Formal Verification of Higher-Order Probabilistic Programs

Reasoning about Approximation, Convergence, Bayesian Inference, and Optimization

TETSUYA SATO, University at Buffalo, SUNY, USA
ALEJANDRO AGUIRRE, IMDEA Software Institute, Spain
GILLES BARTHE, IMDEA Software Institute, Spain
MARCO GABOARDI, University at Buffalo, SUNY, USA
DEEPAK GARG, Max Planck Institute for Software Systems, Germany
JUSTIN HSU, University of Wisconsin–Madison, USA

Probabilistic programming provides a convenient *lingua franca* for writing succinct and rigorous descriptions of probabilistic models and inference tasks. Several probabilistic programming languages, including Anglican, Church or Hakaru, derive their expressiveness from a powerful combination of continuous distributions, conditioning, and higher-order functions. Although very important for practical applications, these features raise fundamental challenges for program semantics and verification. Several recent works offer promising answers to these challenges, but their primary focus is on foundational semantics issues.

In this paper, we take a step further by developing a suite of logics, collectively named PPV, for proving properties of programs written in an expressive probabilistic higher-order language with continuous sampling operations and primitives for conditioning distributions. Our logics mimic the comfortable reasoning style of informal proofs using carefully selected axiomatizations of key results from probability theory. The versatility of our logics is illustrated through the formal verification of several intricate examples from statistics, probabilistic inference, and machine learning. We further show expressiveness by giving sound embeddings of existing logics. In particular, we do this in a parametric way by showing how the semantics idea of (unary and relational) $\top\top$-lifting can be internalized in our logics. The soundness of PPV follows by interpreting programs and assertions in quasi-Borel spaces (QBS), a recently proposed variant of Borel spaces with a good structure for interpreting higher order probabilistic programs.

CCS Concepts: • **Theory of computation → Probabilistic computation**; **Logic and verification**; **Higher order logic**; • **Computing methodologies** → *Machine learning algorithms*;

Additional Key Words and Phrases: probabilistic programming, formal reasoning, relational type systems

ACM Reference Format:
Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. 2019. Formal Verification of Higher-Order Probabilistic Programs: Reasoning about Approximation, Convergence, Bayesian Inference, and Optimization. *Proc. ACM Program. Lang.* 3, POPL, Article 38 (January 2019), 30 pages. https://doi.org/10.1145/3290351

Authors' addresses: Tetsuya Sato, CSE, University at Buffalo, SUNY, USA, tetsuyas@buffalo.edu; Alejandro Aguirre, IMDEA Software Institute, Spain, alejandro.aguirre@imdea.org; Gilles Barthe, IMDEA Software Institute, Spain, gjbarthe@gmail.com; Marco Gaboardi, CSE, University at Buffalo, SUNY, USA, gaboardi@buffalo.edu; Deepak Garg, Max Planck Institute for Software Systems, Germany, dg@mpi-sws.org; Justin Hsu, CS, University of Wisconsin–Madison, USA, email@justinh.su.

2475-1421/2019/1-ART38
https://doi.org/10.1145/3290351

1 INTRODUCTION

Probabilistic programming is *en vogue* in statistics and machine learning, where modern probabilistic programming languages are viewed as a convenient *lingua franca* for writing classical statistical estimators, and for describing probabilistic models and performing probabilistic inference. A key strength of many modern probabilistic programming languages is their expressiveness, which allows programmers to give succinct descriptions for a broad range of probabilistic models, and to program specialized inference algorithms when generic algorithms do not perform well. This expressiveness has led to significant theoretical challenges. Specifically, many probabilistic programming languages adopt a combination of features that goes beyond standard program semantics and program verification. In this paper, we consider functional probabilistic programming languages and focus on the following elements:

- *sampling*: the first key ingredient of a probabilistic programming language is a construct to sample from (continuous) distributions. A popular way to expose this mechanism is the monadic approach, where probabilities are modelled as effects. Languages feature a type constructor M for probability measures and monadic operations for sampling from continuous distributions or composing probabilistic computations.
- *conditioning*: the second key ingredient of probabilistic programming languages is a conditioning operator, which can be used to build a conditional distribution that incorporates observations from the real world. Conditioning is often performed through specific constructs, such as `observe` or `query`, which scale a distribution to a measure according to a likelihood function, and then normalize the resulting measure back to a distribution.
- *higher-order functions*: probabilistic models and statistical tasks are often described in a natural way by means of functional higher-order programs. The modularity that higher-order functions provide is useful for writing likelihood functions, weighting functions, parametric models, etc. These components facilitate writing concise and expressive probabilistic computations.

Examples of probabilistic programming languages that incorporate the features above are Anglican, Church, and Hakaru. For example, Anglican [Wood et al. 2014] extends Clojure with constructs for basic probability distributions and an operation `observe`, which is used to build conditional distributions with respect to a predicate representing an observation of random variables. Church [Goodman et al. 2008] supports in a simply typed lambda calculus a similar conditioning operation named `query`, Hakaru [Narayanan et al. 2016] supports these features as a domain-specific language embedded in Haskell.

Despite their popularity, higher-order probabilistic programming languages pose significant challenges for semantics and verification. In particular, a classical result [Aumann 1961] shows that the category of measurable spaces is not Cartesian closed, and thus it cannot be used to give denotational models for higher-order probabilistic languages. Aumann's negative result has triggered a long line of research, culminating in several recent proposals for semantic models of higher-order probabilistic languages. One such proposal, relevant to our work, is the notion of the quasi-Borel space (QBS) [Heunen et al. 2017], which has a rich categorical structure and yields an elegant denotational model for higher-order probabilistic programs.

While a denotational model facilitates formal reasoning about probabilistic programs, the resulting style of reasoning is typically hard to use. As with more standard programming languages, we would prefer to use other techniques, such as equational methods and program logics, to structure the arguments at a higher level. Several recent papers have started to look at this. For instance, Staton [2017] and Culpepper and Cobb [2017] have recently proposed equational methods for proving equivalences between higher-order probabilistic programs. Culpepper and Cobb [2017]

propose an equational framework based on observational equivalence and logical relations, while Staton [2017] proposes a semantics method for equational reasoning which can be used for program equivalence. These two methods are important steps towards more general high-level reasoning techniques. However, their main focus is program equivalence and they do not directly support arbitrary program properties. Moreover, their approach is based on techniques which are difficult to directly apply to complex examples. As a result, for more complex examples the only currently viable approach is to resort directly to the denotational semantics; for instance, Ścibior et al. [2017] use semantic methods to prove the correctness of higher-order Bayesian inference.

Our work: The long-term goal of our research is to build practical verification tools for higher-order probabilistic programs, and to leverage these tools for building libraries of formally verified algorithms from machine learning and statistics. This paper makes an initial step towards this goal and justifies its feasibility by introducing a framework, called the Probabilistic Programming Verification framework (PPV), for proving (unary and relational) properties of probabilistic higher-order programs with discrete and continuous distributions. PPV is:

- *expressive*: it can reason about different properties of probabilistic programs, including approximation, convergence, probabilistic inference and optimization.
- *practical*: it supports lean derivations that abstract away from lower-level concerns, like measurability.
- *sound*: it can be soundly interpreted in the category of quasi-Borel spaces.

PPV's design is based on three different logics: PL, UPL and RPL. These logics are presented in the style of Aguirre et al. [2017]: PL is an *intuitionistic logic* for reasoning about higher-order programs using a style inspired by HOL [Jacobs and Melham 1993] based on judgments of the form $\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi$. UPL is a *unary program logic* which manipulates judgments of the form $\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e \colon \tau \mid \phi$. Finally, RPL is a *relational program logic* which manipulates judgments of the form $\Gamma \mid \Psi \vdash_{\mathrm{RPL}} e \colon \tau \sim e' \colon \tau' \mid \phi'$. Here Γ is a simple typing context; τ and τ' are the simple types of the expressions e and e'; Ψ is a set of assumed assertions; ϕ is a postcondition; and ϕ' is a relational postcondition. The proof systems are equi-expressive, but the UPL and RPL are closer to the syntax-directed style of reasoning generally favored in unary and relational program verification, respectively. We define an interpretation of assertions in the category of QBS predicates and prove that the logics are sound with respect to the interpretation. This interpretation guarantees that every subset of a quasi-Borel space yields an object in the category. As a consequence, assertions of the logic are interpreted set-theoretically, and extensionality is valid. This facilitates formal reasoning and formal verification.

To further ease program verification, we define carefully crafted axiomatizations of fundamental probabilistic definitions and results, including expectations as well as concentration bounds. Following Ścibior et al. [2017], we validate the soundness of these axiomatizations using synthetic measure theory for the QBS framework. This ensures that a derivation based on our proof system and axioms is valid in quasi-Borel spaces. A consequence of this approach is that, in order to verify programs, a user of PPV can focus on higher-level reasoning about probabilistic programs, rather than the specific details of QBS.

We validate our design through a series of examples from statistics, Bayesian inference and machine learning. We also demonstrate that our systems can be used as a framework where other program logics can be embedded. We show this in a parametrized way by using PPV to define a family of *graded* $\top\top$*-liftings*, a logical relation-like technique to construct predicates/relations over probability distributions, starting from predicates/relations over values. As a concrete application, we embed two recent probabilistic logics: a *union bound* logic for reasoning about accuracy [Barthe

et al. 2016b], and a logic for reasoning about probability distributions through couplings [Aguirre et al. 2018].

Overall, our work provides a fresh, verification-oriented perspective on quasi-Borel spaces, and contributes to establish their status as a sound, simple and natural theoretical framework for practical verification of higher-order probabilistic programs.

In technical terms, our framework follows the presentation style introduced by Aguirre et al. [2017] to reason about deterministic higher-order programs. We extend this approach to higher-order probabilistic programs with continuous random variables. A similar approach has been used for discrete random variables by Aguirre et al. [2018] in order to reason about unary and relational properties of Markov chains. Our contribution differs significantly from the one by Aguirre et al. [2018]. Assertions in their framework are non-probabilistic and are interpreted first over deterministic values, and then over distributions over values by probabilistic lifting. Instead, in PPV we can reason about (monadic) probabilistic expressions directly in assertions. This is a key component in expressing probabilistic properties such as the convergence of the expectation of an expression directly. Moreover, Aguirre et al. [2018] support analysis of probabilistic programs via coupling arguments only. PPV's proof rules are more expressive: they allow reasoning about probabilities within the logic.

2 PPV BY EXAMPLE

In this section we introduce the general ideas behind PPV through two examples.

Continuous Observations: Two Uniform Samples. This warm-up example serves as an introduction to Bayesian conditioning and how we can reason about it in our system. Let us consider the following program `twoUs`:

$$\begin{aligned}\texttt{twoUs} \equiv\ & \texttt{let}\, u_1 = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\, u_2 = \texttt{Uniform}(0,1)\ \texttt{in}\\ & \texttt{let}\, y = u_1 \otimes u_2\ \texttt{in}\\ & \texttt{query}\ \ y \Rightarrow \lambda x.(\texttt{if}\ \pi_1(x) < .5 \vee \pi_2(x) > .5\ \texttt{then}\ 1\ \texttt{else}\ 0)\end{aligned}$$

The first line defines two uniform distributions u_1 and u_2. The second line pairs the two distributions together using the product measure of u_1 and u_2 which we denote $u_1 \otimes u_2$ (this is defined formally in Section 3). Then, the third line performs Bayesian conditioning on this product measure using the construction `query`. The *prior* y gets conditioned by the *likelihood function* corresponding to the observation $\pi_1(x) < .5 \vee \pi_2(x) > .5$, and a *posterior* is computed. In this simple example, this is morally equivalent to giving score 1 to the traces that do satisfy the assertion, and score 0 to the ones that do not satisfy it, and rescaling the distribution. In general, we can use the conditioning construct with an arbitrary likelihood function to perform more general inference. After the observation, the posterior is a uniform distribution over the set $\{(x_1, x_2) \mid x_1 < .5 \vee x_2 > .5\}$.

The simple property we will show is that $\Pr_{(x_1,x_2)\sim\mu}[x_1 > .5] = 1/3$, where μ is the posterior after the observation and the pair (x_1, x_2) is distributed by μ. This is expressed in the unary logic UPL—since this is a unary property—through the following judgment:

$$\vdash_{\text{UPL}} \texttt{twoUs} : M[\texttt{real} \times \texttt{real}] \mid \Pr_{z\sim\mathbf{r}}[\pi_1(z) > .5] = 1/3$$

where the distinguished variable $\mathbf{r}$ in the logical assertion represents the given term `twoUs` and the variable z is bound by $\Pr_{z\sim\mathbf{r}}[\ldots]$ and it is used to represent the value sampled from the probability distribution $\mathbf{r}$. We show informally how to derive this assertion. The system UPL allows us to reason in a syntax-directed manner. Since the program starts with three let bindings, the first step will be to apply the rule for let bindings three times. This rule, which we will present formally in Section 6, moves u_1, u_2 and y plus the logical assertions about them into the context. The resulting

judgement is:

$$\begin{array}{l}\Pr_{z\sim u_1}[z > .5] = 1/2, \Pr_{z\sim u_2}[z < .5] = 1/2, \Pr_{z\sim u_1}[\top] = 1, \Pr_{z\sim u_2}[\top] = 1, y = u_1 \otimes u_2 \\ \vdash_{\text{UPL}} \texttt{query}\, y \Rightarrow \lambda x.(\texttt{if}\ \pi_1(x) < .5 \vee \pi_2(x) > .5\ \texttt{then}\ 1\ \texttt{else}\ 0) \colon M[\texttt{real} \times \texttt{real}] \mid \\ \quad \Pr_{z\sim \mathbf{r}}[\pi_1(z) > .5] = 1/3\end{array}$$

where for simplicity we omitted the typing context. The logical assertions on u_1 and u_2 can be easily discharged using the assumption that they are distributed uniformly as $\texttt{Uniform}(0, 1)$, i.e. uniformly between 0 and 1. To finish the proof, we want to use the fact that `query` corresponds to conditioning. In UPL we can do this using the following special rule that internalizes the Bayesian properties of `query`:[1]

$$\frac{\Gamma, x\colon \tau \vdash e'\colon \texttt{bool} \quad \Gamma, x\colon \tau \vdash e''\colon \texttt{bool} \quad \Gamma \vdash e\colon M[\tau]}{\Gamma \mid \Psi \vdash_{\text{UPL}} \texttt{query}\, e \Rightarrow \lambda x.(\texttt{if}\ e'\ \texttt{then}\ 1\ \texttt{else}\ 0)\colon M[\tau] \mid \Pr_{y\sim \mathbf{r}}[e''[y/x]] = \frac{\Pr_{x\sim e}[e' \wedge e'']}{\Pr_{x\sim e}[e']}}\ [\text{Bayes}]$$

This rule corresponds to a natural reasoning principle (derived by Bayes' theorem) for `query` when we have a boolean condition as the likelihood function: the probability of an event e'' under the *posterior* distribution is equal to the probability of the intersection of the event e'' and the observation e', under the *prior* distribution e, divided by the probability of e' under the prior distribution e.

To apply this rule we need to rewrite the postcondition into the appropriate shape: a fraction that has the probability of a conjunction of events in the numerator and the probability of the observed event in the denominator. This can be done in UPL through *subtyping* which lets us reason directly in the logic PL, where we can prove the following judgment:

$$\begin{array}{l}\Pr_{z\sim u_1}[z > .5] = 1/2, \Pr_{z\sim u_2}[z < .5] = 1/2, \Pr_{z\sim u_1}[\top] = 1, \Pr_{z\sim u_2}[\top] = 1, y = u_1 \otimes u_2 \\ \vdash_{\text{PL}} \frac{\Pr_{z\sim y}[(\pi_1(z)<.5 \vee \pi_2(z)>.5) \wedge (\pi_1(z)>.5)]}{\Pr_{z\sim y}[\pi_1(z)<.5 \vee \pi_2(z)>.5]} = \frac{1/4}{3/4} = 1/3\end{array}$$

Using this equivalence and subtyping we can rewrite the judgment we need to prove as follows:

$$\begin{array}{l}\Pr_{z\sim u_1}[z > .5] = 1/2, \Pr_{z\sim u_2}[z < .5] = 1/2, \Pr_{z\sim u_1}[\top] = 1, \Pr_{z\sim u_2}[\top] = 1, y = u_1 \otimes u_2 \\ \vdash_{\text{UPL}} \texttt{query}\, y \Rightarrow \lambda x.(\texttt{if}\ \pi_1(x) < .5 \vee \pi_2(x) > .5\ \texttt{then}\ 1\ \texttt{else}\ 0) : M[\texttt{real} \times \texttt{real}] \mid \\ \quad \Pr_{z\sim \mathbf{r}}[\pi_2(z) > .5] = \frac{\Pr_{z\sim y}[(\pi_1(z)<.5 \vee \pi_2(z)>.5) \wedge (\pi_1(z)>.5)]}{\Pr_{z\sim y}[\pi_1(z)<.5 \vee \pi_2(z)>.5]}\end{array}$$

and this can be proved by applying the [Bayes] rule above, concluding the proof. We saw different components of PPVat work here: unary rules, subtyping, and a special rule for `query`. All these components can be assembled in more complex examples, as we show in Section 8.

Monte Carlo Approximation. As a second example, we show how to use PPV to reason about other classical applications that do not use observations. We consider reasoning about expected value and variance of distributions. Concretely, we show convergence in probability of an implementation of the naive Monte Carlo approximation. This algorithm considers a distribution d and a real-valued function h, and tries to approximate the expected value of $h(x)$ where x is sampled from d by sampling a number i of values from d and computing their mean.

Consider the following implementation of Monte Carlo approximation:

$$\begin{array}{l}\texttt{MonteCarlo} \equiv \texttt{letrec}\ f(i\colon \texttt{nat}) = \texttt{if}(i \le 0)\ \texttt{then}\ \texttt{return}(0) \\ \qquad\qquad \texttt{else}\ \texttt{mlet}\ m = f(i-1)\ \texttt{in}\ \texttt{mlet}\ x = d\ \texttt{in}\ \texttt{return}((1/i) * (h(x) + m * (i-1)))\end{array}$$

Our goal is to prove the convergence in probability of this algorithm, that is, the result can be made as accurate as desired by increasing the sample size (denoted by i above and n below). This is

[1] We introduce the rule here to give some intuition, but this is also discussed in Section 6 after introducing PPV.

formalized in the following UPL judgment (we omit the typing context for simplicity):

$$\begin{array}{l}(\mathbb{E}_{x\sim d}[1] = 1), (\sigma^2 = \mathrm{Var}_{x\sim d}[h(x)]), (\mu = \mathbb{E}_{x\sim d}[h(x)]), (\varepsilon > 0) \vdash_{\mathrm{UPL}} \\ \texttt{MonteCarlo}\colon \texttt{nat} \to M[\texttt{real}] \mid \forall n, (n > 0) \implies \Pr_{y\sim \mathbf{rn}}[|y - \mu| \geq \varepsilon] \leq \sigma^2/n\varepsilon^2\end{array} \tag{1}$$

Formally, we are showing that the probability that the computed mean y differs from the actual mean μ by more than ε is upper bounded by a value that depends inversely on n—more samples lead to better estimates. To derive (1) in UPL we need to perform two steps:

- Calculating the mass, mean, and variance of `MonteCarlo` in UPL:

$$\begin{array}{l}(\mathbb{E}_{x\sim d}[1] = 1), (\sigma^2 = \mathrm{Var}_{x\sim d}[h(x)]), (\mu = \mathbb{E}_{x\sim d}[h(x)]), (\varepsilon > 0) \vdash_{\mathrm{UPL}} \\ \texttt{MonteCarlo}\colon \texttt{nat} \to M[\texttt{real}] \mid \\ \forall n\colon \texttt{nat}.(n > 0) \implies (\mathbb{E}_{y\sim \mathbf{rn}}[1] = 1) \wedge (\mathbb{E}_{y\sim \mathbf{rn}}[y] = \mu) \wedge (\mathrm{Var}_{y\sim \mathbf{rn}}[y] = \sigma^2/n)\end{array} \tag{2}$$

- Applying the Chebyshev inequality (formula 17 in Section 5.1) to (2) using subtyping.

We focus on the proof of (2), which is carried out by induction on n. In our system, the rule for `letrec` lets us prove inductive properties of (terminating) recursive functions by introducing an inductive hypothesis into the set of assertions that can only be instantiated for smaller arguments. After applying this rule, the new goal is:

$$\phi_{\mathrm{IH}} \equiv \forall n\colon \texttt{nat}.(n < i) \implies (n > 0) \implies (\mathbb{E}_{y\sim f(n)}[1] = 1) \wedge (\mathbb{E}_{y\sim f(n)}[y] = \mu) \wedge (\mathrm{Var}_{y\sim f(n)}[y] = \sigma^2/n)$$

On this, we can apply a rule for case distinction according to the two branches of the if-then-else, which gives us the following two premises:

$$\begin{array}{l}\Psi, (i \leq 0) \vdash \texttt{return}(0) \mid \psi \\ \Psi, (i > 0) \vdash \texttt{mlet}\, m = f(i-1)\, \texttt{in}\, (\texttt{mlet}\, x = d\, \texttt{in}\, \texttt{return}(\tfrac{1}{i}(h(x) + m * (i-1)))) \mid \psi\end{array}$$

where $\Psi = (\mathbb{E}_{x\sim d}[1] = 1), (\mu = \mathbb{E}_{x\sim d}[h(x)]), (\sigma^2 = \mathrm{Var}_{x\sim d}[h(x)]), (i > 0), \phi_{\mathrm{IH}}$ and $\psi = (\mathbb{E}_{y\sim \mathbf{r}i}[1] = 1) \wedge (\mathbb{E}_{y\sim \mathbf{r}i}[y] = \mu) \wedge (\mathrm{Var}_{y\sim \mathbf{r}i}[y] = \sigma^2/i)$. The first premise is obvious since the assumptions $(i > 0)$ and $(i \leq 0)$ are contradictory. The second premise follows from subtyping applied to a PL-judgment that is proved by instantiating the induction hypothesis with $i - 1$ and applying axioms on expected values. This concludes the proof.

Again, we have seen here several different components of PPV: unary rules (including the rule for inductive reasoning), subtyping, and the use of equations and axioms. We further illustrate these components of PPV as well as others in verifying more involved examples (including relational examples) in Section 8.

Remark: In this work, we assume that `query` is always defined: we don't consider programs "observing" events with zero probability. We make this simplification to focus here on program verification without the need to reason about whether a `query` statement is defined or not. This approach was used, for example, in Barthe et al. [2016a] to reason about differential privacy for Bayesian processes. We believe that the problem of identifying ways to reason about when a `query` statement is defined is an important one, but it is orthogonal to the formal reasoning we consider here. Other work has focused on this problem [Borgström et al. 2011; Heunen et al. 2017; Ścibior et al. 2017; Shan and Ramsey 2017]. In a similar way, we consider only programs that terminate, without stipulating a specific method to prove termination.

3 HPPROG: A HIGHER-ORDER PROBABILISTIC PROGRAMMING LANGUAGE

We present the probabilistic language HPProg we use in this paper. The language is an extension of the simply-typed lambda calculus with products, coproducts, natural numbers, lists, (terminating)

recursion and *the monadic type for probability*. The types of HPProg are defined by the following grammar.

$$\begin{array}{lcll} \tilde{\tau} & ::= & \mathtt{unit} \mid \mathtt{bool} \mid \mathtt{nat} \mid \mathtt{real} \mid \mathtt{pReal} \mid \tilde{\tau} \times \tilde{\tau} \mid \mathtt{list}(\tilde{\tau}) & \text{(Basic Types)} \\ \tau & ::= & \tilde{\tau} \mid M[\tau] \mid \tau \to \tau \mid \tau \times \tau \mid \mathtt{list}(\tau) & \text{(Types)} \end{array}$$

We distinguish two sorts of types: Basic Types and Types. The former, as the name suggests, include standard basic types (where `pReal` is the type of positive real numbers), and products and lists of them. The latter include a monadic type $M[\tau]$ for general measures on τ, as well as function and product types. As we will see later in Section 7, Basic Types will be interpreted in standard Borel spaces, while for general Types we will need quasi-Borel spaces. The language of HPProg expressions is defined by the following grammar.

$$\begin{array}{lcl} e & ::= & x \mid c \mid f \mid e\ e \mid \lambda x.e \mid \langle e, e\rangle \mid \pi_i(e) \mid \mathtt{case}\ e\ \mathtt{with}\ [d_i \overline{x_i} \Rightarrow e_i]_i \mid \mathtt{letrec}\ f x = e \\ & & \mid \mathtt{return}\ e \mid \mathtt{bind}\ e\ e \mid \mathtt{query}\ e\ \Rightarrow\ e \mid \mathtt{Uniform}(e, e) \mid \mathtt{Bern}(e) \mid \mathtt{Gauss}(e, e) \end{array}$$

Most of the constructs are standard. We use c to range over a set of basic constants and f to range over a set of primitive functions. We have monadic constructions `return` e and `bind` e_1 e_2 for the monadic type $M[\tau]$, a conditioning construction `query` $e_1 \Rightarrow e_2$ for computing the posterior distribution given a prior distribution e_1, and a likelihood function e_2, and primitives representing basic probability distributions.

HPProg expressions are simply typed, using rules that are mostly standard. We show only selected rules here:

$$\frac{\Gamma \vdash e : M[\tau] \qquad \Gamma \vdash e' : \tau \to \mathtt{pReal}}{\Gamma \vdash \mathtt{query}\ e\ \Rightarrow\ e' : M[\tau]} \qquad \frac{\Gamma \vdash e : M[\tau] \qquad \Gamma \vdash e' : \tau \to M[\tau']}{\Gamma \vdash \mathtt{bind}\, e\, e' : M[\tau']}$$

$$\frac{\Gamma, f : I \to \sigma, x : I \vdash e : \sigma \qquad I \in \{\mathtt{nat}, \mathtt{list}(\tau)\} \qquad \mathit{Terminate}(f, x, e)}{\Gamma \vdash \mathtt{letrec}\, f x = e : I \to \sigma}$$

Here, $\mathit{Terminate}(f, x, e)$ is any termination criterion which ensures that all recursive calls are on smaller arguments. We also consider a basic equational theory for expressions based on β-reduction, extensionality and monadic rules. These are also standard and we omit them here. We enrich this equational theory with axioms and equations reflecting common reasoning principles for probabilistic programming in Section 5.

For convenience, we use some syntactic sugar: $(\mathtt{let}\, x\ =\ e_1\ \mathtt{in}\ e_2) \equiv (\lambda x.\ e_2)e_1$, $(\mathtt{mlet}\, x\ =\ e_1\ \mathtt{in}\ e_2) \equiv \mathtt{bind}\, e_1\ (\lambda x.\ e_2)$, and $e_1 \otimes e_2 \equiv \mathtt{bind}\, e_1\ (\lambda x.\ \mathtt{bind}\, e_2\ (\lambda y.\, \mathtt{return}\langle x, y\rangle))$. Thanks to the commutativity of M (the Fubini-Tonelli equality; see Section 5), the semantics of $e_1 \otimes e_2$ is exactly the *product measure* of e_1 and e_2.

4 PL: A LOGIC FOR PROBABILISTIC PROGRAMS

In this section we introduce a logic, named PL, for reasoning about higher-order probabilistic programs. This logic forms the basis of PPV. PL contains basic predicates over expressions of HPProg. To support more natural verification in PPV, we enrich PL with a set of axioms encompassing a wide variety of reasoning principles over probabilistic programs.

PL follows the style of higher-order simple predicate logic (HOL) [Jacobs and Melham 1993], where quantified variables can be of arbitrary types, but extends HOL with assertions about probabilistic constructions. Terms and formulas of PL are defined by the following grammar:

$$\begin{array}{lcll} t & ::= & e \mid \mathbb{E}_{x \sim t}[t(x)] \mid \mathtt{scale}(t, t) \mid \mathtt{normalize}(t) & \text{(enriched expressions)} \\ \phi & ::= & (t = t) \mid (t < t) \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \implies \phi \mid \neg\phi \mid \forall x : \tau.\phi \mid \exists x : \tau.\phi & \text{(logical formulas)} \end{array}$$

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash t' : \tau \quad t =_{\beta\iota\mu} t'}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} t = t'}\ [\mathrm{CONV}] \qquad \frac{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[t/x] \quad \Gamma \mid \Psi \vdash_{\mathrm{PL}} t = u}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[u/x]}\ [\mathrm{SUBST}]$$

$$\frac{\phi \in \Psi}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi}\ [\mathrm{AX}] \qquad \frac{\Gamma \mid \Psi, \psi \vdash_{\mathrm{PL}} \phi}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \psi \implies \phi}\ [\Rightarrow_I] \qquad \frac{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \psi \implies \phi \quad \Gamma \mid \Psi \vdash_{\mathrm{PL}} \psi}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi}\ [\Rightarrow_E]$$

Fig. 1. Selection of rules for the PL logic.

Enriched expressions enrich HPProg expressions e with constructions for expectations $\mathbb{E}_{x\sim t}[t(x)]$, rescaling of measures $\mathtt{scale}(t, t)$, and normalization $\mathtt{normalize}(t)$. A logical formula ϕ is a formula built over equalities and inequalities between enriched expressions.

Similar to expressions in HPProg, we consider only well-typed enriched expressions. Typing rules for the additional constructs of PL are the following.

$$\frac{\Gamma \vdash t_1 : M[\tau] \quad \Gamma \vdash t_2 : \tau \to \mathtt{pReal}}{\Gamma \vdash \mathbb{E}_{x\sim t_1}[t_2(x)] : \mathtt{pReal}} \qquad \frac{\Gamma \vdash t_1 : M[\tau] \quad \Gamma \vdash t_2 : \tau \to \mathtt{pReal}}{\Gamma \vdash \mathtt{scale}(t_1, t_2) : M[\tau]} \qquad \frac{\Gamma \vdash t : M[\tau]}{\Gamma \vdash \mathtt{normalize}(t) : M[\tau]}$$

Intuitively, $\mathbb{E}_{x\sim t_1}[t_2(x)]$ is the expected value of the function t_2 over the distribution t_1; $\mathtt{scale}(t_1, t_2)$ is a distribution obtained from an underlying measure t_1 by rescaling its components by means of the density function t_2; $\mathtt{normalize}(t)$ is the normalization of a measure t to a probability distribution (a measure with mass 1). Expectations of real-valued functions are defined by the difference of positive and negative parts. Precisely, for given $\Gamma \vdash t_1 : M[\tau]$ and $\Gamma \vdash t_2 : \tau \to \mathtt{real}$, we define the expectation as the following syntactic sugar:[2]

$$\mathbb{E}_{x\sim t_1}[t_2(x)] \equiv \mathbb{E}_{x\sim t_1}[\mathtt{if}\ t_2(x) > 0\ \mathtt{then}\ |t_2(x)|\ \mathtt{else}\ 0] - \mathbb{E}_{x\sim t_1}[\mathtt{if}\ t_2(x) < 0\ \mathtt{then}\ |t_2(x)|\ \mathtt{else}\ 0]$$

We can also define variance and probability in terms of expectation:

$$\Pr_{x\sim e}[e'] \equiv \mathbb{E}_{x\sim e}[\mathtt{if}\ e'\ \mathtt{then}\ 1\ \mathtt{else}\ 0] \qquad \mathrm{Var}_{x\sim e_1}[e_2] \equiv \mathbb{E}_{x\sim e_1}[(e_2)^2] - (\mathbb{E}_{x\sim e_1}[e_2])^2$$

A PL judgment has the form $\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi$ where Γ is a context assigning types to variables, Ψ is a set of formulas well-formed in the context Γ, and ϕ is a formula also well-formed in Γ. Rules to derive well-formedness judgments $\Gamma \vdash \phi\ \mathrm{wf}$ are rather standard and we omit them here. We often refer to Ψ as the *precondition*. The proof rules of PL are rather standard, so we show only a selection in Figure 1.

We extend equational rules and axioms for standard expressions to enriched expressions. We also introduce some axioms specific to enriched expressions in Section 5.

5 AXIOMS AND EQUATIONS OF ASSERTIONS FOR STATISTICS

In this section, we introduce axioms and equations for the logic PL. First, we have the standard equational theory for (enriched) expressions covering α-conversion, β-reduction, extensionality, and the monadic rules of the monadic type M. We omit these standard rules here. The monadic type M also has commutativity (the Fubini-Tonelli equality), represented by the following equation:

$$(\mathtt{bind}\, e_1\ \lambda x.(\mathtt{bind}\, e_2\ \lambda y.e(x, y))) = (\mathtt{bind}\, e_2\ \lambda y.(\mathtt{bind}\, e_1\ \lambda x.e(x, y)) \quad (x, y\mathrm{fresh}) \tag{3}$$

We introduce some equalities pertaining to expectation. We have the monotonicity and linearity of expectation (axioms 4, 5), and we also have the Cauchy-Schwarz inequality (axiom 6). Finally,

[2]We use absolute values $|-| : \mathtt{real} \to \mathtt{pReal}$ to adjust the typing. The right-hand side is undefined if both expectations are infinity. We could avoid this kind of undefinedness by stipulating $\infty - \infty = -\infty$, but we leave it undefined since this actually never shows up in our concrete examples.

we can transform variables in expressions related to expectation by substitution (axiom 7).

$$(\forall x\colon \tau.\ e' \geq 0) \implies \mathbb{E}_{x\sim e}[e'] \geq 0 \tag{4}$$

$$\mathbb{E}_{x\sim e}[e_1 * e_2] = e_1 * \mathbb{E}_{x\sim e}[e_2] \quad (x \notin \mathrm{FV}(e_1)), \qquad \mathbb{E}_{x\sim e}[e_1 + e_2] = \mathbb{E}_{x\sim e}[e_1] + \mathbb{E}_{x\sim e}[e_2] \tag{5}$$

$$(\mathbb{E}_{x\sim e}[e_1 * e_2])^2 \leq \mathbb{E}_{x\sim e}[e_1^2] * \mathbb{E}_{x\sim e}[e_2^2] \tag{6}$$

$$\mathbb{E}_{x\sim \mathtt{bind}\, e\ \lambda y.\, \mathtt{return}(e')}[e''] = \mathbb{E}_{y\sim e}[e''[e'/x]] \tag{7}$$

We also introduce some basic equalities pertaining to observation, rescaling, and normalization.

$$\mathbb{E}_{x\sim d'}[h(x)\cdot g(x)] = \mathbb{E}_{x\sim \mathtt{scale}(d',g)}[h(x)]. \tag{8}$$

$$(\mathtt{scale}(\mathtt{scale}(e_1, e_2),\ e_3) = (\mathtt{scale}(e_1, \lambda x.(e_2(x) * e_3(x))), \quad e = \mathtt{scale}(e, \lambda\_.1) \tag{9}$$

$$(\mathtt{mlet}\, x = \mathtt{scale}(e_1, e_2)\ \mathtt{in}\ e_3(x)) = (\mathtt{mlet}\, x = e_1\ \mathtt{in}\ \mathtt{scale}(e_3(x), \lambda u.e_2(x))) \tag{10}$$

$$\mathtt{scale}(e_1, e_2) \otimes \mathtt{scale}(e_3, e_4) = \mathtt{scale}(e_1 \otimes e_2, \lambda w.e_2(\pi_1(w)) * e_4(\pi_2(w))) \tag{11}$$

$$\mathbb{E}_{y\sim e}[1] < \infty \implies (\mathtt{bind}\, e'\ \lambda x.e) = \mathtt{scale}(e, \mathbb{E}_{y\sim e'}[1]) \quad (x \notin \mathrm{FV}(e)) \tag{12}$$

$$(\mathtt{query}\, e_1 \Rightarrow e_2) = \mathtt{normalize}(\mathtt{scale}(e_1, e_2)) \tag{13}$$

$$\mathtt{normalize}(e) = \mathtt{scale}(e, \lambda u.1/\mathbb{E}_{x\sim e}[1]) \quad (u \notin \mathrm{FV}(\mathbb{E}_{x\sim e}[1])) \tag{14}$$

$$0 < \alpha < \infty \implies \mathtt{normalize}(\mathtt{scale}(e_1, e_2)) = \mathtt{normalize}(\mathtt{scale}(e_1, \alpha * e_2)) \tag{15}$$

We could also introduce the axioms for particular distributions such as $\mathbb{E}_{x\sim \mathtt{Bern}(e)}[\mathtt{if}\ x\ \mathtt{then}\ 1\ \mathtt{else}\ 0] = e$ $(0 \leq e \leq 1)$ and $\mathbb{E}_{x\sim \mathtt{Gauss}(e_1,e_2)}[x] = e_1$, but we do not do this here.

5.1 Markov and Chebyshev Inequalities

The axioms we introduced above are not only very basic but also very expressive. For instance, we can prove the Markov inequality (16) and the Chebyshev inequality (17) in PL using these axioms.

$$d\colon M[\mathtt{real}],\ a\colon \mathtt{real} \vdash_{\mathrm{PL}} (a > 0) \implies \Pr_{x\sim d}[|x| \geq a] \leq \mathbb{E}_{x\sim d}[|x|]/a. \tag{16}$$

$$\begin{aligned} d\colon M[\mathtt{real}], b\colon \mathtt{real}, \mu\colon \mathtt{real} \quad &\vdash_{\mathrm{PL}} \mathbb{E}_{x\sim d}[1] = 1 \wedge \mu = \mathbb{E}_{x\sim d}[x] \wedge b^2 > 0 \\ &\implies \Pr_{x\sim d}[|x-\mu| \geq b] \leq \mathrm{Var}_{x\sim d}[x]/b^2. \end{aligned} \tag{17}$$

6 UNARY/RELATIONAL LOGIC

In this section, we introduce two specializations of PL. The first one, UPL, is a unary logic to verify *unary properties* of probabilistic programs. More concretely, UPL can be considered as a collection of inference rules derivable in PL specialized in proving formulas of form $\phi(e)$ by following the syntactic structure of the distinguished subterm e rather than the syntactic structure of ϕ itself.

The second one, RPL, is a relational logic to verify *relational properties* of probabilistic programs. Similarly, it can be seen as a collection of inference rules derivable in PL to prove formulas of the form $\phi(e_1, e_2)$ by following the syntactic structure of e_1 and e_2.

6.1 The Unary Logic UPL

Judgments in the unary logic UPL have the shape $\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e\colon \tau \mid \phi$ where Γ is a context, Ψ is a set of assertions on the context variables, e is a HPProg expression, τ a type, and ϕ is an assertion (possibly) containing a distinguished variable $\mathbf{r}$ of type τ which is used to refer to the expression e in the formula ϕ.

We give in Figure 2 a selection of proof rules in UPL. We have two groups of rules, one for pure computations and the other for probabilistic computations. The rules are mostly *syntax-directed*, with the exception of the rule [u-SUB]. We present a selection of the pure rules, the rest of them are as in UHOL [Aguirre et al. 2017]. The rule [u-ABS] turns an assertion about the bound variable

Rules for pure constructions.

$$\frac{\Gamma \vdash x : \tau \quad \Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[x/\mathbf{r}]}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} x : \tau \mid \phi}\ [\text{u-VAR}] \qquad \frac{\Gamma, x : \tau \mid \Psi, \phi' \vdash_{\mathrm{UPL}} t : \sigma \mid \phi}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \lambda x : \tau.t : \tau \to \sigma \mid \forall x.\phi' \Longrightarrow \phi[\mathbf{r}x/\mathbf{r}]}\ [\text{u-ABS}]$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} t : \sigma \mid \phi' \quad \Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi'[t/\mathbf{r}] \implies \phi[t/\mathbf{r}]}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} t : \sigma \mid \phi}\ [\text{u-SUB}]$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} t : \tau \to \sigma \mid \forall x.\phi' \Longrightarrow \phi[\mathbf{r}x/\mathbf{r}] \quad \Gamma \mid \Psi \vdash_{\mathrm{UPL}} u : \tau \mid \phi'}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} tu : \sigma \mid \phi[u/x]}\ [\text{u-APP}]$$

$$\frac{\begin{array}{c}\mathit{Terminate}(f, x, e) \\ \Gamma, x : I, f : I \to \sigma \mid \Psi, \phi', \forall m.|m| < |x| \implies \phi'[m/x] \implies \phi[m/x][fm/\mathbf{r}] \vdash_{\mathrm{UPL}} e : \sigma \mid \phi\end{array}}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{letrec}\, fx = e : I \to \sigma \mid \forall x.\phi' \implies \phi[\mathbf{r}x/\mathbf{r}]}\ [\text{u-LETREC}]$$

Rules for probabilistic constructions.

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e : \tau \mid \phi[\mathtt{return}(\mathbf{r})/\mathbf{r}]}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{return}(e) : M[\tau] \mid \phi}\ [\text{u-RET}]$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e : M[\tau_1] \mid \phi_1 \quad \Gamma \mid \Psi \vdash_{\mathrm{UPL}} e' : \tau_1 \to M[\tau_2] \mid \forall s : M[\tau_1].(\phi_1[s/\mathbf{r}] \Longrightarrow \phi_2[\mathtt{bind}\, s\, \mathbf{r}/\mathbf{r}])}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{bind}\, e\, e' : M[\tau_2] \mid \phi_2}\ [\text{u-BIND}]$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e : M[\tau] \mid \phi_1 \quad \Gamma \mid \Psi \vdash_{\mathrm{UPL}} e' : \tau \to \mathtt{pReal} \mid \forall s : M[\tau].(\phi_1[s/\mathbf{r}] \Longrightarrow \phi_2[\mathtt{query}\, s \Rightarrow \mathbf{r}/\mathbf{r}])}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{query}\, e \Rightarrow e' : M[\tau] \mid \phi_2}\ [\text{u-QRY}]$$

Fig. 2. A selection of UPL rules.

into a precondition of its lambda abstraction. The rule [u-APP] proves a postcondition of a function application provided that the argument satisfies the precondition of the function. The rule [u-LETREC] allows proving properties of terminating recursive functions by introducing an induction hypothesis in the context.

In the case of monadic computations, we have rules for monadic return, bind and `query`. It is worth noticing that in the second premise of both the rules [u-BIND] and [u-QRY], the assertion quantifies over elements in $M[\tau_1]$, while the input type of the function is just τ_1. This follows the spirit of the interpretation (see Section 7), where the Kleisli lifting $(-)^{\#}$ is used to lift a function $\tau_1 \to M[\tau_2]$ to a function $M[\tau_1] \to M[\tau_2]$. The quantification over distributions, rather than over elements, is essential to establish a connection with the assertion on the first premise. This will be useful to simplify the verification of our examples.

We can prove that, despite being syntax directed, UPL does not lose expressiveness relative to PL: The following theorem shows that the unary logic UPL is sound and complete with respect to the underlying logic PL.

Theorem 6.1 (Equi-derivability of PL and UPL). *The judgment* $\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[e/\mathbf{r}]$ *is derivable if and only if the judgment* $\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e : \tau \mid \phi$ *is derivable.*

6.2 The Relational Logic RPL

Judgments in the relational logic RPL have the shape $\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e_1 : \tau_1 \sim e_2 : \tau_2 \mid \phi$, where Γ is a context, Ψ is a set of assertions on the context, e_1 and e_2 are HPProg expressions, τ_1 and τ_2 are types, and ϕ is an assertion (possibly) containing two distinguished variables $\mathbf{r}_1$ of type τ_1 and $\mathbf{r}_2$ of type τ_2 which are used to refer to the expressions e_1 and e_2 in the formula ϕ. We give in Figure 3 a selection of proof rules in RPL. We present three groups of rules. The first group consists of relational rules for pure computations. The second group consists of two-sided relational rules for

Relational rules for pure constructions - two-sided

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_1 \mid \Psi, \phi' \vdash_{\mathrm{RPL}} t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} \lambda x_1.t_1 : \tau_1 \to \sigma_1 \sim \lambda x_2.t_2 : \tau_2 \to \sigma_2 \mid \forall x_1.\forall x_2 \phi' \implies \phi[\mathbf{r}_1 x_1/\mathbf{r}_1, \mathbf{r}_2 x_2/\mathbf{r}_2]} \text{ [r-ABS]}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi' \quad \Gamma \mid \Psi \vdash_{\mathrm{RPL}} t_1 : \sigma_1 \to \tau_1 \sim t_2 : \sigma_2 \to \tau_2 \mid \forall x_1.\forall x_2.\phi'[x_1/\mathbf{r}_1, x_2/\mathbf{r}_2] \implies \phi[\mathbf{r}_1 x_1/\mathbf{r}_1, \mathbf{r}_2 x_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} t_1 u_1 : \sigma_1 \sim t_2 u_2 : \sigma_2 \mid \phi} \text{ [r-APP]}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \quad \Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi'[t_1/\mathbf{r}_1, t_2/\mathbf{r}_2] \implies \phi[t_1/\mathbf{r}_1, t_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi} \text{ [r-SUB]}$$

Relational rules for probabilistic constructions - two-sided

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 : \tau_1 \sim e_2 : \tau_2 \mid \phi[\mathtt{return}(\mathbf{r}_1)/\mathbf{r}_1,\ \mathtt{return}(\mathbf{r}_2)/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}}\ \mathtt{return}(e_1) : M[\tau_1] \sim \mathtt{return}(e_2) : M[\tau_2] \mid \phi} \text{ [r-RET]}$$

$$\frac{\begin{array}{c}\phi = \forall s_1 : M[\tau_1].\forall s_2 : M[\tau_2].\ (\phi_1[s_1/\mathbf{r}_1, s_2/\mathbf{r}_2] \implies \phi_2[\mathtt{bind}\, s_1\ \mathbf{r}_1/\mathbf{r}_1, \mathtt{bind}\, s_2\ \mathbf{r}_2/\mathbf{r}_2]) \\ \Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 : M[\tau_1] \sim e_2 : M[\tau_2] \mid \phi_1 \quad \Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1' : \tau_1 \to M[\tau_3] \sim e_2' : \tau_2 \to M[\tau_4] \mid \phi\end{array}}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} \mathtt{bind}\, e_1\ e_1' : M[\tau_3] \sim \mathtt{bind}\, e_2\ e_2' : M[\tau_4] \mid \phi_2} \text{ [r-BIND]}$$

$$\frac{\begin{array}{c}\phi = \forall s_1 : M[\tau_1], s_2 : M[\tau_2].\phi_1[s_1/\mathbf{r}_1, s_2/\mathbf{r}_2] \implies \phi_2[\mathtt{query}\, s_1 \Rightarrow \mathbf{r}_1/\mathbf{r}_1, \mathtt{query}\, s_2 \Rightarrow \mathbf{r}_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 : M[\tau_1] \sim e_2 : M[\tau_2] \mid \phi_1 \quad \Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1' : \tau_1 \to \mathtt{pReal} \sim e_2' : \tau_2 \to \mathtt{pReal} \mid \phi\end{array}}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} \mathtt{query}\, e_1 \Rightarrow e_1' : M[\tau_1] \sim \mathtt{query}\, e_2 \Rightarrow e_2' : M[\tau_2] \mid \phi_2} \text{ [r-QRY]}$$

Relational rules for probabilistic constructions - one-sided

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 : \tau_1 \sim e_2 : \tau_2 \mid \phi[\mathtt{return}(\mathbf{r}_1)/\mathbf{r}_1]}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}}\ \mathtt{return}(e_1) : M[\tau_1] \sim e_2 : \tau_2 \mid \phi} \text{ [r-RET-L]}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e_1 : M[\tau_1] \mid \phi_1 \\ \Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1' : \tau_1 \to M[\tau_3] \sim e_2 : M[\tau_2] \mid \forall s_1 : M[\tau_1].\phi_1[s_1/\mathbf{r}] \implies \phi_2[\mathtt{bind}\, s_1\ \mathtt{in}\ \mathbf{r}_1/\mathbf{r}_1]\end{array}}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} \mathtt{bind}\, e_1\ e_1' : M[\tau_3] \sim e_2 : M[\tau_2] \mid \phi_2} \text{ [r-BIND-L]}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e_1 : M[\tau_1] \mid \phi_1 \\ \Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1' : \tau_1 \to \mathtt{pReal} \sim e_2 : M[\tau_2] \mid \forall s_1 : M[\tau_1].\phi_1[s_1/\mathbf{r}_1] \implies \phi_2[\mathtt{query}\, s_1 \Rightarrow \mathbf{r}_1/\mathbf{r}_1]\end{array}}{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} \mathtt{query}\, e_1 \Rightarrow e_1' : M[\tau_1] \sim e_2 \mid \phi_2} \text{ [r-QRY-L]}$$

Fig. 3. A selection of RPL rules.

probabilistic computations, meaning that the terms on both sides of the judgment have the same top-level constructor. Finally, the third group consists of one-sided relational rules for probabilistic computations, meaning that one of terms has a specific top-level constructor while the other is arbitrary (not analyzed by the rule). Here we show just the left-sided rules that have the constructor on the left; right-sided rules are symmetrical. As in the unary case, we use an approach that is mostly *syntax-directed* except for the [r-SUB] rule.

The rules for pure computations are similar to the ones from RHOL [Aguirre et al. 2017] and we present only a selection. For the probabilistic constructions, we have relational rules for the monadic return and bind, and for `query`. These rules are the natural generalization of the unary rules to the relational case. In particular, in all the rules for `bind` and `query` we use assertions quantifying over distributions, similarly to what we have in UPL, to establish a connection between the different assertions.

The equi-derivability result for UPL can be lifted to the relational setting: RPL is also sound and complete with respect to the logic PL.

Theorem 6.2 (Equi-derivability of PL and RPL). *The judgment* $\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[e_1/\mathbf{r}_1,\ e_2/\mathbf{r}_2]$ *is derivable if and only if* $\Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 \colon \tau_1 \sim e_2 \colon \tau_2 \mid \phi$ *is derivable.*

A comment on product types and RPL . One could effectively embed the whole of RPL into UPL by replicating the set of rules of RPL as UPL rules for every possible product type, and rewriting the distinguished $\mathbf{r}_1, \mathbf{r}_2$ in the refinements to $\pi_1(\mathbf{r}), \pi_2(\mathbf{r})$. For instance, the two-sided [r-ABS] rule would be rewritten as a UPL rule [u-ABSxABS] for the product of two arrow types. Similarly, all the one-sided rules would be written as unary rules directed by only one side of the product type, while ignoring the other.

However, we believe that this style of presentation would be significantly more cumbersome, and, moreover, it would hide the fact that we are trying to express and prove a relational property of two programs that execute independently.

6.3 Special Rules

As already discussed in the introduction, we enrich PPV with special rules that can ease verification. One example is the use the following Bayesian law expressing a general fact about the way we can reason about probabilistic inference when the observation is a boolean:

$$\frac{\Gamma, x \colon \tau \vdash e' \colon \mathtt{bool} \quad \Gamma, x \colon \tau \vdash e'' \colon \mathtt{bool} \quad \Gamma \vdash e \colon M[\tau]}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{query}\ e \Rightarrow \lambda x.(\mathtt{if}\ e'\ \mathtt{then}\ 1\ \mathtt{else}\ 0) \colon M[\tau] \mid \Pr_{y \sim \mathbf{r}}[e''[y/x]] = \frac{\Pr_{x \sim e}[e' \wedge e'']}{\Pr_{x \sim e}[e']}}\ [\text{Bayes}]$$

This rule can be derived by first using [u-QRY], and then reasoning in PL through the [u-SUB] rule, which is why the premises are just simply typed assumptions. In particular, in PL we use the characterization of `query` given in Section 5.

We also introduce a [LET] rule, which can be derived by desugaring the `let` notation:

$$\frac{\Gamma \mid \Psi \vdash e \colon \tau_1 \mid \phi_1 \quad \Gamma, x : \tau_1 \mid \Psi, \phi_1[x/\mathbf{r}] \vdash e' \colon \tau_2 \mid \phi_2}{\Gamma \mid \Psi \vdash \mathtt{let}\ x = e\ \mathtt{in}\ e' \colon \tau_2 \mid \phi_2}\ [\text{LET}]$$

Notice that Theorem 6.1 can be used to convert UPL derivation trees into PL ones and vice versa. Similarly, Theorem 6.2 is used to convert RPL proofs to PLproofs. These conversions are useful to switch between the different levels of our system and to reason in whichever one is more convenient. To this end, we introduce the following *admissible* rules:

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e \colon \tau \mid \phi}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[e/\mathbf{r}]}\ [\text{conv-UPL}] \qquad \frac{\Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 \colon \tau_1 \sim e_2 \colon \tau_2 \mid \phi}{\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi[e_1/\mathbf{r}_1,\ e_2/\mathbf{r}_2]}\ [\text{conv-RPL}]$$

7 SEMANTICS

7.1 Background

In this section we present the semantic foundation of PPV. We start by recalling the definition of quasi-Borel spaces [Heunen et al. 2017] and by showing how we can use them to define monads for probability measures [Ścibior et al. 2017]. We use these constructions in the next section to give the semantics of programs on which we will build our logic.

Quasi-Borel Spaces. We introduce here the category **QBS** of *quasi-Borel spaces*. Intuitively, the category **QBS** is a relaxation of the category **Meas** of measurable spaces. **QBS** has a nice categorical structure—it is Cartesian closed and retains important properties coming from measure theory. Before introducing quasi-Borel spaces, we fix some notation. We use $\mathbb{R}$ to denote the real line

equipped with the standard Borel algebra. We use $\coprod_{i\in\mathbb{N}} S_i$ to denote the coproduct of a countable family of sets $\{S_i\}_{i\in\mathbb{N}}$, and $[\alpha_i]_{i\in\mathbb{N}}$ for the copairing of functions α_i for $i \in \mathbb{N}$.

Definition 7.1 (Heunen et al. [2017]). The category **QBS** is the category of quasi-Borel spaces and morphisms between them, where a *quasi-Borel space* (X, M_X) (with respect to $\mathbb{R}$) is a set X equipped with a subset M_X of functions in $\mathbb{R} \to X$ such that (1) If $\alpha\colon \mathbb{R} \to X$ is constant then $\alpha \in M_X$. (2) If $\alpha \in M_X$ and $f\colon \mathbb{R} \to \mathbb{R}$ is measurable then $\alpha \circ f \in M_X$. (3) If the family $\{S_i\}_{i\in\mathbb{N}}$ is a countable partition of $\mathbb{R}$, i.e. $\mathbb{R} = \coprod_{i\in\mathbb{N}} S_i$, with each set S_i Borel, and if $\alpha_i \in M_X$ $(\forall i \in \mathbb{N})$ then the copairing $[\alpha_i|_{S_i}]_{i\in\mathbb{N}}$ of $\alpha_i|_{S_i} : S_i \to X$ belongs to M_X.

A morphism from a quasi-Borel space (X, M_X) to a quasi-Borel space (Y, M_Y) is a function $f\colon X \to Y$ such that $f \circ \alpha \in M_Y$ holds for any $\alpha \in M_X$.

As shown by Heunen et al. [2017], the category **QBS** has a convenient structure to interpret probabilistic programs. That is, it is well-pointed and Cartesian closed and we have the usual structure for currying and uncurrying functions; it has products and coproducts with distributivity between them; every standard Borel space Ω can be converted to a quasi-Borel space and every measurable function $f\colon \Omega_1 \to \Omega_2$ is a morphism $f\colon \Omega_1 \to \Omega_2$ in **QBS**. Hence, a **QBS** can be used to interpret a probabilistic functional language. See Heunen et al. [2017]; Ścibior et al. [2017] for more details.

The category **QBS** has also a convenient structure to *reason* about probabilistic programs. In particular, the forgetful functor $|-|\colon \mathbf{QBS} \to \mathbf{Set}$ erasing the quasi-Borel structure does not change the underlying structure of functions. This property is fundamental for the design of the category **Pred**(**QBS**) of predicates over quasi-Borel spaces.

Measures on quasi-Borel spaces. Quasi-Borel spaces were introduced to support measure theory in a Cartesian closed category. In particular, given a measure on some standard Borel space Ω we can define a measure over quasi-Borel spaces.

Definition 7.2 (Ścibior et al. [2017]). A measure on a quasi-Borel space (X, M_X) is a triple (Ω, α, ν) where Ω is a standard Borel space, $\alpha\colon \Omega \to X$ is a morphism in **QBS**, and ν is a σ-finite measure over Ω.

For a measure $\mu = (\Omega, \alpha, \nu)$ on X and a function $f : X \to \mathbb{R}$ in **QBS**, we define integration over quasi-Borel spaces in terms of integration over Borel spaces: $\int_X f\, d\mu \stackrel{\text{def}}{=} \int_\Omega (f \circ \alpha)\, d\nu$. Equivalence of measures in **QBS** is defined in terms of equality of integrations:

$$(\Omega, \alpha, \nu) \approx (\Omega', \alpha', \nu') \stackrel{\text{def}}{=} \forall f : X \to \mathbb{R} \text{ in } \mathbf{QBS}.\ \int_\Omega (f \circ \alpha)\, d\nu = \int_{\Omega'} (f \circ \alpha')\, d\nu'.$$

In the following, it will be convenient to work with equivalence classes of measures which we denote by $[\Omega, \alpha, \nu]$. Every equivalence class for a measure (Ω, α, ν) also contains a measure over $\mathbb{R}$ defined in the appropriate way [Heunen et al. 2017]. We are now ready to define a monad for measures.

Definition 7.3 (Ścibior et al. [2017]). The monad of σ-finite measures $\mathfrak{M}$ is defined as follows.

- For any X in **QBS**, $\mathfrak{M}X$ is the set of equivalence classes of σ-finite measures equipped with the quasi-Borel structure given by the following definition

$$M_{\mathfrak{M}X} = \left\{ \lambda r.[D_r, \alpha(r,-), \mu_r] \;\middle|\; \begin{array}{l} D \subseteq_{\text{measurable}} \mathbb{R} \times \Omega,\ \mu\colon \sigma\text{-finite measure on } \Omega, \\ \alpha\colon D \to X,\ D_r = \{\, \omega \mid (r,\omega) \in D \,\},\ \mu_r = \mu|_{D_r} \end{array} \right\}$$

- The unit $\eta_X\colon X \to \mathfrak{M}X$ is defined by $\eta_X(x) = [1, \lambda * .\ x, \mathbf{d}_*]$.
- The Kleisli lifting is defined for any $f\colon X \to \mathfrak{M}Y$ and $[\Omega, \alpha, \nu] \in \mathfrak{M}X$ as

$$f^\sharp[\Omega, \alpha, \nu] = [D, \beta, (\nu \otimes \nu')|_D]$$

where $D = \{(r, \omega) \mid \omega \in D_r\}$ and $\beta(-) = \lambda r.\beta(r, -)$ are defined for every $\gamma : \Omega \to \mathbb{R}$ and $\gamma^* : \mathbb{R} \to \Omega$ satisfying $\gamma^* \circ \gamma = \mathrm{id}_\Omega$ through $(f \circ \alpha)(\gamma^*(r)) = [D_r, \beta(r, -), \nu']$.

Let us unpack in part this definition. The set of functions $M_{\mathfrak{M}X}$ can be seen as a set of (uncountable) families of measures, indexed by r, supporting infinite measures. The Kleisli lifting uses the fact that each $(f \circ \alpha)(\gamma^*(-))$ is a function in $M_{\mathfrak{M}Y}$, that D built as a product measure starting from r and D_r is measurable, and β is a morphism from D to Y.

Thanks to the Fubini-Tonelli theorem, the monad $\mathfrak{M}$ on **QBS** is commutative strong with respect to Cartesian products. We can also use the structure of **QBS** to define the product measure of $[\Omega, \alpha, \nu]$ and $[\Omega', \alpha', \nu']$ as $[(\Omega \times \Omega'), (\alpha \times \alpha'), (\nu \otimes \nu')]$. Using the isomorphism $\mathfrak{M}1 \cong [0, \infty]$, usual integration $\int f \, d\mu$ for $f : \mathbb{R} \to [0, \infty]$ and $\mu \in \mathfrak{M}(\mathbb{R})$ corresponds to $f^\sharp(\mu)$. We can define the *mass* $|\mu|$ of measure $\mu = [\Omega, \alpha, \nu]$ by $\int_X 1 d\mu$ which is the same as the mass $|\nu|$ of base measure ν. The monad $\mathfrak{M}$ captures general measures. For example, we can define a *null measure* as $\mathbf{0} = [\Omega, \alpha, 0]$.

In the sequel, we will also use a commutative monad $\mathfrak{P}$ on **QBS** obtained by restricting the monad $\mathfrak{M}$ to *subprobability* measures. We have the canonical inclusion $\mathfrak{P}X \subseteq \mathfrak{M}X$.

7.2 Semantics for PPV

In order to give meaning to the logical formulas of PL, we first need to give meaning to expressions in HPProg and to enriched expressions in PL. We do this by interpreting types as **QBS** objects as shown below:

$$[\![\mathtt{unit}]\!] \stackrel{\text{def}}{=} 1, \quad [\![\mathtt{bool}]\!] \stackrel{\text{def}}{=} 1 + 1, \quad [\![\mathtt{nat}]\!] \stackrel{\text{def}}{=} \mathbb{N}, \quad [\![\mathtt{real}]\!] \stackrel{\text{def}}{=} \mathbb{R}, \quad [\![\mathtt{pReal}]\!] \stackrel{\text{def}}{=} [0, \infty],$$
$$[\![\tau_1 \to \tau_2]\!] \stackrel{\text{def}}{=} [\![\tau_1]\!] \Rightarrow [\![\tau_2]\!], [\![\tau_1 \times \tau_2]\!] \stackrel{\text{def}}{=} [\![\tau_1]\!] \times [\![\tau_2]\!], [\![\mathtt{list}(\tau)]\!] \stackrel{\text{def}}{=} \coprod_{n \in \mathbb{N}} [\![\tau]\!]^n, [\![M[\tau]]\!] \stackrel{\text{def}}{=} \mathfrak{M}([\![\tau]\!])$$

where 1 is the terminal object in **QBS**; $\coprod_{n \in \mathbb{N}} [\![\tau]\!]^n$ is the coproduct of the countable family $[\![\tau]\!]^n = [\![\tau]\!] \times \cdots \times [\![\tau]\!]$ (n times); $([\![\tau_1]\!] \Rightarrow [\![\tau_2]\!])$ is the exponential object in **QBS**. We interpret each term $\Gamma \vdash e : \tau$ as a morphism $[\![\Gamma]\!] \to [\![\tau]\!]$ in **QBS**, where, as usual, the interpretation $[\![\Gamma]\!]$ of a context Γ is the product of the interpretations of its components. Pure computations are interpreted using the Cartesian closed structure of **QBS** where we can interpret recursive terms based on recursive data types ($I = \mathtt{list}(\tau), \mathtt{nat}$) by means of a fixed point operator fix iterating functions until termination. Since the termination criterion $\mathrm{Terminate}(f, x, e)$ ensures that all recursive calls are on smaller arguments, the operator fix is well-defined: for any $n \in I$ with $|n| < k$, $\mathrm{fix}(\lambda f.\lambda x.e)(n)$ is defined within k steps.

$$[\![\Gamma \vdash \mathtt{letrec}\ f x = e : I \to \sigma]\!] \stackrel{\text{def}}{=} \mathrm{fix}([\![\Gamma \vdash \lambda f : I \to \sigma.\lambda x : I.\ e : (I \to \sigma) \to (I \to \sigma)]\!])$$

We interpret `return` and `bind` using the structure of the monad $\mathfrak{M}$ of measures on **QBS**.

$$[\![\Gamma \vdash \mathtt{return}\ e : M[\tau]]\!] \stackrel{\text{def}}{=} \eta_{[\![\tau]\!]} \circ [\![\Gamma \vdash e : \tau]\!]$$
$$[\![\Gamma \vdash \mathtt{bind}\ e_1\ e_2 : M[\tau_2]]\!] \stackrel{\text{def}}{=} [\![\Gamma \vdash e_2 : \tau_1 \to M[\tau_2]]\!]^\sharp \circ \mathrm{st}_{[\![\Gamma]\!], [\![\tau_1]\!]}(\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \vdash e_1 : M[\tau_1]]\!] \rangle)$$

where η, $(-)^\sharp$, and st are the unit, the Kleisli lifting, and the tensorial strength of the commutative monad $\mathfrak{M}$. To interpret the other constructions we first introduce two semantics constructions for scaling and normalizing:

$$\mathrm{scale}(\nu, f) \stackrel{\text{def}}{=} (\mathfrak{M}(\pi_2) \circ \mathrm{dst}_{1,X} \circ \langle f, \eta_X \rangle)^\sharp(\nu). \qquad \mathrm{normalize}(\nu) \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & |\nu| = 0, \infty \\ \nu/|\nu| & \text{(otherwise)} \end{cases}.$$

where dst is the double strength of the commutative monad $\mathfrak{M}$, and $|\nu|$ is the mass of ν. In the definition of $\mathrm{scale}(\nu, f)$, the construction $\mathfrak{M}(\pi_2) \circ \mathrm{dst}_{1,X} \circ \langle f, \eta_X \rangle$ corresponds to a function mapping an element $x \in X$ to a Dirac distribution centered at x and scaled by $f(x)$, whose domain is then lifted

to measures using the Kleisli lifting. To achieve this, we use the equivalence $[\![\mathtt{pReal}]\!] = [0,\infty] \cong \mathfrak{M}1$, and pairing and projection constructions to manage the duplication of x. The definition of scale(v) is more straightforward and reflects the semantics we described before.

Using these constructions we can interpret the corresponding syntactic constructions.

$$[\![\Gamma \vdash \mathtt{scale}(t,t') \colon M[\tau]]\!] \stackrel{\text{def}}{=} \mathrm{scale}([\![\Gamma \vdash t \colon M[\tau]]\!], [\![\Gamma \vdash t' \colon \tau \to \mathtt{pReal}]\!])$$
$$[\![\Gamma \vdash \mathtt{normalize}(t) \colon M[\tau]]\!] \stackrel{\text{def}}{=} \mathrm{normalize}([\![\Gamma \vdash t \colon M[\tau]]\!])$$

We can now interpret `query` as follows:

$$[\![\Gamma \vdash \mathtt{query}\ e \implies e' \colon M[\tau]]\!] \stackrel{\text{def}}{=} \mathrm{normalize}(\mathrm{scale}([\![\Gamma \vdash e \colon M[\tau]]\!], [\![\Gamma \vdash e' \colon \tau \to \mathtt{pReal}]\!]))$$

Using the equivalence $[\![\mathtt{pReal}]\!] = [0,\infty] \cong \mathfrak{M}1$ again, we interpret expectation as:

$$[\![\Gamma \vdash \mathbb{E}_{x\sim t}[t'(x)] \colon \mathtt{pReal}]\!] \stackrel{\text{def}}{=} \lambda\gamma \in [\![\Gamma]\!].\ \left([\![\Gamma \vdash t' \colon \tau \to \mathtt{pReal}]\!](\gamma)\right)^{\sharp}([\![\Gamma \vdash t \colon M[\tau]]\!](\gamma)).$$

The primitives of basic probability distributions `Uniform`, `Bern`, `Gauss` are interpreted by rescaling a measure (given as a constant) with density functions (cf. Section 8.2), and the usual operations on real numbers are given by embedding measurable real functions in **QBS**.

To interpret formulas in PL we use the category **Pred**(**QBS**) of predicates on quasi-Borel spaces. This will be useful to see these formulas as assertions in the unary logic UPL and the relational logic RPL. This is actually the main reason why we use quasi-Borel spaces: we want an assertion logic whose predicates support both higher-order computations and continuous probability. The structure of the category **Pred**(**QBS**) is the following:

- An object is a pair (X, P) where $X \in \mathbf{QBS}$ and $P \subseteq X$.
- A morphism $f \colon (X,P) \to (Y,Q)$ is $f \colon X \to Y \in \mathbf{QBS}$ such that $\forall x \in P.\, f(x) \in Q$.

An important property of this category is that *every arbitrary subset* P of a quasi-Borel space X forms an object (X,P) in **Pred**(**QBS**). This allows us to interpret all logical operations, including universal quantifiers, in a set-theoretic way.

Notice that this category can be seen as the total category of the fibration $q \colon \mathbf{Pred}(\mathbf{QBS}) \to \mathbf{QBS}$ given by the following change-of-base of the fibration $p \colon \mathbf{Pred} \to \mathbf{Set}$ along the forgetful functor $|-| \colon \mathbf{QBS} \to \mathbf{Set}$. Here **Pred** is the category of predicates and all predicate-preserving maps, and the fibration p extracts underlying sets of predicates. Then, all fibrewise properties of the fibration p are inherited by the fibration q. For detail, see [Jacobs 1999, Section 1.5–1.8].

We are now ready to interpret formulas in PL. We interpret a typed formula $\Gamma \vdash \phi\ \mathrm{wf}$ as an object $[\![\Gamma \vdash \phi\ \mathrm{wf}]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], (\!|\Gamma \vdash \phi\ \mathrm{wf}|\!))$ in **Pred**(**QBS**) where the predicate part $(\!|\Gamma \vdash \phi\ \mathrm{wf}|\!)$ is interpreted inductively. We give here a selection of the inductive rules defining the interpretation:

$$(\!|\Gamma \vdash \top\ \mathrm{wf}|\!) \stackrel{\text{def}}{=} [\![\Gamma]\!], \quad (\!|\Gamma \vdash \forall x \colon \tau \phi\ \mathrm{wf}|\!) \stackrel{\text{def}}{=} \textstyle\bigcap_{y \in [\![\tau]\!]} \{\, \gamma \in [\![\Gamma]\!] \mid (\gamma, y) \in (\!|\Gamma, x \colon \tau \vdash \phi\ \mathrm{wf}|\!) \,\},$$
$$(\!|\Gamma \vdash \bot\ \mathrm{wf}|\!) \stackrel{\text{def}}{=} \emptyset, \quad (\!|\Gamma \vdash t_1 = t_2\ \mathrm{wf}|\!) \stackrel{\text{def}}{=} \{\, \gamma \in [\![\Gamma]\!] \mid [\![\Gamma \vdash t_1 \colon \tau]\!](\gamma) = [\![\Gamma \vdash t_2 \colon \tau]\!](\gamma) \,\},$$
$$(\!|\Gamma \vdash \phi_1 \wedge \phi_2\ \mathrm{wf}|\!) \stackrel{\text{def}}{=} (\!|\Gamma \vdash \phi_1\ \mathrm{wf}|\!) \cap (\!|\Gamma \vdash \phi_2\ \mathrm{wf}|\!), \qquad (\!|\Gamma \vdash \neg\phi\ \mathrm{wf}|\!) \stackrel{\text{def}}{=} [\![\Gamma]\!] \setminus (\!|\Gamma \vdash \phi\ \mathrm{wf}|\!),$$

This interpretation is well-behaved with respect to substitution. In particular, the substitution $\phi[t/x]$ of x by an enriched expression t can be interpreted by the inverse image $(\!|\Gamma \vdash \phi[t/x]\ \mathrm{wf}|\!) = \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \vdash t \colon \tau]\!] \rangle^{-1}(\!|\Gamma, x \colon \tau \vdash \phi\ \mathrm{wf}|\!)$. Using this property, we can show that the logic PL is sound with respect to the semantics that we defined above.

Theorem 7.4 (PL Soundness). *If a judgment* $\Gamma \mid \Psi \vdash_{\mathrm{PL}} \phi$ *is derivable then we have the inclusion* $(\bigcap_{\psi\in\Psi} (\!|\Gamma \vdash \psi\ \mathrm{wf}|\!)) \subseteq (\!|\Gamma \vdash \phi\ \mathrm{wf}|\!)$ *of predicates, which is equivalent to having a morphism* $\mathrm{id}_{[\![\Gamma]\!]} \colon [\![\Gamma \vdash \bigwedge_{\psi\in\Psi}\psi\ \mathrm{wf}]\!] \to [\![\Gamma \vdash \phi\ \mathrm{wf}]\!]$ *in the category* **Pred**(**QBS**).

Here, the soundness of PL axioms introduced in Section 5 is proved from the basic facts discussed by Ścibior et al. [2017], in particular, the isomorphism $\mathfrak{M}1 \cong [0,\infty]$, the commutativity of the monad $\mathfrak{M}$, the correspondence between $f^\sharp(\mu)$ and usual integration $\int f d\,\mu$ for any $f : \mathbb{R} \to [0,\infty]$ and $\mathfrak{M}(\mathbb{R})$, and that measurable functions between standard Borel spaces are exactly morphisms in **QBS**.

Using Theorem 6.1 and Theorem 7.4, we can prove the soundness of UPL.

Corollary 7.5 (UPL Soundness). *If* $\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e : \tau \mid \phi$ *then* $\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \vdash e : \tau]\!] \rangle : [\![\Gamma \vdash \bigwedge_{\psi\in\Psi}\psi \ \mathrm{wf}]\!] \to [\![\Gamma, \mathbf{r} : \tau \vdash \phi \ \mathrm{wf}]\!]$ *in* **Pred**(**QBS**).

Similarly, using Theorem 6.2 and Theorem 7.4, we can prove the soundness of RPL.

Corollary 7.6 (RPL Soundness). *If* $\Gamma \mid \Psi \vdash_{\mathrm{RPL}} e_1 : \tau_1 \sim e_2 : \tau_2 \mid \phi$ *then* $\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \vdash e_1 : \tau_1]\!], [\![\Gamma \vdash e_2 : \tau_2]\!] \rangle : [\![\Gamma \vdash \bigwedge_{\psi\in\Psi}\psi \ \mathrm{wf}]\!] \to [\![\Gamma, \mathbf{r}_1 : \tau_1, \mathbf{r}_2 : \tau_2 \vdash \phi \ \mathrm{wf}]\!]$ *in* **Pred**(**QBS**).

8 EXAMPLES

In Section 2 we showed two examples of how to use PPV to reason about probabilistic inference and Monte Carlo approximation. In this section, we demonstrate further how PPV can be used to verify a wide range of properties of probabilistic programs. We will start by showing how to reason formally about probabilistic program *slicing* for continuous random variables as a relational property. We will then consider an example of the use of PPV to reason about the *convergence* of probabilistic inference. We will then move to some statistical applications: we will show how to reason about *mean estimation* of distributions, about the *approximation properties* of importance sampling. Finally, we will show how to use PPV for a proper machine learning task by showing how one can reason about the *Lipschitz continuity* of a generalized iteration algorithm useful for reinforcement learning.

8.1 Slicing of Probabilistic Programs

In this example, we show how PPV can be used to reason about relational properties of probabilistic programs with continuous random variables. Specifically, we show that a combination of relational reasoning in RPL, and equational reasoning in PL allow us to reason about slicing of probabilistic programs [Amtoft and Banerjee 2016]. Slicing is a program analysis technique that can be used to speed up probabilistic inference tasks. Previous work has shown how to slice probabilistic programs with discrete random variables in an efficient way. Here, we consider the problem of checking the correctness of a slice, when the program contains continuous random variables. We look at an example adapted from Amtoft and Banerjee [2016]. Consider the following two programs `left` and `right`:

$$\begin{aligned}
\texttt{left} \equiv\ & \texttt{let}\ x = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\ y = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\ z = x \otimes y\ \texttt{in} \\
& \texttt{mlet}\ v = (\texttt{query}\ \ z \Rightarrow \lambda w.\, \texttt{if}\ \pi_2(w) > 0.5\ \texttt{then}\ 1\ \texttt{else}\ 0)\ \texttt{in}\ \ \texttt{return}(\pi_1(v)) \\
\texttt{right} \equiv\ & \texttt{let}\ x = \texttt{Uniform}(0,1)\ \texttt{in}\ x
\end{aligned}$$

Intuitively, even if `query` in `left` is applied to the product measure z, and not just to the measure of y, the conditioning concerns only y and it does not affect the distribution of x. Indeed, `right` is a correct slice of `left`. We can show this in RPL by proving the following judgment.

$$\vdash_{\mathrm{RPL}} \texttt{left} : M[\texttt{real}] \sim \texttt{right} : M[\texttt{real}] \mid \mathbf{r}_1 = \mathbf{r}_2$$

To prove this judgment, we first apply the relational [LET] rule, which allows us to introduce an assumption about x on both sides. Then we apply a sequence of asynchronous [LET-L] rules on

the program on the left, which introduce preconditions about y and z into the context:

$$x = \texttt{Uniform}(0,1), y = \texttt{Uniform}(0,1), z = x \otimes y \vdash_{\mathrm{RPL}}$$
$$\texttt{mlet}\, v = (\texttt{query}\ z \Rightarrow \lambda w.\, \texttt{if}\, \pi_2(w) > 0.5\, \texttt{then}\, 1\, \texttt{else}\, 0) \texttt{in}\ \texttt{return}(\pi_1(v)) \sim x \mid \mathbf{r}_1 = \mathbf{r}_2$$

To prove this judgement we rely on the equalities on monadic bind, rescaling, and conditioning in Section 5. Starting from the HPProg term on the left, by applying the equations (13), (14), and (9), we reduce it to $\texttt{mlet}\, v = (x \otimes X)\ \texttt{in}\ \texttt{return}(\pi_1(v))$ where X is a *normalized* distribution defined by the term $\texttt{query}\, y \Rightarrow \lambda w_2.\, \texttt{if}\, w_2 > 0.5\, \texttt{then}\, 1\, \texttt{else}\, 0$. We then conclude this is equal to x by applying the equality (9) and the equality

$$\texttt{mlet}\, w = e_1 \otimes e_2\ \texttt{in}\ \texttt{return}\, \pi_1(w) = \texttt{scale}(e_1, \mathbb{E}_{x \sim e_2}[1])$$

proved from the equalities (12), (7) and monadic laws.

Using RPL we can also reason about situations where we cannot slice a program. Adapting again from Amtoft and Banerjee [2016], let us consider the following two programs `left` and `right`:

$$\begin{aligned}
\texttt{left} \equiv\ & \texttt{let}\, x = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\, y = \texttt{Uniform}(0,1)\ \texttt{in}\, \texttt{let}\, z = x \otimes y\ \texttt{in} \\
& \texttt{mlet}\, v = (\texttt{query}\, z \Rightarrow \lambda w.\, \texttt{if}\, \pi_1(w) + \pi_2(w) > 0.5\, \texttt{then}\, 1\, \texttt{else}\, 0)\ \texttt{in}\, \texttt{return}(\pi_1(v)) \\
\texttt{right} \equiv\ & \texttt{let}\, x = \texttt{Uniform}(0,1)\ \texttt{in}\ x
\end{aligned}$$

Now we prove that it is not correct to slice `left` into `right` by means of the judgment below:

$$\vdash_{\mathrm{RPL}} \texttt{left}\colon M[\texttt{real}] \sim \texttt{right}\colon M[\texttt{real}] \mid \mathbf{r}_1 \neq \mathbf{r}_2$$

The proof for this judgment follows the structure of the proof of the previous example. The main difference is that now we need to see the first coordinate of the variable w in the conditioning. To prove that `left` and `right` are different, we use the probabilistic inference in the first example to prove $\vdash_{\mathrm{UPL}} \texttt{left} \mid \Pr_{y \sim \mathbf{r}}[y > .5] > 1/2$ using the the [Bayes] rule and the following calculation: $\frac{\Pr_w[\pi_1(w) > .5]}{\Pr_w[\pi_1(w) + \pi_2(w) > .5]} \geq \frac{\Pr_x[x > .5]}{1 - \Pr_x[x > .25] * \Pr_y[y > .25]} = \frac{8}{15} > \frac{1}{2}$.

Similarly, we can look at the following two programs:

$$\begin{aligned}
\texttt{left} \equiv\ & \texttt{mlet}\, x = \texttt{Uniform}(0,1)\ \texttt{in} \\
& \texttt{mlet}\, _ = \big(\texttt{if}\, x > .5\ \texttt{then}\ \big(\texttt{let}\, y = \texttt{Uniform}(0,1)\ \texttt{in}\, \texttt{let}\, z = \texttt{return}(x) \otimes y\ \texttt{in} \\
& \texttt{query}\ z \Rightarrow \lambda w.\, \texttt{if}\, \pi_2(w) > .5\, \texttt{then}\, 1\, \texttt{else}\, 0\big)\, \texttt{else}\, \texttt{return}(x \otimes x)\big) \texttt{in}\ \texttt{return}(x) \\
\texttt{right} \equiv\ & \texttt{mlet}\, x = \texttt{Uniform}(0,1)\ \texttt{in}\, \texttt{return}(x)
\end{aligned}$$

and show that we can slice `left` into `right`.

A key point in deriving the slicing property of the above examples is the equation $\texttt{mlet}\, w = e_1 \otimes e_2\ \texttt{in}\ \texttt{return}\, \pi_1(w) = \texttt{scale}(e_1, \mathbb{E}_{x \sim e_2}[1])$ of splitting product measure, which is obtained by applying the axioms in Section 5. When $e_2 \equiv \texttt{query}\ e_3 \Rightarrow e_4$, we have $\texttt{mlet}\, w = e_1 \otimes e_2\ \texttt{in}\ \texttt{return}\, \pi_1(w) = e_1$ since our conditioning construction is *normalized*, and hence $\mathbb{E}_{x \sim e_2}[1] = 1$. On the other hand, when e_2 consists of *unnormalized* conditioning, we may have the non-slicing $\texttt{mlet}\, w = e_1 \otimes e_2\ \texttt{in}\ \texttt{return}\, \pi_1(w) \neq e_1$ because $\mathbb{E}_{x \sim e_2}[1] < 1$. This is an advantage of our conditioning operator. Since we renormalize in conditioning construction, we can slice the algorithm `left` into `right` in the third example.

Putting the first and the third example together we can consider the following two programs `left` and `right`:

$$\begin{aligned}
\texttt{left} \equiv\ & \texttt{mlet}\ x = \texttt{Uniform}(0,1)\ \texttt{in}\\
& \texttt{mlet}\ _ = \big(\texttt{if}\ x > .5\ \texttt{then}\ \big(\texttt{let}\ y = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\ z = \texttt{return}(x) \otimes y\ \texttt{in}\\
& \qquad \texttt{query}\ z \Rightarrow \lambda w.\, \texttt{if}\ \pi_2(w) > .5\ \texttt{then}\ 1\ \texttt{else}\ 0\big)\ \texttt{else}\ \texttt{return}(x \otimes x)\big)\texttt{in}\\
& \texttt{let}\ u = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\ k = \texttt{return}(x) \otimes u\ \texttt{in}\\
& \texttt{mlet}\ v = (\texttt{query}\ k \Rightarrow \lambda w.\, \texttt{if}\ \pi_2(w) > .5\ \texttt{then}\ 1\ \texttt{else}\ 0)\ \texttt{in}\ \texttt{return}(\pi_1(v))
\end{aligned}$$

$$\begin{aligned}
\texttt{right} \equiv\ & \texttt{mlet}\ x = \texttt{Uniform}(0,1)\ \texttt{in}\\
& \texttt{mlet}\ _ = \big(\texttt{if}\ x > .5\ \texttt{then}\ \big(\texttt{let}\ y = \texttt{Uniform}(0,1)\ \texttt{in}\ \texttt{let}\ z = \texttt{return}(x) \otimes y\ \texttt{in}\\
& \qquad \texttt{query}\ z \Rightarrow \lambda w.\, \texttt{if}\ \pi_2(w) > .5\ \texttt{then}\ 1\ \texttt{else}\ 0\big)\ \ \texttt{else}\ \texttt{return}(x \otimes x)\big)\\
& \texttt{in}\ \texttt{return}(x)
\end{aligned}$$

Again, we want to show that `right` is a correct slice of `left` by proving that $\vdash_{\mathrm{RPL}} \texttt{left}\colon M[\texttt{real}] \sim \texttt{right}\colon M[\texttt{real}] \mid \mathbf{r}_1 = \mathbf{r}_2$. The proof of this judgment can be carried out mostly in RPL, by using the similarity between the two programs `left` and `right`. The proof starts by using relational reasoning, and afterwards reuses the proof of the first example. This shows that reasoning relationally about slicing can be better than reasoning directly about equivalence by computing the two distributions.

8.2 Gaussian Mean Learning: Convergence and Stability

Probabilistic programs are often used as models for probabilistic inference tasks in data analysis. We now show how PPV can be used to reason about such processes. Taking the example of the closed-form Bayesian update, we show how to use PPV to reason about two quite common properties, *convergence* and *stability* under changes of priors. These two properties allow us to illustrate two different aspects of PPV: 1) The support it offers for reasoning about iterative probabilistic tasks and for reasoning about densities of random variables, and 2) The support it offers for relational reasoning about measures of divergence of one distribution with respect to another. To show this, we first prove the convergence of the iterative closed-form learning of the mean of a Gaussian distribution (with fixed variance). We then prove this process stable for a precise notion of stability formulated in terms of Kullback-Leibler (KL) divergence.

Let us start by considering the following implementation `GaussLearn` of an algorithm for Bayesian learning of the mean of a Gaussian distribution with known variance σ^2 from a sample list L:

$$\texttt{GaussLearn} \equiv \lambda p.\, \texttt{letrec}\, f(L) = \texttt{case}\, L\ \texttt{with}\ [] \Rightarrow p, y :: ls \Rightarrow \texttt{query}\ f(ls)\ \Rightarrow\ \texttt{GPDF}(y, \sigma^2)$$

where $\texttt{GPDF}(y,\sigma^2)$ is a shorthand for the density function $\lambda r.\, \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{(r-y)^2}{2\sigma^2})$ of a Gaussian distribution $\texttt{Gauss}(y,\sigma^2)$ with mean y and variance σ^2. This algorithm starts by assuming a prior p on the unknown mean. Then, on each iteration, a sample y is read from the list and the prior gets updated by observing it as a Gaussian with mean y and variance σ^2.

We now want to show two properties of this algorithm. The first property we show is convergence: the mean of the posterior should roughly converge to the mean of the data, but we need to take into account that the posterior also depends on the prior. More precisely, when the prior is also a Gaussian, we can show that:

$$\begin{aligned}
(\sigma > 0), (\xi > 0) \vdash_{\mathrm{RPL}} & \texttt{GaussLearn} \sim \texttt{Total} \mid \forall L'\colon \texttt{list}(\texttt{real}).\ \forall n\colon \texttt{nat}.\ (n = |L'|)\\
& \Longrightarrow\ \mathbf{r}_1(\texttt{Gauss}(\delta, \xi^2))(L') = \texttt{Gauss}(\tfrac{\mathbf{r}_2(L') * \xi^2 + \delta * \sigma^2}{n * \xi^2 + \sigma^2},\ \tfrac{\xi^2 * \sigma^2}{n * \xi^2 + \sigma^2})
\end{aligned} \tag{18}$$

where `Total` is an algorithm summing all the elements of a list L.

$$\texttt{Total} \equiv \texttt{letrec}\, f(L\colon \texttt{list}(\texttt{real})) = \texttt{case}\, L\, \texttt{with}\, [] \Rightarrow 0, y :: ls \Rightarrow y + f(ls).$$

This judgement states that, if the prior on the mean is a Gaussian of mean δ and variance ξ^2, then the posterior is a Gaussian with mean close to the mean of $\texttt{Total}(L)$ and variance close to 0, but that they are still influenced by the parameters δ, ξ^2 of the prior.

The proof of this judgment proceeds relationally by first applying the one-sided [ABS-L] rule to introduce the prior in the context. Then the proof continues synchronously by applying the [r-LETREC] and [r-LISTCASE] rules. To conclude the proof we need to show the following two premises corresponding to the base case and to the inductive step:

$$\begin{array}{l}(\sigma > 0), (\xi > 0), \phi_{\text{ind.hyp}}, (L = []), d_{\text{prior}} = \texttt{Gauss}(\delta, \xi^2), (n = |L|) \\ \quad \vdash_{\text{RPL}} d_{\text{prior}} \sim 0 \mid \mathbf{r}_1 = \texttt{Gauss}(\frac{\mathbf{r}_2 * \xi^2 + \delta * \sigma^2}{n * \xi^2 + \sigma^2}, \frac{\xi^2 * \sigma^2}{n * \xi^2 + \sigma^2}) \\ (\sigma > 0), (\xi > 0), \phi_{\text{ind.hyp}}, (L = y :: ls), d_{\text{prior}} = \texttt{Gauss}(\delta, \xi^2), (n = |L|) \\ \quad \vdash_{\text{RPL}} \texttt{query}\, f_1(ls) \Rightarrow \texttt{Gauss}(y, \sigma^2) \sim y + f_2(ls) \mid \mathbf{r}_1 = \texttt{Gauss}(\frac{\mathbf{r}_2 * \xi^2 + \delta * \sigma^2}{n * \xi^2 + \sigma^2}, \frac{\xi^2 * \sigma^2}{n * \xi^2 + \sigma^2})\end{array}$$

The first premise is obvious. The second premise requires a little more work, and can be proved by applying [r-QRY-L] and [r-SUB] rules and several equations in PL. We first show in PL that Gaussian distributions are conjugate prior with respect to the Gaussian likelihood function by applying the equations on rescaling, normalization, and conditioning.

$$\vdash_{\text{PL}} (\sigma > 0) \wedge (\xi > 0) \implies \texttt{query}\, \texttt{Gauss}(\delta, \xi^2) \Rightarrow \texttt{GPDF}(z, \sigma^2) = \texttt{Gauss}(\frac{z\xi^2 + \delta\sigma^2}{\xi^2 + \sigma^2}, \frac{\xi^2\sigma^2}{\xi^2 + \sigma^2}).$$

Then, we apply [r-QRY-L] and [r-SUB] to the premise (8.2) to introduce the observations in the precondition, and apply the above fact and the induction hypothesis.

The second property we show is stability. If we run `GaussLearn` twice with different prior Gaussian distributions, we can show that the posteriors will be close if the list of samples is long enough and not diverging. This closeness is defined in terms of the Kullback-Leibler (KL) divergence. The KL divergence of two distributions with known density functions, can be defined by expectations: $(d_1 = \texttt{scale}(d_2, f)) \implies (\text{KL}(d_1 \,||\, d_2) = \mathbb{E}_{x \sim d_1}[\log f(x)])$. In particular, the KL divergence of two Gaussian distributions can be calculated as follows:

$$\text{KL}(\texttt{Gauss}(\mu_1, \sigma_1^2) \,||\, \texttt{Gauss}(\mu_2, \sigma_2^2)) = (\log|\sigma_2| - \log|\sigma_1|) + (\sigma_1^2 + (\mu_1 - \mu_2)^2)/\sigma_2^2 - 1/2 \tag{19}$$

Formally, we want to prove the following judgment.

$$\begin{array}{l}\sigma : \texttt{real}, \delta : \texttt{real}, \xi : \texttt{real}, \delta_2 : \texttt{real}, \xi_2 : \texttt{real} \mid (\sigma > 0), (\xi > 0), (\xi_2 > 0) \\ \quad \vdash_{\text{RPL}} \texttt{GaussLearn} \sim \texttt{GaussLearn} \mid \forall L' : \texttt{list(real)}.\forall \varepsilon : \texttt{real}.\forall C : \texttt{real}. \\ \quad\quad (\varepsilon > 0) \implies \exists N : \texttt{nat}.(|L'| > N) \wedge |\texttt{Total}(L')| < C * |L'| \\ \quad\quad\quad \implies \text{KL}(\mathbf{r}_1(\texttt{Gauss}(\delta, \xi^2))(L') \,||\, \mathbf{r}_1(\texttt{Gauss}(\delta_2, \xi_2^2))(L')) < \varepsilon\end{array} \tag{20}$$

Intuitively, this states that if the algorithm is run twice with different Gaussian priors, and the mean of the data is bounded by some C, then the KL divergence of the posteriors can be made as small as desired by increasing the size of the data. In other words, the effect of the prior on the posterior can be minimized by having enough samples.

By simple calculations, we can prove in PL the following assertion in a similar way as proofs of convergence of sequences using the epsilon-delta definition of limit.

$$\begin{array}{l}\vdash_{\text{PL}} \forall L' : \texttt{list(real)}.\forall \varepsilon : \texttt{real}.\forall C : \texttt{real}. \\ \quad (\varepsilon > 0) \implies \exists N : \texttt{nat}.(|L'| > N) \wedge |\texttt{Total}(L')| < C * |L'| \implies \\ \quad \left|\frac{\texttt{Total}(L') * \xi^2 + \delta_2 * \sigma^2}{|L'| * \xi^2 + \sigma^2} - \frac{\texttt{Total}(L') * \xi_2^2 + \delta_2 * \sigma^2}{|L'| * \xi_2^2 + \sigma^2}\right| < \varepsilon \wedge \left|\log \frac{n * \xi^2 * \xi_2^2 + \xi^2 * \sigma^2}{n * \xi^2 * \xi_2^2 + \xi_2^2 * \sigma^2}\right| < \varepsilon\end{array} \tag{21}$$

To prove (20), we want to combine (18) with (19) and (21). To do this, we apply the relational [r-SUB] rule to the judgment (20), which has the following PL premise:

$$\begin{aligned}&\vdash_{\mathrm{PL}} \top \implies \forall L' \colon \mathtt{list}(\mathtt{real}).\forall \varepsilon \colon \mathtt{real}.\forall C \colon \mathtt{real}.\\ &\quad (\varepsilon > 0) \implies \exists N \colon \mathtt{nat}.(|L'| > N) \wedge |\mathtt{Total}(L')| < C * |L'|\\ &\qquad \implies \mathrm{KL}(\mathtt{GaussLearn}(\mathtt{Gauss}(\delta, \xi^2))(L') \,||\, \mathtt{GaussLearn}(\mathtt{Gauss}(\delta_2, \xi_2^2))(L')) < \varepsilon.\end{aligned}$$

We prove this in PLby first applying the rule [conv-RPL] to (18) and then using (19) and (21).

8.3 Sample Size Required in Importance Sampling

As another example of a common statistical task, we use PPV to show the correctness of self-normalizing importance sampling. Importance sampling is an efficient variant of Monte Carlo approximation to estimate the expected value $\mathbb{E}_{x\sim d'}[h(x)]$ when sampling from d' is not convenient. The idea is to sample from a different distribution d and then rescale the samples by using the density function g of d'. The most interesting aspect of this example is that correctness is formulated as a probability bound on the difference between the mean of the distribution d' and the empirical mean. This shows once again that PPV supports reasoning about such probabilistic bounds, which are quite widespread in statistical applications. However, here we want to go a step further and show that we can reason about probability bounds that are parametric in the *number of available data samples*. This quantity is often crucial for both theoretical understanding and practical reasons, since data is an expensive resource. For our specific example, we rely on recent work by Chatterjee and Diaconis [2018] and use their theorem as the correctness statement. This example also shows the usefulness of the equations of Section 5 in high-level reasoning.

The following algorithm `SelfNormIS` is an implementation of self-normalizing importance sampling.

$$\begin{aligned}\mathtt{SelfNormIS} &\equiv \lambda n \colon \mathtt{nat}.(\mathtt{mlet}\, z = \mathtt{SumLoop}(n)(g)(h)\ \mathtt{in}\ \mathtt{return}(\pi_1(z)/\pi_2(z)))\\ \mathtt{SumLoop} &\equiv \mathtt{letrec}\, f(i \colon \mathtt{nat}) = \lambda g \colon \tau \to \mathtt{real}.\lambda h \colon \tau \to \mathtt{real}.\\ &\qquad \mathtt{if}(i \le 0)\ \mathtt{then}\ \mathtt{return}\langle 0, 0\rangle\ \mathtt{else}\ \mathtt{mlet}\, x = d\ \mathtt{in}\ \mathtt{mlet}\, m = f(i-1)(g)(h)\mathtt{in}\\ &\qquad \mathtt{return}\langle (1/i)(\pi_1(m) + (i-1) * h(x) * g(x)), (1/i)(\pi_2(m) + (i-1) * g(x))\rangle.\end{aligned}$$

This algorithm approximates $\mathbb{E}_{x\sim d'}[h(x)] = \int h(x)g(x)\,dx$ by taking samples $X_1 \ldots X_n \sim d$ and computing the ratio $(\frac{1}{n}\sum_{i=1}^n g(X_i)h(X_i))/(\frac{1}{n}\sum_{i=1}^n g(X_i))$ of weighed sum instead. Note that `SumLoop` is the subroutine calculating the numerator $\frac{1}{n}\sum_{i=1}^n g(X_i)h(X_i)$ and denominator $\frac{1}{n}\sum_{i=1}^n g(X_i)$ of empirical expected value from the same samples $X_i \sim d$.

We verify a recent result on the sample size required in self-normalizing importance sampling. The goal is to prove the following PPV representation of Theorem 1.2 of Chatterjee and Diaconis [2018]:

$$\begin{aligned}&d \colon M[\tau], g \colon \tau \to \mathtt{real}, h \colon \tau \to \mathtt{real} \vdash_{\mathrm{UPL}} \mathtt{SelfNormIS} \colon \mathtt{nat} \to M[\mathtt{real}]\ |\\ &\quad \forall d' \colon M[\tau].\forall \mu \colon \mathtt{real}.\forall \sigma \colon \mathtt{real}.\forall C \colon \mathtt{real}.\forall t \colon \mathtt{real}.\forall L \colon \mathtt{real}.\forall \varepsilon \colon \mathtt{real}.\\ &\qquad \phi \wedge (\varepsilon > \mathtt{sqrt}(\exp(-t/4) + 2\mathtt{sqrt}(\Pr_{y\sim d'}[\log(g(y)) > L + t/2])))\\ &\qquad\quad \implies \forall k \colon \mathtt{nat}.k > \exp(L+t) \implies \Pr_{y\sim \mathtt{r}(k)(g)(h)}[|y-\mu| \ge \tfrac{2\varepsilon \mathtt{sqrt}(\sigma^2+\mu^2)}{1-\varepsilon}] \le 2\varepsilon\end{aligned} \tag{22}$$

Here, C is supposed to be an unknown normalization factor of g. The following assertion ϕ is the assumption that gives the required sample size.

$$\begin{aligned}\phi \equiv\ &(\mathbb{E}_{x\sim d}[1] = 1) \wedge (\sigma^2 = \mathrm{Var}_{x\sim d}[h(x) * g(x)]) \wedge (\mu = \mathbb{E}_{y\sim d'}[h(y)]) \wedge (t \ge 0)\\ &\wedge (d' = \mathtt{scale}(d, g/C)) \wedge (C > 0) \wedge (\mathbb{E}_{y\sim d'}[1] = 1) \wedge (L = \mathbb{E}_{x\sim d'}[\log g(y)])\end{aligned}$$

The previous theorem gives a bound on the probability that the estimate differs too much from the actual expected value μ. The proof of the judgment (22) is involved, and requires several steps. First,

we prove a version of the theorem for naive (non self-normalizing) importance sampling [Chatterjee and Diaconis 2018, Theorem 1.1]. Then, we extend this result to self-normalizing importance sampling. Naive importance sampling is defined as:

$$\begin{aligned}\texttt{Naive} \equiv\ & \texttt{letrec}\, f(i\colon \texttt{nat}) = \lambda g\colon \tau \to \texttt{real}.\lambda h\colon \tau \to \texttt{real}.\\ & \texttt{if}(i \le 0)\ \texttt{then}\ (\texttt{return}\ 0)\ \texttt{else}\ \texttt{mlet}\ x = d\ \texttt{in}\ \texttt{mlet}\ m = f(i-1)(g)(h)\texttt{in}\\ & \texttt{return}\ \tfrac{1}{i}(m + (i-1) * h(x) * g(x))\end{aligned}$$

Here Naive computes $\frac{1}{n}\sum_{i=1}^{n} g(X_i)h(X_i)$. We want to show:

$$\begin{aligned}&\vdash_{\text{UPL}} \texttt{Naive}\colon \texttt{nat} \to (\tau \to \texttt{real}) \to (\tau \to \texttt{real}) \to M[\texttt{real}]\ |\\ &\forall d'\colon M[\tau].\forall \mu\colon \texttt{real}.\forall \sigma\colon \texttt{real}.\forall C\colon \texttt{real}.\forall t\colon \texttt{real}.\forall L\colon \texttt{real}.\forall \varepsilon\colon \texttt{real}.\\ &\phi \implies \forall k\colon \texttt{nat}.k > \exp(L+t)\\ &\quad\implies \mathbb{E}_{w\sim \mathbf{r}(k)(g/C)(h)}[|w-\mu|]\\ &\qquad \le \texttt{sqrt}(\sigma^2+\mu^2) * (\exp(-t/4) + 2\texttt{sqrt}(\Pr_{y\sim d'}[\log(g(y)) > L + t/2]))\end{aligned} \tag{23}$$

Notice that we need the normalization factor C.

The main "tricks" in the proof are the Cauchy-Schwartz inequality and introducing the function $h_2 = \lambda x\colon \tau.(\texttt{if}\ g(x) \le k * \exp(-t/2)\ \texttt{then}\ 1\ \texttt{else}\ 0) * h(x)$. We first check the following in PL, where μ' is defined as $\mathbb{E}_{y\sim d'}[h_2(y)]$.

$$\begin{aligned}&\mathbb{E}_{w\sim \texttt{Naive}(k)(g/C)(h)}[|w-\mu|]\\ &\quad= \mathbb{E}_{y\sim \texttt{SumLoop2}(k)(g/C)(h)(\lambda x\colon \tau.(\texttt{if}\ g(x)\le k*\exp(-t/2)\texttt{then1else0})*h(x))}[|\pi_1(y)-\mu|]\\ &\quad\le \mathbb{E}[|\pi_2(y)-\mu'|] + \mathbb{E}[|\pi_1(y)-\pi_2(y)|] + \mathbb{E}[|\mu-\mu'|]\\ &\quad\le \texttt{sqrt}(\sigma^2+\mu^2) * (\exp(-t/4) + 2\texttt{sqrt}(\Pr_{y\sim d'}[\log(g(y)) > L + t/2])).\end{aligned}$$

In the first step, we use the equivalence of Naive and an alternate version of SumLoop that we denote SumLoop2. Here, we introduce the helper function h_2 in the expression. The second step is applying axioms on expectations (the triangle-inequality). In the last step, we use Cauchy-Schwartz inequality and the inequality $h_2 \le h$, which follows from the definition of h_2.

Finally, we prove our goal (22) from the just-established judgment (23). Define $b \equiv \exp(-t/4) + 2\texttt{sqrt}(\Pr_{y\sim d'}[\log(g(y)) > L + t/2])$ and $\delta \equiv \texttt{sqrt}(b) * \texttt{sqrt}(\sigma^2+\mu^2)$, and assume $\varepsilon > \texttt{sqrt}(b)$. The main part of the proof is the following inequality in PL.

$$\begin{aligned}&\Pr_{z\sim \texttt{SumLoop}(k)(g)(h)}[|\tfrac{\pi_1(z)}{\pi_2(z)} - \mu| \ge \tfrac{2\varepsilon\texttt{sqrt}(\sigma^2+\mu^2)}{1-\varepsilon}]\\ &\quad\le \Pr_{w\sim \texttt{Naive}(k)(g/C)(h)}[|w-\mu| \le \delta] + \Pr_{w\sim \texttt{Naive}(k)(g/C)(1)}[|w-1| \le \texttt{sqrt}(b)]\\ &\quad\le \mathbb{E}_{w\sim \texttt{Naive}(k)(g/C)(h)}[|w-\mu|]/\delta + \mathbb{E}_{w\sim \texttt{Naive}(k)(g/C)(1)}[|w-1|]/\texttt{sqrt}(b)\\ &\quad\le \tfrac{b*\texttt{sqrt}(\sigma^2+\mu^2)}{\delta} + \tfrac{b}{\texttt{sqrt}(b)} \le 2\varepsilon\end{aligned}$$

The first step is proved by switching from Naive to SumLoop which requires some structural reasoning and calculations on real numbers supported by PL. The second step follows from the Markov inequality, and the last step follows from the definitions of b and δ.

8.4 Verifying Lipschitz GVI Algorithm

As our final example, we show that PPV can be used to reason about a reinforcement learning task through relational reasoning about Lipschitz continuity and about statistical distances. This is another example of the usefulness of relational reasoning (in a different domain). The example also shows how the expressiveness of PL allows us to reason about notions like Lipschitz continuity and statistical distances.

GVI (Generalized Value Iteration) is a reinforcement learning algorithm to optimize a value function on a Markov Decision Process $(\tau_S, \tau_A, R, T, \gamma)$ where τ_S is a space of states, τ_A is a set of actions, $R\colon \tau_S \times \tau_A \to \texttt{real}$ is a reward function, $T\colon \tau_S \times \tau_A \to D[\tau_S]$ is a transition dynamic and γ

is a discount factor. Our assumption is that the optimal value function satisfies a specific condition, called a Bellman equation: $Q(s,a) = R(s,a) + \mathbb{E}_{s' \sim T(s,a)}[f(Q)(s'))]$, where $f : (\tau_S \times \tau_A \to \texttt{real}) \to (\tau_S \to \texttt{real})$ is a backup operator (usually we take $\max_{a:\ \tau_A}$).

Asadi et al. [2018] show that, under some constraints, the GVI algorithm returns Lipschitz-continuous value functions, which are convenient for modeling learning algorithms over the MDP. The following program `LipGVI` is an implementation of GVI algorithm:

$$\texttt{LipGVI} \equiv \texttt{letrec}\, h(k\colon \texttt{nat}) = \lambda Q'\colon \tau_S \times \tau_A \to \texttt{real}.(\lambda(s,a)\colon \tau_S \times \tau_A.R(s,a) + \gamma g(Q')(s))h(k-1)$$

The algorithm receives an estimate Q' of the value function and updates it using a function g which is assumed to be an approximation of $\lambda Q'.\ \lambda s.\ \mathbb{E}_{s' \sim T(s,a)}[f(Q')(s'))]$. We want to verify the Lipschitz continuity of the result of the algorithm `LipGVI`. Before stating this, we need to add to PL necessary operators and metrics. A function $f : X \to \texttt{real}$ is Lipschitz continuous if there exists a finite $K(f)$ such that $K(f) = \sup_{x_1,x_2 \in X}(|f(x_1) - f(x_2)|/\mathrm{dist}_X(x_1,x_2))$. To define this concept in PL we start by defining the sup operator:

$$(a = \sup_{x:\ \tau \text{ s.t. } \phi(x)} e(x)) \equiv \forall x\colon \tau.\phi(x) \implies (e(x) \le a) \wedge \forall b\colon \tau.(\forall x\colon \tau.\ \phi(x) \implies e(x) \le b) \implies a \le b$$

Next, we implement the notions of the Lipschitz constant and the Wasserstein metric (sometimes known as the Kantorovic metric):

$$\begin{aligned}(a = K_{d_1,d_2}(f)) &\equiv a = \sup_{\langle s_1,s_2\rangle\colon \tau_1 \times \tau_1} d_2(f(s_1), f(s_2))/d_1(s_1,s_2)\\ (a = W_{d_1}(\mu_1,\mu_2)) &\equiv a = \sup_{f:\ \tau_1 \to \texttt{real} \text{ s.t. } K_{d_1,d_{\mathbb{R}}}(f) \le 1} (\mathbb{E}_{s\sim\mu_1}[f(s)] - \mathbb{E}_{s\sim\mu_2}[f(s)])\end{aligned}$$

where $d_{\mathbb{R}}\colon \texttt{real} \times \texttt{real} \to \texttt{pReal}$ is the usual metric in the real line. The standard lemmas on summation and composition for Lipschitz constants (see, e.g., [Asadi et al. 2018, Lemmas 1 and 2]) can be proved in PL by unfolding.

Now we are in a position to state the main theorem by Asadi et al. [2018] in PPV:

$$\begin{aligned}&\forall Q\colon \tau_S \times \tau_A \to \texttt{real}.K_{d_S,d_{\mathbb{R}}}(f(Q)) \le \sup_{a:\ \tau_A} K_{d_S,d_{\mathbb{R}}}(\lambda s\colon \tau_S.Q(s,a))\\ &g = \lambda Q'.\lambda s.\ \mathbb{E}_{s'\sim T(s,a)}[f(Q')(s'))], \forall s\colon \tau_S.\forall a\colon \tau_A.\mathbb{E}_{s'\sim T(s,a)}[1] = 1, \gamma K_{d_S,W}(T) < 1\\ &\vdash_{\mathrm{UPL}} \texttt{LipGVI}\colon \texttt{nat} \to (\tau_S \times \tau_A \to \texttt{real}) \to (\tau_S \times \tau_A \to \texttt{real}) \mid\\ &\forall Q\colon \tau_S \times \tau_A \to \texttt{real}.\forall \varepsilon\colon \texttt{real}.\forall K_1\colon \texttt{real}.(\varepsilon > 0) \wedge (K_1 = \sup_{a:\ \tau_A} K_{d_S,d_{\mathbb{R}}}(Q)(a))\\ &\quad \implies \exists k\colon \texttt{nat}.\forall K_2\colon \texttt{real}.\forall K_3\colon \texttt{real}.\forall K_4\colon \texttt{real}.(K_2 = \sup_{a:\ \tau_A} K_{d_S,d_{\mathbb{R}}}(\mathbf{r}(k))(a))\\ &\quad\ \wedge (K_3 = K_{d_S,d_{\mathbb{R}}}(R)) \wedge (K_4 = K_{d_S,W}(T)) \implies K_2 \le K_3/(1 - \gamma * K_4) + \varepsilon\end{aligned} \tag{24}$$

Here, d_S is a distance function on the state space. The logical assumptions are the losslessness of the transition dynamics T, and the definition $g = \lambda Q'.\ \lambda s.\ \mathbb{E}_{s'\sim T(s,a)}[f(Q')(s'))]$. We also introduce four slack variables K_1, K_2, K_3, and K_4 to use the above syntactic sugar for Lipschitz constants. The judgment (24) itself is proved inductively as in the paper [Asadi et al. 2018]. The key part of the proof is showing the inequality:

$$K_{d_S,d_{\mathbb{R}}}(\lambda s\colon \tau_S.\ \mathbb{E}_{s'\sim T(s,a)}[f(Q')(s'))]) \le K_{d_S,d_{\mathbb{R}}}(\lambda s'\colon \tau_S.\ f(Q')(s'))) \cdot K_{d_S,W_{d_S}}(\lambda s\colon \tau_S.T(s,a))$$

Suppose $K_1 = K_{d_S,d_{\mathbb{R}}}(\lambda s'\colon \tau_S.\ f(Q')(s')))$ and $K_2 = K_{d_S,W_{d_S}}(\lambda s\colon \tau_S.T(s,a))$. What we prove in our framework is that $z = K_{d_S,d_{\mathbb{R}}}(\lambda s\colon \tau_S.\ \mathbb{E}_{s'\sim T(s,a)}[f(Q')(s'))])$ implies $z \le K_1 * K_2$. By unfolding and applying linearity of expectation, we obtain:

$$\begin{aligned}&z = K_{d_S,d_{\mathbb{R}}}(\lambda s\colon \tau_S.\ \mathbb{E}_{s'\sim T(s,a)}[f(Q')(s'))])\\ &\quad\iff z = \sup_{s_1,s_2\colon \tau_S} \frac{K_1 \cdot (\mathbb{E}_{s'\sim T(s_1,a)}[f(Q')(s'))/K_1] - \mathbb{E}_{s'\sim T(s_2,a)}[f(Q')(s'))/K_1])}{d_S(s_1,s_2)}.\end{aligned}$$

Here $1 = K_{d_S, d_{\mathbb{R}}}(\lambda s' \colon \tau_S.\ f(Q')(s'))/K_1)$ holds from the property $d_{\mathbb{R}}(\alpha \cdot x, \alpha \cdot y) = \alpha \cdot d_{\mathbb{R}}(x, y)$ of $d_{\mathbb{R}}$ $(0 \le \alpha)$ and the losslessness $\forall s \colon \tau_S . \forall a \colon \tau_A .\ \mathbb{E}_{s' \sim T(s,a)}[1] = 1$ of the function T. Hence, we conclude $z \le K_1 * K_2$.

9 DOMAIN-SPECIFIC REASONING PRINCIPLES

Paper proofs of randomized algorithms typically use proof techniques to abstract away unimportant details. In this section, we show how PPV can support custom proof techniques in the form of domain-specific logics for reasoning about higher-order programs. Specifically, we show that the $\top\top$-lifting construction by Katsumata [2014]—roughly, a categorical construction useful for building different refinements of the probability distribution monad—can be smoothly incorporated in PPV. As concrete examples, we instantiate the unary $\top\top$-lifting construction to a logic for reasoning about the probability of failure using the so-called union bound [Barthe et al. 2016b], and the binary $\top\top$-lifting construction to a logic for reasoning about probabilistic coupling.

9.1 Embedding Unary Graded $\top\top$-lifting

Roughly speaking, the $\top\top$-lifting of a monad is given by a large intersection of inverse images of some predicate, called *lifting parameters*. We can internalize this construction of $\top\top$-lifting in PPV using a large intersection of assertions as $\forall x \colon \tau . \phi_x$, and the inverse image $\phi[e/y]$ of an assertion ϕ along an expression e. First, we internalize a general construction of *graded* $\top\top$-lifting (along the fibration $q \colon \mathbf{Pred}(\mathbf{QBS}) \to \mathbf{QBS}$) in the unary logic UPL. Then we instantiate it to construct a unary graded $\top\top$-lifting for reasoning about the probability of failure using a union bound.

These instantiations of $\top\top$-liftings need subprobability measures. Accordingly, we introduce a new monadic type $D[\tau]$ describing the set of *subprobability* measures over τ. We interpret D by $[\![D[\tau]]\!] \stackrel{\text{def}}{=} \mathfrak{P}[\![\tau]\!]$, and interpret monadic structures in the same way as the ones on the monadic type M. Furthermore, we assume that for every type τ, $D[\tau]$ is a subtype of $M[\tau]$. We introduce the following axioms enabling syntactic conversions from distributions in $D[\mathtt{unit}]$ to real numbers in $[0, 1]$.

$$\frac{\Gamma \vdash e \colon D[\mathtt{unit}]}{\Gamma \vdash_{\mathrm{PL}} e = \mathtt{scale}(\mathtt{return}(*), \lambda z \colon \mathtt{unit}.\ \mathbb{E}_{y \sim e}[1])} \qquad \frac{\Gamma \vdash e \colon D[\tau]}{\Gamma \vdash_{\mathrm{PL}} 0 \le \mathbb{E}_{y \sim e}[1] \le 1} \tag{25}$$

General Construction. We define a graded $\top\top$-lifting for the monadic type D. Consider a type ζ equipped with a preordered monoid structure $(\zeta, 1_\zeta, \cdot_\zeta, \le_\zeta)$, and an arbitrary type θ. A *lifting parameter* is a well-typed formula S satisfying $\Gamma, \mathbf{k} \colon \zeta, \mathbf{l} \colon D[\theta] \vdash S$ wf and the following monotonicity condition:

$$\Gamma, \mathbf{k} \colon \zeta, \mathbf{l} \colon D[\theta] \vdash_{\mathrm{PL}} \forall \alpha \colon \zeta . \forall \beta \colon \zeta . (\alpha \le_\zeta \beta \implies (S[\alpha/\mathbf{k}] \implies S[\beta/\mathbf{k}]))$$

Roughly speaking, a lifting parameter is a family of predicates $D[\theta]$ which is monotone with respect to the parameter in the preordered monoid ζ. The type θ can differ, depending on the application. For example, in the embedding of the union bound logic, the type θ is set to $\mathtt{unit}$.

For any assertion $\Gamma, \mathbf{r}' \colon \tau \vdash \phi$ wf and an expression $\Gamma \vdash \alpha \colon \zeta$, we define the *unary* $\top\top$*-lifting* $\Gamma, \mathbf{r} \colon D[\tau] \vdash \mathfrak{U}_S^\alpha \phi$ wf for the lifting parameter S by the following assertion.

$$\mathfrak{U}_S^\alpha \phi \equiv \forall \beta \colon \zeta . \forall f \colon \tau \to D[\theta] . ((\forall x \colon \tau . \phi[x/\mathbf{r}'] \implies S[\beta/\mathbf{k}, f(x)/\mathbf{l}]) \implies S[\alpha \cdot \beta/\mathbf{k}, \mathtt{bind}\, \mathbf{r}\ f/\mathbf{l}])$$

Notice that the parameters β and f range over *all* elements in the types ζ and $\tau \to D[\theta]$ respectively. By regarding $\mathfrak{U}_S$ as a constructor of graded $\top\top$-lifting, we obtain the following graded monadic rules.

Theorem 9.1 (Graded Monadic Laws of $\mathfrak{U}_S$). *The following rules are derivable:*

$$\Gamma \mid \Psi \vdash_{\mathrm{PL}} \forall \alpha\colon \zeta.\forall \beta\colon \zeta.(\alpha \leq_\zeta \beta) \implies \mathfrak{U}_S^\alpha \phi \implies \mathfrak{U}_S^\beta \phi$$

$$\Gamma \mid \Psi \vdash_{\mathrm{PL}} \forall \alpha\colon \zeta.(\forall x\colon \tau.\phi_1[x/\mathbf{r}'] \implies \phi_2[x/\mathbf{r}']) \implies (\mathfrak{U}_S^\alpha \phi_1 \implies \mathfrak{U}_S^\alpha \phi_2)$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e\colon \tau \mid \phi[\mathbf{r}/\mathbf{r}']}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{return}(e)\colon D[\tau] \mid \mathfrak{U}_S^{1_\zeta} \phi} \qquad \frac{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} e\colon D[\tau] \mid \mathfrak{U}_S^\alpha \phi \quad \Gamma \mid \Psi \vdash_{\mathrm{UPL}} e'\colon \tau \to D[\tau'] \mid \forall x\colon \tau.\phi[x/\mathbf{r}] \implies (\mathfrak{U}_S^\beta \phi')[\mathbf{r}x/\mathbf{r}]}{\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathtt{bind}\, e\, e'\colon D[\tau'] \mid \mathfrak{U}_S^{\alpha\cdot\beta} \phi'}$$

The proofs follow by unfolding the constructor $\mathfrak{U}_S$. Furthermore, the graded monadic laws (Theorem 9.1) are proved only using the structure of the preordered monoid for grading, the monotonicity of the lifting parameter, α-conversions, $\beta\eta$-reductions, the monadic laws of D, and proof rules of PL. Moreover, the construction of $\top\top$-lifting can be applied to any monadic type.

Embedding the Union Bound Logic. We show that the predicate lifting given in the semantic model of the union bound logic [Barthe et al. 2016b] can be implemented as a graded unary $\top\top$-lifting in PPV. Concretely, we give a lifting parameter S such that the graded $\top\top$-lifting $\mathfrak{U}_S$ corresponds to the predicate lifting for the union bound logic.

Consider the additive monoid structure with usual ordering $(\mathtt{pReal}, 0, +, \leq)$. We define the lifting parameter $\mathbf{k}\colon \mathtt{pReal}, \mathbf{l}\colon D[\mathtt{unit}] \vdash S$ wf by $S = (\mathbb{E}_{y\sim \mathbf{l}}[1] \leq \mathbf{k})$. The monotonicity of S is obvious. As we proved above, we have the monadic rules for the graded $\top\top$-lifting $\mathfrak{U}_S$. Next, we prove that the graded $\top\top$-lifting $\mathfrak{U}_S$ describes the probability of failure:

Proposition 9.2. *The following reduction is derivable in PL.*

$$\frac{\Gamma, \mathbf{r}'\colon \tau \vdash e\colon \mathtt{bool} \qquad \Gamma, \mathbf{r}'\colon \tau \mid \Psi \vdash_{\mathrm{PL}} \neg\phi \iff (e = \mathtt{true})}{\Gamma, \mathbf{r}\colon D[\tau] \mid \Psi \vdash_{\mathrm{PL}} \mathfrak{U}_S^\alpha(\neg\phi) \iff \Pr_{X\sim \mathbf{r}}[e[X/\mathbf{r}']] \leq \alpha}$$

Intuitively, this proposition holds because $\mathfrak{U}_S^\alpha(\neg\phi) \iff \Pr[\phi] \leq \alpha$. The second premise requires ϕ to be a measurable assertion, i.e., there is an indicator function $\lambda \mathbf{r}'\colon \tau.e$ of ϕ.

9.2 Embedding Relational $\top\top$-lifting

Similar to unary graded $\top\top$-lifting, we can also define relational graded $\top\top$-lifting by just switching the type of assertions from predicates to relations. As a concrete example, we instantiate the (non-graded) relational $\top\top$-lifting for reasoning about probabilistic coupling. Consider a preordered monoid $(\zeta, 1_\zeta, \cdot_\zeta, \leq_\zeta)$ and a pair of arbitrary types θ_1 and θ_2. A lifting parameter for relational $\top\top$-lifting is a well-typed formula of the form $\Gamma, \mathbf{k}\colon \zeta, \mathbf{l}_1\colon D[\theta_1], \mathbf{l}_2\colon D[\theta_2] \vdash S$ wf satisfying the following monotonicity condition:

$$\Gamma, \mathbf{k}\colon \zeta, \mathbf{l}_1\colon D[\theta_1], \mathbf{l}_2\colon D[\theta_2] \vdash_{\mathrm{PL}} \forall \alpha\colon \zeta.\forall \beta\colon \zeta.(\alpha \leq_\zeta \beta \implies (S[\alpha/\mathbf{k}] \implies S[\beta/\mathbf{k}])).$$

Intuitively, a lifting parameter for relational graded $\top\top$-lifting is a monotone family of relations between $D[\theta_1]$ and $D[\theta_2]$ with respect to the preordered monoid ζ.

For any assertion $\Gamma, \mathbf{r}'\colon \tau_1, \mathbf{r}'\colon \tau_2 \vdash \phi$ wf and an expression $\Gamma \vdash \alpha\colon \zeta$ we define its *relational lifting* $\Gamma, \mathbf{r}_1\colon D[\tau_1], \mathbf{r}_2\colon D[\tau_2] \vdash \mathfrak{R}_S^\alpha \phi$ wf for the lifting parameter S as the following assertion.

$$\begin{aligned}\mathfrak{R}_S^\alpha \phi \equiv \forall \beta\colon \zeta.\forall f_1\colon \tau_1 \to D[\theta_1].\forall f_2\colon \tau_2 \to D[\theta_2].\\ (\forall x_1\colon \tau_1.\forall x_2\colon \tau_2.\phi[x_1/\mathbf{r}_1', x_2/\mathbf{r}_2'] \implies S[\beta/\mathbf{k}, f_1(x_1)/\mathbf{l}_1, f_2(x_2)/\mathbf{l}_2])\\ \implies S[\alpha\cdot\beta/\mathbf{k}, \mathtt{bind}\,\mathbf{r}_1\ f_1/\mathbf{l}_1, \mathtt{bind}\,\mathbf{r}_2\ f_2/\mathbf{l}_2])).\end{aligned}$$

We can prove two-sided graded monadic laws for $\mathfrak{R}_S$, analogous to those for graded unary $\top\top$-lifting $\mathfrak{U}_S$. We omit these here.

Embedding the Modality for Relational Coupling of Distributions. As an example of this relational construction, we show how to internalize in our framework the modality for relational probabilistic coupling defined by Aguirre et al. [2018]. We say that two probability distributions μ_1 and μ_2 are *coupled* over a relation $R \subseteq X \times Y$ if $\forall S \subseteq X.\, \Pr_{x\sim\mu_1}[S] \leq \Pr_{y\sim\mu_2}[R(S)]$. To internalize this construction we now need to supply the appropriate lifting parameters. First, we take the grading monoid to be the trivial one on the unit type $\mathtt{unit}$. Then, we set the lifting parameter $\mathbf{k}\colon \mathtt{unit}, \mathbf{l}_1\colon D[\mathtt{unit}], \mathbf{l}_2\colon D[\mathtt{unit}] \vdash S$ wf by $S = (\mathbb{E}_{y\sim\mathbf{l}_1}[1] \leq \mathbb{E}_{y\sim\mathbf{l}_2}[1])$, which is equivalent to the usual inequality on $[0,1]$. The assertion S obviously satisfies the monotonicity of lifting parameter. Hence we obtain the $\top\top$-lifting $\Gamma, \mathbf{r}_1\colon D[\tau_1], \mathbf{r}_2\colon D[\tau_2] \vdash \Re_S\phi$ for the lifting parameter S. What we need to prove is that the lifting $\Re_S$ actually describes the above inequality of probabilistic dominance. In other words, we need to prove the following fundamental property in PL.

Proposition 9.3 (Aguirre et al. [2018, Lemma 2]).

$$\begin{aligned}\Gamma, \mathbf{r}_1\colon D[\tau_1], \mathbf{r}_2\colon D[\tau_2] \mid \Psi \vdash_{\mathrm{PL}} \Re_S\phi \implies{}& \forall f_1\colon \tau_1 \to \mathtt{bool}.\forall f_2\colon \tau_2 \to \mathtt{bool}.\\ & \forall y\colon \tau_2.((f_2(y) = \mathtt{true}) \implies \forall x\colon \tau_1.\phi[x/\mathbf{r}_1', y/\mathbf{r}_2'] \implies (f_1(x) = \mathtt{true}))\\ & \implies \Pr_{x\sim\mathbf{r}_1}[f_1(x)] \leq \Pr_{y\sim\mathbf{r}_1}[f_2(y)]\end{aligned}$$

Intuitively f_1 and f_2 encode indicator functions χ_A and χ_B respectively, where $\phi(A) \subseteq B$. The proof follows Katsumata and Sato [2015, Theorem 12], again using the equivalence $D[\mathtt{unit}] \cong [0,1]$ axiomatized in (25) and axioms on scaling of measures.

Specializing the assertion ϕ can establish useful probabilistic properties. For instance, taking ϕ to be the equality relation yields the following property.

Corollary 9.4 (Aguirre et al. [2018, Corollary 1]). *If $\tau_1 = \tau_2 = \tau$ then*

$$\begin{aligned}&\Gamma, \mathbf{r}\colon D[\tau] \mid \Psi \vdash_{\mathrm{PL}}\\ &\quad \Re_S(\mathbf{r}_1' = \mathbf{r}_2') \iff (\forall g\colon \tau \to \mathtt{real}.(\forall x\colon \tau.0 \leq g(x) \leq 1) \implies \mathbb{E}_{x\sim\mathbf{r}_1}[g(x)] \leq \mathbb{E}_{y\sim\mathbf{r}_2}[g(y)])\end{aligned}$$

If we take g to be the indicator function of a (measurable) set A, the conclusion shows that the measure of A in $\mathbf{r}_1$ is smaller than the measure of A in $\mathbf{r}_2$. Since the assertion ϕ is symmetric, we can also conclude the inequality in the other direction, hence showing that the measure of A must be equal in $\mathbf{r}_1$ and in $\mathbf{r}_2$. Since equality holds for all measurable A, $\mathbf{r}_1$ and $\mathbf{r}_2$ must denote equal probability measures.

10 RELATED WORK

Semantics of probabilistic programs. The semantics of probabilistic programs has been extensively studied starting from the seminal work of Kozen [1981]. Imperative first-order programs with continuous distributions have a well-understood interpretation based on the Giry monad [Giry 1982] over the category **Meas** of measurable spaces and measurable functions [Panangaden 1999]. However, this approach does not naturally extend to the higher-order setting since **Meas** is not Cartesian closed [Aumann 1961]. In addition, although **Meas** has a symmetric monoidal closed structure [Culbertson and Sturtz 2013], the Giry monad is not strong with respect to the canonical one [Sato 2018].

The category **QBS** [Heunen et al. 2017] of quasi-Borel spaces was introduced as an "extension" of **Meas** that is Cartesian closed and that can be used to interpret higher-order probabilistic programs with continuous distributions. The category of s-finite kernels [Staton 2017] gives a denotational semantics to observe-like statements in these models, including our construct `query`. In particular, it supports infinite measures and rescaling of measures. These are useful to give semantics to programs and to devise equational rules to reason about the equivalence of programs. The monad $\mathfrak{M}$ of measures on quasi-Borel spaces we use in this paper was introduced by Ścibior et al. [2017]

based on these constructions. One reason we chose **QBS** is that it has an obvious forgetful functor **QBS** $\to$ **Set** giving the identity on functions. This is a key property to allow set-theoretic reasoning in PPV.

An alternative approach has been proposed by Ehrhard et al. [2017]. They use a domain-theoretic approach based on the category **Cstab** of cones and stable functions, extending previous work on probabilistic coherent spaces [Ehrhard et al. 2014]. For our work, **QBS** is a more natural choice than **Cstab** for two reasons. First, the categorical structure needed for `query`-like statements has already been studied in **QBS**. Second, we are interested in terminating programs and so we do not need the domain-theoretic structure of **Cstab**. Other models related to both **QBS** and **Cstab** that one could consider are the ones by Tix et al. [2009] and Keimel and Plotkin [2017]. Several other papers have studied models for higher-order probabilistic programming starting from the seminal papers on probabilistic powerdomains by Jones and Plotkin [1989] and Saheb-Djahromi [1980]. A non-exhaustive list includes Castellan et al. [2018]; Goubault-Larrecq and Jung [2014]; Jung and Tix [1998]; Mislove [2017]; Varacca et al. [2004]. Many of these model only partially support the features we need. There is also recent work that studies the semantics of probabilistic programming from an operational perspective. Borgström et al. [2016] propose distribution-based and sample-based operational semantics for an untyped lambda calculus with continuous random variables. Their calculus also contains primitives for scaling and failing which allow them to model different kinds of `query`-like constructs. Culpepper and Cobb [2017] propose an entropy-based operational semantics for a simply typed lambda calculus with continuous random variables and propose an operational equational theory for it based on logical relations. The focus of their work is program equivalence, as reflected in the form of their judgments. In contrast, we start from an expressive predicate logic for probabilistic computations which allows us to express many different (unary and relational) properties, not just equivalence.

Verification of probabilistic programs. Starting from the seminal work on Probabilistic Propositional Dynamic Logic by Kozen [1985], several papers have proposed program logics for the verification of imperative probabilistic programs. McIver and Morgan [2005]; Morgan et al. [1996] propose a predicative logic to reason about an imperative language with probabilistic and nondeterministic choice. Both these program logics allow reasoning about the expected value of a single real-valued function on program states. Many subsequent papers build on this idea [Audebaud and Paulin-Mohring 2009; Gretz et al. 2013; Hurd et al. 2005; Kaminski et al. 2016; Katoen et al. 2010]. Other papers focus on program logics where the pre-condition and post-condition are probabilistic assertions about the input and output distributions Chadha et al. [2007]; den Hartog [2002]; Ramshaw [1979]; Rand and Zdancewic [2015]. Barthe et al. [2018] propose an assertion-based logic, named ELLORA, using expectation for verifying properties of imperative probabilistic programs with discrete random variables. Our assertion logic PL is similar in spirit to the one of ELLORA, but it further supports continuous distributions and the verification of higher-order programs. On the other hand, ELLORA has powerful rules for probabilistic while loops that PPV does not support. It would be interesting to explore if similar rules can also be added to PPV. Formalizations of measure and integration theory in general purpose interactive theorem provers have been considered in many papers [Audebaud and Paulin-Mohring 2009; Coble 2010; Hölzl and Heller 2011; Hurd 2003; Richter 2004]. Avigad et al. [2014] recently completed a proof of the Central Limit theorem, which is the principle underlying concentration bounds. Hölzl [2016] formalized discrete-time Markov chains and Markov decision processes. These and other existing formalizations have been used to verify several case studies, but they are scattered and not easily accessible for our purposes.

Relational Verification. Several papers have explored relational program logics or relational type systems for the verification of different relational properties. Aguirre et al. [2017] propose

UHOL/RHOL for the unary and relational verification of higher-order, non-probabilistic, terminating programs. UHOL and RHOL are based on a combination of logics for expressing (unary and relational) postconditions, and syntax-directed proof rules for establishing them. Since only terminating non-probabilistic programs are considered, the logic and the proof rules can be shown sound in set-theory. Our broad approach to setting up PPV is directly inspired from this work, but we work with probabilistic programs and, therefore, introduce a new monadic type for general/continuous measures along with constructs for conditioning. As a result, we have to interpret the logic and proof system in **QBS**, not set theory, and had to re-work the entire soundness proof from scratch.

The framework U^C/R^C [Radiček et al. 2017] is an extention of Aguirre et al. [2017] for reasoning about costs of non-probabilistic, terminating programs. This work introduces a monad, but this monad merely pairs a computation with its cost. The entire development still has a simple model in set theory. GUHOL and GRHOL [Aguirre et al. 2018] are extensions of Aguirre et al. [2017] to reason about unary and relational properties of Markov chains. These systems include a monad for distributions, but the development is limited to discrete distributions, and relational probabilistic reasoning is limited to coupling. The framework has an interpretation in the topos of trees (which is an extension of set theory with step counting) extended with a Giry monad. Importantly, pre- and post-conditions are *non*-probabilistic and are interpreted first over deterministic values, and then over distributions over values by lifting. Moreover, in [Aguirre et al. 2018], the proof rules only allow analysis of properties of programs via coupling arguments. This differs considerably from what we present here. Indeed, in PPV we can reason about (monadic) probabilistic expressions directly in assertions. For example, we can directly express and prove convergence properties of the expectation of an expression, which is impossible in the work of Aguirre et al. [2018]. In addition, [Aguirre et al. 2018] support only discrete distributions while we handle continuous distributions. As we have shown, the probabilistic coupling of GRHOL can be embedded in PPV by $\top\top$-lifting, but PPV does not cover all features of GRHOL. The reason is the difference in the goals of verification: PPV verifies the static behavior of probabilistic programs such as expected values and equality between probability measures. In contrast, GRHOL verifies behaviors of entire Markov chains.

Barthe et al. [2016a] study a relational type system PrivInfer for Bayesian inference on a functional programming language. Our framework PPV is more flexible since it supports continuous probability distributions while PrivInfer supports only discrete probabilities. In the future, we expect to internalize the continuous variant of PrivInfer's (f, δ)-lifting proposed in PPV, in a manner similar to $\top\top$-lifting.

11 CONCLUSION

In this paper we have introduced a framework PPV supporting the (unary and relational) verification of probabilistic programs including constructions for higher-order computations, continuous distributions and conditioning. PPV combines axiomatizations of basic probabilistic constructions with rules of three different logics in order to ease the verification of examples from probabilistic inference, statistics, and machine learning. The soundness of our approach relies on quasi-Borel spaces, a recently proposed semantic framework for probabilistic programs. All these components make PPV a useful framework for the practical verification of higher-order probabilistic programs.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under CCF Grant No. 1637532 and under CNS Grant No. 1565365.

REFERENCES

Alejandro Aguirre, Gilles Barthe, Lars Birkedal, Ales Bizjak, Marco Gaboardi, and Deepak Garg. 2018. Relational Reasoning for Markov Chains in a Probabilistic Guarded Lambda Calculus. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. 214–241. https://doi.org/10.1007/978-3-319-89884-1_8

Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2017. A Relational Logic for Higher-order Programs. *Proc. ACM Program. Lang.* 1, ICFP, Article 21 (Aug. 2017), 29 pages. https://doi.org/10.1145/3110265

Torben Amtoft and Anindya Banerjee. 2016. A Theory of Slicing for Probabilistic Control Flow Graphs. In *Foundations of Software Science and Computation Structures*, Bart Jacobs and Christof Löding (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 180–196.

Kavosh Asadi, Evan Cater, Dipendra Misra, and Michael L. Littman. 2018. Equivalence Between Wasserstein and Value-Aware Model-based Reinforcement Learning. *ArXiv e-prints* (June 2018). arXiv:cs.LG/1806.01265

Philippe Audebaud and Christine Paulin-Mohring. 2009. Proofs of Randomized Algorithms in Coq. *Sci. Comput. Program.* 74, 8 (2009), 568–589. https://doi.org/10.1016/j.scico.2007.09.002

Robert J. Aumann. 1961. Borel structures for function spaces. *Illinois J. Math.* 5, 4 (12 1961), 614–630. http://projecteuclid.org/euclid.ijm/1255631584

Jeremy Avigad, Johannes Hölzl, and Luke Serafin. 2014. A formally verified proof of the Central Limit Theorem. *CoRR* abs/1405.7012 (2014). http://arxiv.org/abs/1405.7012

Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An Assertion-Based Program Logic for Probabilistic Programs. In *Programming Languages and Systems*, Amal Ahmed (Ed.). Springer International Publishing, Cham, 117–144.

Gilles Barthe, Gian Pietro Farina, Marco Gaboardi, Emilio Jesús Gallego Arias, Andy Gordon, Justin Hsu, and Pierre-Yves Strub. 2016a. Differentially Private Bayesian Programming. In *ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 68–79. https://doi.org/10.1145/2976749.2978371

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016b. A Program Logic for Union Bounds. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. 107:1–107:15. https://doi.org/10.4230/LIPIcs.ICALP.2016.107

Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. 2011. Measure Transformer Semantics for Bayesian Machine Learning. In *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. 77–96. https://doi.org/10.1007/978-3-642-19718-5_5

Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*. 33–46. https://doi.org/10.1145/2951913.2951942

Simon Castellan, Pierre Clairambault, Hugo Paquet, and Glynn Winskel. 2018. The concurrent game semantics of Probabilistic PCF. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. 215–224. https://doi.org/10.1145/3209108.3209187

Rohit Chadha, Luís Cruz-Filipe, Paulo Mateus, and Amílcar C. Sernadas. 2007. Reasoning about probabilistic sequential programs. *Theoretical Computer Science* 379, 1 (2007), 142 – 165. https://doi.org/10.1016/j.tcs.2007.02.040

Sourav Chatterjee and Persi Diaconis. 2018. The sample size required in importance sampling. *Ann. Appl. Probab.* 28, 2 (04 2018), 1099–1135. https://doi.org/10.1214/17-AAP1326

Aaron R. Coble. 2010. *Anonymity, information, and machine-assisted proof.* Technical Report UCAM-CL-TR-785. University of Cambridge, Computer Laboratory.

Jared. Culbertson and Kirk. Sturtz. 2013. Bayesian machine learning via category theory. *ArXiv e-prints* (Dec. 2013). arXiv:math.CT/1312.1445

Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. 368–392. https://doi.org/10.1007/978-3-662-54434-1_14

Jeremy den Hartog. 2002. *Probabilistic extensions of semantical models.* Ph.D. Dissertation. Vrije Universiteit Amsterdam.

Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2017. Measurable Cones and Stable, Measurable Functions: A Model for Probabilistic Higher-order Programming. *Proc. ACM Program. Lang.* 2, POPL, Article 59 (Dec. 2017), 28 pages. https://doi.org/10.1145/3158147

Thomas Ehrhard, Christine Tasson, and Michele Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 309–320. https://doi.org/10.1145/2535838.2535865

Michèle Giry. 1982. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, B. Banaschewski (Ed.). Lecture Notes in Mathematics, Vol. 915. Springer Berlin Heidelberg, 68–85. https://doi.org/10.1007/BFb0092872

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*. 220–229. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1346&proceeding_id=24

Jean Goubault-Larrecq and Achim Jung. 2014. QRB, QFS, and the Probabilistic Powerdomain. *Electr. Notes Theor. Comput. Sci.* 308 (2014), 167–182. https://doi.org/10.1016/j.entcs.2014.10.010

Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2013. Prinsys - On a Quest for Probabilistic Loop Invariants. In *Quantitative Evaluation of Systems - 10th International Conference, QEST 2013*. 193–208.

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. 1–12. https://doi.org/10.1109/LICS.2017.8005137

Johannes Hölzl. 2016. Markov chains and Markov decision processes in Isabelle/HOL. (2016). http://home.in.tum.de/~hoelzl/mdptheory/hoelzl2016markov-draft.pdf

Johannes Hölzl and Armin Heller. 2011. Three Chapters of Measure Theory in Isabelle/HOL. In *Interactive Theorem Proving, ITP 2011 (Lecture Notes in Computer Science)*, Marko C. J. D. van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk (Eds.), Vol. 6898. Springer, 135–151.

Joe Hurd. 2003. *Formal verification of probabilistic algorithms*. Technical Report UCAM-CL-TR-566. University of Cambridge, Computer Laboratory.

Joe Hurd, Annabelle McIver, and Carroll Morgan. 2005. Probabilistic guarded commands mechanized in HOL. *Theor. Comput. Sci.* 346, 1 (2005), 96–112.

Bart Jacobs. 1999. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam.

Bart Jacobs and Thomas F. Melham. 1993. Translating Dependent Type Theory into Higher Order Logic. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*. 209–229. https://doi.org/10.1007/BFb0037108

Claire Jones and Gordon D. Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. 186–195. https://doi.org/10.1109/LICS.1989.39173

Achim Jung and Regina Tix. 1998. The troublesome probabilistic powerdomain. *Electr. Notes Theor. Comput. Sci.* 13 (1998), 70–91. https://doi.org/10.1016/S1571-0661(05)80216-6

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. arXiv:cs.LO/1601.01001

Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. 2010. Linear-Invariant Generation for Probabilistic Programs: Automated Support for Proof-Based Methods. 390–406. https://doi.org/10.1007/978-3-642-15769-1_24

Shin-ya Katsumata and Tetsuya Sato. 2015. Codensity Liftings of Monads. In *Conference on Algebra and Coalgebra in Computer Science (CALCO 2015) (Leibniz Intern. Proc. in Informatics (LIPIcs))*, Vol. 35. Schloss Dagstuhl, 156–170. https://doi.org/10.4230/LIPIcs.CALCO.2015.156

Shin-ya Katsumata. 2014. Parametric Effect Monads and Semantics of Effect Systems. In *ACM Symposium on Principles of Programming Languages (POPL '14)*. ACM, New York, NY, USA, 633–645. https://doi.org/10.1145/2535838.2535846

Klaus Keimel and Gordon D. Plotkin. 2017. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science* 13, 1 (2017). https://doi.org/10.23638/LMCS-13(1:2)2017

Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. System Sci.* 22, 3 (1981), 328 – 350. https://doi.org/10.1016/0022-0000(81)90036-2

Dexter Kozen. 1985. A Probabilistic PDL. *J. Comput. Syst. Sci.* 30, 2 (1985), 162–178.

A. McIver and C. Morgan. 2005. *Abstraction, Refinement, and Proof for Probabilistic Systems*. Springer.

Michael W. Mislove. 2017. Discrete Random Variables Over Domains, Revisited. In *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*. 185–202. https://doi.org/10.1007/978-3-319-51046-0_10

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (1996), 325–353.

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*. 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

Prakash Panangaden. 1999. The Category of Markov Kernels. *Electronic Notes in Theoretical Computer Science* 22 (1999), 171 – 187. https://doi.org/10.1016/S1571-0661(05)80602-4

Ivan Radiček, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Florian Zuleger. 2017. Monadic Refinements for Relational Cost Analysis. *Proc. ACM Program. Lang.* 2, POPL, Article 36 (Dec. 2017), 32 pages. https://doi.org/10.1145/3158124

Lyle Harold Ramshaw. 1979. *Formalizing the Analysis of Algorithms.* Ph.D. Dissertation. Computer Science.

Robert Rand and Steve Zdancewic. 2015. VPHL: A Verified Partial-Correctness Logic for Probabilistic Programs. In *Mathematical Foundations of Program Semantics (MFPS XXXI)*.

Stefan Richter. 2004. Formalizing Integration Theory with an Application to Probabilistic Algorithms. In *Theorem Proving in Higher Order Logics, 17th International Conference, (TPHOL) 2004 (Lecture Notes in Computer Science)*, Konrad Slind, Annette Bunker, and Ganesh Gopalakrishnan (Eds.), Vol. 3223. Springer, 271–286.

Nasser Saheb-Djahromi. 1980. CPO'S of Measures for Nondeterminism. *Theor. Comput. Sci.* 12 (1980), 19–37. https://doi.org/10.1016/0304-3975(80)90003-1

Tetsuya Sato. 2018. The Giry monad is not strong for the canonical symmetric monoidal closed structure on Meas. *Journal of Pure and Applied Algebra* 222, 10 (2018), 2888 – 2896. https://doi.org/10.1016/j.jpaa.2017.11.004

Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. 2017. Denotational Validation of Higher-order Bayesian Inference. *Proc. ACM Program. Lang.* 2, POPL, Article 60 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158148

Chung-chieh Shan and Norman Ramsey. 2017. Exact Bayesian inference by symbolic disintegration. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. 130–144.

Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. 855–879. https://doi.org/10.1007/978-3-662-54434-1_32

Regina Tix, Klaus Keimel, and Gordon D. Plotkin. 2009. Semantic Domains for Combining Probability and Non-Determinism. *Electr. Notes Theor. Comput. Sci.* 222 (2009), 3–99. https://doi.org/10.1016/j.entcs.2009.01.002

Daniele Varacca, Hagen Völzer, and Glynn Winskel. 2004. Probabilistic Event Structures and Domains. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*. 481–496. https://doi.org/10.1007/978-3-540-28644-8_31

Frank D. Wood, Jan-Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*. 1024–1032. http://jmlr.org/proceedings/papers/v33/wood14.html