Formalizing Quasi-Borel Spaces in Lean 4

Anthony Vandikas and Kiarash Sotoudeh

University of Toronto

Abstract. Quasi-Borel spaces provide an alternative to the measure-theoretic formulation of probability theory that naturally supports higher-order functions, making them well suited for formalizing the semantics of higher-order probabilistic programming languages. Quasi-Borel spaces also form a conservative extension of standard Borel spaces, preserving the usual results of measure-theoretic probability.

Our goal is to obtain machine-checked semantics for probabilistic programming languages within Lean. To this end, we have formalized quasi-Borel spaces as a library in Lean 4, building on mathlib's existing development of measure theory and probability theory. We have integrated our developments of quasi-Borel spaces with mathlib's existing automation for proving function properties (i.e. `@[fun_prop]`).

1 Introduction

Probability theory is traditionally formalized through measure theory [Kallenberg, 2021]. One drawback of this approach is that it does not generalize well to higher-order functions: the category of measurable spaces is not cartesian closed. In particular, there is no measurable space structure on functions $(A \to B)$ such that $\text{apply} : (A \to B) \times A \to B = (f, x) \mapsto f(x)$ is measurable.

Quasi-Borel spaces [Heunen et al., 2017] are a conservative extension of standard Borel spaces and hence form an alternative foundation for probability theory. In contrast to measurable spaces, quasi-Borel spaces form a cartesian-closed category, which makes them particularly well-suited for higher-order probability theory. Quasi-Borel spaces achieve this flexibility by taking random variables, rather than measurable sets, as primitive.

Our goal is to obtain machine-checked semantics for higher-order probabilistic programming languages. To this end, we have formalized quasi-Borel spaces as a library in Lean 4 [de Moura and Ullrich, 2021], building on mathlib's [The mathlib Community, 2020] existing development of measure theory and probability theory.

Our contributions are as follows: (1) We formalize the category of quasi-Borel spaces, including morphisms, products, coproducts, and exponentials. (2) We formalize probability measures and prove they form a strong commutative monad. (3) Building on the core theory, we provide several additional constructions, including subtypes, quotients, and lists. We prove that each construction's basic operations are morphisms. (4) We compare our formalization to an existing formalization in Isabelle [Hirata et al., 2022].

2 Core Formalization

In this section, we present the core of our formalization, which includes quasi-Borel spaces, morphisms, products, coproducts, exponentials, and probability measures. We assume reader familiarity with basic measure theory [Kallenberg, 2021]: σ-algebras, measurable spaces, measurable sets, measurable functions, measures, Borel sets, and probability spaces.

2.1 Quasi-Borel Spaces

We recall the definition of quasi-Borel space by Heunen et al. [2017].

Definition 1 (Heunen et al. [2017]). *A* quasi-Borel space *is a set X together with a set $M_X \subseteq [\mathbb{R} \to X]$ satisfying*

```
class QuasiBorelSpace (A : Type*) where
  IsVar : (ℝ → A) → Prop
  isVar_const (x : A) : IsVar (fun _ : ℝ ↦ x)
  isVar_comp {f φ} : Measurable f → IsVar φ → IsVar (fun r ↦ φ (f r))
  isVar_cases {ix : ℝ → ℕ} {φ : ℕ → ℝ → A} :
    Measurable ix → (∀n, IsVar (φ n)) → IsVar (fun r ↦ φ (ix r) r)
```

Fig. 1. Quasi-Borel spaces as a `class` definition.

```
def ofMeasurableSpace {A} [MeasurableSpace A] : QuasiBorelSpace A where
  IsVar φ := Measurable φ
  ...
instance : QuasiBorelSpace ℝ := ofMeasurableSpace
instance : QuasiBorelSpace ℕ := ofMeasurableSpace
... -- and so on for other base types (Prop, Bool, etc.)
```

Fig. 2. Converting `MeasurableSpaces` to `QuasiBorelSpaces`.

(1) $\phi \circ f \in M_X$ *if* $\phi \in M_X$ *and* $f : \mathbb{R} \to \mathbb{R}$ *is measurable;*
(2) $\phi \in M_X$ *if* $\phi : \mathbb{R} \to X$ *is constant;*
(3) if $\mathbb{R} = \biguplus_{i \in \mathbb{N}} S_i$*, with each set* S_i *Borel, and* $\phi_1, \phi_2, \ldots \in M_X$*, then* ψ *is in* M_X*, where* $\psi(r) = \phi_i(r)$ *for* $r \in S_i$*.*

We call the set M_X a set of *random variables*, i.e., quasi-Borel spaces axiomatize random variables instead of measurable sets.

We formalize Definition 1 as a *semi-bundled* [The mathlib Community, 2020] `class` definition in Figure 1. We formalize the set of variables M_X as an associated predicate `IsVar`, and translate properties (1-2) of Definition 1 in a straightforward fashion. Instead of translating property (3) directly (which would involve complicated `Subtypes`), we use a simplified reformulation based on the observation that a countable Borel partitioning $\mathbb{R} = \biguplus_{i \in \mathbb{N}} S_i$ is equivalent to a measurable function $ix : \mathbb{R} \to \mathbb{N}$ assigning each $r \in \mathbb{R}$ to a unique partition index $ix(r)$. Definition 1 involves measurability over $\mathbb{R}$ and $\mathbb{N}$, hence our formalization builds on the existing formalization of measure theory in mathlib [The mathlib Community, 2020].

From this point on we make use of the following `variable` declaration:

```
variable {A B C I : Type*} {P : I → Type*} [∀ i, QuasiBorelSpace (P i)]
    [QuasiBorelSpace A] [QuasiBorelSpace B] [QuasiBorelSpace C]
```

Any measurable space gives rise to a quasi-Borel space by defining random variables to be measurable functions [Heunen et al., 2017, III-B]. We formalize this construction in Figure 2. The `ofMeasurableSpace` function is particularly useful for quickly constructing `QuasiBorelSpace` instances for common *unparameterized* types such as $\mathbb{N}$ and $\mathbb{R}$ which already have `MeasurableSpace` instances.

2.2 Morphisms

Definition 2 (Heunen et al. [2017]). *A* morphism *of quasi-Borel spaces* $(X, M_X) \to (Y, M_Y)$ *is a function* $f : X \to Y$ *such that* $f \circ \phi \in M_Y$ *if* $\phi \in M_X$*.*

Morphisms (Definition 2) are the analogue of measurable functions for quasi-Borel spaces. Intuitively, a morphism preserves membership in the set of variables (M_X), similar to how measurable functions preserve measurable sets. We formalize Definition 2 in Figure 3, along with some useful lemmas.

The `IsHom` predicate, as well as the `isHom_id`, `isHom_const`, and `isHom_comp` lemmas, are annotated with the `@[fun_prop]` attribute. This attribute integrates `IsHom` with mathlib's `fun_prop` tactic for automatically proving function properties. We give an example of its utility in Section 2.4.

A function $f : \mathbb{R} \to X$ into a quasi-Borel space (X, M_X) is a random variable (i.e., a member of M_X) iff it is a morphism (see Figure 3, `lemma isVar_iff_isHom`). We structure the remainder of our formalization around this fact: we never deal with `IsVar` directly except when defining a `QuasiBorelSpace` instance.

```
def IsHom (f : A → B) : Prop := ∀ {|φ|}, IsVar φ → IsVar (f ∘ φ)
lemma isHom_id : IsHom (fun x : A ↦ x)
lemma isHom_const (x : B) : IsHom (fun _ : A ↦ y)
lemma isHom_comp {f : B → C} {g : A → B} (hf : IsHom f) (hg : IsHom g) :
  IsHom (fun x ↦ f (g x))
lemma isVar_iff_isHom (f : ℝ → A) : IsVar f ↔ IsHom f
attribute [fun_prop] IsHom isHom_id isHom_const isHom_comp
```

Fig. 3. `QuasiBorelSpace` morphisms and basic lemmas.

```
instance : QuasiBorelSpace ((i : I) → P i) where
  IsVar φ := ∀ i, IsHom (φ · i)
  ···
lemma Pi.isHom_apply (i : I) : IsHom (fun (f : (i : I) → P i) ↦ f i)
lemma Pi.isHom_pi {f : A → ∀ i, P i} (hf : ∀ i, IsHom (f · i)) : IsHom f
attribute [fun_prop] Pi.isHom_apply Pi.isHom_pi
```

Fig. 4. Products.

2.3 Products and Coproducts

We provide `QuasiBorelSpace` constructions for arbitrary products `((i : I) → P i)` and coproducts `((i : I) × P i)`. Our construction for products (Figure 4) is straightforward and follows the description given by Heunen et al. [2017]. Our construction for coproducts (Figure 5) is considerably more complicated and generalizes the construction given by Heunen et al. [2017] from countable index types (`I`) to arbitrary index types.

Our coproduct construction builds on a type `Var` (Figure 5). Intuitively, an element ψ : `Var P` is an "intensional" representation of a random variable $\varphi : \mathbb{R} \to$ `(x : I) × P x`. This is witnessed by a `CoeFun` instance providing a conversion from `Var P` to $\mathbb{R} \to$ `(x : I) × P x`. The index returned by φ is given by `embed ∘ index`, hence φ assigns each element of $\mathbb{R}$ to some element of a countable subset of `I`. This essentially "internalizes" the countability requirement on `I` in the original construction by Heunen et al. [2017]: arbitrary (non-countable) index types are allowed because the definition of `Var` ensures only a countable subset of `I` is used.

In addition to the `QuasiBorelSpace` instances, we also provide `IsHom` lemmas for all of the basic operations of the product and coproduct types, i.e., their constructors (e.g., `isHom_mk`) and eliminators (e.g., `isHom_elim`). We mark these lemmas with the `fun_prop` attribute to enable automatic reasoning about morphism properties, a pattern we adopt consistently throughout our formalization.

2.4 Exponentials

The central feature of quasi-Borel spaces is that they form a *cartesian-closed category*, i.e., given two quasi-Borel spaces `A` and `B`, we can define a quasi-Borel space structure for morphisms from `A` to `B`. We define a type for such functions, `QuasiBorelHom A B` (equivalently `A →q B`) in Figure 7. We endow `A →q B` with a `FunLike` instance to allow elements of `A →q B` to be invoked like normal functions, and finally construct a `QuasiBorelSpace` instance for `A →q B`. Heunen et al. [2017] mention in their proof notes that the coproduct construction (Section 2.3) is useful for the proof of `isVar_cases`; we found that the coproduct construction was not necessary.

As with products and coproducts, we provide `IsHom` proofs for `QuasiBorelHom`'s construction and projection functions (`isHom_mk` and `isHom_eval`), and mark these proofs with `fun_prop`. The `QuasiBorelHom.property` field is assigned a default value of `by fun_prop`, which enables very succinct constructions of complicated morphisms. For example, the `curry` function for `QuasiBorelHom`s is simply defined as follows:

```
def curry (f : A × B →q C) : A →q B →q C :=
  .mk fun x ↦ .mk fun y ↦ f (x, y)
```

2.5 Probability Measures

Definition 3 (Heunen et al. [2017]). *A* pre-probability measure *on a quasi-Borel space* (X, M_X) *is a pair* (ϕ, μ) *of* $\phi \in M_X$ *and a probability measure* μ *on* $\mathbb{R}$. *A* probability measure *on a*

```
instance : QuasiBorelSpace ((i : I) × P i) where
  IsVar φ := ∃ ψ : Var P, ∀r, φ r = ψ r
  ...
lemma Sigma.isHom_mk (i) : IsHom (⟨i, ·⟩ : P i → Sigma P)
lemma Sigma.isHom_elim {f : (i : I) × P i → A}
  (hf : ∀ i, IsHom (fun x ↦ f ⟨i, x⟩)) : IsHom f
```

Fig. 5. Coproducts.

```
structure Var (P : I → Type*) [∀ i, QuasiBorelSpace (P i)] where
  embed : ℕ → I
  index : ℝ → ℕ
  var : (i : ℕ) → ℝ → P (embed i)
  isHom_var : ∀i, IsHom (var i)
  measurable_index : Measurable[_, ⊤] index
instance : CoeFun (Var P) (fun _ ↦ ℝ → (x : I) × P x) where
  coe φ r := ⟨φ.embed (φ.index r), φ.var (φ.index r) r⟩
```

Fig. 6. Coproduct Variables.

quasi-Borel space (X, M_X) *is an element of the quotient* $P(X) = \{(\phi, \mu)$ probability measure on $(X, M_X)\}/\sim$, *where*

$$(\phi, \mu) \sim (\phi', \mu') \iff \int (f \circ \phi)\, \mathrm{d}\mu = \int (f \circ \phi')\, \mathrm{d}\mu$$

for all morphisms $f : (X, M_X) \to \mathbb{R}$.

We formalize Definition 3 in Figure 8. Intuitively, a probability measure in a quasi-Borel space is just a push-forward of some (standard) probability measure on $\mathbb{R}$ along some morphism. Accordingly, the definition of integral for (pre-)probability measures (also Figure 8) reduces to the measure-theoretic integral on $\mathbb{R}$.

Definition 4 (Heunen et al. [2017]). *The quasi-Borel space structure for probability measures is defined by*

$$M_{P(X)} = \{\, \beta : \mathbb{R} \to P(X) \mid \exists \phi \in M_X.\, \exists g : \mathbb{R} \to_{\mathbf{q}} G(\mathbb{R}).\, \forall r \in \mathbb{R}.\, \beta(r) = [\phi, g(r)] \,\}$$

where $G(\mathbb{R})$ *is the set of (measure-theoretic) probability measures on* $\mathbb{R}$, *and* $[\phi, g(r)]$ *denotes the equivalence class of* $(\phi, g(r))$.

We formalize Definition 4 in Figure 9. For convenience, we bundle all the existentially quantified variables of Definition 4 in a `structure` (similarly to coproducts in Section 2.3).

Finally, we prove that `ProbabilityMeasure` forms a strong commutative monad, i.e., we define

```
noncomputable def unit (x : A) : ProbabilityMeasure A
noncomputable def bind (f : A → B) (x : ProbabilityMeasure A) :
  ProbabilityMeasure B
```

such that the following lemmas hold:

```
/-- Morphism Lemmas -/
lemma isHom_unit : IsHom (unit (A := A))
lemma isHom_bind
    {f : C → A → ProbabilityMeasure B} (hf : IsHom (Function.uncurry f))
    {g : C → ProbabilityMeasure A} (hg : IsHom g)
    : IsHom (fun x ↦ bind (f x) (g x))
/-- Monad Lemmas -/
lemma bind_unit {f : A → ProbabilityMeasure B} (hf : IsHom f) (x : A) :
  bind f (unit x) = f x
lemma unit_bind (μ : ProbabilityMeasure A) :
  bind unit μ = μ
lemma bind_bind
    {f : B → ProbabilityMeasure C} (hf : IsHom f)
    {g : A → ProbabilityMeasure B} (hg : IsHom g)
    (μ : ProbabilityMeasure A)
```

```
structure QuasiBorelHom (A : Type*) [QuasiBorelSpace A]
                        (B : Type*) [QuasiBorelSpace B] where
  toFun : A → B
  property : IsHom toFun := by fun_prop
infixr:25 " →ᵠ " => QuasiBorelHom
instance : FunLike (A →ᵠ B) A B where
  coe := toFun
instance : QuasiBorelSpace (A →ᵠ B) where
  IsVar φ := IsHom (fun x : ℝ × A ↦ φ x.1 x.2)
  ...
lemma QuasiBorelHom.isHom_eval : IsHom (fun p : (A →ᵠ B) × A ↦ p.1 p.2)
lemma QuasiBorelHom.isHom_mk
    {f : A → B → C} (hf : IsHom (fun x : A × B ↦ f x.1 x.2)) :
    IsHom (fun x ↦ mk (f x) (by fun_prop))
attribute [fun_prop] QuasiBorelHom.isHom_eval QuasiBorelHom.isHom_mk
```

Fig. 7. Exponentials.

```
structure PreProbabilityMeasure (A : Type*) [QuasiBorelSpace A] where
  eval : ℝ →ᵠ A
  base : ProbabilityMeasure ℝ
noncomputable def PreProbabilityMeasure.lintegral (f : A → ENNReal) :
    PreProbabilityMeasure A → ENNReal
  | ⟨φ, μ⟩ => ∫⁻ x, f (φ x) ∂μ
instance PreProbabilityMeasure.setoid (A : Type*) [QuasiBorelSpace A] :
    Setoid (PreProbabilityMeasure A) where
  r μ₁ μ₂ := ∀{f}, IsHom f → μ₁.lintegral f = μ₂.lintegral f
  iseqv := ...
def ProbabilityMeasure (A : Type*) [QuasiBorelSpace A] :=
  Quotient (PreProbabilityMeasure.setoid A)
noncomputable def lintegral (k : A → ENNReal) (μ : ProbabilityMeasure A) : ENNReal :=
  μ.out.lintegral k
```

Fig. 8. Probability Measures.

```
    : bind f (bind g μ) = bind (fun x ↦ bind f (g x)) μ
/-- Commutativity Lemma -/
lemma bind_comm
  {f : A → B → ProbabilityMeasure C} (hf : IsHom (Function.uncurry f))
  (μ₁ : ProbabilityMeasure A) (μ₂ : ProbabilityMeasure B) :
  bind (fun x ↦ bind (f x) μ₂) μ₁ = bind (fun x ↦ bind (f · x) μ₁) μ₂
```

We prove each lemma by proving the analogous lemma for `PreProbabilityMeasure` and following the proof notes given by Heunen et al. [2017]. There is one exception: we did not find that `isHom_bind` was provable by simply "expanding the definitions" ([Heunen et al., 2017, §V-D]). Instead, we follow the approach taken by Hirata et al. [2022]. First, we prove the following weaker version of `isHom_bind`':

```
lemma isHom_bind' {f : A → ProbabilityMeasure B} (hf : IsHom f) :
  IsHom (bind f)
```

Then we explicitly construct the monadic strength operator:

```
noncomputable def str (x : A) (μ : ProbabilityMeasure B) :
  ProbabilityMeasure (A × B)
lemma isHom_str : IsHom (Function.uncurry (str (A := A) (B := B)))
```

The proof of `isHom_bind` then follows from

```
bind (f x) m = bind (fun x : (B →ᵠ ProbabilityMeasure C) × A ↦ x.1 x.2)
  (str ⟨f x, hf x⟩ m)
```

for all `f : A → B → ProbabilityMeasure C`, `m : ProbabilityMeasure B`, and `hf : ∀x, IsHom (f x)`.

3 Extended Formalization

In this section, we demonstrate the utility of the core formalization presented in Section 2.1 by constructing `QuasiBorelSpace` instances for several important types not discussed in the original

```
structure Var (A : Type*) [QuasiBorelSpace A] where
  eval : ℝ →ₐ A
  base : ℝ → ProbabilityMeasure ℝ
  measurable_base : Measurable base := by fun_prop
instance : QuasiBorelSpace (PreProbabilityMeasure A) where
  IsVar φ := ∃ ψ : Var A, ∀r, φ r ≈ ψ r
  ...
instance : QuasiBorelSpace (ProbabilityMeasure A) := lift fun x ↦ x.out
```

Fig. 9. `QuasiBorelSpace` instance for probability measures.

```
def lift {X} (f : X → A) : QuasiBorelSpace X where
  IsVar φ := IsVar fun x ↦ f (φ x)
  ...
lemma isHom_of_lift {X} (f : X → A) : IsHom[lift f, _] f
lemma isHom_to_lift {X} (f : X → A) (g : B → X)
  : IsHom[_, lift f] g ↔ IsHom (fun x ↦ f (g x))
```

Fig. 10. Lifting `QuasiBorelSpace` instances.

work on quasi-Borel spaces [Heunen et al., 2017]. For each type, we prove that its core operations (i.e., constructors and eliminators) are morphisms.

3.1 Exploiting Existing Instances

Manually constructing `QuasiBorelSpace` instances and `IsHom` proofs for more complicated types and functions quickly becomes tedious. An alternative is to build `QuasiBorelSpace` instances from existing instances, which we enable by providing the function `lift` (Figure 10). Additional `IsHom` proofs for `lift`-based instances are derived using the lemmas `isHom_of_lift` and `isHom_to_lift`.

We illustrate the case of binary coproducts (`A ⊕ B`). First, we construct its `QuasiBorelSpace` instance via the standard injection into `(b : Bool) × (if b then A else B)`:

```
def encode : A ⊕ B → (b : Bool) × (if b then A else B)
  | .inl x => ⟨true, x⟩
  | .inr x => ⟨false, x⟩
instance : QuasiBorelSpace (A ⊕ B) := lift encode
```

The `encode` function is automatically a morphism via `isHom_of_lift`:

```
lemma isHom_encode : IsHom (encode (A := A) (B := B)) := isHom_of_lift encode
```

The next step is to show that the constructors (`Sum.inl`, `Sum.inr`) and eliminators (`Sum.elim`) are morphisms. We show only the case for `Sum.inl`. Our goal is:

```
lemma isHom_inl : IsHom (Sum.inl : A → A ⊕ B) := ...
```

By `isHom_to_lift`, this reduces to `IsHom (encode ∘ Sum.inl)`, which is equivalent (by definition) to `IsHom ⟨true, ·⟩`. The proof is completed by `Sigma.isHom_mk`.

In our formalization, we construct many `QuasiBorelSpace` instances for new types by using `lift` with an injection into an existing type with a `QuasiBorelSpace` instance. Table 1 lists the types with instances constructed in this manner, along with the codomain of the associated injection.

3.2 Quotients

Our formalization includes a `QuasiBorelSpace` instance for quotients, given in Figure 11, which we also use to construct the `QuasiBorelSpace` instance for multi-sets (Section 3.1).

4 Comparison to the Isabelle Formalization

Hirata et al. [2022] develop a formalization of quasi-Borel spaces in Isabelle. We briefly list the differences between their formalization and ours:

Type	Target Type
`Subtype A`	`A`
`A ⊕ B`	`(b : Bool) × (if b then A else B)`
`Option A`	`Unit ⊕ A`
`List A`	`(n : ℕ) × (Fin n → A)`
`Multiset A`	`Quotient (List.isSetoid A)`
`Finset A`	`{ xs : Multiset A // Multiset.Nodup xs }`

Table 1. Types with `QuasiBorelSpace` instances constructed using `lift`.

```
instance [QuasiBorelSpace A] : QuasiBorelSpace (Quotient S) where
  IsVar φ := ∃ ψ : ℝ → A, IsHom ψ ∧ ∀r, φ r = ⟦ψ r⟧
  ...
lemma Quotient.isHom_mk : IsHom (fun x ↦ (⟦x⟧ : Quotient S))
lemma Quotient.isHom_lift {f : A → B} (hf₁ : IsHom f)
    (hf₂ : ∀ x y, x ≈ y → f x = f y) : IsHom (Quotient.lift f hf₂ : Quotient S → B)
attribute [fun_prop] Quotient.isHom_mk Quotient.isHom_lift
```

Fig. 11. Quotients.

1. We give a construction for all coproducts (Section 2.3), whereas Hirata et al. [2022] only construct countable coproducts.
2. Our formalization is in Lean 4, a dependently typed proof assistant, whereas Hirata et al. [2022] use Isabelle/HOL, which is based on higher-order logic with simple types. A key difference is that all types in Isabelle/HOL are inhabited: this is not the case in a dependently typed proof assistant, and results in more complicated constructions (e.g. see Section 2.3).
3. We give quasi-Borel space constructions for many additional types beyond what is described by Heunen et al. [2017] or formalized by Hirata et al. [2022] (Section 3.1).
4. Hirata et al. [2022] formalize Bayesian linear regression. This is absent from our formalization.

5 Conclusion and Future Work

We have presented our formalization of quasi-Borel spaces in Lean. Going beyond the core constructions given by Heunen et al. [2017], we also show how to construct quasi-Borel space structures for many common types. In some cases, our formalization effort led to simpler and more general constructions. We have presented only a small portion of our full development, which totals over 8000 lines of Lean code. The source code for our full developments can be found at `https://github.com/YellPika/quasi-borel-spaces`.

Our formalization of quasi-Borel spaces is intended as a step towards formalizing the semantics of higher-order probabilistic programs within Lean. Thus, future work includes the following tasks:

- Construct a `QuasiBorelSpace` instance for `WTypes`, which paves the way for automatically constructing `QuasiBorelSpace` instances for arbitrary inductive types.
- Formalize the basic constructions and properties of *quasi-Borel pre-domains* [Vákár et al., 2019], which combine quasi-Borel spaces with domain theory [Abramsky and Jung, 1995] and provide a foundation for reasoning about probabilistic programming languages with unrestricted recursion. We have formalized all the basic constructions (morphisms, products, coproducts, and exponentials). It remains to formalize the powerdomain construction given by Vákár et al. [2019], which is necessary to model probabilistic computation with general recursion.

Bibliography

Samson Abramsky and Achim Jung. *Domain Theory*, pages 1–168. Oxford University Press, Oxford, 1995. ISBN 9781383026108. https://doi.org/10.1093/oso/9780198537625.003.0001. URL http://dx.doi.org/10.1093/oso/9780198537625.003.0001.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, page 625–635, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-79875-8. https://doi.org/10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017. https://doi.org/10.1109/lics.2017.8005137. URL http://dx.doi.org/10.1109/LICS.2017.8005137.

Michikazu Hirata, Yasuhiko Minamide, and Tetsuya Sato. Quasi-borel spaces. *Archive of Formal Proofs*, February 2022. ISSN 2150-914x. https://isa-afp.org/entries/Quasi_Borel_Spaces.html, Formal proof development.

Olav Kallenberg. *Foundations of Modern Probability*. Springer International Publishing, 2021. ISBN 9783030618711. https://doi.org/10.1007/978-3-030-61871-1. URL http://dx.doi.org/10.1007/978-3-030-61871-1.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New Orleans, LA, USA, January 2020. ACM. https://doi.org/10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019. ISSN 2475-1421. https://doi.org/10.1145/3290349. URL http://dx.doi.org/10.1145/3290349.