Semantic Foundations of Higher-Order Probabilistic Programs in Isabelle/HOL

Michikazu Hirata
School of Computing, Tokyo Institute of Technology, Japan

Yasuhiko Minamide
School of Computing, Tokyo Institute of Technology, Japan

Tetsuya Sato
School of Computing, Tokyo Institute of Technology, Japan

Abstract

Higher-order probabilistic programs are used to describe statistical models and machine-learning mechanisms. The programming languages for them are equipped with three features: higher-order functions, sampling, and conditioning. In this paper, we propose an Isabelle/HOL library for probabilistic programs supporting all of those three features. We extend our previous quasi-Borel theory library in Isabelle/HOL. As a basis of the theory, we formalize *s-finite kernels*, which is considered as a theoretical foundation of first-order probabilistic programs and a key to support conditioning of probabilistic programs. We also formalize the Borel isomorphism theorem which plays an important role in the quasi-Borel theory. Using them, we develop the *s-finite measure monad* on quasi-Borel spaces. Our extension enables us to describe higher-order probabilistic programs with conditioning directly as an Isabelle/HOL term whose type is that of morphisms between quasi-Borel spaces. We also implement the *qbs prover* for checking well-typedness of an Isabelle/HOL term as a morphism between quasi-Borel spaces. We demonstrate several verification examples of higher-order probabilistic programs with conditioning.

2012 ACM Subject Classification Theory of computation → Denotational semantics; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Higher-order probabilistic program, s-finite kernel, Quasi-Borel spaces, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.18

Funding *Michikazu Hirata*: supported by JSPS Research Fellowships for Young Scientists and JSPS KAKENHI Grant Number 23KJ0905
Yasuhiko Minamide: supported by JSPS KAKENHI Grant Number 19K11899 and 20H04162
Tetsuya Sato: supported by JSPS KAKENHI Grant Number 20K19775

1 Introduction

Probabilistic programs are used to describe statistical models and machine-learning mechanisms. Programmers can conduct statistical inference just by writing statistical models as programs, without implementing complex inference algorithms by themselves. Higher-order probablistic programming languages, e.g. Anglican [29] and Church [9], integrate fundamental features of probabilistic programming languages such as sampling and conditioning into expressive higher-order funcaional languages and have been an active research topic recently.

Let us see a concrete example by Staton [25, Section 2.2]. The following probabilistic program uses two language features: higher-order functions and conditioning.

```
1.  λf. do {
2.    let T = uniform(0,24) in
3.    query T (λt. let r = f t in
4.                 exponential_pdf r 0.0167)
5.    }
```

This program is higher-order since `f` is given as a parameter. The `query` command at line 3 receives a prior distribution and a likelihood, and then returns the posterior distribution. We will explain details of this program in Section 4.2.

Basically, probabilistic programs are interpreted as measurable functions between measurable spaces. Various measure-theoretic structures such as the Giry monad [8] and s-finite kernels [24] are used for such semantic models. However, there is a difficulty to interpret higher-order functions. The result by Aumann [2] implies that there is no suitable measurable space corresponding to the function type `real` ⇒ `real`. In order to overcome this difficulty, Heunen et al. have introduced quasi-Borel spaces and the probability monad on it [10]. The theory provides a suitable denotational semantics for higher-order probabilistic programs. Ścibior et al. have developed the s-finite measure monad[1] on quasi-Borel spaces [22], which enable us to treat infinite measures and to denote higher-order probabilistic programs with conditioning.

In previous work, we have formalized the quasi-Borel spaces and the probability monad in Isabelle/HOL [11]. Using them, we have verified the Monte Carlo approximation algorithm. Our previous work can treat probabilistic programs supporting higher-order functions and sampling but not conditioning. Affeldt et al. have formalized s-finite kernels in Coq [1]. They have embedded a probabilistic program using s-finite kernels. Their work can treat probabilistic programs supporting sampling and conditioning but not higher-order functions.

In this paper, we propose an Isabelle/HOL library for probabilistic programs supporting all of higher-order functions, sampling, and conditioning by extending our previous work. Our contributions are the following.

1. We formalize s-finite kernels and the Borel isomorphism theorem. They are a theoretical basis of quasi-Borel theory, especially a basis of the s-finite measure monad.
2. We develop proof automation for checking *well-typedness* of probabilistic programs and construct the s-finite measure monad.
3. We implement several program examples from previous works and prove their properties.

Our library enables us to interpret an Isabelle/HOL term as a probabilistic program and that makes it easier to write probabilistic programs and reason about them in Isabelle/HOL. Our *qbs prover* for automated *type checking* is also helpful to reason about probabilistic programs. Both of previous formalizations by us [11] and Affeldt et al. [1] use de Bruijn index to describe programs, which makes it harder to read and write programs. Our previous work spent around 450 lines to prove integrability and the weak law of large numbers of the Monte Carlo approximation algorithm, while we have spent around 140 lines to prove them in our new formalization.

In Section 2, we review the standard library for measure theory in Isabelle/HOL. Then we formalize s-finite kernels and the Borel isomorphism theorem. In Section 3, we review our previous formalization of quasi-Borel spaces. Then we discuss our proof automation and formalization of the s-finite measure monad. In Section 4, we show three verification examples of probabilistic programs. In Section 5, we conclude our work.

[1] Details of definition vary among prior studies. In their original paper, they have introduced the σ-finite measure monad. Later Vákár et al. reformulate it as s-finite measure monad [27, 28](see also Section 3.3).

2 Measure Theory

Measure theory is a theoretical basis of probability theory and quasi-Borel theory. We first review the standard definitions of measure theory library in Isabelle/HOL. Then we formalize s-finite kernels which are used to construct the s-finite measure monad in Section 3.3. We also formalize the Borel isomorphism theorem, which plays an important role in quasi-Borel theory.

2.1 Measure Theory in Isabelle/HOL

We use the Isabelle/HOL's libraries: HOL-Analysis and HOL-Probability [3, 7, 12, 13, 16, 18]. The type *'a measure* denotes the type of measures on the type *'a*. A measure $M :: 'a\ measure$ consists of three components:

$$space\ M :: 'a\ set, \qquad sets\ M :: 'a\ set\ set, \qquad emeasure\ M :: 'a\ set \Rightarrow ennreal,$$

where the type *ennreal* denotes the type of extended non-negative real numbers. They correspond to the space, the measurable sets, and the measure, respectively. We often write M for *emeasure M* using a coercion. The triple $(space\ M, sets\ M, emeasure\ M)$ forms a measure space, that is, *sets M* is a σ-algebra on *space M* and *emeasure M* is a countably additive function on *sets M* such that $emeasure\ M\ \emptyset = 0$. We use a measure M as a measurable space when we are not interested in its measure. The library defines the constant $borel :: ('a :: topological\text{-}space)\ measure$ on topological space type class. The *borel* denotes the Borel space, that is, $sets\ borel = \sigma[\{U.\ open\ U\}]$ is the least measurable sets including all open sets. We denote the borel space on real numbers by $\mathbb{R}$, the borel space on extended non-negative real numbers by $\mathbb{R}_{\geq 0}$, the discrete space on natural numbers by $\mathbb{N}$, and the discrete space on boolean by $\mathbb{B}$.

A function f from *space M* to *space N* is called measurable if $f^{-1}\ A \cap space\ M \in sets\ M$ for all $A \in sets\ N$. The set of measurable functions from M to N is denoted by $M \to_M N$. For a measurable function $f \in M \to_M \mathbb{R}$, the Lebesgue integral of f w.r.t. M is denoted by[2] $\int x.\ f\ x\ \partial M$. The Lebesgue integral on a restricted set A is denoted by $\int x{\in}A.\ f\ x\ \partial M$.

For a measure M, the predicate *subprob-space M* means that M is a sub-probability space, that is, $M\ (space\ M) \leq 1$. The predicate *finite-measure M* means that M is a finite measure, that is, $M\ (space\ M) < \infty$. The predicate *sigma-finite-measure M* means that M is a σ-finite measure, that is, there exists a countable disjoint measurable sets $\forall i{::}nat.\ A\ i \in sets\ M$ such that $(\bigcup i.\ A\ i) = space\ M$ and $\forall i.\ M\ (A\ i) < \infty$.

Throughout this paper, we use the following constructions.

The Lebesgue Measure[3]	$sets\ lborel = \sigma[\{U.\ open\ U\}]$
	$emeasure\ lborel\ (a, b] = b - a$
Product Measure	$sets\ (M \otimes_M N) = \sigma[\{A \times B.\ A \in sets\ M \wedge B \in sets\ N\}]$
	$emeasure\ (M \otimes_M N)\ (A \times B) = emeasure\ M\ A * emeasure\ N\ B$
	where *sigma-finite-measure N*, $A \in sets\ M$, and $B \in sets\ N$.
Image Measure	$sets\ (distr\ M\ N\ f) = sets\ N$
	$emeasure\ (distr\ M\ N\ f)\ A = emeasure\ (f^{-1}\ A \cap space\ M)$
	where $f \in M \to_M N$ and $A \in sets\ N$.

[2] In the library, the real-valued integral and the extended non-negative real-valued integral are defined separately. Although we do not distinguish them for simplicity in this paper.

[3] Strictly speaking, *completion lborel* is the Lebesgue measure.

2.2 S-Finite Kernel

Staton introduced a semantic model of first-order probabilistic programs with conditioning using s-finite kernels [24]. S-finite kernels are suitable for program semantics: they support a bind-like operation satisfying desired equations, which are a basis of the s-finite measure monad on quasi-Borel spaces. We formalize s-finite kernels and related notions. For the terminology of s-finite measures/kernels, we refer to the work by Staton [24].

S-finite Measures. A measure M is called a *s-finite measure* if M is represented as a countable sum of finite measures. All σ-finite measures, such as the Lebesgue measure, are also s-finite measures. We formalize s-finite measures with the **locale** command which introduces a context.

locale *s-finite-measure* =
 fixes M :: *'a measure*
 assumes $\exists\, Mi :: nat \Rightarrow$ *'a measure*.
 $(\forall\, i.\ sets\ (Mi\ i) = sets\ M) \wedge (\forall\, i.\ finite\text{-}measure\ (Mi\ i))$
 $\wedge\ (\forall\, A{\in}sets\ M.\ M\ A = (\sum i.\ Mi\ i\ A))$

sublocale *sigma-finite-measure* $\subseteq$ *s-finite-measure*

The symbol $\sum i.$ sums over all natural numbers (i.e. $\sum_{i=0}^{\infty}$ in usual mathematics). We remark that s-finite measures may not be σ-finite in general. For instance, the measure $M\ \{0\} = \infty$ on the singleton space $\{0\}$ is not σ-finite but s-finite, because it is equal to the countable infinite sum of the Dirac measure δ_0.

We have formalized basic lemmas related to s-finite measures. One of the important lemma is a restricted Fubini-Tonelli theorem for reordering iterated integrations. The general Fubini-Tonelli theorem does not hold for s-finite measures because product measures are not determined uniquely. However, the (binary) product measures in Isabelle/HOL work well with the Fubini-Tonelli theorem. In mathematics, the product measure is usually defined as the unique measure satisfying $(M \otimes_M N)\ (A \times B) = M\ A * N\ B$, while Isabelle/HOL's library defines the product measure as $(M \bigotimes_M N)\ A = (\int x.\ (\int y.\ indicator\ A\ (x,y)\ \partial N)\ \partial M)$. Using Isabelle/HOL's definition, we can prove Fubini-Tonelli theorem by almost similar ways as the proofs for σ-finite measures.

S-finite Kernels. Roughly speaking, s-finite kernels are generalization of probabilistic processes that return s-finite measures. They are defined as countable sums of finite kernels. In general, classes of kernels are not closed under compositions, but it is convenient that s-finite kernels are so. We first formalize *measure kernels* with the **locale** command.

locale *measure-kernel* =
 fixes M :: *'a measure*
 and N :: *'b measure*
 and κ :: *'a* $\Rightarrow$ *'b measure*
 assumes $\bigwedge x.\ x \in space\ M \Longrightarrow sets\ (\kappa\ x) = sets\ N$
 and $\bigwedge B.\ B \in sets\ N \Longrightarrow (\lambda x.\ \kappa\ x\ B) \in M \to_M \mathbb{R}_{\geq 0}$
 and $space\ M \neq \emptyset \Longrightarrow space\ N \neq \emptyset$

The third assumption $space\ M \neq \emptyset \Longrightarrow space\ N \neq \emptyset$ in *measure-kernel* is required in order to define the operator $\gg\!=_k$ in a convenient way, later. We formalize *finite kernels*, *sub-probability kernels*, and *s-finite kernels* as sublocales of measure kernels.

locale *finite-kernel* = *measure-kernel* +
 assumes $\exists\, r{<}\infty.\ \forall\, x{\in}\ space\ M.\ \kappa\ x\ (space\ N) < r$

locale *subprob-kernel* = *measure-kernel* +
  assumes $\bigwedge x.\ x \in space\ M \Longrightarrow subprob\text{-}space\ (\kappa\ x)$

locale *s-finite-kernel* = *measure-kernel* +
  assumes $\exists\, ki.\ (\forall\, i.\ finite\text{-}kernel\ M\ N\ (ki\ i)) \wedge (\forall\, x{\in}space\ M.\ \forall\, A{\in}sets\ N.\ \kappa\ x\ A = (\sum i.\ ki\ i\ x\ A)))$

We define the operation $M \gg\!=_k \kappa$ for an s-finite measure M and *measure-kernel* $M\ N\ \kappa$, which satisfies the following properties when M is not an empty space.

$$sets\ (M \gg\!=_k \kappa) = sets\ N, \quad (M \gg\!=_k \kappa)\ B = (\int x.\ (\kappa\ x\ B)\ \partial M)$$

If M is an empty space, we cannot obtain the measurable structure of N from M and κ (recall the definition of *measure-kernel*). Hence, $M \gg\!=_k \kappa$ is set to return the discrete empty space as a *default value*. Due to this definition, we need the assumption *space* $M \neq \emptyset \Longrightarrow$ *space* $N \neq \emptyset$ in *measure-kernel*. Without this assumption, we will get stuck to prove *compositionality* of s-finite kernels later.

The operation *bind*, which has been already defined in the Isabelle/HOL's library, satisfies the same equations as the above equation for $\gg\!=_k$ when κ is a sub-probability kernel. Unfortunately, *bind* is defined through the *join* operator of the Giry monad and thus we do not have the above equations for general measure kernels. Hence we need to introduce the operator $\gg\!=_k$ and prove lemmas similar to ones of *bind*.

The following are important properties for constructing the s-finite measure monad in Section 3.3 (called compositionality, associativity, and commutativity, respectively).

lemma
  assumes *s-finite-kernel* $M\ N\ \kappa$ **and** *s-finite-kernel* $(M \otimes_M N)\ L\ (\lambda(x, y).\ \kappa'\ x\ y)$
  shows *s-finite-kernel* $M\ L\ (\lambda x.\ \kappa\ x \gg\!=_k \kappa'\ x)$

lemma
  assumes $sets\ \mu = sets\ M$
    and *s-finite-kernel* $M\ N\ \kappa$ **and** *s-finite-kernel* $N\ L\ \kappa'$
  shows $\mu \gg\!=_k (\lambda x.\ \kappa\ x \gg\!=_k \kappa') = \mu \gg\!=_k \kappa \gg\!=_k \kappa'$

lemma
  assumes $sets\ \mu = sets\ M$ **and** $sets\ \nu = sets\ N$
    and *s-finite-measure* μ **and** *s-finite-measure* ν **and** *s-finite-kernel* $(M \bigotimes_M N)\ L\ (\lambda(x,y).\ f\ x\ y)$
  shows $\mu \gg\!=_k (\lambda x.\ \nu \gg\!=_k (\lambda y.\ f\ x\ y)) = \nu \gg\!=_k (\lambda y.\ \mu \gg\!=_k (\lambda x.\ f\ x\ y))$

The Dirac measure on M is denoted by *return* M in Isabelle/HOL. It forms a unit of $\gg\!=_k$.

lemma
  assumes $sets\ M = sets\ N$
  shows $M \gg\!=_k return\ N = M$

lemma
  assumes *measure-kernel* $M\ N\ \kappa$
    and $x \in space\ M$
  shows $return\ M\ x \gg\!=_k \kappa = \kappa\ x$

2.3 The Borel Isomorphism Theorem

We prove the Borel isomorphism theorem. The theorem is a key to construct the *s-finite measure monad* and represent s-finite measures as *measures* on quasi-Borel spaces in Section 3.

A separable complete metrizable topological space is called a Polish space. A measurable space generated from a Polish space is called a standard Borel space. For example, $\mathbb{N}$ and $\mathbb{R}$ are standard Borel spaces. We have the following theorems related to standard Borel spaces.

▶ **Theorem 1** (The Borel isomorphism theorem). *A standard Borel space is either countable discrete space or isomorphic to* $\mathbb{R}$.

▶ **Corollary 2.** *For a non-empty standard Borel space* M*, the following statement holds.* ($\star$) *There exist measurable functions* $\textit{to-real}_M$ *and* $\textit{from-real}_M$ *such that*

$$\textit{to-real}_M \in M \to_M \mathbb{R}\ , \quad \textit{from-real}_M \in \mathbb{R} \to_M M\ ,$$
$$\forall x{\in}\textit{space}\ M.\ \textit{from-real}_M\ (\textit{to-real}_M\ x) = x\ .$$

We have proved the Borel isomorphism (Theorem 1) mainly referring to the textbook by Srivastava [23] and the lecture note by Biskup [6], which was available online. Corollary 2 follows immediately from the Borel isomorphism theorem[4]. We will use measurable functions in ($\star$) in two situations. One is when we construct the s-finite measure monad on quasi-Borel spaces. The other is when we represent s-finite measures as *measures* on quasi-Borel spaces. In our previous work [11], we defined the standard Borel spaces as measurable spaces which satisfy the condition ($\star$). An advantage of our new formalization is that we can obtain many instances easily with the type classes using the following lemma.

lemma *standard-borel* (*borel* :: ('*a* :: *polish-space*) *measure*)

Here, *standard-borel* M means that M is a standard Borel space. A binary or countable product space of standard Borel spaces is again a standard Borel space.

lemma
  assumes *standard-borel* M **and** *standard-borel* N
  shows *standard-borel* ($M \otimes_M N$)
lemma
  assumes *countable* I **and** $\bigwedge i.\ i \in I \Longrightarrow$ *standard-borel* ($M\ i$)
    shows *standard-borel* ($\Pi_M\ i{\in}I.\ M\ i$)

In the proof of the Borel isomorphism theorem we use metric spaces and topological spaces. The Isabelle/HOL's libraries include formalization of metric spaces by type classes, and topological spaces by type classes and abstract data types. Type class based formalization is not suitable in our situation because we want to change their metrics or topologies during the proof and work with sub-spaces. Thus we have formalized set-based metric spaces and used the existing library of abstract topology with some extensions. Recent work on *types-to-sets* [15, 17, 19] might be used to simplify our formalization.

3 Quasi-Borel Spaces

The theory of quasi-Borel spaces is introduced by Heunen et al. [10] to give a semantic model of programming language supporting both continuous random samplings and higher-order functions. The theory provides a suitable semantics of higher-order probabilistic programs because quasi-Borel spaces always have function spaces with desired properties while measurable spaces do not in general. Furthermore, s-finite measures on standard Borel spaces are represented as measures on quasi-Borel spaces and integration is also performed in quasi-Borel theory.

We formalized the quasi-Borel spaces and the probability monad in our previous work [11]. In this section, we first review our previous formalization, then discuss our extensions: proof automation for quasi-Borel spaces and formalization of the s-finite measure monad.

[4] In fact, the converse also holds: a measurable space satisfying ($\star$) is a standard Borel space. Hence the condition ($\star$) is another characterization of standard Borel spaces. This fact is called as Kuratowski's theorem by Heunen et al. [10]. We have not proved it yet.

3.1 Quasi-Borel Spaces in Isabelle/HOL

The type *'a quasi-borel* denotes quasi-Borel spaces on the type *'a*. A quasi-Borel space $X :: 'a$ *quasi-borel* has two components:

$$\textit{qbs-space}\ X :: 'a\ \textit{set}, \qquad \textit{qbs-Mx}\ X :: (\textit{real} \Rightarrow 'a)\ \textit{set}.$$

They satisfy the following four conditions.

- If $\alpha \in \textit{qbs-Mx}\ X$ and r is a real number, then $\alpha\ r \in \textit{qbs-space}\ X$.
- If $\alpha \in \textit{qbs-Mx}\ X$ and $f \in \mathbb{R} \to_M \mathbb{R}$, then $\alpha \circ f \in \textit{qbs-Mx}\ X$.
- If $x \in \textit{qbs-space}\ X$, then $(\lambda r.\ x) \in \textit{qbs-Mx}\ X$.
- If $(\forall i.\ \alpha\ i \in \textit{qbs-Mx}\ X)$ and $P \in \mathbb{R} \to_M \mathbb{N}$, then $(\lambda r.\ \alpha\ (P\ r)\ r) \in \textit{qbs-Mx}\ X$.

Intuitively, an element of *qbs-Mx X* is a *random variable* whose sample space is the set of real numbers. We sometimes write $x \in X$ instead of $x \in \textit{qbs-space}\ X$ by declaring a coercion.

The set of morphisms (structure-preserving functions) from $X :: 'a$ *quasi-borel* to $Y :: 'b$ *quasi-borel* is defined as follows.

$$\begin{array}{lcl} X \to_Q Y & :: & ('a \Rightarrow 'b)\ \textit{set} \\ X \to_Q Y & = & \{f.\ \forall \alpha \in \textit{qbs-Mx}\ X.\ f \circ \alpha \in \textit{qbs-Mx}\ Y\} \end{array}$$

Quasi-Borel spaces and morphisms form a Cartesian closed category with countable coproducts. Hence, there always exist product spaces $X \otimes_Q Y$, list spaces *list-qbs* X[5], and function spaces $X \Rightarrow_Q Y$ such that $\textit{qbs-space}\ (X \Rightarrow_Q Y) = X \to_Q Y$. Throughout this paper, we assume that all functions are morphisms. In our extension of quasi-Borel theory library, we define the set of morphisms $X \to_Q Y$ as an abbreviation of $\textit{qbs-space}\ (X \Rightarrow_Q Y)$ for the proof automation presented in Section 3.2.

Connection between Measurable Spaces and Quasi-Borel Spaces

There are conversions between measurable spaces and quasi-Borel spaces. Using the conversions, we can easily derive from theorems in the measure theory library that basic functions, such as $+$ and $-$, are morphisms. The conversions L and R return the following structures.

$$\begin{array}{ll} L :: 'a\ \textit{quasi-borel} \Rightarrow 'a\ \textit{measure} & R :: 'a\ \textit{measure} \Rightarrow 'a\ \textit{quasi-borel} \\ \textit{space}\ (L\ X) = \textit{qbs-space}\ X & \textit{qbs-space}\ (R\ M) = \textit{space}\ M \\ \textit{sets}\ (L\ X) = \{U \cap \textit{qbs-space}\ X \mid U. & \textit{qbs-Mx}\ (R\ M) = \mathbb{R} \to_M M \\ \quad \forall \alpha \in \textit{qbs-Mx}\ X.\ \alpha -` U \in \textit{sets}\ \mathbb{R}\} & \end{array}$$

We use a measurable space M as a quasi-Borel space $R\ M$. For instance, the quasi-Borel space $\mathbb{R}$ has the following structure: $\textit{qbs-space}\ \mathbb{R} = \textit{UNIV}$ (the universal set of real numbers) and $\textit{qbs-Mx}\ \mathbb{R} = \mathbb{R} \to_M \mathbb{R}$.

The conversions have the following properties.

▶ **Theorem 3** (cf. [10, Propositions 15]). **(i)** $X \to_Q R\ M = L\ X \to_M M$.
(ii) *If M is a standard borel space, then* $\textit{sets}\ (L\ (R\ M)) = \textit{sets}\ M$

Theorem 3 (i) implies that R and L forms an adjunction between the category of measurable spaces and the category of quasi-Borel spaces, and (ii) implies that the adjunction can be restricted to an adjoint equivalence on standard Borel spaces.

[5] The space of lists on X is defined using the isomorphism $\textit{List}[X] \cong \coprod_{n \in \mathbb{N}} \prod_{0 \le i < n} X$.

3.2 Proof Automation

The Isabelle/HOL's measure theory library provides the automated *measurability prover*. In the context of measure theory, one often needs to show measurability: $A \in \mathit{sets}\ M$ or $f \in M \to_M N$. In pen and paper mathematics, measurability proofs are often omitted since they are trivial, while one needs to show measurability each time in the formal proof. The measurability prover automates such proofs of measurability and greatly reduces the cost of proofs. Similar to measure theory, we often need to prove that some function is a morphism, $f \in X \to_Q Y$, in the context of quasi-Borel theory. We have implemented automated *qbs prover*. Unlike measurable spaces, quasi-Borel spaces have function spaces, hence our qbs prover is similar to type checking of a simply-typed functional programming language.

We construct the qbs prover which tries to prove $x \in \mathit{qbs\text{-}space}\ X$ automatically. The qbs prover can also be used to solve morphism statements $f \in X \to_Q Y$ and $\alpha \in \mathit{qbs\text{-}Mx}\ X$ because we have $X \to_Q Y = \mathit{qbs\text{-}space}\ (X \Rightarrow_Q Y)$ and $\mathit{qbs\text{-}Mx}\ X = \mathbb{R} \to_Q X$.

We regard $(\lambda x.\ e) \in X \Rightarrow_Q Y$ as the typing judgment $x : X \vdash e : Y$, and $e \in \mathit{qbs\text{-}space}\ X$ as $\vdash e : X$. Then solving $x \in \mathit{qbs\text{-}space}\ X$ is equivalent to solving the corresponding typing judgment. The qbs prover tries to solve typing judgments with the following method:

Algorithm. We prepare two sets of introduction rules: Rule_1 and Rule_2. Then repeat the following steps.

- Try to apply a rule in Rule_1.
- If none of the rules in Rule_1 is applied, then try to apply a rule in Rule_2.

Rule_1 and Rule_2 consist of (at least) the following inference rules.

- Rule_1

$$\frac{}{x : X \vdash x : X}\ \mathrm{ID} \qquad \frac{\vdash e : Y}{x : X \vdash e : Y}\ \mathrm{CONST}\ (x \text{ does not occur free in } e)$$

After $e \in \mathit{qbs\text{-}space}\ X$ is proved, it may be added as an axiom of Rule_1.

$$\frac{}{\vdash e : X}\ \mathrm{AXIOMS}$$

- Rule_2

$$\frac{\vdash f : X \Rightarrow_Q Y \qquad \vdash x : X}{\vdash f\ x : Y}\ \mathrm{APP}_1 \qquad \frac{x : X \vdash e_1 : Y \Rightarrow_Q Z \qquad x : X \vdash e_2 : Y}{x : X \vdash e_1\ e_2 : Z}\ \mathrm{APP}_2$$

$$\frac{z : X \otimes_Q Y \vdash f[\mathit{fst}\ z/x, \mathit{snd}\ z/y] : Z}{x : X \vdash (\lambda y.\ f) : Y \Rightarrow_Q Z}\ \mathrm{CURRY}$$

For CURRY, we need to have $\mathit{fst} \in X \otimes_Q Y \Rightarrow_Q X$ and $\mathit{snd} \in X \otimes_Q Y \Rightarrow_Q Y$ as axioms of Rule_1. There are mainly two reasons why we divide the rules. First, the rule CONST might overlap with APP_2 or CURRY. Because the rule CONST should be applied first, we add CONST to Rule_1. The other reason is that to prevent terms from being split in certain situation. We sometimes add rules for composition of terms, for example $\mathit{emeasure}\ M\ A \in \mathbb{R}_{\geq 0}$, to Rule_1. If we apply a rule in Rule_2 first, then the composed term will be split by the rule APP_1 or APP_2, that is not what we want the prover to do.

The following code is an example usage of the qbs prover.

```
lemma
  assumes [qbs]: f ∈ ℝ ⇒Q ℝ
  shows (λx. 1 + f x) ∈ ℝ ⇒Q ℝ
  by qbs
```

In the above code, we add $f \in \mathbb{R} \Rightarrow_Q \mathbb{R}$ to the axioms of Rule_1 using the attribute $[qbs]$. Rule_1 is configured by our library so that the axioms contain $r \in \mathbb{R}$ and $(+) \in \mathbb{R} \Rightarrow_Q \mathbb{R} \Rightarrow_Q \mathbb{R}$. Then we call the qbs prover by the tactic *qbs* which immediately solves the goal.

However, it cannot handle assumptions on typing of lambda abstraction well. It fails for the following example.

```
lemma
  assumes [qbs]: (λx. f x c) ∈ X ⇒Q Y
  shows (λx z. f x c) ∈ X ⇒Q Z ⇒Q Y
```

Implementation Note. We have implemented the qbs prover using raw ML code. There are some points to be noted.

- The following theorem corresponds to the rule APP_2 in Isabelle/HOL.
  ```
  lemma
    assumes f ∈ X ⇒Q Y ⇒Q Z and g ∈ X ⇒Q Y
    shows (λx. f x (g x)) ∈ X ⇒Q Z
  ```
  When applying the rule APP_2, we need to instantiate f and g in the lemma so that higher-order unification achieves an intended unification.
- When applying the rule CURRY, we should check by pattern matching that the goal is a lambda abstraction. Otherwise, it may overlap with APP_2 by eta-expanding $e_1\ e_2$ when the term has a function type.

We expect that this *typing* algorithm works in a similar situation where we want to restrict function spaces and constants in Isabelle/HOL. In our situation, function spaces are restricted to the set of morphisms.

3.3 The s-Finite Measure Monad

The s-finite measure monad on quasi-Borel spaces was introduced by Ścibior et al. [22] as the σ-finite measure monad. Then, it was reformulated as a submonad of the continuation monad $[0,\infty]^{[0,\infty]^{(-)}}$ by Vákár et al. [27, 28]. The details of the definition vary among these previous studies[6], and we could not find detailed proofs of monad laws and commutativity in any of them. We thus recover the detailed proofs first, and then we formalize them. We choose the definition given in Yang's lecture slide [30], because it is suitable for formalization in Isabelle/HOL. Its definition is quite similar to the probability monad introduced by Heunen et al. [10]. The probability monad is derived from the monad laws and the commutativity of the Giry monad, while the s-finite measure monad is derived from the properties of s-finite kernels and $\gg\!=_k$.

First, we define measures on quasi-Borel spaces to treat infinite measures such as the Lebesgue measure. Intuitively, a measure is a pair consisting of an s-finite measure μ on $\mathbb{R}$ and a random variable $\alpha \in qbs\text{-}Mx\ X$. We also introduce the equivalence relation $\sim$ of measures on quasi-Borel spaces defined by relating pairs with equal image measures. In our implementation, we use a triple (X,α,μ) rather than a pair (α,μ) because X cannot be inferred from α in simple type system.

[6] Thanks to the Borel-isomorphism theorem and the fact that s-finite measures can be rewritten as pushforward of σ-finite measures, those definitions are essentially equivalent.

▶ **Definition 4.** *A measure on a quasi-Borel space X is an equivalence class $[\![X, \alpha, \mu]\!]_{\mathsf{sfin}}$ where $\alpha \in$ qbs-Mx X, μ is a s-finite measure, and sets $\mu =$ sets $\mathbb{R}$. The equivalence relation is defined by $(X, \alpha, \mu) \sim (Y, \beta, \nu) \iff X = Y \wedge$ distr $(L\ X)\ \mu\ \alpha =$ distr $(L\ Y)\ \nu\ \beta$.*

We call a measure on a quasi-Borel space a *qbs-measure* in order to distinguish it from measures in measure theory. Using **quotient-type** command [14], we define the type *'a qbs-measure* which denotes the type of qbs-measures.

Any qbs-measure can be converted to an s-finite measure by the following function.

$$\begin{aligned} \textit{qbs-l} &:: \textit{'a qbs-measure} \Rightarrow \textit{'a measure} \\ \textit{qbs-l}\ [\![X, \alpha, \mu]\!]_{\mathsf{sfin}} &= \textit{distr}\ (L\ X)\ \mu\ \alpha \end{aligned}$$

The function *qbs-l* is injective by its definition (recall the definition of qbs-measures).

Next, we construct the s-finite measure monad.

▶ **Lemma 5.** *The quasi-Borel space of qbs-measures on X has the following structure.*

$$\begin{aligned} \textit{monadM-qbs} &:: \textit{'a quasi-borel} \Rightarrow \textit{'a qbs-measure quasi-borel} \\ \textit{qbs-space}\ (\textit{monadM-qbs}\ X) &= \{s.\ s \textit{ is a qbs-measure on } X\} \\ \textit{qbs-Mx}\ (\textit{monadM-qbs}\ X) &= \{\lambda r.\ [\![X, \alpha, k\ r]\!]_{\mathsf{sfin}}\ |\alpha\ k.\ \alpha \in \textit{qbs-Mx}\ X \wedge \textit{s-finite-kernel}\ \mathbb{R}\ \mathbb{R}\ k\} \end{aligned}$$

Notice that we use the s-finite kernel in the equation of *qbs-Mx* (*monadM-qbs* X). The proof of being quasi-Borel spaces is almost the same as the one of the probability monad. In the proof, we use that $\mathbb{N} \otimes_M \mathbb{R}$ is standard Borel. It is shown by the facts that $\mathbb{N}$ and $\mathbb{R}$ are standard Borel spaces, and the product measurable space of standard Borel spaces is again a standard Borel space.

The return (unit) operator and bind operator are defined as follows.

$$\begin{aligned} \textit{return}_Q &:: \textit{'a quasi-borel} \Rightarrow \textit{'a} \Rightarrow \textit{'a qbs-measure} \\ \textit{return}_Q\ X\ x &= [\![X, \lambda r.\ x, \nu]\!]_{\mathsf{sfin}} \\ \gg\!= &:: \textit{'a qbs-measure} \Rightarrow (\textit{'a} \Rightarrow \textit{'b qbs-measure}) \Rightarrow \textit{'b qbs-measure} \\ [\![X, \alpha, \mu]\!]_{\mathsf{sfin}} \gg\!= f &= [\![Y, \beta, \mu \gg\!=_k k]\!]_{\mathsf{sfin}} \end{aligned}$$

In the above definition,

- ν is an arbitrary probability measure on $\mathbb{R}$,
- $\beta \in$ *qbs-Mx* Y and *s-finite-kernel* $\mathbb{R}\ \mathbb{R}\ k$,
- $f \circ \alpha = (\lambda r.\ [\![X, \beta, k\ r]\!]_{\mathsf{sfin}})$.

Such β and k always exist since $f \circ \alpha \in$ *qbs-Mx* (*monadM-qbs* X).

The return operator and bind operator are defined in Isabelle/HOL as follows.

definition *return$_Q$* :: *'a quasi-borel* ⇒ *'a* ⇒ *'a qbs-measure* **where**
return$_Q$ X x ≡ [\![*X*, λr. *x*, *SOME μ. real-distribution μ*]\!]$_{\mathsf{sfin}}$

definition *bind-qbs* :: [*'a qbs-measure*, *'a* ⇒ *'b qbs-measure*] ⇒ *'b qbs-measure* **where**
bind-qbs s f ≡ (*let*
(*X*, *α*, *μ*) = *rep-qbs-measure s*;
Y = *qbs-space-of* (*f* (*α undefined*));
(*β*, *k*) = (*SOME* (*β*, *k*). *f* ∘ *α* = (λr. [\![*Y*, *β*, *k r*]\!]$_{sfin}$) ∧ *β* ∈ *qbs-Mx Y* ∧ *s-finite-kernel* ℝ ℝ *k*)
in [\![*Y*, *β*, *μ* ≫=$_k$ *k*]\!]$_{\mathsf{sfin}}$)

Here, *SOME x. P x* denotes some x satisfying P (Hilbert's ε), *real-distribution* μ means that μ is a probability measure on $\mathbb{R}$, *rep-qbs-measure* s returns a representative of the qbs-measure s, and *qbs-space-of* s returns the underlying space of s.

▶ **Theorem 6.** *The triple* $(\mathit{monadM\text{-}qbs}, \mathit{return}_Q, \gg\!=)$ *forms a commutative strong monad.*

The monad inherits properties of s-finite kernels and $\gg\!=_k$ which we have shown in Section 2.2. Proof of the laws for commutative strong monad is similar to the one of the probability monad. In the proof, we use that $\mathbb{R} \otimes_M \mathbb{R}$ is standard Borel.

The Probability Monad

We obtain the probability monad on quasi-Borel spaces by restricting *monadM-qbs X* as follows.

definition *monadP-qbs X* ≡ *sub-qbs* (*monadM-qbs X*) {*s*. *prob-space* (*qbs-l s*)}

The *sub-qbs X A* returns the sub space of a quasi-Borel space.

$$\begin{aligned} \mathit{qbs\text{-}space}\ (\mathit{sub\text{-}qbs}\ X\ A) &= \mathit{qbs\text{-}space}\ X \cap A \\ \mathit{qbs\text{-}Mx}\ (\mathit{sub\text{-}qbs}\ X\ A) &= \{\alpha.\ \alpha \in \mathit{qbs\text{-}Mx}\ X \wedge (\forall r.\ \alpha\ r \in A)\} \end{aligned}$$

The triple $(\mathit{monadP\text{-}qbs}, \mathit{return}_Q, \gg\!=)$ also forms a commutative strong monad. This monad has the exactly same structure with the probability monad in our previous work.

Integration

Integration with qbs-measure is defined through the Lebesgue integration. For $f \in X \to_Q \mathbb{R}$ and $s \in \mathit{qbs\text{-}space}\ (\mathit{monadM\text{-}qbs}\ X)$, the integration $(\int_Q x.\ f\ x\ \partial s)$ is defined by $(\int_Q x.\ f\ x\ \partial s) = (\int x.\ f\ x\ \partial(\mathit{qbs\text{-}l}\ s))$. The notions of *integrable* and *almost everywhere* are defined in a similar way.

For an s-finite measure M on a standard Borel space, integration w.r.t. M in measure theory is represented as integration in quasi-Borel theory. We define the inverse function of *qbs-l*, by $\mathit{qbs\text{-}l}^{-1}\ M = [\![M, \mathit{from\text{-}real}_M, \mathit{distr}\ \mathbb{R}\ M\ \mathit{to\text{-}real}_M]\!]_{\mathsf{sfin}}$. Using these conversions *qbs-l* and $\mathit{qbs\text{-}l}^{-1}$, we obtain $(\int x.\ f\ x\ \partial M) = (\int_Q x.\ f\ x\ \partial(\mathit{qbs\text{-}l}^{-1}\ M))$. We thus may regard an s-finite measure M on a standard Borel space as a qbs-measure $\mathit{qbs\text{-}l}^{-1}\ M$ on $R\ M$, and regard a qbs-measure s as an s-finite measure *qbs-l s*.

For instance, we can represent the Lebesgue measure as a qbs measure. Recall that the Lebesgue measure is σ-finite, hence it is s-finite.

definition $\mathit{lborel}_Q \equiv \mathit{qbs\text{-}l}^{-1}\ \mathit{lborel}$
lemma $\mathit{qbs\text{-}l}\ \mathit{lborel}_Q = \mathit{lborel}$
corollary $(\int_Q x.\ f\ x\ \partial \mathit{lborel}_Q) = (\int x.\ f\ x\ \partial \mathit{lborel})$

4 Probabilistic Programs

Let us implement a probabilistic programming language supporting higher-order functions, sampling, and conditioning with quasi-Borel spaces and the s-finite measure monad. We discuss three examples in this section.

4.1 The Language

We use Isabelle/HOL terms as probabilistic programs. The language design is inspired by `HPProg` introduced by Sato et al. [21]. We first briefly review the type system and semantics of `HPProg`. The language `HPProg` is a higher-order functional probabilistic programming

language based on simply-typed lambda calculus along with the monadic type for distributions. Types are defined inductively as follows.

$$T ::= \mathsf{nat} \mid \mathsf{bool} \mid \mathsf{real} \mid \mathsf{preal} \mid \mathsf{list}[T] \mid T \times T \mid T \Rightarrow T \mid M[T].$$

The type preal denotes the type of $[0, \infty]$ and $M[T]$ denotes the type of distributions (measures) on T. In the semantics, types are interpreted as quasi-Borel spaces.

$$[\![\mathsf{nat}]\!] = \mathbb{N}, \quad [\![\mathsf{bool}]\!] = \mathbb{B}, \quad [\![\mathsf{real}]\!] = \mathbb{R}, \quad [\![\mathsf{preal}]\!] = \mathbb{R}_{\geq 0}, \quad [\![\mathsf{list}[T]]\!] = \textit{list-qbs}\ [\![T]\!]$$
$$[\![T_1 \times T_2]\!] = [\![T_1]\!] \otimes_Q [\![T_2]\!], \quad [\![T_1 \Rightarrow T_2]\!] = [\![T_1]\!] \Rightarrow_Q [\![T_2]\!], \quad [\![M[T]]\!] = \textit{monadM-qbs}\ [\![T]\!].$$

A typing judgment $\Gamma \vdash t : T$ is interpreted as "$[\![t]\!]$ is a morphism from $[\![\Gamma]\!]$ to $[\![T]\!]$". A typing judgment $\vdash t : T$ is interpreted as "$[\![t]\!] \in [\![T]\!]$".

According to this semantics, an Isabelle/HOL term is interpreted as a probabilistic program. We say that an Isabelle/HOL term t is a program of type T if $t \in \textit{qbs-space}\ T$. Many standard constants in Isabelle/HOL are programs.

$$(+) \in \mathbb{R} \Rightarrow_Q \mathbb{R} \Rightarrow_Q \mathbb{R}\ , \qquad (-) \in \mathbb{R} \Rightarrow_Q \mathbb{R} \Rightarrow_Q \mathbb{R}\ , \qquad (*) \in \mathbb{R} \Rightarrow_Q \mathbb{R} \Rightarrow_Q \mathbb{R}$$

$$[] \in \textit{list-qbs}\ X\ , \qquad \textit{Cons} \in X \Rightarrow_Q \textit{list-qbs}\ X \Rightarrow_Q \textit{list-qbs}\ X$$

$$\textit{rec-list} \in Y \Rightarrow_Q (X \Rightarrow_Q \textit{list-qbs}\ X \Rightarrow_Q Y \Rightarrow_Q Y) \Rightarrow_Q \textit{list-qbs}\ X \Rightarrow_Q Y$$

Operators for distributions are also programs.

$$\textit{return}_Q\ X \in X \Rightarrow_Q \textit{monadM-qbs}\ X$$

$$(\gg\!=) \in \textit{monadM-qbs}\ X \Rightarrow_Q (X \Rightarrow_Q \textit{monadM-qbs}\ Y) \Rightarrow_Q \textit{monadM-qbs}\ Y$$

$$(\otimes_{Q\,mes}) \in \textit{monadM-qbs}\ X \Rightarrow_Q \textit{monadM-qbs}\ Y \Rightarrow_Q \textit{monadM-qbs}\ (X \otimes_Q Y)$$

$$\textit{Uniform} \in \mathbb{R} \Rightarrow_Q \mathbb{R} \Rightarrow_Q \textit{monadM-qbs}\ \mathbb{R}\ , \qquad \textit{Gauss} \in \mathbb{R} \Rightarrow_Q \mathbb{R} \Rightarrow_Q \textit{monadM-qbs}\ \mathbb{R}$$

The program $(\otimes_{Q\,mes})$ is defined for $p \in \textit{monadM-qbs}\ X$ and $q \in \textit{monadM-qbs}\ Y$ by

$$p \otimes_{Q\,mes} q = p \gg\!= (\lambda x.\ q \gg\!= (\lambda y.\ \textit{return}_Q\ (X \otimes_Q Y)\ (x,y)))$$

which denotes their product distribution[7]. The program *Uniform a b* denotes the continuous uniform distribution between a and b. The program *Gauss* μ σ denotes the Gaussian distribution with the average μ and the standard deviation σ.

Let us compare the language implementation with our previous work [11]. Our previous language implementation lift Isabelle/HOL constants to constant functions in order to accommodate contexts. For instance, a real number r is described as $(\lambda env.\ r) \in \Gamma \Rightarrow_Q \mathbb{R}$. The variables are projections from contexts and thus programs are written in de Bruijn index, that is, variables are identified by natural numbers. Although using de Bruijn index makes it almost straightforward to write type checking proofs, it causes low readability and cumbersome renaming of variables during proofs. By contrast, our new implementation uses Isabelle/HOL terms directly. This approach is similar to CryptHOL by Basin et al. [5, 16],

[7] Because the s-finite measure monad is commutative, we have

$$p \otimes_{Q\,mes} q = q \gg\!= (\lambda y.\ p \gg\!= (\lambda x.\ \textit{return}_Q\ (X \otimes_Q Y)\ (x,y))).$$

where they have embedded a functional probabilistic programming language for discrete distributions in order to verify cryptographic algorithms. The benefit is that it is much more readable and easier to work with terms when writing programs and reasoning about programs. Our qbs prover presented in Section 3.2 almost automates type checking even though programs are written as Isabelle/HOL terms. As we will demonstrate in later sections, program verification can be done directly in Isabelle/HOL.

The *query* Command

We define the *query* command which enables one to write conditional distributions. The *query* has the following type:

$$\mathit{query} \in \mathit{monadM\text{-}qbs}\ X \Rightarrow_Q (X \Rightarrow_Q \mathbb{R}_{\geq 0}) \Rightarrow_Q \mathit{monadM\text{-}qbs}\ X\ .$$

For a prior distribution s and a likelihood f, $\mathit{query}\ s\ f$ returns the posterior distribution. The *query* command is defined through two operators: $\mathit{density}_Q$ (`scale` in `HPProg`) and $\mathit{normalize}_Q$.

definition $\mathit{query} \equiv (\lambda s\ f.\ \mathit{normalize\text{-}qbs}\ (\mathit{density\text{-}qbs}\ s\ f))$

The operator $\mathit{density}_Q$ takes a qbs-measure s and a non-negative function f and rescales s with the density function f. The $\mathit{density}_Q$ satisfies following properties.

$$\mathit{density}_Q \in \mathit{monadM\text{-}qbs}\ X \Rightarrow_Q (X \Rightarrow_Q \mathbb{R}_{\geq 0}) \Rightarrow_Q \mathit{monadM\text{-}qbs}\ X$$

$$(\int_Q x.\ g\ x\ \partial(\mathit{density}_Q\ s\ f)) = (\int_Q x.\ f\ x * g\ x\ \partial s)$$

The operator $\mathit{normalize}_Q$ normalizes a qbs-measure s on X. If $\mathit{qbs\text{-}l}\ s\ X = 0$ or ∞, then $\mathit{normalize}_Q\ s$ returns the null-measure on X.

$$\mathit{normalize}_Q \in \mathit{monadM\text{-}qbs}\ X \Rightarrow_Q \mathit{monadM\text{-}qbs}\ X$$

The *condition* Command

We introduce the *condition* command, which produces a conditional distribution with a predicate. The *condition* command has the following type and defined using the *query* command and the indicator function as follows.

$$\mathit{condition} \in \mathit{monadM\text{-}qbs}\ X \Rightarrow_Q (X \Rightarrow_Q \mathbb{B}) \Rightarrow_Q \mathit{monadM\text{-}qbs}\ X$$

definition $\mathit{condition}\ s\ P \equiv \mathit{query}\ s\ (\lambda x.\ \mathit{if}\ P\ x\ \mathit{then}\ 1\ \mathit{else}\ 0)$

4.2 Example: What time is it?

We formalize the example from Staton [25, Section 2.2], which we have shown in introduction. This example uses two language features: higher-order function and conditioning. Let us consider the following situation.

- We want to know what time it is.
- We know the rate of bikes per hour, which depends on time.
- We observed a 1 minute gap between two bikes.
- What time is it?

We define the program *whattime* as follows.

```
definition whattime :: (real ⇒ real) ⇒ real qbs-measure where
whattime ≡ (λf. do {
                let T = Uniform 0 24 in
                query T (λt. let r = f t in
                          exponential-density r (1 / 60))
              })
```

The program *whattime* receives a function f which determines the rate of bikes per hour. Then the program returns the posterior after observing a 1 minute gap between two bikes. The return value $f\ t$ denotes the rate of bikes per hour at the time t, and the time gap between two bikes follows the exponential distribution $\mathrm{Exp}(f\ t)$. Thus, the likelihood is calculated using the density function *exponential-density* of the exponential distribution. We can prove *whattime* is a program just by unfolding the definition thanks to our qbs prover presented in Section 3.2.

```
lemma whattime ∈ (ℝ ⇒Q ℝ) ⇒Q monadM-qbs ℝ
  by(simp add: whattime-def)
```

As explained by Staton, the posterior is computed as follows.

```
lemma
  assumes f ∈ ℝ ⇒Q ℝ and U ∈ sets ℝ and ⋀t. f t ≥ 0
  defines N ≡ (∫ t∈{0<..<24}. (f t * exp (− 1/ 60 * f t)) ∂lborel)
  assumes N ≠ 0 and N ≠ ∞
  shows 𝒫(t in whattime f. t ∈ U) = (∫ t∈{0<..<24}∩U. (f t * exp (− 1/ 60 * f t)) ∂lborel) / N
```

4.3 Example: Two Dice

As a second example, we formalize the example from Sampson [20, Section 2.3]. This example uses two language features: sampling and conditioning. We consider the following problem.

- We role two dice.
- We observe at least one die is 4.
- What is the sum of the two dice?

We describe the distribution of the sum of the two dice as follows.

```
definition two-dice :: nat qbs-measure where
two-dice ≡ do {
             let die1 = die;
             let die2 = die;
             let twodice = die1 ⊗Qmes die2;
             (x,y) ← condition twodice (λ(x,y). x = 4 ∨ y = 4);
             returnQ ℕ (x + y)
            }
```

Here, *die* ∈ *monadM-qbs* ℕ denotes the distribution of rolling a fair die. The program picks a sample from the conditional distribution, then returns the sum of dice. The program *two-dice* has the following type.

```
lemma two-dice ∈ monadM-qbs ℕ
  by(simp add: two-dice-def)
```

We show the probabilities where the program takes each possible value.

```
lemma
 𝒫(x in two-dice. x = 5)  = 2 / 11   𝒫(x in two-dice. x = 6)  = 2 / 11
 𝒫(x in two-dice. x = 7)  = 2 / 11   𝒫(x in two-dice. x = 8)  = 1 / 11
 𝒫(x in two-dice. x = 9)  = 2 / 11   𝒫(x in two-dice. x = 10) = 2 / 11
```

4.4 Example: Gaussian Mean Learning

As a final example, let us formalize the example from Sato et. al. [21, Section 8.2]. We implement the Gaussian Mean Learning algorithm and prove two properties: convergence and stability under change of priors. In a common situation in statistical modeling or machine learning, we try to infer unknown parameters from a sample list. For instance, let us consider the following situation.

- We want to know the mean of a Gaussian distribution with a known standard deviation.
- We have a sample sequence from the Gaussian distribution.
- What is the posterior of the mean?

The following algorithm does Bayesian learning of the mean of a Gaussian distribution with a known standard deviation σ from a sample list.

```
primrec GaussLearn' :: [real, real qbs-measure, real list] ⇒ real qbs-measure where
  GaussLearn' - p [] = p
| GaussLearn' σ p (y#ls) = query (GaussLearn' σ p ls) (normal-density y σ)
```

Here, *normal-density* $y\ \sigma$ is the density function of the Gaussian distribution *Gauss* $y\ \sigma$ with mean y.

The program *GaussLearn'* receives a standard deviation σ, a prior p and a sample list L. In each iteration, the program picks a sample from L, then updates the prior. Our qbs prover can show that *GaussLearn'* is a program because *GaussLearn'* is a primitive recursive function[8].

```
lemma GaussLearn' ∈ ℝ ⇒Q monadM-qbs ℝ ⇒Q list-qbs ℝ ⇒Q monadM-qbs ℝ
  by (simp add: GaussLearn'-def)
```

From now on, we fix $\sigma > 0$ and abbreviate *GaussLearn'* σ as *GaussLearn*.

The first property, convergence, is described as follows.

```
lemma
  assumes ξ > 0 and n = length L
  shows GaussLearn (Gauss δ ξ) L =
        Gauss ((Total L * ξ² + δ * σ²) / (n * ξ² + σ²)) (sqrt ((ξ² * σ²) / (n * ξ² + σ²)))
```

Here, the program *Total* $\in$ *list-qbs* $\mathbb{R} \Rightarrow_Q \mathbb{R}$ sums up all elements of a list. The above statement says that if the prior of the mean is *Gauss* $\delta\ \xi$, then the posterior is also a Gaussian distribution. Furthermore, its mean and standard deviation are close to the average of the samples and 0, respectively, when n is sufficiently large.

Next, let us see the second property, stability under change of priors. We show that if we run *GaussLearn* from two different priors and give a large sample list whose average is bounded, then the resulting posteriors will be close. We measure the difference between distributions by the Kullback-Leiber (KL) divergence. The KL divergence is provided as *KL-divergence* in the standard Isabelle/HOL library. If p and q are probability distributions on $\mathbb{R}$ which have positive density functions f and g, respectively, then we have the following well-known form of KL divergence:

```
KL-divergence (exp 1) p q = (∫ x. g x * ln (g x / f x) ∂lborel)
```

The second property is stated as follows.

[8] Internally, the **primrec** command defines a primitive recursive function using recursors such as *rec-nat* and *rec-list*.

```
lemma GaussLearn-KL-divergence:
  fixes a b c d ε K :: real
  assumes ε > 0 and b > 0 and d > 0
  shows ∃N. ∀L. length L > N ⟶ |Total L / length L| < K ⟶
          KL-divergence (exp 1) (GaussLearn (Gauss a b) L) (GaussLearn (Gauss c d) L) < ε
```

Intuitively, the above property says that if we run *GaussLearn* with two different Gauss distributions, then we can make the distance of posteriors as close as we want with a large sample list whose average is bounded.

5 Conclusion

We have implemented s-finite kernels, the Borel isomorphism theorem, proof automation for quasi-Borel spaces, and the s-finite measure monad. Using our formalization, we can directly treat probabilistic programs presented in previous works and prove their properties. Our work enables us to denote probabilistic programs supporting all of higher-order functions, samplings, and conditioning, while our previous work [11] does not support conditioning and the work by Affeldt et al. [1] does not support higher-order functions.

There are several researches related to probabilistic programs with proof assistants. Eberl et al. have constructed an executable first-order functional probabilistic programming language which computes density functions in Isabelle, and proved its correctness [7]. Basin et al. have implemented CryptHOL for rigorous game-based proofs in Isabelle/HOL [5, 16]. They shallowly embedded a functional programming language, and verified cryptographic algorithms. For machine learning verification, Bagnall and Stewart have embedded MLCert in Coq [4], and Tristan et al. have implemented a simplified measure-theoretic semantics of probabilistic programs based on the reparameterizations to the uniform distibution on the unit interval and partially automated verification in Lean [26]. Zhang and Amin formalized a formal semantics for a core probabilistic programming language and proved that logical relatedness implies contextual equivalence using axiomatized measure theory in Coq [31].

There are future extensions of our work. In our formalization, we have manually constructed quasi-Borel spaces on basic data types defined inductively, such as lists and options. Then we show that constructors and recursors (primitive recursive function operator) are morphisms. We expect that we can automate this process. Furthermore, we think that it is also possible to show that general wellfounded recursive functions, which may not be primitive recursive, are morphisms.

References

1 Reynald Affeldt, Cyril Cohen, and Ayumu Saito. Semantics of probabilistic programs using s-finite kernels in Coq. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, pages 3–16, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3573105.3575691.

2 Robert J. Aumann. Borel structures for function spaces. *Illinois Journal of Mathematics*, 5(4):614–630, 1961. doi:10.1215/ijm/1255631584.

3 Jeremy Avigad, Johannes Hölzl, and Luke Serafin. A formally verified proof of the central limit theorem. *Journal of Automated Reasoning*, 59(4):389–423, 2017. doi:10.1007/s10817-017-9404-x.

4 Alexander Bagnall and Gordon Stewart. Certifying the true error: Machine learning in Coq with verified generalization guarantees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:2662–2669, 2019. doi:10.1609/aaai.v33i01.33012662.

5 David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar, Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *Journal of Cryptology*, 33:494–566, 2020.

6 Marek Biskup. Lecture note of math245b in UCLA. `https://web.archive.org/web/20210506130459/https://www.math.ucla.edu/~biskup/245b.1.20w/`, 2020. Accessed: January 17. 2023.

7 Manuel Eberl, Johannes Hölzl, and Tobias Nipkow. A verified compiler for probability density functions. In *European Symposium on Programming (ESOP 2015)*, volume 9032 of *LNCS*, pages 80–104. Springer, 2015. `doi:10.1007/978-3-662-46669-8_4`.

8 Michèle Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, pages 68–85, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. `doi:10.1007/BFb0092872`.

9 Noah D. Goodman et al. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 220–229. AUAI Press, 2008.

10 Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS 2017. IEEE Press, 2017. `doi:10.1109/lics.2017.8005137`.

11 Michikazu Hirata, Yasuhiko Minamide, and Tetsuya Sato. Program logic for higher-order probabilistic programs in Isabelle/HOL. In Michael Hanus and Atsushi Igarashi, editors, *Functional and Logic Programming*, pages 57–74, Cham, 2022. Springer International Publishing.

12 Johannes Hölzl. Markov processes in Isabelle/HOL. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 100–111. Association for Computing Machinery, 2017. `doi:10.1145/3018610.3018628`.

13 Johannes Hölzl, Armin Heller, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk. Three chapters of measure theory in Isabelle/HOL. In *Interactive Theorem Proving*, ITP 2011, pages 135–151. Springer Berlin Heidelberg, 2011.

14 Cezary Kaliszyk and Christian Urban. Quotients revisited for Isabelle/HOL. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC, pages 1639–1644, New York, NY, USA, 2011. Association for Computing Machinery. `doi:10.1145/1982185.1982529`.

15 Ondřej Kunčar and Andrei Popescu. From types to sets by local type definitions in higher-order logic. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving*, pages 200–218, Cham, 2016. Springer International Publishing.

16 Andreas Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In *Programming Languages and Systems*, pages 503–531, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. `doi:10.1007/978-3-662-49498-1_20`.

17 Mihails Milehins. An extension of the framework types-to-sets for Isabelle/HOL. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2022, pages 180–196, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3497775.3503674`.

18 Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

19 Andrei Popescu and Dmitriy Traytel. Admissible types-to-pers relativization in higher-order logic. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. `doi:10.1145/3571235`.

20 Adrian Sampson. Probabilistic programming. `http://adriansampson.net/doc/ppl.html`. Accessed: January 25. 2023.

21 Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. Formal verification of higher-order probabilistic programs: reasoning about approximation, convergence, bayesian inference, and optimization. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019. `doi:10.1145/3290351`.

22 Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. Denotational

validation of higher-order bayesian inference. *Proc. ACM Program. Lang.*, 2(POPL), 2017. doi:10.1145/3158148.

23 Shashi Mohan Srivastava. *A Course on Borel Sets*. Springer, 1998. doi:10.1007/b98956.

24 Sam Staton. Commutative semantics for probabilistic programming. In Hongseok Yang, editor, *Programming Languages and Systems*, pages 855–879, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

25 Sam Staton. *Probabilistic Programs as Measures*, pages 43–74. Cambridge University Press, 2020. doi:10.1017/9781108770750.003.

26 Jean-Baptiste Tristan, Joseph Tassarotti, Koundinya Vajjha, Michael L. Wick, and Anindya Banerjee. Verification of ML systems via reparameterization, 2020. arXiv:2007.06776.

27 Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi:10.1145/3290349.

28 Matthijs Vákár and Luke Ong. On s-finite measures and kernels, 2018. doi:10.48550/ARXIV.1810.01837.

29 Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, pages 1024–1032, 2014.

30 Hongseok Yang. Semantics of higher-order probabilistic programs with continuous distributions. https://alfa.di.uminho.pt/~nevrenato/probprogschool_slides/Hongseok.pdf. Accessed: February 8. 2023.

31 Yizhou Zhang and Nada Amin. Reasoning about "reasoning about reasoning": Semantics and contextual equivalence for probabilistic programs with nested queries and recursion. *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi:10.1145/3498677.