Semantics of Probabilistic Program Traces

Alexander K. Lew
MIT
alexlew@mit.edu

Jan-Willem van de Meent
University of Amsterdam
j.w.vandemeent@uva.nl

Eli Sennesh
Northeastern University
esennesh@ccs.neu.edu

Vikash K. Mansinghka
MIT
vkm@mit.edu

1 Introduction

Reifying random choices that probabilistic programs make into traces has long been an important technique for both implementors [13, 14, 20, 21] and theorists [2, 16] of PPLs. But recently, many languages [1, 3, 4, 10, 17] have made traces a user-facing concern, rather than an implementation detail: users label each random sample in their programs with a name, and traces are dictionaries recording the names and values of all random choices encountered during program execution. Traces are often meant to be inspected, and explicitly reasoned about; for example, when invoking inference algorithms, users may have to ensure certain properties of the distributions over traces their programs encode, a task for which several static analyses have been proposed [7–9, 18]. In developing these analyses, researchers have defined restricted formal calculi for modeling aspects of modern, trace-based languages. But these calculi are crafted specially to model only the details that are relevant for a particular analysis, and as such, feature restrictions that make them unsuitable for "off-the-shelf" reuse when reasoning about new analyses or program transformations. For the same reason, they also fail to capture some interesting aspects of real-world PPLs.

This extended abstract presents a minimal language (Fig. 1) modeling modern, trace-based PPLs, and a denotational semantics that assigns to each program an s-finite measure over a space of high-level traces (Fig. 2). It then demonstrates how to use this formal calculus to define program transformations and prove them sound, using as examples weighted sampling and density evaluation (Figs. 3 and 4). Finally, it shows how to reason about higher-level inference algorithms that compose these program transformations, via the example of importance sampling with a custom proposal (Fig. 5).

We aim to: (1) give theorists a self-contained model of a typical trace-based PPL, (2) give PPL developers a blueprint for formal reasoning about program semantics & new program transformations, and (3) provide a common reference point for discussion and comparison of many real-world PPLs.

2 Denotational Semantics

Our language is a simply-typed $\lambda$-calculus (Fig. 1) extended with minimal constructs for building traced probabilistic

Figure 1. Grammar and selected typing rules of our calculus. The symbol $c$ ranges over constants of any type, including ground types (e.g., $\texttt{false}$ or 3.14), distribution types (e.g., primitives for Gaussian and Bernoulli distributions), and function types (built-in operations, including projections $\pi_1$, $\pi_2$ for pairs, and if $\tau : \mathbb{B} \to (1 \to \tau) \to (1 \to \tau)$ to $\tau$.

We write \texttt{let} $x = t_1$ in $t_2$ as sugar for $(\lambda x. t_1) t_2$.

Semantics of types (quasi-Borel spaces) \( \mathbb{B} \models \texttt{B} \models [t] = 1 \)

\[
\begin{align*}
[D \sigma] & := M \sigma \\
[M \tau] & := M \tau \times \tau \to \tau \\
[\text{Str}] & := \mathbb{R} \times [\text{Str}] \\
[\text{Trace}] & := \mathbb{R} \times \tau_1 \times \tau_2 \\
\end{align*}
\]

Semantics of terms (quasi-Borel functions mapping environments to values)

\[
\begin{align*}
(c)(y) & := c \quad c : \mathbb{B} \\
(x)(y) & := x \quad x : \mathbb{B} \\
\lambda x. t & := \text{lift}(\lambda x. t) \quad \text{lift} : \text{lift}(\lambda x. t)(x \mapsto a) \\
\end{align*}
\]

\[
\begin{align*}
[t_1 t_2](y) & := [t_1](y) [t_2](y) \\
\lambda v. t & := \text{lift}(\lambda v. t) \quad \text{lift} : \text{lift}(\lambda v. t)(v \mapsto d, \lambda u. [t]) \\
\end{align*}
\]

\[
\begin{align*}
\text{sample}(t_1, t_2)(y) & := (\text{lift}[[t_2]](y) \mapsto t_1)(y, dx, \lambda v. [t_2](y)) \\
\text{factor}(t)(y) & := (e^{f[t]}(y) \circ \delta)[t] \quad \text{lift} : \text{lift}(\lambda t. f[t])(y) \\
\end{align*}
\]

\[
\begin{align*}
\text{do}(x \leftarrow t; m) & := \left( \text{lf}(d) \mapsto u \mapsto ([t](u, v) \circ \delta)[m][([t_1](y)(v))]) \right)(y \mapsto m) \\
\lambda u. [m](y) & := \left( \text{lf}(d) \mapsto u \mapsto ([t_1](y)(v)) \right)(y) \\
\lambda u. [m](y) & := \left( \text{lf}(d) \mapsto u \mapsto ([t_1](y)(v))) \right)(y) \\
\end{align*}
\]

Figure 2. Our semantics, interpreting types as quasi-Borel spaces and terms as quasi-Borel functions. Primitive distributions $D \sigma$ are interpreted as measures over the space denoted by $\sigma (M \sigma)$, whereas compound probabilistic programs $M \tau$ are interpreted as pairing a measure over traces $(M \tau)$ with a "value function" mapping traces to outputs $(T \to \tau)$. programs: \texttt{sample}(dist, name) for drawing a named sample from a primitive distribution, and \texttt{factor}(w) for factoring a non-negative number $w$ into the likelihood. Our semantics
(Fig. 2) interprets each type \( r \) in the language as a quasi-Borel space \([r]\) (an alternative to measurable spaces suitable for higher-order PPL semantics [5]). We highlight several key points, and refer the reader to Appendix B for full details:

- A novel contribution is a quasi-Borel space \( \mathcal{T} \) of traces. Previous work has modeled traces as lists [16] or trees [13] of reals, recording primitive uniform draws. But to model the density calculations in many PPLs, traces must record the heterogeneous values returned by diverse primitive distributions, e.g. Bernoulli booleans and Gaussian reals. Our traces are dictionaries mapping string names to heterogeneous values of ground type. The syntax \{\} builds an empty trace, \((t_1 \mapsto t_2)\) builds a trace mapping name \( t_1 \) to value \( t_2 \), and \(\text{concat} \ t_1 \ t_2\) (which we abbreviate \( t_1 \uplus t_2 \)) concatenates two traces (or returns the empty trace if names overlap). The primitive \(\text{pop}_x\), \( t_1 \ t_2 \) looks up the name \( t_2 \) in the trace \( t_1 \), and if it finds a value of type \( \sigma \), returns it, and, if a trace containing the remainder of the entries; otherwise, it returns a default value of type \( \sigma \) and the empty trace.

- The type \( M \) of probabilistic programs is interpreted in the quasi-Borel space \( \mathcal{M}\mathcal{T} \times (\mathcal{T} \to [r]) \): a program denotes both a quasi-Borel measure over traces, and a function from traces to return values. The probabilistic program \( \text{return} \ t \) that deterministically computes the term \( t \) denotes a Dirac distribution over the empty trace \{\}, together with a function mapping the empty trace to \([r]\). The program \( \text{sample}(t_1, t_2) \) denotes a probability distribution over singleton traces, and when programs are sequenced using \( \text{do} \), their traces concatenate. Programs that use a name twice denote the zero measure.

### 3 Program Transformations

PPLs typically automate core operations that form a computational interface to the measure a program denotes, by transforming the program or interpreting it in a non-standard way [19]. Figs. 3 and 4 show transformations implementing the following simple interface, for a measure \( \mu \) over traces:

- **Density evaluation**: given a trace \( u \), evaluate \( \frac{d\mu}{d\mathcal{B}_{\text{Trace}}}(u) \), where \(\mathcal{B}_{\text{Trace}} \) is a base measure over \( \mathcal{T} \) (Appendix C).
- **Weighted sampling**: sample a pair \((u, w)\) of a trace and weight s.t. \( E[w \cdot 1_A(u)] = \mu(A) \) for measurable \( A \subseteq \mathcal{T} \).

Similar transformations are well-known in the PPL community, and can be validated using various proof techniques, including via Scibior et al. [16]’s framework for “inference transformations,” or by modeling them with algebraic effects [11, 12, 15]. In Appendix C, we prove the following correctness result using logical relations:

**Proposition 3.1.** Let \( (\mu, f) = [\rho] \) for some \( \rho : M \tau \). Then:

- \( \llbracket \text{density}(p) \rrbracket \) is a density of \( \mu \) with respect to \( \mathcal{B}_{\text{Trace}} \), and
- \( \mathcal{V}(\text{wsamp}(p)) \) (where \( \mathcal{V}(\mu, f) = f_*\mu \)) is a probability measure, and \( \mathcal{I}(w \odot \delta_u) (\mathcal{V}(\text{wsamp}(p))) (d(u, w)) = \mu \).

**Top-level wrapper** \( (t : M \tau) \mapsto \text{density}(t) : \text{Trace} \to \mathbb{R}) \)

**Density** \( t \) := \( \lambda \hat{\mu}. \text{let} \ (w, \hat{u}) = t \in \text{isempty}(u') \cdot w \)

**Transforming types** (identity on ground types \( \sigma \))

\[ \rho(D \sigma) = \sigma \quad \rho(M \tau) = \text{Trace} \to \mathbb{R} \times \rho(\tau) \to \text{Trace} \]

\[ \rho(t_1 \times t_2) = \rho(t_1) \times \rho(t_2) \quad \rho(t_1 \rightarrow t_2) = \rho(t_1) \to \rho(t_2) \]

**Transforming terms**

\[ \rho(c) = c \rho(x) := x \quad \rho(\lambda x.t) := \lambda \hat{\mu}. \text{return} t \]

\[ \rho(\text{sample}(t_1, t_2)) := \lambda \hat{\mu}. \text{let} \ (\hat{w}, \hat{u}) = \text{pop}_x \ t_2 \text{ in } \text{when} t_1 : D \sigma] \]

\[ \rho(\text{do}(t)) := \rho(t) \quad \rho(\text{return} t) := \lambda \hat{\mu}. (\text{return} t, \hat{u}) \]

**Top-level wrapper** \( (t : M \tau) \mapsto \text{wsamp}(t) : \text{M} \text{(Trace} \times \mathbb{R}) \)

\[ \text{wsamp}(t) := \text{do}((u, w) \leftarrow \omega(t); \text{return} (u, w)) \]

**Transforming types** (identity on ground types \( \sigma \))

\[ \omega(D \sigma) = D \sigma \quad \omega(M \tau) = \text{M} \text{(Trace} \times \mathbb{R} \times \omega(t)) \]

\[ \omega(t_1 \times t_2) := \omega(t_1) \times \omega(t_2) \quad \omega(t_1 \rightarrow t_2) := \omega(t_1) \rightarrow \omega(t_2) \]

**Transforming terms**

\[ \omega(c) = c \omega(x) := x \quad \omega(\lambda x.t) := \lambda x. \omega(t) \]

\[ \omega(\text{sample}(t_1, t_2)) := \text{do}(x \leftarrow \text{sample}(t_1, t_2); \text{return} ((t_1 \mapsto x), 1, x)) \]

\[ \omega(\text{do}(t)) := \omega(t) \quad \omega(\text{return} t) := \omega(t) \quad \omega(\text{return} t) := \omega(t) \]

**Top-level wrapper** \( (t : M \tau) \mapsto \text{wsamp}(t) : \text{M} \text{(Trace} \times \mathbb{R}) \)

\[ \text{wsamp}(t) := \text{do}((\hat{w}, \hat{u}) \leftarrow \omega(t); (\hat{i}, \hat{w}, \hat{u}) \leftarrow \omega(\text{do}(m)); \text{return} (\hat{i}, \hat{w}, \hat{u})) \]

**Figure 3.** Trace density program transformation.

**Figure 4.** Weighted trace sampler program transformation.

**Figure 5.** Importance sampling with a custom proposal.

### 4 Sound Inference

Inference algorithms can be built using the operations presented in Sec. 3. For example, Fig. 5 implements an importance sampler targeting \( p \) but using \( q's \) sampler as a proposal:

**Proposition 4.1.** Let \( \nu : M \tau \), let \( \mu_p, \mu_q \) be the measures they denote, and let \( \nu = \mathcal{V}(\text{importance}(p, q)) \), a probability measure over \( \mathbb{T} \times \mathbb{R}_{\geq 0} \). If \( \mu_q \neq 0 \) and \( \mu_p \ll \mu_q \), then \( \pi_1 \nu = \pi_1(\mathcal{V}(\text{wsamp}(q))) \), and \( \mathcal{I}(\pi_2(x) \odot \delta_{\nu(x)}(dx)) = \mu_p \).
5 Discussion

An interesting direction for future work, which we sketch in Appendix E, is to understand today’s trace-based PPL landscape in terms of how each PPL extends Fig. 1’s language, and what computational interface it exposes to the measures denoted by programs. The choice of computational interface affects both the modeling constructs the PPL can expose (e.g., we could not expose a tractable density operation if our language featured a normalize construct), and the expressible inference algorithms (e.g., HMC cannot be implemented against Sec. 3’s interface, as it requires gradients).

References

Appendix

A Background and Notation

Quasi-Borel Spaces. We carry out our development in the category of quasi-Borel spaces [5], an alternative to measurable spaces which we review briefly here.

Definition A.1 (quasi-Borel space). A quasi-Borel space $X$ is a tuple $([X], M_X)$ of a carrier set $|X|$ and a set $M_X \subseteq [\mathbb{R} \rightarrow |X|]$ of admissible random elements, satisfying:

- (Closure under measurable precomposition.) If $\phi \in M_X$ and $f : \mathbb{R} \rightarrow M_X$ is measurable, $\phi \circ f \in M_X$.
- (Constant maps.) If $\phi = \lambda r.x$ for some $x \in |X|$, then $\phi \in M_X$.
- (Closure under piecewise gluing.) If $\{A_i\}_{i \in \mathbb{N}}$ is a countable partition of $\mathbb{R}$ and there exist $\phi_i \in M_X$ such that $\phi(r) = \phi_i(r)$ whenever $r \in A_i$, then $\phi \in M_X$.

Definition A.2 (quasi-Borel function). If $X$ and $Y$ are quasi-Borel spaces, a quasi-Borel function $f : X \rightarrow Y$ is a function from $|X|$ to $|Y|$ satisfying the property that for all $\phi \in M_X$, $f \circ \phi \in M_Y$.

Example A.3 (Standard Borel Spaces). Any standard Borel space (measurable space that is either finite, countable, or measurable isomorphic to $\mathbb{R}$) is also a quasi-Borel space, by choosing $M_X$ to be the measurable functions from $\mathbb{R}$ to $X$. The quasi-Borel functions between standard Borel spaces are exactly the measurable functions between them as measurable spaces.

Definition A.4 (quasi-Borel measure). Let $X$ be a quasi-Borel space. A quasi-Borel measure on $X$ is an equivalence class of $(\mu, \alpha)$ pairs, where $\mu$ is a (measure-theoretic) measure on $\mathbb{R}$, and $\alpha \in M_X$. Two pairs $(\mu_1, \alpha_1)$ and $(\mu_2, \alpha_2)$ are equivalent if for all quasi-Borel $f : X \rightarrow \mathbb{R}_{\geq 0}$, $\int f(\alpha_1(x)) \mu_1(dx) = \int f(\alpha_2(x)) \mu_2(dx)$.

Proposition A.5. There is a strong commutative monad $\mathcal{M}$ in QBS, taking $X$ to the space $M_X$ of measures on $X$.

Notation. Following Scibior et al. [16], we use synthetic measure theory notation to concisely write down quasi-Borel measures:

- For $x \in |X|$, we write $\delta_x$ for the Dirac delta distribution at $x$. (We can represent it as a quasi-Borel measure by taking $\mu = U(0,1)$ and $\alpha = \lambda r.x$.)
- For $\mu \in [M X]$, and $w : X \rightarrow \mathbb{R}_{\geq 0}$, we write $w \circ \mu$ for the measure with density $w$ with respect to $\mu$. For $w \in \mathbb{R}_{\geq 0}$, we abuse notation and write $w \circ \mu$ instead of $(\lambda x.w) \circ \mu$.
- For $f : X \rightarrow \mathbb{R}_{\geq 0}$ and $\rho = [(\mu, \alpha)] \in [M X]$, we write $\int_X f(x)\mu(dx)$ for the integral $\int f(\alpha(r))\mu(dr)$.
- If $k : X \rightarrow MY$, then we write $K(x, dy)$ in integral expressions, rather than $k(x)(dy)$.
- For $k : X \rightarrow MY$, we write $\int_X k(x)\mu(dx)$ for the measure over $Y$ that integrates a function $g : Y \rightarrow \mathbb{R}_{\geq 0}$ by computing $\int \int g(y)p(dx)k(x, dy)$.
- For $f : X \rightarrow Y$, and $\mu : MY$, $f \circ \mu : MY$ is the pushforward of $\mu$ by $f$, integrating a function $g : Y \rightarrow \mathbb{R}_{\geq 0}$ by integrating $g \circ f$ under $\mu$.

B Core Calculus Syntax and Semantics: Details

Language syntax: types. Our language (Fig. 1) has as ground types $1$ (the singleton type), the Booleans $\mathbb{B}$, the reals $\mathbb{R}$, the strings $\text{Str}$, tuples $\sigma_1 \times \sigma_2$ of other ground types, and a new type of traces, which we discuss below. In addition, the language features function types $\tau_1 \rightarrow \tau_2$, types $D \sigma$ of primitive distributions over each ground type $\sigma$, and monadic types $M \tau$ representing "traced probabilistic computations returning $\tau."$

Language syntax: terms. Our language’s terms include constants $c$ of various types (e.g., $\text{false}$ is a Boolean constant, $3.14$ is a real-value constant, $\text{normal}$ is a constant of type $\mathbb{R} \times \mathbb{R} \rightarrow D \mathbb{R}$, and $+$ is a constant of type $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$), variables $x$, function terms $\lambda x.t$, and function application expressions $t_1.t_2$ (where $t_1$ has function type, and $t_2$ is its argument). We also include syntax for constructing empty ($\{\}$) and singleton ($\{\text{name} \leftrightarrow \text{value}\}$) traces, and a standard $\text{let}$ expression for defining local variables. For probabilistic programming, $\text{return}$ $t$ constructs a deterministic probabilistic program of type $M \tau$ that just computes a value of type $\tau$; $\text{sample}(t_1, t_2)$ is the program that samples from distribution $t_1$ at name $t_2$; $\text{factor}(t)$ factors a non-negative real weight into the likelihood, creating a possibly unnormalized probabilistic program; and $\text{do}(x \leftarrow t_1; m)$ is used to build larger probabilistic computations, that first run a computation $t_1$, assigning $x$ to the result, then run the remainder of the computation $m$. (Our syntax is inspired by Haskell’s $\text{do}$-notation for sequencing monadic computations.)

Semantics of types. Our semantics (Fig. 2) interprets each type $\tau$ as a quasi-Borel space $\llbracket \tau \rrbracket$. Our ground types are all interpreted as standard Borel spaces, measurable spaces that are either finite, countable, or isomorphic to $\mathbb{R}$, and as such extend canonically to quasi-Borel spaces: $\llbracket \mathbb{R} \rrbracket = (\mathbb{R}, B(\mathbb{R}))$, $\llbracket \text{Str} \rrbracket = (\text{Str}, \mathcal{P}(\text{Str}))$, $\llbracket \mathbb{B} \rrbracket = (\{\text{True, False}\}, \mathcal{P}(\{\text{True, False}\}))$,
\[ [1] = \{(1, 0), (0, 0)\} \text{ and } [\sigma_1 \times \sigma_2] = \{[\sigma_1] \times [\sigma_2]\}. \]

For the ground type \( \text{Trace} \), we define a new standard Borel space, \( \mathcal{T} \) see below. Our semantics interprets function types as quasi-Borel function spaces (just as in Scibor et al. [16]), and for the types \( D \sigma \) of distributions over ground types, we have \([D \sigma] = M[\sigma]\), the quasi-Borel space of measures over \([\sigma]\). Because \([\sigma]\) is always standard Borel, these are just the ordinary (measure-theoretic) measures over \([\sigma]\).

Importantly, our semantics interprets monadic probabilistic programs \( M \tau \), not as (directly) denoting quasi-Borel measures over \([\tau]\), but rather as denoting pairs containing a quasi-Borel measure over traces, and a \( \mathcal{T} \rightarrow [\tau] \) function mapping traces to return values. If \( t \) is a term of type \( M \tau \), we write \( \mathcal{V}[\tau] \) for the marginal measure over \( \tau \) that "forgets the trace," \( \mathcal{V}[\tau] = \pi_\tau([\tau]), (\sigma_1([\tau])) \). However, a key thesis of this work is that it is useful to keep around the specific distribution over traces implemented by the user’s program, and not treat all programs with equivalent marginal output distributions as equivalent.

**QBS \( \mathcal{T} \) of traces.** We define a measurable space of traces that is standard Borel, and thus is also a quasi-Borel space. First, we define a set \( S \) of trace shapes, lexicographically sorted lists of \((k, \tau)\) pairs, where \( k \in \text{Str} \) is a string-valued name, \( \tau \) is a ground type, and no name appears more than once in the list. Then, for \( i \geq 0 \), we define \( T_i = \{(s, u) \mid s \in S \land u \in \mathcal{X}_{(i, k) \in S}[\sigma_i]\}, \) where \([\cdot]\) is an inductively defined family of (SBS-valued) semantic functions: we have \([\mathbb{R}]_i = (\mathbb{R}, \mathcal{B}(\mathbb{R})), [\text{Str}]_i = (\text{Str}, \mathcal{P}(\text{Str})), [\mathbb{B}]_i = ((\text{True}, \text{False}), \mathcal{P}(\text{True}, \text{False})), [1]_i = ((()), \mathcal{P}((()))), [\sigma_1 \times \sigma_2]_i = [\sigma_1]_i \times [\sigma_2]_i. \] The traces \( \mathcal{T} \) is a measurable space constructed inductively. For the ground type Trace, we have \( \mathcal{T} \) measurable as a subset of \( \mathcal{X}_{(i, k) \in S}[\sigma_i] \).

We expose several primitive functions for dealing with traces:

- \( \text{has}_\sigma : \text{Trace} \rightarrow \text{Str} \rightarrow \mathbb{R} \): returns 1 if the given trace has a value of type \( \sigma \) at the given name, 0 otherwise.
- \( \text{pop}_\sigma : \text{Trace} \rightarrow \text{Str} \rightarrow \sigma \times \text{Trace} \): if the given trace has a value of type \( \sigma \) at the given name, return it, along with a modified trace that deletes that entry. Otherwise, return a default value of type \( \sigma \) in the current environment
- \( \text{Trace}(), \text{Trace}(\{\}) \) for \( M \tau \) which have as denotations pairs of a measure over traces, and a function value mapping a trace to the expression’s value under that trace. The terms of type \( M \tau \) are:

- return \( t \), which denotes a Dirac distribution over empty traces (because it makes no random choices), together with the value function that ignores the input trace and just returns \([ \tau ](y)\) (the value of \( t \) in the current environment \( y \)).

- sample\((t_1, t_2)\), which denotes the marginal distribution arising by sampling \( x \sim \{t_1\}(y) \) (recall that \( t_1 : D \sigma \) denotes a primitive distribution), and returning the trace \( \{t_2\}(y) \mapsto x \). The return value function takes a trace \( u \), and uses \( \text{pop}_\sigma \) to look up the name \( [t_2]\)(y) \) in the trace, and return the value found there.

- factor \( t \) factors the exponent of a given log weight into the density. As such, its measure over traces is the dirac measure over \( \{\} \), scaled by \( e^{f(t)} \). Its return value function trivially returns \( () \), the empty tuple of type 1.

- do\((x \leftarrow t; m)\) sequences two probabilistic computations. The measure over traces that it denotes first generates a trace \( u_1 \sim \pi_1([\tau])(y) \), computes the return value \( v = \pi_2([\tau])(y)(u) \), then generates the rest of the trace \( u_2 \sim \pi_1([\tau])(y)(x \mapsto v) \). If \( u_1 \) and \( u_2 \) do not have disjoint sets of trace names (which is checked by the disjunction primitive), the zero measure is returned, indicating error. Otherwise, the concatenation \( u_1 + u_2 \) is returned. The value function passes a given trace into \([\tau](y)\)'s return value function to get \( v \), then into \([\text{do}(m)](y)(x \mapsto v)\)'s return value function to get a final return value.

**C Computational Interface: Details**

**Reasoning about the density transformation.** First, in order to talk rigorously about densities, we need to define the reference measures with respect to which densities are computed:

\[1^\text{This} inductive definition is designed to allow traces to contain other traces, which is one way of modeling hierarchical addresses like those used in Gen [3]. If only simple ground types (strings, reals, Booleans) were allowed, the inductive definition would not be required, and we could simply set \( \mathcal{T} = T_0 \).]
Definition C.1. For each ground type in our language, we assign a base measure \( B_\sigma \); for the reals we choose the Lebesgue measure, for 1, Str, and \( B \), we choose the counting measure, and for \( \tau_1 \times \tau_2 \), we choose the product of \( B_{\tau_1} \) and \( B_{\tau_2} \). For Trace, we first define base measures for the sets \( T_i \), considered in the previous section: in particular, we choose the measure that assigns to a subset \( A \subseteq T_i \), the measure \( \sum_{e \in A} (N_{(k,\sigma)} \circ E)(\{ (u, v) \mid (u, v) \in A \}) \), where \( E_{\text{Trace}} \) is interpreted as the base measure for \( T_{i-1} \). Then, for the base measure over the space of all traces, for \( A \subseteq T_i \), we set \( E_{\text{Trace}}(A) = \sum_{t \in A} E_{T_i} = \{ t \in A \mid t \in T_i \land \forall j < i, t \notin T_j \} \).

Now, we can establish the correctness of the density program transformation given in Fig. 3. The main translation itself is one line: it translates a term \( t : M \tau \) into a term \( \triangledown t : \tau \rightarrow \rho(t) : \rightarrow \mathbb{R} \). But the top-level translation relies crucially on a “macro” \( \rho \), that compositionally rewrites the program according to the rules in Fig. 3. The macro \( \rho \) operates on terms, but Fig. 3 also defines it on types; the invariant is that given a term of type \( \tau \), \( \rho(t) \) will translate it into a term of type \( \tau \).

Logical relations for correctness of the density macro. What is the goal of this macro? We first clearly define a specification for \( \rho \). For each type \( \tau \), we define a relation on \( [\tau] \times \left[ \rho(\tau) \right] \), which is meant to capture what it would mean for \( \rho \) to be doing its job correctly: if \( s : \rho(\tau) \) is a correct translation of \( t : \tau \), then \( [\tau], [s] \) should be related. The relation is defined as follows:

- For ground types \( \sigma \), \( R_{\sigma} := \{ (x, x) \mid x \in \sigma \} \). This reflects that the translation \( \rho \) does nothing to terms of ground type—a value is a ‘correct translation’ of another value only if the two values are exactly equal.
- For distribution types \( D \sigma, R_{D\sigma} := \{ (d, p) \mid p \circ B_\sigma = d \} \). In other words, a distribution \( d \in D \sigma \) is related to a function \( p \in \mathbb{R} \) if \( p \) is a correct density of \( d \).
- For probabilistic program types \( M \tau, R_{M\tau} := \{ (\mu, f, p) \mid (\text{isempty} \circ \pi_3 \circ p) \circ (\pi_1 \circ p) \circ E_{\text{Trace}} = \mu \land (f \circ \pi_2 \circ p) \in \mathcal{R}_\tau \land \forall u \in \{ u \in \mathcal{T} \mid \pi_1(p(u)) = \} \}, \forall t \in \mathcal{T} \mid \text{disj}(u, t), \pi_1(p(u + t)) = t \}. \) Unpacking this, we see that it is a relation between the denotations of probabilistic programs (containing a measure \( \mu \) over traces and a function \( f \) that computes \( \tau \) values based on traces) and their translations under \( \rho \), into functions \( p : \tau \rightarrow \mathbb{R} \times \rho(\tau) \times \mathcal{T} \). It places three requirements on the translation \( p \):
  1. \( (\text{isempty} \circ \pi_3 \circ p) \circ (\pi_1 \circ p) \circ E_{\text{Trace}} = \mu \) and \( (\mu, f, p) \in \mathcal{R}_{\tau} \).
  2. \( f \circ \pi_2 \circ p \) is a distribution over \( \mathcal{R}_\tau \).
  3. \( \forall u \in \{ u \in \mathcal{T} \mid \pi_1(p(u)) = \} \}, \forall t \in \mathcal{T} \mid \text{disj}(u, t), \pi_1(p(u + t)) = t \). This requirement states that the third return value from \( p \) returns the “consummed” part of the input trace. More precisely, for any trace \( u \) on which \( p \) returns an empty “unconsumed” trace, if we extend \( u \) with extra choices \( t \), we expect \( p \)’s third return value to just be \( t \).
- For product types \( \tau_1 \times \tau_2, R_{\tau_1 \times \tau_2} := \{ ((x, y), (x, \pi_2(y))) \mid (x, \pi_2(y)) \in \mathcal{R}_{\tau_1} \land (y, \pi_2(y)) \in \mathcal{R}_{\tau_1} \} \).
- For function types \( \tau_1 \rightarrow \tau_2, R_{\tau_1 \rightarrow \tau_2} := \{ (f(x), f(y)) \mid (x, \pi_2(y)) \in \mathcal{R}_{\tau_1} \land (f(x), \pi_2(y)) \in \mathcal{R}_{\tau_1} \} \).

We now show what is often called the fundamental lemma of a logical relations argument:

**Lemma C.2.** For every term \( \Gamma \vdash t : \tau \) in our language, and every environment \( (\gamma, \gamma_p) \in \mathcal{R}_\tau \), for every \( \rho(\Gamma) \vdash \rho(t) : \rho(\tau) \) \( (\Gamma \vdash t : \tau) \), \( \left[ \rho(\Gamma) \vdash \rho(t) : \rho(\tau) \right] \in \mathcal{R}_\tau \).

**Proof sketch.** We proceed by induction on the language’s syntax:

- For constants \( c : \tau \), we assume given constants \( c_\rho : \rho(\tau) \) satisfying \( (c, c_\rho) \in \mathcal{R}_\tau \). In particular, this means that primitive distributions, such as \( \text{flip} : \mathbb{R} \) must come equipped with densities, such as \( \text{flip}_\rho : \mathbb{R} \rightarrow \mathbb{R} \) (which would be \( \lambda b.0.5 \) in the case of a fair coin).
- For variables \( x : \tau \), we appeal to the hypothesis that the environments \( (\gamma, \gamma_p) \) satisfy the relation.
- For \( \text{sample}(t_1, t_2) \), we first apply the inductive hypothesis to establish that \( \left[ \rho(t_1) \right] \left( \gamma_p \right) \) is a correct density function for the primitive distribution \( \left[ t_1 \right](\gamma) \). Then we can see that all three correctness criteria for translations of \( M \tau \) terms are satisfied:
  1. \( \pi_1(\left[ \text{sample}(t_1, t_2) \right](\gamma)) \) is supported on one possible trace shape, the shape \( s = \{ \left[ t_2 \right](\gamma, \sigma) \} \), where \( \sigma \) is the type over which the primitive distribution \( \left[ t_1 \right](\gamma) \) is defined. Therefore, the density of a trace \( (s', v) \) is 0 if \( s' \neq s \), and otherwise,
it is the density of $v$ under $\langle t_1 \rangle \langle \gamma \rangle$, with respect to $\mathbb{B}_\sigma$. Our translation outputs a density that is zero if $\langle t_2 \rangle \langle \gamma \rangle$ does not appear in the input trace, and otherwise outputs the correct density of the value stored there with respect to $\mathbb{B}_\sigma$, using $\rho \{ t_1 \}$. If the trace contains extra entries, then the third return value from our translation will be a non-empty trace, and so we will still satisfy the requirement that $\lambda u. \text{isempty}(\pi_3(\rho(u))) \cdot \pi_1(\rho(u))$ is a correct trace density (it will be zero when the third return value is non-empty).

2. As required, the second output of our translation applies the return-value function from the semantics of sample to produce the value returned by sample on a particular trace (the result of $\text{pop}_\sigma$).

3. By the definition of $\text{pop}_\sigma$, we have that when the given trace does contain the name $\langle t_2 \rangle \langle \gamma \rangle$, the third return value of our translation will be the remainder of the trace.

- For factor, the third return value is the input trace, so the only trace to which we assign any mass is the empty trace (which accords with the semantics of factor). On that trace, our density is equal to $e^{-1}(\gamma)$, which matches the semantics.
- For do $(x \leftarrow t; m)$, we appeal to the inductive hypothesis for the correctness of $\rho \{ t \}$ and $\rho \{ \text{do} \{ m \} \}$, and to the product rule for densities of joint distributions.
- The argument for $\lambda x.t$ and $t_1 t_2$ (forming functions and applying them) is standard, and follows from the way we’ve defined $\mathcal{R}_{\sigma_1 \rightarrow \sigma_2}$. See, e.g., Huot et al. [6].

Having proven the fundamental lemma, the full correctness result is straight-forward.

**Proposition C.3.** For $\rho : M \tau$, $\langle \text{density}(\rho) \rangle$ is a density of $\pi_1(\rho)$ with respect to $\mathbb{B}_{\text{trace}}$.

**Proof.** By the fundamental lemma, $\langle \rho \rangle \in \mathcal{R}_{\sigma_1}$, which implies that $\lambda u. \text{isempty}(\pi_3(\rho(u))) \cdot \pi_1(\rho(u))$ is a correct trace density of $\pi_1(\rho)$. This function is precisely what $\langle \text{density} \rangle \rho$ computes. □

**Weighted sampler.** A similar argument applies to the weighted sampler program transformation, where our logical relations are now defined over $\tau \times \omega \{ \tau \}$. These relations are:

- For ground types $\sigma$, $\mathcal{R}_\sigma = \{(x, x) \mid x \in \sigma\}$.
- For distributions $D \sigma$, $\mathcal{R}_\sigma = \{(x, x) \mid x \in D \sigma\}$, i.e., the transformation leaves primitive distributions unchanged.
- For products and functions, the same inductive definitions as in the logical relations for $\rho$.
- For probabilistic program types $M \tau$, $\mathcal{R}_{\mathcal{M} \tau} = \{(v, f), (v, g) \mid v \text{ a probability measure } \land (f, \pi_3 \circ g) \in \mathcal{R}_\tau \land \hat{\delta}_u((\pi_1, \pi_2)), \mathcal{V}(v, g)(d(u, w)) = \mu\}$. This says that the translation of a probabilistic program is another probabilistic program that is normalized (denotes a probability measure, not an unnormalized measure), has the same return-value-function over traces, and – when the returned weight is factored in – denotes the same measure over traces.

An analogous fundamental lemma can be proven for this logical relation, and then the correctness of $\text{wsamp}$ follows. Together with the proposition above, this establishes Prop. 3.1.

**D Sound Inference: Details**

Unfolding the semantics of do, let, and return, we get that $\mathcal{V}\langle \text{importance}(p, q) \rangle$ is equal to

$$\bigcup \bigcup \delta_i(\hat{\omega} \cdot e^{\langle \text{importance}(p, q) \rangle} \langle \text{importance}(p, q) \rangle \langle \text{wsamp}(q) \rangle) \langle d(\hat{i}, \omega) \rangle.$$  

Using the correctness proof for densities, this can be rewritten to

$$\bigcup \bigcup \delta_i(\hat{\omega} \cdot e^{\langle \text{importance}(p, q) \rangle} \langle \text{wsamp}(q) \rangle) \langle d(\hat{i}, \omega) \rangle.$$  

The pushforward of this measure by $\pi_1$ simply returns the sampled $\hat{i}$ unchanged, and so is clearly equal to the pushforward of $\mathcal{V}\langle \text{wsamp}(q) \rangle$ by $\pi_1$. Furthermore, letting $\nu = \mathcal{V}\langle \text{importance}(p, q) \rangle$, we have

$$\bigcup \bigcup \pi_2(x) \circ \delta_{\pi_1(x)} \nu(dx) = \bigcup \bigcup \delta_i(\hat{\omega} \cdot \frac{d(\pi_1 \circ \rho)}{d(\pi_1 \circ q)}(\hat{i}) \circ \delta_i \langle \text{wsamp}(q) \rangle) (d(\hat{i}, \omega))$$

$$= \bigcup \bigcup \frac{d(\pi_1 \circ \rho)}{d(\pi_1 \circ q)} (\hat{i}) \circ \delta_i (\pi_1 \circ q) (d(\hat{i}))$$

$$= \bigcup \bigcup \delta_i (\pi_1 \circ q) (d(\hat{i}))$$

$$= \pi_1 \circ \rho.$$
as desired. The second line uses the correctness result for \texttt{wsamp}, and the third uses the definition of the Radon-Nikodym derivative.

\section*{E The Traced PPL Landscape}

One thesis of this abstract is that a useful lens through which to view the landscape of existing PPLs is to ask the following questions:

\begin{itemize}
  \item What core operations, like \texttt{density} and \texttt{wsamp}, do the languages implement?
  \item How does the choice of core operations affect what constructs the language can expose?
  \item How does the choice of core operations affect the inference algorithms that can be automated using those core operations?
\end{itemize}

A full study of these questions for different languages is left to future work, but we make some observations here, pointing to directions that may be interesting to explore further:

\begin{itemize}
  \item In the Gen PPL \cite{gen}, there is no \texttt{factor} statement, and all programs encode probability measures over traces. Instead of \texttt{wsamp}, Gen programs support exact (unweighted) simulation.
  \item In ProbTorch \cite{probtorch}, there is a core operation that partially constrains the execution of a probabilistic program using the trace of another program, but (unlike our \texttt{density} operation) allows the trace to be incomplete or to contain extraneous variables not sampled by the program. It is still unclear to us how exactly to state the general specification and correctness theorem for this operation, in terms of the measure over traces denoted by the program. Gen features a similar but more restricted operation, called \texttt{generate}, which, given a partial trace as input, returns a properly weighted sample for the posterior over complete traces, conditioning on the partial trace. (This is more restricted than ProbTorch, in that Gen does not permit the partial trace to contain extraneous or auxiliary variables that are \textit{not} present in the model.)
  \item The Pyro \cite{pyro} and Gen \cite{gen} languages contain control flow combinators, like \texttt{plate} in Pyro or \texttt{Map} in Gen, that are semantically equivalent to particular loops or recursions, but are translated specially by the PPL’s automated operations, to yield more efficient implementations of densities, gradient estimators, or other operations.
\end{itemize}