i*)
      and ⋀*i*. *i* ∉ *I* ⟹ (λ*r*. α *r i*) = (λ*r*. *undefined*)
    shows α ∈ *prod-qbs-Mx I M*
  using *assms* **by**(*auto simp*: *prod-qbs-Mx-def*)

lemma *prod-qbs-MxE*:
  assumes α ∈ *prod-qbs-Mx I M*
  shows ⋀*i*. *i* ∈ *I* ⟹ (λ*r*. α *r i*) ∈ *qbs-Mx* (*M i*)
    and ⋀*i*. *i* ∉ *I* ⟹ (λ*r*. α *r i*) = (λ*r*. *undefined*)
    and ⋀*i r*. *i* ∉ *I* ⟹ α *r i* = *undefined*
  using *assms* **by**(*auto simp*: *prod-qbs-Mx-def dest*: *fun-cong*[**where** *g*=(λ*r*. *undefined*)])

definition *PiQ* :: *'a set* ⇒ (*'a* ⇒ *'b quasi-borel*) ⇒ (*'a* ⇒ *'b*) *quasi-borel* **where**
PiQ I M ≡ *Abs-quasi-borel* (Π_E *i*∈*I*. *qbs-space* (*M i*), *prod-qbs-Mx I M*)

syntax
  -PiQ :: *pttrn* ⇒ *'i set* ⇒ *'a quasi-borel* ⇒ (*'i* => *'a*) *quasi-borel* (‹(*3*Π_Q -∈-./ -)› *10*)
syntax-consts
  -PiQ == *PiQ*
translations
  Π_Q *x*∈*I*. *M* == *CONST PiQ I* (λ*x*. *M*)

lemma *PiQ-f*: *prod-qbs-Mx I M* ⊆ *UNIV* → (Π_E *i*∈*I*. *qbs-space* (*M i*))
  using *prod-qbs-MxE* **by** *fastforce*

lemma *PiQ-closed1*: *qbs-closed1* (*prod-qbs-Mx I M*)
proof(*rule qbs-closed1I*)
  fix α *f*
  assume *h*:α ∈ *prod-qbs-Mx I M*
         f ∈ *real-borel* $\rightarrow_M$ *real-borel*
  show α ∘ *f* ∈ *prod-qbs-Mx I M*
  proof(*rule prod-qbs-MxI*)
    fix *i*
    assume *i* ∈ *I*
    from *prod-qbs-MxE*(*1*)[*OF h*(*1*) *this*]
    have (λ*r*. α *r i*) ∘ *f* ∈ *qbs-Mx* (*M i*)

using $h(2)$ **by** *auto*
thus $(\lambda r.\ (\alpha \circ f)\ r\ i) \in$ *qbs-Mx* $(M\ i)$
by(*simp add*: *comp-def*)
next
fix i
assume $i \notin I$
from *prod-qbs-MxE*(*3*)[*OF h*(*1*) *this*]
show $(\lambda r.\ (\alpha \circ f)\ r\ i) = (\lambda r.$ *undefined*$)$
by *simp*
qed
qed

lemma *PiQ-closed2*: *qbs-closed2* $(\Pi_E\ i{\in}I.$ *qbs-space* $(M\ i))$ (*prod-qbs-Mx I M*)
proof(*rule qbs-closed2I*)
fix x
assume $h{:}x \in (\Pi_E\ i{\in}I.$ *qbs-space* $(M\ i))$
show $(\lambda r.\ x) \in$ *prod-qbs-Mx I M*
proof(*rule prod-qbs-MxI*)
fix i
assume $hi{:}i \in I$
then have $x\ i \in$ *qbs-space* $(M\ i)$
using h **by** *auto*
thus $(\lambda r.\ x\ i) \in$ *qbs-Mx* $(M\ i)$
by *auto*
next
show $\bigwedge i.\ i \notin I \Longrightarrow (\lambda r.\ x\ i) = (\lambda r.$ *undefined*$)$
using h **by** *auto*
qed
qed

lemma *PiQ-closed3*: *qbs-closed3* (*prod-qbs-Mx I M*)
proof(*rule qbs-closed3I*)
fix *P Fi*
assume $h{:}\bigwedge i{::}nat.\ P -\text{`}\ \{i\} \in$ *sets real-borel*
$\bigwedge i{::}nat.\ Fi\ i \in$ *prod-qbs-Mx I M*
show $(\lambda r.\ Fi\ (P\ r)\ r) \in$ *prod-qbs-Mx I M*
proof(*rule prod-qbs-MxI*)
fix i
assume $hi{:}i \in I$
show $(\lambda r.\ Fi\ (P\ r)\ r\ i) \in$ *qbs-Mx* $(M\ i)$
using *prod-qbs-MxE*(*1*)[*OF h*(*2*) *hi*] *qbs-closed3-dest*[*OF h*(*1*),*of* $\lambda j\ r.\ Fi\ j\ r$
i]
by *auto*
next
show $\bigwedge i.\ i \notin I \Longrightarrow$
$(\lambda r.\ Fi\ (P\ r)\ r\ i) = (\lambda r.$ *undefined*$)$
using *prod-qbs-MxE*[*OF h*(*2*)] **by** *auto*
qed
qed

lemma *PiQ-correct*: *Rep-quasi-borel* (*PiQ I M*) = (Π_E *i*∈*I*. *qbs-space* (*M i*), *prod-qbs-Mx I M*)
 by(*auto intro*!: *Abs-quasi-borel-inverse PiQ-f is-quasi-borel-intro simp*: *PiQ-def PiQ-closed1 PiQ-closed2 PiQ-closed3*)

lemma *PiQ-space*[*simp*]: *qbs-space* (*PiQ I M*) = (Π_E *i*∈*I*. *qbs-space* (*M i*))
 by(*simp add*: *qbs-space-def PiQ-correct*)

lemma *PiQ-Mx*[*simp*]: *qbs-Mx* (*PiQ I M*) = *prod-qbs-Mx I M*
 by(*simp add*: *qbs-Mx-def PiQ-correct*)

lemma *qbs-morphism-component-singleton*:
 assumes *i* ∈ *I*
 shows (λ*x*. *x i*) ∈ (Π_Q *i*∈*I*. (*M i*)) $\to_Q$ *M i*
 by(*auto intro*!: *qbs-morphismI simp*: *prod-qbs-Mx-def comp-def assms*)

lemma *product-qbs-canonical1*:
 assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *Y* $\to_Q$ *X i*
  and ⋀*i*. *i* ∉ *I* ⟹ *f i* = (λ*y*. *undefined*)
 shows (λ*y i*. *f i y*) ∈ *Y* $\to_Q$ (Π_Q *i*∈*I*. *X i*)
 using *qbs-morphismE*(*3*)[*simplified comp-def*,*OF assms*(*1*)] *assms*(*2*)
 by(*auto intro*!: *qbs-morphismI prod-qbs-MxI*)

lemma *product-qbs-canonical2*:
 assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *Y* $\to_Q$ *X i*
  ⋀*i*. *i* ∉ *I* ⟹ *f i* = (λ*y*. *undefined*)
  g ∈ *Y* $\to_Q$ (Π_Q *i*∈*I*. *X i*)
  ⋀*i*. *i* ∈ *I* ⟹ *f i* = (λ*x*. *x i*) ∘ *g*
  and *y* ∈ *qbs-space Y*
 shows *g y* = (λ*i*. *f i y*)
proof(*rule 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*
  moreover have (λ*r*. *y*) ∈ *qbs-Mx Y*
   using *assms*(*5*) **by** *simp*
  ultimately show *?thesis*
   using *assms*(*2*,*3*) *qbs-morphismE*(*3*)[*OF assms*(*3*) -]
   by(*fastforce simp*: *prod-qbs-Mx-def*)
 qed
qed

lemma *merge-qbs-morphism*:
merge I J ∈ $(\Pi_Q\ i{\in}I.\ (M\ i)) \bigotimes_Q (\Pi_Q\ j{\in}J.\ (M\ j)) \to_Q (\Pi_Q\ i{\in}I{\cup}J.\ (M\ i))$
proof(*rule qbs-morphismI*)
  fix α
  assume *h*:$\alpha \in$ *qbs-Mx* $((\Pi_Q\ i{\in}I.\ (M\ i)) \bigotimes_Q (\Pi_Q\ j{\in}J.\ (M\ j)))$
  show *merge I J* $\circ\ \alpha \in$ *qbs-Mx* $(\Pi_Q\ i{\in}I{\cup}J.\ (M\ i))$
  proof(*simp*, *rule prod-qbs-MxI*)
    fix *i*
    assume $i \in I \cup J$
    then consider $i \in I \mid i \in I \wedge i \in J \mid i \notin I \wedge i \in J$
      by *auto*
    then show $(\lambda r.\ (merge\ I\ J \circ \alpha)\ r\ i) \in$ *qbs-Mx* $(M\ i)$
      apply *cases*
      using *h*
      by(*auto simp*: *merge-def pair-qbs-Mx-def split-beta′ dest*: *prod-qbs-MxE*)
  next
    fix *i*
    assume $i \notin I \cup J$
    then show $(\lambda r.\ (merge\ I\ J \circ \alpha)\ r\ i) = (\lambda r.\ undefined)$
      using *h*
      by(*auto simp*: *merge-def pair-qbs-Mx-def split-beta′ dest*: *prod-qbs-MxE*)
  qed
qed

The following lemma corresponds to [1] Proposition 19(1).

lemma *r-preserves-product′*:
  measure-to-qbs $(\Pi_M\ i{\in}I.\ M\ i) = (\Pi_Q\ i{\in}I.$ *measure-to-qbs* $(M\ i))$
proof(*rule qbs-eqI*)
  show *qbs-Mx* (*measure-to-qbs* $(Pi_M\ I\ M)) =$ *qbs-Mx* $(\Pi_Q\ i{\in}I.$ *measure-to-qbs* $(M\ i))$
  proof *auto*
    fix *f*
    assume *h*:$f \in$ *real-borel* $\to_M Pi_M\ I\ M$
    show $f \in$ *prod-qbs-Mx I* $(\lambda i.$ *measure-to-qbs* $(M\ i))$
    proof(*rule prod-qbs-MxI*)
      fix *i*
      assume *1*:$i \in I$
      show $(\lambda r.\ f\ r\ i) \in$ *qbs-Mx* (*measure-to-qbs* $(M\ i))$
        using *measurable-comp*[*OF h measurable-component-singleton*[*OF 1*,*of M*]]
        by (*simp add*: *comp-def*)
    next
      fix *i*
      assume *1*:$i \notin I$
      then show $(\lambda r.\ f\ r\ i) = (\lambda r.\ undefined)$
        using *measurable-space*[*OF h*] *1*
        by(*auto simp*: *space-PiM PiE-def extensional-def*)
    qed
  next
    fix *f*

assume *h*:$f \in$ *prod-qbs-Mx I* (λi. *measure-to-qbs* (*M i*))
show $f \in$ *real-borel* $\to_M$ Pi_M *I M*
apply(*rule measurable-PiM-single'*)
using *prod-qbs-MxE*(*1*)[*OF h*] **apply** *auto*[*1*]
using *PiQ-f*[*of I M*] *h* **by** *auto*
qed
qed

$\prod_{i=0,1} X_i \cong X_1 \times X_2$.

lemma *product-binary-product*:
$\exists f\ g.\ f \in (\Pi_Q\ i{\in}UNIV.\ if\ i\ then\ X\ else\ Y) \to_Q X \bigotimes_Q Y \wedge g \in X \bigotimes_Q Y \to_Q (\Pi_Q\ i{\in}UNIV.\ if\ i\ then\ X\ else\ Y) \wedge$
$g \circ f = id \wedge f \circ g = id$
by(*auto intro*!: *exI*[**where** $x{=}\lambda f.$ (*f True*, *f False*)] *exI*[**where** $x{=}\lambda xy\ b.$ *if b then fst xy else snd xy*] *qbs-morphismI*
simp: *prod-qbs-Mx-def pair-qbs-Mx-def comp-def all-bool-eq ext*)

end

2.4 Coproduct Spaces

theory *Binary-CoProduct-QuasiBorel*
imports *Measure-QuasiBorel-Adjunction*
begin

2.4.1 Binary Coproduct Spaces

definition *copair-qbs-Mx* :: ['*a quasi-borel*, '*b quasi-borel*] ⇒ (*real* => '*a* + '*b*) *set*
where
copair-qbs-Mx X Y ≡
$\{g.\ \exists\ S \in$ *sets real-borel*.
$(S = \{\} \longrightarrow (\exists\ \alpha 1{\in}$ *qbs-Mx X*. $g = (\lambda r.$ *Inl* $(\alpha 1\ r)))) \wedge$
$(S = UNIV \longrightarrow (\exists\ \alpha 2{\in}$ *qbs-Mx Y*. $g = (\lambda r.$ *Inr* $(\alpha 2\ r)))) \wedge$
$((S \neq \{\} \wedge S \neq UNIV) \longrightarrow$
$(\exists\ \alpha 1{\in}$ *qbs-Mx X*.
$\exists\ \alpha 2{\in}$ *qbs-Mx Y*.
$g = (\lambda r{::}real.\ (if\ (r \in S)\ then\ Inl\ (\alpha 1\ r)\ else\ Inr\ (\alpha 2\ r)))))\}$

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 followin 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 real-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(*auto*)

fix *g* :: *real* ⇒ ′*a* + ′*b*
assume *g* ∈ *copair-qbs-Mx X Y*
then obtain *S* **where** *hs*:*S*∈ *sets real-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* ⇒ ′*a* + ′*b*
where *g*′ ≡ (λ*r*::*real*. (*if* (*r* ∈ *S*′) *then Inl* (α1 *r*) *else Inr* (α2 *r*)))
from ‹*qbs-space X* ≠ {} ∧ *qbs-space Y* ≠ {}› *h1* ‹α2 ∈ *qbs-Mx Y*›
have *g*′ ∈ *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 = (λr. Inl (α1 r)))
      else if qbs-space X = {} ∧ qbs-space Y ≠ {} then
                (∃ α2∈ qbs-Mx Y. g = (λr. Inr (α2 r)))
      else
          (∃ S ∈ sets real-borel. ∃ α1∈ qbs-Mx X. ∃ α2∈ qbs-Mx Y.
           g = (λr::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 = (λr. 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 = (λr. 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 have
        ∃ S ∈ sets real-borel. ∃ α1∈ qbs-Mx X. ∃ α2∈ qbs-Mx Y.
           g = (λr::real. (if (r ∈ S) then Inl (α1 r) else Inr (α2 r))) by simp
    then show ?thesis
    proof(auto simp add: exE)
      fix S
      fix α1
      fix α2
      assume S ∈ sets real-borel
             α1 ∈ qbs-Mx X
             α2 ∈ qbs-Mx Y
             g = (λr. if r ∈ S then Inl (α1 r)
                                  else Inr (α2 r))
      consider S = {} | S = UNIV | S ≠ {} ∧ S ≠ UNIV by auto
```

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


lemma copair-qbs-f[simp]: 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 real-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 by auto
  next
```

assume *S* = *UNIV*
then show *?thesis*
using *hs* **by** *auto*
next
assume *S* ≠ {} ∧ *S* ≠ *UNIV*
then have ∃ *α1*∈ *qbs-Mx X*. ∃ *α2*∈ *qbs-Mx Y*.
g = (λ*r*::*real*. (*if* (*r* ∈ *S*) *then Inl* (*α1 r*) *else Inr* (*α2 r*))) **using** *hs* **by** *simp*
then show *?thesis*
by(*auto simp add*: *exE*)
qed
qed

lemma *copair-qbs-closed1*: *qbs-closed1* (*copair-qbs-Mx X Y*)
proof(*auto simp add*: *qbs-closed1-def*)
fix *g*
fix *f*
assume *g* ∈ *copair-qbs-Mx X Y*
f ∈ *real-borel* $\rightarrow_M$ *real-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* = (λ*r*. *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*
using ‹*f* ∈ *real-borel* $\rightarrow_M$ *real-borel*› **by** *auto*
moreover have *g* ∘ *f* = (λ*r*. *Inl* ((*α1* ∘ *f*) *r*))
using *h1* **by** *auto*
ultimately show *?thesis*
using ‹*qbs-space X* ≠ {} ∧ *qbs-space Y* = {}› **by**(*force simp add*: *copair-qbs-Mx2-def*)
next
assume (*qbs-space X* = {} ∧ *qbs-space Y* ≠ {})
then obtain *α2* **where** *h2*:*α2*∈ *qbs-Mx Y* ∧ *g* = (λ*r*. *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*
using ‹*f* ∈ *real-borel* $\rightarrow_M$ *real-borel*› **by** *auto*
moreover have *g* ∘ *f* = (λ*r*. *Inr* ((*α2* ∘ *f*) *r*))
using *h2* **by** *auto*
ultimately show *?thesis*

```
      using ‹(qbs-space X = {} ∧ qbs-space Y ≠ {})› by(force simp add: copair-qbs-Mx2-def)
  next
    assume qbs-space X ≠ {} ∧ qbs-space Y ≠ {}
    then have ∃ S ∈ sets real-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(auto simp add: exE)
      fix S
      fix α1
      fix α2
      assume S ∈ sets real-borel
             α1∈ qbs-Mx X
             α2 ∈ qbs-Mx Y
             g = (λr. if r ∈ S then Inl (α1 r) else Inr (α2 r))
      have f −‘ S ∈ sets real-borel
        using ‹f ∈ real-borel →M real-borel› ‹S ∈ sets real-borel›
        by (simp add: measurable-sets-borel)
      moreover have α1 ∘ f ∈ qbs-Mx X
        using ‹α1∈ qbs-Mx X› ‹f ∈ real-borel →M real-borel› qbs-decomp
        by(auto simp add: qbs-closed1-def)
      moreover have α2 ∘ f ∈ qbs-Mx Y
        using ‹α2∈ qbs-Mx Y› ‹f ∈ real-borel →M real-borel› qbs-decomp
        by(auto simp add: qbs-closed1-def)
      moreover have
        (λr. if r ∈ S then Inl (α1 r) else Inr (α2 r)) ∘ f = (λr. if r ∈ f −‘ S then Inl ((α1 ∘ f) r) else Inr ((α2 ∘ f) r))
        by auto
        ultimately show (λr. if r ∈ S then Inl (α1 r)  else Inr (α2 r)) ∘ f ∈ copair-qbs-Mx2 X Y
          using ‹qbs-space X ≠ {} ∧ qbs-space Y ≠ {}› by(force simp add: copair-qbs-Mx2-def)
    qed
  qed
  thus g ∘ f ∈ copair-qbs-Mx X Y
    using copair-qbs-Mx-equiv by auto
qed

lemma copair-qbs-closed2: qbs-closed2 (qbs-space X <+> qbs-space Y) (copair-qbs-Mx X Y)
proof(auto simp add: qbs-closed2-def)
  fix x
  assume x ∈ qbs-space X
  define α1 :: real ⇒ - where α1 ≡ (λr. x)
  have α1 ∈ qbs-Mx X using ‹x ∈ qbs-space X› qbs-decomp
    by(force simp add: qbs-closed2-def α1-def )
  moreover have (λr. Inl x) = (λl. Inl (α1 l)) by (simp add: α1-def)
  moreover have {} ∈ sets real-borel by auto
```

ultimately show (*λr. Inl x*) ∈ *copair-qbs-Mx X Y*
by(*auto simp add*: *copair-qbs-Mx-def*)
next
fix *y*
assume *y* ∈ *qbs-space Y*
define *α2* :: *real* ⇒ - **where** *α2* ≡ (*λr. y*)
have *α2* ∈ *qbs-Mx Y* **using** ‹*y* ∈ *qbs-space Y*› *qbs-decomp*
by(*force simp add*: *qbs-closed2-def α2-def*)
moreover have (*λr. Inr y*) = (*λl. Inr* (*α2 l*)) **by** (*simp add*: *α2-def*)
moreover have *UNIV* ∈ *sets real-borel* **by** *auto*
ultimately show (*λr. Inr y*) ∈ *copair-qbs-Mx X Y*
unfolding *copair-qbs-Mx-def*
by(*auto intro*!: *bexI*[**where** *x*=*UNIV*])
qed

lemma *copair-qbs-closed3*: *qbs-closed3* (*copair-qbs-Mx X Y*)
proof(*auto simp add*: *qbs-closed3-def*)
fix *P* :: *real* ⇒ *nat*
fix *Fi* :: *nat* ⇒ *real* ⇒- + -
assume ∀ *i. P* −‘ {*i*} ∈ *sets real-borel*
∀ *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 ‹∀ *i. P* −‘ {*i*} ∈ *sets real-borel*› *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* = (λ*r*. *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* = (λ*r*. *Inr* (*α2 i r*))
by(*rule choice*)
then obtain *α2* :: *nat* ⇒ *real* ⇒ -
where *h2*: ∀ *i*. *α2 i* ∈ *qbs-Mx Y* ∧ *Fi i* = (λ*r*. *Inr* (*α2 i r*)) **by** *auto*
define *β* :: *real* ⇒ -
where *β* ≡ (λ*r*. *α2* (*P r*) *r*)
from ‹∀ *i*. *P* −‘ {*i*} ∈ *sets real-borel*› *h2 qbs-decomp*
have *β* ∈ *qbs-Mx Y*
by(*simp add*: *β-def*)
moreover have (λ*r*. *Fi* (*P r*) *r*) = (λ*r*. *Inr* (*β r*))
using *h2* **by**(*simp add*: *β-def*)
ultimately show *?thesis*
using ‹*qbs-space X* = {} ∧ *qbs-space Y* ≠ {}› **by** (*auto simp add*: *copair-qbs-Mx2-def*)
next
assume *qbs-space X* ≠ {} ∧ *qbs-space Y* ≠ {}
then have ∀ *i*. ∃ *Si*. *Si* ∈ *sets real-borel* ∧ (∃ *α1i*∈ *qbs-Mx X*. ∃ *α2i*∈ *qbs-Mx Y*.
Fi i = (λ*r*::*real*. (*if* (*r* ∈ *Si*) *then Inl* (*α1i r*) *else Inr* (*α2i r*))))
using ‹∀ *i*. *Fi i* ∈ *copair-qbs-Mx2 X Y*› **by** (*auto simp add*: *copair-qbs-Mx2-def*)
then have ∃ *S*. ∀ *i*. *S i* ∈ *sets real-borel* ∧ (∃ *α1i*∈ *qbs-Mx X*. ∃ *α2i*∈ *qbs-Mx Y*.
Fi i = (λ*r*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1i r*) *else Inr* (*α2i r*))))
by(*rule choice*)
then obtain *S* :: *nat* ⇒ *real set*
where *hs* :∀ *i*. *S i* ∈ *sets real-borel* ∧ (∃ *α1i*∈ *qbs-Mx X*. ∃ *α2i*∈ *qbs-Mx Y*.
Fi i = (λ*r*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1i r*) *else Inr* (*α2i r*))))
by *auto*
then have ∀ *i*. ∃ *α1i*. *α1i* ∈ *qbs-Mx X* ∧ (∃ *α2i*∈ *qbs-Mx Y*.
Fi i = (λ*r*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1i r*) *else Inr* (*α2i r*))))
by *blast*
then have ∃ *α1*. ∀ *i*. *α1 i* ∈ *qbs-Mx X* ∧ (∃ *α2i*∈ *qbs-Mx Y*.
Fi i = (λ*r*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1 i r*) *else Inr* (*α2i r*))))
by(*rule choice*)
then obtain *α1*
where *h1*: ∀ *i*. *α1 i* ∈ *qbs-Mx X* ∧ (∃ *α2i*∈ *qbs-Mx Y*.
Fi i = (λ*r*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1 i r*) *else Inr* (*α2i r*))))
by *auto*
define *β1* :: *real* ⇒ -
where *β1* ≡ (λ*r*. *α1* (*P r*) *r*)
from ‹∀ *i*. *P* −‘ {*i*} ∈ *sets real-borel*› *h1 qbs-decomp*
have *β1* ∈ *qbs-Mx X*
by(*simp add*: *β1-def*)
from *h1* **have** ∀ *i*. ∃ *α2i*. *α2i*∈ *qbs-Mx Y* ∧

Fi i = (*λr*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1 i r*) *else Inr* (*α2i r*)))
by *auto*
then have ∃ *α2*. ∀ *i*. *α2 i*∈ *qbs-Mx Y* ∧
Fi i = (*λr*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1 i r*) *else Inr* (*α2 i r*)))
by(*rule choice*)
then obtain *α2*
where *h2*: ∀ *i*. *α2 i*∈ *qbs-Mx Y* ∧
Fi i = (*λr*::*real*. (*if* (*r* ∈ *S i*) *then Inl* (*α1 i r*) *else Inr* (*α2 i r*)))
by *auto*
define *β2* :: *real* ⇒ -
where *β2* ≡ (*λr*. *α2* (*P r*) *r*)
from ‹∀ *i*. *P* −‘ {*i*} ∈ *sets real-borel*› *h2 qbs-decomp*
have *β2* ∈ *qbs-Mx Y*
by(*simp add*: *β2-def*)
define *A* :: *nat* ⇒ *real set*
where *A* ≡ (*λi*. *S i* ∩ *P* −‘ {*i*})
have ∀ *i*. *A i* ∈ *sets real-borel*
using *A-def* ‹∀ *i*. *P* −‘ {*i*} ∈ *sets real-borel*› *hs* **by** *blast*
define *S'* :: *real set*
where *S'* ≡ {*r*. *r* ∈ *S* (*P r*)}
have *S'* = (⋃ *i*::*nat*. *A i*)
by(*auto simp add*: *S'-def A-def*)
hence *S'* ∈ *sets real-borel*
using ‹∀ *i*. *A i* ∈ *sets real-borel*› **by** *auto*
from *h2* **have** (*λr*. *Fi* (*P r*) *r*) = (*λr*. (*if r* ∈ *S' then Inl* (*β1 r*)
else Inr (*β2 r*)))
by(*auto simp add*: *β1-def β2-def S'-def*)
thus (*λr*. *Fi* (*P r*) *r*) ∈ *copair-qbs-Mx2 X Y*
using ‹*qbs-space X* ≠ {} ∧ *qbs-space Y* ≠ {}› ‹*S'* ∈ *sets real-borel*› ‹*β1* ∈ *qbs-Mx X*› ‹*β2* ∈ *qbs-Mx Y*›
by(*auto simp add*: *copair-qbs-Mx2-def*)
qed
thus (*λr*. *Fi* (*P r*) *r*) ∈ *copair-qbs-Mx X Y*
using *copair-qbs-Mx-equiv* **by** *auto*
qed

lemma *copair-qbs-correct*: *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 copair-qbs-f simp*: *copair-qbs-closed2 copair-qbs-closed1 copair-qbs-closed3*)

lemma *copair-qbs-space*[*simp*]: *qbs-space* (*copair-qbs X Y*) = *qbs-space X* <+> *qbs-space Y*
by(*simp add*: *qbs-space-def copair-qbs-correct*)

lemma *copair-qbs-Mx*[*simp*]: *qbs-Mx* (*copair-qbs X Y*) = *copair-qbs-Mx X Y*
by(*simp add*: *qbs-Mx-def copair-qbs-correct*)

lemma *Inl-qbs-morphism*:
  $Inl \in X \rightarrow_Q X <+>_Q Y$
proof(*rule qbs-morphismI*)
  fix α
  assume $\alpha \in$ *qbs-Mx X*
  moreover have $Inl \circ \alpha = (\lambda r.\ Inl\ (\alpha\ r))$ **by** *auto*
  ultimately show $Inl \circ \alpha \in$ *qbs-Mx* $(X <+>_Q Y)$
    by(*auto simp add*: *copair-qbs-Mx-def*)
qed

lemma *Inr-qbs-morphism*:
  $Inr \in Y \rightarrow_Q X <+>_Q Y$
proof(*rule qbs-morphismI*)
  fix α
  assume $\alpha \in$ *qbs-Mx Y*
  moreover have $Inr \circ \alpha = (\lambda r.\ Inr\ (\alpha\ r))$ **by** *auto*
  ultimately show $Inr \circ \alpha \in$ *qbs-Mx* $(X <+>_Q Y)$
    by(*auto intro*!: *bexI*[**where** *x*=*UNIV*] *simp add*: *copair-qbs-Mx-def*)
qed

lemma *case-sum-preserves-morphisms*:
  assumes $f \in X \rightarrow_Q Z$
    and $g \in Y \rightarrow_Q Z$
  shows *case-sum f g* $\in X <+>_Q Y \rightarrow_Q Z$
proof(*rule qbs-morphismI*;*auto*)
  fix α
  assume $\alpha \in$ *copair-qbs-Mx X Y*
  then obtain *S* **where** *hs*:$S \in$ *sets real-borel* $\wedge$
  $(S = \{\} \longrightarrow (\exists\ \alpha 1 \in$ *qbs-Mx X*. $\alpha = (\lambda r.\ Inl\ (\alpha 1\ r)))) \wedge$
  $(S = UNIV \longrightarrow (\exists\ \alpha 2 \in$ *qbs-Mx Y*. $\alpha = (\lambda r.\ Inr\ (\alpha 2\ r)))) \wedge$
  $((S \neq \{\} \wedge S \neq UNIV) \longrightarrow$
    $(\exists\ \alpha 1 \in$ *qbs-Mx X*.
     $\exists\ \alpha 2 \in$ *qbs-Mx Y*.
        $\alpha = (\lambda r{::}real.\ (if\ (r \in S)\ then\ Inl\ (\alpha 1\ r)\ else\ Inr\ (\alpha 2\ r)))))$
    by (*auto simp add*: *copair-qbs-Mx-def*)
  consider $S = \{\} \mid S = UNIV \mid S \neq \{\} \wedge S \neq UNIV$ **by** *auto*
  then show *case-sum f g* $\circ\ \alpha \in$ *qbs-Mx Z*
  proof *cases*
    assume $S = \{\}$
    then obtain $\alpha 1$ **where** *h1*: $\alpha 1 \in$ *qbs-Mx X* $\wedge\ \alpha = (\lambda r.\ Inl\ (\alpha 1\ r))$
      using *hs* **by** *auto*
    then have $f \circ \alpha 1 \in$ *qbs-Mx Z*
      using *assms* **by**(*auto simp add*: *qbs-morphism-def*)
    moreover have *case-sum f g* $\circ\ \alpha = f \circ \alpha 1$
      using *h1* **by** *auto*
    ultimately show *?thesis* **by** *simp*
  next
    assume $S = UNIV$

```
  then obtain α2 where h2: α2∈ qbs-Mx Y ∧ α = (λr. Inr (α2 r))
    using hs by auto
  then have g ∘ α2 ∈ qbs-Mx Z
    using assms by(auto simp add: qbs-morphism-def)
  moreover have case-sum f g ∘ α = g ∘ α2
    using h2 by auto
  ultimately show ?thesis by simp
next
  assume S ≠ {} ∧ S ≠ UNIV
  then obtain α1 α2 where h: α1∈ qbs-Mx X ∧ α2∈ qbs-Mx Y ∧
       α = (λr::real. (if (r ∈ S) then Inl (α1 r) else Inr (α2 r)))
    using hs by auto
  define F :: nat ⇒ real ⇒ -
    where F ≡ (λi r. (if i = 0 then (f ∘ α1) r
                                  else (g ∘ α2) r))
  define P :: real ⇒ nat
    where P ≡ (λr. if r ∈ S then 0 else 1)
  have f ∘ α1 ∈ qbs-Mx Z
    using assms h by(simp add: qbs-morphism-def)
  have g ∘ α2 ∈ qbs-Mx Z
    using assms h by(simp add: qbs-morphism-def)
  have ∀ i. F i ∈ qbs-Mx Z
  proof(auto simp add: F-def)
    fix i :: nat
    consider i = 0 | i ≠ 0 by auto
    then show (λr. if i = 0 then (f ∘ α1) r else (g ∘ α2) r) ∈ qbs-Mx Z
    proof cases
      assume i = 0
     then have (λr. if i = 0 then (f ∘ α1) r else (g ∘ α2) r) = f ∘ α1 by auto
      then show ?thesis
        using ‹f ∘ α1 ∈ qbs-Mx Z› by simp
    next
      assume i ≠ 0
     then have (λr. if i = 0 then (f ∘ α1) r else (g ∘ α2) r) = g ∘ α2 by auto
      then show ?thesis
        using ‹g ∘ α2 ∈ qbs-Mx Z› by simp
    qed
  qed
  moreover have ∀ i. P −‘{i} ∈ sets real-borel
  proof
    fix i :: nat
    consider i = 0 | i = 1 | i ≠ 0 ∧ i ≠ 1 by auto
    then show P −‘{i} ∈ sets real-borel
    proof cases
      assume i = 0
      then show ?thesis
        using hs by(simp add: P-def)
    next
      assume i = 1
```

```
      then show ?thesis
        using hs by (simp add: P-def borel-comp)
    next
      assume i ≠ 0 ∧ i ≠ 1
      then show ?thesis by(simp add: P-def)
    qed
  qed
  ultimately have (λr. F (P r) r) ∈ qbs-Mx Z
    by simp
  moreover have case-sum f g ∘ α = (λr. F (P r) r)
    using h by(auto simp add: F-def P-def)
  ultimately show case-sum f g ∘ α ∈ qbs-Mx Z by simp
 qed
qed


lemma map-sum-preserves-morphisms:
 assumes f ∈ X →Q Y
     and g ∈ X' →Q Y'
   shows map-sum f g ∈ X <+>Q X' →Q Y <+>Q Y'
proof(rule qbs-morphismI,simp)
 fix α
 assume α ∈ copair-qbs-Mx X X'
 then obtain S where hs:S∈ sets real-borel ∧
 (S = {}   ⟶ (∃ α1∈ qbs-Mx X. α = (λr. Inl (α1 r)))) ∧
 (S = UNIV ⟶ (∃ α2∈ qbs-Mx X'. α = (λr. Inr (α2 r)))) ∧
 ((S ≠ {} ∧ S ≠ UNIV) ⟶
   (∃ α1∈ qbs-Mx X.
   ∃ α2∈ qbs-Mx X'.
       α = (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 map-sum f g ∘ α ∈ copair-qbs-Mx Y Y'
 proof cases
   assume S = {}
   then obtain α1 where h1: α1∈ qbs-Mx X ∧ α = (λr. Inl (α1 r))
     using hs by auto
   define f' :: real ⇒ - where f' ≡ f ∘ α1
   then have f' ∈ qbs-Mx Y
     using assms h1 by(simp add: qbs-morphism-def)
   moreover have map-sum f g ∘ α = (λr. Inl (f' r))
     using h1 by (auto simp add: f'-def)
   moreover have {} ∈ sets real-borel by simp
   ultimately show ?thesis
     by(auto simp add: copair-qbs-Mx-def)
 next
   assume S = UNIV
   then obtain α2 where h2: α2∈ qbs-Mx X' ∧ α = (λr. Inr (α2 r))
     using hs by auto
```

```
    define g′ :: real ⇒ - where g′ ≡ g ∘ α2
    then have g′ ∈ qbs-Mx Y′
      using assms h2 by(simp add: qbs-morphism-def)
    moreover have map-sum f g ∘ α = (λr. Inr (g′ r))
      using h2 by (auto simp add: g′-def)
    ultimately show ?thesis
      by(auto intro!: bexI[where x=UNIV] simp add: copair-qbs-Mx-def)
  next
    assume S ≠ {} ∧ S ≠ UNIV
    then obtain α1 α2 where h: α1∈ qbs-Mx X ∧ α2∈ qbs-Mx X′ ∧
          α = (λr::real. (if (r ∈ S) then Inl (α1 r) else Inr (α2 r)))
      using hs by auto
    define f′ :: real ⇒ - where f′ ≡ f ∘ α1
    define g′ :: real ⇒ - where g′ ≡ g ∘ α2
    have f′ ∈ qbs-Mx Y
      using assms h by(auto simp: f′-def)
    moreover have g′ ∈ qbs-Mx Y′
      using assms h by(auto simp: g′-def)
    moreover have map-sum f g ∘ α = (λr::real. (if (r ∈ S) then Inl (f′ r) else
Inr (g′ r)))
      using h by(auto simp add: f′-def g′-def)
    moreover have S ∈ sets real-borel using hs by simp
    ultimately show ?thesis
      using ‹S ≠ {} ∧ S ≠ UNIV› by(auto simp add: copair-qbs-Mx-def)
  qed
qed


end
```

2.4.2 Countable Coproduct Spaces

```
theory CoProduct-QuasiBorel

imports
 Product-QuasiBorel
 Binary-CoProduct-QuasiBorel
begin

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

lemma coprod-qbs-MxI:
  assumes f ∈ real-borel →M count-space I
      and ⋀i. i ∈ range f ⟹ α i ∈ qbs-Mx (X i)
    shows (λr. (f r, α (f r) r)) ∈ coprod-qbs-Mx I X
  using assms unfolding coprod-qbs-Mx-def by blast
```

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

lemma *coproduct-qbs-Mx-eq*:
coprod-qbs-Mx I X = *coprod-qbs-Mx′ I X*
proof *auto*
 fix α
 assume α ∈ *coprod-qbs-Mx I X*
 then obtain *f* β **where** *hfb*:
  f ∈ *real-borel* $\rightarrow_M$ *count-space I*
  ⋀*i*. *i* ∈ *range f* ⟹ β *i* ∈ *qbs-Mx* (*X i*) α = (λ*r*. (*f r*, β (*f r*) *r*))
  unfolding *coprod-qbs-Mx-def* **by** *blast*
 define β′ **where** β′ ≡ (λ*i*. *if i* ∈ *range f then* β *i*
          else if qbs-space (*X i*) ≠ {} *then* (*SOME* γ. γ ∈ *qbs-Mx*
(*X i*))
          else β *i*)
 have *1*:α = (λ*r*. (*f r*, β′ (*f r*) *r*))
  by(*simp add*: *hfb*(*3*) β′*-def*)
 have *2*:⋀*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 (λ*r*. *x*) ∈ *qbs-Mx* (*X i*) **by** *auto*
  thus β′ *i* ∈ *qbs-Mx* (*X i*)
   by(*cases i* ∈ *range f*) (*auto simp*: β′*-def hfb*(*2*) *hne intro*!: *someI2*[**where**
a=λ*r*. *x*])
 qed
 show α ∈ *coprod-qbs-Mx′ I X*
  using *hfb*(*1*,*2*) *1 2* **by**(*auto simp*: *coprod-qbs-Mx′-def intro*!: *exI*[**where** *x*=*f*]
exI[**where** *x*=β′])
next
 fix α
 assume α ∈ *coprod-qbs-Mx′ I X*
 then obtain *f* β **where** *hfb*:
  f ∈ *real-borel* $\rightarrow_M$ *count-space I* ⋀*i*. *qbs-space* (*X i*) ≠ {} ⟹ β *i* ∈ *qbs-Mx*
(*X i*)
  ⋀*i*. *i* ∈ *range f* ⟹ β *i* ∈ *qbs-Mx* (*X i*) α = (λ*r*. (*f r*, β (*f r*) *r*))
  unfolding *coprod-qbs-Mx′-def* **by** *blast*
 show α ∈ *coprod-qbs-Mx I X*
  by(*auto simp*: *hfb*(*4*) *intro*!: *coprod-qbs-MxI*[*OF hfb*(*1*) *hfb*(*3*)])
qed

definition *coprod-qbs* :: [*′a set*, *′a* ⇒ *′b quasi-borel*] ⇒ (*′a* × *′b*) *quasi-borel* **where**
coprod-qbs I X ≡ *Abs-quasi-borel* (*SIGMA i*:*I*. *qbs-space* (*X i*), *coprod-qbs-Mx I X*)

syntax
-*coprod-qbs* :: *pttrn* ⇒ *'i set* ⇒ *'a quasi-borel* ⇒ (*'i* × *'a*) *quasi-borel* (‹(*3*$\amalg_Q$ -∈-./ -)› *10*)
syntax-consts
-*coprod-qbs* ⇌ *coprod-qbs*
translations
$\amalg_Q$ *x*∈*I*. *M* ⇌ *CONST coprod-qbs I* (λ*x*. *M*)

lemma *coprod-qbs-f*[*simp*]: *coprod-qbs-Mx I X* ⊆ *UNIV* → (*SIGMA i*:*I*. *qbs-space* (*X i*))
  by(*fastforce simp*: *coprod-qbs-Mx-def dest*: *measurable-space*)

lemma *coprod-qbs-closed1*: *qbs-closed1* (*coprod-qbs-Mx I X*)
proof(*rule qbs-closed1I*)
  fix α *f*
  assume α ∈ *coprod-qbs-Mx I X*
    and *1*[*measurable*]: *f* ∈ *real-borel* $\rightarrow_M$ *real-borel*
  then obtain β *g* **where** *ha*:
    ⋀*i*. *i* ∈ *range g* ⟹ β *i* ∈ *qbs-Mx* (*X i*) α = (λ*r*. (*g r*, β (*g r*) *r*)) **and** [*measurable*]:*g* ∈ *real-borel* $\rightarrow_M$ *count-space I*
    by(*fastforce simp*: *coprod-qbs-Mx-def*)
  then have ⋀*i*. *i* ∈ *range g* ⟹ β *i* ∘ *f* ∈ *qbs-Mx* (*X i*)
    by *simp*
  thus α ∘ *f* ∈ *coprod-qbs-Mx I X*
    by(*auto intro*!: *coprod-qbs-MxI*[**where** *f*=*g* ∘ *f* **and** α=λ*i*. β *i* ∘ *f*,*simplified comp-def*] *simp*: *ha*(*2*) *comp-def*)
qed

lemma *coprod-qbs-closed2*: *qbs-closed2* (*SIGMA i*:*I*. *qbs-space* (*X i*)) (*coprod-qbs-Mx I X*)
proof(*rule qbs-closed2I*,*auto*)
  fix *i x*
  assume *i* ∈ *I x* ∈ *qbs-space* (*X i*)
  then show (λ*r*. (*i*,*x*)) ∈ *coprod-qbs-Mx I X*
    by(*auto simp*: *coprod-qbs-Mx-def intro*!: *exI*[**where** *x*=λ*r*. *i*])
qed

lemma *coprod-qbs-closed3*:
 qbs-closed3 (*coprod-qbs-Mx I X*)
proof(*rule qbs-closed3I*)
  fix *P Fi*
  assume *h*:⋀*i* :: *nat*. *P* −‘ {*i*} ∈ *sets real-borel*
        ⋀*i* :: *nat*. *Fi i* ∈ *coprod-qbs-Mx I X*
  then have ∀*i*. ∃*fi* α*i*. *Fi i* = (λ*r*. (*fi r*, α*i* (*fi r*) *r*)) ∧ *fi* ∈ *real-borel* $\rightarrow_M$ *count-space I* ∧ (∀*j*. (*j* ∈ *range fi* ∨ *qbs-space* (*X j*) ≠ {}) ⟶ α*i j* ∈ *qbs-Mx* (*X j*))
    by(*auto simp*: *coproduct-qbs-Mx-eq coprod-qbs-Mx'-def*)
  then obtain *fi* **where**
   ∀*i*. ∃α*i*. *Fi i* = (λ*r*. (*fi i r*, α*i* (*fi i r*) *r*)) ∧ *fi i* ∈ *real-borel* $\rightarrow_M$ *count-space I*

$\wedge$ ($\forall$ *j*. (*j* $\in$ *range* (*fi i*) $\vee$ *qbs-space* (*X j*) $\neq$ {}) $\longrightarrow$ *αi j* $\in$ *qbs-Mx* (*X j*))
by(*fastforce intro*!: *choice*)
then obtain *αi* **where**
$\forall$ *i*. *Fi i* = (λ*r*. (*fi i r*, *αi i* (*fi i r*) *r*)) $\wedge$ *fi i* $\in$ *real-borel* $\to_M$ *count-space I* $\wedge$ ($\forall$ *j*. (*j* $\in$ *range* (*fi i*) $\vee$ *qbs-space* (*X j*) $\neq$ {}) $\longrightarrow$ *αi i j* $\in$ *qbs-Mx* (*X j*))
by(*fastforce intro*!: *choice*)
then have *hf*:
$\bigwedge$*i*. *Fi i* = (λ*r*. (*fi i r*, *αi i* (*fi i r*) *r*)) $\bigwedge$*i*. *fi i* $\in$ *real-borel* $\to_M$ *count-space I* $\bigwedge$*i j*. *j* $\in$ *range* (*fi i*) $\Longrightarrow$ *αi i j* $\in$ *qbs-Mx* (*X j*) $\bigwedge$*i j*. *qbs-space* (*X j*) $\neq$ {} $\Longrightarrow$ *αi i j* $\in$ *qbs-Mx* (*X j*)
by *auto*

define *f′* **where** *f′* ≡ (λ*r*. *fi* (*P r*) *r*)
define *α′* **where** *α′* ≡ (λ*i r*. *αi* (*P r*) *i r*)
have *1*:(λ*r*. *Fi* (*P r*) *r*) = (λ*r*. (*f′ r*, *α′* (*f′ r*) *r*))
by(*simp add*: *α′-def f′-def hf*)
have *f′* $\in$ *real-borel* $\to_M$ *count-space I*
proof −
note [*measurable*] = *separate-measurable*[*OF h*(*1*)]
have (λ(*n*,*r*). *fi n r*) $\in$ *count-space UNIV* $\bigotimes_M$ *real-borel* $\to_M$ *count-space I*
by(*auto intro*!: *measurable-pair-measure-countable1 simp*: *hf*)
hence [*measurable*]:(λ(*n*,*r*). *fi n r*) $\in$ *nat-borel* $\bigotimes_M$ *real-borel* $\to_M$ *count-space I*
using *measurable-cong-sets*[*OF sets-pair-measure-cong*[*OF sets-borel-eq-count-space*],*of real-borel real-borel*]
by *auto*
thus *?thesis*
using *measurable-comp*[*of* λ*r*. (*P r*, *r*) - - (λ(*n*,*r*). *fi n r*)]
by(*simp add*: *f′-def*)
qed
moreover have $\bigwedge$*i*. *i* $\in$ *range f′* $\Longrightarrow$ *α′ 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*. *αi j i* $\in$ *qbs-Mx* (*X i*)
by(*simp add*: *hf*(*4*))
then show *α′ i* $\in$ *qbs-Mx* (*X i*)
by(*auto simp*: *α′-def h*(*1*) *intro*!: *qbs-closed3-dest*[*of P* λ*j*. *αi j i*])
qed
ultimately show (λ*r*. *Fi* (*P r*) *r*) $\in$ *coprod-qbs-Mx I X*
by(*auto intro*!: *coprod-qbs-MxI simp*: *1*)
qed

lemma *coprod-qbs-correct*: *Rep-quasi-borel* (*coprod-qbs I X*) = (*SIGMA i*:*I*. *qbs-space* (*X i*), *coprod-qbs-Mx I X*)
unfolding *coprod-qbs-def*
using *is-quasi-borel-intro*[*OF coprod-qbs-f coprod-qbs-closed1 coprod-qbs-closed2 coprod-qbs-closed3*]
by(*fastforce intro*!: *Abs-quasi-borel-inverse*)

lemma *coproduct-qbs-space*[*simp*]: *qbs-space* (*coprod-qbs I X*) = (*SIGMA i*:*I*. *qbs-space* (*X i*))
by(*simp add*: *coprod-qbs-correct qbs-space-def*)

lemma *coproduct-qbs-Mx*[*simp*]: *qbs-Mx* (*coprod-qbs I X*) = *coprod-qbs-Mx I X*
by(*simp add*: *coprod-qbs-correct qbs-Mx-def*)

lemma *ini-morphism*:
assumes $j \in I$
shows $(\lambda x.\ (j,x)) \in X\ j \rightarrow_Q (\amalg_Q\ i{\in}I.\ X\ i)$
by(*fastforce intro*!: *qbs-morphismI exI*[**where** $x=\lambda r.\ j$] *simp*: *coprod-qbs-Mx-def comp-def assms*)

lemma *coprod-qbs-canonical1*:
assumes *countable I*
and $\bigwedge i.\ i \in I \Longrightarrow f\ i \in X\ i \rightarrow_Q Y$
shows $(\lambda(i,x).\ f\ i\ x) \in (\amalg_Q\ i\ {\in}I.\ X\ i) \rightarrow_Q Y$
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* (*coprod-qbs 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*]:$g \in real\text{-}borel \rightarrow_M count\text{-}space\ I$
by(*fastforce simp*: *coprod-qbs-Mx-def*)
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-morphismE*(*3*)[*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-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 *coprod-qbs-canonical1'*:
assumes *countable I*
and $\bigwedge i.\ i \in I \Longrightarrow (\lambda x.\ f\ (i,x)) \in X\ i \rightarrow_Q Y$
shows $f \in (\amalg_Q\ i\ {\in}I.\ X\ i) \rightarrow_Q Y$

using *coprod-qbs-canonical1*[**where** *f*=*curry f*] *assms* **by**(*auto simp*: *curry-def*)

$\coprod_{i=0,1} X_i \cong X_1 + X_2$.

lemma *coproduct-binary-coproduct*:
$\exists f\ g.\ f \in (\Pi_Q\ i{\in}UNIV.\ if\ i\ then\ X\ else\ Y) \rightarrow_Q X <\!+\!>_Q Y \wedge g \in X <\!+\!>_Q Y \rightarrow_Q (\Pi_Q\ i{\in}UNIV.\ if\ i\ then\ X\ else\ Y) \wedge$
$g \circ f = id \wedge f \circ g = id$
proof(*auto intro*!: *exI*[**where** $x=\lambda(b,z).$ *if b then Inl z else Inr z*] *exI*[**where** x=*case-sum* ($\lambda z.$ (*True*,z)) ($\lambda z.$ (*False*,z))])
show ($\lambda(b, z).$ *if b then Inl z else Inr z*) $\in (\Pi_Q\ i{\in}UNIV.\ if\ i\ then\ X\ else\ Y) \rightarrow_Q X <\!+\!>_Q Y$
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* (Π_Q $i{\in}UNIV$. *if i then X else Y*)
then obtain f β **where** *hf*:
$\alpha = (\lambda r.\ (f\ r,\ \beta\ (f\ r)\ r))$ $f \in$ *real-borel* $\rightarrow_M$ *count-space UNIV* $\bigwedge i.\ i \in range\ f \Longrightarrow \beta\ i \in$ *qbs-Mx* (*if i then X else Y*)
by(*auto simp*: *coprod-qbs-Mx-def*)
consider *range f* = {*True*} | *range f* = {*False*} | *range f* = {*True*,*False*}
by *auto*
thus ($\lambda(b, z).$ *if b then Inl z else Inr z*) $\circ\ \alpha \in$ *qbs-Mx* ($X <\!+\!>_Q Y$)
proof *cases*
case *1*
then have $\bigwedge r.\ f\ r = True$
by *auto*
then show *?thesis*
using *hf*(*3*)
by(*auto intro*!: *bexI*[**where** *x*={}] *bexI*[**where** $x=\beta$ *True*] *simp*: *copair-qbs-Mx-def split-beta′ comp-def hf*(*1*))
next
case *2*
then have $\bigwedge r.\ f\ r = False$
by *auto*
then show *?thesis*
using *hf*(*3*)
by(*auto intro*!: *bexI*[**where** *x*=*UNIV*] *bexI*[**where** $x=\beta$ *False*] *simp*: *copair-qbs-Mx-def split-beta′ comp-def hf*(*1*))
next
case *3*
then have *4*:*f* −‘ {*True*} $\in$ *sets real-borel*
using *measurable-sets*[*OF hf*(*2*)] **by** *simp*
have *5*:*f* −‘ {*True*} $\neq$ {} $\wedge$ *f* −‘ {*True*} $\neq$ *UNIV*
using *3*
by (*metis empty-iff imageE insertCI vimage-singleton-eq*)
have *6*:β *True* $\in$ *qbs-Mx X* β *False* $\in$ *qbs-Mx Y*
using *hf*(*3*)[*of True*] *hf*(*3*)[*of False*] **by**(*auto simp*: *3*)
show *?thesis*
apply(*simp add*: *copair-qbs-Mx-def*)
apply(*intro bexI*[*OF - 4*])

apply(*simp add*: *5*)
apply(*intro bexI*[*OF* - *6*(*1*)] *bexI*[*OF* - *6*(*2*)])
apply(*auto simp add*: *hf*(*1*) *comp-def*)
done
qed
qed
next
show *case-sum* (*Pair True*) (*Pair False*) $\in$ *X* <+>$_Q$ *Y* $\to_Q$ (Π_Q *i*$\in$*UNIV*. *if i then X else Y*)
proof(*rule qbs-morphismI*)
fix α
assume $\alpha \in$ *qbs-Mx* (*X* <+>$_Q$ *Y*)
then obtain *S* **where** *hs*:
S $\in$ *sets real-borel* *S* = {} $\longrightarrow$ ($\exists$ $\alpha 1\in$ *qbs-Mx X*. α = (λr. *Inl* ($\alpha 1$ *r*))) *S* = *UNIV* $\longrightarrow$ ($\exists$ $\alpha 2\in$ *qbs-Mx Y*. α = (λr. *Inr* ($\alpha 2$ *r*)))
(*S* $\neq$ {} $\wedge$ *S* $\neq$ *UNIV*) $\longrightarrow$ ($\exists$ $\alpha 1\in$ *qbs-Mx X*. $\exists$ $\alpha 2\in$ *qbs-Mx Y*. α = (λr::*real*. (*if* (*r* $\in$ *S*) *then Inl* ($\alpha 1$ *r*) *else Inr* ($\alpha 2$ *r*))))
by(*auto simp*: *copair-qbs-Mx-def*)
consider *S* = {} | *S* = *UNIV* | *S* $\neq$ {} $\wedge$ *S* $\neq$ *UNIV* **by** *auto*
thus *case-sum* (*Pair True*) (*Pair False*) $\circ$ α $\in$ *qbs-Mx* (Π_Q *i*$\in$*UNIV*. *if i then X else Y*)
proof *cases*
case *1*
then obtain $\alpha 1$ **where** *ha*:
$\alpha 1\in$ *qbs-Mx X* α = (λr. *Inl* ($\alpha 1$ *r*))
using *hs*(*2*) **by** *auto*
hence *case-sum* (*Pair True*) (*Pair False*) $\circ$ α = (λr. (*True*, $\alpha 1$ *r*))
by *auto*
thus *?thesis*
by(*auto intro*!: *coprod-qbs-MxI simp*: *ha*)
next
case *2*
then obtain $\alpha 2$ **where** *ha*:
$\alpha 2\in$ *qbs-Mx Y* α = (λr. *Inr* ($\alpha 2$ *r*))
using *hs*(*3*) **by** *auto*
hence *case-sum* (*Pair True*) (*Pair False*) $\circ$ α = (λr. (*False*, $\alpha 2$ *r*))
by *auto*
thus *?thesis*
by(*auto intro*!: *coprod-qbs-MxI simp*: *ha*)
next
case *3*
then obtain $\alpha 1$ $\alpha 2$ **where** *ha*:
$\alpha 1\in$ *qbs-Mx X* $\alpha 2\in$ *qbs-Mx Y* α = (λr. (*if* (*r* $\in$ *S*) *then Inl* ($\alpha 1$ *r*) *else Inr* ($\alpha 2$ *r*)))
using *hs*(*4*) **by** *auto*
define *f* :: *real* $\Rightarrow$ *bool* **where** *f* $\equiv$ (λr. *r* $\in$ *S*)
define α' **where** α' $\equiv$ (λi. *if i then* $\alpha 1$ *else* $\alpha 2$)
have *case-sum* (*Pair True*) (*Pair False*) $\circ$ α = (λr. (*f r*, α' (*f r*) *r*))
by(*auto simp*: *f-def* α'*-def* *ha*(*3*))

thus *?thesis*
  using *hs(1)*
  by(*auto intro*!: *coprod-qbs-MxI simp*: *ha* α'*-def f-def*)
qed
qed
next
show (λ(*b*, *z*). *if b then Inl z else Inr z*) $\circ$ *case-sum* (*Pair True*) (*Pair False*) = *id*
  by (*auto simp add*: *sum.case-eq-if*)
qed

2.4.3 Lists

abbreviation *list-of X* $\equiv \Pi_Q$ *n*$\in$(*UNIV* :: *nat set*). (Π_Q *i*$\in${..<*n*}. *X*)
abbreviation *list-nil* :: *nat* $\times$ (*nat* $\Rightarrow$ *'a*) **where**
list-nil $\equiv$ (*0*, λ*n*. *undefined*)
abbreviation *list-cons* :: [*'a*, *nat* $\times$ (*nat* $\Rightarrow$ *'a*)] $\Rightarrow$ *nat* $\times$ (*nat* $\Rightarrow$ *'a*) **where**
list-cons x l $\equiv$ (*Suc* (*fst l*), (λ*n*. *if n* = *0 then x else* (*snd l*) (*n* − *1*)))

definition *list-head* :: *nat* $\times$ (*nat* $\Rightarrow$ *'a*) $\Rightarrow$ *'a* **where**
list-head l = *snd l 0*
definition *list-tail* :: *nat* $\times$ (*nat* $\Rightarrow$ *'a*) $\Rightarrow$ *nat* $\times$ (*nat* $\Rightarrow$ *'a*) **where**
list-tail l = (*fst l* − *1*, λ*m*. (*snd l*) (*Suc m*))

lemma *list-simp1*:
list-nil $\neq$ *list-cons x l*
by *simp*

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*
thus *a* = *b*
  by(*simp add*: *assms*)
next
have *fst al* = *fst bl*
  using *assms* **by** *simp*
moreover have *snd al* = *snd bl*
proof
  fix *n*
  have *snd al n* = *snd* (*list-cons a al*) (*Suc n*)
    by *simp*
  also have ... = *snd* (*list-cons b bl*) (*Suc n*)
    by (*simp add*: *assms*)
  also have ... = *snd bl n*

```
      by simp
    finally show snd al n = snd bl n .
  qed
  ultimately show al = bl
    by (simp add: prod.expand)
qed

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

lemma list-simp4:
  assumes l ∈ qbs-space (list-of X)
  shows list-tail (list-cons a l) = l
  using assms by(simp-all add: list-tail-def)

lemma list-decomp1:
  assumes l ∈ qbs-space (list-of X)
  shows l = list-nil ∨
        (∃ a l'. a ∈ qbs-space X ∧ l' ∈ qbs-space (list-of 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
  next
    case hn:(Suc n')
    define f' where f' ≡ λm. f (Suc m)
    have l = list-cons (f 0) (n',f')
    proof(simp add: hl hn, standard)
      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
    moreover have (n', f') ∈ qbs-space (list-of X)
    proof(simp,rule PiE-I)
      show ⋀x. x ∈ {..<n'} ⟹ f' x ∈ qbs-space X
        using assms hl hn by(fastforce simp: f'-def)
    next
      fix x
      assume 1:x ∉ {..<n'}
      thus f' x = undefined
        using hl assms hn by(auto simp: f'-def)
    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*))])
  qed
qed

lemma *list-simp5*:
  assumes $l \in$ *qbs-space* (*list-of X*)
      and $l \neq$ *list-nil*
    shows $l =$ *list-cons* (*list-head l*) (*list-tail l*)
proof −
  obtain $a\ l'$ **where** *hl*:
  $a \in$ *qbs-space X* $l' \in$ *qbs-space* (*list-of X*) $l =$ *list-cons* $a\ l'$
    using *list-decomp1*[*OF assms*(*1*)] *assms*(*2*) **by** *blast*
  hence *list-head* $l = a$ *list-tail* $l = l'$
    using *list-simp3 list-simp4* **by** *auto*
  thus *?thesis*
    using *hl*(*3*) *list-simp2* **by** *auto*
qed

lemma *list-simp6*:
 list-nil $\in$ *qbs-space* (*list-of X*)
  by *simp*

lemma *list-simp7*:
  assumes $a \in$ *qbs-space X*
      and $l \in$ *qbs-space* (*list-of X*)
    shows *list-cons a l* $\in$ *qbs-space* (*list-of X*)
  using *assms* **by**(*fastforce simp*: *PiE-def extensional-def*)

lemma *list-destruct-rule*:
  assumes $l \in$ *qbs-space* (*list-of X*)
          P list-nil
      and $\bigwedge a\ l'.\ a \in$ *qbs-space X* $\Longrightarrow l' \in$ *qbs-space* (*list-of X*) $\Longrightarrow$ *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 $l \in$ *qbs-space* (*list-of X*)
          P list-nil
      and $\bigwedge a\ l'.\ a \in$ *qbs-space X* $\Longrightarrow l' \in$ *qbs-space* (*list-of X*) $\Longrightarrow P\ l' \Longrightarrow P$ (*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*(*1*,*2*) **by** *simp*
next
case *ih*:(*Suc n*)
then obtain *a l′* **where** *hl*:
a ∈ *qbs-space X l′* ∈ *qbs-space* (*list-of X*) *l* = *list-cons a l′*
using *list-decomp1* **by** *blast*
have *P l′*
using *ih hl*(*3*)
by(*auto intro*!: *ih*(*1*)[*OF - hl*(*2*),*of snd l′*])
from *assms*(*3*)[*OF hl*(*1*,*2*) *this*]
show *?case*
by(*simp add*: *hl*(*3*))
qed
qed

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* (λ*n*. *f* (*Suc n*))

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

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

lemma *to-list-simp2*:
assumes *l* ∈ *qbs-space* (*list-of X*)
shows *to-list* (*list-cons a l*) = *a* # *to-list l*
using *assms* **by**(*auto simp*:*PiE-def to-list-def*)

lemma *from-list-length*:
fst (*from-list l*) = *length l*
by(*induction l*, *simp-all*)

lemma *from-list-in-list-of*:
assumes *set l* ⊆ *qbs-space X*
shows *from-list l* ∈ *qbs-space* (*list-of X*)
using *assms* **by**(*induction l*) (*auto simp*: *PiE-def extensional-def Pi-def*)

lemma *from-list-in-list-of′*:
shows *from-list l* ∈ *qbs-space* (*list-of* (*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* (*list-of 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*,*simp add*: *to-list-simp2*[*OF from-list-in-list-of'*])

lemma *to-list-from-list-ident*:
assumes *l* ∈ *qbs-space* (*list-of X*)
shows (*from-list* ∘ *to-list*) *l* = *l*
proof(*rule list-induct-rule*[*OF assms*])
fix *a l'*
assume *h*: *l'* ∈ *qbs-space* (*list-of 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* (*list-of 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*])

end

2.5 Function Spaces

theory *Exponent-QuasiBorel*
imports *CoProduct-QuasiBorel*
begin

2.5.1 Function Spaces

definition *exp-qbs-Mx* :: [*'a quasi-borel*, *'b quasi-borel*] ⇒ (*real* ⇒ *'a* => *'b*) *set*
where
exp-qbs-Mx X Y ≡ {*g* :: *real* ⇒ *'a* ⇒ *'b*. *case-prod g* ∈ $\mathbb{R}_Q \bigotimes_Q X \rightarrow_Q Y$}

definition *exp-qbs* :: [*'a quasi-borel*, *'b quasi-borel*] ⇒ (*'a* ⇒ *'b*) *quasi-borel* (**infixr** ‹$\Rightarrow_Q$› *61*) **where**
$X \Rightarrow_Q Y$ ≡ *Abs-quasi-borel* ($X \rightarrow_Q Y$, *exp-qbs-Mx X Y*)

lemma *exp-qbs-f*[*simp*]: *exp-qbs-Mx X Y* ⊆ *UNIV* → (*X* :: *'a quasi-borel*) $\rightarrow_Q$ (*Y* :: *'b quasi-borel*)
proof(*auto intro*!: *qbs-morphismI*)
  fix *f α r*
  assume *h*:*f* ∈ *exp-qbs-Mx X Y*
        α ∈ *qbs-Mx X*
  have *f r* ∘ *α* = (*λy*. *case-prod f* (*r*,*y*)) ∘ *α*
    by *auto*
  also have ... ∈ *qbs-Mx Y*
    using *qbs-morphism-Pair1'*[*of r* $\mathbb{R}_Q$ *case-prod f X Y*] *h*
    by(*auto simp*: *exp-qbs-Mx-def*)
  finally show *f r* ∘ *α* ∈ *qbs-Mx Y* **.**
qed

lemma *exp-qbs-closed1*: *qbs-closed1* (*exp-qbs-Mx X Y*)
proof(*rule qbs-closed1I*)
  fix *a*
  fix *f*
  assume *h*:*a* ∈ *exp-qbs-Mx X Y*
        f ∈ *real-borel* $\rightarrow_M$ *real-borel*
  have *a* ∘ *f* = (*λr y*. *case-prod a* (*f r*,*y*)) **by** *auto*
  moreover have *case-prod* ... ∈ $\mathbb{R}_Q \bigotimes_Q X \rightarrow_Q Y$
  proof −
    have *case-prod* (*λr y*. *case-prod a* (*f r*,*y*)) = *case-prod a* ∘ *map-prod f id*
      by *auto*
    also have ... ∈ $\mathbb{R}_Q \bigotimes_Q X \rightarrow_Q Y$
      using *h*
    by(*auto intro*!: *qbs-morphism-comp qbs-morphism-map-prod simp*: *exp-qbs-Mx-def*)
    finally show *?thesis* **.**
  qed
  ultimately show *a* ∘ *f* ∈ *exp-qbs-Mx X Y*
    by (*simp add*: *exp-qbs-Mx-def*)
qed

lemma *exp-qbs-closed2*: *qbs-closed2* ($X \rightarrow_Q Y$) (*exp-qbs-Mx X Y*)
  by(*auto intro*!: *qbs-closed2I qbs-morphism-snd'' simp*: *exp-qbs-Mx-def split-beta'*)

lemma *exp-qbs-closed3*:*qbs-closed3* (*exp-qbs-Mx X Y*)
proof(*rule qbs-closed3I*)

fix P :: *real* $\Rightarrow$ *nat*
fix *Fi* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ -
assume *h*:$\bigwedge i$. P −‘ $\{i\}$ $\in$ *sets real-borel*
$\bigwedge i$. *Fi i* $\in$ *exp-qbs-Mx X Y*
show $(\lambda r$. *Fi* $(P\ r)\ r)$ $\in$ *exp-qbs-Mx X Y*
unfolding *exp-qbs-Mx-def*
proof(*auto intro*!: *qbs-morphismI*)
fix α β
assume h':α $\in$ *pair-qbs-Mx* $\mathbb{R}_Q$ *X*
have *1*:$\bigwedge i$. $(\lambda(r,x)$. *Fi i r x*$)$ $\circ$ α $\in$ *qbs-Mx Y*
using *qbs-morphismE*(*3*)[*OF h*(*2*)[*simplified exp-qbs-Mx-def*,*simplified*]] h'
by(*simp add*: *exp-qbs-Mx-def*)
have *2*:$\bigwedge i$. $(P \circ (\lambda r.\ fst\ (\alpha\ r)))$ −‘ $\{i\}$ $\in$ *sets real-borel*
using *separate-measurable*[*OF h*(*1*)] h'
by(*auto intro*!: *measurable-separate simp*: *pair-qbs-Mx-def comp-def*)
show $(\lambda(r,\ y)$. *Fi* $(P\ r)\ r\ y)$ $\circ$ α $\in$ *qbs-Mx Y*
using *qbs-closed3-dest*[*OF 2*,*of* λi. $(\lambda(r,x)$. *Fi i r x*$)$ $\circ$ α,*OF 1*]
by(*simp add*: *comp-def split-beta'*)
qed
qed

lemma *exp-qbs-correct*: *Rep-quasi-borel* (*exp-qbs X Y*) = ($X \to_Q Y$, *exp-qbs-Mx X Y*)
unfolding *exp-qbs-def*
by(*auto intro*!: *Abs-quasi-borel-inverse exp-qbs-f simp*: *exp-qbs-closed1 exp-qbs-closed2 exp-qbs-closed3*)

lemma *exp-qbs-space*[*simp*]: *qbs-space* (*exp-qbs X Y*) = $X \to_Q Y$
by(*simp add*: *qbs-space-def exp-qbs-correct*)

lemma *exp-qbs-Mx*[*simp*]: *qbs-Mx* (*exp-qbs X Y*) = *exp-qbs-Mx X Y*
by(*simp add*: *qbs-Mx-def exp-qbs-correct*)

lemma *qbs-exp-morphismI*:
assumes $\bigwedge \alpha\ \beta$. $\alpha \in$ *qbs-Mx X* $\Longrightarrow$
$\beta \in$ *pair-qbs-Mx real-quasi-borel Y* $\Longrightarrow$
$(\lambda(r,x).\ (f \circ \alpha)\ r\ x) \circ \beta \in$ *qbs-Mx Z*
shows $f \in X \to_Q$ *exp-qbs Y Z*
using *assms*
by(*auto intro*!: *qbs-morphismI simp*: *exp-qbs-Mx-def comp-def*)

definition *qbs-eval* :: $(('a \Rightarrow 'b) \times 'a) \Rightarrow 'b$ **where**
qbs-eval a $\equiv$ (*fst a*) (*snd a*)

lemma *qbs-eval-morphism*:
qbs-eval $\in$ (*exp-qbs X Y*) $\bigotimes_Q X \to_Q Y$
proof(*rule qbs-morphismI*,*simp*)

fix *f*
assume $f \in$ *pair-qbs-Mx* (*exp-qbs X Y*) *X*
let *?f1* = *fst* ∘ *f*
let *?f2* = *snd* ∘ *f*
define *g* :: *real* ⇒ *real* × -
where *g* ≡ λ*r*.(*r*,*?f2 r*)
have *g* ∈ *qbs-Mx* (*real-quasi-borel* $\bigotimes_Q$ *X*)
proof(*auto simp add*: *pair-qbs-Mx-def*)
have *fst* ∘ *g* = *id* **by**(*auto simp add*: *g-def comp-def*)
thus *fst* ∘ *g* ∈ *real-borel* $\to_M$ *real-borel* **by**(*auto simp add*: *measurable-ident*)
next
have *snd* ∘ *g* = *?f2* **by**(*auto simp add*: *g-def*)
thus *snd* ∘ *g* ∈ *qbs-Mx X*
using ‹*f* ∈ *pair-qbs-Mx* (*exp-qbs X Y*) *X*› *pair-qbs-Mx-def* **by** *auto*
qed
moreover have *?f1* ∈ *exp-qbs-Mx X Y*
using ‹*f* ∈ *pair-qbs-Mx* (*exp-qbs X Y*) *X*›
by(*simp add*: *pair-qbs-Mx-def*)
ultimately have (λ(*r*,*x*). (*?f1 r x*)) ∘ *g* ∈ *qbs-Mx Y*
by (*auto simp add*: *exp-qbs-Mx-def qbs-morphism-def*)
(*metis* (*mono-tags*, *lifting*) *case-prod-conv comp-apply cond-case-prod-eta*)
moreover have (λ(*r*,*x*). (*?f1 r x*)) ∘ *g* = *qbs-eval* ∘ *f*
by(*auto simp add*: *case-prod-unfold g-def qbs-eval-def*)
ultimately show *qbs-eval* ∘ *f* ∈ *qbs-Mx Y* **by** *simp*
qed

lemma *curry-morphism*:
curry ∈ *exp-qbs* (*X* $\bigotimes_Q$ *Y*) *Z* $\to_Q$ *exp-qbs X* (*exp-qbs Y Z*)
proof(*auto intro*!: *qbs-morphismI simp*: *exp-qbs-Mx-def*)
fix *k* α α′
assume *h*:(λ(*r*, *xy*). *k r xy*) ∈ $\mathbb{R}_Q \bigotimes_Q X \bigotimes_Q Y \to_Q Z$
α ∈ *pair-qbs-Mx* $\mathbb{R}_Q$ *X*
α′ ∈ *pair-qbs-Mx* $\mathbb{R}_Q$ *Y*
define β **where**
β ≡ (λ*r*. (*fst* (α (*fst* (α′ *r*))),(*snd* (α (*fst* (α′ *r*))), *snd* (α′ *r*))))
have (λ(*x*, *y*). ((λ(*x*, *y*). (*curry* ∘ *k*) *x y*) ∘ α) *x y*) ∘ α′ = (λ(*r*, *xy*). *k r xy*) ∘ β
by(*simp add*: *curry-def split-beta′ comp-def β-def*)
also have ... ∈ *qbs-Mx Z*
proof −
have β ∈ *qbs-Mx* ($\mathbb{R}_Q \bigotimes_Q X \bigotimes_Q Y$)
using *h*(*2*,*3*) *qbs-closed1-dest*[*of* - - (λ*x*. *fst* (α′ *x*))]
by(*auto simp*: *pair-qbs-Mx-def β-def comp-def*)
thus *?thesis*
using *h* **by** *auto*
qed
finally show (λ(*x*, *y*). ((λ(*x*, *y*). (*curry* ∘ *k*) *x y*) ∘ α) *x y*) ∘ α′ ∈ *qbs-Mx Z* **.**
qed

lemma *curry-preserves-morphisms*:

assumes $f \in X \bigotimes_Q Y \to_Q Z$
shows *curry* $f \in X \to_Q$ *exp-qbs* Y Z
by(*rule qbs-morphismE*(*2*)[*OF curry-morphism*,*simplified*,*OF assms*])

lemma *uncurry-morphism*:
case-prod $\in$ *exp-qbs* X (*exp-qbs* Y Z) $\to_Q$ *exp-qbs* ($X \bigotimes_Q Y$) Z
proof(*auto intro*!: *qbs-morphismI simp*: *exp-qbs-Mx-def*)
fix k α
assume h:($\lambda(x, y).\ k\ x\ y) \in \mathbb{R}_Q \bigotimes_Q X \to_Q$ *exp-qbs* Y Z
$\alpha \in$ *pair-qbs-Mx* $\mathbb{R}_Q$ ($X \bigotimes_Q Y$)
have ($\lambda(x, y)$. (*case-prod* $\circ$ k) x y) $\circ$ α = ($\lambda(r,y)$. k (*fst* (α r)) (*fst* (*snd* (α r))) y) $\circ$ (λr. (r,*snd* (*snd* (α r))))
by(*simp add*: *split-beta*′ *comp-def*)
also have ... $\in$ *qbs-Mx* Z
proof(*rule qbs-morphismE*(*3*)[**where** X=$\mathbb{R}_Q \bigotimes_Q Y$])
have (λr. k (*fst* (α r)) (*fst* (*snd* (α r)))) = ($\lambda(x, y)$. k x y) $\circ$ (λr. (*fst* (α r),*fst* (*snd* (α r))))
by *auto*
also have ... $\in$ *qbs-Mx* (*exp-qbs* Y Z)
apply(*rule qbs-morphismE*(*3*)[**where** X=$\mathbb{R}_Q \bigotimes_Q X$])
using h(*2*) **by**(*auto simp*: h(*1*) *pair-qbs-Mx-def comp-def*)
finally show ($\lambda(r, y)$. k (*fst* (α r)) (*fst* (*snd* (α r))) y) $\in \mathbb{R}_Q \bigotimes_Q Y \to_Q Z$
by(*simp add*: *exp-qbs-Mx-def*)
next
show (λr. (r, *snd* (*snd* (α r)))) $\in$ *qbs-Mx* ($\mathbb{R}_Q \bigotimes_Q Y$)
using h(*2*) **by**(*simp add*: *pair-qbs-Mx-def comp-def*)
qed
finally show ($\lambda(x, y)$. (*case-prod* $\circ$ k) x y) $\circ$ $\alpha \in$ *qbs-Mx* Z **.**
qed

lemma *uncurry-preserves-morphisms*:
assumes $f \in X \to_Q$ *exp-qbs* Y Z
shows *case-prod* $f \in X \bigotimes_Q Y \to_Q Z$
by(*rule qbs-morphismE*(*2*)[*OF uncurry-morphism*,*simplified*,*OF assms*])

lemma *arg-swap-morphism*:
assumes $f \in X \to_Q$ *exp-qbs* Y Z
shows ($\lambda y\ x.\ f\ x\ y) \in Y \to_Q$ *exp-qbs* X Z
using *curry-preserves-morphisms*[*OF qbs-morphism-pair-swap*[*OF uncurry-preserves-morphisms*[*OF assms*]]]
by *simp*

lemma *exp-qbs-comp-morphism*:
assumes $f \in W \to_Q$ *exp-qbs* X Y
and $g \in W \to_Q$ *exp-qbs* Y Z
shows (λw. g w $\circ$ f w) $\in W \to_Q$ *exp-qbs* X Z
proof(*rule qbs-exp-morphismI*)
fix α β
assume h: $\alpha \in$ *qbs-Mx* W

β ∈ *pair-qbs-Mx* $\mathbb{R}_Q$ *X*
have (λ(*r*, *x*). ((λ*w*. *g w* ∘ *f w*) ∘ α) *r x*) ∘ β= *case-prod g* ∘ (λ*r*. ((α ∘ (*fst* ∘ β)) *r*, *case-prod f* ((α ∘ (*fst* ∘ β)) *r*, (*snd* ∘ β) *r*)))
by(*simp add*: *split-beta′ comp-def*)
also have ... ∈ *qbs-Mx Z*
proof −
have (λ*r*. ((α ∘ (*fst* ∘ β)) *r*, *case-prod f* ((α ∘ (*fst* ∘ β)) *r*, (*snd* ∘ β) *r*))) ∈ *qbs-Mx* (*W* $\bigotimes_Q$ *Y*)
proof(*auto simp add*: *pair-qbs-Mx-def*)
have *fst* ∘ (λ*r*. (α (*fst* (β *r*)), *f* (α (*fst* (β *r*))) (*snd* (β *r*)))) = α ∘ (*fst* ∘ β)
by (*simp add*: *comp-def*)
also have ... ∈ *qbs-Mx W*
using *qbs-decomp*[*of W*] *h*
by(*simp add*: *pair-qbs-Mx-def qbs-closed1-def*)
finally show *fst* ∘ (λ*r*. (α (*fst* (β *r*)), *f* (α (*fst* (β *r*))) (*snd* (β *r*)))) ∈ *qbs-Mx W* **.**
next
have [*simp*]:*snd* ∘ (λ*r*. (α (*fst* (β *r*)), *f* (α (*fst* (β *r*))) (*snd* (β *r*)))) = *case-prod f* ∘ (λ*r*. ((α ∘ (*fst* ∘ β)) *r*, (*snd* ∘ β) *r*))
by(*simp add*: *comp-def*)
have (λ*r*. ((α ∘ (*fst* ∘ β)) *r*, (*snd* ∘ β) *r*)) ∈ *qbs-Mx* (*W* $\bigotimes_Q$ *X*)
proof(*auto simp add*: *pair-qbs-Mx-def*)
have *fst* ∘ (λ*r*. (α (*fst* (β *r*)), *snd* (β *r*)))= α ∘ (*fst* ∘ β)
by (*simp add*: *comp-def*)
also have ... ∈ *qbs-Mx W*
using *qbs-decomp*[*of W*] *h*
by(*simp add*: *pair-qbs-Mx-def qbs-closed1-def*)
finally show *fst* ∘ (λ*r*. (α (*fst* (β *r*)), *snd* (β *r*))) ∈ *qbs-Mx W* **.**
next
show *snd* ∘ (λ*r*. (α (*fst* (β *r*)), *snd* (β *r*))) ∈ *qbs-Mx X*
using *h*
by(*simp add*: *pair-qbs-Mx-def comp-def*)
qed
hence *case-prod f* ∘ (λ*r*. ((α ∘ (*fst* ∘ β)) *r*, (*snd* ∘ β) *r*)) ∈ *qbs-Mx Y*
using *uncurry-preserves-morphisms*[*OF assms*(*1*)] **by** *auto*
thus *snd* ∘ (λ*r*. (α (*fst* (β *r*)), *f* (α (*fst* (β *r*))) (*snd* (β *r*)))) ∈ *qbs-Mx Y*
by *simp*
qed
thus *?thesis*
using *uncurry-preserves-morphisms*[*OF assms*(*2*)]
by *auto*
qed
finally show (λ(*r*, *x*). ((λ*w*. *g w* ∘ *f w*) ∘ α) *r x*) ∘ β ∈ *qbs-Mx Z* **.**
qed

lemma *case-sum-morphism*:
case-prod case-sum ∈ *exp-qbs X Z* $\bigotimes_Q$ *exp-qbs Y Z* $\to_Q$ *exp-qbs* ($X <+>_Q Y$) *Z*
proof(*rule qbs-exp-morphismI*)

```
fix α β
assume h0:α ∈ qbs-Mx (exp-qbs X Z ⨂Q exp-qbs Y Z)
         β ∈ pair-qbs-Mx ℝQ (X <+>Q Y)
let ?α1 = fst ∘ α
let ?α2 = snd ∘ α
let ?β1 = fst ∘ β
let ?β2 = snd ∘ β
have h:?α1 ∈ exp-qbs-Mx X Z
       ?α2 ∈ exp-qbs-Mx Y Z
       ?β1 ∈ real-borel →M real-borel
       ?β2 ∈ copair-qbs-Mx X Y
  using h0 by (auto simp add: pair-qbs-Mx-def)
 hence ∃ S∈sets real-borel. (S = {} ⟶ (∃ α1∈qbs-Mx X. ?β2 = (λr. Inl (α1 r)))) ∧
                          (S = UNIV ⟶ (∃ α2∈qbs-Mx Y. ?β2 = (λr. Inr (α2 r)))) ∧
                         (S ≠ {} ∧ S ≠ UNIV ⟶
                       (∃ α1∈qbs-Mx X. ∃ α2∈qbs-Mx Y. ?β2 = (λr. if r ∈ S then Inl (α1 r) else Inr (α2 r))))
  by(simp add: copair-qbs-Mx-def)
then obtain S :: real set where hs:
  S∈sets real-borel ∧ (S = {} ⟶ (∃ α1∈qbs-Mx X. ?β2 = (λr. Inl (α1 r)))) ∧
                      (S = UNIV ⟶ (∃ α2∈qbs-Mx Y. ?β2 = (λr. Inr (α2 r)))) ∧
                       (S ≠ {} ∧ S ≠ UNIV ⟶
                        (∃ α1∈qbs-Mx X. ∃ α2∈qbs-Mx Y. ?β2 = (λr. if r ∈ S then Inl (α1 r) else Inr (α2 r))))
  by auto
show (λ(r, x). ((λ(x, y). case-sum x y) ∘ α) r x) ∘ β ∈ qbs-Mx Z
proof −
   have (λ(r, x). ((λ(x, y). case-sum x y) ∘ α) r x) ∘ β = (λr. case-sum (?α1 (?β1 r)) (?α2 (?β1 r)) (?β2 r))
       (is ?lhs = ?rhs)
    by(auto simp: split-beta' comp-def) (metis comp-apply)
  also have ... ∈ qbs-Mx Z
      (is ?f ∈ -)
  proof −
    consider S = {} | S = UNIV | S ≠ {} ∧ S ≠ UNIV by auto
    then show ?thesis
    proof cases
      case 1
      then obtain α1 where h1:
       α1∈qbs-Mx X ∧ ?β2 = (λr. Inl (α1 r))
        using hs by auto
     then have (λr. case-sum (?α1 (?β1 r)) (?α2 (?β1 r)) (?β2 r)) = (λr. ?α1 (?β1 r) (α1 r))
        by simp
      also have ... = case-prod ?α1 ∘ (λr. (?β1 r,α1 r))
        by auto
      also have ... ∈ ℝQ →Q Z
```

apply(*rule qbs-morphism-comp*[*of* - - $\mathbb{R}_Q \bigotimes_Q X$])
apply(*rule qbs-morphism-tuple*)
using *h*(*3*)
apply *blast*
using *qbs-Mx-is-morphisms h1*
apply *blast*
using *qbs-Mx-is-morphisms*[*of* $\mathbb{R}_Q \bigotimes_Q X$] *h*(*1*)
by (*simp add*: *exp-qbs-Mx-def*)
finally show *?thesis*
using *qbs-Mx-is-morphisms* **by** *auto*
next
case *2*
then obtain $\alpha 2$ **where** *h2*:
$\alpha 2 \in$*qbs-Mx Y* $\wedge$ $?\beta 2 = (\lambda r.\ Inr\ (\alpha 2\ r))$
using *hs* **by** *auto*
then have $(\lambda r.\ case\text{-}sum\ (?\alpha 1\ (?\beta 1\ r))\ (?\alpha 2\ (?\beta 1\ r))\ (?\beta 2\ r)) = (\lambda r.\ ?\alpha 2\ (?\beta 1\ r)\ (\alpha 2\ r))$
by *simp*
also have ... $=$ *case-prod* $?\alpha 2 \circ (\lambda r.\ (?\beta 1\ r,\alpha 2\ r))$
by *auto*
also have ... $\in \mathbb{R}_Q \to_Q Z$
apply(*rule qbs-morphism-comp*[*of* - - $\mathbb{R}_Q \bigotimes_Q Y$])
apply(*rule qbs-morphism-tuple*)
using *h*(*3*)
apply *blast*
using *qbs-Mx-is-morphisms h2*
apply *blast*
using *qbs-Mx-is-morphisms*[*of* $\mathbb{R}_Q \bigotimes_Q Y$] *h*(*2*)
by (*simp add*: *exp-qbs-Mx-def*)
finally show *?thesis*
using *qbs-Mx-is-morphisms* **by** *auto*
next
case *3*
then obtain $\alpha 1$ $\alpha 2$ **where** *h3*:
$\alpha 1 \in$*qbs-Mx X* $\wedge$ $\alpha 2 \in$*qbs-Mx Y* $\wedge$ $?\beta 2 = (\lambda r.\ if\ r \in S\ then\ Inl\ (\alpha 1\ r)\ else\ Inr\ (\alpha 2\ r))$
using *hs* **by** *auto*
define P :: *real* $\Rightarrow$ *nat*
where $P \equiv (\lambda r.\ if\ r \in S\ then\ 0\ else\ 1)$
define γ :: *nat* $\Rightarrow$ *real* $\Rightarrow$ -
where $\gamma \equiv (\lambda n\ r.\ if\ n = 0\ then\ ?\alpha 1\ (?\beta 1\ r)\ (\alpha 1\ r)\ else\ ?\alpha 2\ (?\beta 1\ r)\ (\alpha 2\ r))$
then have $(\lambda r.\ case\text{-}sum\ (?\alpha 1\ (?\beta 1\ r))\ (?\alpha 2\ (?\beta 1\ r))\ (?\beta 2\ r)) = (\lambda r.\ \gamma\ (P\ r)\ r)$
by(*auto simp add*: *P-def* γ*-def h3*)
also have ... $\in$ *qbs-Mx Z*
proof $-$
have $\forall i.\ P$ $-$‘ $\{i\} \in$ *sets real-borel*
using *hs borel-comp*[*of S*] **by**(*simp add*: *P-def*)

```
      moreover have∀ i. γ i ∈ qbs-Mx Z
      proof
        fix i :: nat
        consider i = 0 | i ≠ 0 by auto
        then show γ i ∈ qbs-Mx Z
        proof cases
          case 1
          then have γ i = (λr. ?α1 (?β1 r) (α1 r))
            by(simp add: γ-def)
          also have ... = case-prod ?α1 ∘ (λr. (?β1 r,α1 r))
            by auto
          also have ... ∈ ℝ_Q →_Q Z
            apply(rule qbs-morphism-comp[of - - ℝ_Q ⨂_Q X])
             apply(rule qbs-morphism-tuple)
            using h(3)
              apply blast
            using qbs-Mx-is-morphisms h3
             apply blast
            using qbs-Mx-is-morphisms[of ℝ_Q ⨂_Q X] h(1)
            by (simp add: exp-qbs-Mx-def)
          finally show ?thesis
            using qbs-Mx-is-morphisms by auto
        next
          case 2
          then have γ i = (λr. ?α2 (?β1 r) (α2 r))
            by(simp add: γ-def)
          also have ... = case-prod ?α2 ∘ (λr. (?β1 r,α2 r))
            by auto
          also have ... ∈ ℝ_Q →_Q Z
            apply(rule qbs-morphism-comp[of - - ℝ_Q ⨂_Q Y])
             apply(rule qbs-morphism-tuple)
            using h(3)
              apply blast
            using qbs-Mx-is-morphisms h3
             apply blast
            using qbs-Mx-is-morphisms[of ℝ_Q ⨂_Q Y] h(2)
            by (simp add: exp-qbs-Mx-def)
          finally show ?thesis
            using qbs-Mx-is-morphisms by auto
        qed
      qed
      ultimately show ?thesis
        using qbs-decomp[of Z]
        by(simp add: qbs-closed3-def)
    qed
    finally show ?thesis .
  qed
qed
finally show ?thesis .
```

qed
qed

lemma *not-qbs-morphism*:
Not $\in \mathbb{B}_Q \to_Q \mathbb{B}_Q$
by(*auto intro*!: *bool-qbs-morphism*)

lemma *or-qbs-morphism*:
$(\vee) \in \mathbb{B}_Q \to_Q$ *exp-qbs* $\mathbb{B}_Q$ $\mathbb{B}_Q$
by(*auto intro*!: *bool-qbs-morphism*)

lemma *and-qbs-morphism*:
$(\wedge) \in \mathbb{B}_Q \to_Q$ *exp-qbs* $\mathbb{B}_Q$ $\mathbb{B}_Q$
by(*auto intro*!: *bool-qbs-morphism*)

lemma *implies-qbs-morphism*:
$(\longrightarrow) \in \mathbb{B}_Q \to_Q \mathbb{B}_Q \Rightarrow_Q \mathbb{B}_Q$
by(*auto intro*!: *bool-qbs-morphism*)

lemma *less-nat-qbs-morphism*:
$(<) \in \mathbb{N}_Q \to_Q$ *exp-qbs* $\mathbb{N}_Q$ $\mathbb{B}_Q$
by(*auto intro*!: *nat-qbs-morphism*)

lemma *less-real-qbs-morphism*:
$(<) \in \mathbb{R}_Q \to_Q$ *exp-qbs* $\mathbb{R}_Q$ $\mathbb{B}_Q$
proof(*rule curry-preserves-morphisms*[**where** f=(λ(z :: *real* $\times$ *real*). *fst z* < *snd z*),*simplified curry-def*,*simplified*])
have (λz. *fst z* < *snd z*) $\in$ *real-borel* $\bigotimes_M$ *real-borel* $\to_M$ *bool-borel*
using *borel-measurable-pred-less*[*OF measurable-fst measurable-snd*,*simplified measurable-cong-sets*[*OF refl sets-borel-eq-count-space*[*symmetric*],*of borel* $\bigotimes_M$ *borel*]]
by *simp*
thus (λz. *fst z* < *snd z*) $\in \mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q \to_Q \mathbb{B}_Q$
by *auto*
qed

lemma *rec-list-morphism′*:
rec-list′ $\in$ *qbs-space* (*exp-qbs Y* (*exp-qbs* (*exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) (*exp-qbs* (*list-of X*) *Y*)))
apply(*simp*,*rule curry-preserves-morphisms*[**where** f=λyf. *rec-list′* (*fst yf*) (*snd yf*),*simplified curry-def*, *simplified*])
apply(*rule arg-swap-morphism*)
proof(*rule coprod-qbs-canonical1′*)
fix *n*
show ($\lambda x\ y$. *rec-list′* (*fst y*) (*snd y*) (n, x)) $\in$ (Π_Q $i \in \{..<n\}$. X) $\to_Q$ *exp-qbs* (Y $\bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) Y
proof(*induction n*)

case *0*
show *?case*
proof(*rule curry-preserves-morphisms*[*of* $(\lambda(x,y).$ *rec-list'* (*fst y*) (*snd y*) (*0*, *x*)), *simplified*],*rule qbs-morphismI*)
fix α
assume *h*:$\alpha \in$ *qbs-Mx* ((Π_Q *i*∈{..<*0*::*nat*}. *X*) $\bigotimes_Q$ *Y* $\bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)))
have $\bigwedge r.$ *fst* (α *r*) = ($\lambda n.$ *undefined*)
proof −
fix *r*
have $\bigwedge i.$ ($\lambda r.$ *fst* (α *r*) *i*) = ($\lambda r.$ *undefined*)
using *h* **by**(*auto simp*: *exp-qbs-Mx-def prod-qbs-Mx-def pair-qbs-Mx-def comp-def split-beta'*)
thus *fst* (α *r*) = ($\lambda n.$ *undefined*)
by(*fastforce dest*: *fun-cong*)
qed
hence ($\lambda(x, y).$ *rec-list'* (*fst y*) (*snd y*) (*0*, *x*)) $\circ\ \alpha$ = ($\lambda x.$ *fst* (*snd* (α *x*)))
by(*auto simp*: *rec-list'-simp1 comp-def split-beta'*)
also have ... $\in$ *qbs-Mx Y*
using *h* **by**(*auto simp*: *pair-qbs-Mx-def comp-def*)
finally show ($\lambda(x, y).$ *rec-list'* (*fst y*) (*snd y*) (*0*, *x*)) $\circ\ \alpha \in$ *qbs-Mx Y* **.**
qed
next
case *ih*:(*Suc n*)
show *?case*
proof(*rule qbs-morphismI*)
fix α
assume *h*:$\alpha \in$ *qbs-Mx* (Π_Q *i*∈{..<*Suc n*}. *X*)
define α' **where** $\alpha' \equiv$ ($\lambda r.$ *snd* (*list-tail* (*Suc n*, α *r*)))
define *a* **where** *a* $\equiv$ ($\lambda r.$ α *r 0*)
then have *ha*:*a* $\in$ *qbs-Mx X*
using *h* **by**(*auto simp*: *prod-qbs-Mx-def*)
have *1*:$\alpha' \in$ *qbs-Mx* (Π_Q *i*∈{..<*n*}. *X*)
using *h* **by**(*fastforce simp*: *prod-qbs-Mx-def list-tail-def* α'*-def*)
hence *2*: $\bigwedge r.$ (*n*, α' *r*) $\in$ *qbs-space* (*list-of X*)
using *qbs-Mx-to-X*[*of* α'] **by** *fastforce*
have *3*: $\bigwedge r.$ (*Suc n*, α *r*) $\in$ *qbs-space* (*list-of X*)
using *qbs-Mx-to-X*[*of* α] *h* **by** *fastforce*
have *4*: $\bigwedge r.$ (*n*, α' *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*]) *auto*
have *6*: ($\lambda r.$ (*n*, α' *r*)) $\in$ *qbs-Mx* (*list-of X*)
using *1* **by**(*auto intro*!: *coprod-qbs-MxI*)

have ($\lambda x\ y.$ *rec-list'* (*fst y*) (*snd y*) (*Suc n*, *x*)) $\circ\ \alpha$ = ($\lambda r\ y.$ *rec-list'* (*fst y*) (*snd y*) (*Suc n*, α *r*))
by *auto*

also have ... = $(\lambda r\ y.\ snd\ y\ (a\ r)\ (n, \alpha'\ r)\ (rec\text{-}list'\ (fst\ y)\ (snd\ y)\ (n, \alpha'\ r)))$
by(*simp only*: *5 rec-list'-simp2*[*OF 2*])
also have ... $\in$ *qbs-Mx* (*exp-qbs* ($Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) *Y*)
proof −
have $(\lambda(r,y).\ snd\ y\ (a\ r)\ (n, \alpha'\ r)\ (rec\text{-}list'\ (fst\ y)\ (snd\ y)\ (n, \alpha'\ r))) = (\lambda(y,x1,x2,x3).\ y\ x1\ x2\ x3) \circ (\lambda(r,y).\ (snd\ y,\ a\ r,\ (n, \alpha'\ r),\ rec\text{-}list'\ (fst\ y)\ (snd\ y)\ (n, \alpha'\ r)))$
by *auto*
also have ... $\in \mathbb{R}_Q \bigotimes_Q (Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) $\rightarrow_Q Y$
proof(*rule qbs-morphism-comp*[**where** *Y=exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\bigotimes_Q X \bigotimes_Q$ *list-of X* $\bigotimes_Q Y$])
show $(\lambda(r, y).\ (snd\ y,\ a\ r,\ (n, \alpha'\ r),\ rec\text{-}list'\ (fst\ y)\ (snd\ y)\ (n, \alpha'\ r))) \in \mathbb{R}_Q \bigotimes_Q Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\rightarrow_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\bigotimes_Q X \bigotimes_Q$ *list-of X* $\bigotimes_Q Y$
proof(*auto simp*: *split-beta' intro*!: *qbs-morphism-tuple*[*OF qbs-morphism-snd''*[*OF snd-qbs-morphism*] *qbs-morphism-tuple*[*of* $\lambda(r, y).\ a\ r\ \mathbb{R}_Q \bigotimes_Q Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) *X*], *OF* - *qbs-morphism-tuple*[*of* $\lambda(r,y).\ (n, \alpha'\ r)$],*of list-of X* $\lambda(r,y).$ *rec-list'* (*fst y*) (*snd y*) $(n, \alpha'\ r)$,*simplified split-beta'*])
show $(\lambda x.\ a\ (fst\ x)) \in \mathbb{R}_Q \bigotimes_Q Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\rightarrow_Q X$
using *ha qbs-Mx-is-morphisms*[*of X*] *qbs-morphism-fst''*[*of a* $\mathbb{R}_Q$ *X*] **by** *auto*
next
show $(\lambda x.\ (n, \alpha'\ (fst\ x))) \in \mathbb{R}_Q \bigotimes_Q Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\rightarrow_Q$ *list-of X*
using *qbs-morphism-fst''*[*of* $\lambda r.\ (n, \alpha'\ r)\ \mathbb{R}_Q$ *list-of X*] *qbs-Mx-is-morphisms*[*of list-of X*] *6* **by** *auto*
next
show $(\lambda x.\ rec\text{-}list'\ (fst\ (snd\ x))\ (snd\ (snd\ x))\ (n, \alpha'\ (fst\ x))) \in \mathbb{R}_Q \bigotimes_Q Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\rightarrow_Q Y$
using *qbs-morphismE*(*3*)[*OF ih 1*, *simplified comp-def*] *uncurry-preserves-morphisms*[*of* $(\lambda x\ y.\ rec\text{-}list'\ (fst\ y)\ (snd\ y)\ (n, \alpha'\ x))\ \mathbb{R}_Q\ Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) *Y*] *qbs-Mx-is-morphisms*[*of exp-qbs* ($Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) *Y*]
by(*fastforce simp*: *split-beta'*)
qed
next
show $(\lambda(y, x1, x2, x3).\ y\ x1\ x2\ x3) \in$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\bigotimes_Q X \bigotimes_Q$ *list-of X* $\bigotimes_Q Y \rightarrow_Q Y$
proof(*rule qbs-morphismI*)
fix β
assume $\beta \in$ *qbs-Mx* (*exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*)) $\bigotimes_Q X \bigotimes_Q$ *list-of X* $\bigotimes_Q Y$)
then have $\exists\ \beta 1\ \beta 2\ \beta 3\ \beta 4.\ \beta = (\lambda r.\ (\beta 1\ r,\ \beta 2\ r,\ \beta 3\ r,\ \beta 4\ r)) \wedge \beta 1 \in$ *qbs-Mx* (*exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) $\wedge\ \beta 2 \in$ *qbs-Mx X* $\wedge\ \beta 3 \in$ *qbs-Mx* (*list-of X*) $\wedge\ \beta 4 \in$ *qbs-Mx Y*
by(*auto intro*!: *exI*[**where** *x=fst* $\circ$ β] *exI*[**where** *x=fst* $\circ$ *snd*

$\circ\ \beta$] *exI*[**where** x=*fst* $\circ$ *snd* $\circ$ *snd* $\circ\ \beta$] *exI*[**where** x=*snd* $\circ$ *snd* $\circ$ *snd* $\circ\ \beta$] *simp*:*pair-qbs-Mx-def comp-def*)
then obtain $\beta 1\ \beta 2\ \beta 3\ \beta 4$ **where** *hb*:
$\beta = (\lambda r.\ (\beta 1\ r,\ \beta 2\ r,\ \beta 3\ r,\ \beta 4\ r))$ $\beta 1 \in$ *qbs-Mx* (*exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) $\beta 2 \in$ *qbs-Mx X* $\beta 3 \in$ *qbs-Mx* (*list-of X*) $\beta 4 \in$ *qbs-Mx Y*
by *auto*
hence *hbq*:$(\lambda(((r,x1),x2),x3).\ \beta 1\ r\ x1\ x2\ x3) \in ((\mathbb{R}_Q \bigotimes_Q X) \bigotimes_Q$ *list-of* $X) \bigotimes_Q Y \to_Q Y$
by(*simp add*: *exp-qbs-Mx-def*) (*meson uncurry-preserves-morphisms*)
have $(\lambda(y,\ x1,\ x2,\ x3).\ y\ x1\ x2\ x3) \circ \beta = (\lambda(((r,x1),x2),x3).\ \beta 1\ r\ x1\ x2\ x3) \circ (\lambda r.\ (((r,\beta 2\ r),\ \beta 3\ r),\ \beta 4\ r))$
by(*auto simp*: *hb*(*1*))
also have ... $\in \mathbb{R}_Q \to_Q Y$
using *hb*(*2*−*5*)
by(*auto intro*!: *qbs-morphism-comp*[*OF qbs-morphism-tuple*[*OF qbs-morphism-tuple*[*OF qbs-morphism-tuple*[*OF qbs-morphism-ident'*]]] *hbq*] *simp*: *qbs-Mx-is-morphisms*)
finally show $(\lambda(y,\ x1,\ x2,\ x3).\ y\ x1\ x2\ x3) \circ \beta \in$ *qbs-Mx Y*
by(*simp add*: *qbs-Mx-is-morphisms*)
qed
qed
finally show *?thesis*
by(*simp add*: *exp-qbs-Mx-def*)
qed
finally show $(\lambda x\ y.$ *rec-list'* (*fst y*) (*snd y*) (*Suc n*, x)) $\circ\ \alpha \in$ *qbs-Mx* (*exp-qbs* ($Y \bigotimes_Q$ *exp-qbs X* (*exp-qbs* (*list-of X*) (*exp-qbs Y Y*))) Y) **.**
qed
qed
qed *simp*

end

3 Probability Spaces

3.1 Probability Measures

theory *Probability-Space-QuasiBorel*
imports *Exponent-QuasiBorel*
begin

3.1.1 Probability Measures

type-synonym *'a qbs-prob-t* = *'a quasi-borel* ∗ (*real* ⇒ *'a*) ∗ *real measure*

locale *in-Mx* =
fixes X :: *'a quasi-borel*
and α :: *real* ⇒ *'a*

assumes *in-Mx*[*simp*]:$\alpha \in$ *qbs-Mx X*

locale *qbs-prob* = *in-Mx X* α + *real-distribution* μ
for *X* :: *'a quasi-borel* **and** α **and** μ
begin
declare *prob-space-axioms*[*simp*]

lemma *m-in-space-prob-algebra*[*simp*]:
$\mu \in$ *space* (*prob-algebra real-borel*)
using *space-prob-algebra*[*of real-borel*] **by** *simp*
end

locale *pair-qbs-probs* = *qp1*:*qbs-prob X* α μ + *qp2*:*qbs-prob Y* β ν
for *X* :: *'a quasi-borel***and** α μ **and** *Y* :: *'b quasi-borel* **and** β ν
begin

sublocale *pair-prob-space* μ ν
by *standard*

lemma *ab-measurable*[*measurable*]:
map-prod α $\beta \in$ *real-borel* $\bigotimes_M$ *real-borel* $\to_M$ *qbs-to-measure* ($X \bigotimes_Q Y$)
using *qbs-morphism-map-prod*[*of* α $\mathbb{R}_Q$ *X* β $\mathbb{R}_Q$ *Y*] *qp1.in-Mx qp2.in-Mx l-preserves-morphisms*[*of* $\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$ $X \bigotimes_Q Y$]
by(*auto simp*: *qbs-Mx-is-morphisms*)

lemma *ab-g-in-Mx*[*simp*]:
map-prod α $\beta \circ$ *real-real.g* $\in$ *pair-qbs-Mx X Y*
using *qbs-closed1-dest*[*OF qp1.in-Mx*] *qbs-closed1-dest*[*OF qp2.in-Mx*]
by(*auto simp add*: *pair-qbs-Mx-def comp-def*)

sublocale *qbs-prob* $X \bigotimes_Q Y$ *map-prod* α $\beta \circ$ *real-real.g distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*
by(*auto simp*: *qbs-prob-def in-Mx-def*)

end

locale *pair-qbs-prob* = *qp1*:*qbs-prob X* α μ + *qp2*:*qbs-prob Y* β ν
for *X* :: *'a quasi-borel***and** α μ **and** *Y* :: *'a quasi-borel* **and** β ν
begin

sublocale *pair-qbs-probs*
by *standard*

lemma *same-spaces*[*simp*]:
assumes $Y = X$
shows $\beta \in$ *qbs-Mx X*
by(*simp add*: *assms*[*symmetric*])

end

```
lemma prob-algebra-real-prob-measure:
  p ∈ space (prob-algebra (real-borel)) = real-distribution p
proof
  assume p ∈ space (prob-algebra real-borel)
  then show real-distribution p
    unfolding real-distribution-def real-distribution-axioms-def
    by(simp add: space-prob-algebra sets-eq-imp-space-eq)
next
  assume real-distribution p
  then interpret rd: real-distribution p .
  show p ∈ space (prob-algebra real-borel)
    by (simp add: space-prob-algebra rd.prob-space-axioms)
qed

lemma qbs-probI:
  assumes α ∈ qbs-Mx X
      and sets μ = sets borel
      and prob-space μ
    shows qbs-prob X α μ
  using assms
  by(auto intro!: qbs-prob.intro simp: in-Mx-def real-distribution-def real-distribution-axioms-def)

lemma qbs-empty-not-qbs-prob :¬ qbs-prob (empty-quasi-borel) f M
  by(simp add: qbs-prob-def in-Mx-def)

definition qbs-prob-eq :: ['a qbs-prob-t, 'a qbs-prob-t] ⇒ bool where
  qbs-prob-eq p1 p2 ≡
   (let (qbs1, a1, m1) = p1;
        (qbs2, a2, m2) = p2 in
     qbs-prob qbs1 a1 m1 ∧ qbs-prob qbs2 a2 m2 ∧ qbs1 = qbs2 ∧
       distr m1 (qbs-to-measure qbs1) a1 = distr m2 (qbs-to-measure qbs2) a2)

definition qbs-prob-eq2 :: ['a qbs-prob-t, 'a qbs-prob-t] ⇒ bool where
  qbs-prob-eq2 p1 p2 ≡
   (let (qbs1, a1, m1) = p1;
        (qbs2, a2, m2) = p2 in
     qbs-prob qbs1 a1 m1 ∧ qbs-prob qbs2 a2 m2 ∧ qbs1 = qbs2 ∧
       (∀f ∈ qbs1 →Q real-quasi-borel.
            (∫ x. f (a1 x) ∂ m1) = (∫ x. f (a2 x) ∂ m2)))

definition qbs-prob-eq3 :: ['a qbs-prob-t, 'a qbs-prob-t] ⇒ bool where
  qbs-prob-eq3 p1 p2 ≡
     (let (qbs1, a1, m1) = p1;
          (qbs2, a2, m2) = p2 in
      (qbs-prob qbs1 a1 m1 ∧ qbs-prob qbs2 a2 m2 ∧ qbs1 = qbs2 ∧
       (∀f ∈ qbs1 →Q real-quasi-borel.
          (∀ k ∈ qbs-space qbs1. 0 ≤ f k) ⟶
            (∫ x. f (a1 x) ∂ m1) = (∫ x. f (a2 x) ∂ m2))))
```

definition *qbs-prob-eq4* :: [$'a$ *qbs-prob-t*, $'a$ *qbs-prob-t*] ⇒ *bool* **where**
qbs-prob-eq4 p1 p2 ≡
(*let* (*qbs1*, *a1*, *m1*) = *p1*;
(*qbs2*, *a2*, *m2*) = *p2 in*
(*qbs-prob qbs1 a1 m1* ∧ *qbs-prob qbs2 a2 m2* ∧ *qbs1* = *qbs2* ∧
(∀ *f* ∈ *qbs1* $\rightarrow_Q$ $\mathbb{R}_{Q\geq 0}$.
($\int^+ x.$ *f* (*a1 x*) ∂ *m1*) = ($\int^+ x.$ *f* (*a2 x*) ∂ *m2*))))

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

lemma(**in** *qbs-prob*) *qbs-prob-eq2-refl*[*simp*]:
qbs-prob-eq2 (*X*,α,μ) (*X*,α,μ)
by(*simp add*: *qbs-prob-eq2-def qbs-prob-axioms*)

lemma(**in** *qbs-prob*) *qbs-prob-eq3-refl*[*simp*]:
qbs-prob-eq3 (*X*,α,μ) (*X*,α,μ)
by(*simp add*: *qbs-prob-eq3-def qbs-prob-axioms*)

lemma(**in** *qbs-prob*) *qbs-prob-eq4-refl*[*simp*]:
qbs-prob-eq4 (*X*,α,μ) (*X*,α,μ)
by(*simp add*: *qbs-prob-eq4-def qbs-prob-axioms*)

lemma(**in** *pair-qbs-prob*) *qbs-prob-eq-intro*:
assumes *X* = *Y*
and *distr* μ (*qbs-to-measure X*) α = *distr* ν (*qbs-to-measure X*) β
shows *qbs-prob-eq* (*X*,α,μ) (*Y*,β,ν)
using *assms qp1.qbs-prob-axioms qp2.qbs-prob-axioms*
by(*auto simp add*: *qbs-prob-eq-def*)

lemma(**in** *pair-qbs-prob*) *qbs-prob-eq2-intro*:
assumes *X* = *Y*
and ⋀*f*. *f* ∈ *qbs-to-measure X* $\rightarrow_M$ *real-borel*
⟹ ($\int x.$ *f* (α *x*) ∂ μ) = ($\int x.$ *f* (β *x*) ∂ ν)
shows *qbs-prob-eq2* (*X*,α,μ) (*Y*,β,ν)
using *assms qp1.qbs-prob-axioms qp2.qbs-prob-axioms*
by(*auto simp add*: *qbs-prob-eq2-def*)

lemma(**in** *pair-qbs-prob*) *qbs-prob-eq3-intro*:
assumes *X* = *Y*
and ⋀*f*. *f* ∈ *qbs-to-measure X* $\rightarrow_M$ *real-borel* ⟹ (∀ *k* ∈ *qbs-space X*. *0* ≤ *f k*)
⟹ ($\int x.$ *f* (α *x*) ∂ μ) = ($\int x.$ *f* (β *x*) ∂ ν)
shows *qbs-prob-eq3* (*X*,α,μ) (*Y*,β,ν)
using *assms qp1.qbs-prob-axioms qp2.qbs-prob-axioms*
by(*auto simp add*: *qbs-prob-eq3-def*)

lemma(**in** *pair-qbs-prob*) *qbs-prob-eq4-intro*:
  assumes $X = Y$
    and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \rightarrow_M$ *ennreal-borel*
      $\Longrightarrow (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\ \nu)$
   shows *qbs-prob-eq4* (X,α,μ) (Y,β,ν)
  using *assms qp1.qbs-prob-axioms qp2.qbs-prob-axioms*
  by(*auto simp add*: *qbs-prob-eq4-def*)

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

lemma *qbs-prob-eq2-dest*:
  assumes *qbs-prob-eq2* (X,α,μ) (Y,β,ν)
  shows *qbs-prob* $X\ \alpha\ \mu$
    qbs-prob $Y\ \beta\ \nu$
    $Y = X$
   and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \rightarrow_M$ *real-borel*
    $\Longrightarrow (\int x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int x.\ f\ (\beta\ x)\ \partial\ \nu)$
  using *assms* **by**(*auto simp*: *qbs-prob-eq2-def*)

lemma *qbs-prob-eq3-dest*:
  assumes *qbs-prob-eq3* (X,α,μ) (Y,β,ν)
  shows *qbs-prob* $X\ \alpha\ \mu$
    qbs-prob $Y\ \beta\ \nu$
    $Y = X$
   and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \rightarrow_M$ *real-borel* $\Longrightarrow (\forall\ k \in$ *qbs-space* $X.\ 0 \leq f$
$k)$
    $\Longrightarrow (\int x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int x.\ f\ (\beta\ x)\ \partial\ \nu)$
  using *assms* **by**(*auto simp*: *qbs-prob-eq3-def*)

lemma *qbs-prob-eq4-dest*:
  assumes *qbs-prob-eq4* (X,α,μ) (Y,β,ν)
  shows *qbs-prob* $X\ \alpha\ \mu$
    qbs-prob $Y\ \beta\ \nu$
    $Y = X$
   and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \rightarrow_M$ *ennreal-borel*
    $\Longrightarrow (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\ \nu)$
  using *assms* **by**(*auto simp*: *qbs-prob-eq4-def*)

definition *qbs-prob-t-ennintegral* :: [*'a qbs-prob-t*, *'a* ⇒ *ennreal*] ⇒ *ennreal* **where**
qbs-prob-t-ennintegral p f ≡
  (*if f* ∈ (*fst p*) $\rightarrow_Q$ *ennreal-quasi-borel*
   then $(\int^+ x.\ f$ (*fst* (*snd p*) x) ∂ (*snd* (*snd p*))) *else 0*)

definition *qbs-prob-t-integral* :: ['*a qbs-prob-t*, '*a* ⇒ *real*] ⇒ *real* **where**
qbs-prob-t-integral p f ≡
  (*if f* ∈ (*fst p*) $\rightarrow_Q$ $\mathbb{R}_Q$
   then (∫ *x*. *f* (*fst* (*snd p*) *x*) ∂ (*snd* (*snd p*)))
   else 0)

definition *qbs-prob-t-integrable* :: ['*a qbs-prob-t*, '*a* ⇒ *real*] ⇒ *bool* **where**
qbs-prob-t-integrable p f ≡ *f* ∈ *fst p* $\rightarrow_Q$ *real-quasi-borel* ∧ *integrable* (*snd* (*snd p*))
(*f* ∘ (*fst* (*snd p*)))

definition *qbs-prob-t-measure* :: '*a qbs-prob-t* ⇒ '*a measure* **where**
qbs-prob-t-measure p ≡ *distr* (*snd* (*snd p*)) (*qbs-to-measure* (*fst p*)) (*fst* (*snd p*))

lemma *qbs-prob-eq-symp*:
 symp qbs-prob-eq
  by(*simp add*: *symp-def qbs-prob-eq-def*)

lemma *qbs-prob-eq-transp*:
 transp qbs-prob-eq
  by(*simp add*: *transp-def qbs-prob-eq-def*)

quotient-type '*a qbs-prob-space* = '*a qbs-prob-t* / *partial*: *qbs-prob-eq*
  morphisms *rep-qbs-prob-space qbs-prob-space*
proof(*rule part-equivpI*)
  let *?U* = *UNIV* :: '*a set*
  let *?Uf* = *UNIV* :: (*real* ⇒ '*a*) *set*
  let *?f* = (λ-. *undefined*) :: *real* ⇒ '*a*
  have *qbs-prob* (*Abs-quasi-borel* (*?U*,*?Uf*)) *?f* (*return borel 0*)
  proof(*auto simp add*: *qbs-prob-def in-Mx-def*)
    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*)
  next
    show *real-distribution* (*return borel 0*)
     by (*simp add*: *prob-space-return real-distribution-axioms-def real-distribution-def*)
  qed
  thus ∃ *x* :: '*a qbs-prob-t* . *qbs-prob-eq x x*
    unfolding *qbs-prob-eq-def*
    by(*auto intro*!: *exI*[**where** *x*=(*Abs-quasi-borel* (*?U*,*?Uf*), *?f*, *return borel 0*)])
qed (*simp-all add*: *qbs-prob-eq-symp qbs-prob-eq-transp*)

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

lemma *qbs-prob-space-induct*:
assumes $\bigwedge X\ \alpha\ \mu.$ *qbs-prob* $X\ \alpha\ \mu \Longrightarrow P$ (*qbs-prob-space* (X,α,μ))
shows $P\ s$
apply(*rule qbs-prob-space.abs-induct*)
using *assms* **by**(*auto simp*: *qbs-prob-eq-def*)

lemma *qbs-prob-space-induct'*:
assumes $\bigwedge X\ \alpha\ \mu.$ *qbs-prob* $X\ \alpha\ \mu \Longrightarrow s =$ *qbs-prob-space* $(X,\alpha,\mu) \Longrightarrow P$ (*qbs-prob-space* (X,α,μ))
shows $P\ s$
by (*metis* (*no-types*, *lifting*) *Rep-qbs-prob-space-inverse assms case-prodE qbs-prob-eq-def qbs-prob-space.abs-def qbs-prob-space.rep-prop qbs-prob-space-def*)

lemma *rep-qbs-prob-space*:
$\exists X\ \alpha\ \mu.\ p =$ *qbs-prob-space* $(X,\ \alpha,\ \mu) \wedge$ *qbs-prob* $X\ \alpha\ \mu$
by(*rule qbs-prob-space.abs-induct*,*auto simp add*: *qbs-prob-eq-def*)

lemma(**in** *qbs-prob*) *in-Rep*:
$(X,\ \alpha,\ \mu) \in$ *Rep-qbs-prob-space* (*qbs-prob-space* (X,α,μ))
by (*metis mem-Collect-eq qbs-prob-eq-refl qbs-prob-space.abs-def qbs-prob-space.abs-inverse qbs-prob-space-def*)

lemma(**in** *qbs-prob*) *if-in-Rep*:
assumes $(X',\alpha',\mu') \in$ *Rep-qbs-prob-space* (*qbs-prob-space* (X,α,μ))
shows $X' = X$
qbs-prob $X'\ \alpha'\ \mu'$
qbs-prob-eq $(X,\alpha,\mu)\ (X',\alpha',\mu')$
proof −
have h:$X' = X$
by (*metis assms mem-Collect-eq qbs-prob-eq-dest*(*3*) *qbs-prob-eq-refl qbs-prob-space.abs-def qbs-prob-space.abs-inverse qbs-prob-space-def*)
have [*simp*]:*qbs-prob* $X'\ \alpha'\ \mu'$
by (*metis assms mem-Collect-eq prod-cases3 qbs-prob-eq-dest*(*2*) *qbs-prob-space.rep-prop*)
have [*simp*]:*qbs-prob-eq* $(X,\alpha,\mu)\ (X',\alpha',\mu')$
by (*metis assms mem-Collect-eq qbs-prob-eq-refl qbs-prob-space.abs-def qbs-prob-space.abs-inverse qbs-prob-space-def*)
show $X' = X$
qbs-prob $X'\ \alpha'\ \mu'$
qbs-prob-eq $(X,\alpha,\mu)\ (X',\alpha',\mu')$
by *simp-all* (*simp add*: *h*)
qed

lemma(**in** *qbs-prob*) *in-Rep-induct*:
assumes $\bigwedge Y\ \beta\ \nu.\ (Y,\beta,\nu) \in$ *Rep-qbs-prob-space* (*qbs-prob-space* (X,α,μ)) $\Longrightarrow$ $P\ (Y,\beta,\nu)$
shows P (*rep-qbs-prob-space* (*qbs-prob-space* (X,α,μ)))
unfolding *rep-qbs-prob-space-def qbs-prob-space.rep-def*
by(*rule someI2*[**where** $a=(X,\alpha,\mu)$]) (*use in-Rep assms* **in** *auto*)

lemma *qbs-prob-eq-2-implies-3* :
  assumes *qbs-prob-eq2 p1 p2*
  shows *qbs-prob-eq3 p1 p2*
  using *assms* **by**(*auto simp*: *qbs-prob-eq2-def qbs-prob-eq3-def*)

lemma *qbs-prob-eq-3-implies-1* :
  assumes *qbs-prob-eq3* (*p1* :: *'a qbs-prob-t*) *p2*
  shows *qbs-prob-eq p1 p2*
proof(*rule prod-cases3*[**where** *y*=*p1*],*rule prod-cases3*[**where** *y*=*p2*],*simp*)
  fix *X Y* :: *'a quasi-borel*
  fix α β μ ν
  assume *p1* = (X,α,μ) *p2* = (Y,β,ν)
  then have *h*:*qbs-prob-eq3* (X,α,μ) (Y,β,ν)
    using *assms* **by** *simp*
  then interpret *qp* : *pair-qbs-prob* X α μ Y β ν
    by(*auto intro*!: *pair-qbs-prob.intro simp*: *qbs-prob-eq3-def*)
  note [*simp*] = *qbs-prob-eq3-dest*(*3*)[*OF h*]

  show *qbs-prob-eq* (X,α,μ) (Y,β,ν)
  proof(*rule qp.qbs-prob-eq-intro*)
   show *distr* μ (*qbs-to-measure X*) α = *distr* ν (*qbs-to-measure X*) β
    proof(*rule measure-eqI*)
      fix *U*
      assume *hu*:$U \in$ *sets* (*distr* μ (*qbs-to-measure X*) α)
    have *measure* (*distr* μ (*qbs-to-measure X*) α) *U* = *measure* (*distr* ν (*qbs-to-measure X*) β) *U*
          (**is** *?lhs* = *?rhs*)
      proof −
        have *?lhs* = *measure* μ (α −' *U* $\cap$ *space* μ)
          by(*rule measure-distr*) (*use hu* **in** *simp-all*)
        also have ... = $integral^L$ μ (*indicat-real* (α −' *U*))
          by *simp*
        also have ... = ($\int$ *x*. *indicat-real U* (α *x*) $\partial\mu$)
           using *indicator-vimage*[*of* α *U*] *Bochner-Integration.integral-cong*[*of* μ - *indicat-real* (α −' *U*) λx. *indicat-real U* (α *x*)]
          by *auto*
        also have ... = ($\int$ *x*. *indicat-real U* (β *x*) $\partial\nu$)
          using *qbs-prob-eq3-dest*(*4*)[*OF h*,*of indicat-real U*] *hu*
          by *simp*
        also have ... = $integral^L$ ν (*indicat-real* (β −' *U*))
       using *indicator-vimage*[*of* β *U*,*symmetric*] *Bochner-Integration.integral-cong*[*of* ν - λx. *indicat-real U* (β *x*) *indicat-real* (β −' *U*)]
          by *blast*
        also have ... = *measure* ν (β −' *U* $\cap$ *space* ν)
          by *simp*
        also have ... = *?rhs*
          by(*rule measure-distr*[*symmetric*]) (*use hu* **in** *simp-all*)
        finally show *?thesis* **.**

```
    qed
      thus emeasure (distr μ (qbs-to-measure X) α) U = emeasure (distr ν
(qbs-to-measure X) β) U
      using qp.qp2.finite-measure-distr[of β] qp.qp1.finite-measure-distr[of α]
      by(simp add: finite-measure.emeasure-eq-measure)
  qed simp
 qed simp
qed

lemma qbs-prob-eq-1-implies-2 :
 assumes qbs-prob-eq p1 (p2 :: 'a qbs-prob-t)
 shows qbs-prob-eq2 p1 p2
proof(rule prod-cases3[where y=p1],rule prod-cases3[where y=p2],simp)
 fix X Y :: 'a quasi-borel
 fix α β μ ν
 assume p1 = (X,α,μ) p2 = (Y,β,ν)
 then have h:qbs-prob-eq (X,α,μ) (Y,β,ν)
  using assms by simp
 then interpret qp : pair-qbs-prob X α μ Y β ν
  by(auto intro!: pair-qbs-prob.intro simp: qbs-prob-eq-def)
 note [simp] = qbs-prob-eq-dest(3)[OF h]

 show qbs-prob-eq2 (X,α,μ) (Y,β,ν)
 proof(rule qp.qbs-prob-eq2-intro)
  fix f :: 'a ⇒ real
  assume [measurable]:f ∈ borel-measurable (qbs-to-measure X)
  show (∫ r. f (α r) ∂ μ) = (∫ r. f (β r) ∂ ν)
     (is ?lhs = ?rhs)
  proof –
   have ?lhs = (∫ x. f x ∂(distr μ (qbs-to-measure X) α))
    by(simp add: Bochner-Integration.integral-distr[symmetric])
   also have ... = (∫ x. f x ∂(distr ν (qbs-to-measure X) β))
    by(simp add: qbs-prob-eq-dest(4)[OF h])
   also have ... = ?rhs
    by(simp add: Bochner-Integration.integral-distr)
   finally show ?thesis .
  qed
 qed simp
qed

lemma qbs-prob-eq-1-implies-4 :
 assumes qbs-prob-eq p1 p2
 shows qbs-prob-eq4 p1 p2
proof(rule prod-cases3[where y=p1],rule prod-cases3[where y=p2],simp)
 fix X Y :: 'a quasi-borel
 fix α β μ ν
 assume p1 = (X,α,μ) p2 = (Y,β,ν)
 then have h:qbs-prob-eq (X,α,μ) (Y,β,ν)
  using assms by simp
```

then interpret *qp* : *pair-qbs-prob X* α μ *Y* β ν
by(*auto intro*!: *pair-qbs-prob.intro simp*: *qbs-prob-eq-def*)
note [*simp*] = *qbs-prob-eq-dest*(*3*)[*OF h*]

show *qbs-prob-eq4* (*X*,α,μ) (*Y*,β,ν)
proof(*rule qp.qbs-prob-eq4-intro*)
fix *f* ::'*a* ⇒ *ennreal*
assume [*measurable*]:*f* ∈ *borel-measurable* (*qbs-to-measure X*)
show ($\int^+$ *x*. *f* (α *x*) $\partial\mu$) = ($\int^+$ *x*. *f* (β *x*) $\partial\nu$)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *integral*^{N} (*distr* μ (*qbs-to-measure X*) α) *f*
by(*simp add*: *nn-integral-distr*)
also have ... = *integral*^{N} (*distr* ν (*qbs-to-measure X*) β) *f*
by(*simp add*: *qbs-prob-eq-dest*(*4*)[*OF h*])
also have ... = *?rhs*
by(*simp add*: *nn-integral-distr*)
finally show *?thesis* **.**
qed
qed *simp*
qed

lemma *qbs-prob-eq-4-implies-3* :
assumes *qbs-prob-eq4 p1 p2*
shows *qbs-prob-eq3 p1 p2*
proof(*rule prod-cases3*[**where** *y*=*p1*],*rule prod-cases3*[**where** *y*=*p2*],*simp*)
fix *X Y* :: '*a quasi-borel*
fix α β μ ν
assume *p1* = (*X*,α,μ) *p2* = (*Y*,β,ν)
then have *h*:*qbs-prob-eq4* (*X*,α,μ) (*Y*,β,ν)
using *assms* **by** *simp*
then interpret *qp* : *pair-qbs-prob X* α μ *Y* β ν
by(*auto intro*!: *pair-qbs-prob.intro simp*: *qbs-prob-eq4-def*)
note [*simp*] = *qbs-prob-eq4-dest*(*3*)[*OF h*]

show *qbs-prob-eq3* (*X*,α,μ) (*Y*,β,ν)
proof(*rule qp.qbs-prob-eq3-intro*)
fix *f* :: '*a* ⇒ *real*
assume [*measurable*]:*f* ∈ *borel-measurable* (*qbs-to-measure X*)
and *h*′: ∀ *k*∈*qbs-space X*. *0* ≤ *f k*
show ($\int$ *x*. *f* (α *x*) $\partial\mu$) = ($\int$ *x*. *f* (β *x*) $\partial\nu$)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *enn2real* ($\int^+$ *x*. *ennreal* (*f* (α *x*)) $\partial\mu$)
using *h*′ **by**(*auto simp*: *integral-eq-nn-integral*[**where** *f*=(λ*x*. *f* (α *x*))]
qbs-Mx-to-X(*2*))
also have ... = *enn2real* ($\int^+$ *x*. (*ennreal* ∘ *f*) (α *x*) $\partial\mu$)
by *simp*
also have ... = *enn2real* ($\int^+$ *x*. (*ennreal* ∘ *f*) (β *x*) $\partial\nu$)

using *qbs-prob-eq4-dest*(*4*)[*OF h*,*of ennreal* ∘ *f*] **by** *simp*
also have ... = *enn2real* ($\int^+$ *x*. *ennreal* (*f* (β *x*)) ∂ν)
by *simp*
also have ... = *?rhs*
using *h′* **by**(*auto simp*: *integral-eq-nn-integral*[**where** *f*=(λ*x*. *f* (β *x*))]
qbs-Mx-to-X(*2*))
finally show *?thesis* .
qed
qed *simp*
qed

lemma *qbs-prob-eq-equiv12* :
qbs-prob-eq = *qbs-prob-eq2*
using *qbs-prob-eq-1-implies-2 qbs-prob-eq-2-implies-3 qbs-prob-eq-3-implies-1*
by *blast*

lemma *qbs-prob-eq-equiv13* :
qbs-prob-eq = *qbs-prob-eq3*
using *qbs-prob-eq-1-implies-2 qbs-prob-eq-2-implies-3 qbs-prob-eq-3-implies-1*
by *blast*

lemma *qbs-prob-eq-equiv14* :
qbs-prob-eq = *qbs-prob-eq4*
using *qbs-prob-eq-2-implies-3 qbs-prob-eq-3-implies-1 qbs-prob-eq-1-implies-4 qbs-prob-eq-4-implies-3*
qbs-prob-eq-1-implies-2
by *blast*

lemma *qbs-prob-eq-equiv23* :
qbs-prob-eq2 = *qbs-prob-eq3*
using *qbs-prob-eq-1-implies-2 qbs-prob-eq-2-implies-3 qbs-prob-eq-3-implies-1*
by *blast*

lemma *qbs-prob-eq-equiv24* :
qbs-prob-eq2 = *qbs-prob-eq4*
using *qbs-prob-eq-2-implies-3 qbs-prob-eq-4-implies-3 qbs-prob-eq-3-implies-1 qbs-prob-eq-1-implies-4*
qbs-prob-eq-1-implies-2
by *blast*

lemma *qbs-prob-eq-equiv34*:
qbs-prob-eq3 = *qbs-prob-eq4*
using *qbs-prob-eq-3-implies-1 qbs-prob-eq-1-implies-4 qbs-prob-eq-4-implies-3*
by *blast*

lemma *qbs-prob-eq-equiv31* :
qbs-prob-eq = *qbs-prob-eq3*
using *qbs-prob-eq-1-implies-2 qbs-prob-eq-2-implies-3 qbs-prob-eq-3-implies-1*
by *blast*

lemma *qbs-prob-space-eq*:

  assumes *qbs-prob-eq* (X,α,μ) (Y,β,ν)
  shows *qbs-prob-space* (X,α,μ) = *qbs-prob-space* (Y,β,ν)
  using *Quotient3-rel*[*OF Quotient3-qbs-prob-space*] *assms*
  by *blast*

lemma(**in** *pair-qbs-prob*) *qbs-prob-space-eq*:
  assumes $Y = X$
    and *distr* μ (*qbs-to-measure* X) α = *distr* ν (*qbs-to-measure* X) β
   shows *qbs-prob-space* (X,α,μ) = *qbs-prob-space* (Y,β,ν)
  using *assms qbs-prob-eq-intro qbs-prob-space-eq* **by** *auto*

lemma(**in** *pair-qbs-prob*) *qbs-prob-space-eq2*:
  assumes $Y = X$
    and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \to_M$ *real-borel*
      $\Longrightarrow (\int x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int x.\ f\ (\beta\ x)\ \partial\ \nu)$
   shows *qbs-prob-space* (X,α,μ) = *qbs-prob-space* (Y,β,ν)
 using *qbs-prob-space-eq assms qbs-prob-eq-2-implies-3*[*of* (X,α,μ) (Y,β,ν)] *qbs-prob-eq-3-implies-1*[*of* (X,α,μ) (Y,β,ν)] *qbs-prob-eq2-intro qbs-prob-eq-dest*(*4*)
  by *blast*

lemma(**in** *pair-qbs-prob*) *qbs-prob-space-eq3*:
  assumes $Y = X$
    and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \to_M$ *real-borel* $\Longrightarrow$ ($\forall k \in$ *qbs-space* X. $0 \le f$
k)
      $\Longrightarrow (\int x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int x.\ f\ (\beta\ x)\ \partial\ \nu)$
   shows *qbs-prob-space* (X,α,μ) = *qbs-prob-space* (Y,β,ν)
 using *assms qbs-prob-eq-3-implies-1*[*of* (X,α,μ) (Y,β,ν)] *qbs-prob-eq3-intro qbs-prob-space-eq qbs-prob-eq-dest*(*4*)
  by *blast*

lemma(**in** *pair-qbs-prob*) *qbs-prob-space-eq4*:
  assumes $Y = X$
    and $\bigwedge f.\ f \in$ *qbs-to-measure* $X \to_M$ *ennreal-borel*
      $\Longrightarrow (\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu) = (\int^+ x.\ f\ (\beta\ x)\ \partial\ \nu)$
   shows *qbs-prob-space* (X,α,μ) = *qbs-prob-space* (Y,β,ν)
  using *assms qbs-prob-eq-4-implies-3*[*of* (X,α,μ) (Y,β,ν)] *qbs-prob-space-eq3*[*OF assms*(*1*)] *qbs-prob-eq3-dest*(*4*) *qbs-prob-eq4-intro*
  by *blast*

lemma(**in** *pair-qbs-prob*) *qbs-prob-space-eq-inverse*:
  assumes *qbs-prob-space* (X,α,μ) = *qbs-prob-space* (Y,β,ν)
   shows *qbs-prob-eq* (X,α,μ) (Y,β,ν)
    and *qbs-prob-eq2* (X,α,μ) (Y,β,ν)
    and *qbs-prob-eq3* (X,α,μ) (Y,β,ν)
    and *qbs-prob-eq4* (X,α,μ) (Y,β,ν)
 using *Quotient3-rel*[*OF Quotient3-qbs-prob-space*,*of* (X, α, μ) (Y,β,ν),*simplified*] *assms qp1.qbs-prob-axioms qp2.qbs-prob-axioms*
  by(*simp-all add*: *qbs-prob-eq-equiv13*[*symmetric*] *qbs-prob-eq-equiv12*[*symmetric*] *qbs-prob-eq-equiv14*[*symmetric*])

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

lemma(**in** *qbs-prob*) *qbs-prob-space-qbs-computation*[*simp*]:
qbs-prob-space-qbs (*qbs-prob-space* (X,α,μ)) = X
by(*simp add*: *qbs-prob-space-qbs.abs-eq*)

lemma *rep-qbs-prob-space'*:
assumes *qbs-prob-space-qbs* $s = X$
shows ∃ α μ. s = *qbs-prob-space* (X,α,μ) ∧ *qbs-prob* X α μ
proof −
obtain X' α μ **where** *hs*:
s = *qbs-prob-space* (X', α, μ) *qbs-prob* X' α μ
using *rep-qbs-prob-space*[*of s*] **by** *auto*
then interpret *qp*:*qbs-prob* X' α μ
by *simp*
show *?thesis*
using *assms hs*(*2*) **by**(*auto simp add*: *hs*(*1*))
qed

lift-definition *qbs-prob-ennintegral* :: [*'a qbs-prob-space*, *'a* ⇒ *ennreal*] ⇒ *ennreal*
is *qbs-prob-t-ennintegral*
by(*auto simp add*: *qbs-prob-t-ennintegral-def qbs-prob-eq-equiv14 qbs-prob-eq4-def*)

lift-definition *qbs-prob-integral* :: [*'a qbs-prob-space*, *'a* ⇒ *real*] ⇒ *real*
is *qbs-prob-t-integral*
by(*auto simp add*: *qbs-prob-eq-equiv12 qbs-prob-t-integral-def qbs-prob-eq2-def*)

syntax
-qbs-prob-ennintegral :: *pttrn* ⇒ *ennreal* ⇒ *'a qbs-prob-space* ⇒ *ennreal* (‹$\int^+{}_Q$((*2* -./ -)/ ∂-)› [*60*,*61*] *110*)

syntax-consts
-qbs-prob-ennintegral ⇌ *qbs-prob-ennintegral*

translations
$\int^+{}_Q$ x. f ∂p ⇌ *CONST qbs-prob-ennintegral p* (λx. f)

syntax
-qbs-prob-integral :: *pttrn* ⇒ *real* ⇒ *'a qbs-prob-space* ⇒ *real* (‹$\int_Q$((*2* -./ -)/ ∂-)› [*60*,*61*] *110*)

syntax-consts
-qbs-prob-integral ⇌ *qbs-prob-integral*

translations
$\int_Q$ x. f ∂p ⇌ *CONST qbs-prob-integral p* (λx. f)

We define the function $l_X \in L(P(X)) \to_M G(X)$.

lift-definition *qbs-prob-measure* :: *'a qbs-prob-space* ⇒ *'a measure*
is *qbs-prob-t-measure*
  by(*auto simp add*: *qbs-prob-eq-def qbs-prob-t-measure-def*)

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

lemma(**in** *qbs-prob*) *qbs-prob-measure-computation*[*simp*]:
  qbs-prob-measure (*qbs-prob-space* (*X*,α,μ)) = *distr* μ (*qbs-to-measure X*) α
  by (*simp add*: *qbs-prob-measure.abs-eq qbs-prob-t-measure-def*)

definition *qbs-emeasure* ::*'a qbs-prob-space* ⇒ *'a set* ⇒ *ennreal* **where**
qbs-emeasure s ≡ *emeasure* (*qbs-prob-measure s*)

lemma(**in** *qbs-prob*) *qbs-emeasure-computation*[*simp*]:
  assumes *U* ∈ *sets* (*qbs-to-measure X*)
  shows *qbs-emeasure* (*qbs-prob-space* (*X*,α,μ)) *U* = *emeasure* μ (α −‘ *U*)
  by(*simp add*: *qbs-emeasure-def emeasure-distr*[*OF - assms*])

definition *qbs-measure* ::*'a qbs-prob-space* ⇒ *'a set* ⇒ *real* **where**
qbs-measure s ≡ *measure* (*qbs-prob-measure s*)

interpretation *qbs-prob-measure-prob-space* : *prob-space qbs-prob-measure* (*s*::*'a qbs-prob-space*) **for** *s*
proof(*transfer*,*auto*)
  fix *X* :: *'a quasi-borel*
  fix α μ
  assume *qbs-prob-eq* (*X*,α,μ) (*X*,α,μ)
  then interpret *qp*: *qbs-prob X* α μ
    by(*simp add*: *qbs-prob-eq-def*)
  show *prob-space* (*qbs-prob-t-measure* (*X*,α,μ))
    by(*simp add*: *qbs-prob-t-measure-def qp.prob-space-distr*)
qed

lemma *qbs-prob-measure-space*:
  qbs-space (*qbs-prob-space-qbs s*) = *space* (*qbs-prob-measure s*)
  by(*transfer*,*simp add*: *qbs-prob-t-measure-def*)

lemma *qbs-prob-measure-sets*[*measurable-cong*]:
  sets (*qbs-to-measure* (*qbs-prob-space-qbs s*)) = *sets* (*qbs-prob-measure s*)
  by(*transfer*,*simp add*: *qbs-prob-t-measure-def*)

lemma(**in** *qbs-prob*) *qbs-prob-ennintegral-def*:
  assumes $f \in X \to_Q \mathbb{R}_{Q\geq 0}$
    shows *qbs-prob-ennintegral* (*qbs-prob-space* (*X*,α,μ)) *f* = ($\int^+ x.\ f\ (\alpha\ x)\ \partial\ \mu$)
  by (*simp add*: *assms qbs-prob-ennintegral.abs-eq qbs-prob-t-ennintegral-def*)

lemma(**in** *qbs-prob*) *qbs-prob-ennintegral-def2*:
  assumes $f \in X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
  shows *qbs-prob-ennintegral* (*qbs-prob-space* (X,α,μ)) f = *integral*^{N} (*distr* μ (*qbs-to-measure* X) α) f
  using *assms* **by**(*auto simp add*: *qbs-prob-ennintegral.abs-eq qbs-prob-t-ennintegral-def qbs-prob-t-measure-def nn-integral-distr*)

lemma (**in** *qbs-prob*) *qbs-prob-ennintegral-not-morphism*:
  assumes $f \notin X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
  shows *qbs-prob-ennintegral* (*qbs-prob-space* (X,α,μ)) f = *0*
  by(*simp add*: *assms qbs-prob-ennintegral.abs-eq qbs-prob-t-ennintegral-def*)

lemma *qbs-prob-ennintegral-def2*:
  assumes *qbs-prob-space-qbs s* = (X :: *'a quasi-borel*)
    and $f \in X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
   shows *qbs-prob-ennintegral s f* = *integral*^{N} (*qbs-prob-measure s*) f
  using *assms*
proof(*transfer*,*auto*)
  fix X :: *'a quasi-borel* **and** α μ f
  assume *qbs-prob-eq* (X,α,μ) (X,α,μ)
    and h:$f \in X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
  then interpret *qp* : *qbs-prob* X α μ
   by(*simp add*: *qbs-prob-eq-def*)
  show *qbs-prob-t-ennintegral* (X, α, μ) f = *integral*^{N} (*qbs-prob-t-measure* (X, α, μ)) f
   using *qp.qbs-prob-ennintegral-def2*[*OF h*]
   by(*simp add*: *qbs-prob-ennintegral.abs-eq qbs-prob-t-measure-def*)
qed

lemma(**in** *qbs-prob*) *qbs-prob-integral-def*:
  assumes $f \in X \rightarrow_Q$ *real-quasi-borel*
   shows *qbs-prob-integral* (*qbs-prob-space* (X,α,μ)) f = ($\int x.\ f\ (\alpha\ x)\ \partial\ \mu$)
  by (*simp add*: *assms qbs-prob-integral.abs-eq qbs-prob-t-integral-def*)

lemma(**in** *qbs-prob*) *qbs-prob-integral-def2*:
 qbs-prob-integral (*qbs-prob-space* (X,α,μ)) f = *integral*^{L} (*distr* μ (*qbs-to-measure* X) α) f
proof −
  consider $f \in X \rightarrow_Q \mathbb{R}_Q \mid f \notin X \rightarrow_Q \mathbb{R}_Q$ **by** *auto*
  thus *?thesis*
  proof *cases*
   case *h*:*2*
   then have ¬ *integrable* (*qbs-prob-measure* (*qbs-prob-space* (X,α,μ))) f
    by *auto*
   thus *?thesis*
   using *h* **by**(*simp add*: *qbs-prob-integral.abs-eq qbs-prob-t-integral-def not-integrable-integral-eq*)
  qed (*auto simp add*: *qbs-prob-integral.abs-eq qbs-prob-t-integral-def integral-distr*
)

qed

lemma *qbs-prob-integral-def2*:
  qbs-prob-integral (*s*::′*a qbs-prob-space*) *f* = *integral*^{L} (*qbs-prob-measure s*) *f*
proof(*transfer*,*auto*)
  fix *X* :: ′*a quasi-borel* **and** μ α *f*
  assume *qbs-prob-eq* (X,α,μ) (X,α,μ)
  then interpret *qp* : *qbs-prob X* α μ
    by(*simp add*: *qbs-prob-eq-def*)
  show *qbs-prob-t-integral* (X,α,μ) *f* = *integral*^{L} (*qbs-prob-t-measure* (X,α,μ)) *f*
    using *qp.qbs-prob-integral-def2*
    by(*simp add*: *qbs-prob-t-measure-def qbs-prob-integral.abs-eq*)
qed

definition *qbs-prob-var* :: ′*a qbs-prob-space* ⇒ (′*a* ⇒ *real*) ⇒ *real* **where**
qbs-prob-var s f ≡ *qbs-prob-integral s* (λx. (*f x* − *qbs-prob-integral s f*)^2)

lemma(**in** *qbs-prob*) *qbs-prob-var-computation*:
  assumes *f* ∈ *X* $\to_Q$ *real-quasi-borel*
  shows *qbs-prob-var* (*qbs-prob-space* (X,α,μ)) *f* = ($\int$ *x*. (*f* (α *x*) − ($\int$ *x*. *f* (α *x*) ∂ μ))^2 ∂ μ)
proof −
  have (λx. (*f x* − *qbs-prob-integral* (*qbs-prob-space* (X, α, μ)) *f*)^2) ∈ *X* $\to_Q$ $\mathbb{R}_Q$
    using *assms* **by** *auto*
  thus *?thesis*
   using *assms qbs-prob-integral-def*[*of* λx. (*f x* − *qbs-prob-integral* (*qbs-prob-space* (X,α,μ)) *f*)^2]
    by(*simp add*: *qbs-prob-var-def qbs-prob-integral-def*)
qed

lift-definition *qbs-integrable* :: [′*a qbs-prob-space*, ′*a* ⇒ *real*] ⇒ *bool*
is *qbs-prob-t-integrable*
proof *auto*
  have *H*:⋀ (*X*::′*a quasi-borel*) *Y* α β μ ν *f*.
        qbs-prob-eq (X,α,μ) (Y,β,ν) ⟹ *qbs-prob-t-integrable* (X,α,μ) *f* ⟹ *qbs-prob-t-integrable* (Y,β,ν) *f*
  proof −
    fix *X Y* :: ′*a quasi-borel*
    fix α β μ ν *f*
    assume *H0*:*qbs-prob-eq* (X, α, μ) (Y, β, ν)
        qbs-prob-t-integrable (X, α, μ) *f*
    then interpret *qp*: *pair-qbs-prob X* α μ *Y* β ν
      by(*auto intro*!: *pair-qbs-prob.intro simp*: *qbs-prob-eq-def*)
    have [*measurable*]: *f* ∈ *qbs-to-measure X* $\to_M$ *real-borel*
        and *h*: *integrable* μ (*f* ∘ α)
      using *H0* **by**(*auto simp*: *qbs-prob-t-integrable-def*)
    note [*simp*] = *qbs-prob-eq-dest*(*3*)[*OF H0*(*1*)]

    show *qbs-prob-t-integrable* (Y, β, ν) *f*

unfolding *qbs-prob-t-integrable-def*
proof *auto*
have *integrable* (*distr* μ (*qbs-to-measure* X) α) *f*
using *h* **by**(*simp add*: *comp-def integrable-distr-eq*)
hence *integrable* (*distr* ν (*qbs-to-measure* X) β) *f*
using *qbs-prob-eq-dest*(*4*)[*OF H0*(*1*)] **by** *simp*
thus *integrable* ν ($f \circ \beta$)
by(*simp add*: *comp-def integrable-distr-eq*)
qed
qed
fix X Y :: *'a quasi-borel*
fix α β μ ν
assume *H0*:*qbs-prob-eq* (X, α, μ) (Y, β, ν)
then have *H1*:*qbs-prob-eq* (Y, β, ν) (X, α, μ)
by(*auto simp add*: *qbs-prob-eq-def*)
show *qbs-prob-t-integrable* (X, α, μ) = *qbs-prob-t-integrable* (Y, β, ν)
using *H*[*OF H0*] *H*[*OF H1*] **by** *auto*
qed

lemma(**in** *qbs-prob*) *qbs-integrable-def*:
qbs-integrable (*qbs-prob-space* (X, α, μ)) f = ($f \in X \to_Q \mathbb{R}_Q \wedge$ *integrable* μ ($f \circ$
α))
by (*simp add*: *qbs-integrable.abs-eq qbs-prob-t-integrable-def*)

lemma *qbs-integrable-morphism*:
assumes *qbs-prob-space-qbs* $s = X$
and *qbs-integrable s f*
shows $f \in X \to_Q \mathbb{R}_Q$
using *assms* **by**(*transfer*,*auto simp*: *qbs-prob-t-integrable-def*)

lemma(**in** *qbs-prob*) *qbs-integrable-measurable*[*simp*,*measurable*]:
assumes *qbs-integrable* (*qbs-prob-space* (X,α,μ)) f
shows $f \in$ *qbs-to-measure* $X \to_M$ *real-borel*
using *assms* **by**(*auto simp add*: *qbs-integrable-def*)

lemma *qbs-integrable-iff-integrable*:
(*qbs-integrable* (*s*::*'a qbs-prob-space*) *f*) = (*integrable* (*qbs-prob-measure s*) *f*)
apply *transfer*
subgoal for *s f*
proof(*rule prod-cases3*[**where** *y*=*s*],*simp*)
fix X :: *'a quasi-borel*
fix α μ
assume *qbs-prob-eq* (X,α,μ) (X,α,μ)
then interpret *qp*: *qbs-prob* X α μ
by(*simp add*: *qbs-prob-eq-def*)

show *qbs-prob-t-integrable* (X,α,μ) f = *integrable* (*qbs-prob-t-measure* (X,α,μ))
f
(**is** *?lhs* = *?rhs*)

using *integrable-distr-eq*[*of* α μ *qbs-to-measure X f*]
by(*auto simp add*: *qbs-prob-t-integrable-def qbs-prob-t-measure-def comp-def*)
qed
done

lemma(**in** *qbs-prob*) *qbs-integrable-iff-integrable-distr*:
qbs-integrable (*qbs-prob-space* (X,α,μ)) f = *integrable* (*distr* μ (*qbs-to-measure X*) α) f
by(*simp add*: *qbs-integrable-iff-integrable*)

lemma(**in** *qbs-prob*) *qbs-integrable-iff-integrable*:
assumes $f \in$ *qbs-to-measure X* $\rightarrow_M$ *real-borel*
shows *qbs-integrable* (*qbs-prob-space* (X,α,μ)) f = *integrable* μ ($\lambda x.\ f\ (\alpha\ x)$)
by(*auto intro*!: *integrable-distr-eq*[*of* α μ *qbs-to-measure X f*] *simp*: *assms qbs-integrable-iff-integrable-distr*)

lemma *qbs-integrable-if-integrable*:
assumes *integrable* (*qbs-prob-measure s*) f
shows *qbs-integrable* (*s*::'*a qbs-prob-space*) f
using *assms* **by**(*simp add*: *qbs-integrable-iff-integrable*)

lemma *integrable-if-qbs-integrable*:
assumes *qbs-integrable* (*s*::'*a qbs-prob-space*) f
shows *integrable* (*qbs-prob-measure s*) f
using *assms* **by**(*simp add*: *qbs-integrable-iff-integrable*)

lemma *qbs-integrable-iff-bounded*:
assumes *qbs-prob-space-qbs s* = X
shows *qbs-integrable s f* $\longleftrightarrow$ $f \in X \rightarrow_Q \mathbb{R}_Q \wedge$ *qbs-prob-ennintegral s* ($\lambda x.$ *ennreal* $|f\ x\ |$) $< \infty$
(**is** *?lhs* = *?rhs*)
proof −
obtain α μ **where** *hs*:
qbs-prob X α μ *s* = *qbs-prob-space* (X,α,μ)
using *rep-qbs-prob-space*'[*OF assms*] **by** *auto*
then interpret *qp*:*qbs-prob 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*)
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*
proof −
have [*simp*]:$f \in X \rightarrow_Q \mathbb{R}_Q \Longrightarrow$($\lambda x.$ *ennreal* $|f\ x|$) $\in X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
by *auto*
have $f \in X \rightarrow_Q \mathbb{R}_Q \Longrightarrow$ *qbs-prob-ennintegral s* ($\lambda x.$ *ennreal* $|f\ x|$) = ($\int^+ x.$ *ennreal* (*norm* ($f\ x$)) ∂*qbs-prob-measure s*)
using *qp.qbs-prob-ennintegral-def2*[*of* $\lambda x.$ *ennreal* $|f\ x|$]
by(*auto simp*: *hs*(*2*))
thus *?thesis*

by(*simp add*: *hs*(*2*)) *fastforce*
qed
finally show *?thesis* **.**
qed

lemma *qbs-integrable-cong*:
assumes *qbs-prob-space-qbs s* = *X*
$\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow f\ x = g\ x$
and *qbs-integrable s f*
shows *qbs-integrable s g*
by (*metis Bochner-Integration.integrable-cong assms integrable-if-qbs-integrable qbs-integrable-if-integrable qbs-prob-measure-space*)

lemma *qbs-integrable-const*[*simp*]:
qbs-integrable s ($\lambda x.\ c$)
using *qbs-integrable-iff-integrable*[*of s* $\lambda x.\ c$] **by** *simp*

lemma *qbs-integrable-add*[*simp*]:
assumes *qbs-integrable s f*
and *qbs-integrable s g*
shows *qbs-integrable s* ($\lambda x.\ f\ x + g\ x$)
using *assms qbs-integrable-iff-integrable* **by** *blast*

lemma *qbs-integrable-diff*[*simp*]:
assumes *qbs-integrable s f*
and *qbs-integrable s g*
shows *qbs-integrable s* ($\lambda x.\ f\ x - g\ x$)
by(*rule qbs-integrable-if-integrable*[*OF Bochner-Integration.integrable-diff*[*OF integrable-if-qbs-integrable*[*OF assms*(*1*)] *integrable-if-qbs-integrable*[*OF assms*(*2*)]]])

lemma *qbs-integrable-mult-iff*[*simp*]:
(*qbs-integrable s* ($\lambda x.\ c * f\ x$)) = ($c = 0 \vee$ *qbs-integrable s f*)
using *qbs-integrable-iff-integrable*[*of s* $\lambda x.\ c * f\ x$] *integrable-mult-left-iff*[*of qbs-prob-measure s c f*] *qbs-integrable-iff-integrable*[*of s f*]
by *simp*

lemma *qbs-integrable-mult*[*simp*]:
assumes *qbs-integrable s f*
shows *qbs-integrable s* ($\lambda x.\ c * f\ x$)
using *assms qbs-integrable-mult-iff* **by** *auto*

lemma *qbs-integrable-abs*[*simp*]:
assumes *qbs-integrable s f*
shows *qbs-integrable s* ($\lambda x.\ |f\ x|$)
by(*rule qbs-integrable-if-integrable*[*OF integrable-abs*[*OF integrable-if-qbs-integrable*[*OF assms*]]])

lemma *qbs-integrable-sq*[*simp*]:
assumes *qbs-integrable s f*

and *qbs-integrable s* ($\lambda x.$ (*f x*)^2)
shows *qbs-integrable s* ($\lambda x.$ (*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)

lemma *qbs-ennintegral-eq-qbs-integral*:
assumes *qbs-prob-space-qbs s* = *X*
qbs-integrable s f
and $\bigwedge x.$ $x \in$ *qbs-space X* $\Longrightarrow$ *0* $\leq$ *f x*
shows *qbs-prob-ennintegral s* ($\lambda x.$ *ennreal* (*f x*)) = *ennreal* (*qbs-prob-integral s*
f)
using *nn-integral-eq-integral*[*OF integrable-if-qbs-integrable*[*OF assms*(*2*)]] *assms*
qbs-prob-ennintegral-def2[*OF assms*(*1*) *qbs-morphism-comp*[*OF qbs-integrable-morphism*[*OF*
assms(*1*,*2*)],*of ennreal* $\mathbb{R}_{Q\geq 0}$,*simplified comp-def*]] *measurable-ennreal*
by (*metis AE-I2 qbs-prob-integral-def2 qbs-prob-measure-space real.standard-borel-r-full-faithful*)

lemma *qbs-prob-ennintegral-cong*:
assumes *qbs-prob-space-qbs s* = *X*
and $\bigwedge x.$ $x \in$ *qbs-space X* $\Longrightarrow$ *f x* = *g x*
shows *qbs-prob-ennintegral s f* = *qbs-prob-ennintegral s g*
proof −
obtain α μ **where** *hs*:
s = *qbs-prob-space* (*X*, α, μ) *qbs-prob X* α μ
using *rep-qbs-prob-space′*[*OF assms*(*1*)] **by** *auto*
then interpret *qp* : *qbs-prob X* α μ
by *simp*
have *H1*:*f* $\circ$ α = *g* $\circ$ α
using *assms*(*2*)
unfolding *comp-def* **apply** *standard*
using *assms*(*2*)[*of* α -] **by** (*simp add*: *qbs-Mx-to-X*(*2*))
consider $f \in X \to_Q \mathbb{R}_{Q\geq 0}$ | $f \notin X \to_Q \mathbb{R}_{Q\geq 0}$ **by** *auto*
then have *qbs-prob-t-ennintegral* (*X*,α,μ) *f* = *qbs-prob-t-ennintegral* (*X*,α,μ) *g*
proof *cases*
case *h*:*1*
then have $g \in X \to_Q \mathbb{R}_{Q\geq 0}$
using *qbs-morphism-cong*[*of X f g* $\mathbb{R}_{Q\geq 0}$] *assms* **by** *simp*
then show *?thesis*
using *h H1* **by**(*simp add*: *qbs-prob-t-ennintegral-def comp-def*)
next
case *h*:*2*
then have $g \notin X \to_Q \mathbb{R}_{Q\geq 0}$
using *assms qbs-morphism-cong*[*of X g f* $\mathbb{R}_{Q\geq 0}$] **by** *auto*
then show *?thesis*
using *h* **by**(*simp add*: *qbs-prob-t-ennintegral-def*)
qed
thus *?thesis*
using *hs*(*1*) **by** (*simp add*: *qbs-prob-ennintegral.abs-eq*)
qed

lemma *qbs-prob-ennintegral-const*:
qbs-prob-ennintegral (*s*::′*a qbs-prob-space*) (λ*x*. *c*) = *c*
using *qbs-prob-ennintegral-def2*[*OF* - *qbs-morphism-const*[*of c* $\mathbb{R}_{Q\geq 0}$ *qbs-prob-space-qbs s*,*simplified*]]
by (*simp add*: *qbs-prob-measure-prob-space.emeasure-space-1*)

lemma *qbs-prob-ennintegral-add*:
assumes *qbs-prob-space-qbs s* = *X*
f ∈ (*X*::′*a quasi-borel*) $\rightarrow_Q$ $\mathbb{R}_{Q\geq 0}$
and *g* ∈ *X* $\rightarrow_Q$ $\mathbb{R}_{Q\geq 0}$
shows *qbs-prob-ennintegral s* (λ*x*. *f x* + *g x*) = *qbs-prob-ennintegral s f* + *qbs-prob-ennintegral s g*
using *qbs-prob-ennintegral-def2*[*of s X* λ*x*. *f x* + *g x*] *qbs-prob-ennintegral-def2*[*OF assms*(*1*,*2*)] *qbs-prob-ennintegral-def2*[*OF assms*(*1*,*3*)] *assms nn-integral-add*[*of f qbs-prob-measure s g*]
by *fastforce*

lemma *qbs-prob-ennintegral-cmult*:
assumes *qbs-prob-space-qbs s* = *X*
and *f* ∈ *X* $\rightarrow_Q$ $\mathbb{R}_{Q\geq 0}$
shows *qbs-prob-ennintegral s* (λ*x*. *c* ∗ *f x*) = *c* ∗ *qbs-prob-ennintegral s f*
using *qbs-prob-ennintegral-def2*[*OF assms*(*1*),*of* λ*x*. *c* ∗ *f x*] *qbs-prob-ennintegral-def2*[*OF assms*(*1*,*2*)] *nn-integral-cmult*[*of f qbs-prob-measure s*] *assms*
by *fastforce*

lemma *qbs-prob-ennintegral-cmult-noninfty*:
assumes *c* ≠ ∞
shows *qbs-prob-ennintegral s* (λ*x*. *c* ∗ *f x*) = *c* ∗ *qbs-prob-ennintegral s f*
proof −
obtain *X* α μ **where** *hs*:
s = *qbs-prob-space* (*X*, α, μ) *qbs-prob X* α μ
using *rep-qbs-prob-space*[*of s*] **by** *auto*
then interpret *qp*: *qbs-prob X* α μ **by** *simp*
consider *f* ∈ *X* $\rightarrow_Q$ $\mathbb{R}_{Q\geq 0}$ | *f* ∉ *X* $\rightarrow_Q$ $\mathbb{R}_{Q\geq 0}$ **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by(*auto intro*!: *qbs-prob-ennintegral-cmult*[**where** *X*=*X*] *simp*: *hs*(*1*))
next
case *2*
consider *c* = *0* | *c* ≠ *0* ∧ *c* ≠ ∞
using *assms* **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*

by(*simp add*: *hs qbs-prob-ennintegral.abs-eq qbs-prob-t-ennintegral-def*)
next
case *h*:*2*
have $(\lambda x.\ c * f\ x) \notin X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
proof(*rule ccontr*)
assume $\neg\ (\lambda x.\ c * f\ x) \notin X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
hence *3*:$(\lambda x.\ c * f\ x) \in$ *qbs-to-measure* $X \rightarrow_M$ *ennreal-borel*
by *auto*
have $f = (\lambda x.\ (1/c) * \ (c * f\ x))$
using *h* **by**(*simp add*: *divide-eq-1-ennreal ennreal-divide-times mult.assoc mult.commute*[*of c*] *mult-divide-eq-ennreal*)
also have ... $\in$ *qbs-to-measure* $X \rightarrow_M$ *ennreal-borel*
using *3* **by** *simp*
finally show *False*
using *2* **by** *auto*
qed
thus *?thesis*
using *qp.qbs-prob-ennintegral-not-morphism 2*
by(*simp add*: *hs*(*1*))
qed
qed
qed

lemma *qbs-prob-integral-cong*:
assumes *qbs-prob-space-qbs s* = *X*
and $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x = g\ x$
shows *qbs-prob-integral s f* = *qbs-prob-integral s g*
by(*simp add*: *qbs-prob-integral-def2*) (*metis Bochner-Integration.integral-cong assms*(*1*) *assms*(*2*) *qbs-prob-measure-space*)

lemma *qbs-prob-integral-nonneg*:
assumes *qbs-prob-space-qbs s* = *X*
and $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow 0 \leq f\ x$
shows $0 \leq$ *qbs-prob-integral s f*
using *qbs-prob-measure-space*[*of s*] *assms*
by(*simp add*: *qbs-prob-integral-def2*)

lemma *qbs-prob-integral-mono*:
assumes *qbs-prob-space-qbs s* = *X*
qbs-integrable (*s* :: *'a qbs-prob-space*) *f*
qbs-integrable s g
and $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow f\ x \leq g\ x$
shows *qbs-prob-integral s f* $\leq$ *qbs-prob-integral s g*
using *integral-mono*[*OF integrable-if-qbs-integrable*[*OF assms*(*2*)] *integrable-if-qbs-integrable*[*OF assms*(*3*)] *assms*(*4*)[*simplified qbs-prob-measure-space*]]
by(*simp add*: *qbs-prob-integral-def2 assms*(*1*) *qbs-prob-measure-space*[*symmetric*])

lemma *qbs-prob-integral-const*:
qbs-prob-integral (*s*::*'a qbs-prob-space*) $(\lambda x.\ c) = c$

by(*simp add*: *qbs-prob-integral-def2 qbs-prob-measure-prob-space.prob-space*)

lemma *qbs-prob-integral-add*:
assumes *qbs-integrable* (*s*::′*a qbs-prob-space*) *f*
and *qbs-integrable s g*
shows *qbs-prob-integral s* (λ*x*. *f x* + *g x*) = *qbs-prob-integral s f* + *qbs-prob-integral s g*
using *Bochner-Integration.integral-add*[*OF integrable-if-qbs-integrable*[*OF assms*(*1*)] *integrable-if-qbs-integrable*[*OF assms*(*2*)]]
by(*simp add*: *qbs-prob-integral-def2*)

lemma *qbs-prob-integral-diff*:
assumes *qbs-integrable* (*s*::′*a qbs-prob-space*) *f*
and *qbs-integrable s g*
shows *qbs-prob-integral s* (λ*x*. *f x* − *g x*) = *qbs-prob-integral s f* − *qbs-prob-integral s g*
using *Bochner-Integration.integral-diff*[*OF integrable-if-qbs-integrable*[*OF assms*(*1*)] *integrable-if-qbs-integrable*[*OF assms*(*2*)]]
by(*simp add*: *qbs-prob-integral-def2*)

lemma *qbs-prob-integral-cmult*:
qbs-prob-integral s (λ*x*. *c* ∗ *f x*) = *c* ∗ *qbs-prob-integral s f*
by(*simp add*: *qbs-prob-integral-def2*)

lemma *real-qbs-prob-integral-def*:
assumes *qbs-integrable* (*s*::′*a qbs-prob-space*) *f*
shows *qbs-prob-integral s f* = *enn2real* (*qbs-prob-ennintegral s* (λ*x*. *ennreal* (*f x*))) − *enn2real* (*qbs-prob-ennintegral s* (λ*x*. *ennreal* (− *f x*)))
using *assms*
proof(*transfer*,*auto*)
fix *X* :: ′*a quasi-borel*
fix α μ *f*
assume *H*:*qbs-prob-eq* (*X*,α,μ) (*X*,α,μ)
qbs-prob-t-integrable (*X*,α,μ) *f*
then interpret *qp*: *qbs-prob X* α μ
by(*simp add*: *qbs-prob-eq-def*)
show *qbs-prob-t-integral* (*X*,α,μ) *f* = *enn2real* (*qbs-prob-t-ennintegral* (*X*,α,μ) (λ*x*. *ennreal* (*f x*))) − *enn2real* (*qbs-prob-t-ennintegral* (*X*,α,μ) (λ*x*. *ennreal* (− *f x*)))
using *H*(*2*) *real-lebesgue-integral-def*[*of* μ *f* ∘ α]
by(*auto simp add*: *comp-def qbs-prob-t-integrable-def qbs-prob-t-integral-def qbs-prob-t-ennintegral-def*)
qed

lemma *qbs-prob-var-eq*:
assumes *qbs-integrable* (*s*::′*a qbs-prob-space*) *f*
and *qbs-integrable s* (λ*x*. $(f\,x)^2$)
shows *qbs-prob-var s f* = *qbs-prob-integral s* (λ*x*. $(f\,x)^2$) − $(\textit{qbs-prob-integral}\ s\ f)^2$

unfolding *qbs-prob-var-def* **using** *assms*
proof(*transfer*,*auto*)
fix *X* :: *'a quasi-borel*
fix *α μ f*
assume *H*:*qbs-prob-eq* (*X*,*α*,*μ*) (*X*,*α*,*μ*)
qbs-prob-t-integrable (*X*,*α*,*μ*) *f*
qbs-prob-t-integrable (*X*,*α*,*μ*) (*λx*. (*f x*)^2)
then interpret *qp*: *qbs-prob X α μ*
by(*simp add*: *qbs-prob-eq-def*)
show *qbs-prob-t-integral* (*X*,*α*,*μ*) (*λx*. (*f x* − *qbs-prob-t-integral* (*X*,*α*,*μ*) *f*)^2) = *qbs-prob-t-integral* (*X*,*α*,*μ*) (*λx*. (*f x*)^2) − (*qbs-prob-t-integral* (*X*,*α*,*μ*) *f*)^2
using *H*(*2*,*3*) *prob-space.variance-eq*[*of μ f* ∘ *α*]
by(*auto simp add*: *qbs-prob-t-integral-def qbs-prob-def qbs-prob-t-integrable-def comp-def qbs-prob-eq-def*)
qed

lemma *qbs-prob-var-affine*:
assumes *qbs-integrable s f*
shows *qbs-prob-var s* (*λx*. *a* ∗ *f x* + *b*) = *a*^{2} ∗ *qbs-prob-var s f*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *qbs-prob-integral s* (*λx*. (*a* ∗ *f x* + *b* − (*a* ∗ *qbs-prob-integral s f* + *b*))^2)
using *qbs-prob-integral-add*[*OF qbs-integrable-mult*[*OF assms*,*of a*] *qbs-integrable-const*[*of s b*]]
by (*simp add*: *qbs-prob-integral-cmult qbs-prob-integral-const qbs-prob-var-def*)
also have ... = *qbs-prob-integral s* (*λx*. (*a* ∗ *f x* − *a* ∗ *qbs-prob-integral s f*)^2)
by *simp*
also have ... = *qbs-prob-integral s* (*λx*. *a*^{2} ∗ (*f x* − *qbs-prob-integral s f*)^2)
by (*metis power-mult-distrib right-diff-distrib*)
also have ... = *?rhs*
by(*simp add*: *qbs-prob-var-def qbs-prob-integral-cmult*[*of s a*^{2}])
finally show *?thesis* **.**
qed

lemma *qbs-prob-integral-Markov-inequality*:
assumes *qbs-prob-space-qbs s* = *X*
and *qbs-integrable s f*
⋀*x*. *x* ∈ *qbs-space X* ⟹ *0* ≤ *f x*
and *0* < *c*
shows *qbs-emeasure s* {*x* ∈ *qbs-space X*. *c* ≤ *f x*} ≤ *ennreal* (*1*/*c* ∗ *qbs-prob-integral s f*)
using *integral-Markov-inequality*[*OF integrable-if-qbs-integrable*[*OF assms*(*2*)] - *assms*(*4*)] *assms*(*3*)
by(*force simp add*: *qbs-prob-integral-def2 qbs-prob-measure-space qbs-emeasure-def assms*(*1*) *qbs-prob-measure-space*[*symmetric*])

lemma *qbs-prob-integral-Markov-inequality'*:
assumes *qbs-prob-space-qbs s* = *X*

qbs-integrable s f
$\bigwedge x.\ x \in$ *qbs-space* (*qbs-prob-space-qbs s*) $\Longrightarrow$ *0* $\leq$ *f x*
and *0* < *c*
shows *qbs-measure s* {*x* ∈ *qbs-space* (*qbs-prob-space-qbs s*). *c* ≤ *f x*} ≤ (*1*/*c* ∗ *qbs-prob-integral s f*)
using *qbs-prob-integral-Markov-inequality*[*OF assms*] *ennreal-le-iff*[*of 1*/*c* ∗ *qbs-prob-integral s f qbs-measure s* {*x* ∈ *qbs-space* (*qbs-prob-space-qbs s*). *c* ≤ *f x*}] *qbs-prob-integral-nonneg*[*of s X f*,*OF assms*(*1*,*3*)] *assms*(*4*)
by(*simp add*: *qbs-measure-def qbs-emeasure-def qbs-prob-measure-prob-space.emeasure-eq-measure assms*(*1*))

lemma *qbs-prob-integral-Markov-inequality-abs*:
assumes *qbs-prob-space-qbs s* = *X*
qbs-integrable s f
and *0* < *c*
shows *qbs-emeasure s* {*x* ∈ *qbs-space X*. *c* ≤ |*f x*|} ≤ *ennreal* (*1*/*c* ∗ *qbs-prob-integral s* (λ*x*. |*f x*|))
using *qbs-prob-integral-Markov-inequality*[*OF assms*(*1*) *qbs-integrable-abs*[*OF assms*(*2*)] - *assms*(*3*)]
by(*simp add*: *assms*(*1*))

lemma *qbs-prob-integral-Markov-inequality-abs′*:
assumes *qbs-prob-space-qbs s* = *X*
qbs-integrable s f
and *0* < *c*
shows *qbs-measure s* {*x* ∈ *qbs-space X*. *c* ≤ |*f x*|} ≤ (*1*/*c* ∗ *qbs-prob-integral s* (λ*x*. |*f x*|))
using *qbs-prob-integral-Markov-inequality′*[*OF assms*(*1*) *qbs-integrable-abs*[*OF assms*(*2*)] - *assms*(*3*)]
by(*simp add*: *assms*(*1*))

lemma *qbs-prob-integral-real-Markov-inequality*:
assumes *qbs-prob-space-qbs s* = $\mathbb{R}_Q$
qbs-integrable s f
and *0* < *c*
shows *qbs-emeasure s* {*r*. *c* ≤ |*f r*|} ≤ *ennreal* (*1*/*c* ∗ *qbs-prob-integral s* (λ*x*. |*f x*|))
using *qbs-prob-integral-Markov-inequality-abs*[*OF assms*]
by *simp*

lemma *qbs-prob-integral-real-Markov-inequality′*:
assumes *qbs-prob-space-qbs s* = $\mathbb{R}_Q$
qbs-integrable s f
and *0* < *c*
shows *qbs-measure s* {*r*. *c* ≤ |*f r*|} ≤ *1*/*c* ∗ *qbs-prob-integral s* (λ*x*. |*f x*|)
using *qbs-prob-integral-Markov-inequality-abs′*[*OF assms*]
by *simp*

lemma *qbs-prob-integral-Chebyshev-inequality*:

assumes *qbs-prob-space-qbs s* = *X*
qbs-integrable s f
qbs-integrable s ($\lambda x.$ $(f\ x)^2$)
and *0* < *b*
shows *qbs-measure s* $\{x \in$ *qbs-space X*. $b \leq |f\ x -$ *qbs-prob-integral s f*$|\} \leq$ *1* / b^2 ∗ *qbs-prob-var s f*
proof −
have *qbs-integrable s* ($\lambda x.$ $|f\ x -$ *qbs-prob-integral s f*$|^2$)
by(*simp*, *rule qbs-integrable-sq*[*OF assms*(*2*,*3*)])
moreover have $\{x \in$ *qbs-space X*. $b^2 \leq |f\ x -$ *qbs-prob-integral s f*$|^2\} = \{x \in$ *qbs-space X*. $b \leq |f\ x -$ *qbs-prob-integral s f*$|\}$
by (*metis* (*mono-tags*, *opaque-lifting*) *abs-le-square-iff abs-of-nonneg assms*(*4*) *less-imp-le power2-abs*)
ultimately show *?thesis*
using *qbs-prob-integral-Markov-inequality*′[*OF assms*(*1*),*of* $\lambda x.$ $|f\ x -$ *qbs-prob-integral s f*$|$^*2* *b*^*2*] *assms*(*4*)
by(*simp add*: *qbs-prob-var-def assms*(*1*))
qed

end

3.2 The Probability Monad

theory *Monad-QuasiBorel*
imports *Probability-Space-QuasiBorel*
begin

3.2.1 The Probability Monad *P*

definition *monadP-qbs-Px* :: ′*a quasi-borel* ⇒ ′*a qbs-prob-space set* **where**
monadP-qbs-Px X ≡ {*s*. *qbs-prob-space-qbs s* = *X*}

locale *in-Px* =
fixes *X* :: ′*a quasi-borel* **and** *s* :: ′*a qbs-prob-space*
assumes *in-Px*:*s* ∈ *monadP-qbs-Px X*
begin

lemma *qbs-prob-space-X*[*simp*]:
qbs-prob-space-qbs s = *X*
using *in-Px* **by**(*simp add*: *monadP-qbs-Px-def*)

end

locale *in-MPx* =
fixes *X* :: ′*a quasi-borel* **and** β :: *real* ⇒ ′*a qbs-prob-space*
assumes *ex*:∃ α∈ *qbs-Mx X*. ∃ *g* ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*.
∀ *r*. β *r* = *qbs-prob-space* (*X*,α,*g r*)
begin

lemma *rep-inMPx*:

$\exists\alpha\ g.\ \alpha \in qbs\text{-}Mx\ X \wedge g \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel \wedge$
$\beta = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X,\alpha,g\ r))$
proof −
obtain *α g* **where** *hb*:
$\alpha \in qbs\text{-}Mx\ X\ g \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel$
$\beta = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X,\alpha,g\ r))$
using *ex* **by** *auto*
thus *?thesis*
by(*auto intro*!: *exI*[**where** *x*=*α*] *exI*[**where** *x*=*g*] *simp*: *hb*)
qed

end

definition *monadP-qbs-MPx* :: *'a quasi-borel* ⇒ (*real* ⇒ *'a qbs-prob-space*) *set*
where
monadP-qbs-MPx X ≡ {*β. in-MPx X β*}

definition *monadP-qbs* :: *'a quasi-borel* ⇒ *'a qbs-prob-space quasi-borel* **where**
monadP-qbs X ≡ *Abs-quasi-borel* (*monadP-qbs-Px X*, *monadP-qbs-MPx X*)

lemma(**in** *qbs-prob*) *qbs-prob-space-in-Px*:
qbs-prob-space (*X*,*α*,*μ*) ∈ *monadP-qbs-Px X*
using *qbs-prob-axioms* **by**(*simp add*: *monadP-qbs-Px-def*)

lemma *rep-monadP-qbs-Px*:
assumes *s* ∈ *monadP-qbs-Px X*
shows ∃*α μ. s* = *qbs-prob-space* (*X*, *α*, *μ*) ∧ *qbs-prob X α μ*
using *rep-qbs-prob-space' assms in-Px.qbs-prob-space-X*
by(*auto simp*: *monadP-qbs-Px-def*)

lemma *rep-monadP-qbs-MPx*:
assumes *β* ∈ *monadP-qbs-MPx X*
shows $\exists\alpha\ g.\ \alpha \in qbs\text{-}Mx\ X \wedge g \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel \wedge$
$\beta = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X,\alpha,g\ r))$
using *assms in-MPx.rep-inMPx*
by(*auto simp*: *monadP-qbs-MPx-def*)

lemma *qbs-prob-MPx*:
assumes *α* ∈ *qbs-Mx X*
and $g \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel$
shows *qbs-prob X α* (*g r*)
using *measurable-space*[*OF assms*(*2*)]
by(*auto intro*!: *qbs-prob.intro simp*: *space-prob-algebra in-Mx-def real-distribution-def real-distribution-axioms-def assms*(*1*))

lemma *monadP-qbs-f*[*simp*]: *monadP-qbs-MPx X* ⊆ *UNIV* → *monadP-qbs-Px X*
unfolding *monadP-qbs-MPx-def*
proof *auto*
fix *β r*

assume *in-MPx X β*
then obtain *α g* **where** *hb*:
α ∈ qbs-Mx X g ∈ real-borel $\rightarrow_M$ *prob-algebra real-borel*
β = (*λr. qbs-prob-space* (*X*,*α*,*g r*))
using *in-MPx.rep-inMPx* **by** *blast*
then interpret *qp* : *qbs-prob X α g r*
by(*simp add*: *qbs-prob-MPx*)
show *β r ∈ monadP-qbs-Px X*
by(*simp add*: *hb*(*3*) *qp.qbs-prob-space-in-Px*)
qed

lemma *monadP-qbs-closed1*: *qbs-closed1* (*monadP-qbs-MPx X*)
unfolding *monadP-qbs-MPx-def in-MPx-def*
apply(*rule qbs-closed1I*)
subgoal for *α f*
apply *auto*
subgoal for *β g*
apply(*auto intro*!: *bexI*[**where** *x*=*β*] *bexI*[**where** *x*=*g*∘*f*])
done
done
done

lemma *monadP-qbs-closed2*: *qbs-closed2* (*monadP-qbs-Px X*) (*monadP-qbs-MPx X*)
unfolding *qbs-closed2-def*
proof
fix *s*
assume *s ∈ monadP-qbs-Px X*
then obtain *α μ* **where** *h*:
qbs-prob X α μ s = *qbs-prob-space* (*X*, *α*, *μ*)
using *rep-qbs-prob-space*′[*of s X*] *monadP-qbs-Px-def* **by** *blast*
then interpret *qp* : *qbs-prob X α μ*
by *simp*
define *g* :: *real* ⇒ *real measure*
where *g* ≡ (*λ-. μ*)
have *g ∈ real-borel* $\rightarrow_M$ *prob-algebra real-borel*
using *h prob-algebra-real-prob-measure*[*of μ*]
by(*simp add*: *qbs-prob-def g-def*)
thus (*λr. s*) ∈ *monadP-qbs-MPx X*
by(*auto intro*!: *bexI*[**where** *x*=*α*] *bexI*[**where** *x*=*g*] *simp*: *monadP-qbs-MPx-def in-MPx-def g-def h*)
qed

lemma *monadP-qbs-closed3*: *qbs-closed3* (*monadP-qbs-MPx* (*X* :: ′*a quasi-borel*))
proof(*rule qbs-closed3I*)
fix *P* :: *real* ⇒ *nat*
fix *Fi*
assume ⋀*i. P* −‘ {*i*} ∈ *sets real-borel*
then have *HP-mble*[*measurable*] : *P ∈ real-borel* $\rightarrow_M$ *nat-borel*

by (*simp add*: *separate-measurable*)
assume $\bigwedge i :: nat.\ Fi\ i \in monadP\text{-}qbs\text{-}MPx\ X$
then have $\forall i.\ \exists \alpha i.\ \exists gi.\ \alpha i \in qbs\text{-}Mx\ X \wedge gi \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel \wedge$
$Fi\ i = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha i, gi\ r))$
using *in-MPx.rep-inMPx*[*of X*] **by**(*simp add*: *monadP-qbs-MPx-def*)
hence $\exists \alpha.\ \forall i.\ \exists gi.\ \alpha\ i \in qbs\text{-}Mx\ X \wedge gi \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel \wedge$
$Fi\ i = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha\ i, gi\ r))$
by(*rule choice*)
then obtain $\alpha :: nat \Rightarrow real \Rightarrow$ - **where**
$\forall i.\ \exists gi.\ \alpha\ i \in qbs\text{-}Mx\ X \wedge gi \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel \wedge$
$Fi\ i = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha\ i, gi\ r))$
by *auto*
hence $\exists g.\ \forall i.\ \alpha\ i \in qbs\text{-}Mx\ X \wedge g\ i \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel \wedge$
$Fi\ i = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha\ i, g\ i\ r))$
by(*rule choice*)
then obtain $g :: nat \Rightarrow real \Rightarrow real\ measure$ **where**
H0: $\bigwedge i.\ \alpha\ i \in qbs\text{-}Mx\ X\ \bigwedge i.\ g\ i \in real\text{-}borel \to_M prob\text{-}algebra\ real\text{-}borel$
$\bigwedge i.\ Fi\ i = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha\ i, g\ i\ r))$
by *blast*
hence *LHS*: $(\lambda r.\ Fi\ (P\ r)\ r) = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha\ (P\ r), g\ (P\ r)\ r))$
by *auto*

— Since $\mathbb{N}\times\mathbb{R}$ is standard, we have measurable functions $nat\text{-}real.f \in \mathbb{N} \bigotimes_M \mathbb{R} \to_M \mathbb{R}$ and $nat\text{-}real.g \in \mathbb{R} \to_M \mathbb{N} \bigotimes_M \mathbb{R}$ such that $nat\text{-}real.g \circ nat\text{-}real.f = id$.

— The proof is divided into 3 steps.

1. Let $\alpha'' = uncurry\ \alpha \circ nat\text{-}real.g$. Then $\alpha'' \in qbs\text{-}Mx\ X$.
2. Let $g'' = G(nat\text{-}real.f) \circ (\lambda r.\ \delta_{P(r)} \bigotimes_M g_{P(r)}\ r$. Then g'' is $\mathbb{R}/G(\mathbb{R})$ measurable.
3. Show that $(\lambda r.\ Fi\ (P\ r)\ r) = (\lambda r.\ qbs\text{-}prob\text{-}space\ (X, \alpha'', g''\ r))$.

— Step 1.
define $\alpha' :: nat \times real \Rightarrow 'a$
where $\alpha' \equiv case\text{-}prod\ \alpha$
define $\alpha'' :: real \Rightarrow 'a$
where $\alpha'' \equiv \alpha' \circ nat\text{-}real.g$

have α-*morp*: $\alpha \in \mathbb{N}_Q \to_Q exp\text{-}qbs\ \mathbb{R}_Q\ X$
using *qbs-Mx-is-morphisms*[*of X*] *H0*(*1*)
by(*auto intro*!: *nat-qbs-morphism*)
hence α'-*morp*: $\alpha' \in \mathbb{N}_Q \bigotimes_Q \mathbb{R}_Q \to_Q X$
unfolding α'-*def*
by(*intro uncurry-preserves-morphisms*)
hence [*measurable*]:$\alpha' \in nat\text{-}borel \bigotimes_M real\text{-}borel \to_M qbs\text{-}to\text{-}measure\ X$
using *l-preserves-morphisms*[*of* $\mathbb{N}_Q \bigotimes_Q \mathbb{R}_Q\ X$]

by(*auto simp add*: *r-preserves-product*)
have *H-Mx*:α'' ∈ *qbs-Mx X*
unfolding α''*-def*
using *qbs-morphism-comp*[*OF real.qbs-morphism-measurable-intro*[*OF nat-real.g-meas*,*simplified r-preserves-product*] α'*-morp*] *qbs-Mx-is-morphisms*[*of X*]
by *simp*

— Step 2.
define g' :: *real* ⇒ (*nat* × *real*) *measure*
where $g' \equiv (\lambda r.$ *return nat-borel* $(P\ r) \bigotimes_M$ $g\ (P\ r)\ r)$
define g'' :: *real* ⇒ *real measure*
where $g'' \equiv (\lambda M.$ *distr M real-borel nat-real.f*$) \circ g'$

have [*measurable*]:($\lambda nr.\ g$ (*fst nr*) (*snd nr*)) ∈ *nat-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
using *measurable-pair-measure-countable1*[*of UNIV* :: *nat set* $\lambda nr.\ g$ (*fst nr*) (*snd nr*),*simplified*,*OF H0*(*2*)] *measurable-cong-sets*[*OF sets-pair-measure-cong*[*of nat-borel count-space UNIV real-borel real-borel*,*OF sets-borel-eq-count-space refl*] *refl*,*of prob-algebra real-borel*]
by *auto*
hence [*measurable*]:($\lambda r.\ g\ (P\ r)\ r$) ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
proof −
have ($\lambda r.\ g\ (P\ r)\ r$) = ($\lambda nr.\ g$ (*fst nr*) (*snd nr*)) ∘ ($\lambda r.$ ($P\ r$, r)) **by** *auto*
also have ... ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
by *simp*
finally show *?thesis* **.**
qed
have g'*-mble*[*measurable*]:g' ∈ *real-borel* $\rightarrow_M$ *prob-algebra* (*nat-borel* $\bigotimes_M$ *real-borel*)
unfolding g'*-def* **by** *simp*
have *H-mble*: g'' ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
unfolding g''*-def* **by** *simp*

— Step 3.
have *H-equiv*:
qbs-prob-space (X, α ($P\ r$), g ($P\ r$) r) = *qbs-prob-space* (X, α'', $g''\ r$) **for** r
proof −
interpret *pqp*: *pair-qbs-prob X* α ($P\ r$) g ($P\ r$) r X α'' g'' r
using *qbs-prob-MPx*[*OF H0*(*1*,*2*)] *measurable-space*[*OF H-mble*,*of r*] *space-prob-algebra*[*of real-borel*] *H-Mx*
by (*simp add*: *pair-qbs-prob.intro qbs-probI*)
interpret *pps*: *pair-prob-space return nat-borel* ($P\ r$) g ($P\ r$) r
using *prob-space-return*[*of P r nat-borel*]
by(*simp add*: *pair-prob-space-def pair-sigma-finite-def prob-space-imp-sigma-finite*)
have [*measurable-cong*]: *sets* (*return nat-borel* ($P\ r$)) = *sets nat-borel*
sets ($g'\ r$) = *sets* (*nat-borel* $\bigotimes_M$ *real-borel*)
using *measurable-space*[*OF* g'*-mble*,*of r*] *space-prob-algebra* **by** *auto*
show *qbs-prob-space* (X, α ($P\ r$), g ($P\ r$) r) = *qbs-prob-space* (X, α'', $g''\ r$)
proof(*rule pqp.qbs-prob-space-eq4*)

fix f
assume [*measurable*]:$f \in$ *qbs-to-measure* $X \rightarrow_M$ *ennreal-borel*
show $(\int^+ x.\ f\ (\alpha\ (P\ r)\ x)\ \partial g\ (P\ r)\ r) = (\int^+ x.\ f\ (\alpha''\ x)\ \partial g''\ r)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+ s.\ f\ (\alpha'\ ((P\ r),s))\ \partial\ (g\ (P\ r)\ r))$
by(*simp add*: α'-*def*)
also have ... = $(\int^+ s.\ (\int^+ i.\ f\ (\alpha'\ (i,\ s))\ \partial$ (*return nat-borel* $(P\ r)))\ \ \partial\ (g\ (P\ r)\ r))$
by(*auto intro*!: *nn-integral-cong simp*: *nn-integral-return*[*of P r nat-borel*])
also have ... = $(\int^+ k.\ (f \circ \alpha')\ k\ \partial$ ((*return nat-borel* $(P\ r)) \bigotimes_M (g\ (P\ r)\ r)))$
by(*auto intro*!: *pps.nn-integral-snd*)
also have ... = $(\int^+ k.\ f\ (\alpha'\ k)\ \partial\ (g'\ r))$
by(*simp add*: g'-*def*)
also have ... = $(\int^+ x.\ f\ x\ \partial$ (*distr* $(g'\ r)$ (*qbs-to-measure* X) α'))
by(*simp add*: *nn-integral-distr*)
also have ... = $(\int^+ x.\ f\ x\ \partial$ (*distr* $(g''\ r)$ (*qbs-to-measure* X) α''))
by(*simp add*: *distr-distr comp-def* g''-*def* α''-*def*)
also have ... = *?rhs*
by(*simp add*: *nn-integral-distr*)
finally show *?thesis* **.**
qed
qed *simp*
qed

have $\forall r.$ *Fi* $(P\ r)\ r$ = *qbs-prob-space* $(X,\ \alpha'',\ g''\ r)$
by (*metis H-equiv LHS*)
thus $(\lambda r.$ *Fi* $(P\ r)\ r) \in$ *monadP-qbs-MPx X*
using *H-mble H-Mx* **by**(*auto simp add*: *monadP-qbs-MPx-def in-MPx-def*)
qed

lemma *monadP-qbs-correct*: *Rep-quasi-borel* (*monadP-qbs X*) = (*monadP-qbs-Px X*, *monadP-qbs-MPx X*)
by(*auto intro*!: *Abs-quasi-borel-inverse monadP-qbs-f simp*: *monadP-qbs-closed2 monadP-qbs-closed1 monadP-qbs-closed3 monadP-qbs-def*)

lemma *monadP-qbs-space*[*simp*] : *qbs-space* (*monadP-qbs X*) = *monadP-qbs-Px X*
by(*simp add*: *qbs-space-def monadP-qbs-correct*)

lemma *monadP-qbs-Mx*[*simp*] : *qbs-Mx* (*monadP-qbs X*) = *monadP-qbs-MPx X*
by(*simp add*: *qbs-Mx-def monadP-qbs-correct*)

lemma *monadP-qbs-empty-iff*:
qbs-space X = {} $\longleftrightarrow$ *qbs-space* (*monadP-qbs X*) = {}
proof *auto*
fix x
assume *1*:*qbs-space X* = {}
$x \in$ *monadP-qbs-Px X*

then obtain α μ **where** *qbs-prob X* α μ
  using *rep-monadP-qbs-Px* **by** *blast*
 thus *False*
  using *empty-quasi-borel-iff[of X] qbs-empty-not-qbs-prob[of* α μ*] 1(1)*
  by *auto*
next
 fix *x*
 assume *1:monadP-qbs-Px X* = {}
     x ∈ *qbs-space X*
 then interpret *qp*: *qbs-prob X* λr. *x return real-borel 0*
  by(*auto intro*!: *qbs-probI prob-space-return*)
 have *qbs-prob-space* (*X*,λr. *x*,*return real-borel 0*) ∈ *monadP-qbs-Px X*
  by(*simp add*: *monadP-qbs-Px-def*)
 thus *False*
  by(*simp add*: *1*)
qed

If $\beta \in MPx$, there exists X α g s.t.β r = [X,α,g r]. We define a function which picks X α g from $\beta \in MPx$.

definition *rep-monadP-qbs-MPx* :: (*real* ⇒ *'a qbs-prob-space*) ⇒ *'a quasi-borel* × (*real* ⇒ *'a*) × (*real* ⇒ *real measure*) **where**
rep-monadP-qbs-MPx β ≡ *let X* = *qbs-prob-space-qbs* (β *undefined*);
    αg = (*SOME k*. (*fst k*) ∈ *qbs-Mx X* ∧ (*snd k*) ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
     ∧ β = (λr. *qbs-prob-space* (*X*,*fst k*,*snd k r*)))
   in (*X*,αg)

lemma *qbs-prob-measure-measurable*[*measurable*]:
qbs-prob-measure ∈ *qbs-to-measure* (*monadP-qbs* (*X* :: *'a quasi-borel*)) $\rightarrow_M$ *prob-algebra* (*qbs-to-measure X*)
proof(*rule qbs-morphism-dest*,*rule qbs-morphismI*,*simp*)
 fix β
 assume β ∈ *monadP-qbs-MPx X*
 then obtain α *g* **where** *hb*:
 α ∈ *qbs-Mx X* β = (λr. *qbs-prob-space* (*X*, α, *g r*))
  and *g*[*measurable*]: *g* ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
  using *in-MPx.rep-inMPx*[*of X* β] *monadP-qbs-MPx-def* **by** *blast*
 have *qbs-prob-measure* ∘ β = (λr. *distr* (*g r*) (*qbs-to-measure X*) α)
 proof
  fix *r*
  interpret *qp* : *qbs-prob X* α *g r*
   using *qbs-prob-MPx*[*OF hb*(*1*) *g*] **by** *simp*
  show (*qbs-prob-measure* ∘ β) *r* = *distr* (*g r*) (*qbs-to-measure X*) α
   by(*simp add*: *hb*(*2*))
 qed
 also have ... ∈ *real-borel* $\rightarrow_M$ *prob-algebra* (*qbs-to-measure X*)
  using *hb* **by** *simp*
 finally show *qbs-prob-measure* ∘ β ∈ *real-borel* $\rightarrow_M$ *prob-algebra* (*qbs-to-measure X*) **.**

qed

lemma *qbs-l-inj*:
inj-on qbs-prob-measure (*monadP-qbs-Px X*)
apply *standard*
apply (*unfold monadP-qbs-Px-def*)
apply *simp*
apply *transfer*
apply (*auto simp*: *qbs-prob-eq-def qbs-prob-t-measure-def*)
done

lemma *qbs-prob-measure-measurable*′[*measurable*]:
qbs-prob-measure ∈ *qbs-to-measure* (*monadP-qbs* (*X* :: ′*a quasi-borel*)) $\rightarrow_M$ *sub-prob-algebra* (*qbs-to-measure X*)
by(*auto simp*: *measurable-prob-algebraD*)

3.2.2 Return

definition *qbs-return* :: [′*a quasi-borel*, ′*a*] ⇒ ′*a qbs-prob-space* **where**
qbs-return X x ≡ *qbs-prob-space* (*X*,*λr. x*,*Eps real-distribution*)

lemma(**in** *real-distribution*) *qbs-return-qbs-prob*:
assumes *x* ∈ *qbs-space X*
shows *qbs-prob X* (*λr. x*) *M*
using *assms*
by(*simp add*: *qbs-prob-def in-Mx-def real-distribution-axioms*)

lemma(**in** *real-distribution*) *qbs-return-computation* :
assumes *x* ∈ *qbs-space X*
shows *qbs-return X x* = *qbs-prob-space* (*X*,*λr. x*,*M*)
unfolding *qbs-return-def*
proof(*rule someI2*[**where** *a*=*M*])
fix *N*
assume *real-distribution N*
then interpret *pqp*: *pair-qbs-prob X λr. x N X λr. x M*
by(*simp-all add*: *pair-qbs-prob-def real-distribution-axioms real-distribution.qbs-return-qbs-prob*[*OF* - *assms*])
show *qbs-prob-space* (*X*, *λr. x*, *N*) = *qbs-prob-space* (*X*, *λr. x*, *M*)
by(*auto intro*!: *pqp.qbs-prob-space-eq simp*: *distr-const*[*of x qbs-to-measure X*] *assms*)
qed (*rule real-distribution-axioms*)

lemma *qbs-return-morphism*:
qbs-return X ∈ *X* $\rightarrow_Q$ *monadP-qbs X*
proof −
interpret *rr* : *real-distribution return real-borel 0*
by(*simp add*: *real-distribution-def real-distribution-axioms-def prob-space-return*)
show *?thesis*
proof(*rule qbs-morphismI*,*simp*)

fix α
assume h:$\alpha \in$ *qbs-Mx X*
then have h':$\bigwedge l$:: *real*. α $l \in$ *qbs-space X*
by *auto*
have $\bigwedge l$. (*qbs-return X* $\circ$ α) l = *qbs-prob-space* (X, α, *return real-borel l*)
proof −
fix l
interpret *pqp*: *pair-qbs-prob X* λr. α *l return real-borel 0 X* α *return real-borel l*
using h' **by**(*simp add*: *pair-qbs-prob-def qbs-prob-def in-Mx-def h real-distribution-def prob-space-return real-distribution-axioms-def*)
have (*qbs-return X* $\circ$ α) l = *qbs-prob-space* (X,λr. α l,*return real-borel 0*)
using *rr.qbs-return-computation*[*OF h'*[*of l*]] **by** *simp*
also have ... = *qbs-prob-space* (X, α, *return real-borel l*)
by(*auto intro*!: *pqp.qbs-prob-space-eq simp*: *distr-return*)
finally show (*qbs-return X* $\circ$ α) l = *qbs-prob-space* (X, α, *return real-borel l*) .
qed
thus *qbs-return X* $\circ$ $\alpha \in$ *monadP-qbs-MPx X*
by(*auto intro*!: *bexI*[**where** x=α] *bexI*[**where** x=λl. *return real-borel l*] *simp*: *h monadP-qbs-MPx-def in-MPx-def*)
qed
qed

lemma *qbs-return-morphism'*:
assumes $f \in X \rightarrow_Q Y$
shows (λx. *qbs-return Y* (f x)) $\in X \rightarrow_Q$ *monadP-qbs Y*
using *qbs-morphism-comp*[*OF assms*(*1*) *qbs-return-morphism*[*of Y*]]
by (*simp add*: *comp-def*)

3.2.3 Bind

definition *qbs-bind* :: *'a qbs-prob-space* ⇒ (*'a* ⇒ *'b qbs-prob-space*) ⇒ *'b qbs-prob-space*
where
qbs-bind s f ≡ (*let* (*qbsx*,α,μ) = *rep-qbs-prob-space s*;
(*qbsy*,β,*g*) = *rep-monadP-qbs-MPx* ($f \circ \alpha$)
in qbs-prob-space (*qbsy*,β,$\mu \gg= g$))

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

lemma(**in** *qbs-prob*) *qbs-bind-computation*:
assumes s = *qbs-prob-space* (X,α,μ)
$f \in X \rightarrow_Q$ *monadP-qbs Y*
$\beta \in$ *qbs-Mx Y*
and [*measurable*]: $g \in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
and ($f \circ \alpha$) = (λr. *qbs-prob-space* (Y,β, g r))
shows *qbs-prob Y* β ($\mu \gg= g$)
$s \gg= f$ = *qbs-prob-space* (Y,β,$\mu \gg= g$)
proof −

interpret *qp-bind*: *qbs-prob Y β μ ≫= g*
using *assms(3,4) space-prob-algebra[of real-borel] measurable-space[OF assms(4)] events-eq-borel measurable-cong-sets[OF events-eq-borel refl,of subprob-algebra real-borel] measurable-prob-algebraD[OF assms(4)]*
by(*auto intro*!: *prob-space-bind[of g real-borel] simp*: *qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def*)
show *qbs-prob Y β (μ ≫= g)*
by (*rule qp-bind.qbs-prob-axioms*)
show *s ≫= f = qbs-prob-space (Y, β, μ ≫= g)*
apply(*simp add*: *assms(1) qbs-bind-def rep-qbs-prob-space-def qbs-prob-space.rep-def*)
apply(*rule someI2*[**where** *a*= (*X*, *α*, *μ*)])
proof *auto*
fix *X′ α′ μ′*
assume *h′*:(*X′*,*α′*,*μ′*) ∈ *Rep-qbs-prob-space* (*qbs-prob-space* (*X*, *α*, *μ*))
from *if-in-Rep[OF this]* **interpret** *pqp1*: *pair-qbs-prob X α μ X′ α′ μ′*
by(*simp add*: *pair-qbs-prob-def qbs-prob-axioms*)
have *h-eq*: *qbs-prob-space* (*X*, *α*, *μ*) = *qbs-prob-space* (*X′*,*α′*,*μ′*)
using *if-in-Rep(3)[OF h′]* **by** (*simp add*: *qbs-prob-space-eq*)
note [*simp*] = *if-in-Rep(1)[OF h′]*
then obtain *β′ g′* **where** *hb′*:
β′ ∈ qbs-Mx Y g′ ∈ real-borel $\rightarrow_M$ *prob-algebra real-borel*
f ∘ α′ = (*λr. qbs-prob-space* (*Y*, *β′*, *g′ r*))
using *in-MPx.rep-inMPx[of Y f ∘ α′] qbs-morphismE(3)[OF assms(2),of α′] pqp1.qp2.qbs-prob-axioms[simplified qbs-prob-def in-Mx-def]*
by(*auto simp*: *monadP-qbs-MPx-def*)
note [*measurable*] = *hb′(2)*
have [*simp*]:⋀*r. qbs-prob-space-qbs* (*f* (*α′ r*)) = *Y*
subgoal for *r*
using *fun-cong[OF hb′(3)] qbs-prob.qbs-prob-space-qbs-computation[OF qbs-prob-MPx[OF hb′(1,2),of r]]*
by *simp*
done
show (*case rep-monadP-qbs-MPx* (*λa. f* (*α′ a*)) *of* (*qbsy*, *β*, *g*) ⇒ *qbs-prob-space* (*qbsy*, *β*, *μ′ ≫= g*)) =
qbs-prob-space (*Y*, *β*, *μ ≫= g*)
unfolding *rep-monadP-qbs-MPx-def Let-def*
proof(*rule someI2*[**where** *a*=(*β′*,*g′*)],*auto simp*: *hb′[simplified comp-def]*)
fix *α″ g″*
assume *h″*:*α″ ∈ qbs-Mx Y*
g″ ∈ real-borel $\rightarrow_M$ *prob-algebra real-borel*
(*λr. qbs-prob-space* (*Y*, *β′*, *g′ r*)) = (*λr. qbs-prob-space* (*Y*, *α″*, *g″ r*))
then interpret *pqp2*: *pair-qbs-prob Y α″ μ′ ≫= g″ Y β μ ≫= g*
using *space-prob-algebra[of real-borel] measurable-space[OF h″(2)] events-eq-borel measurable-cong-sets[OF events-eq-borel refl,of subprob-algebra real-borel] measurable-prob-algebraD[OF h″(2)] h″(3)*
by (*meson pair-qbs-prob-def in-Mx-def pqp1.qp2.real-distribution-axioms prob-algebra-real-prob-measure prob-space-bind′ qbs-probI qbs-prob-def qp-bind.qbs-prob-axioms sets-bind′*)

note [*measurable*] = $h''(2)$
have [*measurable*]:$f \in$ *qbs-to-measure* $X \rightarrow_M$ *qbs-to-measure* (*monadP-qbs* Y)
using *assms*(*2*) *l-preserves-morphisms* **by** *auto*
show *qbs-prob-space* (Y, α'', $\mu' \gg= g''$) = *qbs-prob-space* (Y, β, $\mu \gg= g$)
proof(*rule pqp2.qbs-prob-space-eq*)
show *distr* ($\mu' \gg= g''$) (*qbs-to-measure* Y) α'' = *distr* ($\mu \gg= g$) (*qbs-to-measure* Y) β
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $\mu' \gg=$ (λx. *distr* (g'' x) (*qbs-to-measure* Y) α'')
by(*auto intro*!: *distr-bind*[**where** *K*=*real-borel*] *simp*: *measurable-prob-algebraD*)
also have ... = $\mu' \gg=$ (λx. *qbs-prob-measure* (*qbs-prob-space* (Y,α'',g'' x)))
by(*auto intro*!: *bind-cong simp*: *qbs-prob-MPx*[*OF* h''(*1*,*2*)] *qbs-prob.qbs-prob-measure-computation*)
also have ... = $\mu' \gg=$ (λx. (*qbs-prob-measure* (($f \circ \alpha'$) x)))
by(*simp add*: hb'(*3*) h''(*3*))
also have ... = $\mu' \gg=$ (λx. (*qbs-prob-measure* $\circ$ f) (α' x))
by(*simp add*: *comp-def*)
also have ... = *distr* μ' (*qbs-to-measure* X) $\alpha' \gg=$ *qbs-prob-measure* $\circ$ f
by(*rule bind-distr*[**where** *K*=*qbs-to-measure* Y,*symmetric*],*auto*)
also have ... = *distr* μ (*qbs-to-measure* X) $\alpha \gg=$ *qbs-prob-measure* $\circ$ f
using *pqp1.qbs-prob-space-eq-inverse*(*1*)[*OF h-eq*]
by(*simp add*: *qbs-prob-eq-def*)
also have ... = $\mu \gg=$ (λx. (*qbs-prob-measure* $\circ$ f) (α x))
by(*rule bind-distr*[**where** *K*=*qbs-to-measure* Y],*auto*)
also have ... = $\mu \gg=$ (λx. (*qbs-prob-measure* (($f \circ \alpha$) x)))
by(*simp add*: *comp-def*)
also have ... = $\mu \gg=$ (λx. *qbs-prob-measure* (*qbs-prob-space* (Y,β,g x)))
by(*auto simp*: *assms*(*5*))
also have ... = $\mu \gg=$ (λx. *distr* (g x) (*qbs-to-measure* Y) β)
by(*auto intro*!: *bind-cong simp*: *qbs-prob-MPx*[*OF assms*(*3*)] *qbs-prob.qbs-prob-measure-computation*)
also have ... = *?rhs*
by(*auto intro*!: *distr-bind*[**where** *K*=*real-borel*,*symmetric*] *simp*: *measurable-prob-algebraD*)
finally show *?thesis* **.**
qed
qed *simp*
qed
qed (*rule in-Rep*)
qed

lemma *qbs-bind-morphism$'$*:
assumes $f \in X \rightarrow_Q$ *monadP-qbs* Y
shows (λx. $x \gg= f$) $\in$ *monadP-qbs* $X \rightarrow_Q$ *monadP-qbs* Y
proof(*rule qbs-morphismI*,*simp*)
fix β
assume $\beta \in$ *monadP-qbs-MPx* X
then obtain α g **where** *hb*:
$\alpha \in$ *qbs-Mx* X $g \in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*

```
  β = (λr. qbs-prob-space (X, α, g r))
    using rep-monadP-qbs-MPx by blast
  obtain γ g' where hc:
    γ ∈ qbs-Mx Y g' ∈ real-borel →M prob-algebra real-borel
    f ∘ α = (λr. qbs-prob-space (Y, γ, g' r))
    using rep-monadP-qbs-MPx[of f ∘ α Y] qbs-morphismE(3)[OF assms hb(1),simplified]
    by auto
  note [measurable] = hb(2) hc(2)
  show (λx. x ≫= f) ∘ β ∈ monadP-qbs-MPx Y
  proof −
    have (λx. x ≫= f) ∘ β = (λr. β r ≫= f)
      by auto
    also have ... ∈ monadP-qbs-MPx Y
      unfolding monadP-qbs-MPx-def in-MPx-def
      by(auto intro!: bexI[where x=γ] bexI[where x=λr. g r ≫= g'] simp: hc(1)
hb(3) qbs-prob.qbs-bind-computation[OF qbs-prob-MPx[OF hb(1,2)] - assms hc])
    finally show ?thesis .
  qed
qed

lemma qbs-return-comp:
  assumes α ∈ qbs-Mx X
  shows (qbs-return X ∘ α) = (λr. qbs-prob-space (X,α,return real-borel r))
proof
  fix r
  interpret pqp: pair-qbs-prob X λk. α r return real-borel 0 X α return real-borel r
    by(simp add: assms qbs-Mx-to-X(2)[OF assms] pair-qbs-prob-def qbs-prob-def
in-Mx-def real-distribution-def real-distribution-axioms-def prob-space-return)
  show (qbs-return X ∘ α) r = qbs-prob-space (X, α, return real-borel r)
   by(auto intro!: pqp.qbs-prob-space-eq simp: distr-return pqp.qp1.qbs-return-computation
qbs-Mx-to-X(2)[OF assms])
qed

lemma qbs-bind-return':
  assumes x ∈ monadP-qbs-Px X
  shows x ≫= qbs-return X = x
proof −
  obtain α μ where h1:qbs-prob X α μ x = qbs-prob-space (X, α, μ)
    using assms rep-monadP-qbs-Px by blast
  then interpret qp: qbs-prob X α μ
    by simp
  show ?thesis
   using qp.qbs-bind-computation[OF h1(2) qbs-return-morphism - measurable-return-prob-space
qbs-return-comp[OF qp.in-Mx]]
    by(simp add: h1(2) bind-return'' prob-space-return qbs-probI)
qed

lemma qbs-bind-return:
  assumes f ∈ X →Q monadP-qbs Y
```

and *x* ∈ *qbs-space X*
shows *qbs-return X x* ⪢= *f* = *f x*
proof −
have *f x* ∈ *monadP-qbs-Px Y*
using *assms* **by** *auto*
then obtain *β μ* **where** *hf*:*qbs-prob Y β μ f x* = *qbs-prob-space* (*Y*, *β*, *μ*)
using *rep-monadP-qbs-Px* **by** *blast*
then interpret *rd*: *real-distribution return real-borel 0*
by(*simp add*: *qbs-prob-def prob-space-return real-distribution-def real-distribution-axioms-def*)
interpret *rd′*: *real-distribution μ*
using *hf*(*1*) **by**(*simp add*: *qbs-prob-def*)
interpret *qp*: *qbs-prob X λr. x return real-borel 0*
using *assms*(*2*) **by**(*auto simp*: *qbs-prob-def in-Mx-def rd.real-distribution-axioms*)
show *?thesis*
by(*auto intro*!: *qp.qbs-bind-computation*(*2*)[*OF rd.qbs-return-computation*[*OF assms*(*2*)] *assms*(*1*) - *measurable-const*[*of μ*],*of β*,*simplified bind-const′*[*OF rd.prob-space-axioms rd′.subprob-space-axioms*]]
simp: *hf*[*simplified qbs-prob-def in-Mx-def*] *prob-algebra-real-prob-measure*)
qed

lemma *qbs-bind-assoc*:
assumes *s* ∈ *monadP-qbs-Px X*
f ∈ $X \to_Q$ *monadP-qbs Y*
and *g* ∈ $Y \to_Q$ *monadP-qbs Z*
shows *s* ⪢= (*λx. f x* ⪢= *g*) = (*s* ⪢= *f*) ⪢= *g*
proof −
obtain *α μ* **where** *H0*:*qbs-prob X α μ s* = *qbs-prob-space* (*X*, *α*, *μ*)
using *assms rep-monadP-qbs-Px* **by** *blast*
then have *f* ∘ *α* ∈ *monadP-qbs-MPx Y*
using *assms*(*2*) **by**(*auto simp*: *qbs-prob-def in-Mx-def*)
from *rep-monadP-qbs-MPx*[*OF this*] **obtain** *β g1* **where** *H1*:
β ∈ *qbs-Mx Y g1* ∈ *real-borel* $\to_M$ *prob-algebra real-borel*
(*f* ∘ *α*) = (*λr. qbs-prob-space* (*Y*, *β*, *g1 r*))
by *auto*
hence *g* ∘ *β* ∈ *monadP-qbs-MPx Z*
using *assms* **by**(*simp add*: *qbs-morphism-def*)
from *rep-monadP-qbs-MPx*[*OF this*] **obtain** *γ g2* **where** *H2*:
γ ∈ *qbs-Mx Z g2* ∈ *real-borel* $\to_M$ *prob-algebra real-borel*
(*g* ∘ *β*) = (*λr. qbs-prob-space* (*Z*, *γ*, *g2 r*))
by *auto*
note [*measurable*] = *H1*(*2*) *H2*(*2*)
interpret *rd*: *real-distribution μ*
using *H0*(*1*) **by**(*simp add*: *qbs-prob-def*)
have *LHS*: (*s* ⪢= *f*) ⪢= *g* = *qbs-prob-space* (*Z*, *γ*, *μ* ⪢= *g1* ⪢= *g2*)
by(*rule qbs-prob.qbs-bind-computation*(*2*)[*OF qbs-prob.qbs-bind-computation*[*OF H0 assms*(*2*) *H1*] *assms*(*3*) *H2*])
have *RHS*: *s* ⪢= (*λx. f x* ⪢= *g*) = *qbs-prob-space* (*Z*, *γ*, *μ* ⪢= (*λx. g1 x* ⪢= *g2*))
apply(*auto intro*!: *qbs-prob.qbs-bind-computation*[*OF H0 qbs-morphism-comp*[*OF*

assms(*2*) *qbs-bind-morphism'*[*OF assms*(*3*)],*simplified comp-def*]]
simp: *real-distribution-def real-distribution-axioms-def qbs-prob-def qbs-prob-MPx*[*OF H2*(*1*,*2*),*simplified qbs-prob-def*] *sets-bind'*[*OF measurable-space*[*OF H1*(*2*)] *H2*(*2*)] *prob-space-bind'*[*OF measurable-space*[*OF H1*(*2*)] *H2*(*2*)] *measurable-space*[*OF H2*(*2*)] *space-prob-algebra*[*of real-borel*] *H2*(*1*))
proof
fix *r*
show $((\lambda x.\ f\ x \gg= g) \circ \alpha)\ r = qbs\text{-}prob\text{-}space\ (Z,\ \gamma,\ g1\ r \gg= g2)$ (**is** *?lhs* = *?rhs*) **for** *r*
by(*auto intro*!: *qbs-prob.qbs-bind-computation*(*2*)[*of Y* β]
simp: *qbs-prob-MPx*[*OF H1*(*1*,*2*),*of r*] *assms*(*3*) *H2 fun-cong*[*OF H1*(*3*),*simplified comp-def*])
qed
have *ba*: $\mu \gg= g1 \gg= g2 = \mu \gg= (\lambda x.\ g1\ x \gg= g2)$
by(*auto intro*!: *bind-assoc*[**where** *N=real-borel* **and** *R=real-borel*] *simp*: *measurable-prob-algebraD*)
show *?thesis*
by(*simp add*: *LHS RHS ba*)
qed

lemma *qbs-bind-cong*:
assumes $s \in monadP\text{-}qbs\text{-}Px\ X$
$\bigwedge x.\ x \in qbs\text{-}space\ X \Longrightarrow f\ x = g\ x$
and $f \in X \rightarrow_Q monadP\text{-}qbs\ Y$
shows $s \gg= f = s \gg= g$
proof −
obtain α μ **where** *h0*:
$qbs\text{-}prob\ X\ \alpha\ \mu \quad s = qbs\text{-}prob\text{-}space\ (X,\ \alpha,\ \mu)$
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
then have $f \circ \alpha \in monadP\text{-}qbs\text{-}MPx\ Y$
using *assms*(*3*) *h0*(*1*) **by**(*auto simp*: *qbs-prob-def in-Mx-def*)
from *rep-monadP-qbs-MPx*[*OF this*] **obtain** γ *k* **where** *h1*:
$\gamma \in qbs\text{-}Mx\ Y\ k \in real\text{-}borel \rightarrow_M prob\text{-}algebra\ real\text{-}borel$
$(f \circ \alpha) = (\lambda r.\ qbs\text{-}prob\text{-}space\ (Y,\ \gamma,\ k\ r))$
by *auto*
have *hg*:$g \in X \rightarrow_Q monadP\text{-}qbs\ Y$
using *qbs-morphism-cong*[*OF assms*(*2*,*3*)] **by** *simp*
have *hgs*: $f \circ \alpha = g \circ \alpha$
using *h0*(*1*) *assms*(*2*) **by**(*force simp*: *qbs-prob-def in-Mx-def*)

show *?thesis*
by(*simp add*: *qbs-prob.qbs-bind-computation*(*2*)[*OF h0 assms*(*3*) *h1*]
qbs-prob.qbs-bind-computation(*2*)[*OF h0 hg h1*[*simplified hgs*]])
qed

3.2.4 The Functorial Action $P(f)$

definition *monadP-qbs-Pf* :: [*'a quasi-borel*, *'b quasi-borel*,*'a* ⇒ *'b*,*'a qbs-prob-space*] ⇒ *'b qbs-prob-space* **where**

monadP-qbs-Pf - Y f sx ≡ *sx* ≫= *qbs-return Y* ∘ *f*

lemma *monadP-qbs-Pf-morphism*:
  assumes $f \in X \rightarrow_Q Y$
  shows *monadP-qbs-Pf X Y f* ∈ *monadP-qbs X* $\rightarrow_Q$ *monadP-qbs Y*
  unfolding *monadP-qbs-Pf-def*
 by(*rule qbs-bind-morphism*′[*OF qbs-morphism-comp*[*OF assms qbs-return-morphism*]])

lemma(**in** *qbs-prob*) *monadP-qbs-Pf-computation*:
  assumes *s* = *qbs-prob-space* (*X*,α,μ)
      and $f \in X \rightarrow_Q Y$
    shows *qbs-prob Y* (*f* ∘ α) μ
      and *monadP-qbs-Pf X Y f s* = *qbs-prob-space* (*Y*,*f* ∘ α,μ)
  by(*auto intro*!: *qbs-bind-computation*[*OF assms*(*1*) *qbs-morphism-comp*[*OF assms*(*2*) *qbs-return-morphism*],*of f* ∘ α *return real-borel* ,*simplified bind-return*′′[*OF M-is-borel*]]
               simp: *monadP-qbs-Pf-def qbs-return-comp*[*OF qbs-morphismE*(*3*)[*OF assms*(*2*) *in-Mx*],*simplified comp-assoc*[*symmetric*]] *qbs-morphismE*(*3*)[*OF assms*(*2*) *in-Mx*] *prob-space-return*)

We show that P is a functor i.e. P preserves identity and composition.

lemma *monadP-qbs-Pf-id*:
  assumes *s* ∈ *monadP-qbs-Px X*
  shows *monadP-qbs-Pf X X id s* = *s*
  using *qbs-bind-return*′[*OF assms*] **by**(*simp add*: *monadP-qbs-Pf-def*)

lemma *monadP-qbs-Pf-comp*:
  assumes *s* ∈ *monadP-qbs-Px X*
          $f \in X \rightarrow_Q Y$
      and $g \in Y \rightarrow_Q Z$
    shows ((*monadP-qbs-Pf Y Z g*) ∘ (*monadP-qbs-Pf X Y f*)) *s* = *monadP-qbs-Pf X Z* (*g* ∘ *f*) *s*
proof −
  obtain α μ **where** *h*:
  qbs-prob X α μ *s* = *qbs-prob-space* (*X*, α, μ)
    using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
  hence *qbs-prob Y* (*f* ∘ α) μ
        monadP-qbs-Pf X Y f s = *qbs-prob-space* (*Y*,*f* ∘ α,μ)
    using *qbs-prob.monadP-qbs-Pf-computation*[*OF - - assms*(*2*)] **by** *auto*
 from *qbs-prob.monadP-qbs-Pf-computation*[*OF this assms*(*3*)] *qbs-prob.monadP-qbs-Pf-computation*[*OF h qbs-morphism-comp*[*OF assms*(*2*,*3*)]]
  show *?thesis*
    by(*simp add*: *comp-assoc*)
qed

3.2.5 Join

definition *qbs-join* :: ′*a qbs-prob-space qbs-prob-space* ⇒ ′*a qbs-prob-space* **where**
qbs-join ≡ (λ*sst*. *sst* ≫= *id*)

lemma *qbs-join-morphism*:
qbs-join ∈ *monadP-qbs* (*monadP-qbs X*) $\rightarrow_Q$ *monadP-qbs X*
by(*simp add*: *qbs-join-def qbs-bind-morphism'*[*OF qbs-morphism-ident*])

lemma *qbs-join-computation*:
assumes *qbs-prob* (*monadP-qbs X*) β μ
ssx = *qbs-prob-space* (*monadP-qbs X*,β,μ)
α ∈ *qbs-Mx X*
g ∈ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
and β =(λ*r*. *qbs-prob-space* (*X*,α,*g r*))
shows *qbs-prob X* α (μ ≫= *g*) *qbs-join ssx* = *qbs-prob-space* (*X*,α, μ ≫= *g*)
using *qbs-prob.qbs-bind-computation*[*OF assms*(*1*,*2*) *qbs-morphism-ident assms*(*3*,*4*)]
by(*auto simp*: *assms*(*5*) *qbs-join-def*)

3.2.6 Strength

definition *qbs-strength* :: [*'a quasi-borel*,*'b quasi-borel*,*'a* × *'b qbs-prob-space*] ⇒ (*'a* × *'b*) *qbs-prob-space* **where**
qbs-strength W X = (λ(*w*,*sx*). *let* (-,α,μ) = *rep-qbs-prob-space sx*
in qbs-prob-space (*W* $\bigotimes_Q$ *X*, λ*r*. (*w*,α *r*), μ))

lemma(**in** *qbs-prob*) *qbs-strength-computation*:
assumes *w* ∈ *qbs-space W*
and *sx* = *qbs-prob-space* (*X*,α,μ)
shows *qbs-prob* (*W* $\bigotimes_Q$ *X*) (λ*r*. (*w*,α *r*)) μ
qbs-strength W X (*w*,*sx*) = *qbs-prob-space* (*W* $\bigotimes_Q$ *X*, λ*r*. (*w*,α *r*), μ)
proof −
interpret *qp1*: *qbs-prob W* $\bigotimes_Q$ *X* λ*r*. (*w*,α *r*) μ
by(*auto intro*!: *qbs-probI simp*: *assms*(*1*) *pair-qbs-Mx-def comp-def*)
show *qbs-prob* (*W* $\bigotimes_Q$ *X*) (λ*r*. (*w*,α *r*)) μ
qbs-strength W X (*w*,*sx*) = *qbs-prob-space* (*W* $\bigotimes_Q$ *X*, λ*r*. (*w*,α *r*), μ)
apply(*simp-all add*: *qp1*.*qbs-prob-axioms qbs-strength-def rep-qbs-prob-space-def qbs-prob-space.rep-def*)
apply(*rule someI2*[**where** *a*=(*X*,α,μ)])
proof(*auto simp*: *in-Rep assms*(*2*))
fix *X'* α' μ'
assume *h*:(*X'*,α',μ') ∈ *Rep-qbs-prob-space* (*qbs-prob-space* (*X*, α, μ))
from *if-in-Rep*(*1*,*2*)[*OF this*] **interpret** *pqp*: *pair-qbs-prob W* $\bigotimes_Q$ *X* λ*r*. (*w*, α' *r*) μ' *W* $\bigotimes_Q$ *X* λ*r*. (*w*,α *r*) μ
by(*simp add*: *pair-qbs-prob-def qp1*.*qbs-prob-axioms*)
(*auto intro*!: *qbs-probI simp*: *pair-qbs-Mx-def comp-def assms*(*1*) *qbs-prob-def in-Mx-def*)
note [*simp*] = *qbs-prob-eq2-dest*[*OF if-in-Rep*(*3*)[*OF h*,*simplified qbs-prob-eq-equiv12*]]
show *qbs-prob-space* (*W* $\bigotimes_Q$ *X*, λ*r*. (*w*, α' *r*), μ') = *qbs-prob-space* (*W* $\bigotimes_Q$ *X*, λ*r*. (*w*, α *r*), μ)
proof(*rule pqp.qbs-prob-space-eq2*)
fix *f*
assume *f* ∈ *qbs-to-measure* (*W* $\bigotimes_Q$ *X*) $\rightarrow_M$ *real-borel*
note *qbs-morphism-dest*[*OF qbs-morphismE*(*2*)[*OF curry-preserves-morphisms*[*OF*

qbs-morphism-measurable-intro[*OF this*]] *assms*(*1*),*simplified*]]
show ($\int$ *y*. *f* ((λr. (*w*, α' *r*)) *y*) ∂ μ') = ($\int$ *y*. *f* ((λr. (*w*, α *r*)) *y*) ∂ μ)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\int$ *y*. *curry f w* (α' *y*) ∂ μ') **by** *auto*
also have ... = ($\int$ *y*. *curry f w* (α *y*) ∂ μ)
by(*rule qbs-prob-eq2-dest*(*4*)[*OF if-in-Rep*(*3*)[*OF h*,*simplified qbs-prob-eq-equiv12*],*symmetric*]) *fact*
also have ... = *?rhs* **by** *auto*
finally show *?thesis* **.**
qed
qed *simp*
qed
qed

lemma *qbs-strength-natural*:
assumes *f* ∈ *X* $\to_Q$ *X'*
g ∈ *Y* $\to_Q$ *Y'*
x ∈ *qbs-space X*
and *sy* ∈ *monadP-qbs-Px Y*
shows (*monadP-qbs-Pf* (*X* $\bigotimes_Q$ *Y*) (*X'* $\bigotimes_Q$ *Y'*) (*map-prod f g*) ∘ *qbs-strength X Y*) (*x*,*sy*) = (*qbs-strength X' Y'* ∘ *map-prod f* (*monadP-qbs-Pf Y Y' g*)) (*x*,*sy*)
(**is** *?lhs* = *?rhs*)
proof −
obtain β ν **where** *hy*:
qbs-prob Y β ν *sy* = *qbs-prob-space* (*Y*,β,ν)
using *rep-monadP-qbs-Px*[*OF assms*(*4*)] **by** *auto*
have *qbs-prob* (*X* $\bigotimes_Q$ *Y*) (λr. (*x*, β *r*)) ν
qbs-strength X Y (*x*, *sy*) = *qbs-prob-space* (*X* $\bigotimes_Q$ *Y*, λr. (*x*, β *r*), ν)
using *qbs-prob.qbs-strength-computation*[*OF hy*(*1*) *assms*(*3*) *hy*(*2*)] **by** *auto*
hence *LHS*:*?lhs* = *qbs-prob-space* (*X'* $\bigotimes_Q$ *Y'*,*map-prod f g* ∘ (λr. (*x*, β *r*)),ν)
by(*simp add*: *qbs-prob.monadP-qbs-Pf-computation*[*OF - - qbs-morphism-map-prod*[*OF assms*(*1*,*2*)]]])

have *map-prod f* (*monadP-qbs-Pf Y Y' g*) (*x*,*sy*) = (*f x*,*qbs-prob-space* (*Y'*,*g* ∘ β,ν))
qbs-prob Y' (*g* ∘ β) ν
by(*auto simp*: *qbs-prob.monadP-qbs-Pf-computation*[*OF hy assms*(*2*)]])
hence *RHS*:*?rhs* = *qbs-prob-space* (*X'* $\bigotimes_Q$ *Y'*,λr. (*f x*, (*g* ∘ β) *r*),ν)
using *qbs-prob.qbs-strength-computation*[*OF - - refl*,*of Y' g* ∘ β ν *f x X'*] *assms*(*1*,*3*)
by *auto*

show *?lhs* = *?rhs*
unfolding *LHS RHS*
by(*simp add*: *comp-def*)
qed

lemma *qbs-strength-ab-r*:

assumes $\alpha \in$ *qbs-Mx X*
$\beta \in$ *monadP-qbs-MPx Y*
$\gamma \in$ *qbs-Mx Y*
and [*measurable*]:$g \in$ *real-borel* $\to_M$ *prob-algebra real-borel*
and $\beta = (\lambda r.$ *qbs-prob-space* $(Y,\gamma,g\ r))$
shows *qbs-prob* $(X \bigotimes_Q Y)$ (*map-prod* $\alpha\ \gamma \circ$ *real-real.g*) (*distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*)
qbs-strength X Y $(\alpha\ r,\ \beta\ r) =$ *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* $\alpha\ \gamma \circ$ *real-real.g*, *distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*)
proof $-$
have [*measurable-cong*]: *sets* $(g\ r) =$ *sets real-borel*
sets (*return real-borel r*) = *sets real-borel*
using *measurable-space*[*OF assms*(*4*),*of r*]
by(*simp-all add*: *space-prob-algebra*)
interpret *qp*: *qbs-prob* $X \bigotimes_Q Y$ *map-prod* $\alpha\ \gamma \circ$ *real-real.g distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*
proof(*auto intro*!: *qbs-probI*)
show *map-prod* $\alpha\ \gamma \circ$ *real-real.g* $\in$ *pair-qbs-Mx X Y*
using *qbs-closed1-dest*[*OF assms*(*1*)] *qbs-closed1-dest*[*OF assms*(*3*)]
by(*auto simp*: *comp-def qbs-prob-def in-Mx-def pair-qbs-Mx-def*)
next
show *prob-space* (*distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*)
using *measurable-space*[*OF assms*(*4*),*of r*]
by(*auto intro*!: *prob-space.prob-space-distr simp*: *prob-algebra-real-prob-measure prob-space-pair prob-space-return real-distribution.axioms*(*1*))
qed
interpret *pqp*: *pair-qbs-prob* $X \bigotimes_Q Y\ \lambda l.\ (\alpha\ r,\ \gamma\ l)\ g\ r\ X \bigotimes_Q Y$ *map-prod* $\alpha\ \gamma \circ$ *real-real.g distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*
by(*simp add*: *qbs-prob.qbs-strength-computation*[*OF qbs-prob-MPx*[*OF assms*(*3*,*4*)] *qbs-Mx-to-X*(*2*)[*OF assms*(*1*)] *fun-cong*[*OF assms*(*5*)],*of r*] *pair-qbs-prob-def qp.qbs-prob-axioms*)
have [*measurable*]: *map-prod* $\alpha\ \gamma \in$ *real-borel* $\bigotimes_M$ *real-borel* $\to_M$ *qbs-to-measure* $(X \bigotimes_Q Y)$
proof $-$
have *map-prod* $\alpha\ \gamma \in \mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q \to_Q X \bigotimes_Q Y$
using *assms*(*1*,*3*) **by**(*auto intro*!: *qbs-morphism-map-prod simp*: *qbs-Mx-is-morphisms*)
hence *map-prod* $\alpha\ \gamma \in$ *qbs-to-measure* $(\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q) \to_M$ *qbs-to-measure* $(X \bigotimes_Q Y)$
using *l-preserves-morphisms* **by** *auto*
thus *?thesis*
by *simp*
qed
hence [*measurable*]:$(\lambda l.\ (\alpha\ r,\ \gamma\ l)) \in$ *real-borel* $\to_M$ *qbs-to-measure* $(X \bigotimes_Q Y)$
using *pqp.qp1.in-Mx qbs-Mx-are-measurable* **by** *blast*

show *qbs-prob* $(X \bigotimes_Q Y)$ (*map-prod* $\alpha\ \gamma \circ$ *real-real.g*) (*distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*)
qbs-strength X Y $(\alpha\ r,\ \beta\ r) =$ *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* $\alpha\ \gamma \circ$ *real-real.g*, *distr* (*return real-borel* $r \bigotimes_M g\ r$) *real-borel real-real.f*)
apply(*simp-all add*: *qp.qbs-prob-axioms qbs-prob.qbs-strength-computation*(*2*)[*OF*

qbs-prob-MPx[*OF assms*(*3*,*4*)] *qbs-Mx-to-X*(*2*)[*OF assms*(*1*)] *fun-cong*[*OF assms*(*5*)],*of r*])
proof(*rule pqp.qbs-prob-space-eq*)
show *distr* (*g r*) (*qbs-to-measure* ($X \bigotimes_Q Y$)) ($\lambda l.$ (α *r*, γ *l*)) = *distr* (*distr* (*return real-borel r* $\bigotimes_M$ *g r*) *real-borel real-real.f*) (*qbs-to-measure* ($X \bigotimes_Q Y$)) (*map-prod* α γ $\circ$ *real-real.g*)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *distr* (*g r*) (*qbs-to-measure* ($X \bigotimes_Q Y$)) (*map-prod* α γ $\circ$ *Pair r*)
by(*simp add*: *comp-def*)
also have ... = *distr* (*distr* (*g r*) (*real-borel* $\bigotimes_M$ *real-borel*) (*Pair r*)) (*qbs-to-measure* ($X \bigotimes_Q Y$)) (*map-prod* α γ)
by(*auto intro*!: *distr-distr*[*symmetric*])
also have ... = *distr* (*return real-borel r* $\bigotimes_M$ *g r*) (*qbs-to-measure* ($X \bigotimes_Q Y$)) (*map-prod* α γ)
proof −
have *return real-borel r* $\bigotimes_M$ *g r* = *distr* (*g r*) (*real-borel* $\bigotimes_M$ *real-borel*) ($\lambda l.$ (*r*,*l*))
proof(*auto intro*!: *measure-eqI*)
fix *A*
assume *h*′:*A* ∈ *sets* (*real-borel* $\bigotimes_M$ *real-borel*)
show *emeasure* (*return real-borel r* $\bigotimes_M$ *g r*) *A* = *emeasure* (*distr* (*g r*) (*real-borel* $\bigotimes_M$ *real-borel*) (*Pair r*)) *A*
(**is** *?lhs*′ = *?rhs*′)
proof −
have *?lhs*′ = $\int^+$ *x*. *emeasure* (*g r*) (*Pair x* −‘ *A*) ∂*return real-borel r*
by(*auto intro*!: *pqp.qp1.emeasure-pair-measure-alt simp*: *h*′)
also have ... = *emeasure* (*g r*) (*Pair r* −‘ *A*)
by(*auto intro*!: *nn-integral-return pqp.qp1.measurable-emeasure-Pair simp*: *h*′)
also have ... = *?rhs*′
by(*simp add*: *emeasure-distr*[*OF* - *h*′])
finally show *?thesis* **.**
qed
qed
thus *?thesis* **by** *simp*
qed
also have ... = *?rhs*
by(*rule distr-distr*[*of map-prod* α γ $\circ$ *real-real.g real-borel qbs-to-measure* ($X \bigotimes_Q Y$) *real-real.f return real-borel r* $\bigotimes_M$ *g r*,*simplified comp-assoc*,*simplified*,*symmetric*])
finally show *?thesis* **.**
qed
qed *simp*
qed

lemma *qbs-strength-morphism*:
qbs-strength X Y ∈ $X \bigotimes_Q$ *monadP-qbs Y* $\rightarrow_Q$ *monadP-qbs* ($X \bigotimes_Q Y$)

proof(*rule pair-qbs-morphismI*,*simp*)
  fix α β
  assume *h*:$\alpha \in$ *qbs-Mx X*
          $\beta \in$ *monadP-qbs-MPx Y*
  then obtain γ *g* **where** *hb*:
    $\gamma \in$ *qbs-Mx Y g* $\in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
    $\beta = (\lambda r.$ *qbs-prob-space* $(Y, \gamma, g\ r))$
    using *rep-monadP-qbs-MPx*[*of* β] **by** *blast*
  note [*measurable*] = *hb*(*2*)
  show *qbs-strength X Y* $\circ$ $(\lambda r.\ (\alpha\ r, \beta\ r)) \in$ *monadP-qbs-MPx* $(X \bigotimes_Q Y)$
    using *qbs-strength-ab-r*[*OF h hb*]
    by(*auto intro*!: *bexI*[**where** *x*=*map-prod* α γ $\circ$ *real-real.g*] *bexI*[**where** *x*=λr. *distr* (*return real-borel r* $\bigotimes_M$ *g r*) *real-borel real-real.f*]
          simp: *monadP-qbs-MPx-def in-MPx-def qbs-prob-def in-Mx-def*)
qed

lemma *qbs-bind-morphism*$''$:
$(\lambda(f,x).\ x \gg= f) \in$ *exp-qbs X* (*monadP-qbs Y*) $\bigotimes_Q$ (*monadP-qbs X*) $\rightarrow_Q$ (*monadP-qbs Y*)
proof(*rule qbs-morphism-cong*[*of - qbs-join* $\circ$ (*monadP-qbs-Pf* (*exp-qbs X* (*monadP-qbs Y*) $\bigotimes_Q$ *X*) (*monadP-qbs Y*) *qbs-eval*) $\circ$ (*qbs-strength* (*exp-qbs X* (*monadP-qbs Y*)) *X*)], *auto*)
  fix *f*
  fix *sx*
  assume *h*:*f* $\in$ *X* $\rightarrow_Q$ *monadP-qbs Y*
          sx $\in$ *monadP-qbs-Px X*
  then obtain α μ **where** *h0*:*qbs-prob X* α μ *sx* = *qbs-prob-space* (X, α, μ)
    using *rep-monadP-qbs-Px*[*of sx X*] **by** *auto*
  hence *f* $\circ$ $\alpha \in$ *monadP-qbs-MPx Y*
    using *h*(*1*) **by**(*auto simp*: *qbs-prob-def in-Mx-def*)
  then obtain β *g* **where** *h1*:
  $\beta \in$ *qbs-Mx Y g* $\in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
  $(f \circ \alpha) = (\lambda r.$ *qbs-prob-space* $(Y, \beta, g\ r))$
    using *rep-monadP-qbs-MPx*[*of f* $\circ$ α *Y*] **by** *blast*

  show *qbs-join* (*monadP-qbs-Pf* (*exp-qbs X* (*monadP-qbs Y*) $\bigotimes_Q$ *X*) (*monadP-qbs Y*) *qbs-eval* (*qbs-strength* (*exp-qbs X* (*monadP-qbs Y*)) *X* (*f*, *sx*))) =
          sx $\gg=$ *f*
   by(*simp add*: *qbs-join-computation*[*OF qbs-prob.monadP-qbs-Pf-computation*[*OF qbs-prob.qbs-strength-computation*[*OF h0*(*1*) *- h0*(*2*),*of f exp-qbs X* (*monadP-qbs Y*)] *qbs-eval-morphism*] *h1*(*1*,*2*),*simplified qbs-eval-def comp-def*,*simplified*,*OF h*(*1*) *h1*(*3*)[*simplified comp-def*]] *qbs-prob.qbs-bind-computation*[*OF h0 h*(*1*) *h1*])
next
  show *qbs-join* $\circ$ *monadP-qbs-Pf* (*exp-qbs X* (*monadP-qbs Y*) $\bigotimes_Q$ *X*) (*monadP-qbs Y*) *qbs-eval* $\circ$ *qbs-strength* (*exp-qbs X* (*monadP-qbs Y*)) *X* $\in$ *exp-qbs X* (*monadP-qbs Y*) $\bigotimes_Q$ *monadP-qbs X* $\rightarrow_Q$ *monadP-qbs Y*
    using *qbs-join-morphism monadP-qbs-Pf-morphism*[*OF qbs-eval-morphism*]
    by(*auto intro*!: *qbs-morphism-comp simp*: *qbs-strength-morphism*)
qed

lemma *qbs-bind-morphism'''*:
  $(\lambda f\ x.\ x \gg= f) \in$ *exp-qbs* X (*monadP-qbs* Y) $\to_Q$ *exp-qbs* (*monadP-qbs* X) (*monadP-qbs* Y)
  using *qbs-bind-morphism''* *curry-preserves-morphisms*[*of* $\lambda(f, x)$. *qbs-bind* x f]
  by *fastforce*

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

lemma *qbs-bind-morphism''''*:
  assumes $x \in$ *monadP-qbs-Px* X
  shows $(\lambda f.\ x \gg= f) \in$ *exp-qbs* X (*monadP-qbs* Y) $\to_Q$ *monadP-qbs* Y
  by(*rule qbs-morphismE*(*2*)[*OF arg-swap-morphism*[*OF qbs-bind-morphism''*],*simplified*,*OF assms*])

lemma *qbs-strength-law1*:
  assumes $x \in$ *qbs-space* (*unit-quasi-borel* $\bigotimes_Q$ *monadP-qbs* X)
  shows *snd* x = (*monadP-qbs-Pf* (*unit-quasi-borel* $\bigotimes_Q$ X) X *snd* $\circ$ *qbs-strength unit-quasi-borel* X) x
proof −
  obtain α μ **where** h:
   qbs-prob X α μ (*snd* x) = *qbs-prob-space* (X, α, μ)
    using *rep-monadP-qbs-Px*[*of snd* x X] *assms* **by** *auto*
  have [*simp*]: ((),*snd* x) = x
    using *SigmaE assms* **by** *auto*
  show *?thesis*
   using *qbs-prob.monadP-qbs-Pf-computation*[*OF qbs-prob.qbs-strength-computation*[*OF h*(*1*) - *h*(*2*),*of fst* x *unit-quasi-borel*,*simplified*] *snd-qbs-morphism*]
    by(*simp add*: *h*(*2*) *comp-def*)
qed

lemma *qbs-strength-law2*:
  assumes $x \in$ *qbs-space* ((X $\bigotimes_Q$ Y) $\bigotimes_Q$ *monadP-qbs* Z)
  shows (*qbs-strength* X (Y $\bigotimes_Q$ Z) $\circ$ (*map-prod id* (*qbs-strength* Y Z)) $\circ$ (λ((x,y),z). (x,(y,z)))) x =
     (*monadP-qbs-Pf* ((X $\bigotimes_Q$ Y) $\bigotimes_Q$ Z) (X $\bigotimes_Q$ (Y $\bigotimes_Q$ Z)) (λ((x,y),z). (x,(y,z))) $\circ$ *qbs-strength* (X $\bigotimes_Q$ Y) Z) x
     (**is** *?lhs* = *?rhs*)
proof −
  obtain α μ **where** h:
   qbs-prob Z α μ *snd* x = *qbs-prob-space* (Z, α, μ)
    using *rep-monadP-qbs-Px*[*of snd* x Z] *assms* **by** *auto*
  have *?lhs* = *qbs-prob-space* (X $\bigotimes_Q$ Y $\bigotimes_Q$ Z, λr. (*fst* (*fst* x), *snd* (*fst* x), α r), μ)

using *assms qbs-prob.qbs-strength-computation*[*OF h*(*1*) - *h*(*2*),*of snd* (*fst x*) *Y*]
by(*auto intro*!: *qbs-prob.qbs-strength-computation*)
also have ... = *?rhs*
using *qbs-prob.monadP-qbs-Pf-computation*[*OF qbs-prob.qbs-strength-computation*[*OF h*(*1*) - *h*(*2*),*of fst x X* $\bigotimes_Q$ *Y*] *qbs-morphism-pair-assoc1*] *assms*
by(*auto simp*: *comp-def*)
finally show *?thesis* **.**
qed

lemma *qbs-strength-law3*:
assumes $x \in$ *qbs-space* (*X* $\bigotimes_Q$ *Y*)
shows *qbs-return* (*X* $\bigotimes_Q$ *Y*) *x* = (*qbs-strength X Y* $\circ$ (*map-prod id* (*qbs-return Y*))) *x*
proof −
interpret *qp*: *qbs-prob Y* $\lambda r.$ *snd x return real-borel 0*
using *assms* **by**(*auto intro*!: *qbs-probI simp*: *prob-space-return*)
show *?thesis*
using *qp.qbs-strength-computation*[*OF - qp.qbs-return-computation*[*of snd x Y*],*of fst x X*] *assms*
by(*auto simp*: *qp.qbs-return-computation*[*OF assms*])
qed

lemma *qbs-strength-law4*:
assumes $x \in$ *qbs-space* (*X* $\bigotimes_Q$ *monadP-qbs* (*monadP-qbs Y*))
shows (*qbs-strength X Y* $\circ$ *map-prod id qbs-join*) *x* = (*qbs-join* $\circ$ *monadP-qbs-Pf* (*X* $\bigotimes_Q$ *monadP-qbs Y*) (*monadP-qbs* (*X* $\bigotimes_Q$ *Y*))(*qbs-strength X Y*) $\circ$ *qbs-strength X* (*monadP-qbs Y*)) *x*
(**is** *?lhs* = *?rhs*)
proof −
obtain β μ **where** *h0*:
qbs-prob (*monadP-qbs Y*) β μ *snd x* = *qbs-prob-space* (*monadP-qbs Y*, β, μ)
using *rep-monadP-qbs-Px*[*of snd x monadP-qbs Y*] *assms* **by** *auto*
then obtain γ *g* **where** *h1*:
$\gamma \in$ *qbs-Mx Y g* $\in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
β = ($\lambda r.$ *qbs-prob-space* (*Y*, γ, *g r*))
using *rep-monadP-qbs-MPx*[*of* β *Y*] **by**(*auto simp*: *qbs-prob-def in-Mx-def*)
have *?lhs* = *qbs-prob-space* (*X* $\bigotimes_Q$ *Y*, $\lambda r.$ (*fst x*, γ *r*), $\mu \gg\!\!= g$)
using *qbs-prob.qbs-strength-computation*[*OF qbs-join-computation*(*1*)[*OF h0 h1*] - *qbs-join-computation*(*2*)[*OF h0 h1*],*of fst x X*] *assms*
by *auto*
also have ... = *?rhs*
proof −
have *qbs-strength X Y* $\circ$ ($\lambda r.$ (*fst x*, β *r*)) = ($\lambda r.$ *qbs-prob-space* (*X* $\bigotimes_Q$ *Y*, $\lambda r.$ (*fst x*, γ *r*), *g r*))
proof
show (*qbs-strength X Y* $\circ$ ($\lambda r.$ (*fst x*, β *r*))) *r* = *qbs-prob-space* (*X* $\bigotimes_Q$ *Y*, $\lambda r.$ (*fst x*, γ *r*), *g r*) **for** *r*
using *qbs-prob.qbs-strength-computation*(*2*)[*OF qbs-prob-MPx*[*OF h1*(*1*,*2*),*of*

r] - *fun-cong*[*OF h1*(*3*)],*of fst x X*] *assms*
by *auto*
qed
thus *?thesis*
using *qbs-join-computation*(*2*)[*OF qbs-prob.monadP-qbs-Pf-computation*[*OF qbs-prob.qbs-strength-computation*[*OF h0*(*1*) - *h0*(*2*),*of fst x X*] *qbs-strength-morphism*] - *h1*(*2*),*of* $\lambda r.$ (*fst x*, γ *r*),*symmetric*] *assms h1*(*1*)
by(*auto simp*: *pair-qbs-Mx-def comp-def*)
qed
finally show *?thesis* **.**
qed

lemma *qbs-return-Mxpair*:
assumes $\alpha \in$ *qbs-Mx X*
and $\beta \in$ *qbs-Mx Y*
shows *qbs-return* ($X \bigotimes_Q Y$) (α *r*, β *k*) = *qbs-prob-space* ($X \bigotimes_Q Y$,*map-prod* α β $\circ$ *real-real.g*, *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel k*) *real-borel real-real.f*)
qbs-prob ($X \bigotimes_Q Y$) (*map-prod* α β $\circ$ *real-real.g*) (*distr* (*return real-borel r* $\bigotimes_M$ *return real-borel k*) *real-borel real-real.f*)
proof −
note [*measurable-cong*] = *sets-return*[*of real-borel*]
interpret *qp*: *qbs-prob X* $\bigotimes_Q$ *Y map-prod* α β $\circ$ *real-real.g distr* (*return real-borel r* $\bigotimes_M$ *return real-borel k*) *real-borel real-real.f*
using *qbs-closed1-dest*[*OF assms*(*1*)] *qbs-closed1-dest*[*OF assms*(*2*)]
by(*auto intro*!: *qbs-probI prob-space.prob-space-distr prob-space-pair*
simp: *pair-qbs-Mx-def comp-def prob-space-return*)
show *qbs-return* ($X \bigotimes_Q Y$) (α *r*, β *k*) = *qbs-prob-space* ($X \bigotimes_Q Y$,*map-prod* α β $\circ$ *real-real.g*, *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel k*) *real-borel real-real.f*)
qbs-prob ($X \bigotimes_Q Y$) (*map-prod* α β $\circ$ *real-real.g*) (*distr* (*return real-borel r* $\bigotimes_M$ *return real-borel k*) *real-borel real-real.f*)
proof −
show *qbs-return* ($X \bigotimes_Q Y$) (α *r*, β *k*) = *qbs-prob-space* ($X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g*, *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel k*) *real-borel real-real.f*)
(**is** *?lhs* = *?rhs*)
proof −
have *1*:($\lambda r.$ *qbs-prob-space* (*Y*, β, *return real-borel k*)) $\in$ *monadP-qbs-MPx Y*
by(*auto intro*!: *in-MPx.intro bexI*[**where** $x{=}\beta$] *bexI*[**where** $x{=}\lambda r.$ *return real-borel k*] *simp*: *monadP-qbs-MPx-def assms*(*2*))
have *?lhs* = (*qbs-strength X Y* $\circ$ *map-prod id* (*qbs-return Y*)) (α *r*, β *k*)
by(*intro qbs-strength-law3*[*of* (α *r*, β *k*) *X Y*]) (*use assms* **in** *auto*)
also have ... = *qbs-strength X Y* (α *r*, *qbs-prob-space* (*Y*, β, *return real-borel k*))
using *fun-cong*[*OF qbs-return-comp*[*OF assms*(*2*)]] **by** *simp*
also have ... = *?rhs*
by(*intro qbs-strength-ab-r*(*2*)[*OF assms*(*1*) *1 assms*(*2*) - *refl*,*of r*]) *auto*
finally show *?thesis* **.**

qed
qed(*rule qp.qbs-prob-axioms*)
qed

lemma *pair-return-return*:
assumes $l \in space\ M$
and $r \in space\ N$
shows $return\ M\ l \bigotimes_M return\ N\ r = return\ (M \bigotimes_M N)\ (l,r)$
proof(*auto intro*!: *measure-eqI*)
fix A
assume $h{:}A \in sets\ (M \bigotimes_M N)$
show $emeasure\ (return\ M\ l \bigotimes_M return\ N\ r)\ A = indicator\ A\ (l,\ r)$
(**is** *?lhs = ?rhs*)
proof −
have $?lhs = (\int^+ x.\ \int^+ y.\ indicator\ A\ (x,\ y)\ \partial return\ N\ r\ \partial return\ M\ l)$
by(*auto intro*!: *sigma-finite-measure.emeasure-pair-measure prob-space-imp-sigma-finite simp*: *h prob-space-return assms*)
also have ... $= (\int^+ x.\ indicator\ A\ (x,\ r)\ \partial return\ M\ l)$
using *h* **by**(*auto intro*!: *nn-integral-cong nn-integral-return simp*: *assms*(*2*))
also have ... = *?rhs*
using *h* **by**(*auto intro*!: *nn-integral-return simp*: *assms*)
finally show *?thesis* **.**
qed
qed

lemma *bind-bind-return-distr*:
assumes *real-distribution* μ
and *real-distribution* ν
shows $\mu \ggg= (\lambda r.\ \nu \ggg= (\lambda l.\ distr\ (return\ real\text{-}borel\ r \bigotimes_M return\ real\text{-}borel\ l)\ real\text{-}borel\ real\text{-}real.f))$
$= distr\ (\mu \bigotimes_M \nu)\ real\text{-}borel\ real\text{-}real.f$
(**is** *?lhs = ?rhs*)
proof −
interpret *rd1*: *real-distribution* μ **by** *fact*
interpret *rd2*: *real-distribution* ν **by** *fact*
interpret *pp*: *pair-prob-space* $\mu\ \nu$
by (*simp add*: *pair-prob-space.intro pair-sigma-finite-def rd1.prob-space-axioms rd1.sigma-finite-measure-axioms rd2.prob-space-axioms rd2.sigma-finite-measure-axioms*)
have $?lhs = \mu \ggg= (\lambda r.\ \nu \ggg= (\lambda l.\ distr\ (return\ (real\text{-}borel \bigotimes_M real\text{-}borel)\ (r,l))\ real\text{-}borel\ real\text{-}real.f))$
using *pair-return-return*[*of - real-borel - real-borel*] **by** *simp*
also have ... $= \mu \ggg= (\lambda r.\ \nu \ggg= (\lambda l.\ distr\ (return\ (\mu \bigotimes_M \nu)\ (r,\ l))\ real\text{-}borel\ real\text{-}real.f))$
proof −
have $return\ (real\text{-}borel \bigotimes_M real\text{-}borel) = return\ (\mu \bigotimes_M \nu)$
by(*auto intro*!: *return-sets-cong sets-pair-measure-cong*)
thus *?thesis* **by** *simp*
qed

also have ... = $\mu \gg= (\lambda r.$ *distr* $(\nu \gg= (\lambda l.$ (*return* $(\mu \bigotimes_M \nu)$ $(r, l))))$ *real-borel real-real.f*)
by(*auto intro*!: *bind-cong distr-bind*[*symmetric*,**where** $K{=}\mu \bigotimes_M \nu$])
also have ... = *distr* $(\mu \gg= (\lambda r.\ \nu \gg= (\lambda l.$ *return* $(\mu \bigotimes_M \nu)$ $(r, l))))$ *real-borel real-real.f*
by(*auto intro*!: *distr-bind*[*symmetric*,**where** $K{=}\mu \bigotimes_M \nu$])
also have ... = *?rhs*
by(*simp add*: *pp.pair-measure-eq-bind*[*symmetric*])
finally show *?thesis* **.**
qed

lemma(**in** *pair-qbs-probs*) *qbs-bind-return-qp*:
shows *qbs-prob-space* $(Y, \beta, \nu) \gg= (\lambda y.$ *qbs-prob-space* $(X, \alpha, \mu) \gg= (\lambda x.$ *qbs-return* $(X \bigotimes_Q Y)$ $(x,y)))$ = *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* α $\beta \circ$ *real-real.g*, *distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*)
qbs-prob $(X \bigotimes_Q Y)$ (*map-prod* α $\beta \circ$ *real-real.g*) (*distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*)
proof −
show *qbs-prob-space* $(Y, \beta, \nu) \gg= (\lambda y.$ *qbs-prob-space* $(X, \alpha, \mu) \gg= (\lambda x.$ *qbs-return* $(X \bigotimes_Q Y)$ $(x, y)))$ = *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* α $\beta \circ$ *real-real.g*, *distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* α $\beta \circ$ *real-real.g*, $\nu \gg= (\lambda l.\ \mu \gg= (\lambda r.$ *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel l*) *real-borel real-real.f*)))
proof(*auto intro*!: *qp2.qbs-bind-computation*(*2*) *measurable-bind-prob-space2*[**where** *N=real-borel*] *simp*: *in-Mx*[*simplified*])
show $(\lambda y.$ *qbs-prob-space* $(X, \alpha, \mu) \gg= (\lambda x.$ *qbs-return* $(X \bigotimes_Q Y)$ $(x, y)))$ $\in Y \to_Q$ *monadP-qbs* $(X \bigotimes_Q Y)$
using *qbs-morphism-const*[*of* - *monadP-qbs X Y*,*simplified*,*OF qp1.qbs-prob-space-in-Px*] *curry-preserves-morphisms*[*OF qbs-morphism-pair-swap*[*OF qbs-return-morphism*[*of* $X \bigotimes_Q Y$]]]
by (*auto intro*!: *qbs-bind-morphism*)
next
show $(\lambda y.$ *qbs-prob-space* $(X, \alpha, \mu) \gg= (\lambda x.$ *qbs-return* $(X \bigotimes_Q Y)$ $(x, y)))$ $\circ\ \beta = (\lambda r.$ *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* α $\beta \circ$ *real-real.g*, $\mu \gg= (\lambda l.$ *distr* (*return real-borel l* $\bigotimes_M$ *return real-borel r*) *real-borel real-real.f*)))
by *standard*
(*auto intro*!: *qp1.qbs-bind-computation*(*2*) *qbs-morphism-comp*[*OF qbs-morphism-Pair2*[*of* - *Y*] *qbs-return-morphism*[*of* $X \bigotimes_Q Y$],*simplified comp-def*]
simp: *in-Mx*[*simplified*] *qbs-return-Mxpair*[*OF qp1.in-Mx qp2.in-Mx*] *qbs-Mx-to-X*(*2*))
qed
also have ... = *?rhs*
proof −
have $\nu \gg= (\lambda l.\ \mu \gg= (\lambda r.$ *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel l*) *real-borel real-real.f*)) = *distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*
by(*auto intro*!: *bind-rotate*[*symmetric*,**where** *N=real-borel*] *measurable-prob-algebraD*
simp: *bind-bind-return-distr*[*symmetric*,*OF qp1.real-distribution-axioms*

qp2.real-distribution-axioms])
      thus *?thesis* **by** *simp*
    qed
    finally show *?thesis* **.**
  qed
  show *qbs-prob* ($X \bigotimes_Q Y$) (*map-prod* α β $\circ$ *real-real.g*) (*distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
    by(*rule qbs-prob-axioms*)
qed

lemma(**in** *pair-qbs-probs*) *qbs-bind-return-pq*:
  shows *qbs-prob-space* (X, α, μ) $\gg=$ (λx. *qbs-prob-space* (Y, β, ν) $\gg=$ (λy. *qbs-return* ($X \bigotimes_Q Y$) (x,y))) = *qbs-prob-space* ($X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g*, *distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
       qbs-prob ($X \bigotimes_Q Y$) (*map-prod* α β $\circ$ *real-real.g*) (*distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
proof(*simp-all add*: *qbs-bind-return-qp*(*2*))
 show *qbs-prob-space* (X, α, μ) $\gg=$ (λx. *qbs-prob-space* (Y, β, ν) $\gg=$ (λy. *qbs-return* ($X \bigotimes_Q Y$) (x, y))) = *qbs-prob-space* ($X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g*, *distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
       (**is** *?lhs* = -)
  proof −
   have *?lhs* = *qbs-prob-space* ($X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g*, μ $\gg=$ (λr. ν $\gg=$ (λl. *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel l*) *real-borel real-real.f*)))
   proof(*auto intro*!: *qp1.qbs-bind-computation*(*2*) *measurable-bind-prob-space2*[**where** *N*=*real-borel*])
      show (λx. *qbs-prob-space* (Y, β, ν) $\gg=$ (λy. *qbs-return* ($X \bigotimes_Q Y$) (x, y))) $\in X \to_Q$ *monadP-qbs* ($X \bigotimes_Q Y$)
      using *qbs-morphism-const*[*of* - *monadP-qbs Y X*,*simplified*,*OF qp2.qbs-prob-space-in-Px*] *curry-preserves-morphisms*[*OF qbs-return-morphism*[*of* $X \bigotimes_Q Y$]]
        by(*auto intro*!: *qbs-bind-morphism simp*: *curry-def*)
    next
      show (λx. *qbs-prob-space* (Y, β, ν) $\gg=$ (λy. *qbs-return* ($X \bigotimes_Q Y$) (x, y))) $\circ$ α = (λr. *qbs-prob-space* ($X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g*, ν $\gg=$ (λl. *distr* (*return real-borel r* $\bigotimes_M$ *return real-borel l*) *real-borel real-real.f*)))
        by *standard*
       (*auto intro*!: *qp2.qbs-bind-computation*(*2*) *qbs-morphism-comp*[*OF qbs-morphism-Pair1* [*of* - *X*] *qbs-return-morphism*[*of* $X \bigotimes_Q Y$],*simplified comp-def*]
              simp: *qbs-return-Mxpair*[*OF qp1.in-Mx qp2.in-Mx*] *qbs-Mx-to-X*(*2*))
    qed
    thus *?thesis*
     by(*simp add*: *bind-bind-return-distr*[*OF qp1.real-distribution-axioms qp2.real-distribution-axioms*])
  qed
qed

lemma *qbs-bind-return-rotate*:
  assumes $p \in$ *monadP-qbs-Px X*
     and $q \in$ *monadP-qbs-Px Y*
   shows q $\gg=$ (λy. p $\gg=$ (λx. *qbs-return* ($X \bigotimes_Q Y$) (x,y))) = p $\gg=$ (λx. q $\gg=$

($\lambda y.$ *qbs-return* ($X \bigotimes_Q Y$) (x,y)))
proof −
obtain α μ **where** *hp*:
qbs-prob X α μ p = *qbs-prob-space* (X, α, μ)
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
obtain β ν **where** *hq*:
qbs-prob Y β ν q = *qbs-prob-space* (Y, β, ν)
using *rep-monadP-qbs-Px*[*OF assms*(*2*)] **by** *auto*
interpret *pqp*: *pair-qbs-probs* X α μ Y β ν
by(*simp add*: *pair-qbs-probs-def hp hq*)
show *?thesis*
by(*simp add*: *hp*(*2*) *hq*(*2*) *pqp.qbs-bind-return-pq*(*1*) *pqp.qbs-bind-return-qp*)
qed

lemma *qbs-pair-bind-return1*:
assumes $f \in X \bigotimes_Q Y \to_Q$ *monadP-qbs* Z
$p \in$ *monadP-qbs-Px* X
and $q \in$ *monadP-qbs-Px* Y
shows $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) = (q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.$ *qbs-return* (X
$\bigotimes_Q Y$) (x,y)))) $\gg\!= f$
(**is** *?lhs* = *?rhs*)
proof −
note [*simp*] = *qbs-morphism-const*[*of - monadP-qbs X*,*simplified*,*OF assms*(*2*)]
qbs-morphism-Pair1'[*OF - assms*(*1*)] *qbs-morphism-Pair2'*[*OF -*
assms(*1*)]
curry-preserves-morphisms[*OF qbs-morphism-pair-swap*[*OF qbs-return-morphism*[*of*
$X \bigotimes_Q Y$]],*simplified curry-def*,*simplified*]
qbs-morphism-Pair2'[*OF - qbs-return-morphism*[*of* $X \bigotimes_Q Y$]]
arg-swap-morphism[*OF curry-preserves-morphisms*[*OF assms*(*1*)],*simplified*
curry-def]
curry-preserves-morphisms[*OF qbs-morphism-comp*[*OF qbs-morphism-pair-swap*[*OF*
qbs-return-morphism[*of* $X \bigotimes_Q Y$]] *qbs-bind-morphism'*[*OF assms*(*1*)]],*simplified*
curry-def comp-def,*simplified*]
have [*simp*]:($\lambda y.\ p \gg\!= (\lambda x.\ f\ (x,y))) \in Y \to_Q$ *monadP-qbs* Z
($\lambda y.\ p \gg\!= (\lambda x.$ *qbs-return* ($X \bigotimes_Q Y$) (x,y) $\gg\!= f$)) $\in Y \to_Q$ *monadP-qbs*
Z
by(*auto intro*!: *qbs-bind-morphism*[**where** Y=X] *simp*: *curry-def*)
have *?lhs* = $q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.$ *qbs-return* ($X \bigotimes_Q Y$) (x,y) $\gg\!= f$))
by(*auto intro*!: *qbs-bind-cong*[*OF assms*(*3*),**where** Y=Z] *qbs-bind-cong*[*OF*
assms(*2*),**where** Y=Z] *simp*: *qbs-bind-return*[*OF assms*(*1*)])
also have ... = $q \gg\!= (\lambda y.\ (p \gg\!= (\lambda x.$ *qbs-return* ($X \bigotimes_Q Y$) (x,y))) $\gg\!= f$)
by(*auto intro*!: *qbs-bind-cong*[*OF assms*(*3*),**where** Y=Z] *qbs-bind-assoc*[*OF*
assms(*2*) *- assms*(*1*)] *simp*:)
also have ... = *?rhs*
by(*auto intro*!: *qbs-bind-assoc*[*OF assms*(*3*)*- assms*(*1*)] *qbs-bind-morphism*[**where**
Y=X])
finally show *?thesis* **.**
qed

lemma *qbs-pair-bind-return2*:
assumes $f \in X \bigotimes_Q Y \to_Q$ *monadP-qbs Z*
$p \in$ *monadP-qbs-Px X*
and $q \in$ *monadP-qbs-Px Y*
shows $p \gg= (\lambda x.\ q \gg= (\lambda y.\ f\ (x,y))) = (p \gg= (\lambda x.\ q \gg= (\lambda y.$ *qbs-return* $(X \bigotimes_Q Y)\ (x,y)))) \gg= f$
(**is** *?lhs* = *?rhs*)
proof −
note [*simp*] = *qbs-morphism-const*[*of - monadP-qbs Y*,*simplified*,*OF assms*(*3*)]
qbs-morphism-Pair1'[*OF - assms*(*1*)] *curry-preserves-morphisms*[*OF assms*(*1*),*simplified curry-def*]
qbs-morphism-Pair1'[*OF - qbs-return-morphism*[*of* $X \bigotimes_Q Y$]]
curry-preserves-morphisms[*OF qbs-morphism-comp*[*OF qbs-return-morphism*[*of* $X \bigotimes_Q Y$] *qbs-bind-morphism'*[*OF assms*(*1*)]],*simplified curry-def comp-def*,*simplified*]
curry-preserves-morphisms[*OF qbs-return-morphism*[*of* $X \bigotimes_Q Y$],*simplified curry-def*]
have [*simp*]: $(\lambda x.\ q \gg= (\lambda y.\ f\ (x,\ y))) \in X \to_Q$ *monadP-qbs Z*
$(\lambda x.\ q \gg= (\lambda y.$ *qbs-return* $(X \bigotimes_Q Y)\ (x,\ y) \gg= f)) \in X \to_Q$ *monadP-qbs Z*
by(*auto intro*!: *qbs-bind-morphism*[**where** *Y*=*Y*])
have *?lhs* = $p \gg= (\lambda x.\ q \gg= (\lambda y.$ *qbs-return* $(X \bigotimes_Q Y)\ (x,y) \gg= f))$
by(*auto intro*!: *qbs-bind-cong*[*OF assms*(*2*),**where** *Y*=*Z*] *qbs-bind-cong*[*OF assms*(*3*),**where** *Y*=*Z*] *simp*: *qbs-bind-return*[*OF assms*(*1*)])
also have ... = $p \gg= (\lambda x.\ (q \gg= (\lambda y.$ *qbs-return* $(X \bigotimes_Q Y)\ (x,y))) \gg= f)$
by(*auto intro*!: *qbs-bind-cong*[*OF assms*(*2*),**where** *Y*=*Z*] *qbs-bind-assoc*[*OF assms*(*3*) - *assms*(*1*)]])
also have ... = *?rhs*
by(*auto intro*!: *qbs-bind-assoc*[*OF assms*(*2*) - *assms*(*1*)] *qbs-bind-morphism*[**where** *Y*=*Y*])
finally show *?thesis* **.**
qed

lemma *qbs-bind-rotate*:
assumes $f \in X \bigotimes_Q Y \to_Q$ *monadP-qbs Z*
$p \in$ *monadP-qbs-Px X*
and $q \in$ *monadP-qbs-Px Y*
shows $q \gg= (\lambda y.\ p \gg= (\lambda x.\ f\ (x,y))) = p \gg= (\lambda x.\ q \gg= (\lambda y.\ f\ (x,y)))$
using *qbs-pair-bind-return1*[*OF assms*(*1*) *assms*(*2*) *assms*(*3*)] *qbs-bind-return-rotate*[*OF assms*(*2*) *assms*(*3*)] *qbs-pair-bind-return2*[*OF assms*(*1*) *assms*(*2*) *assms*(*3*)]
by *simp*

lemma(**in** *pair-qbs-probs*) *qbs-bind-bind-return*:
assumes $f \in X \bigotimes_Q Y \to_Q Z$
shows *qbs-prob Z* $(f \circ ($*map-prod* $\alpha\ \beta \circ$ *real-real.g*$))$ $($*distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*$)$
and *qbs-prob-space* $(X,\alpha,\mu) \gg= (\lambda x.$ *qbs-prob-space* $(Y,\beta,\nu) \gg= (\lambda y.$ *qbs-return* $Z\ (f\ (x,y)))) =$ *qbs-prob-space* $(Z,f \circ ($*map-prod* $\alpha\ \beta \circ$ *real-real.g*$)$,*distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*$)$

(**is** *?lhs* = *?rhs*)
proof −
show *qbs-prob Z* ($f \circ$ (*map-prod* α β $\circ$ *real-real.g*)) (*distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
using *qbs-bind-return-qp*(*2*) *qbs-morphismE*(*3*)[*OF assms*] **by**(*simp add*: *qbs-prob-def in-Mx-def*)
next
have *?lhs* = (*qbs-prob-space* (X,α,μ) $\gg=$ (λx. *qbs-prob-space* (Y,β,ν) $\gg=$ (λy. *qbs-return* ($X \bigotimes_Q Y$) (x,y)))) $\gg=$ *qbs-return Z* $\circ f$
using *qbs-pair-bind-return2*[*OF qbs-morphism-comp*[*OF assms qbs-return-morphism*] *qp1.qbs-prob-space-in-Px qp2.qbs-prob-space-in-Px*]
by(*simp add*: *comp-def*)
also have ... = *qbs-prob-space* ($X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g*, *distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*) $\gg=$ *qbs-return Z* $\circ f$
by(*simp add*: *qbs-bind-return-pq*(*1*))
also have ... = *?rhs*
by(*rule monadP-qbs-Pf-computation*[*OF refl assms*,*simplified monadP-qbs-Pf-def*])
finally show *?lhs* = *?rhs* **.**
qed

3.2.7 Properties of Return and Bind

lemma *qbs-prob-measure-return*:
assumes $x \in$ *qbs-space X*
shows *qbs-prob-measure* (*qbs-return X x*) = *return* (*qbs-to-measure X*) *x*
proof −
interpret *qp*: *qbs-prob X* λr. *x return real-borel 0*
by(*auto intro*!: *qbs-probI simp*: *prob-space-return assms*)
show *?thesis*
by(*simp add*: *qp.qbs-return-computation*[*OF assms*] *distr-return*)
qed

lemma *qbs-prob-measure-bind*:
assumes $s \in$ *monadP-qbs-Px X*
and $f \in X \to_Q$ *monadP-qbs Y*
shows *qbs-prob-measure* ($s \gg= f$) = *qbs-prob-measure s* $\gg=$ *qbs-prob-measure* $\circ f$
(**is** *?lhs* = *?rhs*)
proof −
obtain α μ **where** *hs*:
qbs-prob X α μ *s* = *qbs-prob-space* (X, α, μ)
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *blast*
hence $f \circ \alpha \in$ *monadP-qbs-MPx Y*
using *assms*(*2*) **by**(*auto simp*: *qbs-prob-def in-Mx-def*)
then obtain β *g* **where** *hbg*:
$\beta \in$ *qbs-Mx Y g* $\in$ *real-borel* $\to_M$ *prob-algebra real-borel*
($f \circ \alpha$) = (λr. *qbs-prob-space* (Y, β, *g r*))
using *rep-monadP-qbs-MPx* **by** *blast*
note [*measurable*] = *hbg*(*2*)

```
  have [measurable]:f ∈ qbs-to-measure X →M qbs-to-measure (monadP-qbs Y)
    using l-preserves-morphisms assms(2) by auto
  interpret pqp: pair-qbs-probs X α μ Y β μ ≫= g
      by(simp add: pair-qbs-probs-def hs(1) qbs-prob.qbs-bind-computation[OF hs
assms(2) hbg])

  have ?lhs = distr (μ ≫= g) (qbs-to-measure Y) β
    by(simp add: pqp.qp1.qbs-bind-computation[OF hs(2) assms(2) hbg])
  also have ... = μ ≫= (λx. distr (g x) (qbs-to-measure Y) β)
    by(auto intro!: distr-bind[where K=real-borel] measurable-prob-algebraD)
  also have ... = μ ≫= (λx. qbs-prob-measure (qbs-prob-space (Y,β,g x)))
    using measurable-space[OF hbg(2)]
      by(auto intro!: bind-cong qbs-prob.qbs-prob-measure-computation[symmetric]
qbs-probI simp: space-prob-algebra)
  also have ... = μ ≫= (λx. qbs-prob-measure ((f ∘ α) x))
    by(simp add: hbg(3))
  also have ... = μ ≫= (λx. (qbs-prob-measure ∘ f) (α x)) by simp
  also have ... = distr μ (qbs-to-measure X) α ≫= qbs-prob-measure ∘ f
    by(intro bind-distr[symmetric,where K=qbs-to-measure Y]) auto
  also have ... = ?rhs
    by(simp add: hs(2))
  finally show ?thesis .
qed

lemma qbs-of-return:
  assumes x ∈ qbs-space X
  shows qbs-prob-space-qbs (qbs-return X x) = X
  using real-distribution.qbs-return-computation[OF - assms,of return real-borel 0]
        qbs-prob.qbs-prob-space-qbs-computation[of X λr. x return real-borel 0] assms
 by(auto simp: qbs-prob-def in-Mx-def real-distribution-def real-distribution-axioms-def
prob-space-return)

lemma qbs-of-bind:
  assumes s ∈ monadP-qbs-Px X
      and f ∈ X →Q monadP-qbs Y
    shows qbs-prob-space-qbs (s ≫= f) = Y
proof −
  obtain α μ where hs:
   qbs-prob X α μ s = qbs-prob-space (X, α, μ)
    using rep-monadP-qbs-Px[OF assms(1)] by auto
  hence f ∘ α ∈ monadP-qbs-MPx Y
    using assms(2) by(auto simp: qbs-prob-def in-Mx-def)
  then obtain β g where hbg:
      β ∈ qbs-Mx Y g ∈ real-borel →M prob-algebra real-borel
      (f ∘ α) = (λr. qbs-prob-space (Y, β, g r))
    using rep-monadP-qbs-MPx by blast
  show ?thesis
   using qbs-prob.qbs-bind-computation[OF hs assms(2) hbg] qbs-prob.qbs-prob-space-qbs-computation
    by simp
```

qed

3.2.8 Properties of Integrals

lemma *qbs-integrable-return*:
assumes $x \in$ *qbs-space X*
and $f \in X \to_Q \mathbb{R}_Q$
shows *qbs-integrable* (*qbs-return X x*) *f*
using *assms*(*2*) *nn-integral-return*[*of x qbs-to-measure X* $\lambda x.$ $|f\ x|$,*simplified*,*OF assms*(*1*)]
by(*auto intro*!: *qbs-integrable-if-integrable integrableI-bounded*
simp: *qbs-prob-measure-return*[*OF assms*(*1*)])

lemma *qbs-integrable-bind-return*:
assumes $s \in$ *monadP-qbs-Px Y*
$f \in Z \to_Q \mathbb{R}_Q$
and $g \in Y \to_Q Z$
shows *qbs-integrable* ($s \gg\!= (\lambda y.$ *qbs-return Z* ($g\ y$))) f = *qbs-integrable s* ($f \circ g$)
proof −
obtain α μ **where** *hs*:
qbs-prob Y α μ s = *qbs-prob-space* (Y, α, μ)
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
then interpret *qp*: *qbs-prob Y* α μ **by** *simp*
show *?thesis* (**is** *?lhs* = *?rhs*)
proof −
have *qbs-return Z* $\circ$ ($g \circ \alpha$) = ($\lambda r.$ *qbs-prob-space* (Z, $g \circ \alpha$, *return real-borel r*))
by(*rule qbs-return-comp*) (*use assms*(*3*) *qp.in-Mx* **in** *blast*)
hence *hb*:*qbs-prob Z* ($g \circ \alpha$) μ
$s \gg\!= (\lambda y.$ *qbs-return Z* ($g\ y$)) = *qbs-prob-space* (Z, $g \circ \alpha$, μ)
by(*auto intro*!: *qbs-prob.qbs-bind-computation*[*OF hs qbs-morphism-comp*[*OF assms*(*3*) *qbs-return-morphism*,*simplified comp-def*] *qbs-morphismE*(*3*)[*OF assms*(*3*) *qp.in-Mx*],*of return real-borel*,*simplified bind-return''*[*of* μ *real-borel*,*simplified*]])
(*simp-all add*: *comp-def*)
have *?lhs* = *integrable* μ ($f \circ (g \circ \alpha)$)
using *assms*(*2*)
by(*auto intro*!: *qbs-prob.qbs-integrable-iff-integrable*[*OF hb*(*1*),*simplified comp-def*] *simp*: *hb*(*2*) *comp-def*)
also have ... = *?rhs*
using *qbs-morphism-comp*[*OF assms*(*3*,*2*)]
by(*auto intro*!: *qbs-prob.qbs-integrable-iff-integrable*[*OF hs*(*1*),*symmetric*] *simp*: *hs*(*2*) *comp-def*)
finally show *?thesis* **.**
qed
qed

lemma *qbs-prob-ennintegral-morphism*:

assumes $L \in X \to_Q$ *monadP-qbs Y*
and $f \in X \to_Q$ *exp-qbs Y* $\mathbb{R}_{Q\geq 0}$
shows ($\lambda x.$ *qbs-prob-ennintegral* (L x) (f x)) $\in X \to_Q \mathbb{R}_{Q\geq 0}$
proof(*rule qbs-morphismI*,*simp-all*)
fix α
assume *h0*:$\alpha \in$ *qbs-Mx X*
then obtain β g **where** h:
$\beta \in$ *qbs-Mx Y* $g \in$ *real-borel* $\to_M$ *prob-algebra real-borel*
($L \circ \alpha$) = ($\lambda r.$ *qbs-prob-space* (Y, β, g r))
using *rep-monadP-qbs-MPx*[*of* $L \circ \alpha$ Y] *qbs-morphismE*(*3*)[*OF assms*(*1*)] **by** *auto*
note [*measurable*] = *h*(*2*)
have [*measurable*]: ($\lambda(r, y).$ f (α r) (β y)) $\in$ *real-borel* $\bigotimes_M$ *real-borel* $\to_M$ *ennreal-borel*
proof −
have ($\lambda(r, y).$ f (α r) (β y)) = *case-prod f* $\circ$ *map-prod* α β
by *auto*
also have ... $\in \mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q \to_Q \mathbb{R}_{Q\geq 0}$
apply(*rule qbs-morphism-comp*[*OF qbs-morphism-map-prod uncurry-preserves-morphisms*[*OF assms*(*2*)]])
using *h0 h*(*1*) **by**(*auto simp*: *qbs-Mx-is-morphisms*)
finally show *?thesis*
by *auto*
qed
have ($\lambda x.$ *qbs-prob-ennintegral* (L x) (f x)) $\circ$ α = ($\lambda r.$ *qbs-prob-ennintegral* ((L $\circ$ α) r) ((f $\circ$ α) r))
by *auto*
also have ... = ($\lambda r.$ ($\int^+$ $x.$ ($f \circ \alpha$) r (β x) ∂(g r)))
apply *standard*
using *h0* **by**(*auto intro*!: *qbs-prob.qbs-prob-ennintegral-def*[*OF qbs-prob-MPx*[*OF h*(*1*,*2*)]] *qbs-morphismE*(*2*)[*OF assms*(*2*),*simplified*] *simp*: *h*(*3*))
also have ... $\in$ *real-borel* $\to_M$ *ennreal-borel*
using *assms*(*2*) *h0 h*(*1*)
by(*auto intro*!: *nn-integral-measurable-subprob-algebra2*[**where** *N*=*real-borel*] *simp*: *measurable-prob-algebraD*)
finally show ($\lambda x.$ *qbs-prob-ennintegral* (L x) (f x)) $\circ$ α $\in$ *real-borel* $\to_M$ *ennreal-borel* **.**
qed

lemma *qbs-morphism-ennintegral-fst*:
assumes $q \in$ *monadP-qbs-Px Y*
and $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_{Q\geq 0}$
shows ($\lambda x.$ $\int^+_Q$ $y.$ f (x, y) ∂q) $\in X \to_Q \mathbb{R}_{Q\geq 0}$
by(*rule qbs-prob-ennintegral-morphism*[*OF qbs-morphism-const*[*of - monadP-qbs Y*,*simplified*,*OF assms*(*1*)] *curry-preserves-morphisms*[*OF assms*(*2*)],*simplified curry-def*])

lemma *qbs-morphism-ennintegral-snd*:
assumes $p \in$ *monadP-qbs-Px X*
and $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_{Q\geq 0}$

shows $(\lambda y.\ \int^{+}_{Q}\ x.\ f\ (x,\ y)\ \partial p) \in Y \rightarrow_Q \mathbb{R}_{Q\geq 0}$
using *qbs-morphism-ennintegral-fst*[*OF assms*(*1*) *qbs-morphism-pair-swap*[*OF assms*(*2*)]]
by *fastforce*

lemma *qbs-prob-ennintegral-morphism′*:
assumes $f \in X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
shows ($\lambda s.$ *qbs-prob-ennintegral s f*) $\in$ *monadP-qbs* $X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
apply(*rule qbs-prob-ennintegral-morphism*[*of - - X*])
using *qbs-morphism-ident*[*of monadP-qbs X*]
apply (*simp add*: *id-def*)
using *assms qbs-morphism-const*[*of f exp-qbs X* $\mathbb{R}_{Q\geq 0}$]
by *simp*

lemma *qbs-prob-ennintegral-return*:
assumes $f \in X \rightarrow_Q \mathbb{R}_{Q\geq 0}$
and $x \in$ *qbs-space X*
shows *qbs-prob-ennintegral* (*qbs-return X x*) $f = f\ x$
using *assms*
by(*auto intro*!: *nn-integral-return*
simp: *qbs-prob-ennintegral-def2*[*OF qbs-of-return*[*OF assms*(*2*)] *assms*(*1*)]
qbs-prob-measure-return[*OF assms*(*2*)])

lemma *qbs-prob-ennintegral-bind*:
assumes $s \in$ *monadP-qbs-Px X*
$f \in X \rightarrow_Q$ *monadP-qbs Y*
and $g \in Y \rightarrow_Q \mathbb{R}_{Q\geq 0}$
shows *qbs-prob-ennintegral* ($s \gg= f$) g = *qbs-prob-ennintegral s* ($\lambda y.$ (*qbs-prob-ennintegral* ($f\ y$) g))
(**is** *?lhs* = *?rhs*)
proof −
obtain $\alpha\ \mu$ **where** *hs*:
qbs-prob X $\alpha\ \mu$ s = *qbs-prob-space* (X, α, μ)
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
then interpret *qp*: *qbs-prob X* $\alpha\ \mu$ **by** *simp*
obtain $\beta\ h$ **where** *hb*:
$\beta \in$ *qbs-Mx Y* $h \in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
$(f \circ \alpha) = (\lambda r.$ *qbs-prob-space* (Y, β, $h\ r$))
using *rep-monadP-qbs-MPx*[*OF qbs-morphismE*(*3*)[*OF assms*(*2*) *qp.in-Mx*,*simplified*]]
by *auto*
hence *h*:*qbs-prob Y* β ($\mu \gg= h$)
$s \gg= f$ = *qbs-prob-space* (Y, β, $\mu \gg= h$)
using *qp.qbs-bind-computation*[*OF hs*(*2*) *assms*(*2*) *hb*] **by** *auto*
hence *LHS*:*?lhs* = ($\int^{+}\ x.\ g\ (\beta\ x)\ \partial(\mu \gg= h)$)
using *qbs-prob.qbs-prob-ennintegral-def*[*OF h*(*1*) *assms*(*3*)]
by *simp*
note [*measurable*] = *hb*(*2*)

have $\bigwedge r.$ *qbs-prob-ennintegral* (f ($\alpha\ r$)) g = ($\int^{+}\ y.\ g\ (\beta\ y)\ \partial(h\ r)$)
using *qbs-prob.qbs-prob-ennintegral-def*[*OF qbs-prob-MPx*[*OF hb*(*1*,*2*)] *assms*(*3*)]

```
hb(3)[simplified comp-def]
    by metis
  hence ?rhs = (∫+ r. (∫+ y. (g ∘ β) y ∂(h r)) ∂μ)
    by(auto intro!: nn-integral-cong
          simp: qbs-prob.qbs-prob-ennintegral-def[OF hs(1) qbs-prob-ennintegral-morphism[OF
assms(2) qbs-morphism-const[of - exp-qbs Y ℝQ≥0 ,simplified,OF assms(3)]]] hs(2))
  also have ... = (integralN (μ ≫= h) (g ∘ β))
    apply(intro nn-integral-bind[symmetric,of - real-borel])
    using assms(3) hb(1)
    by(auto intro!: measurable-prob-algebraD hb(2))
  finally show ?thesis
    using LHS by(simp add: comp-def)
qed

lemma qbs-prob-ennintegral-bind-return:
  assumes s ∈ monadP-qbs-Px Y
          f ∈ Z →Q ℝQ≥0
      and g ∈ Y →Q Z
   shows qbs-prob-ennintegral (s ≫= (λy. qbs-return Z (g y))) f = qbs-prob-ennintegral
s (f ∘ g)
  apply(simp add: qbs-prob-ennintegral-bind[OF assms(1) qbs-return-morphism'[OF
assms(3)] assms(2)])
  using assms(1,3)
  by(auto intro!: qbs-prob-ennintegral-cong qbs-prob-ennintegral-return[OF assms(2)]
            simp: monadP-qbs-Px-def)

lemma qbs-prob-integral-morphism':
  assumes f ∈ X →Q ℝQ
  shows (λs. qbs-prob-integral s f) ∈ monadP-qbs X →Q ℝQ
proof(rule qbs-morphismI;simp)
  fix α
  assume α ∈ monadP-qbs-MPx X
  then obtain β g where h:
   β ∈ qbs-Mx X g ∈ real-borel →M prob-algebra real-borel
   α = (λr. qbs-prob-space (X, β, g r))
    using rep-monadP-qbs-MPx[of α X] by auto
  note [measurable] = h(2)
  have [measurable]: f ∘ β ∈ real-borel →M real-borel
    using assms h(1) by auto
  have (λs. qbs-prob-integral s f) ∘ α = (λr. ∫ x. f (β x) ∂g r)
    apply standard
   using assms qbs-prob-MPx[OF h(1,2)] by(auto intro!: qbs-prob.qbs-prob-integral-def
simp: h(3))
  also have ... = (λM. integralL M (f ∘ β)) ∘ g
    by (simp add: comp-def)
  also have ... ∈ real-borel →M real-borel
    by(auto intro!: measurable-comp[where N=subprob-algebra real-borel]
             simp: integral-measurable-subprob-algebra measurable-prob-algebraD)
  finally show (λs. qbs-prob-integral s f) ∘ α ∈ real-borel →M real-borel .
```

qed

lemma *qbs-morphism-integral-fst*:
  assumes $q \in$ *monadP-qbs-Px Y*
    and $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
   shows $(\lambda x. \int_Q y.\ f\ (x,\ y)\ \partial q) \in X \to_Q \mathbb{R}_Q$
proof(*rule qbs-morphismI*,*simp-all*)
  fix α
  assume *ha*:$\alpha \in$ *qbs-Mx X*
  obtain $\beta\ \nu$ **where** *hq*:
  qbs-prob $Y\ \beta\ \nu\ q =$ *qbs-prob-space* $(Y,\ \beta,\ \nu)$
   using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
  then interpret *qp*: *qbs-prob* $Y\ \beta\ \nu$ **by** *simp*
  have $(\lambda x. \int_Q y.\ f\ (x,\ y)\ \partial q) \circ \alpha = (\lambda x. \int y.\ f\ (\alpha\ x,\ \beta\ y)\ \partial \nu)$
   apply *standard*
   using *qbs-morphism-Pair1'*[*OF qbs-Mx-to-X*(*2*)[*OF ha*] *assms*(*2*)]
   by(*auto intro*!: *qp.qbs-prob-integral-def*
        simp: *hq*(*2*))
  also have ... $\in$ *borel-measurable borel*
    using *qbs-morphism-comp*[*OF qbs-morphism-map-prod assms*(*2*),*of* $\alpha\ \mathbb{R}_Q\ \beta$
$\mathbb{R}_Q$,*simplified comp-def map-prod-def split-beta'*] *ha qp.in-Mx*
   by(*auto intro*!: *qp.borel-measurable-lebesgue-integral*
        simp: *qbs-Mx-is-morphisms*)
  finally show $(\lambda x. \int_Q y.\ f\ (x,\ y)\ \partial q) \circ \alpha \in$ *borel-measurable borel* **.**
qed

lemma *qbs-morphism-integral-snd*:
  assumes $p \in$ *monadP-qbs-Px X*
    and $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
   shows $(\lambda y. \int_Q x.\ f\ (x,\ y)\ \partial p) \in Y \to_Q \mathbb{R}_Q$
 using *qbs-morphism-integral-fst*[*OF assms*(*1*) *qbs-morphism-pair-swap*[*OF assms*(*2*)]]
  by *simp*

lemma *qbs-prob-integral-morphism*:
  assumes $L \in X \to_Q$ *monadP-qbs Y*
      $f \in X \to_Q$ *exp-qbs* $Y\ \mathbb{R}_Q$
    and $\bigwedge x.\ x \in$ *qbs-space* $X \Longrightarrow$ *qbs-integrable* $(L\ x)\ (f\ x)$
   shows $(\lambda x.$ *qbs-prob-integral* $(L\ x)\ (f\ x)) \in X \to_Q \mathbb{R}_Q$
proof(*rule qbs-morphismI*;*simp*)
  fix α
  assume *h0*:$\alpha \in$ *qbs-Mx X*
  then obtain $\beta\ g$ **where** *h*:
  $\beta \in$ *qbs-Mx Y* $g \in$ *real-borel* $\to_M$ *prob-algebra real-borel*
  $(L \circ \alpha) = (\lambda r.$ *qbs-prob-space* $(Y,\ \beta,\ g\ r))$
   using *rep-monadP-qbs-MPx*[*of* $L \circ \alpha$ *Y*] *qbs-morphismE*(*3*)[*OF assms*(*1*)] **by**
auto
  have $(\lambda x.$ *qbs-prob-integral* $(L\ x)\ (f\ x)) \circ \alpha = (\lambda r.$ *qbs-prob-integral* $((L \circ \alpha)\ r)$
$((f \circ \alpha)\ r))$
   by *auto*

also have ... = ($\lambda r.$ *enn2real* (*qbs-prob-ennintegral* (($L \circ \alpha$) r) ($\lambda x.$ *ennreal* ((f $\circ$ α) r x)))
− *enn2real* (*qbs-prob-ennintegral* (($L \circ \alpha$) r) ($\lambda x.$ *ennreal* (− (f $\circ$ α) r x))))
using *h0 assms*(*3*) **by**(*auto intro*!: *real-qbs-prob-integral-def*)
also have ... $\in$ *real-borel* $\rightarrow_M$ *real-borel*
proof −
have *h2*:$L \circ \alpha \in \mathbb{R}_Q \rightarrow_Q$ *monadP-qbs* Y
using *qbs-morphismE*(*3*)[*OF assms*(*1*) *h0*] **by**(*auto simp*: *qbs-Mx-is-morphisms*)
have [*measurable*]:($\lambda x.$ f (*fst* x) (*snd* x)) $\in$ *qbs-to-measure* ($X \bigotimes_Q Y$) $\rightarrow_M$ *real-borel*
using *uncurry-preserves-morphisms*[*OF assms*(*2*)] **by**(*auto simp*: *split-beta'*)
have *h3*:(λr $x.$ *ennreal* (($f \circ \alpha$) r x)) $\in \mathbb{R}_Q \rightarrow_Q$ *exp-qbs* Y $\mathbb{R}_{Q\geq 0}$
proof(*auto intro*!: *curry-preserves-morphisms*[*of* ($\lambda(r,x).$ *ennreal* (($f \circ \alpha$) r x)),*simplified curry-def*,*simplified*])
have ($\lambda(r, y).$ *ennreal* (f (α r) y)) = *ennreal* $\circ$ *case-prod* f $\circ$ *map-prod* α *id*
by *auto*
also have ... $\in \mathbb{R}_Q \bigotimes_Q Y \rightarrow_Q \mathbb{R}_{Q\geq 0}$
apply(*rule qbs-morphism-comp*[**where** Y=$X \bigotimes_Q Y$])
using *h0 qbs-morphism-map-prod*[*OF* - *qbs-morphism-ident*,*of* α $\mathbb{R}_Q$ X Y]
by(*auto simp*: *qbs-Mx-is-morphisms*)
finally show ($\lambda(r, y).$ *ennreal* (f (α r) y)) $\in$ *qbs-to-measure* ($\mathbb{R}_Q \bigotimes_Q Y$) $\rightarrow_M$ *ennreal-borel*
by *auto*
qed
have *h4*:(λr $x.$ *ennreal* (− ($f \circ \alpha$) r x)) $\in \mathbb{R}_Q \rightarrow_Q$ *exp-qbs* Y $\mathbb{R}_{Q\geq 0}$
proof(*auto intro*!: *curry-preserves-morphisms*[*of* ($\lambda(r,x).$ *ennreal* (− ($f \circ \alpha$) r x)),*simplified curry-def*,*simplified*])
have ($\lambda(r, y).$ *ennreal* (− f (α r) y)) = *ennreal* $\circ$ ($\lambda r.$ − r) $\circ$ *case-prod* f $\circ$ *map-prod* α *id*
by *auto*
also have ... $\in \mathbb{R}_Q \bigotimes_Q Y \rightarrow_Q \mathbb{R}_{Q\geq 0}$
apply(*rule qbs-morphism-comp*[**where** Y=$X \bigotimes_Q Y$])
using *h0 qbs-morphism-map-prod*[*OF* - *qbs-morphism-ident*,*of* α $\mathbb{R}_Q$ X Y]
by(*auto simp*: *qbs-Mx-is-morphisms*)
finally show ($\lambda(r, y).$ *ennreal* (− f (α r) y)) $\in$ *qbs-to-measure* ($\mathbb{R}_Q \bigotimes_Q Y$) $\rightarrow_M$ *ennreal-borel*
by *auto*
qed
have ($\lambda r.$ *qbs-prob-ennintegral* (($L \circ \alpha$) r) ($\lambda x.$ *ennreal* (($f \circ \alpha$) r x))) $\in$ *real-borel* $\rightarrow_M$ *ennreal-borel*
($\lambda r.$ *qbs-prob-ennintegral* (($L \circ \alpha$) r) ($\lambda x.$ *ennreal* (− ($f \circ \alpha$) r x))) $\in$ *real-borel* $\rightarrow_M$ *ennreal-borel*
using *qbs-prob-ennintegral-morphism*[*OF h2 h3*] *qbs-prob-ennintegral-morphism*[*OF h2 h4*]
by *auto*
thus *?thesis* **by** *simp*
qed
finally show ($\lambda x.$ *qbs-prob-integral* (L x) (f x)) $\circ$ α $\in$ *real-borel* $\rightarrow_M$ *real-borel* **.**

qed

lemma *qbs-prob-integral-morphism''*:
assumes $f \in X \rightarrow_Q \mathbb{R}_Q$
and $L \in Y \rightarrow_Q$ *monadP-qbs* X
shows ($\lambda y.$ *qbs-prob-integral* $(L\ y)\ f) \in Y \rightarrow_Q \mathbb{R}_Q$
using *qbs-morphism-comp*[*OF assms*(*2*) *qbs-prob-integral-morphism'*[*OF assms*(*1*)]]
by(*simp add*: *comp-def*)

lemma *qbs-prob-integral-return*:
assumes $f \in X \rightarrow_Q \mathbb{R}_Q$
and $x \in$ *qbs-space* X
shows *qbs-prob-integral* (*qbs-return* $X\ x$) $f = f\ x$
using *assms*
by(*auto intro*!: *integral-return*
simp add: *qbs-prob-integral-def2 qbs-prob-measure-return*[*OF assms*(*2*)])

lemma *qbs-prob-integral-bind*:
assumes $s \in$ *monadP-qbs-Px* X
$f \in X \rightarrow_Q$ *monadP-qbs* Y
$g \in Y \rightarrow_Q \mathbb{R}_Q$
and $\exists K.\ \forall y \in$ *qbs-space* $Y.|g\ y| \le K$
shows *qbs-prob-integral* ($s \gg= f$) $g =$ *qbs-prob-integral* s ($\lambda y.$ (*qbs-prob-integral* ($f\ y$) g))
(**is** *?lhs* = *?rhs*)
proof −
obtain K **where** *hK*:
$\bigwedge y.\ y \in$ *qbs-space* $Y \Longrightarrow |g\ y| \le K$
using *assms*(*4*) **by** *auto*
obtain $\alpha\ \mu$ **where** *hs*:
qbs-prob $X\ \alpha\ \mu\ s =$ *qbs-prob-space* (X, α, μ)
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
then obtain $\beta\ h$ **where** *hb*:
$\beta \in$ *qbs-Mx* $Y\ h \in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
$(f \circ \alpha) = (\lambda r.$ *qbs-prob-space* $(Y, \beta, h\ r))$
using *rep-monadP-qbs-MPx*[*of f* $\circ$ α Y] *qbs-morphismE*(*3*)[*OF assms*(*2*)]
by(*auto simp add*: *qbs-prob-def in-Mx-def*)
note [*measurable*] = *hb*(*2*)
interpret *rd*: *real-distribution* μ **by**(*simp add*: *hs*(*1*)[*simplified qbs-prob-def*])
have *h*:*qbs-prob* $Y\ \beta\ (\mu \gg= h)$
$s \gg= f =$ *qbs-prob-space* $(Y, \beta, \mu \gg= h)$
using *qbs-prob.qbs-bind-computation*[*OF hs assms*(*2*) *hb*] **by** *auto*

hence *?lhs* = ($\int x.\ g\ (\beta\ x)\ \partial(\mu \gg= h)$)
by(*simp add*: *qbs-prob.qbs-prob-integral-def*[*OF h*(*1*) *assms*(*3*)])
also have ... = (*integral*^{L} $(\mu \gg= h)\ (g \circ \beta)$) **by**(*simp add*: *comp-def*)
also have ... = ($\int r.\ (\int y.\ (g \circ \beta)\ y\ \partial(h\ r))\ \partial\mu$)
apply(*rule integral-bind*[*of - real-borel K - - 1*])
using *assms*(*3*) *hb*(*1*) *hK measurable-space*[*OF hb*(*2*)]

by(*auto intro*!: *measurable-prob-algebraD*
simp: *space-prob-algebra prob-space.emeasure-le-1*)
also have ... = *?rhs*
by(*auto intro*!: *Bochner-Integration.integral-cong*
simp: *qbs-prob.qbs-prob-integral-def*[*OF qbs-prob-MPx*[*OF hb*(*1*,*2*)] *assms*(*3*)] *fun-cong*[*OF hb*(*3*),*simplified comp-def*] *hs*(*2*) *qbs-prob.qbs-prob-integral-def*[*OF hs*(*1*) *qbs-prob-integral-morphism''*[*OF assms*(*3*,*2*)]]])
finally show *?thesis* **.**
qed

lemma *qbs-prob-integral-bind-return*:
assumes $s \in$ *monadP-qbs-Px Y*
$f \in Z \to_Q \mathbb{R}_Q$
and $g \in Y \to_Q Z$
shows *qbs-prob-integral* ($s \gg= (\lambda y.$ *qbs-return* $Z\ (g\ y)))\ f =$ *qbs-prob-integral* s $(f \circ g)$
proof −
obtain α μ **where** *hs*:
qbs-prob $Y\ \alpha\ \mu\ s =$ *qbs-prob-space* (Y, α, μ)
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
then interpret *qp*: *qbs-prob* $Y\ \alpha\ \mu$ **by** *simp*
have *hb*:*qbs-prob* $Z\ (g \circ \alpha)\ \mu$
$s \gg= (\lambda y.$ *qbs-return* $Z\ (g\ y)) =$ *qbs-prob-space* $(Z, g \circ \alpha, \mu)$
by(*auto intro*!: *qp.qbs-bind-computation*[*OF hs*(*2*) *qbs-return-morphism'*[*OF assms*(*3*)] *qbs-morphismE*(*3*)[*OF assms*(*3*) *qp.in-Mx*],*of return real-borel*,*simplified bind-return''*[*of* μ *real-borel*,*simplified*] *comp-def*]
simp: *comp-def qbs-return-comp*[*OF qbs-morphismE*(*3*)[*OF assms*(*3*) *qp.in-Mx*],*simplified comp-def*])
thus *?thesis*
by(*simp add*: *hb*(*2*) *qbs-prob.qbs-prob-integral-def*[*OF hb*(*1*) *assms*(*2*)] *hs*(*2*) *qbs-prob.qbs-prob-integral-def*[*OF hs*(*1*) *qbs-morphism-comp*[*OF assms*(*3*,*2*)]]])
qed

lemma *qbs-prob-var-bind-return*:
assumes $s \in$ *monadP-qbs-Px Y*
$f \in Z \to_Q \mathbb{R}_Q$
and $g \in Y \to_Q Z$
shows *qbs-prob-var* ($s \gg= (\lambda y.$ *qbs-return* $Z\ (g\ y)))\ f =$ *qbs-prob-var* $s\ (f \circ g)$
proof −
have *1*:$(\lambda x. (f\ x -$ *qbs-prob-integral* $s\ (f \circ g))^2) \in Z \to_Q \mathbb{R}_Q$
using *assms*(*2*,*3*) **by** *auto*
thus *?thesis*
using *qbs-prob-integral-bind-return*[*OF assms*(*1*) *1 assms*(*3*)] *qbs-prob-integral-bind-return*[*OF assms*]
by(*simp add*: *comp-def qbs-prob-var-def*)
qed

end

3.3 Binary Product Measure

theory *Pair-QuasiBorel-Measure*
imports *Monad-QuasiBorel*
begin

3.3.1 Binary Product Measure

Special case of [1] Proposition 23 where $\Omega = \mathbb{R} \times \mathbb{R}$ and $X = X \times Y$. Let $[\alpha, \mu] \in P(X)$ and $[\beta, \nu] \in P(Y)$. $\alpha \times \beta$ is the α in Proposition 23.

definition *qbs-prob-pair-measure-t* :: [*'a qbs-prob-t*, *'b qbs-prob-t*] ⇒ (*'a* × *'b*) *qbs-prob-t* **where**
qbs-prob-pair-measure-t p q ≡ (*let* (*X*,*α*,*μ*) = *p*;
(*Y*,*β*,*ν*) = *q in*
(*X* ⨂$_Q$ *Y*, *map-prod α β* ∘ *real-real.g*, *distr* (*μ* ⨂$_M$ *ν*) *real-borel real-real.f*))

lift-definition *qbs-prob-pair-measure* :: [*'a qbs-prob-space*, *'b qbs-prob-space*] ⇒ (*'a* × *'b*) *qbs-prob-space* (**infix** ‹⨂$_{Qmes}$› *80*)
is *qbs-prob-pair-measure-t*
unfolding *qbs-prob-pair-measure-t-def*
proof *auto*
fix *X X'* :: *'a quasi-borel*
fix *Y Y'* :: *'b quasi-borel*
fix *α α' μ μ' β β' ν ν'*
assume *h*:*qbs-prob-eq* (*X*,*α*,*μ*) (*X'*,*α'*,*μ'*)
qbs-prob-eq (*Y*,*β*,*ν*) (*Y'*,*β'*,*ν'*)
then have *1*: *X* = *X' Y* = *Y'*
by(*auto simp*: *qbs-prob-eq-def*)
interpret *pqp1*: *pair-qbs-probs X α μ Y β ν*
by(*simp add*: *pair-qbs-probs-def qbs-prob-eq-dest*(*1*)[*OF h*(*1*)] *qbs-prob-eq-dest*(*1*)[*OF h*(*2*)])
interpret *pqp2*: *pair-qbs-probs X' α' μ' Y' β' ν'*
by(*simp add*: *pair-qbs-probs-def qbs-prob-eq-dest*(*2*)[*OF h*(*1*)] *qbs-prob-eq-dest*(*2*)[*OF h*(*2*)])
interpret *pqp*: *pair-qbs-prob X* ⨂$_Q$ *Y map-prod α β* ∘ *real-real.g distr* (*μ* ⨂$_M$ *ν*) *real-borel real-real.f X'* ⨂$_Q$ *Y' map-prod α' β'* ∘ *real-real.g distr* (*μ'* ⨂$_M$ *ν'*) *real-borel real-real.f*
by(*auto intro*!: *qbs-probI pqp1.P.prob-space-distr pqp2.P.prob-space-distr simp*: *pair-qbs-prob-def*)

show *qbs-prob-eq* (*X* ⨂$_Q$ *Y*, *map-prod α β* ∘ *real-real.g*, *distr* (*μ* ⨂$_M$ *ν*) *real-borel real-real.f*) (*X'* ⨂$_Q$ *Y'*, *map-prod α' β'* ∘ *real-real.g*, *distr* (*μ'* ⨂$_M$ *ν'*) *real-borel real-real.f*)
proof(*rule pqp.qbs-prob-space-eq-inverse*(*1*))
show *qbs-prob-space* (*X* ⨂$_Q$ *Y*, *map-prod α β* ∘ *real-real.g*, *distr* (*μ* ⨂$_M$ *ν*) *real-borel real-real.f*)
= *qbs-prob-space* (*X'* ⨂$_Q$ *Y'*, *map-prod α' β'* ∘ *real-real.g*, *distr* (*μ'* ⨂$_M$ *ν'*) *real-borel real-real.f*)

(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* = *qbs-prob-space* $(X, \alpha, \mu) \gg= (\lambda x.$ *qbs-prob-space* $(Y, \beta, \nu) \gg=$ $(\lambda y.$ *qbs-return* $(X \bigotimes_Q Y)$ $(x, y)))$
by(*simp add*: *pqp1.qbs-bind-return-pq*)
also have ... = *qbs-prob-space* $(X', \alpha', \mu') \gg= (\lambda x.$ *qbs-prob-space* $(Y', \beta',$ $\nu') \gg= (\lambda y.$ *qbs-return* $(X' \bigotimes_Q Y')$ $(x, y)))$
using *h* **by**(*simp add*: *qbs-prob-space-eq 1*)
also have ... = *?rhs*
by(*simp add*: *pqp2.qbs-bind-return-pq*)
finally show *?thesis* **.**
qed
qed
qed

lemma(**in** *pair-qbs-probs*) *qbs-prob-pair-measure-computation*:
(*qbs-prob-space* (X,α,μ)) $\bigotimes_{Qmes}$ (*qbs-prob-space* (Y,β,ν)) = *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* α β $\circ$ *real-real.g* , *distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*)
qbs-prob $(X \bigotimes_Q Y)$ (*map-prod* α β $\circ$ *real-real.g*) (*distr* $(\mu \bigotimes_M \nu)$ *real-borel real-real.f*)
by(*simp-all add*: *qbs-prob-pair-measure.abs-eq qbs-prob-pair-measure-t-def qbs-bind-return-pq*)

lemma *qbs-prob-pair-measure-qbs*:
qbs-prob-space-qbs $(p \bigotimes_{Qmes} q)$ = *qbs-prob-space-qbs p* $\bigotimes_Q$ *qbs-prob-space-qbs q*
by(*transfer*,*simp add*: *qbs-prob-pair-measure-t-def Let-def prod.case-eq-if*)

lemma(**in** *pair-qbs-probs*) *qbs-prob-pair-measure-measure*:
shows *qbs-prob-measure* (*qbs-prob-space* (X,α,μ) $\bigotimes_{Qmes}$ *qbs-prob-space* (Y,β,ν)) = *distr* $(\mu \bigotimes_M \nu)$ (*qbs-to-measure* $(X \bigotimes_Q Y)$) (*map-prod* α β)
by(*simp add*: *qbs-prob-pair-measure-computation distr-distr comp-assoc*)

lemma *qbs-prob-pair-measure-morphism*:
case-prod qbs-prob-pair-measure $\in$ *monadP-qbs X* $\bigotimes_Q$ *monadP-qbs Y* $\rightarrow_Q$ *monadP-qbs* $(X \bigotimes_Q Y)$
proof(*rule pair-qbs-morphismI*)
fix βx βy
assume *h*: $\beta x \in$ *qbs-Mx* (*monadP-qbs X*) $\beta y \in$ *qbs-Mx* (*monadP-qbs Y*)
then obtain αx αy *gx gy* **where** *ha*:
$\alpha x \in$ *qbs-Mx X gx* $\in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel* $\beta x = (\lambda r.$ *qbs-prob-space* $(X, \alpha x, gx\ r))$
$\alpha y \in$ *qbs-Mx Y gy* $\in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel* $\beta y = (\lambda r.$ *qbs-prob-space* $(Y, \alpha y, gy\ r))$
using *rep-monadP-qbs-MPx*[*of* βx *X*] *rep-monadP-qbs-MPx*[*of* βy *Y*] **by** *auto*
note [*measurable*] = *ha*(*2*,*5*)
have $(\lambda(x, y).\ x \bigotimes_{Qmes} y) \circ (\lambda r.\ (\beta x\ r, \beta y\ r)) = (\lambda r.$ *qbs-prob-space* $(X \bigotimes_Q Y$, *map-prod* αx αy $\circ$ *real-real.g*, *distr* $(gx\ r \bigotimes_M gy\ r)$ *real-borel real-real.f*))
apply *standard*
using *qbs-prob-MPx*[*OF ha*(*1*,*2*)] *qbs-prob-MPx*[*OF ha*(*4*,*5*)] *pair-qbs-probs.qbs-prob-pair-measure-computat*

X αx - Y αy]
  by (*auto simp*: *ha pair-qbs-probs-def*)
 also have ... ∈ *qbs-Mx* (*monadP-qbs* ($X \bigotimes_Q Y$))
  using *qbs-prob-MPx*[*OF ha*(*1*,*2*)] *qbs-prob-MPx*[*OF ha*(*4*,*5*)] *pair-qbs-probs.ab-g-in-Mx*[*of X αx - Y αy*]
  by(*auto intro*!: *bexI*[**where** *x*=*map-prod αx αy* ∘ *real-real.g*] *bexI*[**where** *x*=λ*r*. *distr* ($gx\ r \bigotimes_M gy\ r$) *real-borel real-real.f*]
   simp: *monadP-qbs-MPx-def in-MPx-def pair-qbs-probs-def*)
 finally show (λ(*x*, *y*). $x \bigotimes_{Qmes} y$) ∘ (λ*r*. (*βx r*, *βy r*)) ∈ *qbs-Mx* (*monadP-qbs* ($X \bigotimes_Q Y$)) **.**
qed

lemma(**in** *pair-qbs-probs*) *qbs-prob-pair-measure-nnintegral*:
 assumes $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_{Q \geq 0}$
 shows ($\int^+_Q$ *z*. *f z* ∂(*qbs-prob-space* (*X*,α,μ) $\bigotimes_{Qmes}$ *qbs-prob-space* (*Y*,β,ν)))
= ($\int^+$ *z*. (*f* ∘ *map-prod* α β) *z* ∂($\mu \bigotimes_M \nu$))
  (**is** *?lhs* = *?rhs*)
proof −
 have *?lhs* = ($\int^+$ *x*. ((*f* ∘ *map-prod* α β) ∘ *real-real.g*) *x* ∂*distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
  by(*simp add*: *qbs-prob-ennintegral-def*[*OF assms*] *qbs-prob-pair-measure-computation*)
 also have ... = ($\int^+$ *x*. ((*f* ∘ *map-prod* α β) ∘ *real-real.g*) (*real-real.f x*) ∂($\mu \bigotimes_M \nu$))
  using *assms* **by**(*intro nn-integral-distr*) *auto*
 also have ... = *?rhs* **by** *simp*
 finally show *?thesis* **.**
qed

lemma(**in** *pair-qbs-probs*) *qbs-prob-pair-measure-integral*:
 assumes $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
  shows ($\int_Q$ *z*. *f z* ∂(*qbs-prob-space* (*X*,α,μ) $\bigotimes_{Qmes}$ *qbs-prob-space* (*Y*,β,ν)))
= (∫ *z*. (*f* ∘ *map-prod* α β) *z* ∂($\mu \bigotimes_M \nu$))
  (**is** *?lhs* = *?rhs*)
proof −
 have *?lhs* = (∫ *x*. ((*f* ∘ *map-prod* α β) ∘ *real-real.g*) *x* ∂*distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*)
  by(*simp add*: *qbs-prob-integral-def*[*OF assms*] *qbs-prob-pair-measure-computation*)
 also have ... = (∫ *x*. ((*f* ∘ *map-prod* α β) ∘ *real-real.g*) (*real-real.f x*) ∂($\mu \bigotimes_M \nu$))
  using *assms* **by**(*intro integral-distr*) *auto*
 also have ... = *?rhs* **by** *simp*
 finally show *?thesis* **.**
qed

lemma *qbs-prob-pair-measure-eq-bind*:
 assumes *p* ∈ *monadP-qbs-Px X*
  and *q* ∈ *monadP-qbs-Px Y*
 shows $p \bigotimes_{Qmes} q$ = *p* ≫= (λ*x*. *q* ≫= (λ*y*. *qbs-return* ($X \bigotimes_Q Y$) (*x*,*y*)))
proof −

obtain α μ **where** *hp*:
p = *qbs-prob-space* (X, α, μ) *qbs-prob* X α μ
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
obtain β ν **where** *hq*:
q = *qbs-prob-space* (Y, β, ν) *qbs-prob* Y β ν
using *rep-monadP-qbs-Px*[*OF assms*(*2*)] **by** *auto*
interpret *pqp*: *pair-qbs-probs* X α μ Y β ν
by(*simp add*: *pair-qbs-probs-def hp hq*)
show *?thesis*
by(*simp add*: *hp*(*1*) *hq*(*1*) *pqp.qbs-prob-pair-measure-computation*(*1*) *pqp.qbs-bind-return-pq*(*1*))
qed

3.3.2 Fubini Theorem

lemma *qbs-prob-ennintegral-Fubini-fst*:
assumes $p \in$ *monadP-qbs-Px* X
$q \in$ *monadP-qbs-Px* Y
and $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_{Q\geq 0}$
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 −
note [*simp*] = *qbs-bind-morphism*[*OF qbs-morphism-const*[*of - monadP-qbs Y*,*simplified*,*OF assms*(*2*)] *curry-preserves-morphisms*[*OF qbs-return-morphism*[*of* $X \bigotimes_Q Y$]],*simplified curry-def*,*simplified*]
qbs-morphism-Pair1'[*OF - qbs-return-morphism*[*of* $X \bigotimes_Q Y$]]
assms(*1*)[*simplified monadP-qbs-Px-def*,*simplified*] *assms*(*2*)[*simplified monadP-qbs-Px-def*,*simplified*]
have *?rhs* = $(\int^+_Q z.\ f\ z\ \partial(p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ qbs\text{-}return\ (X \bigotimes_Q Y)\ (x,y)))))$
by(*simp add*: *qbs-prob-pair-measure-eq-bind*[*OF assms*(*1*,*2*)])
also have ... = $(\int^+_Q x.\ qbs\text{-}prob\text{-}ennintegral\ (q \gg\!= (\lambda y.\ qbs\text{-}return\ (X \bigotimes_Q Y)\ (x, y)))\ f\ \partial p)$
by(*auto intro*!: *qbs-prob-ennintegral-bind*[*OF assms*(*1*) - *assms*(*3*)])
also have ... = $(\int^+_Q x.\ \int^+_Q y.\ qbs\text{-}prob\text{-}ennintegral\ (qbs\text{-}return\ (X \bigotimes_Q Y)\ (x, y))\ f\ \partial q\ \partial p)$
by(*auto intro*!: *qbs-prob-ennintegral-cong qbs-prob-ennintegral-bind*[*OF assms*(*2*) - *assms*(*3*)])
also have ... = *?lhs*
using *assms*(*3*) **by**(*auto intro*!: *qbs-prob-ennintegral-cong qbs-prob-ennintegral-return*)
finally show *?thesis* **by** *simp*
qed

lemma *qbs-prob-ennintegral-Fubini-snd*:
assumes $p \in$ *monadP-qbs-Px* X
$q \in$ *monadP-qbs-Px* Y
and $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_{Q\geq 0}$
shows $(\int^+_Q y.\ \int^+_Q x.\ f\ (x,y)\ \partial p\ \partial q) = (\int^+_Q x.\ f\ x\ \partial(p \bigotimes_{Qmes} q))$
(**is** *?lhs* = *?rhs*)
proof −
note [*simp*] = *qbs-bind-morphism*[*OF qbs-morphism-const*[*of - monadP-qbs X*,*simplified*,*OF*

assms(*1*)] *curry-preserves-morphisms*[*OF qbs-morphism-pair-swap*[*OF qbs-return-morphism*[*of* $X \bigotimes_Q Y$]],*simplified curry-def*,*simplified*]]
qbs-morphism-Pair2'[*OF* - *qbs-return-morphism*[*of* $X \bigotimes_Q Y$]]
assms(*1*)[*simplified monadP-qbs-Px-def*,*simplified*] *assms*(*2*)[*simplified monadP-qbs-Px-def*,*simplified*]
have *?rhs* = ($\int^{+}_Q$ *z*. *f z* ∂(*q* ⤜ (λ*y*. *p* ⤜ (λ*x*. *qbs-return* ($X \bigotimes_Q Y$) (*x*,*y*)))))
by(*simp add*: *qbs-prob-pair-measure-eq-bind*[*OF assms*(*1*,*2*)] *qbs-bind-return-rotate*[*OF assms*(*1*,*2*)])
also have ... = ($\int^{+}_Q$ *y*. *qbs-prob-ennintegral* (*p* ⤜ (λ*x*. *qbs-return* ($X \bigotimes_Q Y$) (*x*, *y*))) *f* ∂*q*)
by(*auto intro*!: *qbs-prob-ennintegral-bind*[*OF assms*(*2*) - *assms*(*3*)])
also have ... = ($\int^{+}_Q$ *y*. $\int^{+}_Q$ *x*. *qbs-prob-ennintegral* (*qbs-return* ($X \bigotimes_Q Y$) (*x*, *y*)) *f* ∂*p* ∂*q*)
by(*auto intro*!: *qbs-prob-ennintegral-cong qbs-prob-ennintegral-bind*[*OF assms*(*1*) - *assms*(*3*)])
also have ... = *?lhs*
using *assms*(*3*) **by**(*auto intro*!: *qbs-prob-ennintegral-cong qbs-prob-ennintegral-return*)
finally show *?thesis* **by** *simp*
qed

lemma *qbs-prob-ennintegral-indep1*:
assumes *p* ∈ *monadP-qbs-Px X*
and *f* ∈ $X \to_Q \mathbb{R}_{Q\geq 0}$
shows ($\int^{+}_Q$ *z*. *f* (*fst z*) ∂(*p* $\bigotimes_{Qmes}$ *q*)) = ($\int^{+}_Q$ *x*. *f x* ∂*p*)
(**is** *?lhs* = -)
proof −
obtain *Y β ν* **where** *hq*:
q = *qbs-prob-space* (*Y*, *β*, *ν*) *qbs-prob Y β ν*
using *rep-qbs-prob-space*[*of q*] **by** *auto*
have *?lhs* = ($\int^{+}_Q$ *y*. $\int^{+}_Q$ *x*. *f x* ∂*p* ∂*q*)
using *qbs-prob-ennintegral-Fubini-snd*[*OF assms*(*1*) *qbs-prob.qbs-prob-space-in-Px*[*OF hq*(*2*)] *qbs-morphism-fst''*[*OF assms*(*2*)]]
by(*simp add*: *hq*(*1*))
thus *?thesis*
by(*simp add*: *qbs-prob-ennintegral-const*)
qed

lemma *qbs-prob-ennintegral-indep2*:
assumes *q* ∈ *monadP-qbs-Px Y*
and *f* ∈ $Y \to_Q \mathbb{R}_{Q\geq 0}$
shows ($\int^{+}_Q$ *z*. *f* (*snd z*) ∂(*p* $\bigotimes_{Qmes}$ *q*)) = ($\int^{+}_Q$ *y*. *f y* ∂*q*)
(**is** *?lhs* = -)
proof −
obtain *X α μ* **where** *hp*:
p = *qbs-prob-space* (*X*, *α*, *μ*) *qbs-prob X α μ*
using *rep-qbs-prob-space*[*of p*] **by** *auto*
have *?lhs* = ($\int^{+}_Q$ *x*. $\int^{+}_Q$ *y*. *f y* ∂*q* ∂*p*)
using *qbs-prob-ennintegral-Fubini-fst*[*OF qbs-prob.qbs-prob-space-in-Px*[*OF hp*(*2*)] *assms*(*1*) *qbs-morphism-snd''*[*OF assms*(*2*)]]

by(*simp add*: *hp*(*1*))
thus *?thesis*
by(*simp add*: *qbs-prob-ennintegral-const*)
qed

lemma *qbs-ennintegral-indep-mult*:
assumes $p \in$ *monadP-qbs-Px X*
$q \in$ *monadP-qbs-Px Y*
$f \in X \to_Q \mathbb{R}_{Q\geq 0}$
and $g \in Y \to_Q \mathbb{R}_{Q\geq 0}$
shows ($\int^{+}_Q$ *z*. *f* (*fst z*) $*$ *g* (*snd z*) ∂($p \bigotimes_{Qmes} q$)) = ($\int^{+}_Q$ *x*. *f x* ∂p) $*$ ($\int^{+}_Q$ *y*. *g y* ∂q)
(**is** *?lhs* = *?rhs*)
proof −
have *h*:(λz. *f* (*fst z*) $*$ *g* (*snd z*)) $\in X \bigotimes_Q Y \to_Q \mathbb{R}_{Q\geq 0}$
using *assms*(*4*,*3*)
by(*auto intro*!: *borel-measurable-subalgebra*[*OF l-product-sets*[*of X Y*]] *simp*: *space-pair-measure lr-adjunction-correspondence*)

have *?lhs* = ($\int^{+}_Q$ *x*. $\int^{+}_Q$ *y* .*f x* $*$ *g y* ∂q ∂p)
using *qbs-prob-ennintegral-Fubini-fst*[*OF assms*(*1*,*2*) *h*] **by** *simp*
also have ... = ($\int^{+}_Q$ *x*. *f x* $*$ $\int^{+}_Q$ *y* . *g y* ∂q ∂p)
using *qbs-prob-ennintegral-cmult*[*of q*,*OF* - *assms*(*4*)] *assms*(*2*)
by(*simp add*: *monadP-qbs-Px-def*)
also have ... = *?rhs*
using *qbs-prob-ennintegral-cmult*[*of p*,*OF* - *assms*(*3*)] *assms*(*1*)
by(*simp add*: *ab-semigroup-mult-class.mult.commute*[**where** *b*=*qbs-prob-ennintegral q g*] *monadP-qbs-Px-def*)
finally show *?thesis* **.**
qed

lemma(**in** *pair-qbs-probs*) *qbs-prob-pair-measure-integrable*:
assumes *qbs-integrable* (*qbs-prob-space* (X,α,μ) $\bigotimes_{Qmes}$ *qbs-prob-space* (Y,β,ν)) *f*
shows $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
integrable ($\mu \bigotimes_M \nu$) ($f \circ$ (*map-prod* α β))
proof −
show $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
using *qbs-integrable-morphism*[*OF qbs-prob-pair-measure-qbs assms*]
by *simp*
next
have *1*:*integrable* (*distr* ($\mu \bigotimes_M \nu$) *real-borel real-real.f*) ($f \circ$ (*map-prod* α β $\circ$ *real-real.g*))
using *assms*[*simplified qbs-prob-pair-measure-computation*] *qbs-integrable-def*[*of f*]
by *simp*
have *integrable* ($\mu \bigotimes_M \nu$) (λx. ($f \circ$ (*map-prod* α β $\circ$ *real-real.g*)) (*real-real.f x*))
by(*intro integrable-distr*[*OF* - *1*]) *simp*

thus *integrable* ($\mu \bigotimes_M \nu$) ($f \circ$ *map-prod* α β)
by(*simp add*: *comp-def*)
qed

lemma(**in** *pair-qbs-probs*) *qbs-prob-pair-measure-integrable'*:
assumes $f \in X \bigotimes_Q Y \rightarrow_Q \mathbb{R}_Q$
and *integrable* ($\mu \bigotimes_M \nu$) ($f \circ$ (*map-prod* α β))
shows *qbs-integrable* (*qbs-prob-space* (X,α,μ) $\bigotimes_{Qmes}$ *qbs-prob-space* (Y,β,ν)) f
proof −
have *integrable* (*distr* ($\mu \bigotimes_M \nu$) *real-borel* *real-real.f*) ($f \circ$ (*map-prod* α β $\circ$ *real-real.g*)) = *integrable* ($\mu \bigotimes_M \nu$) (λx. ($f \circ$ (*map-prod* α β $\circ$ *real-real.g*)) (*real-real.f* x))
by(*intro integrable-distr-eq*) (*use assms*(*1*) **in** *auto*)
thus *?thesis*
using *assms qbs-integrable-def*
by(*simp add*: *comp-def qbs-prob-pair-measure-computation*)
qed

lemma *qbs-integrable-pair-swap*:
assumes *qbs-integrable* ($p \bigotimes_{Qmes} q$) f
shows *qbs-integrable* ($q \bigotimes_{Qmes} p$) ($\lambda(x,y)$. f (y,x))
proof −
obtain X α μ **where** *hp*:
p = *qbs-prob-space* (X, α, μ) *qbs-prob* X α μ
using *rep-qbs-prob-space*[*of* p] **by** *auto*
obtain Y β ν **where** *hq*:
q = *qbs-prob-space* (Y, β, ν) *qbs-prob* Y β ν
using *rep-qbs-prob-space*[*of* q] **by** *auto*
interpret *pqp*: *pair-qbs-probs* X α μ Y β ν
by(*simp add*: *pair-qbs-probs-def hp hq*)
interpret *pqp2*: *pair-qbs-probs* Y β ν X α μ
by(*simp add*: *pair-qbs-probs-def hp hq*)

have $f \in X \bigotimes_Q Y \rightarrow_Q \mathbb{R}_Q$
integrable ($\mu \bigotimes_M \nu$) ($f \circ$ *map-prod* α β)
by(*auto simp*: *pqp.qbs-prob-pair-measure-integrable*[*OF assms*[*simplified hp*(*1*) *hq*(*1*)]])
from *qbs-morphism-pair-swap*[*OF this*(*1*)] *pqp.integrable-product-swap*[*OF this*(*2*)]
have ($\lambda(x,y)$. f (y,x)) $\in Y \bigotimes_Q X \rightarrow_Q \mathbb{R}_Q$
integrable ($\nu \bigotimes_M \mu$) (($\lambda(x,y)$. f (y,x)) $\circ$ *map-prod* β α)
by(*simp-all add*: *map-prod-def comp-def split-beta'*)
from *pqp2.qbs-prob-pair-measure-integrable'* [*OF this*]
show *?thesis* **by**(*simp add*: *hp*(*1*) *hq*(*1*))
qed

lemma *qbs-integrable-pair1*:
assumes $p \in$ *monadP-qbs-Px* X
$q \in$ *monadP-qbs-Px* Y

$f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
qbs-integrable p ($\lambda x.\ \int_Q y.\ |f\ (x,y)|\ \partial q$)
and $\bigwedge x.\ x \in$ *qbs-space X* $\Longrightarrow$ *qbs-integrable q* ($\lambda y.\ f\ (x,y)$)
shows *qbs-integrable* ($p \bigotimes_{Qmes} q$) *f*
proof −
obtain α μ **where** *hp*:
p = *qbs-prob-space* (X, α, μ) *qbs-prob X* α μ
using *rep-monadP-qbs-Px*[*OF assms*(*1*)] **by** *auto*
obtain β ν **where** *hq*:
q = *qbs-prob-space* (Y, β, ν) *qbs-prob Y* β ν
using *rep-monadP-qbs-Px*[*OF assms*(*2*)] **by** *auto*
interpret *pqp*: *pair-qbs-probs X* α μ *Y* β ν
by(*simp add*: *pair-qbs-probs-def hp hq*)

have *integrable* ($\mu \bigotimes_M \nu$) ($f \circ$ *map-prod* α β)
proof(*rule pqp.Fubini-integrable*)
show $f \circ$ *map-prod* α β $\in$ *borel-measurable* ($\mu \bigotimes_M \nu$)
using *assms*(*3*) **by** *auto*
next
have ($\lambda x.$ *LINT* $y|\nu$. *norm* (($f \circ$ *map-prod* α β) (x, y))) = ($\lambda x.\ \int_Q y.\ |f\ (x,y)|$ ∂q) $\circ$ α
apply *standard* **subgoal for** *x*
using *qbs-morphism-Pair1'*[*OF qbs-Mx-to-X*(*2*)[*OF pqp.qp1.in-Mx*,*of x*] *assms*(*3*)]
by(*auto intro*!: *pqp.qp2.qbs-prob-integral-def*[*symmetric*] *simp*: *hq*(*1*))
done
moreover have *integrable* μ ...
using *assms*(*4*) *pqp.qp1.qbs-integrable-def*
by (*simp add*: *hp*(*1*))
ultimately show *integrable* μ ($\lambda x.$ *LINT* $y|\nu$. *norm* (($f \circ$ *map-prod* α β) (x, y)))
by *simp*
next
have $\bigwedge x.$ *integrable* ν ($\lambda y.$ ($f \circ$ *map-prod* α β) (x, y))
proof−
fix *x*
have ($\lambda y.$ ($f \circ$ *map-prod* α β) (x, y)) = ($\lambda y.\ f\ (\alpha\ x,y)$) $\circ$ β
by *auto*
moreover have *qbs-integrable* (*qbs-prob-space* (Y, β, ν)) ($\lambda y.\ f\ (\alpha\ x,\ y)$)
by(*auto intro*!: *assms*(*5*)[*simplified hq*(*1*)] *simp*: *qbs-Mx-to-X*)
ultimately show *integrable* ν ($\lambda y.$ ($f \circ$ *map-prod* α β) (x, y))
by(*simp add*: *pqp.qp2.qbs-integrable-def*)
qed
thus *AE x in* μ. *integrable* ν ($\lambda y.$ ($f \circ$ *map-prod* α β) (x, y))
by *simp*
qed
thus *?thesis*
using *pqp.qbs-prob-pair-measure-integrable'*[*OF assms*(*3*)]
by(*simp add*: *hp*(*1*) *hq*(*1*))

qed

lemma *qbs-integrable-pair2*:
assumes $p \in$ *monadP-qbs-Px X*
$q \in$ *monadP-qbs-Px Y*
$f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
qbs-integrable q $(\lambda y. \int_Q x. |f\ (x,y)|\ \partial p)$
and $\bigwedge y.\ y \in$ *qbs-space Y* $\Longrightarrow$ *qbs-integrable p* $(\lambda x.\ f\ (x,y))$
shows *qbs-integrable* $(p \bigotimes_{Qmes} q)\ f$
using *qbs-integrable-pair-swap*[*OF qbs-integrable-pair1* [*OF assms*(*2*,*1*) *qbs-morphism-pair-swap*[*OF assms*(*3*)],*simplified*,*OF assms*(*4*,*5*)]]
by *simp*

lemma *qbs-integrable-fst*:
assumes *qbs-integrable* $(p \bigotimes_{Qmes} q)\ f$
shows *qbs-integrable p* $(\lambda x. \int_Q y.\ f\ (x,y)\ \partial q)$
proof −
obtain $X\ \alpha\ \mu$ **where** *hp*:
$p =$ *qbs-prob-space* (X, α, μ) *qbs-prob* $X\ \alpha\ \mu$
using *rep-qbs-prob-space*[*of p*] **by** *auto*
obtain $Y\ \beta\ \nu$ **where** *hq*:
$q =$ *qbs-prob-space* (Y, β, ν) *qbs-prob* $Y\ \beta\ \nu$
using *rep-qbs-prob-space*[*of q*] **by** *auto*
interpret *pqp*: *pair-qbs-probs* $X\ \alpha\ \mu\ Y\ \beta\ \nu$
by(*simp add*: *hp hq pair-qbs-probs-def*)
have *h0*: $p \in$ *monadP-qbs-Px X* $q \in$ *monadP-qbs-Px Y* $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
using *qbs-integrable-morphism*[*OF - assms*,*simplified qbs-prob-pair-measure-qbs*]
by(*simp-all add*: *monadP-qbs-Px-def hp*(*1*) *hq*(*1*))

show *qbs-integrable p* $(\lambda x. \int_Q y.\ f\ (x, y)\ \partial q)$
proof(*auto simp add*: *pqp.qp1.qbs-integrable-def hp*(*1*))
show $(\lambda x. \int_Q y.\ f\ (x, y)\ \partial q) \in$ *borel-measurable* (*qbs-to-measure X*)
using *qbs-morphism-integral-fst*[*OF h0*(*2*,*3*)] **by** *auto*
next
have *integrable* μ $(\lambda x.$ *LINT* $y|\nu.\ (f \circ$ *map-prod* $\alpha\ \beta)\ (x, y))$
by(*intro pqp.integrable-fst′*) (*rule pqp.qbs-prob-pair-measure-integrable*(*2*)[*OF assms*[*simplified hp*(*1*) *hq*(*1*)]]])
moreover have $\bigwedge x.\ ((\lambda x. \int_Q y.\ f\ (x, y)\ \partial q) \circ \alpha)\ x =$ *LINT* $y|\nu.\ (f \circ$ *map-prod* $\alpha\ \beta)\ (x, y)$
by(*auto intro*!: *pqp.qp2.qbs-prob-integral-def qbs-morphism-Pair1* ′[*OF qbs-Mx-to-X*(*2*)[*OF pqp.qp1.in-Mx*] *h0*(*3*)] *simp*: *hq*)
ultimately show *integrable* μ $((\lambda x. \int_Q y.\ f\ (x, y)\ \partial q) \circ \alpha)$
using *Bochner-Integration.integrable-cong*[*of* $\mu\ \mu\ (\lambda x. \int_Q y.\ f\ (x, y)\ \partial q) \circ \alpha$ $(\lambda x.$ *LINT* $y|\nu.\ (f \circ$ *map-prod* $\alpha\ \beta)\ (x, y))$]
by *simp*
qed
qed

lemma *qbs-integrable-snd*:

assumes *qbs-integrable* ($p \bigotimes_{Qmes} q$) *f*
shows *qbs-integrable q* ($\lambda y. \int_Q x.\ f\ (x,y)\ \partial p$)
using *qbs-integrable-fst*[*OF qbs-integrable-pair-swap*[*OF assms*]]
by *simp*

lemma *qbs-integrable-indep-mult*:
assumes *qbs-integrable p f*
and *qbs-integrable q g*
shows *qbs-integrable* ($p \bigotimes_{Qmes} q$) ($\lambda x.\ f\ (fst\ x) * g\ (snd\ x)$)
proof −
obtain $X\ \alpha\ \mu$ **where** *hp*:
$p = $ *qbs-prob-space* (X, α, μ) *qbs-prob* $X\ \alpha\ \mu$
using *rep-qbs-prob-space*[*of p*] **by** *auto*
obtain $Y\ \beta\ \nu$ **where** *hq*:
$q = $ *qbs-prob-space* (Y, β, ν) *qbs-prob* $Y\ \beta\ \nu$
using *rep-qbs-prob-space*[*of q*] **by** *auto*
interpret *pqp*: *pair-qbs-probs* $X\ \alpha\ \mu\ Y\ \beta\ \nu$
by(*simp add*: *hp hq pair-qbs-probs-def*)
have *h0*: $p \in$ *monadP-qbs-Px* X $q \in$ *monadP-qbs-Px* Y $f \in X \to_Q \mathbb{R}_Q$ $g \in Y \to_Q \mathbb{R}_Q$
using *qbs-integrable-morphism*[*OF* - *assms*(*1*)] *qbs-integrable-morphism*[*OF* - *assms*(*2*)]
by(*simp-all add*: *monadP-qbs-Px-def hp*(*1*) *hq*(*1*))

show *?thesis*
proof(*rule qbs-integrable-pair1*[*OF h0*(*1*,*2*)],*simp-all add*: *assms*(*2*))
show ($\lambda z.\ f\ (fst\ z) * g\ (snd\ z)) \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
using *h0*(*3*,*4*) **by**(*auto intro*!: *borel-measurable-subalgebra*[*OF l-product-sets*[*of X Y*]] *simp*: *space-pair-measure lr-adjunction-correspondence*)
next
show *qbs-integrable p* ($\lambda x. \int_Q y.\ |f\ x * g\ y|\ \partial q$)
by(*auto intro*!: *qbs-integrable-mult*[*OF qbs-integrable-abs*[*OF assms*(*1*)]]
simp only: *idom-abs-sgn-class.abs-mult qbs-prob-integral-cmult ab-semigroup-mult-class.mult.commute*[**w**
$b = \int_Q y.\ |g\ y|\ \partial q$])
qed
qed

lemma *qbs-integrable-indep1*:
assumes *qbs-integrable p f*
shows *qbs-integrable* ($p \bigotimes_{Qmes} q$) ($\lambda x.\ f\ (fst\ x)$)
using *qbs-integrable-indep-mult*[*OF assms qbs-integrable-const*[*of q 1*]]
by *simp*

lemma *qbs-integrable-indep2*:
assumes *qbs-integrable q g*
shows *qbs-integrable* ($p \bigotimes_{Qmes} q$) ($\lambda x.\ g\ (snd\ x)$)
using *qbs-integrable-pair-swap*[*OF qbs-integrable-indep1*[*OF assms*],*of p*]
by(*simp add*: *split-beta'*)

lemma *qbs-prob-integral-Fubini-fst*:
  assumes *qbs-integrable* $(p \bigotimes_{Qmes} q)$ *f*
    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 −
  obtain $X\ \alpha\ \mu$ **where** *hp*:
    p = *qbs-prob-space* (X, α, μ) *qbs-prob* $X\ \alpha\ \mu$
    using *rep-qbs-prob-space*[*of p*] **by** *auto*
  obtain $Y\ \beta\ \nu$ **where** *hq*:
   q = *qbs-prob-space* (Y, β, ν) *qbs-prob* $Y\ \beta\ \nu$
    using *rep-qbs-prob-space*[*of q*] **by** *auto*
  interpret *pqp*: *pair-qbs-probs* $X\ \alpha\ \mu\ Y\ \beta\ \nu$
    by(*simp add*: *hp hq pair-qbs-probs-def*)
  have *h0*: $p \in$ *monadP-qbs-Px* X $q \in$ *monadP-qbs-Px* Y $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
   using *qbs-integrable-morphism*[*OF - assms,simplified qbs-prob-pair-measure-qbs*]
    by(*simp-all add*: *monadP-qbs-Px-def hp*(*1*) *hq*(*1*))

  have *?lhs* = $(\int x. \int_Q y. f\ (\alpha\ x,\ y)\ \partial q\ \partial\mu)$
    using *qbs-morphism-integral-fst*[*OF h0*(*2*) *h0*(*3*)]
    by(*auto intro*!: *pqp.qp1.qbs-prob-integral-def simp*: *hp*(*1*))
  also have ... = $(\int x. \int y. f\ (\alpha\ x,\ \beta\ y)\ \partial\nu\ \partial\mu)$
    using *qbs-morphism-Pair1'*[*OF qbs-Mx-to-X*(*2*)[*OF pqp.qp1.in-Mx*] *h0*(*3*)]
    by(*auto intro*!: *Bochner-Integration.integral-cong pqp.qp2.qbs-prob-integral-def*
          simp: *hq*(*1*))
  also have ... = $(\int z. (f \circ$ *map-prod* $\alpha\ \beta)\ z\ \partial(\mu \bigotimes_M \nu))$
   using *pqp.integral-fst'*[*OF pqp.qbs-prob-pair-measure-integrable*(*2*)[*OF assms*[*simplified hp*(*1*) *hq*(*1*)]]]
    by(*simp add*: *map-prod-def comp-def*)
  also have ... = *?rhs*
    by(*simp add*: *pqp.qbs-prob-pair-measure-integral*[*OF h0*(*3*)] *hp*(*1*) *hq*(*1*))
  finally show *?thesis* **.**
qed

lemma *qbs-prob-integral-Fubini-snd*:
  assumes *qbs-integrable* $(p \bigotimes_{Qmes} q)$ *f*
    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 −
  obtain $X\ \alpha\ \mu$ **where** *hp*:
    p = *qbs-prob-space* (X, α, μ) *qbs-prob* $X\ \alpha\ \mu$
    using *rep-qbs-prob-space*[*of p*] **by** *auto*
  obtain $Y\ \beta\ \nu$ **where** *hq*:
   q = *qbs-prob-space* (Y, β, ν) *qbs-prob* $Y\ \beta\ \nu$
    using *rep-qbs-prob-space*[*of q*] **by** *auto*
  interpret *pqp*: *pair-qbs-probs* $X\ \alpha\ \mu\ Y\ \beta\ \nu$
    by(*simp add*: *hp hq pair-qbs-probs-def*)
  have *h0*: $p \in$ *monadP-qbs-Px* X $q \in$ *monadP-qbs-Px* Y $f \in X \bigotimes_Q Y \to_Q \mathbb{R}_Q$
   using *qbs-integrable-morphism*[*OF - assms,simplified qbs-prob-pair-measure-qbs*]

by(*simp-all add*: *monadP-qbs-Px-def hp*(*1*) *hq*(*1*))

have *?lhs* = ($\int$ *y*. $\int_Q$ *x*. *f* (*x*,β *y*) ∂p $\partial\nu$)
using *qbs-morphism-integral-snd*[*OF h0*(*1*) *h0*(*3*)]
by(*auto intro*!: *pqp.qp2.qbs-prob-integral-def simp*: *hq*(*1*))
also have ... = ($\int$ *y*. $\int$ *x*. *f* (α *x*, β *y*) $\partial\mu$ $\partial\nu$)
using *qbs-morphism-Pair2'*[*OF qbs-Mx-to-X*(*2*)[*OF pqp.qp2.in-Mx*] *h0*(*3*)]
by(*auto intro*!: *Bochner-Integration.integral-cong pqp.qp1.qbs-prob-integral-def simp*: *hp*(*1*))
also have ... = ($\int$ *z*. (*f* $\circ$ *map-prod* α β) *z* $\partial(\mu \bigotimes_M \nu)$)
using *pqp.integral-snd*[*of curry* (*f* $\circ$ *map-prod* α β)] *pqp.qbs-prob-pair-measure-integrable*(*2*)[*OF assms*[*simplified hp*(*1*) *hq*(*1*)]]
by(*simp add*: *map-prod-def comp-def split-beta'*)
also have ... = *?rhs*
by(*simp add*: *pqp.qbs-prob-pair-measure-integral*[*OF h0*(*3*)] *hp*(*1*) *hq*(*1*))
finally show *?thesis* **.**
qed

lemma *qbs-prob-integral-indep1*:
assumes *qbs-integrable p f*
shows ($\int_Q$ *z*. *f* (*fst z*) $\partial(p \bigotimes_{Qmes} q)$) = ($\int_Q$ *x*. *f x* ∂p)
using *qbs-prob-integral-Fubini-snd*[*OF qbs-integrable-indep1*[*OF assms*],*of q*]
by(*simp add*: *qbs-prob-integral-const*)

lemma *qbs-prob-integral-indep2*:
assumes *qbs-integrable q g*
shows ($\int_Q$ *z*. *g* (*snd z*) $\partial(p \bigotimes_{Qmes} q)$) = ($\int_Q$ *y*. *g y* ∂q)
using *qbs-prob-integral-Fubini-fst*[*OF qbs-integrable-indep2*[*OF assms*],*of p*]
by(*simp add*: *qbs-prob-integral-const*)

lemma *qbs-prob-integral-indep-mult*:
assumes *qbs-integrable p f*
and *qbs-integrable q g*
shows ($\int_Q$ *z*. *f* (*fst z*) $*$ *g* (*snd z*) $\partial(p \bigotimes_{Qmes} q)$) = ($\int_Q$ *x*. *f x* ∂p) $*$ ($\int_Q$ *y*. *g y* ∂q)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\int_Q$ *x*. $\int_Q$ *y*. *f x* $*$ *g y* ∂q ∂p)
using *qbs-prob-integral-Fubini-fst*[*OF qbs-integrable-indep-mult*[*OF assms*]]
by *simp*
also have ... = ($\int_Q$ *x*. *f x* $*$ ($\int_Q$ *y*. *g y* ∂q) ∂p)
by(*simp add*: *qbs-prob-integral-cmult*)
also have ... = *?rhs*
by(*simp add*: *qbs-prob-integral-cmult ab-semigroup-mult-class.mult.commute*[**where** *b*=$\int_Q$ *y*. *g y* ∂q])
finally show *?thesis* **.**
qed

lemma *qbs-prob-var-indep-plus*:

assumes *qbs-integrable* $(p \bigotimes_{Qmes} q)$ *f*
qbs-integrable $(p \bigotimes_{Qmes} q)$ $(\lambda z.\ (f\ z)^2)$
qbs-integrable $(p \bigotimes_{Qmes} q)$ *g*
qbs-integrable $(p \bigotimes_{Qmes} q)$ $(\lambda z.\ (g\ z)^2)$
qbs-integrable $(p \bigotimes_{Qmes} q)$ $(\lambda z.\ (f\ z) * (g\ z))$
and $(\int_Q z.\ f\ z * g\ z\ \partial(p \bigotimes_{Qmes} q)) = (\int_Q z.\ f\ z\ \partial(p \bigotimes_{Qmes} q)) * (\int_Q z.\ g\ z\ \partial(p \bigotimes_{Qmes} q))$
shows *qbs-prob-var* $(p \bigotimes_{Qmes} q)$ $(\lambda z.\ f\ z + g\ z)$ = *qbs-prob-var* $(p \bigotimes_{Qmes} q)$ *f* + *qbs-prob-var* $(p \bigotimes_{Qmes} q)$ *g*
unfolding *qbs-prob-var-def*
proof –
show $(\int_Q z.\ (f\ z + g\ z - \int_Q w.\ f\ w + g\ w\ \partial(p \bigotimes_{Qmes} q))^2\ \partial(p \bigotimes_{Qmes} q)) = (\int_Q z.\ (f\ z -$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)\ f)^2\ \partial(p \bigotimes_{Qmes} q)) + (\int_Q z.\ (g\ z -$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)\ g)^2\ \partial(p \bigotimes_{Qmes} q))$
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* $= (\int_Q z.\ ((f\ z - (\int_Q w.\ f\ w\ \partial(p \bigotimes_{Qmes} q))) + (g\ z - (\int_Q w.\ g\ w\ \partial(p \bigotimes_{Qmes} q))))^2\ \partial(p \bigotimes_{Qmes} q))$
by(*simp add*: *qbs-prob-integral-add*[*OF assms*(*1*,*3*)] *add-diff-add*)
also have ... $= (\int_Q z.\ (f\ z - (\int_Q w.\ f\ w\ \partial(p \bigotimes_{Qmes} q)))^2 + (g\ z - (\int_Q w.\ g\ w\ \partial(p \bigotimes_{Qmes} q)))^2 + (2 * f\ z * g\ z - 2 * (\int_Q w.\ f\ w\ \partial(p \bigotimes_{Qmes} q)) * g\ z - (2 * f\ z * (\int_Q w.\ g\ w\ \partial(p \bigotimes_{Qmes} q)) - 2 * (\int_Q w.\ f\ w\ \partial(p \bigotimes_{Qmes} q)) * (\int_Q w.\ g\ w\ \partial(p \bigotimes_{Qmes} q))))\ \partial(p \bigotimes_{Qmes} q))$
by(*simp add*: *comm-semiring-1-class.power2-sum comm-semiring-1-cancel-class.left-diff-distrib′ ring-class.right-diff-distrib*)
also have ... = *?rhs*
using *qbs-prob-integral-add*[*OF qbs-integrable-add*[*OF qbs-integrable-sq*[*OF assms*(*1*,*2*)] *qbs-integrable-sq*[*OF assms*(*3*,*4*)]] *qbs-integrable-diff*[*OF qbs-integrable-diff*[*OF qbs-integrable-mult*[*OF assms*(*5*),*of 2*,*simplified comm-semiring-1-class.semiring-normalization-rules*(*18*)] *qbs-integrable-mult*[*OF assms*(*3*),*of* $2 *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *f*]] *qbs-integrable-diff*[*OF qbs-integrable-mult*[*OF assms*(*1*),*of* $2 *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*,*simplified ab-semigroup-mult-class.mult-ac*(*1*)[**where** *b*=*qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*] *ab-semigroup-mult-class.mult.commute*[**where** *a*=*qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*] *comm-semiring-1-class.semiring-normalization-rules*(*18*)[*of - - qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*]] *qbs-integrable-const*[*of -* $2 *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ $f *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*]]]]
qbs-prob-integral-add[*OF qbs-integrable-sq*[*OF assms*(*1*,*2*)] *qbs-integrable-sq*[*OF assms*(*3*,*4*)]]
qbs-prob-integral-diff[*OF qbs-integrable-diff*[*OF qbs-integrable-mult*[*OF assms*(*5*),*of 2*,*simplified comm-semiring-1-class.semiring-normalization-rules*(*18*)] *qbs-integrable-mult*[*OF assms*(*3*),*of* $2 *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *f*]] *qbs-integrable-diff*[*OF qbs-integrable-mult*[*OF assms*(*1*),*of* $2 *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*,*simplified ab-semigroup-mult-class.mult-ac*(*1*)[**where** *b*=*qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*] *ab-semigroup-mult-class.mult.commute*[**where** *a*=*qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*] *comm-semiring-1-class.semiring-normalization-rules*(*18*)[*of - - qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*]] *qbs-integrable-const*[*of -* $2 *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ $f *$ *qbs-prob-integral* $(p \bigotimes_{Qmes} q)$ *g*]]]
qbs-prob-integral-diff[*OF qbs-integrable-mult*[*OF assms*(*5*),*of 2*,*simplified comm-semiring-1-class.semiring-normalization-rules*(*18*)] *qbs-integrable-mult*[*OF assms*(*3*),*of*

2 ∗ qbs-prob-integral (p $\bigotimes_{Qmes}$ *q) f*]]
qbs-prob-integral-diff[*OF qbs-integrable-mult*[*OF assms(1),of 2 ∗ qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g,simplified ab-semigroup-mult-class.mult-ac(1)*[**where** *b=qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g*] *ab-semigroup-mult-class.mult.commute*[**where** *a=qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g*] *comm-semiring-1-class.semiring-normalization-rules(18)*[*of - - qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g*]] *qbs-integrable-const*[*of - 2 ∗ qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) f ∗ qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g*]]
qbs-prob-integral-cmult[*of p* $\bigotimes_{Qmes}$ *q 2 λz. f z ∗ g z,simplified assms(6) comm-semiring-1-class.semiring-normalization-rules(18)*]
qbs-prob-integral-cmult[*of p* $\bigotimes_{Qmes}$ *q 2 ∗ (*$\int_Q$ *w. f w ∂(p* $\bigotimes_{Qmes}$ *q)) g*]
qbs-prob-integral-cmult[*of p* $\bigotimes_{Qmes}$ *q 2 ∗ (*$\int_Q$ *w. g w ∂(p* $\bigotimes_{Qmes}$ *q)) f,simplified semigroup-mult-class.mult.assoc*[*of 2* $\int_Q$ *w. g w ∂(p* $\bigotimes_{Qmes}$ *q)*] *ab-semigroup-mult-class.mult.commute*[**where** *a=qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g*] *comm-semiring-1-class.semiring-normalization-rules(18)*[*of 2 -* $\int_Q$ *w. g w ∂(p* $\bigotimes_{Qmes}$ *q)*]]
qbs-prob-integral-const[*of p* $\bigotimes_{Qmes}$ *q 2 ∗ qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) f ∗ qbs-prob-integral (p* $\bigotimes_{Qmes}$ *q) g*]
by *simp*
finally show *?thesis* **.**
qed
qed

lemma *qbs-prob-var-indep-plus′*:
assumes *qbs-integrable p f*
qbs-integrable p (λx. (f x)^2*)*
qbs-integrable q g
and *qbs-integrable q (λx. (g x)*^{2}*)*
shows *qbs-prob-var (p* $\bigotimes_{Qmes}$ *q) (λz. f (fst z) + g (snd z)) = qbs-prob-var p f + qbs-prob-var q g*
using *qbs-prob-var-indep-plus*[*OF qbs-integrable-indep1*[*OF assms(1)*] *qbs-integrable-indep1*[*OF assms(2)*] *qbs-integrable-indep2*[*OF assms(3)*] *qbs-integrable-indep2*[*OF assms(4)*] *qbs-integrable-indep-mult*[*OF assms(1) assms(3)*] *qbs-prob-integral-indep-mult*[*OF assms(1) assms(3),simplified qbs-prob-integral-indep1*[*OF assms(1),of q,symmetric*] *qbs-prob-integral-indep2*[*OF assms(3),of p,symmetric*]]]
qbs-prob-integral-indep1[*OF qbs-integrable-sq*[*OF assms(1,2)*]*,of q* $\int_Q$ *z. f (fst z) ∂(p* $\bigotimes_{Qmes}$ *q)*] *qbs-prob-integral-indep2*[*OF qbs-integrable-sq*[*OF assms(3,4)*]*,of p* $\int_Q$ *z. g (snd z) ∂(p* $\bigotimes_{Qmes}$ *q)*]
by(*simp add: qbs-prob-var-def qbs-prob-integral-indep1*[*OF assms(1)*] *qbs-prob-integral-indep2*[*OF assms(3)*])

end

3.4 Measure as QBS Measure

theory *Measure-as-QuasiBorel-Measure*
imports *Pair-QuasiBorel-Measure*

begin

```
lemma distr-id':
  assumes sets N = sets M
          f ∈ N →M N
      and ⋀x. x ∈ space N ⟹ f x = x
    shows distr N M f = N
proof(rule measure-eqI)
  fix A
  assume 0:A ∈ sets (distr N M f)
  then have 1:A ⊆ space N
    by (auto simp: assms(1) sets.sets-into-space)

  have 2:A ∈ sets M
    using 0 by simp
  have 3:f ∈ N →M M
    using assms(2) by(simp add: measurable-cong-sets[OF - assms(1)])
  have f −‘ A ∩ space N = A
  proof −
    have f −‘ A = A ∪ {x. x ∉ space N ∧ f x ∈ A}
    proof(standard;standard)
      fix x
      assume h:x ∈ f −‘ A
      consider x ∈ A | x ∉ A
        by auto
      thus x ∈ A ∪ {x. x ∉ space N ∧ f x ∈ A}
      proof cases
        case 1
        then show ?thesis
          by simp
      next
        case 2
        have x ∉ space N
        proof(rule ccontr)
          assume ¬ x ∉ space N
          then have x ∈ space N
            by simp
          hence f x = x
            by(simp add: assms(3))
          hence f x ∉ A
            by(simp add: 2)
          thus False
            using h by simp
        qed
        thus ?thesis
          using h by simp
      qed
    next
      fix x
      show x ∈ A ∪ {x. x ∉ space N ∧ f x ∈ A} ⟹ x ∈ f −‘ A
        using 1 assms by auto
```

qed
thus *?thesis*
using *1* **by** *blast*
qed
thus *emeasure* (*distr N M f*) *A* = *emeasure N A*
by(*simp add*: *emeasure-distr*[*OF 3 2*])
qed (*simp add*: *assms*(*1*))

Every probability measure on a standard Borel space can be represented as a measure on a quasi-Borel space [1], Proposition 23.

locale *standard-borel-prob-space* = *standard-borel P* + *p*:*prob-space P*
for *P* :: *'a measure*
begin

sublocale *qbs-prob measure-to-qbs P g distr P real-borel f*
by(*auto intro*!: *qbs-probI p.prob-space-distr*)

lift-definition *as-qbs-measure* :: *'a qbs-prob-space* **is**
(*measure-to-qbs P*, *g*, *distr P real-borel f*)
by *simp*

lemma *as-qbs-measure-retract*:
assumes [*measurable*]:$a \in P \to_M$ *real-borel*
and [*measurable*]:$b \in$ *real-borel* $\to_M P$
and [*simp*]:$\bigwedge x.\ x \in space\ P \Longrightarrow (b \circ a)\ x = x$
shows *qbs-prob* (*measure-to-qbs P*) *b* (*distr P real-borel a*)
as-qbs-measure = *qbs-prob-space* (*measure-to-qbs P*, *b*, *distr P real-borel a*)
proof −
interpret *pqp*: *pair-qbs-prob measure-to-qbs P g distr P real-borel f measure-to-qbs P b distr P real-borel a*
by(*auto intro*!: *qbs-probI p.prob-space-distr simp*: *pair-qbs-prob-def*)
show *qbs-prob* (*measure-to-qbs P*) *b* (*distr P real-borel a*)
as-qbs-measure = *qbs-prob-space* (*measure-to-qbs P*, *b*, *distr P real-borel a*)
by(*auto intro*!: *pqp.qbs-prob-space-eq*
simp: *distr-distr distr-id'*[*OF standard-borel-lr-sets-ident*[*symmetric*]]
distr-id'[*OF standard-borel-lr-sets-ident*[*symmetric*] - *assms*(*3*)] *pqp.qp2.qbs-prob-axioms*
as-qbs-measure.abs-eq)
qed

lemma *measure-as-qbs-measure-qbs*:
qbs-prob-space-qbs as-qbs-measure = *measure-to-qbs P*
by *transfer auto*

lemma *measure-as-qbs-measure-image*:
as-qbs-measure ∈ *monadP-qbs-Px* (*measure-to-qbs P*)
by(*auto simp*: *measure-as-qbs-measure-qbs monadP-qbs-Px-def*)

lemma *as-qbs-measure-as-measure*[*simp*]:
distr (*distr P real-borel f*) (*qbs-to-measure* (*measure-to-qbs P*)) *g* = *P*

by(*auto intro*!: *distr-id'*[*OF standard-borel-lr-sets-ident*[*symmetric*]] *simp* : *qbs-prob-t-measure-def distr-distr*)

lemma *measure-as-qbs-measure-recover*:
qbs-prob-measure as-qbs-measure = *P*
by *transfer* (*simp add*: *qbs-prob-t-measure-def*)

end

lemma(**in** *standard-borel*) *qbs-prob-measure-recover*:
assumes $q \in$ *monadP-qbs-Px* (*measure-to-qbs M*)
shows *standard-borel-prob-space.as-qbs-measure* (*qbs-prob-measure q*) = *q*
proof −
obtain α μ **where** *hq*:
q = *qbs-prob-space* (*measure-to-qbs M*, α, μ) *qbs-prob* (*measure-to-qbs M*) α μ
using *rep-monadP-qbs-Px*[*OF assms*] **by** *auto*
then interpret *qp*: *qbs-prob measure-to-qbs M* α μ **by** *simp*
interpret *sp*: *standard-borel-prob-space distr* μ (*qbs-to-measure* (*measure-to-qbs M*)) α
using *qp.in-Mx*
by(*auto intro*!: *prob-space.prob-space-distr*
simp: *standard-borel-prob-space-def standard-borel-sets*[*OF sets-distr*[*of* μ *qbs-to-measure* (*measure-to-qbs M*) α,*simplified standard-borel-lr-sets-ident*,*symmetric*]])
interpret *st*: *standard-borel distr* μ *M* α
by(*auto intro*!: *standard-borel-sets*)
have [*measurable*]:*st.g* $\in$ *real-borel* $\to_M$ *M*
using *measurable-distr-eq2 st.g-meas* **by** *blast*
show *?thesis*
by(*auto intro*!: *pair-qbs-prob.qbs-prob-space-eq*
simp add: *hq*(*1*) *sp.as-qbs-measure.abs-eq pair-qbs-prob-def qp.qbs-prob-axioms sp.qbs-prob-axioms*)
(*simp-all add*: *measure-to-qbs-cong-sets*[*OF sets-distr*[*of* μ *qbs-to-measure* (*measure-to-qbs M*) α,*simplified standard-borel-lr-sets-ident*]])
qed

lemma(**in** *standard-borel-prob-space*) *ennintegral-as-qbs-ennintegral*:
assumes $k \in$ *borel-measurable P*
shows ($\int^{+}_{Q}$ *x*. *k x* ∂*as-qbs-measure*) = ($\int^{+}$ *x*. *k x* ∂*P*)
proof −
have *1*:$k \in$ *measure-to-qbs P* $\to_Q$ $\mathbb{R}_{Q\geq 0}$
using *assms* **by** *auto*
thus *?thesis*
by(*simp add*: *as-qbs-measure.abs-eq qbs-prob-ennintegral-def2*[*OF 1*])
qed

lemma(**in** *standard-borel-prob-space*) *integral-as-qbs-integral*:
($\int_{Q}$ *x*. *k x* ∂*as-qbs-measure*) = ($\int$ *x*. *k x* ∂*P*)
by(*simp add*: *as-qbs-measure.abs-eq qbs-prob-integral-def2*)

lemma(**in** *standard-borel*) *measure-with-args-morphism*:
assumes [*measurable*]:$\mu \in X \rightarrow_M$ *prob-algebra M*
shows *standard-borel-prob-space.as-qbs-measure* $\circ$ μ $\in$ *measure-to-qbs X* $\rightarrow_Q$ *monadP-qbs* (*measure-to-qbs M*)
proof(*auto intro*!: *qbs-morphismI*)
fix α
assume *h*[*measurable*]:$\alpha \in$ *real-borel* $\rightarrow_M$ *X*
have $\forall$ *r*. (*standard-borel-prob-space.as-qbs-measure* $\circ$ μ $\circ$ α) *r* = *qbs-prob-space* (*measure-to-qbs M*, *g*, ((λ*l*. *distr* (μ *l*) *real-borel f*) $\circ$ α) *r*)
proof *auto*
fix *r*
interpret *sp*: *standard-borel-prob-space* μ (α *r*)
using *measurable-space*[*OF assms measurable-space*[*OF h*]]
by(*simp add*: *standard-borel-prob-space-def space-prob-algebra*)
have *1*[*measurable-cong*]: *sets* (μ (α *r*)) = *sets M*
using *measurable-space*[*OF assms measurable-space*[*OF h*]] **by**(*simp add*: *space-prob-algebra*)
have *2*:*f* $\in$ μ (α *r*) $\rightarrow_M$ *real-borel g* $\in$ *real-borel* $\rightarrow_M$ μ (α *r*) $\bigwedge$*x*. *x* $\in$ *space* (μ (α *r*)) $\Longrightarrow$ (*g* $\circ$ *f*) *x* = *x*
using *measurable-space*[*OF assms measurable-space*[*OF h*]]
by(*simp-all add*: *standard-borel-prob-space-def sets-eq-imp-space-eq*[*OF 1*])
show *standard-borel-prob-space.as-qbs-measure* (μ (α *r*)) = *qbs-prob-space* (*measure-to-qbs M*, *g*, *distr* (μ (α *r*)) *real-borel f*)
by(*simp add*: *sp.as-qbs-measure-retract*[*OF 2*] *measure-to-qbs-cong-sets*[*OF subprob-measurableD*(*2*)[*OF measurable-prob-algebraD*[*OF assms*] *measurable-space*[*OF h*]]])
qed
thus *standard-borel-prob-space.as-qbs-measure* $\circ$ μ $\circ$ α $\in$ *monadP-qbs-MPx* (*measure-to-qbs M*)
by(*auto intro*!: *bexI*[**where** *x*=*g*] *bexI*[**where** *x*=(λ*l*. *distr* (μ *l*) *real-borel f*) $\circ$ α] *simp*: *monadP-qbs-MPx-def in-MPx-def*)
qed

lemma(**in** *standard-borel*) *measure-with-args-recover*:
assumes $\mu \in$ *space X* $\rightarrow$ *space* (*prob-algebra M*)
and *x* $\in$ *space X*
shows *qbs-prob-measure* (*standard-borel-prob-space.as-qbs-measure* (μ *x*)) = μ *x*
using *standard-borel-sets*[*of* μ *x*] *funcset-mem*[*OF assms*]
by(*simp add*: *standard-borel-prob-space-def space-prob-algebra standard-borel-prob-space.measure-as-qbs-mea*

3.5 Example of Probability Measures

Probability measures on $\mathbb{R}$ can be represented as probability measures on the quasi-Borel space $\mathbb{R}$.

3.5.1 Normal Distribution

definition *normal-distribution* :: *real* × *real* ⇒ *real measure* **where**
normal-distribution $\mu\sigma$ = (*if 0* < (*snd* $\mu\sigma$) *then density lborel* (λx. *ennreal* (*normal-density* (*fst* $\mu\sigma$) (*snd* $\mu\sigma$) *x*))
else return lborel 0)

lemma *normal-distribution-measurable*:
normal-distribution ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
proof(*rule measurable-prob-algebra-generated*[**where** Ω=*UNIV* **and** *G*=*borel*])
fix *A* :: *real set*
assume *h*:*A* ∈ *sets borel*
have (λx. *emeasure* (*normal-distribution x*) *A*) = (λx. *if 0* < (*snd x*) *then emeasure* (*density lborel* (λr. *ennreal* (*normal-density* (*fst x*) (*snd x*) *r*))) *A*
else emeasure (*return lborel 0*) *A*)
by(*auto simp add*: *normal-distribution-def*)
also have ... ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
proof(*rule measurable-If*)
have [*simp*]:(λx. *indicat-real A* (*snd x*)) ∈ *borel-measurable* ((*borel* $\bigotimes_M$ *borel*) $\bigotimes_M$ *borel*)
proof −
have (λx. *indicat-real A* (*snd x*)) = *indicat-real A* ∘ *snd*
by *auto*
also have ... ∈ *borel-measurable* ((*borel* $\bigotimes_M$ *borel*) $\bigotimes_M$ *borel*)
by (*meson borel-measurable-indicator h measurable-comp measurable-snd*)
finally show *?thesis* **.**
qed
have (λx. *emeasure* (*density lborel* (λr. *ennreal* (*normal-density* (*fst x*) (*snd x*) *r*))) *A*) = (λx. *set-nn-integral lborel A* (λr. *ennreal* (*normal-density* (*fst x*) (*snd x*) *r*)))
using *h* **by**(*auto intro*!: *emeasure-density*)
also have ... = (λx. $\int^+ r$. *ennreal* (*normal-density* (*fst x*) (*snd x*) *r* ∗ *indicat-real A r*)∂*lborel*)
by(*simp add*: *nn-integral-set-ennreal*)
also have ... ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
apply(*auto intro*!: *lborel.borel-measurable-nn-integral*
simp: *split-beta′ measurable-cong-sets*[*OF sets-pair-measure-cong*[*OF refl sets-lborel*]])
unfolding *normal-density-def*
by(*rule borel-measurable-times*) *simp-all*
finally show (λx. *emeasure* (*density lborel* (λr. *ennreal* (*normal-density* (*fst x*) (*snd x*) *r*))) *A*) ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*) **.**
qed *simp-all*
finally show (λx. *emeasure* (*normal-distribution x*) *A*) ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*) **.**
qed (*auto simp add*: *sets.sigma-sets-eq*[*of borel*,*simplified*] *sets.Int-stable prob-space-normal-density normal-distribution-def prob-space-return*)

definition *qbs-normal-distribution* :: *real* ⇒ *real* ⇒ *real qbs-prob-space* **where**

qbs-normal-distribution ≡ *curry* (*standard-borel-prob-space.as-qbs-measure* ∘ *normal-distribution*)

lemma *qbs-normal-distribution-morphism*:
qbs-normal-distribution ∈ $\mathbb{R}_Q \to_Q$ *exp-qbs* $\mathbb{R}_Q$ (*monadP-qbs* $\mathbb{R}_Q$)
unfolding *qbs-normal-distribution-def*
by(*rule curry-preserves-morphisms*[*OF real.measure-with-args-morphism*[*OF normal-distribution-measurable*,*simplified r-preserves-product*]])

context
fixes μ σ :: *real*
assumes *sigma*:$\sigma > 0$
begin

interpretation *n-dist*:*standard-borel-prob-space normal-distribution* (μ,σ)
by(*simp add*: *standard-borel-prob-space-def sigma prob-space-normal-density normal-distribution-def*)

lemma *qbs-normal-distribution-def2*:
qbs-normal-distribution μ σ = *n-dist.as-qbs-measure*
by(*simp add*: *qbs-normal-distribution-def*)

lemma *qbs-normal-distribution-integral*:
($\int_Q$ *x*. *f x* ∂ (*qbs-normal-distribution* μ σ)) = ($\int$ *x*. *f x* ∂ (*density lborel* (λ*x*. *ennreal* (*normal-density* μ σ *x*))))
by(*simp add*: *qbs-normal-distribution-def2 n-dist.integral-as-qbs-integral*)
(*simp add*: *normal-distribution-def sigma*)

lemma *qbs-normal-distribution-expectation*:
assumes *f* ∈ *real-borel* $\to_M$ *real-borel*
shows ($\int_Q$ *x*. *f x* ∂ (*qbs-normal-distribution* μ σ)) = ($\int$ *x*. *normal-density* μ σ *x* ∗ *f x* ∂ *lborel*)
by(*simp add*: *qbs-normal-distribution-integral assms integral-real-density integral-density*)

end

3.5.2 Uniform Distribution

definition *interval-uniform-distribution* :: *real* ⇒ *real* ⇒ *real measure* **where**
interval-uniform-distribution a b ≡ (*if a* < *b then uniform-measure lborel* {*a*<..<*b*}
else return lborel 0)

lemma *sets-interval-uniform-distribution*[*measurable-cong*]:
sets (*interval-uniform-distribution a b*) = *borel*
by(*simp add*: *interval-uniform-distribution-def*)

lemma *interval-uniform-distribution-meaurable*:

(*λr. interval-uniform-distribution* (*fst r*) (*snd r*)) ∈ *real-borel* ⨂$_M$ *real-borel* →$_M$ *prob-algebra real-borel*
proof(*rule measurable-prob-algebra-generated*[**where** Ω=*UNIV* **and** *G*=*range* (λ(*a*, *b*). {*a*<..<*b*})])
  show *sets real-borel* = *sigma-sets UNIV* (*range* (λ(*a*, *b*). {*a*<..<*b*}))
    by(*simp add*: *borel-eq-box*)
next
  show *Int-stable* (*range* (λ(*a*, *b*). {*a*<..<*b*::*real*}))
    by(*fastforce intro*!: *Int-stableI simp*: *split-beta′ image-iff*)
next
  show *range* (λ(*a*, *b*). {*a*<..<*b*}) ⊆ *Pow UNIV*
    by *simp*
next
  fix *a*
  show *prob-space* (*interval-uniform-distribution* (*fst a*) (*snd a*))
   by(*simp add*: *interval-uniform-distribution-def prob-space-return prob-space-uniform-measure*)
next
  fix *a*
  show *sets* (*interval-uniform-distribution* (*fst a*) (*snd a*)) = *sets real-borel*
    by(*simp add*: *interval-uniform-distribution-def*)
next
  fix *A*
  assume *A* ∈ *range* (λ(*a*, *b*). {*a*<..<*b*::*real*})
  then obtain *a b* **where** *ha*:*A* = {*a*<..<*b*} **by** *auto*
  consider *b* ≤ *a* | *a* < *b* **by** *fastforce*
  then show (λ*x*. *emeasure* (*interval-uniform-distribution* (*fst x*) (*snd x*)) *A*) ∈ *real-borel* ⨂$_M$ *real-borel* →$_M$ *ennreal-borel*
      (**is** *?f* ∈ *?meas*)
  proof *cases*
    case *1*
    then show *?thesis*
      by(*simp add*: *ha*)
  next
    case *h2*:*2*
    have *?f* = (λ*x*. *if fst x* < *snd x then ennreal* (*min* (*snd x*) *b* − *max* (*fst x*) *a*) / *ennreal* (*snd x* − *fst x*) *else indicator A 0*)
    proof(*standard*; *auto simp*: *interval-uniform-distribution-def ha*)
      fix *x y* :: *real*
      assume *hxy*:*x* < *y*
      consider *b* ≤ *x* | *a* ≤ *x* ∧ *x* < *b* | *x* < *a* ∧ *a* < *y* | *y* ≤ *a*
        using *h2* **by** *fastforce*
      thus *emeasure lborel* ({*max x a*<..<*min y b*}) / *ennreal* (*y* − *x*) = *ennreal* (*min y b* − *max x a*) / *ennreal* (*y* − *x*)
        by *cases* (*use hxy ennreal-neg h2* **in** *auto*)
    qed
    also have ... ∈ *?meas*
      by *simp*
    finally show *?thesis* **.**
  qed

qed

definition *qbs-interval-uniform-distribution* :: *real* ⇒ *real* ⇒ *real qbs-prob-space*
where
qbs-interval-uniform-distribution ≡ *curry* (*standard-borel-prob-space.as-qbs-measure* ∘ (*λr. interval-uniform-distribution* (*fst r*) (*snd r*)))

lemma *qbs-interval-uniform-distribution-morphism*:
qbs-interval-uniform-distribution ∈ $\mathbb{R}_Q \rightarrow_Q$ *exp-qbs* $\mathbb{R}_Q$ (*monadP-qbs* $\mathbb{R}_Q$)
unfolding *qbs-interval-uniform-distribution-def*
using *curry-preserves-morphisms*[*OF real.measure-with-args-morphism*[*OF interval-uniform-distribution-meaurable*,*simplified r-preserves-product*]] **.**

context
fixes *a b* :: *real*
assumes *a-less-than-b*:*a* < *b*
begin

definition *ab-qbs-uniform-distribution* ≡ *qbs-interval-uniform-distribution a b*

interpretation *ab-u-dist*: *standard-borel-prob-space interval-uniform-distribution a b*
by(*auto intro*!: *prob-space-uniform-measure simp*: *interval-uniform-distribution-def standard-borel-prob-space-def prob-space-return*)

lemma *qbs-interval-uniform-distribution-def2*:
ab-qbs-uniform-distribution = *ab-u-dist.as-qbs-measure*
by(*simp add*: *qbs-interval-uniform-distribution-def ab-qbs-uniform-distribution-def*)

lemma *qbs-uniform-distribution-expectation*:
assumes *f* ∈ $\mathbb{R}_Q \rightarrow_Q \mathbb{R}_{Q\geq 0}$
shows ($\int^{+}_Q$ *x*. *f x* ∂*ab-qbs-uniform-distribution*) = ($\int^{+}$*x* ∈ {*a*<..<*b*}. *f x* ∂*lborel*) / (*b* − *a*)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\int^{+}$*x*. *f x* ∂(*interval-uniform-distribution a b*))
using *assms* **by**(*auto simp*: *qbs-interval-uniform-distribution-def2 intro*!:*ab-u-dist.ennintegral-as-qbs-enninte*
dest:*ab-u-dist.qbs-morphism-dest*[*simplified measure-to-qbs-cong-sets*[*OF sets-interval-uniform-distribution*]])
also have ... = *?rhs*
using *assms*
by(*auto simp*: *interval-uniform-distribution-def a-less-than-b intro*!:*nn-integral-uniform-measure*[**where** *M*=*lborel* **and** *S*={*a*<..<*b*},*simplified emeasure-lborel-Ioo*[*OF order.strict-implies-order*[*OF a-less-than-b*]]])
finally show *?thesis* **.**
qed

end

3.5.3 Bernoulli Distribution

definition *qbs-bernoulli* :: *real* ⇒ *bool qbs-prob-space* **where**
qbs-bernoulli ≡ *standard-borel-prob-space.as-qbs-measure* ∘ (λ*x*. *measure-pmf* (*bernoulli-pmf* *x*))

lemma *bernoulli-measurable*:
(λ*x*. *measure-pmf* (*bernoulli-pmf x*)) ∈ *real-borel* $\rightarrow_M$ *prob-algebra bool-borel*
proof(*rule measurable-prob-algebra-generated*[**where** Ω=*UNIV* **and** *G*=*UNIV*],*simp-all*)
fix *A* :: *bool set*
have *A* ⊆ {*True*,*False*}
by *auto*
then consider *A* = {} | *A* = {*True*} | *A* = {*False*} | *A* = {*False*, *True*}
by *auto*
thus (λ*a*. *emeasure* (*measure-pmf* (*bernoulli-pmf a*)) *A*) ∈ *borel-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-bernoulli ∈ $\mathbb{R}_Q \rightarrow_Q$ *monadP-qbs* $\mathbb{B}_Q$
using *bool.measure-with-args-morphism*[*OF bernoulli-measurable*]
by (*simp add*: *qbs-bernoulli-def*)

lemma *qbs-bernoulli-measure*:
qbs-prob-measure (*qbs-bernoulli p*) = *measure-pmf* (*bernoulli-pmf p*)
using *bool.measure-with-args-recover*[*of* λ*x*. *measure-pmf* (*bernoulli-pmf x*) *real-borel p*] *bernoulli-measurable*
by(*simp add*: *measurable-def qbs-bernoulli-def*)

context
fixes *p* :: *real*
assumes *pgeq-0*[*simp*]:*0* ≤ *p* **and** *pleq-1*[*simp*]:*p* ≤ *1*
begin

lemma *qbs-bernoulli-expectation*:
($\int_Q$ *x*. *f x* ∂*qbs-bernoulli p*) = *f True* ∗ *p* + *f False* ∗ (*1* − *p*)
by(*simp add*: *qbs-prob-integral-def2 qbs-bernoulli-measure*)

end

end

3.6 Bayesian Linear Regression

theory *Bayesian-Linear-Regression*
imports *Measure-as-QuasiBorel-Measure*
begin

We formalize the Bayesian linear regression presented in [1] section VI.

3.6.1 Prior

abbreviation $\nu \equiv$ *density lborel* ($\lambda x.$ *ennreal* (*normal-density 0 3 x*))

interpretation ν: *standard-borel-prob-space* ν
by(*simp add*: *standard-borel-prob-space-def prob-space-normal-density*)

term ν*.as-qbs-measure* :: *real qbs-prob-space*
definition *prior* :: (*real* $\Rightarrow$ *real*) *qbs-prob-space* **where**
prior $\equiv$ *do* { $s \leftarrow \nu$*.as-qbs-measure* ;
$b \leftarrow \nu$*.as-qbs-measure* ;
qbs-return ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) ($\lambda r.\ s * r + b$)}

lemma ν*-as-qbs-measure-eq*:
ν*.as-qbs-measure* = *qbs-prob-space* ($\mathbb{R}_Q$,*id*,ν)
by(*simp add*: ν*.as-qbs-measure-retract*[*of id id*] *distr-id'* *measure-to-qbs-cong-sets*[*OF sets-density*] *measure-to-qbs-cong-sets*[*OF sets-lborel*])

interpretation ν*-qp*: *pair-qbs-prob* $\mathbb{R}_Q$ *id* ν $\mathbb{R}_Q$ *id* ν
by(*auto intro*!: *qbs-probI prob-space-normal-density simp*: *pair-qbs-prob-def*)

lemma ν*-as-qbs-measure-in-Pr*:
ν*.as-qbs-measure* $\in$ *monadP-qbs-Px* $\mathbb{R}_Q$
by(*simp add*: ν*-as-qbs-measure-eq* ν*-qp.qp1.qbs-prob-space-in-Px*)

lemma *sets-real-real-real*[*measurable-cong*]:
sets (*qbs-to-measure* (($\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$) $\bigotimes_Q \mathbb{R}_Q$)) = *sets* ((*borel* $\bigotimes_M$ *borel*) $\bigotimes_M$ *borel*)
by (*metis pair-standard-borel.l-r-r-sets pair-standard-borel-def r-preserves-product real.standard-borel-axioms real-real.standard-borel-axioms*)

lemma *lin-morphism*:
$(\lambda(s,\ b)\ r.\ s * r + b) \in \mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q \rightarrow_Q \mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$
apply(*simp add*: *split-beta'*)
apply(*rule curry-preserves-morphisms*[*of* $\lambda(x,r).$ *fst* $x * r +$ *snd* x,*simplified curry-def split-beta'*,*simplified*])
by *auto*

lemma *lin-measurable*[*measurable*]:
$(\lambda(s,\ b)\ r.\ s * r + b) \in$ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *qbs-to-measure* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$)
using *lin-morphism l-preserves-morphisms*[*of* $\mathbb{R}_Q \bigotimes_Q \mathbb{R}_Q$ *exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$]
by *auto*

lemma *prior-computation*:
qbs-prob ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) (($\lambda(s,\ b)\ r.\ s * r + b$) $\circ$ *real-real.g*) (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*)
prior = *qbs-prob-space* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$, ($\lambda(s,\ b)\ r.\ s * r + b$) $\circ$ *real-real.g*, *distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*)
using ν*-qp.qbs-bind-bind-return*[*OF lin-morphism*]

by(*simp-all add*: *prior-def* ν*-as-qbs-measure-eq map-prod-def*)

The following lemma corresponds to the equation (5).

lemma *prior-measure*:
qbs-prob-measure prior = *distr* ($\nu \bigotimes_M \nu$) (*qbs-to-measure* (*exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$)) (λ(*s*,*b*) *r*. *s* ∗ *r* + *b*)
by(*simp add*: *prior-computation*(*2*) *qbs-prob.qbs-prob-measure-computation*[*OF prior-computation*(*1*)]) (*simp add*: *distr-distr comp-def*)

lemma *prior-in-space*:
prior ∈ *qbs-space* (*monadP-qbs* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$))
using *qbs-prob.qbs-prob-space-in-Px*[*OF prior-computation*(*1*)]
by(*simp add*: *prior-computation*(*2*))

3.6.2 Likelihood

abbreviation *d* μ *x* ≡ *normal-density* μ (*1/2*) *x*

lemma *d-positive* : *0* < *d* μ *x*
by(*simp add*: *normal-density-pos*)

definition *obs* :: (*real* ⇒ *real*) ⇒ *ennreal* **where**
obs f ≡ *d* (*f 1*) *2.5* ∗ *d* (*f 2*) *3.8* ∗ *d* (*f 3*) *4.5* ∗ *d* (*f 4*) *6.2* ∗ *d* (*f 5*) *8*

lemma *obs-morphism*:
obs ∈ $\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q \rightarrow_Q \mathbb{R}_{Q\geq 0}$
proof(*rule qbs-morphismI*)
fix α
assume α ∈ *qbs-Mx* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$)
then have [*measurable*]:(λ(*x*,*y*). α *x y*) ∈ *real-borel* $\bigotimes_M$ *real-borel* $\rightarrow_M$ *real-borel*
by(*auto simp*: *exp-qbs-Mx-def*)
show *obs* ∘ α ∈ *qbs-Mx* $\mathbb{R}_{Q\geq 0}$
by(*auto simp*: *comp-def obs-def normal-density-def*)
qed

lemma *obs-measurable*[*measurable*]:
obs ∈ *qbs-to-measure* (*exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$) $\rightarrow_M$ *ennreal-borel*
using *obs-morphism* **by** *auto*

3.6.3 Posterior

lemma *id-obs-morphism*:
(λf. (*f*,*obs f*)) ∈ $\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q \rightarrow_Q (\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q) \bigotimes_Q \mathbb{R}_{Q\geq 0}$
by(*rule qbs-morphism-tuple*[*OF qbs-morphism-ident′ obs-morphism*])

lemma *push-forward-measure-in-space*:
monadP-qbs-Pf ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) (($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) $\bigotimes_Q \mathbb{R}_{Q\geq 0}$) ($\lambda f$. (*f*,*obs f*)) *prior* ∈ *qbs-space* (*monadP-qbs* (($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) $\bigotimes_Q \mathbb{R}_{Q\geq 0}$))
by(*rule qbs-morphismE*(*2*)[*OF monadP-qbs-Pf-morphism*[*OF id-obs-morphism*] *prior-in-space*])

lemma *push-forward-measure-computation*:
qbs-prob $((\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q) \bigotimes_Q \mathbb{R}_{Q\geq 0})$ $(\lambda l.$ $(((\lambda(s,\ b)\ r.\ s * r + b) \circ$ *real-real.g*$)$ l, $((obs \circ (\lambda(s,\ b)\ r.\ s * r + b)) \circ$ *real-real.g*$)$ $l))$ $($*distr* $(\nu \bigotimes_M \nu)$ *real-borel* *real-real.f*$)$
monadP-qbs-Pf $(\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q)$ $((\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q) \bigotimes_Q \mathbb{R}_{Q\geq 0})$ $(\lambda f.\ (f,\ obs\ f))$ *prior* = *qbs-prob-space* $((\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q) \bigotimes_Q \mathbb{R}_{Q\geq 0},$ $(\lambda l.$ $(((\lambda(s,\ b)\ r.\ s * r + b) \circ$ *real-real.g*$)$ l, $((obs \circ (\lambda(s,\ b)\ r.\ s * r + b)) \circ$ *real-real.g*$)$ $l)),$ *distr* $(\nu \bigotimes_M \nu)$ *real-borel* *real-real.f*$)$
using *qbs-prob.monadP-qbs-Pf-computation*[*OF prior-computation id-obs-morphism*]
by(*auto simp*: *comp-def*)

3.6.4 Normalizer

We use the unit space for an error.

definition *norm-qbs-measure* :: ($'a \times$ *ennreal*) *qbs-prob-space* $\Rightarrow$ $'a$ *qbs-prob-space* + *unit* **where**
norm-qbs-measure p $\equiv$ (*let* (XR,$\alpha\beta$,ν) = *rep-qbs-prob-space p in*
if emeasure (*density* ν (*snd* $\circ$ $\alpha\beta$)) *UNIV* = *0 then Inr* ()
else if emeasure (*density* ν (*snd* $\circ$ $\alpha\beta$)) *UNIV* = ∞ *then Inr* ()
else Inl (*qbs-prob-space* (*map-qbs fst XR*, *fst* $\circ$ $\alpha\beta$, *density* ν ($\lambda r.$ *snd* ($\alpha\beta$ r) / *emeasure* (*density* ν (*snd* $\circ$ $\alpha\beta$)) *UNIV*))))

lemma *norm-qbs-measure-qbs-prob*:
assumes *qbs-prob* $(X \bigotimes_Q \mathbb{R}_{Q\geq 0})$ $(\lambda r.\ (\alpha\ r,\ \beta\ r))$ μ
emeasure (*density* μ β) *UNIV* $\neq$ *0*
and *emeasure* (*density* μ β) *UNIV* $\neq \infty$
shows *qbs-prob X* α (*density* μ ($\lambda r.$ (β r) / *emeasure* (*density* μ β) *UNIV*))
proof −
interpret *qp*: *qbs-prob* $X \bigotimes_Q \mathbb{R}_{Q\geq 0}$ $\lambda r.$ $(\alpha\ r,\ \beta\ r)$ μ
by *fact*
have *ha*[*simp*]: $\alpha \in$ *qbs-Mx X*
and *hb*[*measurable*] :$\beta \in$ *real-borel* $\rightarrow_M$ *ennreal-borel*
using *assms* **by**(*simp-all add*: *qbs-prob-def in-Mx-def pair-qbs-Mx-def comp-def*)
show *?thesis*
proof(*rule qbs-probI*)
show *prob-space* (*density* μ ($\lambda r.$ β r / *emeasure* (*density* μ β) *UNIV*))
proof(*rule prob-spaceI*)
show *emeasure* (*density* μ ($\lambda r.$ β r / *emeasure* (*density* μ β) *UNIV*)) (*space* (*density* μ ($\lambda r.$ β r / *emeasure* (*density* μ β) *UNIV*))) = *1*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *emeasure* (*density* μ ($\lambda r.$ β r / *emeasure* (*density* μ β) *UNIV*)) *UNIV*
by *simp*
also have ... = $(\int^{+} r{\in}UNIV.$ (β r / *emeasure* (*density* μ β) *UNIV*) $\partial\mu)$
by(*intro emeasure-density*) *auto*
also have ... = *integral*^{N} μ ($\lambda r.$ β r / *emeasure* (*density* μ β) *UNIV*)

by *simp*
also have ... = (*integral*^{N} μ β) / *emeasure* (*density* μ β) *UNIV*
by(*simp add*: *nn-integral-divide*)
also have ... = ($\int^+ r \in UNIV.\ \beta\ r\ \partial\mu$) / *emeasure* (*density* μ β) *UNIV*
by(*simp add*: *emeasure-density*)
also have ... = *1*
using *assms*(*2*,*3*) **by**(*simp add*: *emeasure-density divide-eq-1-ennreal*)
finally show *?thesis* **.**
qed
qed
qed *simp-all*
qed

lemma *norm-qbs-measure-computation*:
assumes *qbs-prob* ($X \bigotimes_Q \mathbb{R}_{Q\geq 0}$) ($\lambda r.$ (α r, β r)) μ
shows *norm-qbs-measure* (*qbs-prob-space* ($X \bigotimes_Q \mathbb{R}_{Q\geq 0}$, ($\lambda r.$ (α r, β r)), μ)) =
(*if emeasure* (*density* μ β) *UNIV* = *0 then Inr* ()
else if emeasure
(*density* μ β) *UNIV* = ∞ *then Inr* ()
else Inl (*qbs-prob-space*
(X, α, *density* μ ($\lambda r.$ (β r) / *emeasure* (*density* μ β) *UNIV*))))
proof −
interpret *qp*: *qbs-prob* $X \bigotimes_Q \mathbb{R}_{Q\geq 0}$ $\lambda r.$ (α r, β r) μ
by *fact*
have *ha*: $\alpha \in$ *qbs-Mx X*
and *hb*[*measurable*] :$\beta \in$ *real-borel* $\rightarrow_M$ *ennreal-borel*
using *assms* **by**(*simp-all add*: *qbs-prob-def in-Mx-def pair-qbs-Mx-def comp-def*)
show *?thesis*
unfolding *norm-qbs-measure-def*
proof(*rule qp.in-Rep-induct*)
fix *XR* $\alpha\beta'$ μ'
assume (*XR*,$\alpha\beta'$,μ') $\in$ *Rep-qbs-prob-space* (*qbs-prob-space* ($X \bigotimes_Q \mathbb{R}_{Q\geq 0}$, $\lambda r.$ (α r, β r), μ))
from *qp.if-in-Rep*[*OF this*]
have *h*:*XR* = $X \bigotimes_Q \mathbb{R}_{Q\geq 0}$
qbs-prob XR $\alpha\beta'$ μ'
qbs-prob-eq ($X \bigotimes_Q \mathbb{R}_{Q\geq 0}$, $\lambda r.$ (α r, β r), μ) (*XR*, $\alpha\beta'$, μ')
by *auto*
have *hint*: $\bigwedge f.\ f \in X \bigotimes_Q \mathbb{R}_{Q\geq 0} \rightarrow_Q \mathbb{R}_{Q\geq 0} \Longrightarrow (\int^+ x.\ f\ (\alpha\ x,\ \beta\ x)\ \partial\mu) = (\int^+ x.\ f\ (\alpha\beta'\ x)\ \partial\mu')$
using *h*(*3*)[*simplified qbs-prob-eq-equiv14*] **by**(*simp add*: *qbs-prob-eq4-def*)
interpret *qp'*: *qbs-prob XR* $\alpha\beta'$ μ'
by *fact*
have *ha'*: *fst* $\circ$ $\alpha\beta'$ $\in$ *qbs-Mx X* ($\lambda x.$ *fst* ($\alpha\beta'$ x)) $\in$ *qbs-Mx X*
and *hb'*[*measurable*]: *snd* $\circ$ $\alpha\beta'$ $\in$ *real-borel* $\rightarrow_M$ *ennreal-borel* ($\lambda x.$ *snd* ($\alpha\beta'$ x)) $\in$ *real-borel* $\rightarrow_M$ *ennreal-borel* ($\lambda x.$ *fst* ($\alpha\beta'$ x)) $\in$ *real-borel* $\rightarrow_M$ *qbs-to-measure X*
using *h* **by**(*simp-all add*: *qbs-prob-def in-Mx-def pair-qbs-Mx-def comp-def*)
have *fstX*: *map-qbs fst XR* = *X*
by(*simp add*: *h*(*1*) *pair-qbs-fst*)

have *he*:*emeasure* (*density* μ β) *UNIV* = *emeasure* (*density* μ' (*snd* $\circ$ $\alpha\beta'$)) *UNIV*
using *hint*[*OF snd-qbs-morphism*] **by**(*simp add*: *emeasure-density*)

show (*let a* = (*XR*,$\alpha\beta'$,μ') *in case a of* (*XR*, $\alpha\beta$, ν') $\Rightarrow$ *if emeasure* (*density* ν' (*snd* $\circ$ $\alpha\beta$)) *UNIV* = *0 then Inr* ()
else if emeasure (*density* ν' (*snd* $\circ$ $\alpha\beta$)) *UNIV* = ∞ *then Inr* ()
else Inl (*qbs-prob-space* (*map-qbs fst XR*, *fst* $\circ$ $\alpha\beta$, *density* ν' (λr. *snd* ($\alpha\beta$ *r*) / *emeasure* (*density* ν' (*snd* $\circ$ $\alpha\beta$)) *UNIV*))))
= (*if emeasure* (*density* μ β) *UNIV* = *0 then Inr* ()
else if emeasure (*density* μ β) *UNIV* = ∞ *then Inr* ()
else Inl (*qbs-prob-space* (*X*, α, *density* μ (λr. β *r* / *emeasure* (*density* μ β) *UNIV*))))
proof(*auto simp*: *he*[*symmetric*] *fstX*)
assume *het0*:*emeasure* (*density* μ β) *UNIV* $\neq$ $\top$
emeasure (*density* μ β) *UNIV* $\neq$ *0*
interpret *pqp*: *pair-qbs-prob X fst* $\circ$ $\alpha\beta'$ *density* μ' (λr. *snd* ($\alpha\beta'$ *r*) / *emeasure* (*density* μ β) *UNIV*) *X* α *density* μ (λr. β *r* / *emeasure* (*density* μ β) *UNIV*)
apply(*auto intro*!: *norm-qbs-measure-qbs-prob simp*: *pair-qbs-prob-def assms het0*)
using *het0*
by(*auto intro*!: *norm-qbs-measure-qbs-prob*[*of X fst* $\circ$ $\alpha\beta'$ *snd* $\circ$ $\alpha\beta'$,*simplified*,*OF h*(*2*)[*simplified h*(*1*)]] *simp*: *he*)

show *qbs-prob-space* (*X*, *fst* $\circ$ $\alpha\beta'$, *density* μ' (λr. *snd* ($\alpha\beta'$ *r*) / *emeasure* (*density* μ β) *UNIV*)) = *qbs-prob-space* (*X*, α, *density* μ (λr. β *r* / *emeasure* (*density* μ β) *UNIV*))
proof(*rule pqp.qbs-prob-space-eq4*)
fix *f*
assume *hf*[*measurable*]:*f* $\in$ *qbs-to-measure X* $\rightarrow_M$ *ennreal-borel*
show ($\int^+$ *x*. *f* ((*fst* $\circ$ $\alpha\beta'$) *x*) ∂*density* μ' (λr. *snd* ($\alpha\beta'$ *r*) / *emeasure* (*density* μ β) *UNIV*)) = ($\int^+$ *x*. *f* (α *x*) ∂*density* μ (λr. β *r* / *emeasure* (*density* μ β) *UNIV*))
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\int^+$ *x*. (λxr. (*snd xr*) / *emeasure* (*density* μ β) *UNIV* $*$ *f* (*fst xr*)) ($\alpha\beta'$ *x*) $\partial\mu'$)
by(*auto simp*: *nn-integral-density*)
also have ... = ($\int^+$ *x*. (λxr. (*snd xr*) / *emeasure* (*density* μ β) *UNIV* $*$ *f* (*fst xr*)) (α *x*,β *x*) $\partial\mu$)
by(*intro hint*[*symmetric*]) (*auto intro*!: *pair-qbs-morphismI*)
also have ... = *?rhs*
by(*simp add*: *nn-integral-density*)
finally show *?thesis* **.**
qed
qed *simp*
qed
qed

qed

lemma *norm-qbs-measure-morphism*:
norm-qbs-measure $\in$ *monadP-qbs* $(X \bigotimes_Q \mathbb{R}_{Q\geq 0}) \rightarrow_Q$ *monadP-qbs* X <+>$_Q$ 1_Q
proof(*rule qbs-morphismI*)
fix γ
assume $\gamma \in$ *qbs-Mx* (*monadP-qbs* $(X \bigotimes_Q \mathbb{R}_{Q\geq 0})$)
then obtain α g **where** *hc*:
$\alpha \in$ *qbs-Mx* $(X \bigotimes_Q \mathbb{R}_{Q\geq 0})$ $g \in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
$\gamma = (\lambda r.$ *qbs-prob-space* $(X \bigotimes_Q \mathbb{R}_{Q\geq 0}, \alpha, g\ r))$
using *rep-monadP-qbs-MPx*[*of* γ $(X \bigotimes_Q \mathbb{R}_{Q\geq 0})$] **by** *auto*
note [*measurable*] = *hc*(*2*) *measurable-prob-algebraD*[*OF hc*(*2*)]
have *setsg*[*measurable-cong*]:$\bigwedge r.$ *sets* $(g\ r)$ = *sets real-borel*
using *measurable-space*[*OF hc*(*2*)] **by**(*simp add*: *space-prob-algebra*)
then have *ha*: *fst* $\circ\ \alpha \in$ *qbs-Mx X*
and *hb*[*measurable*]: *snd* $\circ\ \alpha \in$ *real-borel* $\rightarrow_M$ *ennreal-borel* $(\lambda x.$ *snd* $(\alpha\ x)) \in$ *real-borel* $\rightarrow_M$ *ennreal-borel* $\bigwedge r.$ *snd* $\circ\ \alpha \in g\ r\ \rightarrow_M$ *ennreal-borel* $\bigwedge r.$ $(\lambda x.$ *snd* $(\alpha\ x)) \in g\ r\ \rightarrow_M$ *ennreal-borel*
using *hc*(*1*) **by**(*auto simp add*: *pair-qbs-Mx-def measurable-cong-sets*[*OF setsg refl*] *comp-def*)
have *emeas-den-meas*[*measurable*]: $\bigwedge U.$ $U \in$ *sets real-borel* $\Longrightarrow$ $(\lambda r.$ *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) $U) \in$ *real-borel* $\rightarrow_M$ *ennreal-borel*
by(*simp add*: *emeasure-density*)
have *S-setsc*:*UNIV* $-$ $(\lambda r.$ *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) *UNIV*) $-$‘ $\{0,\infty\}$ $\in$ *sets real-borel*
using *measurable-sets-borel*[*OF emeas-den-meas*] **by** *simp*
have *space-non-empty*:*qbs-space* (*monadP-qbs X*) $\neq \{\}$
using *ha qbs-empty-equiv monadP-qbs-empty-iff* [*of X*] **by** *auto*
have *g-meas*:$(\lambda r.$ *if* $r \in$ (*UNIV* $-$ $(\lambda r.$ *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) *UNIV*) $-$‘ $\{0,\infty\}$) *then density* $(g\ r)$ $(\lambda l.$ ((*snd* $\circ\ \alpha$) l) / *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) *UNIV*) *else return real-borel* $0) \in$ *real-borel* $\rightarrow_M$ *prob-algebra real-borel*
proof $-$
have H:$\bigwedge \Omega$ M N c $f.$ $\Omega \cap$ *space* $M \in$ *sets* $M \Longrightarrow c \in$ *space* $N \Longrightarrow$
$f \in$ *measurable* (*restrict-space* M Ω) $N \Longrightarrow (\lambda x.$ *if* $x \in \Omega$ *then* $f\ x$ *else* $c) \in$ *measurable* M N
by(*simp add*: *measurable-restrict-space-iff*)
show *?thesis*
proof(*rule H*)
show (*UNIV* $-$ $(\lambda r.$ *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) *UNIV*) $-$‘ $\{0, \infty\}$) $\cap$ *space real-borel* $\in$ *sets real-borel*
using *S-setsc* **by** *simp*
next
show $(\lambda r.$ *density* $(g\ r)$ $(\lambda l.$ ((*snd* $\circ\ \alpha$) l) / *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) *UNIV*)) $\in$ *restrict-space real-borel* (*UNIV* $-$ $(\lambda r.$ *emeasure* (*density* $(g\ r)$ (*snd* $\circ\ \alpha$)) *UNIV*) $-$‘ $\{0,\infty\}$) $\rightarrow_M$ *prob-algebra real-borel*
proof(*rule measurable-prob-algebra-generated*[**where** Ω=*UNIV* **and** G=*sets real-borel*])

fix *a*

assume *a* ∈ *space* (*restrict-space real-borel* (*UNIV* − (λ*r*. *emeasure* (*density* (*g r*) (*snd* ∘ α)) *UNIV*) −‘ {*0*, ∞}))
then have *1*:(∫ + *x*. *snd* (α *x*) ∂*g a*) ≠ *0* (∫ + *x*. *snd* (α *x*) ∂*g a*) ≠ ∞
by(*simp-all add*: *space-restrict-space emeasure-density*)
show *prob-space* (*density* (*g a*) (λ*l*. (*snd* ∘ α) *l* / *emeasure* (*density* (*g a*) (*snd* ∘ α)) *UNIV*))
using *1*
by(*auto intro*!: *prob-spaceI simp*: *emeasure-density nn-integral-divide divide-eq-1-ennreal*)
next
fix *U*
assume *1*:*U* ∈ *sets real-borel*
then have *2*:⋀*a*. *U* ∈ *sets* (*g a*) **by** *auto*
show (λ*a*. *emeasure* (*density* (*g a*) (λ*l*. (*snd* ∘ α) *l* / *emeasure* (*density* (*g a*) (*snd* ∘ α)) *UNIV*)) *U*) ∈ (*restrict-space real-borel* (*UNIV* − (λ*r*. *emeasure* (*density* (*g r*) (*snd* ∘ α)) *UNIV*) −‘ {*0*, ∞})) $\rightarrow_M$ *ennreal-borel*
using *1*
by(*auto intro*!: *measurable-restrict-space1 nn-integral-measurable-subprob-algebra2*[**where** *N*=*real-borel*] *simp*: *emeasure-density emeasure-density*[*OF* - *2*])
qed (*simp-all add*: *setsg sets.Int-stable sets.sigma-sets-eq*[*of real-borel*,*simplified*])
qed (*simp add*:*space-prob-algebra prob-space-return*)
qed

show *norm-qbs-measure* ∘ γ ∈ *qbs-Mx* (*monadP-qbs X* <+>$_Q$ *unit-quasi-borel*)
apply(*auto intro*!: *bexI*[*OF* - *S-setsc*] *bexI*[**where** *x*=(λ*r*. ())] *bexI*[**where** *x*=λ*r*. *qbs-prob-space* (*X*,*fst* ∘ α,*if r* ∈ (*UNIV* − (λ*r*. *emeasure* (*density* (*g r*) (*snd* ∘ α)) *UNIV*) −‘ {*0*,∞}) *then density* (*g r*) (λ*l*. ((*snd* ∘ α) *l*) / *emeasure* (*density* (*g r*) (*snd* ∘ α)) *UNIV*) *else return real-borel 0*)]
simp: *copair-qbs-Mx-equiv copair-qbs-Mx2-def space-non-empty*[*simplified*])
apply *standard*
apply(*simp add*: *hc*(*3*) *norm-qbs-measure-computation*[*of* - *fst* ∘ α *snd* ∘ α,*simplified*,*OF qbs-prob-MPx*[*OF hc*(*1*,*2*)]]])
apply(*simp add*: *monadP-qbs-MPx-def in-MPx-def*)
apply(*auto intro*!: *bexI*[*OF* - *ha*] *bexI*[*OF* - *g-meas*])
done
qed

The following is the semantics of the entire program.

definition *program* :: (*real* ⇒ *real*) *qbs-prob-space* + *unit* **where**
program ≡ *norm-qbs-measure* (*monadP-qbs-Pf* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) (($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) $\bigotimes_Q$ $\mathbb{R}_{Q\geq 0}$) (λ*f*. (*f*,*obs f*)) *prior*)

lemma *program-in-space*:
program ∈ *qbs-space* (*monadP-qbs* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) <+>$_Q$ 1_Q)
unfolding *program-def*
by(*rule qbs-morphismE*(*2*)[*OF norm-qbs-measure-morphism push-forward-measure-in-space*])

We calculate the normalizing constant.

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 $\mu\ \sigma\ x$ = *normal-density* $x\ \sigma\ \mu$
by(*simp add*: *normal-density-def power2-commute*)

lemma *normal-density-plus-shift*:
normal-density $\mu\ \sigma\ (x + y)$ = *normal-density* $(\mu - x)\ \sigma\ y$
by(*simp add*: *normal-density-def add.commute diff-diff-eq2*)

lemma *normal-density-times*:
assumes $\sigma > 0\ \sigma' > 0$
shows *normal-density* $\mu\ \sigma\ x$ * *normal-density* $\mu'\ \sigma'\ x = (1 / sqrt\ (2 * pi * (\sigma^2 + \sigma'^2))) * exp\ (- (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2)))$ * *normal-density* $((\mu*\sigma'^2 + \mu'*\sigma^2)/(\sigma^2 + \sigma'^2))\ (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))\ x$
(**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 $?lhs = exp\ (- ((x - \mu)^2 / (2 * \sigma^2))) * exp\ (- ((x - \mu')^2 / (2 * \sigma'^2))) / (sqrt\ (2 * pi * \sigma^2) * sqrt\ (2 * pi * \sigma'^2))$
by(*simp add*: *normal-density-def*)
also have ... $= exp\ (- ((x - \mu)^2 / (2 * \sigma^2)) - ((x - \mu')^2 / (2 * \sigma'^2))) / (sqrt\ (2 * pi * \sigma^2) * sqrt\ (2 * pi * \sigma'^2))$
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 ... = $(exp\ (- (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2))) / (sqrt\ (2 * pi * \sigma^2) * sqrt\ (2 * pi * \sigma'^2))) * sqrt\ (2 * pi * (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))^2) * normal\text{-}density\ ((\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))\ (\sigma * \sigma' / sqrt\ (\sigma^2 + \sigma'^2))\ x$
by(*simp add*: *exp-add*[*of* $- (x - (\mu * \sigma'^2 + \mu' * \sigma^2) / (\sigma^2 + \sigma'^2))^2 / (2 * (\sigma * \sigma'$

/ *sqrt* $(\sigma^2 + \sigma'^2))^2) - (\mu - \mu')^2$ / $(2 * (\sigma^2 + \sigma'^2))$,*simplified*] *normal-density-def*)
also have ... = *?rhs*
proof −
have *exp* $(- (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2)))$ / (*sqrt* $(2 * pi * \sigma^2)$ * *sqrt* $(2 * pi * \sigma'^2))$ * *sqrt* $(2 * pi * (\sigma * \sigma'$ / *sqrt* $(\sigma^2 + \sigma'^2))^2) = 1$ / *sqrt* $(2 * pi * (\sigma^2 + \sigma'^2))$ * *exp* $(- (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2)))$
using *assms* **by**(*simp add*: *real-sqrt-mult*)
thus *?thesis*
by *simp*
qed
finally show *?thesis* **.**
qed

lemma *normal-density-times'*:
assumes $\sigma > 0$ $\sigma' > 0$
shows a * *normal-density* $\mu\ \sigma\ x$ * *normal-density* $\mu'\ \sigma'\ x = a * (1$ / *sqrt* $(2 * pi * (\sigma^2 + \sigma'^2)))$ * *exp* $(- (\mu - \mu')^2 / (2 * (\sigma^2 + \sigma'^2)))$ * *normal-density* $((\mu*\sigma'^2 + \mu'*\sigma^2)/(\sigma^2 + \sigma'^2))$ $(\sigma * \sigma'$ / *sqrt* $(\sigma^2 + \sigma'^2))$ x
using *normal-density-times*[*OF assms*,*of* $\mu\ x\ \mu'$]
by (*simp add*: *mult.assoc*)

lemma *normal-density-times-minusx*:
assumes $\sigma > 0$ $\sigma' > 0$ $a \neq a'$
shows *normal-density* $(\mu - a*x)$ σ y * *normal-density* $(\mu' - a'*x)$ σ' $y = (1 / |a' - a|)$ * *normal-density* $((\mu' - \mu)/(a' - a))$ (*sqrt* $((\sigma^2 + \sigma'^2)/(a' - a)^2))$ x * *normal-density* $(((\mu - a*x)*\sigma'^2 + (\mu' - a'*x)*\sigma^2)/(\sigma^2 + \sigma'^2))$ $(\sigma * \sigma'$ / *sqrt* $(\sigma^2 + \sigma'^2))$ y
proof −
have *non0*:$a' - a \neq 0$
using *assms*(*3*) **by** *simp*
have 1 / *sqrt* $(2 * pi * (\sigma^2 + \sigma'^2))$ * *exp* $(- (\mu - a * x - (\mu' - a' * x))^2 / (2 * (\sigma^2 + \sigma'^2))) = 1 / |a' - a|$ * *normal-density* $((\mu' - \mu) / (a' - a))$ (*sqrt* $((\sigma^2 + \sigma'^2) / (a' - a)^2))$ x
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = 1 / *sqrt* $(2 * pi * (\sigma^2 + \sigma'^2))$ * *exp* $(- ((a' - a) * x - (\mu' - \mu))^2 / (2 * (\sigma^2 + \sigma'^2)))$
by(*simp add*: *ring-class.left-diff-distrib group-add-class.diff-diff-eq2 add.commute add-diff-eq*)
also have ... = 1 / *sqrt* $(2 * pi * (\sigma^2 + \sigma'^2))$ * *exp* $(- ((a' - a)^2 * (x - (\mu' - \mu)/(a' - a))^2) / (2 * (\sigma^2 + \sigma'^2)))$
proof −
have $((a' - a) * x - (\mu' - \mu))^2 = ((a' - a) * (x - (\mu' - \mu)/(a' - a)))^2$
using *non0* **by**(*simp add*: *ring-class.right-diff-distrib*[*of* $a'-a$ x])
also have ... = $(a' - a)^2 * (x - (\mu' - \mu)/(a' - a))^2$
by(*simp add*: *monoid-mult-class.power2-eq-square*)
finally show *?thesis*
by *simp*
qed

also have ... = *1 / sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ'^2)) ∗ *sqrt* (*2* ∗ *pi* ∗ (*sqrt* ((σ^2 + σ'^2)/(*a'* − *a*)^2))^2) ∗ *normal-density* ((*μ'* − *μ*) / (*a'* − *a*)) (*sqrt* ((σ^2 + σ'^2) / (*a'* − *a*)^2)) *x*
using *non0* **by** (*simp add*: *normal-density-def*)
also have ... = *?rhs*
proof −
have *1 / sqrt* (*2* ∗ *pi* ∗ (σ^2 + σ'^2)) ∗ *sqrt* (*2* ∗ *pi* ∗ (*sqrt* ((σ^2 + σ'^2)/(*a'* − *a*)^2))^2) = *1* / |*a'* − *a*|
using *assms* **by**(*simp add*: *real-sqrt-divide*[*symmetric*]) (*simp add*: *real-sqrt-divide*)
thus *?thesis*
by *simp*
qed
finally show *?thesis* **.**
qed
thus *?thesis*
by(*simp add*:*normal-density-times*[*OF assms*(*1*,*2*),*of μ* − *a*∗*x y μ'* − *a'*∗*x*])
qed

The following is the normalizing constant of the program.

abbreviation *C* ≡ *ennreal* ((*4* ∗ *sqrt 2* / (pi^2 ∗ *sqrt* (*66961* ∗ *pi*))) ∗ (*exp* (−(*1674761* / *1674025*))))

lemma *program-normalizing-constant*:
emeasure (*density* (*distr* (*ν* $\bigotimes_M$ *ν*) *real-borel real-real.f*) (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*)) *UNIV* = *C*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∫⁺ *x*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) *x* ∂ (*distr* (*ν* $\bigotimes_M$ *ν*) *real-borel real-real.f*))
by(*simp add*: *emeasure-density*)
also have ... = (∫⁺ *z*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*)) *z* ∂(*ν* $\bigotimes_M$ *ν*))
using *nn-integral-distr*[*of real-real.f ν* $\bigotimes_M$ *ν real-borel obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*,*simplified*]
by(*simp add*: *comp-def*)
also have ... = (∫⁺ *x*. ∫⁺ *y*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*)) (*x*, *y*) ∂*ν* ∂*ν*)
by(*simp only*: *ν-qp.nn-integral-snd*[**where** *f*=(*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*)),*simplified*,*symmetric*])
(*simp add*: *ν-qp.Fubini*[**where** *f*=(*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*)),*simplified*])
also have ... = (∫⁺ *x*. *2* / *45* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ *normal-density* (*5* / *3*) (*sqrt* (*181* / *180*)) *x* ∂*ν*)
proof(*rule nn-integral-cong*[**where** *M*=*ν*,*simplified*])
fix *x*
have [*measurable*]: (λ*y*. *obs* (λ*r*. *x* ∗ *r* + *y*)) ∈ *real-borel* $\rightarrow_M$ *ennreal-borel*
using *measurable-Pair2*[*of obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*)] **by** *auto*
show (∫⁺ *y*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*)) (*x*, *y*) ∂*ν*) = *2* / *45* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ *normal-density* (*5* / *3*) (*sqrt* (*181* / *180*)) *x*

(**is** *?lhs′* = *?rhs′*)
proof −
have *?lhs′* = (∫⁺ *y. ennreal* (*d* (*5* / *2* − *x*) *y* ∗ *d* (*19* / *5* − *x* ∗ *2*) *y* ∗ *d* (*9* / *2* − *x* ∗ *3*) *y* ∗ *d* (*31* / *5* − *x* ∗ *4*) *y* ∗ *d* (*8* − *x* ∗ *5*) *y* ∗ *normal-density 0 3 y*) ∂*lborel*)
by(*simp add*: *nn-integral-density obs-def normal-density-mu-x-swap*[**where** *x*=*5*/*2*] *normal-density-mu-x-swap*[**where** *x*=*19*/*5*] *normal-density-mu-x-swap*[**where** *x*=*9*/*2*] *normal-density-mu-x-swap*[**where** *x*=*31*/*5*] *normal-density-mu-x-swap*[**where** *x*=*8*] *normal-density-plus-shift ab-semigroup-mult-class.mult.commute*[*of ennreal* (*normal-density 0 3* -)] *ennreal-mult′*[*symmetric*])
also have ... = (∫⁺ *y. ennreal* (*2* / *45* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ *normal-density* (*5* / *3*) (*sqrt* (*181* / *180*)) *x* ∗ *normal-density* (*20* / *181* ∗ *9* ∗ (*5* − *3* ∗ *x*)) (*3* / (*2* ∗ *sqrt 5*) / *sqrt* (*181* / *20*)) *y*) ∂*lborel*)
proof(*rule nn-integral-cong*[**where** *M*=*lborel*,*simplified*])
fix *y*
have *d* (*5* / *2* − *x*) *y* ∗ *d* (*19* / *5* − *x* ∗ *2*) *y* ∗ *d* (*9* / *2* − *x* ∗ *3*) *y* ∗ *d* (*31* / *5* − *x* ∗ *4*) *y* ∗ *d* (*8* − *x* ∗ *5*) *y* ∗ *normal-density 0 3 y* = *2* / *45* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ *normal-density* (*5* / *3*) (*sqrt* (*181* / *180*)) *x* ∗ *normal-density* (*20* / *181* ∗ *9* ∗ (*5* − *3* ∗ *x*)) (*3* / (*2* ∗ *sqrt 5*) / *sqrt* (*181* / *20*)) *y*
(**is** *?lhs′′* = *?rhs′′*)
proof −
have *?lhs′′* = *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ *normal-density* (*63* / *20* − (*3* / *2*) ∗ *x*) (*sqrt 2* / *4*) *y* ∗ *d* (*9* / *2* − *x* ∗ *3*) *y* ∗ *d* (*31* / *5* − *x* ∗ *4*) *y* ∗ *d* (*8* − *x* ∗ *5*) *y* ∗ *normal-density 0 3 y*
proof −
have *d* (*5* / *2* − *x*) *y* ∗ *d* (*19* / *5* − *x* ∗ *2*) *y* = *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ *normal-density* (*63* / *20* − (*3* / *2*) ∗ *x*) (*sqrt 2* / *4*) *y*
by(*simp add*: *normal-density-times-minusx*[*of 1*/*2 1*/*2 1 2 5*/*2 x y 19*/*5*,*simplified ab-semigroup-mult-class.mult.commute*[*of 2 x*],*simplified*])
(*simp add*: *monoid-mult-class.power2-eq-square real-sqrt-divide division-ring-class.diff-divide-distrib*)
thus *?thesis*
by *simp*
qed
also have ... = *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ (*2* / *3*) ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ *normal-density* (*18* / *5* − *2* ∗ *x*) (*1* / (*2* ∗ *sqrt 3*)) *y* ∗ *d* (*31* / *5* − *x* ∗ *4*) *y* ∗ *d* (*8* − *x* ∗ *5*) *y* ∗ *normal-density 0 3 y*
proof −
have *1*:*sqrt 2* ∗ *sqrt 8* / (*8* ∗ *sqrt 3*) = *1* / (*2* ∗ *sqrt 3*)
by(*simp add*: *real-sqrt-divide*[*symmetric*] *real-sqrt-mult*[*symmetric*])
have *normal-density* (*63* / *20* − *3* / *2* ∗ *x*) (*sqrt 2* / *4*) *y* ∗ *d* (*9* / *2* − *x* ∗ *3*) *y* = (*2* / *3*) ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ *normal-density* (*18* / *5* − *2* ∗ *x*) (*1* / (*2* ∗ *sqrt 3*)) *y*
by(*simp add*: *normal-density-times-minusx*[*of sqrt 2* / *4 1* / *2 3* / *2 3 63* / *20 x y 9* / *2*,*simplified ab-semigroup-mult-class.mult.commute*[*of 3 x*],*simplified*])

(*simp add*: *monoid-mult-class.power2-eq-square real-sqrt-divide division-ring-class.diff-divide-distrib 1*)
thus *?thesis*
by *simp*
qed
also have ... = *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ (*2* / *3*) ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ (*1* / *2*) ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ *normal-density* (*17* / *4* − (*5* / *2*) ∗ *x*) (*1* / *4*) *y* ∗ *d* (*8* − *x* ∗ *5*) *y* ∗ *normal-density 0 3 y*
proof −
have *1*:*normal-density* (*18* / *5* − *2* ∗ *x*) (*1* / (*2* ∗ *sqrt 3*)) *y* ∗ *d* (*31* / *5* − *x* ∗ *4*) *y* = (*1* / *2*) ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ *normal-density* (*17* / *4* − *5* / *2* ∗ *x*) (*1* / *4*) *y*
by(*simp add*: *normal-density-times-minusx*[*of 1* / (*2* ∗ *sqrt 3*) *1* / *2 2 4 18* / *5 x y 31* / *5*,*simplified ab-semigroup-mult-class.mult.commute*[*of 4 x*],*simplified*])
(*simp add*: *monoid-mult-class.power2-eq-square real-sqrt-divide division-ring-class.diff-divide-distrib*)
show *?thesis*
by(*simp add*: *1 mult.assoc*)
qed
also have ... = *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ (*2* / *3*) ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ (*1* / *2*) ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ (*2* / *5*) ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ *normal-density* (*5* − *3* ∗ *x*) (*1* / (*2* ∗ *sqrt 5*)) *y* ∗ *normal-density 0 3 y*
proof −
have *1*:*normal-density* (*17* / *4* − *5* / *2* ∗ *x*) (*1* / *4*) *y* ∗ *d* (*8* − *x* ∗ *5*) *y* = (*2* / *5*) ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ *normal-density* (*5* − *3* ∗ *x*) (*1* / (*2* ∗ *sqrt 5*)) *y*
by(*simp add*: *normal-density-times-minusx*[*of 1* / *4 1* / *2 5* / *2 5 17* / *4 x y 8*,*simplified ab-semigroup-mult-class.mult.commute*[*of 5 x*],*simplified*])
(*simp add*: *monoid-mult-class.power2-eq-square real-sqrt-divide division-ring-class.diff-divide-distrib*)
show *?thesis*
by(*simp only*: *1 mult.assoc*)
qed
also have ... = *normal-density* (*13* / *10*) (*1* / *sqrt 2*) *x* ∗ (*2* / *3*) ∗ *normal-density* (*9* / *10*) (*1* / *sqrt 6*) *x* ∗ (*1* / *2*) ∗ *normal-density* (*13* / *10*) (*1* / *sqrt 12*) *x* ∗ (*2* / *5*) ∗ *normal-density* (*3* / *2*) (*1* / *sqrt 20*) *x* ∗ (*1* / *3*) ∗ *normal-density* (*5* / *3*) (*sqrt* (*181* / *180*)) *x* ∗ *normal-density* (*20* / *181* ∗ *9* ∗ (*5* − *3* ∗ *x*)) ((*3* / (*2* ∗ *sqrt 5*))/ *sqrt* (*181* / *20*)) *y*
proof −
have *normal-density* (*5* − *3* ∗ *x*) (*1* / (*2* ∗ *sqrt 5*)) *y* ∗ *normal-density 0 3 y* = (*1* / *3*) ∗ *normal-density* (*5* / *3*) (*sqrt* (*181* / *180*)) *x* ∗ *normal-density* (*20* / *181* ∗ *9* ∗ (*5* − *3* ∗ *x*)) ((*3* / (*2* ∗ *sqrt 5*))/ *sqrt* (*181* / *20*)) *y*
by(*simp add*: *normal-density-times-minusx*[*of 1* / (*2* ∗ *sqrt 5*) *3 3 0 5 x y 0*,*simplified*] *monoid-mult-class.power2-eq-square*)
thus *?thesis*
by(*simp only*: *mult.assoc*)

qed
also have ... = *?rhs''*
by *simp*
finally show *?thesis* **.**
qed
thus *ennreal(d (5 / 2 − x) y ∗ d (19 / 5 − x ∗ 2) y ∗ d (9 / 2 − x ∗ 3) y ∗ d (31 / 5 − x ∗ 4) y ∗ d (8 − x ∗ 5) y ∗ normal-density 0 3 y) = ennreal (2 / 45 ∗ normal-density (13 / 10) (1 / sqrt 2) x ∗ normal-density (9 / 10) (1 / sqrt 6) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density (20 / 181 ∗ 9 ∗ (5 − 3 ∗ x)) (3 / (2 ∗ sqrt 5) / sqrt (181 / 20)) y)*
by *simp*
qed
also have ... = (∫ ⁺ *y. ennreal (normal-density (20 / 181 ∗ 9 ∗ (5 − 3 ∗ x)) (3 / (2 ∗ sqrt 5) / sqrt (181 / 20)) y) ∗ ennreal (2 / 45 ∗ normal-density (13 / 10) (1 / sqrt 2) x ∗ normal-density (9 / 10) (1 / sqrt 6) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x) ∂lborel*)
by(*simp add*: *ab-semigroup-mult-class.mult.commute ennreal-mult'*[*symmetric*])
also have ... = (∫ ⁺ *y. ennreal (2 / 45 ∗ normal-density (13 / 10) (1 / sqrt 2) x ∗ normal-density (9 / 10) (1 / sqrt 6) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x) ∂ (density lborel (λy. ennreal (normal-density (20 / 181 ∗ 9 ∗ (5 − 3 ∗ x)) (3 / (2 ∗ sqrt 5) / sqrt (181 / 20)) y))))*
by(*simp add*: *nn-integral-density*[*of λy. ennreal* (*normal-density* (*20 / 181 ∗ 9 ∗* (*5 − 3 ∗ x*)) (*3 /* (*2 ∗ sqrt 5*) */ sqrt* (*181 / 20*)) *y*) *lborel*,*simplified*,*symmetric*])
also have ... = *?rhs'*
by(*simp add*: *prob-space.emeasure-space-1*[*OF prob-space-normal-density*[*of 3 /* (*2 ∗ sqrt 5 ∗ sqrt* (*181 / 20*)) *20 / 181 ∗ 9 ∗* (*5 − 3 ∗ x*)],*simplified*])
finally show *?thesis* **.**
qed
qed
also have ... = (∫ ⁺ *x. ennreal (2 / 45 ∗ normal-density (13 / 10) (1 / sqrt 2) x ∗ normal-density (9 / 10) (1 / sqrt 6) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density 0 3 x) ∂lborel*)
by(*simp add*: *nn-integral-density ab-semigroup-mult-class.mult.commute ennreal-mult'*[*symmetric*])
also have ... = (∫ ⁺ *x.* (*4 ∗ sqrt 2 /* (pi^2 *∗ sqrt* (*66961 ∗ pi*))) *∗ exp* (− (*1674761 / 1674025*)) *∗ normal-density* (*450072 / 334805*) (*3 ∗ sqrt 181 / sqrt 66961*) *x ∂lborel*)
proof(*rule nn-integral-cong*[**where** *M*=*lborel*,*simplified*])
fix *x*
show *ennreal (2 / 45 ∗ normal-density (13 / 10) (1 / sqrt 2) x ∗ normal-density (9 / 10) (1 / sqrt 6) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density 0 3 x) = ennreal ((4 ∗ sqrt 2 /* (pi^2 *∗ sqrt (66961 ∗ pi))) ∗ exp (− (1674761 / 1674025)) ∗ normal-density (450072 / 334805) (3 ∗ sqrt 181 / sqrt 66961) x)*

proof −

have *2 / 45 ∗ normal-density (13 / 10) (1 / sqrt 2) x ∗ normal-density (9 / 10) (1 / sqrt 6) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density 0 3 x = (4 ∗ sqrt 2 / (pi*^{2} *∗ sqrt (66961 ∗ pi))) ∗ exp (− (1674761 / 1674025)) ∗ normal-density (450072 / 334805) (3 ∗ sqrt 181 / sqrt 66961) x*

(**is** *?lhs′ = ?rhs′*)

proof −

have *?lhs′ = 2 / 45 ∗ exp (− (3 / 25)) / sqrt (4 ∗ pi / 3) ∗ normal-density 1 (1 / sqrt 8) x ∗ normal-density (13 / 10) (1 / sqrt 12) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density 0 3 x*

by(*simp add*: *normal-density-times′ monoid-mult-class.power2-eq-square real-sqrt-mult*[*symmetric*])

also have ... = *(2 / (15 ∗ pi ∗ sqrt 5)) ∗ exp (− (42 / 125)) ∗ normal-density (59 / 50) (1 / sqrt 20) x ∗ normal-density (3 / 2) (1 / sqrt 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density 0 3 x*

proof −

have *1*:*sqrt 8 ∗ sqrt 12 ∗ sqrt (5 / 24) = sqrt 20*

by(*simp add*:*real-sqrt-mult*[*symmetric*])

have *2*:*sqrt (5 ∗ pi / 12) ∗ (45 ∗ sqrt (4 ∗ pi / 3)) = 15 ∗ (pi ∗ sqrt 5)*

by(*simp add*: *real-sqrt-mult*[*symmetric*] *real-sqrt-divide*) (*simp add*: *real-sqrt-mult real-sqrt-mult*[*of 4 5*,*simplified*])

have *2 / 45 ∗ exp (− (3 / 25)) / sqrt (4 ∗ pi / 3) ∗ normal-density 1 (1 / sqrt 8) x ∗ normal-density (13 / 10) (1 / sqrt 12) x = (6 / (45 ∗ pi ∗ sqrt 5)) ∗ exp (− (42 / 125)) ∗ normal-density (59 / 50) (1 / sqrt 20) x*

by(*simp add*: *normal-density-times′ monoid-mult-class.power2-eq-square mult-exp-exp*[*of − (3 / 25) − (27 / 125)*,*simplified*,*symmetric*] *1 2*)

thus *?thesis*

by *simp*

qed

also have ... = *2 / (15 ∗ pi ∗ sqrt pi) ∗ exp (− (106 / 125)) ∗ normal-density (67 / 50) (sqrt 10 / 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x ∗ normal-density 0 3 x*

proof −

have *2 / (15 ∗ pi ∗ sqrt 5) ∗ exp (− (42 / 125)) ∗ normal-density (59 / 50) (1 / sqrt 20) x ∗ normal-density (3 / 2) (1 / sqrt 20) x = 2 / (15 ∗ pi ∗ sqrt pi) ∗ exp (− (106 / 125)) ∗ normal-density (67 / 50) (sqrt 10 /20) x*

by(*simp add*: *normal-density-times′ monoid-mult-class.power2-eq-square mult-exp-exp*[*of − (42 / 125) − (64 / 125)*,*simplified*,*symmetric*] *real-sqrt-divide*)
(*simp add*: *mult.commute*)

thus *?thesis*

by *simp*

qed

also have ... = *((4 ∗ sqrt 5) / (5 ∗ pi*^{2} *∗ sqrt 371)) ∗ exp (− (5961 / 6625)) ∗ normal-density (1786 / 1325) (sqrt 905 / (10 ∗ sqrt 371)) x ∗ normal-density 0 3 x*

proof −

have *1*:*sqrt (371 ∗ pi / 180) ∗ (15 ∗ pi ∗ sqrt pi) = 5 ∗ pi ∗ pi ∗ sqrt*

371 / (2 ∗ sqrt 5)
by(*simp add*: *real-sqrt-mult real-sqrt-divide real-sqrt-mult*[*of 36 5*,*simplified*])
have *22*:*10 = sqrt 5 ∗ 2 ∗ sqrt 5* **by** *simp*
have *2*:*sqrt 10 ∗ sqrt (181 / 180) / (20 ∗ sqrt (371 / 360)) = sqrt 905 / (10 ∗ sqrt 371)*
by(*simp add*: *real-sqrt-mult real-sqrt-divide real-sqrt-mult*[*of 36 5*,*simplified*] *real-sqrt-mult*[*of 36 10*,*simplified*] *real-sqrt-mult*[*of 181 5*,*simplified*])
(*simp add*: *mult.assoc*[*symmetric*] *22*)
have *2 / (15 ∗ pi ∗ sqrt pi) ∗ exp (− (106 / 125)) ∗ normal-density (67 / 50) (sqrt 10 / 20) x ∗ normal-density (5 / 3) (sqrt (181 / 180)) x = 4 ∗ sqrt 5 / (5 ∗ pi*^{2} *∗ sqrt 371) ∗ exp (− (5961 / 6625)) ∗ normal-density (1786 / 1325) (sqrt 905 / (10 ∗ sqrt 371)) x*
by(*simp add*: *normal-density-times′ monoid-mult-class.power2-eq-square mult-exp-exp*[*of − (106 / 125) − (343 / 6625)*,*simplified*,*symmetric*] *1 2*)
(*simp add*: *mult.assoc*)
thus *?thesis*
by *simp*
qed
also have ... = *?rhs′*
proof −
have *1*: *4 ∗ sqrt 5 / (sqrt (66961 ∗ pi / 3710) ∗ (5 ∗ (pi ∗ pi) ∗ sqrt 371)) = 4 ∗ sqrt 2 / (pi*^{2} *∗ sqrt (66961 ∗ pi))*
by(*simp add*: *real-sqrt-mult*[*of 10 371*,*simplified*] *real-sqrt-mult*[*of 5 2*,*simplified*] *real-sqrt-divide monoid-mult-class.power2-eq-square mult.assoc*)
(*simp add*: *mult.assoc*[*symmetric*])
have *2*: *sqrt 905 ∗ 3 / (10 ∗ sqrt 371 ∗ sqrt (66961 / 7420)) = 3 ∗ sqrt 181 / sqrt 66961*
by(*simp add*: *real-sqrt-mult*[*of 371 20*,*simplified*] *real-sqrt-divide real-sqrt-mult*[*of 4 5*,*simplified*] *real-sqrt-mult*[*of 181 5*,*simplified*] *mult.commute*[*of - 3*])
(*simp add*: *mult.assoc*)
show *?thesis*
by(*simp only*: *1*[*symmetric*]) (*simp add*: *normal-density-times′ monoid-mult-class.power2-eq-square mult-exp-exp*[*of − (5961 / 6625) − (44657144 / 443616625)*,*simplified*,*symmetric*] *2*)
qed
finally show *?thesis* **.**
qed
thus *?thesis*
by *simp*
qed
qed
also have ... = (∫⁺ *x. ennreal (normal-density (450072 / 334805) (3 ∗ sqrt 181 / sqrt 66961) x) ∗ (ennreal (4 ∗ sqrt 2 / (pi*^{2} *∗ sqrt (66961 ∗ pi))) ∗ exp (− (1674761 / 1674025))) ∂lborel)*
by(*simp add*: *ab-semigroup-mult-class.mult.commute ennreal-mult′*[*symmetric*])
also have ... = (∫⁺ *x. (ennreal (4 ∗ sqrt 2 / (pi*^{2} *∗ sqrt (66961 ∗ pi))) ∗ exp (− (1674761 / 1674025))) ∂(density lborel (λx. ennreal (normal-density (450072 / 334805) (3 ∗ sqrt 181 / sqrt 66961) x))))*
by(*simp add*: *nn-integral-density*[*symmetric*])

also have ... = *?rhs*
by(*simp add*: *prob-space.emeasure-space-1*[*OF prob-space-normal-density*,*simplified*] *ennreal-mult'*[*symmetric*])
finally show *?thesis* **.**
qed

The program returns a probability measure, rather than error.

lemma *program-result*:
qbs-prob ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$) (($\lambda$(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) (*density* (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*) (λ*r*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) *r* / *C*))
program = *Inl* (*qbs-prob-space* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$, ($\lambda$(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*, *density* (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*) (λ*r*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) *r* / *C*)))
using *norm-qbs-measure-computation*[*OF push-forward-measure-computation*(*1*),*simplified program-normalizing-constant*]
norm-qbs-measure-qbs-prob[*OF push-forward-measure-computation*(*1*),*simplified program-normalizing-constant*]
by(*simp-all add*: *push-forward-measure-computation program-def comp-def*)

lemma *program-inl*:
program ∈ *Inl* ‘ (*qbs-space* (*monadP-qbs* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$)))
using *program-in-space*[*simplified program-result*(*2*)]
by(*auto simp*: *image-def program-result*(*2*))

lemma *program-result-measure*:
qbs-prob-measure (*qbs-prob-space* ($\mathbb{R}_Q \Rightarrow_Q \mathbb{R}_Q$, ($\lambda$(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*, *density* (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*) (λ*r*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) *r* / *C*)))
= *density* (*qbs-prob-measure prior*) (λ*k*. *obs k* / *C*)
(**is** *?lhs* = *?rhs*)
proof −
interpret *qp*: *qbs-prob exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g density* (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*) (λ*r*. (*obs* ∘ (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) *r* / *C*)
by(*rule program-result*(*1*))
have *?lhs* = *distr* (*density* (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*) (λ*r*. *obs* (((λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*) *r*) / *C*)) (*qbs-to-measure* (*exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$)) ((λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*)
using *qp.qbs-prob-measure-computation* **by** *simp*
also have ... = *density* (*distr* (*distr* ($\nu \bigotimes_M \nu$) *real-borel real-real.f*) (*qbs-to-measure* (*exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$)) ((λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*)) (λ*k*. *obs k* / *C*)
by(*simp add*: *density-distr*)
also have ... = *?rhs*
by(*simp add*: *distr-distr comp-def prior-measure*)
finally show *?thesis* **.**
qed

lemma *program-result-measure'*:
qbs-prob-measure (*qbs-prob-space* (*exp-qbs* $\mathbb{R}_Q$ $\mathbb{R}_Q$, (λ(*s*, *b*) *r*. *s* ∗ *r* + *b*) ∘ *real-real.g*,

$density\ (distr\ (\nu \bigotimes_M \nu)\ real\text{-}borel\ real\text{-}real.f)\ (\lambda r.\ (obs \circ (\lambda(s,\ b)\ r.\ s * r + b) \circ real\text{-}real.g)\ r\ /\ C)))$
$= distr\ (density\ (\nu \bigotimes_M \nu)\ (\lambda(s,b).\ obs\ (\lambda r.\ s * r + b)\ /\ C))\ (qbs\text{-}to\text{-}measure\ (exp\text{-}qbs\ \mathbb{R}_Q\ \mathbb{R}_Q))\ (\lambda(s,\ b)\ r.\ s * r + b)$
by(*simp only*: *program-result-measure distr-distr*) (*simp add*: *density-distr split-beta′ prior-measure*)

end

References

[1] C. Heunen, O. Kammar, S. Staton, and H. Yang. A convenient category for higher-order probability theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '17. IEEE Press, 2017.