Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring*

Ryan Culpepper and Andrew Cobb

Northeastern University

Abstract. We present a logical relation for proving contextual equivalence in a probabilistic programming language (PPL) with continuous random variables and with a scoring operation for expressing observations and soft constraints.

Our PPL model is based on a big-step operational semantics that represents an idealized sampler with likelihood weighting. The semantics treats probabilistic non-determinism as a deterministic process guided by a source of entropy. We derive a measure on result values by aggregating (that is, integrating) the behavior of the operational semantics over the entropy space. Contextual equivalence is defined in terms of these measures, taking real events as observable behavior.

We define a logical relation and prove it sound with respect to contextual equivalence. We demonstrate the utility of the logical relation by using it to prove several useful examples of equivalences, including the equivalence of a β_v-redex and its contractum and a general form of expression re-ordering. The latter equivalence is sound for the sampling and scoring effects of probabilistic programming but not for effects like mutation or control.

1 Introduction

A universal probabilistic programming language (PPL) consists of a general-purpose language extended with two probabilistic features: the ability to make non-deterministic (probabilistic) choices and the ability to adjust the likelihood of the current execution, usually used to model conditioning. Programs that use these features in a principled way express probabilistic models, and the execution of such programs corresponds to Bayesian inference.

Universal PPLs include Church [8] and its descendants [13, 22] as well as other systems and models [18, 16, 10, 2, 20, 3]. In contrast, other PPLs [12, 17, 14, 4] limit programs to more constrained structures that can be translated to intermediate representations such as Bayes nets or factor graphs.

* This material is based upon work sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-14-C-0002. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

PPLs can also be divided into those that support continuous random choices and those that support only discrete choices. Most probabilistic programming systems designed for actual use support continuous random variables, and some implement inference algorithms specialized for continuous random variables [21, 4]. On the other hand, much of the literature on the semantics of PPLs has focused on discrete choice—particularly the literature on techniques for proving program equivalence such as logical relations [1] and bisimulation [19]. The semantics that do address continuous random variables and scoring [20, 3] do not focus on contextual equivalence.

This paper addresses the issue of contextual equivalence in a PPL with real arithmetic, continuous random variables, and an explicit scoring operation for expressing observations and soft constraints. We present a model of such a PPL with a big-step operational semantics based on an idealized sampler with likelihood weighting; the program's evaluation is guided by a supply of random numbers from an "entropy space." Based on the operational semantics we construct a measure on the possible results of the program, and we define contextual equivalence in terms of these measures. Finally, we construct a binary logical relation, prove it sound with respect to contextual equivalence, and demonstrate proofs that conversions such as β_v and expression reordering respect contextual equivalence.

Our language and semantics are similar to that of Borgström et al. [3], except our language is simply typed and our treatment of entropy involves splitting rather than concatenating variable-length sequences. Our entropy structure reflects the independence of subexpression evaluations and simplifies the decomposition of value measures into nested integrals.

Compared with semantics for traditional languages, our model of probabilistic programming is further from the world of computable programming languages so that it can be closer to the world of measures and integration, the foundations of probability theory. It is "syntactic" rather than "denotational" in the sense that the notion of "value" includes λ-expressions rather than mathematical functions, but on the other hand these syntactic values can contain arbitrary real numbers in their bodies, and our semantics defines and manipulates measures over spaces of such values. We do not address computability in this paper, but we hope our efforts can be reconciled with previous work on incorporating the real numbers into programming languages [7, 6, 9].

We have formalized the language and logical relation in Coq, based on a high-level axiomatization of measures, integration, and entropy. We have formally proven the soundness of the logical relation as well as some of its applications, including β_v and restricted forms of expression reordering. The formalization can be found at

`https://github.com/cobbal/ppl-ctx-equiv-coq/tree/esop-2017`

The rest of this paper is organized as follows: Section 2 introduces probabilistic programming with some example models expressed in our core PPL. Section 3 reviews some relevant definitions and facts from measure theory. Section 4 presents our PPL model, including its syntax, operational semantics, measure

semantics, and notion of contextual equivalence. In Section 5 we develop a logical relation and show that it is sound with respect to contextual equivalence. Section 6 proves several useful equivalences using using the machinery provided by the logical relation.

2 Probabilistic Programming

In a probabilistic program, random variables are created implicitly as the result of stochastic effects, and dependence between random variables is determined by the flow of values from one random variable to another. Random variables need not correspond to program variables. For example, the following two programs both represent the sum of two random variables distributed uniformly on the unit interval:

- **let** $x =$ **sample**, $y =$ **sample in** $x + y$
- **sample** $+$ **sample**

We write **sample** for the effectful expression that creates a new independent random variable distributed uniformly on the unit interval $[0, 1]$.

Values distributed according to other real-valued distributions can be obtained from a standard uniform by applying the inverse of the distribution's *cumulative distribution function* (CDF). For example, normalinvcdf(**sample**) produces a value from the standard normal distribution, with mean 0 and standard deviation 1. The familiar parameterized normal can then be defined by scaling and shifting as follows:

normal $m\ s \triangleq m + s *$ normalinvcdf(**sample**)

The parameters of a normal random variable can of course depend on other random variables. For example,

$m =$ **normal** 0 *wide*
$f\_ =$ **normal** m *narrow*

defines f as a function that returns random points—a fresh one each time it is called. The points are concentrated narrowly around some common point, randomly chosen once and shared. (We write $f\_$ to emphasize that f ignores its argument.)

The other feature offered by PPLs is some way of expressing *conditioning* on observed evidence. We introduce conditioning via a hypothetical **observe** form. Consider m from the program above. Our *prior belief* about m is that it is somewhere in a wide vicinity of 0. Suppose we amend the program by adding the following observations, however:

observe 9.3 **from** $f\_$
observe 8.9 **from** $f\_$
observe 9.1 **from** $f\_$

Given those observations, you might suspect that *m* is in a fairly narrow region around 9. Bayes' Law quantifies that belief as the *posterior distribution* on *m*, defined in terms of the prior and the observed evidence.

$$p(m|data) \propto p(m) \cdot p(data|m)$$

That is the essence of Bayesian inference: calculating updated distributions on "causes" given observed "effects" and a probabilistic model that relates them.

Some PPLs [13, 5, 15] provide an **observe**-like form to handle conditioning; they vary in what kinds of expressions can occur in the right-hand side of the observation. Other PPLs provide a more primitive facility, called **factor** or **score**, which takes a real number and uses it to scale the likelihood of the current values of all random variables. To represent an observation, one simply calls **factor** with $p(data|x)$; of course, if one is observing the result of a computation, one has to compute the correct probability density. For example, the first observation above would be translated as

factor (normalpdf((9.3 − *m*) ÷ *narrow*) ÷ *narrow*)

The normalpdf operation computes the *probability density function* density of a standard normal, so to calculate the density for a scaled and shifted normal, we must invert the translation by subtracting the mean and dividing by the scale (*narrow*). Then, since probability densities are *derivatives*, to get the correct density of **normal** *m narrow* we must divide by the (absolute value of) the derivative of the translation function from the standard normal—that accounts for the second division by *narrow*.

3 Measures and Integration

This section reviews some basic definitions, theorems, and notations from measure theory. We assume that the reader is familiar with the basic notions of measure theory, including measurable spaces, σ-algebras, measures, and Lebesgue integration—that is, the notion of integrating a function with respect to a measure, not necessarily the Lebesgue measure on $\mathbb{R}$.

We write $\mathbb{R}^{\geq 0}$ for the non-negative reals—that is, $[0, \infty)$—and $\mathbb{R}^+$ for the non-negative reals extended with infinity—that is, $[0, \infty]$.

A measure $\mu : \Sigma_X \to \mathbb{R}^+$ on the measurable space (X, Σ_X) is *finite* if $\mu(X)$ is finite. It is σ-finite if X is the union of countably many X_i and $\mu(X_i)$ is finite for each X_i.

We write $\int_A f(x)\ \mu(dx)$ for the integral of the measurable function $f : X \to \mathbb{R}$ on the region $A \subseteq X$ with respect to the measure $\mu : \Sigma_X \to \mathbb{R}^+$. We occasionally abbreviate this to $\int_A f\ d\mu$ if omitting the variable of integration is convenient. We omit the region of integration A when it is the whole space X.

We rely on the following lemmas concerning the equality of integrals. Tonelli's theorem allows changing the order of integration of non-negative functions. Since all of our integrands are non-negative, it suits our needs better than Fubini's theorem. In particular, Tonelli's theorem holds even when the functions can attain infinite values as well as when the integrals are infinite.

Lemma 1 (Tonelli). *If (X, Σ_X) and (Y, Σ_Y) are measurable spaces and μ_X and μ_Y are σ-finite measures on X and Y, respectively, and $f : X \times Y \to \mathbb{R}^+$ is measurable, then*

$$\int_X \left(\int_Y f(x,y)\ \mu_Y(dy) \right)\ \mu_X(dx) = \int_Y \left(\int_X f(x,y)\ \mu_X(dx) \right)\ \mu_Y(dy)$$

The other main lemma we rely on equates two integrals when the functions and measures may not be the same but are nonetheless related. In particular, there must be a relation such that the measures agree on related sets and the functions have related pre-images—that is, the relation specifies a "coarser" structure on which the measures and functions agree. This lemma is essential for showing the observable equivalence of measures derived from syntactically different expressions.

Lemma 2 (Coarsening). *Let (X, Σ_X) be a measurable space, $M \subseteq (\Sigma_X \times \Sigma_X)$ be a binary relation on measurable sets, $\mu_1, \mu_2 : \Sigma_X \to \mathbb{R}^+$ be measures on X, and $f_1, f_2 : X \to \mathbb{R}^+$ be measurable functions on X. If the measures agree on M-related sets and if the functions have M-related pre-images—that is,*

- $\forall (A_1, A_2) \in M,\ \mu_1(A_1) = \mu_2(A_2)$
- $\forall B \in \Sigma_{\mathbb{R}},\ (f_1^{-1}(B), f_2^{-1}(B)) \in M$

then their corresponding integrals are equal:

$$\int f_1\ d\mu_1 = \int f_2\ d\mu_2$$

Proof. Together the two conditions imply that

$$\forall B \in \Sigma_{\mathbb{R}},\ \mu_1(f_1^{-1}(B)) = \mu_2(f_2^{-1}(B))$$

We apply this equality after rewriting the integrals using the "layer cake" perspective to make the pre-images explicit [11].

$$\begin{aligned} \int f_1\ d\mu_1 &= \int_0^\infty \mu_1(f_1^{-1}(\,[t,\infty]\,))\ dt \\ &= \int_0^\infty \mu_2(f_2^{-1}(\,[t,\infty]\,))\ dt \\ &= \int f_2\ d\mu_2 \end{aligned}$$

□

Even though in the proof we immediately dispense with the intermediate relation M, we find it useful in the applications of the lemma to identify the relationship that justifies the agreement of the functions and measures.

A useful special case of Lemma 2 is when the measures are the same and the relation is equality of measure.

Lemma 3. *Let $f, g : X \to \mathbb{R}^+$ and let μ_X be a measure on X. If $\forall B \in \Sigma_{\mathbb{R}},\ \mu_X(f^{-1}(B)) = \mu_X(g^{-1}(B))$, then $\int f\ d\mu = \int g\ d\mu$.*

Proof. Special case of Lemma 2. □

$$
\begin{aligned}
e &::= r \mid x \mid \lambda x : \tau.\, e \mid e\ e \mid op^n(e_1, \cdots, e_n) \mid \textbf{sample} \mid \textbf{factor}\ e \\
v &::= r \mid x \mid \lambda x : \tau.\, e \\
r &\in \mathbb{R} \\
op^1 &::= \log \mid \exp \mid \text{normalpdf} \mid \text{normalinvcdf} \mid \cdots \\
op^2 &::= + \mid - \mid * \mid \div \mid \cdots \\
\\
\tau &::= \mathbb{R} \mid \tau \to \tau
\end{aligned}
$$

Fig. 1. Syntax

$$
\frac{}{\Gamma \vdash r : \mathbb{R}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma[x \mapsto \tau'] \vdash e : \tau}{\Gamma \vdash (\lambda x : \tau'.\, e) : \tau' \to \tau} \qquad \frac{\Gamma \vdash e : \tau' \to \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash e\ e' : \tau}
$$

$$
\frac{op^n : (\tau_1, \cdots, \tau_n) \to \tau \qquad \Gamma \vdash e_i : \tau_i}{\Gamma \vdash op^n(e_1, \cdots, e_n) : \tau} \qquad \frac{}{\Gamma \vdash \textbf{sample} : \mathbb{R}} \qquad \frac{\Gamma \vdash e : \mathbb{R}}{\Gamma \vdash \textbf{factor}\ e : \mathbb{R}}
$$

$$
\begin{aligned}
\log, \exp, \text{normalpdf}, \text{normalinvcdf} &: (\mathbb{R}) \to \mathbb{R} \\
+, -, *, \div &: (\mathbb{R}, \mathbb{R}) \to \mathbb{R}
\end{aligned}
$$

Fig. 2. Type Rules

4 Syntax and Semantics

This section presents the syntax and semantics of a language for probabilistic programming, based on a functional core extended with real arithmetic and stochastic effects.

We define the semantics of this language in two stages. We first define a big-step evaluation relation based on an idealized sampler with likelihood weighting; the evaluation rules consult an "entropy source" which determines the random behavior of a program. From this sampling semantics we then construct an aggregate view of the program as a *measure* on syntactic values. Contextual equivalence is defined in terms of these value measures.

4.1 Syntax

Figure 1 presents the syntax of our core probabilistic programming language. The language consists of a simply-typed lambda calculus extended with real arithmetic and two effects: a **sample** form for random behavior and a **factor** form for expressing observations and soft constraints. Figure 2 gives the type rules for the language.

The **sample** form returns a real number uniformly distributed between 0 and 1. We assume inverse-CDF and PDF operations—used to produce samples and score observations, respectively—for every primitive real-valued distribution

of interest. Recall from Section 2 that sampling a normal random variable is accomplished as follows:

normal $m\ s \triangleq m + s * \mathsf{normalinvcdf}(\textbf{sample})$

and observing a normal random variable is expressed thus:

factor $(\mathsf{normalpdf}((\mathit{data} - m) \div s) \div s)$

A note on notation: We drop the type annotations on bound variables when they are obvious from the context, and we use syntactic sugar for local bindings and sequencing; for example, we write

let $x =$ **sample in factor** $1 \div x\ ;\ x$

instead of

$(\lambda x : \mathbb{R}.\ ((\lambda \_ : \mathbb{R}.\ x)\ (\textbf{factor}\ 1 \div X)))\ \textbf{sample}$

4.2 Evaluation Relation

If we interpret evaluation as idealized importance sampling, the evaluation relation tells us how to produce a single sample given the initial state of the random number generator. Evaluation is defined via the judgment

$$\sigma \vdash e \Downarrow v, w$$

where $\sigma \in \mathbb{S}$ is an entropy source, e is the expression to evaluate, v is the resulting value, and $w \in \mathbb{R}^{\geq 0}$ is the likelihood weight.

The σ argument acts as the source of randomness—evaluation is a deterministic function of e and σ. Rather than threading σ through evaluation like a store, rules with multiple sub-derivations split the entropy. The indexed family of functions $\pi_i : \mathbb{S} \to \mathbb{S}$ splits the entropy source into independent pieces and $\pi_U : \mathbb{S} \to [0, 1]$ extracts a real number on the unit interval. We discuss the structure of the entropy space further in Section 4.3.

The result of evaluation is a closed value: either a real number or a closed λ-expression. Let $[\![\tau]\!]$ be the set of all closed values of type τ. We consider $[\![\tau]\!]$ a measurable space with a σ-algebra $\Sigma_{[\![\tau]\!]}$. See the comments on measurability at the end of this section. Note that the σ-algebras for function types are defined on syntactic values, not for mathematical functions, so we avoid the issues concerning measurable function spaces [20].

The evaluation rules for the language's functional fragment are unsurprising. For simple expressions, the entropy is ignored and the likelihood weight is 1. For compound expressions, the entropy is split and sent to sub-expression evaluations, and the resulting weights are multiplied together. We assume a partial function δ that interprets the primitive operations.[1] For example, $\delta(+, 1, 2) = 3$ and $\delta(\div, 4, 0)$ is undefined.

[1] No relation to the Dirac measure, also often written δ.

$$\textsc{Const}\quad \frac{}{\sigma \vdash r \Downarrow r, 1} \qquad \textsc{Lam}\quad \frac{}{\sigma \vdash \lambda x.\, e \Downarrow \lambda x.\, e, 1}$$

$$\textsc{App}\quad \frac{\pi_1(\sigma) \vdash e_1 \Downarrow \lambda x.\, e_b, w_1 \qquad \pi_2(\sigma) \vdash e_2 \Downarrow v_2, w_2 \qquad \pi_3(\sigma) \vdash e_b\,[x \mapsto v_2] \Downarrow v, w_3}{\sigma \vdash e_1\ e_2 \Downarrow v, w_1 \cdot w_2 \cdot w_3}$$

$$\textsc{Primop}\quad \frac{\pi_i(\sigma) \vdash e_i \Downarrow v_i, w_i \qquad v = \delta(op^n, v_1, \cdots, v_n) \qquad w = \prod_{i=1}^{n} w_i}{\sigma \vdash op^n(e_1, \cdots, e_n) \Downarrow v, w}$$

$$\textsc{Sample}\quad \frac{r = \pi_U(\sigma)}{\sigma \vdash \textbf{sample} \Downarrow r, 1} \qquad \textsc{Factor}\quad \frac{\sigma \vdash e \Downarrow r, w \qquad r > 0}{\sigma \vdash \textbf{factor}\ e \Downarrow r, r \cdot w}$$

Fig. 3. Evaluation Rules

The rule for **sample** extracts a real number uniformly distributed on the unit interval $[0, 1]$. The **factor** form evaluates its subexpression and interprets it as a likelihood weight to be factored into the weight of the current execution—but only if it is positive.

There are two ways evaluation can fail:

- the argument to **factor** is zero or negative
- the δ function is undefined for an operation with a particular set of arguments, such as for $1 \div 0$ or $\mathsf{log}(-5)$

The semantics does not distinguish these situations; in both cases, no evaluation derivation tree exists for that particular combination of σ and e.

4.3 Entropy Space

The evaluation relation of Section 4.2 represents evaluation of a probabilistic program as a deterministic partial function of points in an entropy space $\mathbb{S}$. To capture the meaning of a program, we must consider the aggregate behavior over the entire entropy space. That requires integration, which in turn requires a measurable space ($\Sigma_{\mathbb{S}}$) and a base measure on entropy ($\mu_{\mathbb{S}} : \Sigma_{\mathbb{S}} \to \mathbb{R}^+$).

The entropy space must support our formulation of evaluation, which roughly corresponds to the following transformation:

$$\begin{array}{l} \\ x_1 \sim D_1 \\ \quad\vdots \\ x_n \sim D_n \end{array} \quad \Rightarrow \quad \begin{array}{l} \sigma \sim \mu_{\mathbb{S}} \\ x_1 = \text{invcdf}_{D_1}(\pi_1(\sigma)) \\ \quad\vdots \\ x_n = \text{invcdf}_{D_n}(\pi_n(\sigma)) \end{array}$$

The entropy space and its associated functions are ours to choose, provided they satisfy the following criteria:

- It must be a probability space. That is, $\mu_{\mathbb{S}}(\mathbb{S}) = 1$.
- It must be able to represent common real-valued random variables. It is sufficient to support a *standard uniform*—that is, a random variable uniformly distributed on the interval $[0, 1]$. Other distributions can be represented via the inverse-CDF transformation.
- It must support multiple *independent* random variables. That is, the entropy space must be isomorphic—in a measure-preserving way—to products of itself: $\mathbb{S} \cong \mathbb{S}^2 \cong \mathbb{S}^n \quad (n \geq 1)$.

The following specification formalizes these criteria.

Definition 4 (Entropy). *$(\mathbb{S}, \Sigma_{\mathbb{S}})$ is a measurable space with measure $\mu_{\mathbb{S}} : \Sigma_{\mathbb{S}} \to \mathbb{R}^+$ and functions*

$$\begin{aligned} \pi_U &: \mathbb{S} \to [0, 1] \\ \pi_{\mathrm{L}}, \pi_{\mathrm{R}} &: \mathbb{S} \to \mathbb{S} \end{aligned}$$

such that the following integral equations hold:

- *$\mu_{\mathbb{S}}(\mathbb{S}) = 1$, and thus for all $k \in \mathbb{R}^+$, $\int k \ \mu_{\mathbb{S}}(d\sigma) = k$,*
- *for all measurable functions $f : [0, 1] \to \mathbb{R}^+$,*

$$\int f(\pi_U(\sigma)) \ \mu_{\mathbb{S}}(d\sigma) = \int_{[0,1]} f(x) \ \lambda(dx)$$

  where λ is the Lebesgue measure, and
- *for all measurable functions $g : \mathbb{S} \times \mathbb{S} \to \mathbb{R}^+$,*

$$\int g(\pi_{\mathrm{L}}(\sigma), \pi_{\mathrm{R}}(\sigma)) \ \mu_{\mathbb{S}}(d\sigma) = \iint g(\sigma_1, \sigma_2) \ \mu_{\mathbb{S}}(d\sigma_1) \ \mu_{\mathbb{S}}(d\sigma_2)$$

That is, π_U interprets the entropy as a standard uniform random variable, and π_{L} and π_{R} split the entropy into a pair of independent parts and return the first or second part, respectively. We generalize from two-way splits to indexed splits via the following family of functions:

Definition 5 (π_i) *Let $\pi_i : \mathbb{S} \to \mathbb{S}$ be the family of functions defined thus:*

$$\begin{aligned} \pi_1(\sigma) &= \pi_{\mathrm{L}}(\sigma) \\ \pi_{n+1}(\sigma) &= \pi_n(\pi_{\mathrm{R}}(\sigma)) \end{aligned}$$

Definition 5 is "wasteful"—for any $n \in \mathbb{N}$, using only $\pi_1(\sigma)$ through $\pi_n(\sigma)$ discards part of the entropy—but that does not cause problems, because the wasted entropy is independent and thus integrates away. Thus the a generalized entropy-splitting identity holds for measurable $f : \mathbb{S}^n \to \mathbb{R}^+$:

$$\int f(\pi_1(\sigma), \cdots, \pi_n(\sigma)) \ \mu_{\mathbb{S}}(d\sigma) = \int \cdots \int f(\sigma_1, \cdots, \sigma_n) \ \mu_{\mathbb{S}}(d\sigma_1) \cdots \mu_{\mathbb{S}}(d\sigma_n)$$

Our preferred concrete representation of $\mathbb{S}$ is the countable product of unit intervals, $[0,1]^\omega$, sometimes called the Hilbert cube. The π_{L}, π_{R}, and π_U functions are defined as follows:

$$\pi_{\mathrm{L}}(\langle u_0, u_1, u_2, u_3, \dots \rangle) = \langle u_0, u_2, \dots \rangle$$
$$\pi_{\mathrm{R}}(\langle u_0, u_1, u_2, u_3, \dots \rangle) = \langle u_1, u_3, \dots \rangle$$
$$\pi_U(\langle u_0, u_1, u_2, u_3, \dots \rangle) = u_0$$

The σ-algebra $\Sigma_{\mathbb{S}}$ is the Borel algebra of the product topology (cf Tychonoff's Theorem). The basis of the product topology is the set of products of intervals, only finitely many of which are not the whole unit interval $U = [0,1]$:

$$\left\{ \left(\prod_{i=1}^{k} (a_i, b_i) \right) \times U^\omega \quad | \quad 0 \le a_i \le b_i \le 1, k \in \mathbb{N} \right\}$$

We define the measure $\mu_{\mathbb{S}}$ on a basis element as follows:

$$\mu_{\mathbb{S}} \left(\left(\prod_{i=1}^{k} (a_i, b_i) \right) \times U^\omega \right) = \prod_{i=1}^{k} (b_i - a_i)$$

That uniquely determines the measure $\mu_{\mathbb{S}} : \Sigma_{\mathbb{S}} \to \mathbb{R}^+$ by the Carathéodory extension theorem.

Another representation of entropy is the Borel space on $[0,1]$ with (restricted) Lebesgue measure and bit-splitting π_{L} and π_{R}. In fact, both of these representations are examples of *standard atomless probability spaces*, and all such spaces are isomorphic (modulo null sets). In the rest of the paper, we rely only on the guarantees of Definition 4, not on the precise representation of $\mathbb{S}$.

4.4 Measure Semantics

We represent the aggregate behavior of a closed expression as a measure, obtained by integrating the behavior of the evaluation relation over the entropy space. If $\vdash e : \tau$ then $\mu_e : \Sigma_{[\![\tau]\!]} \to \mathbb{R}^+$ is the *value measure of* e, defined as follows:

Definition 6 (Value Measure)

$$\mu_e(V) = \int \mathrm{evalin}(e, V, \sigma)\ \mu_{\mathbb{S}}(d\sigma)$$
$$\mathrm{evalin}(e, V, \sigma) = \mathbb{I}_V(\mathrm{ev}(e, \sigma)) \cdot \mathrm{ew}(e, \sigma)$$
$$\mathrm{ev}(e, \sigma) = \begin{cases} v & \textit{if } \sigma \vdash e \Downarrow v, w \\ \bot & \textit{otherwise} \end{cases}$$
$$\mathrm{ew}(e, \sigma) = \begin{cases} w & \textit{if } \sigma \vdash e \Downarrow v, w \\ 0 & \textit{otherwise} \end{cases}$$

The evaluation relation $\sigma \vdash e \Downarrow v, w$ is a partial function of (σ, e)—nondeterministic behavior is represented as deterministic dependence on the entropy σ. From this partial function we define a total evaluation function $\mathrm{ev}(e, \sigma)$ and a total weighting function $\mathrm{ew}(e, \sigma)$. The $\mathrm{evalin}(e, V, \sigma)$ function takes a measurable outcome set of interest and checks whether the result of evaluation falls within that set. If so, it produces the weight of the evaluation; otherwise, it produces 0. We write $\mathbb{I}_X$ for the indicator function for X, which returns 1 if its argument is in X and 0 otherwise.

Integrating $\mathrm{evalin}(e, V, \sigma)$ over the entire entropy space yields the value measure μ_e. Strictly speaking, the definition above defines μ_e as a measure on $[\![\tau]\!]_\perp$, but since $\mathrm{ev}(e, \sigma) = \perp$ only when $\mathrm{ew}(e, \sigma) = 0$, μ_e never assigns any weight to $\perp$ and thus we can consider it a measure on $[\![\tau]\!]$.

The following theorem shows that the value measure is an adequate representation of the behavior of a program.

Theorem 7. *Let $f : [\![\tau]\!] \to \mathbb{R}^+$ be measurable, and let $\vdash e : \tau$. Then*

$$\int f(v)\ \mu_e(dv) = \int f(\mathrm{ev}(e, \sigma)) \cdot \mathrm{ew}(e, \sigma)\ \mu_{\mathbb{S}}(d\sigma)$$

Proof. First consider the case where f is an indicator function $\mathbb{I}_X$:

$$\begin{aligned}
\int f(v)\ \mu_e(dv) &= \int \mathbb{I}_X(v)\ \mu_e(dv) && (f = \mathbb{I}_X) \\
&= \mu_e(X) && \text{(integral of indicator function)} \\
&= \int \mathbb{I}_X(\mathrm{ev}(e, \sigma)) \cdot \mathrm{ew}(e, \sigma)\ \mu_{\mathbb{S}}(d\sigma) && \text{(Definition 6)} \\
&= \int f(\mathrm{ev}(e, \sigma)) \cdot \mathrm{ew}(e, \sigma)\ \mu_{\mathbb{S}}(d\sigma) && (f = \mathbb{I}_X)
\end{aligned}$$

The equality extends to simple functions—linear combinations of characteristic functions—by the linearity of integration and to measurable functions as the suprema of sets of simple functions. □

Measurability For the integral defining $\mu_e(V)$ to be well-defined, $\mathrm{evalin}(e, V, \sigma)$ must be measurable when considered as a function of σ. Furthermore, in later proofs we will need the $\mathrm{ev}(\cdot, \cdot)$ and $\mathrm{ew}(\cdot, \cdot)$ functions to be measurable with respect to the product space on their arguments. More precisely, if we consider a type-indexed family of functions

$$\mathrm{ev}_\tau : \mathrm{Expr}[\![\tau]\!] \times \mathbb{S} \to [\![\tau]\!]$$

then we need each ev_τ to be measurable in $\Sigma_{[\![\tau]\!]}$ with respect to the product measurable space $\Sigma_{\mathrm{Expr}[\![\tau]\!]} \times \Sigma_{\mathbb{S}}$, and likewise for ew_τ. Note that the space of values $[\![\tau]\!]$ is a subset of the expressions $\mathrm{Expr}[\![\tau]\!]$, so we can take $\Sigma_{[\![\tau]\!]}$ to be $\Sigma_{\mathrm{Expr}[\![\tau]\!]}$ restricted to the values. But we must still define $\Sigma_{\mathrm{Expr}[\![\tau]\!]}$ and show the functions are measurable.

We do not present a direct proof of measurability in this paper. Instead, we rely again on Borgström et al. [3]: we treat their language, for which they have proven measurability, as a meta-language. Interpreters for the ev and ew functions of our language can be written as terms in this meta-language, and thus their measurability result can be carried over to our language. We take $\mathbb{S} = [0, 1]$ and extend the meta-language with the measurable functions π_{L}, π_{R}, and π_U. The definition of $\Sigma_{\mathrm{Expr}[\![\tau]\!]}$ is induced by the encoding function that represents our object terms as values in their meta-language and the structure of their measurable space of expressions.

4.5 Digression: Interpretation of Probabilistic Programs

In general, the goal of a probabilistic programming language is to interpret programs as probability distributions.

If a program's value measure is finite and non-zero, then it can be normalized to yield a probability distribution. The following examples explore different classes of such programs:

- Continuous measures: **sample**, normalinvcdf(**sample**), etc.
- Discrete measures: **if sample** $<$ 0.2 **then** 1 **else** 0
- Sub-probability measures:

  let x = normalinvcdf(**sample**) **in if** $x < 0$ **then factor** 0 **else** x

- Mixtures of discrete and continuous: for example,

  let x = **sample in if** $x < 0.5$ **then** 0 **else** x

  has a point mass at 0 and is continuous on (0.5, 1).

Our language, however, includes programs that have no interpretation as distributions:

- Zero measure: **factor** 0
- Infinite (but σ-finite) measures. For example,

  let x = **sample in factor** $(1 \div x)$; x

  has infinite measure because $\int_0^1 \frac{1}{x}\, dx$ is infinite. But the measure is σ-finite because each interval $[\frac{1}{n}, 1]$ has finite measure and the union of all such intervals covers $(0, 1]$, the support of the measure.
- Non-σ-finite measures. For example,

  let x = **sample in factor** $(1 \div x)$; 0

  has $\mu(0) = \infty$. (We conjecture that all value measures definable in this language are either σ-finite or have a point with infinite weight.)

Zero measures indicate unsatisfiable constraints; more precisely, the set of successful evaluations may not be empty, merely measure zero.

$$C ::= [\,] \mid \lambda x : \tau.\, C \mid C\ e \mid e\ C \mid op(e, \cdots, C, e, \cdots) \mid \textbf{factor}\ C$$

$$\frac{}{[\,] : \emptyset} \qquad \frac{C : \Gamma}{(\lambda x : \tau.\, C) : \Gamma[x \mapsto \tau]}$$

$$\frac{C : \Gamma}{C\ e : \Gamma} \qquad \frac{C : \Gamma}{e\ C : \Gamma} \qquad \frac{C : \Gamma}{op(e, \cdots, C, e, \cdots) : \Gamma} \qquad \frac{C : \Gamma}{\textbf{factor}\ C : \Gamma}$$

Fig. 4. Contexts

Infinite value measures arise only from the use of **factor**; the value measure of a program that does not contain **factor** is always a sub-probability measure. It may not be a probability measure—recall that $1 \div 0$ and **factor** 0 both cause execution to fail. We could eliminate infinite-measure programs by sacrificing expressiveness. For example, if the valid arguments to **factor** were restricted to the range $(0, 1]$, as in Börgstrom et al. [3], only sub-probability measures would be expressible. But there are good reasons to allow **factor** with numbers greater than 1, such as representing the observation of a normal random variable with a small variance—perhaps a variance computed from another random variable. There is no simple syntactic rule that excludes the infinite-measure programs above without also excluding some useful applications of **factor**.

Note that the theorems in this paper apply to *all* programs, regardless of whether they can be interpreted as probability distributions. In particular, we apply Lemma 1 (Tonelli) only to integrals over $\mu_{\mathbb{S}}$, which is finite.

4.6 Contextual Equivalence

Two expressions are contextually equivalent ($=_{\text{ctx}}$) if for all closing program contexts C their observable aggregate behavior is the same. We take *programs* to be real-valued closed expressions; their *observable behavior* consists of their value measures ($\Sigma_{\mathbb{R}} \to \mathbb{R}^+$).

Figure 4 defines contexts and their relationship with type environments. The relation $C : \Gamma$ means that C provides bindings satisfying Γ to the expression placed in its hole.

Definition 8 (Contextual equivalence) *If $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$, then e_1 and e_2 are contextually equivalent ($e_1 =^{\Gamma}_{\text{ctx}} e_2$) if and only if for all contexts C such that $C : \Gamma$ and $\vdash C[e_1] : \mathbb{R}$ and $\vdash C[e_2] : \mathbb{R}$ and for all measurable sets $A \in \Sigma_{\mathbb{R}}$,*

$$\mu_{C[e_1]}(A) = \mu_{C[e_2]}(A)$$

Instances of contextual equivalence are difficult to prove directly because of the quantification over all syntactic contexts.

Definition 9 ($\approx$)

$$\Gamma \vdash e_1 \approx e_2 : \tau \iff \forall(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!],\ (e_1 \cdot \gamma_1, e_2 \cdot \gamma_2) \in \mathcal{E}[\![\tau]\!]$$

$$(r_1, r_2) \in \mathcal{V}[\![\mathbb{R}]\!] \iff r_1 = r_2$$
$$(\lambda x.\, e_1, \lambda x.\, e_2) \in \mathcal{V}[\![\tau_1 \to \tau_2]\!] \iff \forall(v_1, v_2) \in \mathcal{V}[\![\tau_1]\!],\ (e_1\,[x \mapsto v_1], e_2\,[x \mapsto v_2]) \in \mathcal{E}[\![\tau_2]\!]$$

$$(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!] \iff \gamma_1 \models \Gamma \ \wedge\ \gamma_2 \models \Gamma \ \wedge\ \forall x \in \mathrm{dom}(\Gamma),\ (\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\![\Gamma(x)]\!]$$

$$(e_1, e_2) \in \mathcal{E}[\![\tau]\!] \iff \forall(A_1, A_2) \in \mathcal{A}[\![\tau]\!],\ \mu_{e_1}(A_1) = \mu_{e_2}(A_2)$$

$$(A_1, A_2) \in \mathcal{A}[\![\tau]\!] \iff A_1, A_2 \in \Sigma_{[\![\tau]\!]} \ \wedge\ \forall(v_1, v_2) \in \mathcal{V}[\![\tau]\!],\ (v_1 \in A_1 \iff v_2 \in A_2)$$

Fig. 5. Logical relation and auxiliary relations

5 A Logical Relation for Contextual Equivalence

In this section we develop a logical relation for proving expressions contextually equivalent. Membership in the logical relation implies contextual equivalence but is easier to prove directly. We prove soundness via compatibility lemmas, one for each kind of compound expression. The fundamental property (a form of reflexivity) enables simplifications to the logical relation that we take advantage of in Section 6 when applying the relation to particular equivalences.

Figure 5 defines the relation

$$\Gamma \vdash e_1 \approx e_2 : \tau$$

and its auxiliary relations (Definition 9). In a deterministic language, we would construct the relation so that two expressions are related if they produce related values when evaluated with related substitutions. In our probabilistic language, two expressions are related if they have related *value measures* when evaluated with related substitutions. (The notation $e \cdot \gamma$ indicates the substitution γ applied to the expression e.)

The $\approx$ relation depends on the following auxiliary relations:

- $\mathcal{V}[\![\tau]\!]$ relates closed values of type τ. Real values are related if they are identical, and functions are related if they take related inputs to related evaluation configurations ($\mathcal{E}[\![\tau]\!]$).
- $\mathcal{G}[\![\Gamma]\!]$ relates substitutions. Variables are mapped to related values.
- $\mathcal{E}[\![\tau]\!]$ relates closed expressions. Expressions are related if their value measures agree on measurable sets related by $\mathcal{A}[\![\tau]\!]$.
- $\mathcal{A}[\![\tau]\!]$ relates measurable sets of values.

When comparing value measures, we must not demand complete equality of the measures; instead, we only require that they agree on $\mathcal{V}[\![\tau]\!]$-closed measurable

value sets. To see why, consider the expressions $\lambda x.\, x + 2$ and $\lambda x.\, x + 1 + 1$. As values, they are related by $\mathcal{V}[\![\mathbb{R} \to \mathbb{R}]\!]$. As expressions, we want them to be related by $\mathcal{E}[\![\mathbb{R} \to \mathbb{R}]\!]$, but their value measures are not identical; they are Dirac measures on different—but related—syntactic values. In particular:

- $\mu_{\lambda x.\, x+2}(\{\lambda x.\, x + 2\}) = 1$, but
- $\mu_{\lambda x.\, x+1+1}(\{\lambda x.\, x + 2\}) = 0$

The solution is to compare measures only on related measurable sets. For every value in the set given to the first measure, we must include every related value in the set given to the second measure (and vice versa). This relaxation on measure equivalence preserves the spirit of "related computations produce related results."

Lemma 10 (Symmetry and transitivity). *$\mathcal{V}[\![\tau]\!]$, $\mathcal{G}[\![\Gamma]\!]$, $\mathcal{E}[\![\tau]\!]$, $\mathcal{A}[\![\tau]\!]$, and $\approx$ are symmetric and transitive.*

Proof. The symmetry and transitivity of $\approx$ and $\mathcal{G}$ follow from that of $\mathcal{E}$ and $\mathcal{V}$.

We prove symmetry and transitivity of $\mathcal{V}[\![\tau]\!]$, $\mathcal{E}[\![\tau]\!]$, and $\mathcal{A}[\![\tau]\!]$ simultaneously by induction on τ. For a given τ, the properties of $\mathcal{E}[\![\tau]\!]$ and $\mathcal{A}[\![\tau]\!]$ follow from $\mathcal{V}[\![\tau]\!]$. The $\mathbb{R}$ case is trivial. Transitivity for $\mathcal{V}[\![\tau' \to \tau]\!]$ is subtle; given $(v_1, v_3) \in \mathcal{V}[\![\tau']\!]$, we must find a v_2 such that $(v_1, v_2) \in \mathcal{V}[\![\tau']\!]$ and $(v_2, v_3) \in \mathcal{V}[\![\tau']\!]$ in order to use transitivity of $\mathcal{E}[\![\tau]\!]$ (induction hypothesis). But we can use symmetry and transitivity of $\mathcal{V}[\![\tau']\!]$ (also induction hypotheses) to show $(v_1, v_1) \in \mathcal{V}[\![\tau']\!]$, so v_1 is a suitable value for v_2. □

The reflexivity of $\mathcal{V}$, $\mathcal{E}$, $\mathcal{G}$, and $\approx$ is harder to prove. In fact, it is a corollary of the fundamental property of the logical relation (Theorem 15).

5.1 Compatibility Lemmas

The compatibility lemmas show that expression pairs built from related components are themselves related. Equivalently, they allow the substitution of related expressions in single-frame contexts. Given the compatibility lemmas, soundness with respect to contextual equivalence with arbitrary contexts is a short inductive hop away.

Lemma 11 (Lambda Compatibility).

$$\frac{\Gamma, x : \tau' \vdash e_1 \approx e_2 : \tau}{\Gamma \vdash (\lambda x : \tau'.\, e_1) \approx (\lambda x : \tau'.\, e_2) : \tau' \to \tau}$$

Proof. Let $(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!]$. We must prove that $\lambda x.\, e_i \cdot \gamma_i$ are in $\mathcal{E}[\![\tau' \to \tau]\!]$—that is, the corresponding value measures $\mu_{\lambda x.\, e_i \cdot \gamma_i}$ agree on all $(A_1, A_2) \in \mathcal{A}[\![\tau' \to \tau]\!]$.

The value measure $\mu_{\lambda x.\, e_i \cdot \gamma_i}$ is concentrated at $\lambda x.\, e_i \cdot \gamma_i$ with weight 1, so the measures are related if those closures are related in $\mathcal{V}[\![\tau' \to \tau]\!]$. That in turn requires that $(e_i \cdot \gamma_i)\,[x \mapsto v_i]$ be related in $\mathcal{E}[\![\tau]\!]$ for $(v_1, v_2) \in \mathcal{V}[\![\tau']\!]$. That follows from $\Gamma, x : \tau' \vdash e_1 \approx e_2 : \tau$, instantiated at $[\gamma_i, x \mapsto v_i]$. □

Lemma 12 (App Compatibility).

$$\frac{\Gamma \vdash e_1 \approx e_2 : \tau' \to \tau \qquad \Gamma \vdash e_1' \approx e_2' : \tau'}{\Gamma \vdash e_1\ e_1' \approx e_2\ e_2' : \tau}$$

Proof. By the premises, the $\mu_{e_i \cdot \gamma_i}$ measures agree on $\mathcal{A}[\![\tau' \to \tau]\!]$, and the $\mu_{e_i' \cdot \gamma_i}$ measures agree on $\mathcal{A}[\![\tau']\!]$. Our strategy is to use Lemma 2 (Coarsening) to rewrite the integrals after unpacking the the definition of the value measures and the App rule. The applyin function defined as follows

$$\text{applyin}(\lambda x.\, e, v', A, \sigma) = \text{evalin}(e\,[x \mapsto v'], A, \sigma)$$

is useful for expressing the unfolding of the App rule.

Let $(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!]$. We must prove the expressions $(e_i\ e_i') \cdot \gamma_i$ are in $\mathcal{E}[\![\tau]\!]$.

After unfolding $\mathcal{E}$ and introducing $(A_1, A_2) \in \mathcal{A}[\![\tau]\!]$, we must show the corresponding value measures agree:

$$\mu_{(e_1\ e_1') \cdot \gamma_1}(A_1) = \mu_{(e_2\ e_2') \cdot \gamma_2}(A_2)$$

We rewrite each side as follows:

$$\begin{aligned}
&\mu_{(e_i\ e_i') \cdot \gamma_i}(A_i) \\
&= \int \text{evalin}((e_i\ e_i') \cdot \gamma_i, A_i, \sigma)\ \mu_{\mathbb{S}}(d\sigma) && \text{(by Definition 6)} \\
&= \int \text{applyin}(\text{ev}(e_i \cdot \gamma_i, \pi_1(\sigma)), \text{ev}(e_i' \cdot \gamma_i, \pi_2(\sigma)), A_i, \pi_3(\sigma)) \\
&\qquad \cdot \text{ew}(e_i \cdot \gamma_i, \pi_1(\sigma)) \cdot \text{ew}(e_i' \cdot \gamma_i, \pi_2(\sigma))\ \mu_{\mathbb{S}}(d\sigma) && \text{(by App)} \\
&= \iiint \text{applyin}(\text{ev}(e_i \cdot \gamma_i, \sigma_1), \text{ev}(e_i' \cdot \gamma_i, \sigma_2), A_i, \sigma_3) \\
&\qquad \cdot \text{ew}(e_i \cdot \gamma_i, \sigma_1) \cdot \text{ew}(e_i' \cdot \gamma_i, \sigma_2)\ \mu_{\mathbb{S}}(d\sigma_3)\ \mu_{\mathbb{S}}(d\sigma_2)\ \mu_{\mathbb{S}}(d\sigma_1) \\
& && \text{(by Proposition 4)} \\
&= \iiint \text{applyin}(v, v', A_i, \sigma_3)\ \mu_{\mathbb{S}}(d\sigma_3)\ \mu_{e_i' \cdot \gamma_i}(dv')\ \mu_{e_i \cdot \gamma_i}(dv) && \text{(by Theorem 7)}
\end{aligned}$$

After rewriting both sides, we have the goal

$$\begin{aligned}
&\iiint \text{applyin}(v, v', A_1, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mu_{e_1' \cdot \gamma_1}(dv')\ \mu_{e_1 \cdot \gamma_1}(dv) \\
&= \iiint \text{applyin}(v, v', A_2, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mu_{e_2' \cdot \gamma_2}(dv')\ \mu_{e_2 \cdot \gamma_2}(dv)
\end{aligned}$$

We show this equality via Lemma 2 (Coarsening) using the binary relation $\mathcal{A}[\![\tau' \to \tau]\!]$. By the induction hypothesis we have that $\mu_{e_1 \cdot \gamma_1}, \mu_{e_2 \cdot \gamma_2}$ agree on sets in $\mathcal{A}[\![\tau' \to \tau]\!]$. That leaves one other premise to discharge: the functions

must have related pre-images. Let $B \in \Sigma_{\mathbb{R}}$. We must show the pre-images are related by $\mathcal{A}[\![\tau' \to \tau]\!]$, where each pre-image is

$$\left(v \mapsto \iint \text{applyin}(v, v', A_i, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mu_{e_i' \cdot \gamma_i}(dv')\right)^{-1}(B)$$

To show that the function pre-images are in $\mathcal{A}[\![\tau' \to \tau]\!]$, we show something stronger: for related values the function values are the same.

Let $(v_1, v_2) \in \mathcal{V}[\![\tau' \to \tau]\!]$. We will show that

$$\begin{aligned}&\iint \text{applyin}(v_1, v', A_1, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mu_{e_1' \cdot \gamma_i}(dv') \\ =&\iint \text{applyin}(v_2, v', A_2, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mu_{e_2' \cdot \gamma_i}(dv')\end{aligned}$$

We show this by again applying Lemma 2 (Coarsening), this time with the relation $\mathcal{A}[\![\tau']\!]$. Again, the induction hypothesis tells us that the measures $\mu_{e_i' \cdot \gamma_i}$ agree on sets in $\mathcal{A}[\![\tau']\!]$. We follow the same strategy for showing the function pre-images related. Let $(v_1', v_2') \in \mathcal{V}[\![\tau']\!]$. We must show

$$\int \text{applyin}(v_1, v_1', A_1, \sigma)\ \mu_{\mathbb{S}}(d\sigma) = \int \text{applyin}(v_2, v_2', A_2, \sigma)\ \mu_{\mathbb{S}}(d\sigma)$$

Since $v_1, v_2 : \tau' \to \tau$, they must be abstractions. Let $v_1 = \lambda x : \tau'.\, e_1''$ and likewise for v_2. Then the goal reduces to

$$\int \text{evalin}(e_1''\,[x \mapsto v_1']\,, A_1, \sigma)\ \mu_{\mathbb{S}}(d\sigma) = \int \text{evalin}(e_2''\,[x \mapsto v_2']\,, A_2, \sigma)\ \mu_{\mathbb{S}}(d\sigma)$$

That is, by the definition of value measure, the following:

$$\mu_{e_1''[x \mapsto v_1']}(A_1) = \mu_{e_2''[x \mapsto v_2']}(A_2)$$

That follows from $(v_1, v_2) \in \mathcal{V}[\![\tau' \to \tau]\!]$ and the definitions of $\mathcal{V}$ and $\mathcal{E}$. □

Lemma 13 (Op Compatibility).

$$\frac{\Gamma \vdash e_i \approx e_i' : \tau_i \qquad op^n : (\tau_1, \cdots, \tau_n) \to \tau}{\Gamma \vdash op^n(e_1, \cdots, e_n) \approx op^n(e_1', \cdots, e_n') : \tau}$$

Proof. Similar to but simpler than Lemma 12. Since all operations take real-valued arguments, this proof does not rely on Lemma 2. We rely on the fact that δ, the function that interprets primitive operations, takes related arguments to related results, which holds trivially because reals are related only when they are identical. □

Lemma 14 (Factor Compatibility).

$$\frac{\Gamma \vdash e \approx e' : \mathbb{R}}{\Gamma \vdash \textbf{factor}\ e \approx \textbf{factor}\ e' : \mathbb{R}}$$

Proof. Similar to Lemma 13. □

5.2 Fundamental Property

Theorem 15 (Fundamental Property). *If $\Gamma \vdash e : \tau$ then $\Gamma \vdash e \approx e : \tau$.*

Proof. By induction on $\Gamma \vdash e : \tau$.

- **Case** x. Let $(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!]$. We must prove the value measures $\mu_{x \cdot \gamma_i}$ agree on related $(A_1, A_2) \in \mathcal{A}[\![\tau]\!]$. The measures are concentrated on $\gamma_i(x)$ with weight 1, so they agree if those values are related by $\mathcal{V}[\![\tau]\!]$, which they do because the substitutions are related by $\mathcal{G}[\![\Gamma]\!]$.
- **Case** r. The value measures are identical Dirac measures concentrated at r.
- **Case sample**. The value measures are identical.
- **Case** $\lambda x : \tau_1 . e_2$. By Lemma 11.
- **Case** $e\ e'$. By Lemma 12.
- **Case** $op^n(e_1, \cdots, e_n)$. By Lemma 13.
- **Case factor** e. By Lemma 14.

□

Corollary 16 (Reflexivity). *$\mathcal{V}[\![\tau]\!]$, $\mathcal{G}[\![\Gamma]\!]$, and $\mathcal{E}[\![\tau]\!]$ are reflexive.*

One consequence of the fundamental property is that the $\mathcal{A}[\![\tau]\!]$, a binary relation on measurable sets, is the least reflexive relation on measurable sets closed under the $\mathcal{V}[\![\tau]\!]$ relation. We define $\mathcal{A}'[\![\tau]\!]$ as the collection of $\mathcal{V}[\![\tau]\!]$-closed measurable sets. To show two expressions related by $\mathcal{E}[\![\tau]\!]$ it is sufficient to compare their corresponding measures applied to sets in $\mathcal{A}'[\![\tau]\!]$.

Definition 17.

$$A \in \mathcal{A}'[\![\tau]\!] \iff A \in \Sigma_{[\![\tau]\!]} \ \wedge\ \forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!],\ (v_1 \in A \iff v_2 \in A)$$

Lemma 18. *If $(A_1, A_2) \in \mathcal{A}[\![\tau]\!]$ then $A_1 = A_2$, and if $A \in \mathcal{A}'[\![\tau]\!]$ then $(A, A) \in \mathcal{A}[\![\tau]\!]$.*

Proof. By reflexivity of $\mathcal{V}$. □

Corollary 19.

$$(e_1, e_2) \in \mathcal{E}[\![\tau]\!] \iff \forall A \in \mathcal{A}'[\![\tau]\!],\ \mu_{e_1}(A) = \mu_{e_2}(A)$$

Another consequence of the fundamental property is that to prove two expressions related by $\approx$ it suffices to show that they are $\mathcal{E}[\![\tau]\!]$-related when paired with the same arbitrary substitution.

Lemma 20 (Same Substitution Suffices). *If $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$, and if $(e_1 \cdot \gamma, e_2 \cdot \gamma) \in \mathcal{E}[\![\tau]\!]$ for all $\gamma \models \Gamma$, then $\Gamma \vdash e_1 \approx e_2 : \tau$.*

Proof. Let $(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!]$; we must show $(e_1 \cdot \gamma_1, e_2 \cdot \gamma_2) \in \mathcal{E}[\![\tau]\!]$. The premise gives us $(e_1 \cdot \gamma_1, e_2 \cdot \gamma_1) \in \mathcal{E}[\![\tau]\!]$, and we have $(e_2 \cdot \gamma_1, e_2 \cdot \gamma_2) \in \mathcal{E}[\![\tau]\!]$ from the fundamental property (Theorem 15) for e_2. Finally, transitivity (Lemma 10) yields $(e_1 \cdot \gamma_1, e_2 \cdot \gamma_2) \in \mathcal{E}[\![\tau]\!]$. □

Together, Lemmas 19 and 20 simplify the task of proving instances of $\approx$ via arguments about the shape of big-step evaluations and entropy pre-images, as we will see in Section 6.

5.3 Soundness

The logical relation is sound with respect to contextual equivalence.

Theorem 21 (Soundness). *If* $\Gamma \vdash e_1 \approx e_2 : \tau$*, then* $e_1 =^{\Gamma}_{\text{ctx}} e_2$.

Proof. First show $\vdash C[e_1] \approx C[e_2] : \mathbb{R}$ by induction on C, using the compatibility lemmas (11–14). Then unfold the definitions of $\approx$ and $\mathcal{E}[\![\mathbb{R}]\!]$ to get the equivalence of the measures. □

In the next section, we demonstrate the utility of the logical relation by proving a few example equivalences.

6 Proving Equivalences

Having shown that $\approx$ is sound with respect to $=_{\text{ctx}}$, we can now prove instances of contextual equivalence by proving instances of the $\approx$ relation in lieu of thinking about arbitrary real-typed syntactic contexts.

Specific equivalence proofs fall into two classes, which we characterize as structural and deep based on the kind of reasoning involved. Structural equivalences include β_v and commutativity of expressions. In a structural equivalence, the same evaluations happen, just in different regions of the entropy space because the access patterns have been shuffled around. Deep equivalences include conjugacy relationships and other facts about probability distributions; they involve interactions between intermediate measures and mathematical operations. Deep equivalences are a lightweight form of denotational reasoning restricted to the ground type $\mathbb{R}$.

6.1 Structural Equivalences

The first equivalence we prove is β_v, the workhorse of call-by-value functional programming. Unrestricted β conversions (call-by-name) do not preserve equivalence in this language, of course, because they can duplicate (or eliminate) effects. But there is another subset of β conversions, which we call β_S, that moves arbitrary effectful expressions around while avoiding duplication. In particular, β_S permits the reordering of expressions in a way that is unsound for languages with mutation and many other effects but sound for probabilistic programming.

Theorem 22 (β_v). *If* $\Gamma \vdash (\lambda x.\, e)\ v : \tau$*, then* $\Gamma \vdash (\lambda x.\, e)\ v \approx e[x \mapsto v] : \tau$.

We present two proofs of this theorem. The first proof shows a correspondence between evaluation derivations for the redex and contractum.

Proof (by derivation correspondence). For simplicity we assume that the bound variables of $\lambda x.\, e$ are unique and distinct from the domain of Γ; thus the substitution $e\,[x \mapsto v]$ does not need to rename variables to avoid capturing free references in v.

Let $\gamma \models \Gamma$ and let $A \in \mathcal{A}'[\![\tau]\!]$. By Lemmas 19 and 20, it is sufficient to show that

$$\mu_{((\lambda x.\, e)\ v)\cdot\gamma}(A) = \mu_{(e[x\mapsto v])\cdot\gamma}(A)$$

that is,

$$\int \text{evalin}(((\lambda x.\, e)\ v)\cdot\gamma, A, \sigma)\ \mu_{\mathbb{S}}(d\sigma) = \int \text{evalin}((e\,[x\mapsto v])\cdot\gamma, A, \sigma)\ \mu_{\mathbb{S}}(d\sigma)$$

By Lemma 3, it suffices to show that for all $W \in \Sigma_{\mathbb{R}}$, the entropy pre-images have the same measure. That is,

$$\mu_{\mathbb{S}}(\text{evalin}(((\lambda x.\, e)\ v)\cdot\gamma, A, \cdot)^{-1}(W)) = \mu_{\mathbb{S}}(\text{evalin}((e\,[x\mapsto v])\cdot\gamma, A, \cdot)^{-1}(W))$$

Every evaluation of $((\lambda x.\, e)\ v)\cdot\gamma$ has the following form:

$$\dfrac{\pi_2(\sigma) \vdash v\cdot\gamma \Downarrow v\cdot\gamma, 1 \qquad \boxed{\dfrac{\pi_1(\sigma) \vdash \lambda x.\, e\cdot\gamma \Downarrow \lambda x.\, e\cdot\gamma, 1 \quad \begin{array}{c}\sigma' \vdash v\cdot\gamma \Downarrow v\cdot\gamma, 1\\ \vdots \\ \pi_3(\sigma) \vdash (e\,[x\mapsto v])\cdot\gamma \Downarrow v_r, w\end{array}}{}\ \Delta_1}{\sigma \vdash ((\lambda x.\, e)\ v)\cdot\gamma \Downarrow v_r, w}$$

The application of the λ-expression and the syntactic value argument are both trivial. The evaluation of the body expression depends on e; it contains zero or more leaf evaluations of x yielding $v\cdot\gamma$. These leaf evaluations ignore their entropy argument and have weight 1. We refer to the structure of the e evaluation as Δ_1.

Likewise, every evaluation derivation of $(e\,[x\mapsto v])\cdot\gamma$ has the following form:

$$\boxed{\dfrac{\begin{array}{c}\sigma'' \vdash v\cdot\gamma \Downarrow v\cdot\gamma, 1\\ \vdots \\ \sigma \vdash (e\,[x\mapsto v])\cdot\gamma \Downarrow v_r, w\end{array}}{}\ \Delta_2}$$

Δ_2 has exactly the same structure as Δ_1. Consequently, the two expressions evaluate the same if Δ_1 and Δ_2 receive the same entropy. In short:

$$\sigma \vdash ((\lambda x.\, e)\ v)\cdot\gamma \Downarrow v_r, w \iff \pi_3(\sigma) \vdash (e\,[x\mapsto v])\cdot\gamma \Downarrow v_r, w$$

Let $S_1, S_2 \subseteq \mathbb{S}$ be the entropy pre-images of the two expressions:

$$S_1 = \text{evalin}(((\lambda x.\, e)\ v)\cdot\gamma, A, \cdot)^{-1}(W)$$
$$S_2 = \text{evalin}((e\,[x\mapsto v])\cdot\gamma, A, \cdot)^{-1}(W)$$

We conclude that $S_1 = \pi_3^{-1}(S_2)$ and thus the pre-images have the same measure. □

The second proof rewrites the measure of the redex into that of the contractum using integral identities.

Proof (by integral rewriting). As in the first proof, let $\gamma \models \Gamma$ and $A \in \mathcal{A}'[\![\tau]\!]$. It will be sufficient to show that

$$\mu_{((\lambda x.\, e)\ v)\cdot\gamma}(A) = \mu_{(e[x\mapsto v])\cdot\gamma}(A)$$

Using the same steps as in Lemma 12, we can express the value measure of an application as an integral by the value measures of its subexpressions.

$$\mu_{((\lambda x.\, e)\ v)\cdot\gamma}(A) = \iiint \mathrm{applyin}(v', v'', A, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mu_{v\cdot\gamma}(dv'')\ \mu_{\lambda x.\, e\cdot\gamma}(dv')$$

Both the subexpressions $\lambda x.\, e\cdot\gamma$ and $v\cdot\gamma$ are values, so their value measures are Dirac. We complete the proof using the fact that integration by a Dirac measure is equivalent to substitution.

$$\begin{aligned}
&= \iiint \mathrm{applyin}(v', v'', A, \sigma)\ \mu_{\mathbb{S}}(d\sigma)\ \mathrm{dirac}_{v\cdot\gamma}(dv'')\ \mathrm{dirac}_{\lambda x.\, e\cdot\gamma}(dv') \\
&= \int \mathrm{applyin}(\lambda x.\, e\cdot\gamma, v\cdot\gamma, A, \sigma)\ \mu_{\mathbb{S}}(d\sigma) && \text{(integration by Dirac)} \\
&= \int \mathrm{evalin}((e\,[x\mapsto v])\cdot\gamma, A, \sigma)\ \mu_{\mathbb{S}}(d\sigma) && \text{(definition of applyin)} \\
&= \mu_{(e[x\mapsto v])\cdot\gamma}(A) && \text{(definition of } \mu_e\text{)}
\end{aligned}$$

□

The second equivalence concerns reordering expression evaluations. In probabilistic programming, **sample** and **factor** effects can be reordered, as long as they are not duplicated or eliminated. We define simple contexts, a generalization of evaluation contexts, as a class of contexts that an expression may be moved through without changing the number of times it is evaluated.

Definition 23 (Simple Contexts)

$$S ::= [\ ] \mid S\ e \mid e\ S \mid (\lambda x.\, S)\ e \mid op(e, \cdots, S, e, \cdots) \mid \mathbf{factor}\ S$$

Note that $(\lambda x.\, S)\ e$ can also be written $\mathbf{let}\ x = e\ \mathbf{in}\ S$.

Theorem 24 (Substitution into Simple Context). *If $\Gamma \vdash (\lambda x.\, S[x])\ e : \tau$ and x does not occur free in S, then $\Gamma \vdash (\lambda x.\, S[x])\ e \approx S[e] : \tau$.*

Proof. We can prove the following equivalence for a context S^1 consisting of a single frame, such as $([\ ]\ e)$. For all λ-values f where $\Gamma \vdash S^1[f\ e] : \tau$,

$$\Gamma \vdash (\lambda x.\, S^1[f\ x])\ e \approx S^1[f\ e] : \tau$$

The proof is similar to that of Theorem 22, but see also below.

The proof for arbitrary S contexts proceeds by induction on S. The base case, $\Gamma \vdash (\lambda x.\, x)\ e \approx e : \tau$, is easily proven directly. For the inductive case:

$$\begin{aligned}
\Gamma \vdash (\lambda x.\, S^1[S[x]])\ e &\approx (\lambda x.\, S^1[(\lambda y.\, S[y])\ x])\ e && \text{(by Theorem 22)} \\
&\approx S^1[(\lambda y.\, S[y])\ e] && \text{(by single-frame case)} \\
&\approx S^1[S[e]] && \text{(by IH and compatibility of } S^1\text{)}
\end{aligned}$$

□

The β_S and β_v theorems together show that the following terms are equivalent:

- **let** *x* = *e1*, *y* = *e2* **in** *body*
- **let** *y* = *e2*, *x* = *e1* **in** *body*

First *e2* is lifted to the outside with β_S to get **let** *z* = *e2*, *x* = *e1*, *y* = *z* **in** *body*. Then *y* is replaced with *z* in *body* using β_v. Finally, the outer *z* is renamed back to *y*.

This reordering can also be shown directly, and the proof is similar to the $S^1 = (\lambda x.\, [\,])\ e$ case above but simpler to present. It involves a generalization of Lemma 1 (Tonelli).

We first apply the technique from the the second proof of Theorem 22 to express substitution as integration by value measures.

$$\begin{aligned}
\mu_{(\lambda x.\,(\lambda y.\,e_3)\ e_2)\ e_1}(A) &= \int \mu_{((\lambda y.\,e_3)\ e_2)[x \mapsto v_1]}(A)\ \mu_{e_1}(dv_1) \\
&= \iint \mu_{e_3[x \mapsto v_1, y \mapsto v_2]}(A)\ \mu_{e_2}(dv_2)\ \mu_{e_1}(dv_1)
\end{aligned}$$

Doing the same to the other side, we now need to show that the order of integration is interchangable:

$$\iint \mu_{e_3[x \mapsto v_1, y \mapsto v_2]}(A)\ \mu_{e_2}(dv_2)\ \mu_{e_1}(dv_1) = \iint \mu_{e_3[y \mapsto v_2, x \mapsto v_1]}(A)\ \mu_{e_1}(dv_1)\ \mu_{e_2}(dv_2)$$

Since μ_{e_1} and μ_{e_2} may not be σ-finite we cannot immediately apply Lemma 1. Lemma 25 shows exchangability for value measures and completes the proof.

Lemma 25 (μ_e interchangable). *If* $\vdash e_1 : \tau_1$ *and* $\vdash e_2 : \tau_2$ *then for all measurable* $f : [\![\tau_1]\!] \times [\![\tau_2]\!] \to \mathbb{R}^+$,

$$\iint f(v_1, v_2)\ \mu_{e_2}(dv_2)\ \mu_{e_1}(dv_1) = \iint f(v_1, v_2)\ \mu_{e_1}(dv_1)\ \mu_{e_2}(dv_2)$$

Proof. Since integrals about μ_e can be expressed in terms of the σ-finite $\mu_{\mathbb{S}}$, we can apply Lemma 1 (Tonelli) once we have exposed the underlying measures.

$$\begin{aligned}
&\iint f(v_1, v_2)\ \mu_{e_2}(dv_2)\ \mu_{e_1}(dv_1) \\
&= \iint f(\mathrm{ev}(v_1, \sigma_1), \mathrm{ev}(v_2, \sigma_2)) \cdot \mathrm{ew}(v_2, \sigma_2)\ \mu_{\mathbb{S}}(d\sigma_2) \cdot \mathrm{ew}(v_1, \sigma_1)\ \mu_{\mathbb{S}}(d\sigma_1) \\
&\qquad\qquad \text{(Theorem 7)} \\
&= \iint f(\mathrm{ev}(v_1, \sigma_1), \mathrm{ev}(v_2, \sigma_2)) \cdot \mathrm{ew}(v_2, \sigma_2) \cdot \mathrm{ew}(v_1, \sigma_1)\ \mu_{\mathbb{S}}(d\sigma_2)\ \mu_{\mathbb{S}}(d\sigma_1) \\
&\qquad\qquad \text{(linearity of integration)} \\
&= \iint f(\mathrm{ev}(v_1, \sigma_1), \mathrm{ev}(v_2, \sigma_2)) \cdot \mathrm{ew}(v_1, \sigma_1) \cdot \mathrm{ew}(v_2, \sigma_2)\ \mu_{\mathbb{S}}(d\sigma_1)\ \mu_{\mathbb{S}}(d\sigma_2) \\
&\qquad\qquad \text{(Lemma 1)} \\
&= \iint f(v_1, v_2)\ \mu_{e_1}(dv_1)\ \mu_{e_2} dv_2
\end{aligned}$$

□

6.2 Deep Equivalences

In contrast to structural equivalences such as β_v, deep equivalences rely on the specific computations being performed and mathematical relationships between them. They generally concern only expressions of ground type ($\mathbb{R}$). Proving them requires "locally denotational" reasoning about expressions and the real-valued measures (or measure kernels, when free variables are present) they represent.

For example, the following theorem encodes the fact that the sum of two normally-distributed random variables is normally distributed.

Theorem 26 (Sum of normals with variable parameters). *Let*

$$\begin{aligned}
e_1 &= \mathbf{normal}\ x_{m_1}\ x_{s_1} + \mathbf{normal}\ x_{m_2}\ x_{s_2} \\
e_2 &= \mathbf{normal}\ (x_{m_1} + x_{m_2})\ \sqrt{x_{s_1}^2 + x_{s_2}^2}
\end{aligned}$$

and let $\Gamma(x_{m_1}) = \Gamma(x_{m_2}) = \Gamma(x_{s_1}) = \Gamma(x_{s_2}) = \mathbb{R}$. *Then* $\Gamma \vdash e_1 \approx e_2 : \mathbb{R}$.

Proof. Let $(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma]\!]$. Since x_{m_1}, x_{m_2}, x_s have ground type, the substitutions agree on their values: let $m_1 = \gamma_1(x_{m_1}) = \gamma_2(x_{m_1})$ and likewise for m_2, s_1, and s_2.

We must show $(e_1 \cdot \gamma_1, e_2 \cdot \gamma_2) \in \mathcal{E}[\![\mathbb{R}]\!]$; that is, $\mu_{e_1 \cdot \gamma_1}(A) = \mu_{e_2 \cdot \gamma_2}(A)$ for all $A \in \Sigma_{\mathbb{R}}$. The value measures of the **normal** expressions are actually the measures of normally-distributed random variables. This reasoning relies on the meaning assigned to the normalinvcdf operation as well as $+$ and $*$; recall that

$$\mathbf{normal}\ m\ s \triangleq m + s * \mathsf{normalinvcdf}(\mathbf{sample})$$

Then we apply the fact from probability that the sum of two normal random variables is a normal random variable. □

6.3 Combining Equivalences

The transitivity of the logical relation permits equivalence proofs to be decomposed into smaller, simpler steps, using the compatibility lemmas to focus in and rewrite subexpressions of the main expression of interest.

Theorem 27 (Sum of normals). *Let*

$$e_1 = \mathbf{normal}\ e_{m_1}\ e_{s_1} + \mathbf{normal}\ x_{m_2}\ e_{s_1}$$
$$e_2 = \mathbf{normal}\ (e_{m_1} + e_{m_2})\ \sqrt{e_{s_1}^2 + e_{s_2}^2}$$

and let $\Gamma \vdash e_1 : \mathbb{R}$ *and* $\Gamma \vdash e_2 : \mathbb{R}$. *Then* $\Gamma \vdash e_1 \approx e_2 : \mathbb{R}$.

Proof. This theorem is just like Theorem 26 except with expressions instead of variables for the parameters to the normal distributions. We use β_S (Theorem 24) "in reverse" to move the expressions out and replace them with variables, then we apply the variable case (Theorem 26), then we use β_S again to move the parameter expressions back in. □

7 Conclusion

We have defined a logical relation to help prove expressions contextually equivalent in a probabilistic programming language with continuous random variables and a scoring operation. We have proven it sound and demonstrated its usefulness with a number of applications to both structural equivalences like β_v and deep equivalences like the sum of normals.

Acknowledgments We thank Amal Ahmed for her guidance on logical relations, and we thank Theophilos Giannakopoulos, Mitch Wand, and Olin Shivers for many helpful discussions and suggestions.

Bibliography

[1] Bizjak, A., Birkedal, L.: Step-Indexed Logical Relations for Probability, pp. 279–294. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

[2] Borgström, J., Gordon, A.D., Greenberg, M., Margetson, J., Gael, J.V.: Measure transformer semantics for bayesian machine learning. Logical Methods in Computer Science 9(3) (2013)

[3] Borgström, J., Lago, U.D., Gordon, A.D., Szymczak, M.: A lambda-calculus foundation for universal probabilistic programming. In: *Conf. Rec. 21st ACM International Conference on Functional Programming* (Sep 2016)

[4] Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M.A., Guo, J., Li, P., Riddell, A.: Stan: A probabilistic programming language. Journal of Statistical Software (2016)

[5] Culpepper, R.: Gamble. `https://github.com/rmculpepper/gamble` (2015)

[6] Edalat, A., Escardó, M.H.: Integration in Real PCF. Inf. Comput. 160(1), 128–166 (Jul 2000)

[7] Escardó, M.H.: PCF extended with real numbers. Theor. Comput. Sci. 162(1), 79–115 (Aug 1996)

[8] Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: A language for generative models. In: UAI. pp. 220–229 (2008)

[9] Huang, D., Morrisett, G.: An application of computable distributions to the semantics of probabilistic programming languages. In: ESOP '16 (2016)

[10] Kiselyov, O., Shan, C.C.: Embedded probabilistic programming. In: Proc. IFIP TC 2 Working Conference on Domain-Specific Languages. pp. 360–384. DSL '09 (2009)

[11] Lieb, E.H., Loss, M.: Analysis, Graduate Studies in Mathematics, vol. 14. American Mathematical Society (1997)

[12] Lunn, D.J., Thomas, A., Best, N., Spiegelhalter, D.: Winbugs – a bayesian modelling framework: Concepts, structure, and extensibility. Statistics and Computing 10(4), 325–337 (Oct 2000)

[13] Mansinghka, V., Selsam, D., Perov, Y.: Venture: a higher-order probabilistic programming platform with programmable inference (Mar 2014), `http://arxiv.org/abs/1404.0099`

[14] Minka, T., Winn, J., Guiver, J., Webster, S., Zaykov, Y., Yangel, B., Spengler, A., Bronskill, J.: Infer.NET 2.6 (2014), Microsoft Research Cambridge. `http://research.microsoft.com/infernet`

[15] Narayanan, P., Carette, J., Romano, W., Shan, C.c., Zinkov, R.: Probabilistic Inference by Program Transformation in Hakaru (System Description), pp. 62–79. Springer International Publishing (2016)

[16] Park, S., Pfenning, F., Thrun, S.: A probabilistic language based on sampling functions. ACM Trans. Program. Lang. Syst. 31(1), 4:1–4:46 (Dec 2008)

[17] Pfeffer, A.: Figaro: An object-oriented probabilistic programming language. Tech. rep., Charles River Analytics (2009)
[18] Ramsey, N., Pfeffer, A.: Stochastic lambda calculus and monads of probability distributions. In: *Conf. Rec. 29th ACM Symposium on Principles of Programming Languages*. pp. 154–165 (2002)
[19] Sangiorgi, D., Vignudelli, V.: Environmental bisimulations for probabilistic higher-order languages. In: *Conf. Rec. 43rd ACM Symposium on Principles of Programming Languages*. pp. 595–607. POPL '16 (2016)
[20] Staton, S., Yang, H., Heunen, C., Kammar, O., Wood, F.: Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In: *Proc. 31st IEEE Symposium on Logic in Computer Science* (2016)
[21] Wingate, D., Goodman, N.D., Stuhlmüller, A., Siskind, J.M.: Nonstandard interpretations of probabilistic programs for efficient inference. In: Adv. in Neural Inform. Processing Syst. vol. 24 (2011)
[22] Wood, F., van de Meent, J.W., Mansinghka, V.: A new approach to probabilistic programming inference. In: Proc. 17th International Conference on Artificial Intelligence and Statistics. pp. 1024–1032 (2014)