Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints

Sam Staton Hongseok Yang
Frank Wood
University of Oxford
{sam.staton,hongseok.yang}@cs.ox.ac.uk
fwood@robots.ox.ac.uk

Chris Heunen
University of Edinburgh
chris.heunen@ed.ac.uk

Ohad Kammar
University of Cambridge
ohad.kammar@cl.cam.ac.uk

Abstract

We study the semantic foundation of expressive probabilistic programming languages, that support higher-order functions, continuous distributions, and soft constraints (such as Anglican, Church, and Venture). We define a metalanguage (an idealised version of Anglican) for probabilistic computation with the above features, develop both operational and denotational semantics, and prove soundness, adequacy, and termination. They involve measure theory, stochastic labelled transition systems, and functor categories, but admit intuitive computational readings, one of which views sampled random variables as dynamically allocated read-only variables. We apply our semantics to validate nontrivial equations underlying the correctness of certain compiler optimisations and inference algorithms such as sequential Monte Carlo simulation. The language enables defining probability distributions on higher-order functions, and we study their properties.

1. Introduction

Probabilistic programming is the idea to use programs to specify probabilistic models; probabilistic programming languages blend programming constructs with probabilistic primitives. This helps scientists express complicated models succinctly. Moreover, such languages come with generic inference algorithms, relieving the programmer of the nontrivial task of (algorithmically) answering queries about her probabilistic models. This is useful in *e.g.* machine learning.

Several higher-order probabilistic programming languages have recently attracted a substantial user base. Some languages (such as Infer.net [21], PyMC [26], and Stan [33]) are less expressive but provide powerful inference algorithms, while others (such as Anglican [34], Church [12], and Venture [20]) have less efficient inference algorithms but more expressive power. We consider the more expressive languages, that support higher-order functions, continuous distributions, and soft constraints. More precisely, we consider a programming language (§3) with higher-order functions (§6) as well as the following probabilistic primitives.

Sampling The command $\mathsf{sample}(t)$ draws a sample from a distribution described by t, which may range over the real numbers.

Soft constraints The command $\mathsf{score}(t)$ puts a score t (a positive real number) on the current execution trace. This is typically used to record that some particular datum was observed as being drawn from a particular distribution; the lower the score, the more surprising the observation.

Normalisation The command $\mathsf{norm}(u)$ runs a simulation algorithm over the program fragment u. This takes the scores into account and returns a new, normalised probability distribution. The argument to sample might be a primitive distribution, or a distribution defined by normalizing another program. This is called a *nested query*, by analogy with database programming.

Such languages currently lack formal exact semantics. The aim of this paper is to provide just such a foundation as a basis for formal reasoning, improving the unsatisfactory current situation. Most expressive probabilistic programming languages are now explained in terms of their Monte Carlo simulation algorithms. The simplest such algorithm, using importance and rejection sampling, is the *de facto* semantics against which other algorithms are 'proved approximately correct'. Such 'semantics' are hard to handle and extend.

We provide two styles of semantics, operational and denotational. For first-order probabilistic programs, the denotational semantics is straightforward: types are interpreted as measurable spaces, and terms are interpreted as measurable functions (§4). Operational semantics is more complicated. For discrete distributions, an operational semantics might be a probabilistic transition system, but for continuous distributions, it must be a stochastic relation (labelled Markov process). We resolve this by equipping the set of configurations with the structure of a measurable space (§5).

The advantage to the operational semantics is that it is easily extended to higher-order programs (§7). Denotational semantics for higher-order programs poses a problem, because measurable spaces do not support the usual β/η theory of functions: they do not form a Cartesian closed category (indeed, $\mathbb{R}^{\mathbb{R}}$ does not exist as a measurable space [3]). Earlier semantics deal with this by either excluding higher-order functions or considering only discrete distributions. We resolve this by moving from the category of measurable spaces, where standard probability theory takes place, to a functor category based on it (§8). The former embeds in the latter, so we can still interpret first-order concepts. But the functor category does have well-behaved function spaces, so we can also interpret higher-order concepts. Moreover, by lifting the monad that takes probability distributions [11] to the functor category, we can also interpret continuous distributions. Finally, we can interpret observations by considering probability distributions with continuous density, irrespective of the categorical machinery (§9).

The denotational semantics is sound and adequate with respect to the operational semantics (§5.3,8.3), which means one can use the denotational model to directly check program equations whilst respecting computational issues. For example:

- we demonstrate a key program equation for sequential Monte Carlo simulation (§4.1);
- we show that every term of first-order type is equal to one without λ-abstractions or application, and hence is interpreted as a measurable function (Proposition 8.3).

2. Preliminaries

We recall basic definitions and facts of measure theory.

Definition 2.1. A *σ-algebra* on a set X is a family Σ of subsets of X, called *measurable (sub)sets*, which contains X and is closed under complements and countable unions. A *measurable space* is a set with a σ-algebra.

A *probability measure* or *probability distribution* on a measurable space (X, Σ) is a function $p\colon \Sigma \to [0,1]$ to the unit interval satisfying $p(X) = 1$ and $p(\bigcup_{i\in\mathbb{N}} U_i) = \sum_{i\in\mathbb{N}} p(U_i)$ for each sequence $U_1, U_2, \ldots$ of disjoint measurable sets.

A first example is to make a set X into a measurable space by taking the full powerset of X as Σ, yielding a *discrete* measurable space. When X is countable, a probability distribution on (X, Σ) is entirely determined by its values on singleton sets, that is, by specifying a function $p\colon X \to [0,1]$ such that $\sum_{x\in X} p(x) = 1$.

A second example is to combine a collection of measurable spaces $(X_i, \Sigma_i)_{i\in I}$ by *sum* or *product*. The underlying sets in this case are the disjoint union $\sum_{i\in I} X_i$ and product $\prod_{i\in I} X_i$ of sets. The measurable sets in the sum are $\sum_{i\in I} U_i$ for $U_i \in \Sigma_i$. The σ-algebra of the product is the smallest one containing all the subsets $\prod_{i\in I} U_i$ where $U_i \in \Sigma_i$ equals X_i but for a single index i.

For a third example, the real numbers form a measurable space $(\mathbb{R}, \Sigma_{\mathbb{R}})$ under the smallest σ-algebra that contains the open intervals; the measurable sets are called *Borel* sets. Restricting to any measurable subset gives a new measurable space, such as the space $\mathbb{R}_{\geq 0}$ of nonnegative reals and the unit interval $[0,1]$.

A fourth example is to make the set $P(X)$ of all probability measures on a measurable space (X, Σ_X) into a measurable space, by letting $\Sigma_{P(X)}$ be the smallest σ-algebra containing the sets $\{p \in P(X) \mid p(U) \in V\}$ for all $U \in \Sigma_X$ and $V \in \Sigma_{[0,1]}$.

Definition 2.2. Let (X, Σ_X), (Y, Σ_Y) be measurable spaces. A function $f\colon X \to Y$ is *measurable* if $f^{-1}(U) \in \Sigma$ for $U \in \Sigma_Y$.

We can *push forward* a measure along a measurable function: if $p\colon \Sigma_X \to [0,1]$ is a probability measure on (X, Σ_X) and $f\colon X \to Y$ is a measurable function, then $p(U) = p(f^{-1}(U))$ is a probability measure on (Y, Σ_Y).

Definition 2.3. A *stochastic relation* between measurable spaces (X, Σ_X) and (Y, Σ_Y) is a function $r\colon X \times \Sigma_Y \to [0,1]$ such that $r(x, -)\colon \Sigma_Y \to [0,1]$ is a probability distribution for all $x \in X$, and $r(-, V)\colon X \to [0,1]$ is measurable for all $V \in \Sigma_Y$.

Giving a stochastic relation from (X, Σ_X) to (Y, Σ_Y) is equivalent to giving a measurable function $(X, \Sigma_X) \to (P(Y), \Sigma_{P(Y)})$. Stochastic relations $r\colon X \times \Sigma_Y \to [0,1]$ and $s\colon Y \times \Sigma_Z \to [0,1]$ compose associatively to $(s \circ r)\colon X \times \Sigma_Z \to [0,1]$ via the formula

$$(s \circ r)(x, W) = \int_Y s(y, W)\, r(x, \mathrm{d}y).$$

Finally, for a predicate φ, we use the indicator expression $[\varphi]$ to denote 1 if φ holds, and 0 otherwise.

3. A first-order language

This section presents a first-order language for expressing Bayesian probabilistic models and study its properties. The language forms a first-order core of a higher-order extension in Section 6, and provides a simpler setting to illustrate key ideas. For example, the language includes infinitary type and term constructors, constant terms for all measurable functions between measurable spaces, and constructs for specifying Bayesian probabilistic models, namely, operations for sampling distributions, scoring samples, and normalizing distributions based on scores. This highly permissive and slightly unusual syntax is not meant to be a useful programming language itself. Rather, its purpose is to serve as a semantic metalanguage to which a practical programming language compiles, and to provide a common mathematical setting for studying high-level constructs for probabilistic computation.

Types The language has types

$$\mathbb{A}, \mathbb{B} \ ::= \ \mathbb{R} \mid \mathsf{P}(\mathbb{A}) \mid 1 \mid \mathbb{A} \times \mathbb{B} \mid \sum_{i\in I} \mathbb{A}_i$$

where I ranges over countable sets. A type $\mathbb{A}$ stands for a measurable space $[\![\mathbb{A}]\!]$. For example, $\mathbb{R}$ denotes the measurable space of reals, $\mathsf{P}(\mathbb{A})$ is the space of probability measures on $\mathbb{A}$, and 1 is the (discrete) measurable space on the singleton set. The other type constructors correspond to products and sums of measurable spaces. Notice that countable sums are allowed, enabling us to express usual ground types in programming languages via standard encoding. For instance, the type for booleans is $1 + 1$, and that for natural numbers $\sum_{i\in\mathbb{N}} 1$.

Terms We distinguish typing judgements: $\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}$ for deterministic terms, and $\Gamma \vdash_{\mathsf{p}} t\colon \mathbb{A}$ for probabilistic terms (see also e.g. [19, 25, 29]). In both, $\mathbb{A}$ is a type, and Γ is a list of variable/type pairs. Variables stand for deterministic terms, making the following substitution rule derivable:

$$\frac{\Gamma, x\colon \mathbb{A} \vdash_{\mathsf{z}} u\colon \mathbb{B} \quad \Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}}{\Gamma \vdash_{\mathsf{z}} u[t/x]\colon \mathbb{B}} \ (\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\})$$

Intuitively, probabilistic terms $\Gamma \vdash_{\mathsf{p}} t\colon \mathbb{A}$ express computations with effects from two different sources: during evaluation, t may sample a value from a probability distribution, or it may update a variable storing the current *score*, a nonnegative real number expressing to what extent sampled values (from a prior distribution) are compatible with observed data. Evaluating deterministic terms $\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}$, on the other hand, does not generate such effects.

Formally, a context $\Gamma = (x_1\colon \mathbb{A}_1, \ldots, x_n\colon \mathbb{A}_n)$ means a measurable space $[\![\Gamma]\!] \stackrel{\text{def}}{=} \prod_{i=1}^n [\![\mathbb{A}_i]\!]$. Both deterministic terms $\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}$ and probabilistic terms $\Gamma \vdash_{\mathsf{p}} t'\colon \mathbb{A}$ denote measurable functions from $[\![\Gamma]\!]$, but they have different codomains. The former has codomain $[\![\mathbb{A}]\!]$, whereas the latter has codomain $P(\mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!])$. Elements of $P(\mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!])$ are probability distributions on pairs $(r, a) \in \mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!]$, where a is the value obtained through various probabilistic choices, and r the corresponding score.

Sums and products The language includes variables, and constructors and destructors for sum and product types.

$$\frac{}{\Gamma, x\colon \mathbb{A}, \Gamma' \vdash_{\mathsf{d}} x\colon \mathbb{A}} \qquad \frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}_i}{\Gamma \vdash_{\mathsf{d}} (i, t)\colon \sum_{i\in I} \mathbb{A}_i}$$

$$\frac{\Gamma \vdash_{\mathsf{d}} t\colon \sum_{i\in I} \mathbb{A}_i \quad (\Gamma, x\colon \mathbb{A}_i \vdash_{\mathsf{z}} u_i\colon \mathbb{B})_{i\in I}}{\Gamma \vdash_{\mathsf{z}} \mathsf{case}\ t\ \mathsf{of}\ \{(i, x) \Rightarrow u_i\}_{i\in I}\colon \mathbb{B}} \ (\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\})$$

$$\frac{}{\Gamma \vdash_{\mathsf{d}} *\colon 1} \qquad \frac{\Gamma \vdash_{\mathsf{d}} t_j\colon \mathbb{A}_j \text{ for all } j \in \{0, 1\}}{\Gamma \vdash_{\mathsf{d}} (t_0, t_1)\colon \mathbb{A}_0 \times \mathbb{A}_1} \qquad \frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}_0 \times \mathbb{A}_1}{\Gamma \vdash_{\mathsf{d}} \pi_j(t)\colon \mathbb{A}_j}$$

In the rules for sums, I may be infinite. In the last rule, j is 0 or 1. We use some standard syntactic sugar, such as false and true for the injections in the type $\mathsf{bool} = 1 + 1$, and if for case in that instance.

Sequencing We include the standard constructs (e.g. [19, 22]).

$$\frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}}{\Gamma \vdash_{\mathsf{p}} \mathsf{return}(t)\colon \mathbb{A}} \qquad \frac{\Gamma \vdash_{\mathsf{p}} t\colon \mathbb{A} \quad \Gamma, x\colon \mathbb{A} \vdash_{\mathsf{p}} u\colon \mathbb{B}}{\Gamma \vdash_{\mathsf{p}} \mathsf{let}\ x = t\ \mathsf{in}\ u\colon \mathbb{B}}$$

Language-specific constructs The language has constant terms for all measurable functions.

$$\frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}}{\Gamma \vdash_{\mathsf{d}} f(t)\colon \mathbb{B}} \ (f\colon [\![\mathbb{A}]\!] \to [\![\mathbb{B}]\!] \text{ measurable}) \qquad (1)$$

In particular, all the usual distributions are in the language, including the Dirac distribution $dirac(x)$ concentrated on outcome x, the Gaussian distribution $gauss(\mu, \sigma)$ with mean μ and standard deviation σ, the Bernoulli distribution $bern(p)$ with success probability p, the exponential distribution $exp(r)$ with rate r, and the Beta distribution $beta(\alpha, \beta)$ with parameters α, β.[1] For example, from the measurable functions $42.0\colon 1 \to \mathbb{R}$, $e^{(-)}\colon \mathbb{R} \to \mathbb{R}$, $gauss\colon \mathbb{R}\times\mathbb{R} \to P(\mathbb{R})$ and $<\colon \mathbb{R}\times\mathbb{R} \to 1+1$ we can derive:

$$\frac{}{\Gamma \vdash_{\mathsf{d}} 42.0\colon \mathbb{R}} \qquad \frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{R}}{\Gamma \vdash_{\mathsf{d}} e^t\colon \mathbb{R}}$$

$$\frac{\Gamma \vdash_{\mathsf{d}} \mu\colon \mathbb{R} \quad \Gamma \vdash_{\mathsf{d}} \sigma\colon \mathbb{R}}{\Gamma \vdash_{\mathsf{d}} gauss(\mu,\sigma)\colon \mathsf{P}(\mathbb{R})} \qquad \frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{R} \quad \Gamma \vdash_{\mathsf{d}} u\colon \mathbb{R}}{\Gamma \vdash_{\mathsf{d}} t < u\colon \mathsf{bool}}$$

The following terms form the core of our language.

$$\frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathsf{P}(\mathbb{A})}{\Gamma \vdash_{\mathsf{p}} \mathsf{sample}(t)\colon \mathbb{A}} \qquad \frac{\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{R}}{\Gamma \vdash_{\mathsf{p}} \mathsf{score}(t)\colon 1}$$

The first term samples a value from a distribution t, and the second updates the score variable v using t: if t is nonnegative, the term computes the multiplication $t\cdot v$ and stores the result in the variable; if t is negative, it stores 0.0 instead. Since both of these terms express effects, they are typed under $\vdash_{\mathsf{p}}$ instead of $\vdash_{\mathsf{d}}$. The argument t in $\mathsf{score}(t)$ is usually the density of a probability distribution at an observed data point. For instance, in the example

$$\mathsf{score}(density\_gauss(2.0, (\mu, \sigma)))$$

the observed datum is 2.0, the term computes the density of the normal distribution at this datum, and multiplies it to the score variable. The result of this multiplication grows as μ approaches 2.0. Thus the term scores an evaluation up to the term itself higher when the probabilistic choices of evaluation make σ closer to 2.0.

Normalization Two representative tasks of Bayesian inference are to calculate the so-called *posterior distribution* and *model evidence*. Let us illustrate these tasks.

$$\begin{array}{l} \mathsf{let}\ x = \mathsf{sample}(bern(0.25))\ \mathsf{in} \\ \mathsf{let}\ y = \mathsf{if}\ x\ \mathsf{then}\ \mathsf{score}(5.0)\ \mathsf{else}\ \mathsf{score}(2.0)\ \mathsf{in} \\ \mathsf{return}(x) \end{array}$$

Evaluation of this term generates two samples: $(5.0, \mathsf{true})$ and $(2.0, \mathsf{false})$, with probabilities 0.25 and 0.75. Calculating the posterior distribution means normalizing the distribution of these sampled booleans according to their scores, giving a new distribution on bool that assigns $(0.25\cdot 5.0)/(0.25\cdot 5.0 + 0.75\cdot 2.0) \approx 0.45$ to true and $(0.75\cdot 2.0)/(0.25\cdot 5.0+0.75\cdot 2.0) \approx 0.55$ to false. Calculating the model evidence just averages scores according to the original probabilities: $(0.25\cdot 5.0 + 0.75\cdot 2.0) = 2.75$. Intuitively, scores express that the sample $x = \mathsf{true}$ matches an observation better, and change the probability of $x = \mathsf{true}$ from 0.25 to 0.45.

The language includes a term $\mathsf{norm}(t)$ denoting the results of these posterior and model evidence calculations. This term converts a probabilistic term t into a deterministic value, which is its normalized distribution together with the model evidence. The conversion might fail because the model evidence can be zero or infinite, which is notified by $\mathsf{norm}(t)$ by raising an appropriate error.

$$\frac{\Gamma \vdash_{\mathsf{p}} t\colon \mathbb{A}}{\Gamma \vdash_{\mathsf{d}} \mathsf{norm}(t)\colon \mathbb{R}\times \mathsf{P}(\mathbb{A}) + 1 + 1}$$

This construct is being trialled in probabilistic programming languages (such as Anglican). Our first-order language and semantics give a clear formal meaning, enabling mathematical investigation.

[1] The normal distribution is defined for positive standard deviations, but our typing rule also uses the case $\sigma \le 0$. We make the ad-hoc yet safe choice $gauss(\mu,\sigma) = gauss(0.0, 1.0)$, and assume such extensions throughout.

4. Denotational semantics

This section discusses the natural denotational semantics of the first-order language. As described, types $\mathbb{A}$ and contexts Γ are interpreted as measurable spaces $[\![\mathbb{A}]\!]$ and $[\![\Gamma]\!]$, whereas for terms:

- Deterministic terms $\Gamma \vdash_{\mathsf{d}} t\colon \mathbb{A}$ are interpreted as measurable functions $[\![t]\!]\colon [\![\Gamma]\!] \to [\![\mathbb{A}]\!]$, providing a result for each valuation of the context.
- Probabilistic terms $\Gamma \vdash_{\mathsf{p}} t\colon \mathbb{A}$ are interpreted as measurable functions $[\![t]\!]\colon [\![\Gamma]\!] \to P(\mathbb{R}_{\ge 0} \times [\![\mathbb{A}]\!])$, providing a probability measure on (score,result) pairs for each valuation of the context.

The basic idea can be traced back a long way (*e.g.* [17]) but our treatment of score and norm appear to be novel.

Sums and products The interpretation of deterministic terms follows the usual pattern of the internal language of a distributive category (*e.g.* [27]). For instance, $[\![\Gamma, x\colon \mathbb{A}, \Gamma' \vdash_{\mathsf{d}} x\colon \mathbb{A}]\!](\gamma, a, \gamma') \stackrel{\text{def}}{=} a$, and $[\![\Gamma \vdash_{\mathsf{d}} f(t)\colon \mathbb{A}]\!](\gamma) \stackrel{\text{def}}{=} f([\![t]\!](\gamma))$ for measurable $f\colon [\![\mathbb{A}]\!] \to [\![\mathbb{B}]\!]$. This interpretation is actually the same as the usual set-theoretic semantics of the calculus, as one can show by induction that the induced functions $[\![\Gamma]\!] \to [\![A]\!]$ are measurable.

Sequencing For probabilistic terms, we proceed as follows.

$$[\![\mathsf{return}(t)]\!](\gamma)(U) \stackrel{\text{def}}{=} [(1, [\![t]\!](\gamma)) \in U],$$

and $[\![\mathsf{let}\ x = t\ \mathsf{in}\ u]\!](\gamma)(V)$ is

$$\int_{\mathbb{R}_{\ge 0}\times[\![\mathbb{A}]\!]} \Big([\![u]\!](\gamma, x)\big(\{(s,b) \mid (rs, b) \in V\}\big)\Big)\ [\![t]\!](\gamma)(\mathrm{d}(r,x)).$$

As we will explain shortly, these interpretations come from treating $P(\mathbb{R}_{\ge 0}\times(-))$ as a commutative monad, which essentially means the following program equations hold.

$$[\![\mathsf{let}\ x = \mathsf{return}(x)\ \mathsf{in}\ u]\!] = [\![u]\!] \qquad [\![\mathsf{let}\ x = t\ \mathsf{in}\ \mathsf{return}(x)]\!] = [\![t]\!]$$

$$[\![\mathsf{let}\ y = (\mathsf{let}\ x = t\ \mathsf{in}\ u)\ \mathsf{in}\ v]\!] = [\![\mathsf{let}\ x = t\ \mathsf{in}\ \mathsf{let}\ y = u\ \mathsf{in}\ v]\!]$$

$$[\![\mathsf{let}\ x = t\ \mathsf{in}\ \mathsf{let}\ y = u\ \mathsf{in}\ (x,y)]\!] = [\![\mathsf{let}\ y = u\ \mathsf{in}\ \mathsf{let}\ x = t\ \mathsf{in}\ (x,y)]\!]$$

The last equation justifies a useful program optimisation technique [34, §5.5].

Language-specific constructs We use the monad:

$$[\![\mathsf{sample}(t)]\!](\gamma)(U) \stackrel{\text{def}}{=} [\![t]\!](\gamma)(\{a \mid (1,a) \in U\})$$

$$[\![\mathsf{score}(t)]\!](\gamma)(U) \stackrel{\text{def}}{=} [([\![t]\!](\gamma), *) \in U]$$

Here are some program equations to illustrate the semantics so far.

$$[\![\mathsf{score}(7.0); \mathsf{score}(6.1)]\!] = [\![\mathsf{score}(42.7)]\!]$$

$$\left[\!\!\left[\begin{array}{l}\mathsf{let}\ x = \mathsf{sample}(gauss(0.0, 1.0)) \\ \mathsf{in}\ \mathsf{return}(x > 0.0)\end{array}\right]\!\!\right] = [\![\mathsf{sample}(bern(0.5))]\!]$$

$$\left[\!\!\left[\begin{array}{l}\mathsf{let}\ x = \mathsf{sample}(gauss(0.0, 1.0)) \\ \mathsf{in}\ \mathsf{return}(x > x)\end{array}\right]\!\!\right] = [\![\mathsf{return}(\mathsf{false})]\!]$$

Normalisation Interpret $\mathsf{norm}(t)$ by the natural transformation

$$\iota_X\colon P(\mathbb{R}_{\ge 0}\times X) \to (\mathbb{R}\times P(X)) + 1 + 1 \qquad (2)$$

that computes posterior distribution and model evidence by normalisation and summation. More precisely, $\iota_X(p)$ is

$$\begin{cases} (1, *) & \text{if } e = 0 \\ (2, *) & \text{if } e = \infty \\ \Big(0, \big(e,\ \lambda U.\ \frac{1}{e}\int_{\mathbb{R}_{\ge 0}\times X} r\cdot [x \in U]\ p(\mathrm{d}(r,x))\big)\Big) & \text{otherwise} \end{cases}$$

where $e \stackrel{\text{def}}{=} \int_{\mathbb{R}_{\ge 0}\times X} r\, p(\mathrm{d}(r,x))$, and $[\![\mathsf{norm}(t)]\!](\gamma) \stackrel{\text{def}}{=} \iota([\![t]\!](\gamma))$.

Here are some examples:

$$[\![\mathsf{norm}(\mathsf{score}(0.0))]\!] = (1, *)$$

$$[\![\mathsf{norm}(\mathsf{score}(42.0);\mathsf{return}(7.0))]\!] = \big(0,(42.0,\mathit{dirac}(7.0))\big)$$

$$\left[\!\!\left[\begin{array}{l}\mathsf{norm}\big(\mathsf{let}\,x=\mathsf{sample}(\mathit{bern}(0.5))\\ \quad\mathsf{in}\big(\mathsf{if}\;x\;\mathsf{then}\;\mathsf{score}(7.0)\\ \qquad\mathsf{else}\;\mathsf{score}(3.0)\big);\\ \quad\mathsf{return}(x)\big)\end{array}\right]\!\!\right] = \big(0,(5.0,\mathit{bern}(0.7))\big)$$

$$\left[\!\!\left[\begin{array}{l}\mathsf{norm}\big(\mathsf{let}\,x=\mathsf{sample}(\mathit{exp}(1.0))\\ \quad\mathsf{in}\;\mathsf{score}(e^x)\big)\end{array}\right]\!\!\right] = (2,*)$$

$$\left[\!\!\left[\begin{array}{l}\mathsf{norm}\big(\mathsf{let}\,x=\mathsf{sample}(\mathit{beta}(1,3))\\ \quad\mathsf{in}\;\mathsf{score}(x);\mathsf{return}(x)\big)\end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l}\mathsf{norm}\big(\\ \;\mathsf{score}(1/(1+3));\\ \;\mathsf{sample}(\mathit{beta}(2,3))\big)\end{array}\right]\!\!\right]$$

In the third equation, a score of either 7.0 or 3.0 is assigned depending on the outcome of a fair coin toss. The model evidence is $5.0 = (0.5 \cdot 7.0) + (0.5 \cdot 3.0)$, and the normalised distribution, taking the scores into account, is $\mathit{bern}(\frac{0.5 \cdot 7.0}{5.0})$. The fourth equation shows how infinite model evidence errors can arise when working with infinite distributions. In the last equation, the parameter x of $\mathsf{score}(x)$ represents the probability of true under $\mathit{bern}(x)$. The equation expresses the so called conjugate-prior relationship between Beta and Bernoulli distributions, which has been used to optimise probabilistic programs [35].

Monads The interpretation of let and return arise from the fact that $P\big(\mathbb{R}_{\geq 0} \times (-)\big)$ is a commutative monad on the category of measurable spaces and measurable functions (see also [6, §2.3.1]). Recall that a commutative monad (T, η, μ, σ) in general comprises an endofunctor T together with natural transformations $\eta_X : X \to T(X)$, $\mu_X : T(T(X)) \to T(X)$, $\sigma_{X,Y} : T(X) \times Y \to T(X \times Y)$ satisfying some laws [16]. Using this structure we interpret return and let, for $\Gamma \models_{\mathsf{p}} t \colon \mathbb{A}$, following Moggi [22]:

$$[\![\Gamma \models_{\mathsf{p}} \mathsf{return}(t) \colon \mathbb{A}]\!](\gamma) \stackrel{\text{def}}{=} \eta_{[\![\mathbb{A}]\!]}\big([\![t]\!](\gamma)\big)$$
$$[\![\Gamma \models_{\mathsf{p}} \mathsf{let}\,x = t\,\mathsf{in}\,u \colon \mathbb{B}]\!](\gamma) \stackrel{\text{def}}{=} \mu_{[\![\mathbb{B}]\!]}\big(T([\![u]\!])\big(\sigma_{[\![\Gamma]\!],[\![\mathbb{A}]\!]}(\gamma, [\![t]\!](\gamma))\big)\big)$$

Concretely, we make $P(\mathbb{R}_{\geq 0} \times (-))$ into a monad by combining the standard commutative monad structure [11] on P and the commutative monoid $(\mathbb{R}_{\geq 0}, \cdot, 1)$ with the monoid monad transformer.

4.1 Sequential Monte Carlo simulation

The program equations above justify some simple program transformations. For a more sophisticated one, consider *sequential Monte Carlo simulation* [7]. A key idea of its application to probabilistic programs is: 'Whenever there is a score, it is good to renormalise and resample'. This increases efficiency by avoiding too many program executions with low scores [24, Algorithm 1].

The denotational semantics justifies the soundness of this transformation. For a term with top-level score, *i.e.* a term of the form $(\mathsf{let}\,x = t\,\mathsf{in}\,(\mathsf{score}(u); v))$ where u and v may have x free:

$$\begin{array}{lr}[\![\mathsf{norm}(\mathsf{let}\,x = t\,\mathsf{in}\,(\mathsf{score}(u); v))]\!] & \\ = [\![\mathsf{norm}(\mathsf{case}\;(\mathsf{norm}(\mathsf{let}\,x = t\,\mathsf{in}\,\mathsf{score}(u)))\;\mathsf{of} & 1\\ \qquad (0,(e,d)) \Rightarrow \mathsf{score}(e); \mathsf{let}\,x = \mathsf{sample}(d)\,\mathsf{in}\,v & 2\\ \qquad \mid (1,*) \Rightarrow \mathsf{score}(0); \mathsf{return}(w) & 3\\ \qquad \mid (2,*) \Rightarrow \mathsf{score}(\infty); \mathsf{return}(w))]\!] & 4\end{array}$$

Let us explain the right hand side. Line 1 renormalises the program after the score, and in non-exceptional execution returns the model evidence e and a new normalised distribution d. Line 2 immediately records the evidence e as a score, and then resamples d, using the resampled value in the continuation v. Lines 3 and 4 propagate the errors of 0 or ∞ evidence: w is a deterministic term of the right type, and $\mathsf{score}(\infty)$ is a term that causes an infinite evidence error when normalised; which terms are chosen does not matter.

5. Operational semantics

This section develops operational semantics for the first-order language. There are several reasons to consider this, even though the denotational semantics is arguably straightforward. First, extension to higher-order functions is easier in operational semantics than in denotational semantics. Second, operational semantics conveys computational intuitions that are obscured in the denotational semantics. We expect these computational intuitions to play an important role in studying approximate techniques for performing posterior inference, such as sequential Monte Carlo, in the future.

Sampling probability distributions complicates operational semantics. Sampling from a discrete distribution can immediately affect control flow. For example, in the term

$$\mathsf{let}\,x = \mathsf{sample}(\mathit{bern}(0.5))\;\mathsf{in}\;\mathsf{if}\;x\;\mathsf{then}\;\mathsf{return}(1.1)\;\mathsf{else}\;\mathsf{return}(8.0)$$

the conditional depends on the result of sampling the Bernoulli distribution. The result is 1.1 with probability 0.5 (*cf.* [5, §2.3]).

Sampling a distribution on $\mathbb{R}$ cannot affect control flow, but does introduce another complication. Informally, there is a transition

$$\mathsf{sample}(\mathit{gauss}(0.0, 1.0)) \longrightarrow \mathsf{return}(r)$$

for every real r, but any single transition has zero probability. We can assign non-zero probabilities to sets of transitions; informally:

$$\Pr\Big(\mathsf{sample}(\mathit{gauss}(0.0,1.0)) \longrightarrow \{\mathsf{return}(r) \mid r \leq 0\}\Big) = 0.5.$$

To make this precise we need a σ-algebra on the set of terms, which can be done using *configurations* rather than individual terms. A configuration is a *closure* (*cf.* [13, §3]): a pair $\langle t, \gamma \rangle$ of a term t with free variables and a context γ giving values for those variables as elements of a measurable space.

Sampling a distribution p on $\mathbb{R} + \mathbb{R}$ exhibits both complications:

$$\begin{array}{l}\mathsf{let}\,x = \mathsf{sample}(p)\,\mathsf{in}\,\mathsf{case}\,x\,\mathsf{of}\;(0,r) \Rightarrow \mathsf{return}(r + 1.0)\\ \qquad\qquad\qquad\qquad\qquad\quad \mid (1,r) \Rightarrow \mathsf{return}(r - 1.0)\end{array} \tag{3}$$

The control flow in the case distinction depends on which summand is sampled, but there is potentially a continuous distribution over the return values. We handle this by instantiating the choice of summand in the syntax, but keeping the value of the summand in the closure, so that expression (3) can make a step to the closure

$$\left\langle \begin{array}{l}\mathsf{let}\,x = \mathsf{return}(0,y)\,\mathsf{in}\\ \mathsf{case}\,x\,\mathsf{of}\;(0,r) \Rightarrow \mathsf{return}(r+1.0)\\ \qquad\quad \mid (1,r) \Rightarrow \mathsf{return}(r-1.0)\end{array}, y \mapsto 42.0 \right\rangle.$$

A type is *indecomposable* if it has the form $\mathbb{R}$ or $\mathsf{P}(\mathbb{A})$, and a context Γ is *canonical* if it only involves indecomposable types.

Configurations Let $\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\}$. A z*-configuration* of type $\mathbb{A}$ is a triple $\langle \Gamma, t, \gamma \rangle$ comprising a canonical context Γ, a derivable term $\Gamma \models_{\mathsf{z}} t \colon \mathbb{A}$, and an element γ of the measurable space $[\![\Gamma]\!]$. We identify contexts that merely rename variables, such as

$$\langle (x \colon \mathbb{R}, y \colon \mathsf{P}(\mathbb{R})), f(x,y), (x \mapsto 42.0, y \mapsto \mathit{gauss}(0.0, 1.0)) \rangle$$
$$\approx \langle (u \colon \mathbb{R}, v \colon \mathsf{P}(\mathbb{R})), f(u,v), (u \mapsto 42.0, v \mapsto \mathit{gauss}(0.0, 1.0)) \rangle.$$

We call d-configurations *deterministic configurations*, and p-configurations *probabilistic configurations*; they differ only in typing. We will abbreviate configurations to $\langle t, \gamma \rangle$ when the context Γ is obvious. Each configuration has a unique type, because the language does not include any structural typing rules.

Values v in a canonical context Γ are well-typed deterministic terms of the form

$$v, w \;::=\; x_i \;\mid\; * \;\mid\; (v, w) \;\mid\; (i, v) \tag{4}$$

where x_i is a variable in Γ. Similarly, a probabilistic term t in context Γ is called *probabilistic value* or *p-value* if $t \equiv \mathsf{return}(v_0)$

for some value v_0. Remember from Section 3 that the denotational semantics of values is simple and straightforward.

Write $\mathrm{Con}_{\mathsf{d}}(\mathbb{A})$ and $\mathrm{Con}_{\mathsf{p}}(\mathbb{A})$ for the sets of deterministic and probabilistic configurations of type $\mathbb{A}$, and make them into measurable spaces by declaring $U \subseteq \mathrm{Con}_{\mathsf{z}}(\mathbb{A})$ to be measurable if the set $\{\gamma \in [\![\Gamma]\!] \mid \langle t, \gamma\rangle \in U\}$ is measurable for all judgements $\Gamma \vdash_{\mathsf{z}} t \colon \mathbb{A}$.

$$\mathrm{Con}_{\mathsf{z}}(\mathbb{A}) = \sum_{\substack{(\Gamma,t) \\ \Gamma \text{ canonical,} \\ \Gamma \vdash_{\mathsf{z}} t \,:\, \mathbb{A}}} [\![\Gamma]\!] \tag{5}$$

Further partition $\mathrm{Con}_{\mathsf{z}}(\mathbb{A})$ into $\mathrm{ConV}_{\mathsf{z}}(\mathbb{A})$ and $\mathrm{ConN}_{\mathsf{z}}(\mathbb{A})$ based on whether a term in a configuration is a value or not:

$$\begin{aligned}
\mathrm{ConV}_{\mathsf{d}}(\mathbb{A}) &= \{\langle\Gamma, t, \gamma\rangle \in \mathrm{Con}_{\mathsf{d}}(\mathbb{A}) \mid t \text{ is a value}\} \\
\mathrm{ConN}_{\mathsf{d}}(\mathbb{A}) &= \{\langle\Gamma, t, \gamma\rangle \in \mathrm{Con}_{\mathsf{d}}(\mathbb{A}) \mid t \text{ is not a value}\} \\
\mathrm{ConV}_{\mathsf{p}}(\mathbb{A}) &= \{\langle\Gamma, t, \gamma\rangle \in \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) \mid t \text{ is a p-value}\} \\
\mathrm{ConN}_{\mathsf{p}}(\mathbb{A}) &= \{\langle\Gamma, t, \gamma\rangle \in \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) \mid t \text{ is not a p-value}\}
\end{aligned}$$

Particularly well-behaved values are the *ordered values* $\Gamma \vdash_{\mathsf{d}} v \colon \mathbb{A}$, whose variables appear exactly once, in the same order as in Γ.

Lemma 5.1. *Consider a canonical context Γ, a type $\mathbb{A}$, an ordered value $\Gamma \vdash_{\mathsf{d}} v \colon \mathbb{A}$, and the induced measurable function*

$$[\![v]\!] : [\![\Gamma]\!] \to [\![\mathbb{A}]\!].$$

The collection of all such functions for given $\mathbb{A}$ is countable, and forms a coproduct diagram.

Proof. By induction on the structure of types. The key fact is that every type is a sum of products of indecomposable ones, because the category of measurable spaces is distributive, *i.e.* the canonical map $\sum_{i\in I}(\mathbb{A} \times \mathbb{B}_i) \to \mathbb{A} \times \sum_{i \in I} \mathbb{B}_i$ is an isomorphism. □

For example, $\mathbb{A} = (\mathbb{R} \times \mathsf{bool}) + (\mathbb{R} \times \mathbb{R})$ has 3 ordered values, first $(x \colon \mathbb{R} \vdash_{\mathsf{d}} (0, (x, \mathsf{true})) \colon \mathbb{A})$, second $(x \colon \mathbb{R} \vdash_{\mathsf{d}} (0, (x, \mathsf{false})) \colon \mathbb{A})$, and third $(x \colon \mathbb{R}, y \colon \mathbb{R} \vdash_{\mathsf{d}} (1, (x, y)) \colon \mathbb{A})$, inducing a canonical measurable isomorphism $\mathbb{R} + \mathbb{R} + \mathbb{R} \times \mathbb{R} \cong [\![\mathbb{A}]\!]$.

Evaluation contexts We distinguish three kinds of *evaluation contexts*: $\mathcal{C}[-]$ is a context for a deterministic term with a hole for deterministic terms; $\mathcal{D}[-]$ and $\mathcal{E}[-]$ are contexts for probabilistic terms, the former with a hole for probabilistic terms, the latter with a hole for probabilistic terms.

$$\begin{aligned}
\mathcal{C}[-] \;&::=\; (-) \mid \pi_j\, \mathcal{C}[-] \mid (\mathcal{C}[-], t) \mid (v, \mathcal{C}[-]) \mid (i, \mathcal{C}[-]) \\
&\quad\;\mid \mathsf{case}\ \mathcal{C}[-]\ \mathsf{of}\ \{(i,x) \Rightarrow t_i\}_{i \in I} \mid f(\mathcal{C}[-]) \\
\mathcal{D}[-] \;&::=\; (-) \mid \mathsf{let}\ x = \mathcal{D}[-]\ \mathsf{in}\ t \\
\mathcal{E}[-] \;&::=\; \mathcal{D}[\mathsf{return}[-]] \mid \mathcal{D}[\mathsf{sample}[-]] \mid \mathcal{D}[\mathsf{score}[-]] \\
&\quad\;\mid \mathsf{case}\ \mathcal{D}[-]\ \mathsf{of}\ \{(i,x) \Rightarrow t_i\}_{i \in I}
\end{aligned} \tag{6}$$

where t, t_i are general terms and v is a value.

5.1 Reduction

Using the tools developed so far, we will define a measurable function for describing the reduction of d-configurations, and a stochastic relation for describing that of p-configurations:

$$\begin{aligned}
\longrightarrow \;&:\; \mathrm{ConN}_{\mathsf{d}}(\mathbb{A}) \to \mathrm{Con}_{\mathsf{d}}(\mathbb{A}), \\
\longrightarrow \;&:\; \mathrm{ConN}_{\mathsf{p}}(\mathbb{A}) \times \Sigma_{\mathbb{R}_{\geq 0} \times \mathrm{Con}_{\mathsf{p}}(\mathbb{A})} \to [0,1],
\end{aligned}$$

parameterised by a family of measurable functions

$$\nu_{\mathbb{A}} \;:\; \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) \to \big(\mathbb{R}_{\geq 0} \times P([\![\mathbb{A}]\!])\big) + 1 + 1 \tag{7}$$

indexed by types $\mathbb{A}$.

Reduction of deterministic terms Define a type-indexed family of relations $\longrightarrow \subseteq \mathrm{ConN}_{\mathsf{d}}(\mathbb{A}) \times \mathrm{Con}_{\mathsf{d}}(\mathbb{A})$ as the least one that is closed under the following rules.

$$\langle\Gamma, \pi_j(v_0, v_1), \gamma\rangle \longrightarrow \langle\Gamma, v_j, \gamma\rangle$$

$$\langle\Gamma, \mathsf{case}\ (i', v)\ \mathsf{of}\ \{(i,x) \Rightarrow t_i\}_{i\in I}, \gamma\rangle \longrightarrow \langle\Gamma, t_{i'}[v/x], \gamma\rangle$$

$$\langle\Gamma, f(w), \gamma\rangle \longrightarrow \langle(\Gamma, \Gamma'), v, (\gamma, \gamma')\rangle$$
$$(w \text{ a value} \wedge \Gamma' \vdash_{\mathsf{d}} v \colon \mathbb{A} \text{ an ordered value} \wedge f([\![w]\!](\gamma)) = [\![v]\!](\gamma'))$$

$$\langle\Gamma, \mathsf{norm}(t), \gamma\rangle \longrightarrow \langle(\Gamma, x{:}\mathbb{R}, y{:}\mathsf{P}(\mathbb{B})), (0, (x,y)), \gamma[x \mapsto r, y \mapsto p]\rangle$$
$$(\mathbb{A} = \mathsf{P}(\mathbb{B}) \wedge \nu_{\mathbb{B}}(\langle\Gamma, t, \gamma\rangle) = (0, (r, p)) \wedge x, y \notin \mathit{dom}(\Gamma))$$

$$\langle\Gamma, \mathsf{norm}(t), \gamma\rangle \longrightarrow \langle\Gamma, (i, *), \gamma\rangle$$
$$(\mathbb{A} = \mathsf{P}(\mathbb{B}) \wedge \nu_{\mathbb{B}}(\langle\Gamma, t, \gamma\rangle) = (i, *),\ i \in \{1, 2\})$$

$$\frac{\langle\Gamma, t, \gamma\rangle \longrightarrow \langle\Gamma', t', \gamma'\rangle}{\langle\Gamma, \mathcal{C}[t], \gamma\rangle \longrightarrow \langle\Gamma', \mathcal{C}[t'], \gamma'\rangle} \qquad (\mathcal{C}[-] \text{ is not } (-))$$

The rule for $f(w)$ keeps the original context Γ and the closure γ because they might be used in the continuation, even though they are not used in v. The rules obey the following invariant.

Lemma 5.2. *If $\langle\Gamma, t, \gamma\rangle \longrightarrow \langle\Gamma', t', \gamma'\rangle$, then $\Gamma' = (\Gamma, \Gamma'')$ and $\gamma' = (\gamma, \gamma'')$ for some Γ'' and $\gamma'' \in [\![\Gamma]\!]''$.*

Proof. By induction on the structure of derivations. □

This lemma allows us to confirm that our specification of a relation $\longrightarrow \subseteq \mathrm{ConN}_{\mathsf{d}}(\mathbb{A}) \times \mathrm{Con}_{\mathsf{d}}(\mathbb{A})$ is well-formed ('type preservation').

Proposition 5.3. *The induced relation is a measurable function.*

Proof. There are three things to show: that the relation is entire ('progress'); that the relation is single-valued ('determinacy'); and that the induced function is measurable. All three are shown by induction on the structure of terms. The case of application of measurable functions crucially uses Lemma 5.1. □

Reduction of probabilistic terms Next, we define the stochastic relation $\longrightarrow$ for probabilistic terms, combining two standard approaches: for indecomposable types, which are uncountable, use labelled Markov processes, *i.e.* give a distribution on the measurable set of resulting configurations; for decomposable types (sums, products *etc.*), probabilistic branching is discrete and so a transition system labelled by probabilities suffices.

Proposition 5.4. *Let $(X_i)_{i\in I}$ be an indexed family of measurable spaces. Suppose we are given:*

- *a function $q \colon I \to [0,1]$ such that $\sum_{i\in I} q(i) = 1$ that is only nonzero on a countable subset $I_0 \subseteq I$;*
- *a probability measure q_i on X_i for each $i \in I_0$.*

This determines a probability measure p on $\sum_{i\in I} X_i$ by

$$p(U) = \textstyle\sum_{i \in I} q(i)\, q_i(\{a \mid (i,a) \in U\})$$

for U a measurable subset of $\sum_{i\in I} X_i$,

We will use three other entities to define the desired stochastic relation $\longrightarrow \;:\; \mathrm{ConN}_{\mathsf{p}}(\mathbb{A}) \times \Sigma_{\mathbb{R}_{\geq 0} \times \mathrm{Con}_{\mathsf{p}}(\mathbb{A})} \to [0,1]$.

1. A countably supported probability distribution on the set $\{(\Gamma, t) \mid \Gamma \vdash_{\mathsf{p}} t \colon \mathbb{A}\}$ for each $C \in \mathrm{ConN}_{\mathsf{p}}(\mathbb{A})$. We write $\Pr(C \longrightarrow (\Gamma, t))$ for the probability of (Γ, t).
2. A probability measure on the space $[\![\Gamma]\!]$ for each $C \in \mathrm{ConN}_{\mathsf{p}}(\mathbb{A})$ and (Γ, t) with $\Pr(C \longrightarrow (\Gamma, t)) \neq 0$. Write $\Pr(C \longrightarrow_{\Gamma,t} U)$ for the probability of a measurable subset $U \subseteq [\![\Gamma]\!]$.

$$\Pr(\langle \Gamma, \mathcal{E}[t], \gamma\rangle \longrightarrow (\Gamma', \mathcal{E}[t'])) \stackrel{\text{def}}{=} [\langle \Gamma, t, \gamma\rangle \longrightarrow \langle \Gamma', t', \gamma'\rangle]$$

$$\Pr(\langle \Gamma, \mathcal{D}[t], \gamma\rangle \longrightarrow (\Gamma', \mathcal{D}[t'])) \stackrel{\text{def}}{=} \Pr(\langle \Gamma, t, \gamma\rangle \longrightarrow (\Gamma', t'))$$

$$\Pr(\langle \Gamma, \mathsf{let}\, x = \mathsf{return}(v)\, \mathsf{in}\, t, \gamma\rangle \longrightarrow (\Gamma, t[v/x])) \stackrel{\text{def}}{=} 1$$

$$\Pr(\langle \Gamma, \mathsf{case}\,(j, v)\, \mathsf{of}\, \{(i, x) \Rightarrow t_i\}_{i\in I}, \gamma\rangle \longrightarrow (\Gamma, t_j[v/x])) \stackrel{\text{def}}{=} 1$$

$$\Pr(\langle \Gamma, \mathsf{score}(v), \gamma\rangle \longrightarrow (\Gamma, \mathsf{return}(*))) \stackrel{\text{def}}{=} 1$$

$$\Pr(\langle \Gamma, \mathsf{sample}(v), \gamma\rangle \longrightarrow ((\Gamma, \Gamma'), \mathsf{return}(v')))$$
$$\stackrel{\text{def}}{=} \begin{cases} [\![v]\!](\gamma)(\{[\![v']\!](\gamma') \mid \gamma' \in [\![\Gamma']\!]\}) & \text{if } \Gamma' \vdash_{\mathsf{d}} v' \colon \mathbb{A} \text{ ordered} \\ 0 & \text{otherwise} \end{cases}$$

$$\Pr(\langle \Gamma, \mathcal{E}[t], \gamma\rangle \longrightarrow_{(\Gamma', \mathcal{E}[t'])} U)$$
$$\stackrel{\text{def}}{=} [\langle \Gamma, t, \gamma\rangle \longrightarrow \langle \Gamma', t', \gamma'\rangle \wedge \gamma' \in U]$$

$$\Pr(\langle \Gamma, \mathcal{D}[t], \gamma\rangle \longrightarrow_{(\Gamma', \mathcal{D}[t'])} U) \stackrel{\text{def}}{=} \Pr(\langle \Gamma, t, \gamma\rangle \longrightarrow_{(\Gamma', t')} U)$$

$$\Pr(\langle \Gamma, \mathsf{let}\, x = \mathsf{return}(v)\, \mathsf{in}\, t, \gamma\rangle \longrightarrow_{(\Gamma, t[v/x])} U) \stackrel{\text{def}}{=} [\gamma \in U]$$

$$\Pr(\langle \Gamma, \mathsf{case}\,(j, v)\, \mathsf{of}\, \{(i, x) \Rightarrow t_i\}_{i\in I}, \gamma\rangle \longrightarrow_{\Gamma, t_j[v/x]} U) \stackrel{\text{def}}{=} [\gamma \in U]$$

$$\Pr(\langle \Gamma, \mathsf{score}(v), \gamma\rangle \longrightarrow_{(\Gamma, \mathsf{return}(*))} U) \stackrel{\text{def}}{=} [\gamma \in U]$$

$$\Pr(\langle \Gamma, \mathsf{sample}(v), \gamma\rangle \longrightarrow_{((\Gamma, \Gamma'), \mathsf{return}(v'))} U)$$
$$\stackrel{\text{def}}{=} \frac{[\![v]\!](\gamma)(\{[\![v']\!](\gamma') \mid \gamma' \in [\![\Gamma']\!] \wedge (\gamma, \gamma') \in U\})}{[\![v]\!](\gamma)(\{[\![v']\!](\gamma') \mid \gamma' \in [\![\Gamma']\!]\})}$$

$$\mathrm{Sc}(\langle \Gamma, \mathcal{E}[t], \gamma\rangle) \stackrel{\text{def}}{=} 1 \qquad \mathrm{Sc}(\langle \Gamma, \mathcal{D}[t], \gamma\rangle) \stackrel{\text{def}}{=} \mathrm{Sc}(\langle \Gamma, t, \gamma\rangle)$$

$$\mathrm{Sc}(\langle \Gamma, \mathsf{score}(v), \gamma\rangle) \stackrel{\text{def}}{=} [\![v]\!](\gamma) \qquad \mathrm{Sc}(\langle \Gamma, \mathsf{sample}(v), \gamma\rangle) \stackrel{\text{def}}{=} 1$$

$$\mathrm{Sc}(\langle \Gamma, \mathsf{let}\, x = \mathsf{return}(v)\, \mathsf{in}\, t, \gamma\rangle) \stackrel{\text{def}}{=} 1$$

$$\mathrm{Sc}(\langle \Gamma, \mathsf{case}\,(j, v)\, \mathsf{of}\, \{(i, x) \Rightarrow t_i\}_{i\in I}, \gamma\rangle) \stackrel{\text{def}}{=} 1$$

Figure 1. Entities used to define reduction of probabilistic terms

3. A measurable function $\mathrm{Sc} \colon \mathrm{ConN}_{\mathsf{p}}(\mathbb{A}) \to \mathbb{R}_{\geq 0}$, representing the score of the one-step transition relation. (For one-step transitions, the score is actually deterministic. We did not include it in 2 above for simplicity.)

These three entities are defined by induction on the structure of syntax of $\mathbb{A}$-typed p-configurations in Figure 5.1.

Proposition 5.5. *The map* $\mathrm{ConN}_{\mathsf{p}}(\mathbb{A}) \times \Sigma_{\mathbb{R}_{\geq 0} \times \mathrm{Con}_{\mathsf{p}}(A)} \to [0, 1]$ *that sends* (C, U) *to* $\Pr(C \longrightarrow U)$*, defined as*

$$\sum_{(\Gamma, t)} \Pr\big(C \longrightarrow (\Gamma, t)\big) \Pr\big(C \longrightarrow_{\Gamma, t} \{\gamma \mid (\mathrm{Sc}(C), \langle \Gamma, t, \gamma\rangle) \in U\}\big),$$

is a stochastic relation.

Proof. For each p-configuration $C = \langle \_, t, \_\rangle$, use induction on t to see that the probability distribution $\Pr\big(C \longrightarrow (-)\big)$ on pairs (Γ', t') and the distribution $\Pr(C \longrightarrow_{(-)} (-))$ indexed by such pairs satisfy the conditions in Proposition 5.4. It follows that the partial evaluation $\Pr(C \longrightarrow (-))$ of the function in the statement is a probability measure, so it suffices to establish measurability of the other partial evaluation $\Pr((-) \longrightarrow U)$. Recall that $\mathrm{ConN}_{\mathsf{p}}(\mathbb{A})$ is defined in terms of the sum of measurable spaces, and that all p-configurations in each summand have the same term. Finally, use induction on the term shared by all p-configurations in the summand to see that the restriction of $\Pr((-) \longrightarrow U)$ to each summand is measurable. □

5.2 Termination

To see that the reduction process terminates, we first define the transitive closure. This is subtle, as sampling can introduce countably infinite branching; although each branch will terminate, the required number of steps might not be bounded across all branches.

Use the deterministic transition relation to define an evaluation relation $\Downarrow \subseteq \mathrm{Con}_{\mathsf{d}}(\mathbb{A}) \times \mathrm{ConV}_{\mathsf{d}}(\mathbb{A})$, by setting $C \Downarrow D$ if $\exists n.\, C \Downarrow^n D$, where

$$\frac{}{C \Downarrow^0 C}\ (C \in \mathrm{ConV}_{\mathsf{d}}(\mathbb{A})) \qquad \frac{C \longrightarrow D \quad D \Downarrow^n E}{C \Downarrow^{n+1} E}$$

To define evaluation for probabilistic configurations, we need *sub-stochastic relations*: functions $f \colon X \times \Sigma_Y \to [0, 1]$ that are measurable in X, satisfy $f(x, Y) \leq 1$ for every $x \in X$, and are countably additive in Y, *i.e.* $f(x, \bigcup_{i\in\mathbb{N}} U_i) = \sum_{i\in\mathbb{N}} f(U_i)$ for a sequence $U_1, U_2, \ldots$ of disjoint measurable sets. Thus a stochastic relation (as in Definition 2.3) is a sub-stochastic relation with $f(x, Y) = 1$. Define a sub-stochastic relation

$$\Pr(- \Downarrow -) : \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) \times \Sigma_{\mathbb{R}_{\geq 0} \times \mathrm{ConV}_{\mathsf{p}}(\mathbb{A})} \to [0, 1]$$

by $\Pr(C \Downarrow U) \stackrel{\text{def}}{=} \sum_n \Pr(C \Downarrow^n U)$, where $\Pr(C \Downarrow^0 U)$ is given by $[(1, C) \in U]$, and $\Pr(C \Downarrow^{n+1} U)$ is

$$\int_{(r, D)} \Pr(D \Downarrow^n \{(s, E) \mid (r \cdot s, E) \in U\}) \Pr(C \longrightarrow \mathrm{d}(r, D)).$$

Proposition 5.6 (Termination)**.** *Evaluation of deterministic terms is a function:* $\forall C.\, \exists D.\, C \Downarrow D$*. Evaluation of probabilistic terms is a stochastic relation:* $\forall C.\, \Pr(C \Downarrow (\mathbb{R}_{\geq 0} \times \mathrm{ConV}_{\mathsf{p}}(\mathbb{A}))) = 1$.

Proof. By induction on the structure of terms. □

Termination is hardly surprising because we do not have any recursive constructions. Probabilistic recursion is interesting, *e.g.* the program $(\mathsf{while}\,(\mathsf{sample}(\mathit{bern}(0.5)))\, \mathsf{do}\, \mathsf{skip})$ almost surely terminates. But we omit recursive constructs for now, because our semantic model does not yet handle higher-order recursion, and probabilistic while-languages are already well-understood (*e.g.* [17]). (See also the discussion about domain theory in §8.)

5.3 Soundness

For soundness, extend the denotational semantics to configurations:

- define $s_{\mathsf{d}} \colon \mathrm{Con}_{\mathsf{d}}(\mathbb{A}) \to [\![\mathbb{A}]\!]$ by $\langle \Gamma, t, \gamma\rangle \mapsto [\![t]\!](\gamma)$;
- define $s_{\mathsf{p}} : \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) \times \Sigma_{\mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!]} \to [0, 1]$ similarly by $(\langle \Gamma, t, \gamma\rangle, U) \mapsto [\![t]\!](\gamma)(U)$. We may also use this stochastic relation as a measurable function $s_{\mathsf{p}} \colon \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) \to P(\mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!])$;
- define $s_{V\mathsf{p}} : \mathrm{ConV}_{\mathsf{p}}(\mathbb{A}) \to [\![\mathbb{A}]\!]$ by $\langle \Gamma, \mathsf{return}(v), \gamma\rangle \mapsto [\![v]\!](\gamma)$. Note that in this first-order language, $s_{V\mathsf{p}}$ is a surjection which equates two value configurations iff they are related by weakening, contraction or exchange of variables.

Assumption 5.7. Throughout this section we assume that the normalisation function ν on configurations (7) is perfect, *i.e.* it corresponds to ι, the semantic normalisation function (2):

$$\nu_{\mathbb{A}}(\langle \Gamma, t, \gamma\rangle) = \iota_{[\![\mathbb{A}]\!]}(s_{\mathsf{p}}(\langle \Gamma, t, \gamma\rangle)).$$

Lemma 5.8 (Context extension)**.** *Let* $\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\}$*. Suppose that* $\langle \Gamma, t, \gamma\rangle$ *and* $\langle (\Gamma, \Gamma'), t, (\gamma, \gamma')\rangle$ *are configurations in* $\mathrm{Con}_{\mathsf{d}}(\mathbb{A})$*. Then* $s_{\mathsf{z}}(\langle \Gamma, t, \gamma\rangle) = s_{\mathsf{z}}\langle (\Gamma, \Gamma'), t, (\gamma, \gamma')\rangle$.

Proposition 5.9 (Soundness)**.** *The following diagrams commute in the category of measurable spaces and stochastic relations.*

$$\begin{array}{ccc} \mathrm{ConN}_{\mathsf{d}}(\mathbb{A}) & \xrightarrow{s_{\mathsf{d}}} & [\![\mathbb{A}]\!] \\ \downarrow{\scriptstyle reduction} & \nearrow{\scriptstyle s_{\mathsf{d}}} & \\ \mathrm{Con}_{\mathsf{d}}(\mathbb{A}) & & \end{array} \qquad \begin{array}{ccc} \mathrm{ConN}_{\mathsf{p}}(\mathbb{A}) & \xrightarrow{s_{\mathsf{p}}} & \mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!] \\ \downarrow{\scriptstyle reduction} & & \uparrow{\scriptstyle multiplication} \\ \mathbb{R}_{\geq 0} \times \mathrm{Con}_{\mathsf{p}}(\mathbb{A}) & \xrightarrow[\mathrm{id} \times s_{\mathsf{p}}]{} & \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times [\![\mathbb{A}]\!] \end{array}$$

Proof. By induction on the structure of syntax. The inductive steps with evaluation contexts use the extension Lemma 5.8, which applies by the invariant Lemma 5.2. □

Adequacy The denotational semantics is adequate, in the sense:

$$[\![t]\!](*) = P(\mathbb{R}_{\geq 0} \times s_{V\mathsf{p}})\big(\Pr(\langle \emptyset, t, * \rangle \Downarrow (-))\big) \quad \text{for all} \vDash_{\mathsf{p}} t \colon \mathbb{A}.$$

That is, the denotation $[\![t]\!](*)$ is nothing but pushing forward the probability measure $\Pr(\langle \emptyset, t, * \rangle \Downarrow (-))$ from the operational semantics along the function $s_{V\mathsf{p}}$. This adequacy condition holds because Proposition 5.9 ensures that

$$\Big(\sum_{k \leq n} \Pr(\langle \emptyset, t, * \rangle \Downarrow^k \{(r, C) \mid (r, s_{V\mathsf{p}}(C)) \in U\})\Big) \leq [\![t]\!](*)(U)$$

for all n and U, and Proposition 5.6 then guarantees that the left-hand side of this inequality converges to the right-hand side as n tends to infinity.

6. A higher-order language

This short section extends the first-order language with functions and thunks [18], allowing variables to stand for program fragments. In other words, 'programs are first-class citizens'.

Types Extend the grammar for types with two new constructors.

$$\mathbb{A}, \mathbb{B} \ ::= \ \mathbb{R} \mid \mathsf{P}(\mathbb{A}) \mid 1 \mid \mathbb{A} \times \mathbb{B} \mid \sum_{i \in I} \mathbb{A}_i \mid \mathbb{A} \Rightarrow \mathbb{B} \mid \mathsf{T}(\mathbb{A})$$

Informally, $\mathbb{A} \Rightarrow \mathbb{B}$ contains deterministic functions, and $\mathsf{T}(\mathbb{A})$ contains thunked (*i.e.* suspended) probabilistic programs, so that $A \Rightarrow \mathsf{T}(\mathbb{B})$ contains probabilistic functions. A type is *measurable* if it does not involve $\Rightarrow$ or T, *i.e.* if it is in the grammar of Section 3.

Terms Extend the term language with the following rules. First, the usual abstraction and application of deterministic functions:

$$\frac{\Gamma, x \colon \mathbb{A} \vDash_{\mathsf{d}} t \colon \mathbb{B}}{\Gamma \vDash_{\mathsf{d}} \lambda x.\, t \colon \mathbb{A} \Rightarrow \mathbb{B}} \qquad \frac{\Gamma \vDash_{\mathsf{d}} t \colon \mathbb{A} \Rightarrow \mathbb{B} \quad \Gamma \vDash_{\mathsf{d}} u \colon \mathbb{A}}{\Gamma \vDash_{\mathsf{d}} t\, u \colon \mathbb{B}}$$

Second, we have syntax for thunking and forcing (*e.g.* [18, 22, 25]).

$$\frac{\Gamma \vDash_{\mathsf{p}} t \colon \mathbb{A}}{\Gamma \vDash_{\mathsf{d}} \mathsf{thunk}(t) \colon \mathsf{T}(\mathbb{A})} \qquad \frac{\Gamma \vDash_{\mathsf{d}} t \colon \mathsf{T}(\mathbb{A})}{\Gamma \vDash_{\mathsf{p}} \mathsf{force}(t) \colon \mathbb{A}}$$

All the rules from Section 3 are also still in force, except that for rule (1) to still make sense, we restrict it to only include constant terms for measurable functions $f \colon [\![\mathbb{A}]\!] \to [\![\mathbb{B}]\!]$ between measurable types $\mathbb{A}$ and $\mathbb{B}$.

Examples One reason for higher types is to support code structuring. The separate function types and thunk types allow us to be flexible about calling conventions. For example, sampling can be reified as the ground term

$$\vDash_{\mathsf{d}} \lambda x.\, \mathsf{thunk}(\mathsf{sample}(x)) \colon \mathsf{P}(\mathbb{A}) \Rightarrow \mathsf{T}(\mathbb{A}),$$

which takes a probability measure and returns a suspended program that will sample from it. On the other hand, to reify the normalization construction, we use a different calling convention.

$$\vDash_{\mathsf{d}} \lambda x.\, \mathsf{norm}(\mathsf{force}(x)) \colon \mathsf{T}(\mathbb{A}) \Rightarrow \mathbb{R} \times \mathsf{P}(\mathbb{A}) + 1 + 1$$

This function takes a suspended probabilistic program and returns the result of normalizing it.

Example: higher-order expectation Higher types also allow us to consider probability distributions over programs. For an example, consider this term.

$$\begin{aligned} E_h \ \stackrel{\text{def}}{=}\ & \lambda(d, f) \colon \mathsf{T}(\mathbb{A}) \times (\mathbb{A} \Rightarrow \mathbb{R}). \\ & \quad \mathsf{case}\,(\mathsf{norm}(\mathsf{let}\ a = \mathsf{force}(d)\ \mathsf{in}\ \mathsf{score}(f(a))))\ \mathsf{of} \\ & \qquad (0, (e, y)) \Rightarrow e \\ & \qquad \mid (1, *) \Rightarrow 0.0 \ \mid (2, *) \Rightarrow 0.0 \end{aligned}$$

It has type $\mathsf{T}(\mathbb{A}) \times (\mathbb{A} \Rightarrow \mathbb{R}) \Rightarrow \mathbb{R}$. Intuitively, given a thunked probabilistic term t and a function f that is nonnegative, E_h treats t as a probability distribution on $\mathbb{A}$, and computes the expectation of f on this distribution. Notice that $\mathbb{A}$ can be a higher type, so E_h generalises the usual notion of expectation, which has not been defined for higher types because the category of measurable spaces is not Cartesian closed.

7. Higher-order operational semantics

This section considers operational semantics for the higher-order extension of the language. In an operational intuition, $\mathsf{force}(t)$ forces a suspended computation t to run. For example,

$$\vDash_{\mathsf{d}} \mathsf{thunk}(\mathsf{sample}(gauss(0.0, 1.0))) \colon \mathsf{T}(\mathbb{R})$$

is a suspended computation that, when forced, will sample the normal distribution.

Assumption 7.1. From the operational perspective it is unclear how to deal with sampling from a distribution over functions. For this reason, in this section, we only allow the type $\mathsf{P}(\mathbb{A})$ when $\mathbb{A}$ is a measurable type. We still allow probabilistic terms to have higher-order types, and we still allow $\mathsf{T}(\mathbb{A})$ where $\mathbb{A}$ is higher-order.

7.1 Reduction

We now extend the operational semantics from Section 5 with higher types. Values (4) are extended as follows.

$$v \ ::= \ \ldots \mid \lambda x.t \mid \mathsf{thunk}(t)$$

Evaluation contexts (6) are extended as follows.

$$\mathcal{C}[-] \ ::= \ \ldots \mid \mathcal{C}[-]\, t \mid v\, \mathcal{C}[-] \qquad \mathcal{E}[-] \ ::= \ \ldots \mid \mathcal{D}[\mathsf{force}[-]]$$

There are two additional redexes: $(\lambda x.t)\, v$ and $\mathsf{force}(\mathsf{thunk}(t))$. The deterministic transition relation is extended with this β-rule:

$$\langle \Gamma, (\lambda x.t)\, v, \gamma \rangle \longrightarrow \langle \Gamma, t[v/x], \gamma \rangle.$$

Extend the probabilistic transition relation with the following rules.

$$\begin{aligned} &\Pr\big(\langle \Gamma, \mathsf{force}(\mathsf{thunk}(t)), \gamma \rangle \longrightarrow (\Gamma, t)\big) = 1 \\ &\Pr\big(\langle \Gamma, \mathsf{force}(\mathsf{thunk}(t)), \gamma \rangle \longrightarrow_{(\Gamma, t)} U\big) = [\gamma \in U] \\ &\mathrm{Sc}(\langle \Gamma, \mathsf{force}(\mathsf{thunk}(t)), \gamma \rangle) = 1 \end{aligned}$$

7.2 Termination

The evaluation relations for deterministic and probabilistic configurations of the higher-order language are defined as in Subsection 5.2. The resulting rewriting system still terminates, even though configurations may now include higher-order terms.

Proposition 7.2 (Termination)**.** *Evaluation of deterministic terms is a function:* $\forall C. \exists D. C \Downarrow D$. *Evaluation of probabilistic terms is a stochastic relation:* $\forall C. \Pr(C \Downarrow (\mathbb{R}_{\geq 0} \times \mathrm{ConV}_{\mathsf{p}}(\mathbb{A}))) = 1$.

Proof. We sketch an invariant of higher-order terms that implies the termination property, formulated as unary logical relations via sets

$$\begin{aligned} \mathrm{R}(\Gamma \vDash_{\mathsf{z}} \mathbb{A}) &\subseteq \{t \mid \Gamma \vDash_{\mathsf{z}} t \colon \mathbb{A}\}, \\ \mathrm{R}_{\mathrm{v}}(\Gamma \vDash_{\mathsf{z}} \mathbb{A}) &\subseteq \{t \mid \Gamma \vDash_{\mathsf{z}} t \colon \mathbb{A} \wedge t \text{ a } \mathsf{z}\text{-value}\}, \end{aligned}$$

for each canonical context Γ, type $\mathbb{A}$, and $\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\}$, defined by:

$$\mathrm{R}(\Gamma \vDash_{\mathsf{d}} \mathbb{A}) = \{t \mid \forall\gamma.\, \langle \Gamma, t, \gamma\rangle \Downarrow \langle \Gamma', t', \gamma'\rangle \wedge t' \in \mathrm{R_v}(\Gamma' \vDash_{\mathsf{d}} \mathbb{A})\}$$

$$\mathrm{R}(\Gamma \vDash_{\mathsf{p}} \mathbb{A}) = \{t \mid \forall\gamma.\, \Pr(\langle \Gamma, t, \gamma\rangle \Downarrow (\mathbb{R}_{\geq 0} \times \textstyle\sum_{\Gamma'} \mathrm{R_v}(\Gamma' \vDash_{\mathsf{p}} \mathbb{A}) \times [\![\Gamma']\!])) = 1\}$$

$$\begin{aligned}
\mathrm{R_v}(\Gamma \vDash_{\mathsf{p}} \mathbb{A}) &= \{\mathsf{return}(v) \mid v \in \mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A})\}\\
\mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A}) &= \{x \mid (x \colon \mathbb{A}) \in \Gamma\} \qquad \text{for } \mathbb{A} \text{ indecomposable}\\
\mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} 1) &= \{*\}\\
\mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A}_1 \times \mathbb{A}_2) &= \{(v_1, v_2) \mid \forall j.\, v_j \in \mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A}_j)\}\\
\mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \textstyle\sum \mathbb{A}_i) &= \{(i, v) \mid v \in \mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A}_i)\}\\
\mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} T(\mathbb{A})) &= \{\mathsf{thunk}(t) \mid t \in \mathrm{R}(\Gamma \vDash_{\mathsf{p}} \mathbb{A})\}\\
\mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A} \Rightarrow \mathbb{B}) &= \{\lambda x.t \mid \forall \Gamma' \supseteq \Gamma, u \in \mathrm{R_v}(\Gamma' \vDash_{\mathsf{d}} \mathbb{A}).\ t[u/x] \in \mathrm{R}(\Gamma' \vDash_{\mathsf{d}} \mathbb{B})\}
\end{aligned}$$

Induction on the structure of a term $\Gamma, x_1 \colon \mathbb{A}_1, \ldots x_n \colon \mathbb{A}_n \vDash_{\mathsf{z}} t \colon \mathbb{B}$ for $\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\}$ now proves that $v_i \in \mathrm{R_v}(\Gamma \vDash_{\mathsf{d}} \mathbb{A}_i)$ for $i = 1, \ldots, n$ implies $t[\vec{v}/\vec{x}] \in \mathrm{R}(\Gamma \vDash_{\mathsf{z}} \mathbb{B})$. □

8. Higher-order denotational semantics

This section gives denotational semantics for the higher-order language, without using Assumption 7.1. We are to interpret the new constructs $\mathsf{T}(\mathbb{A})$, thunk, and force. We will interpret probabilistic judgements as Kleisli morphisms $[\![\Gamma]\!] \to T([\![A]\!])$ for a certain monad T, and set $[\![\mathsf{T}(A)]\!] \stackrel{\text{def}}{=} T([\![A]\!])$, so thunk and force embody the correspondence of maps $[\![\Gamma]\!] \to T([\![A]\!])$ and $[\![\Gamma]\!] \to [\![\mathsf{T}(A)]\!]$.

On which category can the monad T live? Interpreting λ-abstraction and application needs a natural 'currying' bijection between morphisms $[\![\Gamma]\!] \times \mathbb{R} \to \mathbb{R}$ and morphisms $[\![\Gamma]\!] \to [\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$. But measurable functions cannot do this: it is known that no measurable space $[\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$ can support such a bijection [3].

We resolve the problem of function spaces by embedding the category of measurable spaces in a larger one, where currying is possible, and that still has the structure to interpret the first order language as before. As the larger category we will take a category of functors $\mathbf{Meas}^{\mathrm{op}} \to \mathbf{Set}$ from a category $\mathbf{Meas}$ of measurable spaces and measurable functions to the category $\mathbf{Set}$ of sets and functions. This idea arises from two traditions. First, we can think of a variable of type $\mathbb{R}$ as a read-only memory cell, as in the operational semantics, and functor categories have long been used to model local memory (*e.g.* [23]). Second, the standard construction for building a Cartesian closed category out of a distributive one is based on functor categories (*e.g.* [28]).

Other models of higher-order programs Semantics of higher-order languages with *discrete* probability are understood well. For terminating programs, there are set-theoretic models based on a distributions monad, and for full recursion one can use probabilistic powerdomains [14] or coherence spaces [9]. It is also plausible one could model continuous distributions in domain theory, since it supports computable real analysis (*e.g.* [8]); this could be interesting because computability is subtle for probabilistic programming (*e.g.* [1]). Nonetheless, we contend it is often helpful to abstract away computability issues when studying probabilistic programming languages, to have access to standard theorems of probability theory to justify program transformations.

8.1 Semantic model

Fix a category $\mathbf{Meas}$ of measurable spaces and measurable functions that is essentially small but large enough for the purposes of Section 4. For example, $\mathbf{Meas}$ could be the category of *standard Borel spaces* [4, 32]: one can show that $[\![\mathbb{A}]\!]$ is standard Borel by induction on $\mathbb{A}$, and the class of all standard Borel spaces is countable up to measurable isomorphism.

In Section 4 we interpreted first-order types $\mathbb{A}$ as measurable spaces $[\![\mathbb{A}]\!]$. We will interpret higher-order types $\mathbb{A}$ as functors $(\!|\mathbb{A}|\!) \colon \mathbf{Meas}^{\mathrm{op}} \to \mathbf{Set}$. Informally, when $\mathbb{A}$ is a first-order type and Γ is a first-order context, we will have $(\!|\mathbb{A}|\!)([\![\Gamma]\!]) \cong \mathbf{Meas}([\![\Gamma]\!], [\![\mathbb{A}]\!]) \approx \{t \mid \Gamma \vDash_{\mathsf{d}} t \colon \mathbb{A}\}$. For a second order type $(\mathbb{A} \Rightarrow \mathbb{B})$, we will have

$$(\!|\mathbb{A} \Rightarrow \mathbb{B}|\!)([\![\Gamma]\!]) \cong \mathbf{Meas}([\![\Gamma]\!] \times [\![\mathbb{A}]\!], [\![\mathbb{B}]\!]) \approx \{t \mid \Gamma, x : \mathbb{A} \vDash_{\mathsf{d}} t \colon \mathbb{B}\}$$

so that β/η equality is built in. To put it another way, $(\!|\mathbb{A}|\!)(\mathbb{R}^n)$ models terms of type $\mathbb{A}$ having n read-only real-valued memory cells.

Lemma 8.1. *For a small category $\mathbf{C}$ with countable sums, consider the category $\overline{\mathbf{C}}$ of countable-product-preserving functors $\mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$, and natural transformations between them.*

- *$\overline{\mathbf{C}}$ has all colimits;*
- *$\overline{\mathbf{C}}$ is Cartesian closed if $\mathbf{C}$ has products that distribute over sums;*
- *There is a full and faithful embedding $\mathbf{y} : \mathbf{C} \to \overline{\mathbf{C}}$, given by $\mathbf{y}(c) \stackrel{\text{def}}{=} \mathbf{C}(-, c)$, which preserves limits and countable sums.*

Proof. See *e.g.* [28, §7], or [15, Theorems 5.56 and 6.25]. The embedding $\mathbf{y}$ is called the *Yoneda embedding*. □

For a simple example, consider the category $\mathbf{CSet}$ of countable sets and functions. It has countable sums and finite products, but is not Cartesian closed. Because every countable set is a countable sum of singletons, the category $\overline{\mathbf{CSet}}$ is equivalent to $\mathbf{Set}$.

Our semantics for the higher-order language will take place in the category $\overline{\mathbf{Meas}}$. Note that products in $\overline{\mathbf{Meas}}$ are pointwise, *e.g.* $(F \times G)(X) = F(X) \times G(X)$ for all $F, G \in \overline{\mathbf{Meas}}$ and all $X \in \mathbf{Meas}$, but sums are not pointwise, *e.g.* $(1 + 1) \in \overline{\mathbf{Meas}}$ is the functor that assigns a measurable space X to the set of its measurable subsets. This is essential for $\mathbf{y}$ to preserve sums.

Distribution types We have to interpret distribution types $\mathsf{P}(\mathbb{A})$ in our functor category $\overline{\mathbf{Meas}}$. How can we interpret a probability distribution on the type $\mathbb{R} \Rightarrow \mathbb{R}$? We can answer this pragmatically, without putting σ-algebra structure on the set of all functions. If $[\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$ were a measurable space, a random variable valued in $[\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$ would be given by a measurable space (X, Σ_X), a probability distribution on it, and a measurable function $X \to [\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$. Despite there being no such measurable space $[\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$, we can speak of uncurried measurable functions $X \times \mathbb{R} \to \mathbb{R}$. Thus we might define a probability distribution on $[\![\mathbb{R} \Rightarrow \mathbb{R}]\!]$ to be a triple

$$\big((X, \Sigma_X),\, p \colon \Sigma_X \to [0,1],\, f \colon X \times \mathbb{R} \to \mathbb{R}\big)$$

of a measurable space (X, Σ_X) of 'codes' with a probability distribution p, and a measurable function f where we think of $f(x, r)$ as 'the function coded x evaluated at r'. These triples should be considered modulo renaming the codes. This is exactly the notion of probability distribution that arises in our functor category.

Lemma 8.2. *For a small category $\mathbf{C}$ with countable sums:*

- *any functor $F \colon \mathbf{C} \to \mathbf{C}$ extends to a functor $\overline{F} \colon \overline{\mathbf{C}} \to \overline{\mathbf{C}}$ satisfying $\overline{F} \circ \mathbf{y} \cong \mathbf{y} \circ F$, given by*
$$\overline{F}(G)(b) = \Big(\textstyle\sum_a G(a) \times \mathbf{C}\big(b, F(a)\big)\Big) \,/\!\sim$$
where the equivalence relation $\sim$ is the least one satisfying $(a', x, Fg \circ f) \sim (a, Gg(x), f)$;
- *similarly, any functor $F \colon \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ in two arguments extends to a functor $\overline{F} \colon \overline{\mathbf{C}} \times \overline{\mathbf{C}} \to \overline{\mathbf{C}}$, with $\overline{F} \circ (\mathbf{y} \times \mathbf{y}) \cong \mathbf{y} \circ F$:*
$$\overline{F}(G, H)(c) = \Big(\textstyle\sum_{a,b} G(a) \times H(b) \times \mathbf{C}\big(c, F(a, b)\big)\Big) \,/\!\sim$$

- *any natural transformation* $\alpha\colon F \to G$ *between functors* $F, G\colon \mathbf{C} \to \mathbf{C}$ *lifts to a natural transformation* $\overline{\alpha}\colon \overline{F} \to \overline{G}$, *and similarly for functors* $\mathbf{C} \times \mathbf{C} \to \mathbf{C}$;
- *and this is functorial, i.e.* $\overline{G \circ F} \cong \overline{G} \circ \overline{F}$ *and* $\overline{\beta \circ \alpha} = \overline{\beta} \circ \overline{\alpha}$.

Proof. $\overline{F}(G)$ is the left *Kan extension* of G along F, see *e.g.* [15]. Direct calculation shows $\overline{F}(G)$ preserves products if G does. □

Thus the commutative monads P and $T = P(\mathbb{R}_{\geq 0} \times (-))$ on $\mathbf{Meas}$ lift to commutative monads $\overline{P}$ and $\overline{T} \cong \overline{P}(\mathbf{y}\mathbb{R}_{\geq 0} \times (-))$ on $\overline{\mathbf{Meas}}$. The latter monad captures the informal importance-sampling semantics advocated by the designers of Anglican [34].

8.2 Conservativity

We interpret the types of the higher order language as objects in $\overline{\mathbf{Meas}}$ using its categorical structure.

$$(\!|\textstyle\sum_{i \in I} \mathbb{A}_i|\!) \stackrel{\text{def}}{=} \sum_{i \in I} (\!|\mathbb{A}_i|\!) \qquad (\!|\mathbb{R}|\!) \stackrel{\text{def}}{=} \mathbf{y}\mathbb{R} \qquad (\!|\mathsf{P}(\mathbb{A})|\!) \stackrel{\text{def}}{=} \overline{P}((\!|\mathbb{A}|\!))$$
$$(\!|\mathbb{A} \times \mathbb{B}|\!) \stackrel{\text{def}}{=} (\!|\mathbb{A}|\!) \times (\!|\mathbb{B}|\!) \qquad (\!|1|\!) \stackrel{\text{def}}{=} 1 \qquad (\!|\mathsf{T}(\mathbb{A})|\!) \stackrel{\text{def}}{=} \overline{T}((\!|\mathbb{A}|\!))$$
$$(\!|\mathbb{A} \Rightarrow \mathbb{B}|\!) \stackrel{\text{def}}{=} (\!|\mathbb{A}|\!) \Rightarrow (\!|\mathbb{B}|\!)$$

All first-order types $\mathbb{A}$ have a natural isomorphism $(\!|\mathbb{A}|\!) \cong \mathbf{y}[\![\mathbb{A}]\!]$. We extend this interpretation to contexts.

$$(\!|x_1 : \mathbb{A}_1, \ldots, x_n : \mathbb{A}_n|\!) \stackrel{\text{def}}{=} \prod_{i=1}^{n} (\!|\mathbb{A}_i|\!)$$

Deterministic terms $\Gamma \vdash_{\mathsf{d}} t \colon \mathbb{A}$ are interpreted as natural transformations $(\!|\Gamma|\!) \to (\!|\mathbb{A}|\!)$ in $\overline{\mathbf{Meas}}$, and probabilistic terms $\Gamma \vdash_{\mathsf{p}} t \colon \mathbb{A}$ as natural transformations $(\!|\Gamma|\!) \to \overline{T}(\!|\mathbb{A}|\!)$, by induction on the structure of terms as in Section 4. Application and λ-abstraction are interpreted as usual in Cartesian closed categories [27]. Thunk and force are trivial from the perspective of the denotational semantics, because $(\!|\mathsf{T}\mathbb{A}|\!) = T(\!|\mathbb{A}|\!)$. To interpret $\mathsf{norm}(t)$, use Lemma 8.2 to extend the normalisation functions $T(X) \to \mathbb{R}_{\geq 0} \times P(X) + 1 + 1$ between measurable spaces (2) to natural transformations $\overline{T}(F) \to \mathbf{y}(\mathbb{R}_{\geq 0}) \times \overline{P}(F) + 1 + 1$. This interpretation conserves the first-order semantics of Section 4.

Proposition 8.3. *For* $\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\}$, *and first-order* Γ *and* $\mathbb{A}$:

- $[\![t]\!] = [\![u]\!]$ *if and only if* $(\!|t|\!) = (\!|u|\!)$ *for first-order* $\Gamma \vdash_{\mathsf{z}} t, u \colon \mathbb{A}$;
- *every term* $\Gamma \vdash_{\mathsf{z}} t \colon \mathbb{A}$ *has* $(\!|t|\!) = (\!|u|\!)$ *for a first-order* $\Gamma \vdash_{\mathsf{z}} u \colon \mathbb{A}$.

Proof. We treat $\mathsf{z} = \mathsf{d}$; the other case is similar. By induction on the structure of terms, $(\!|t|\!) = \mathbf{y}[\![t]\!] \colon (\!|\Gamma|\!) \to (\!|\mathbb{A}|\!)$. The first point follows from faithfulness of $\mathbf{y}$; the second from fullness and (1). □

One interesting corollary is that the interpretation of a term of first-order type is always a measurable function, even if the term involves thunking, λ-abstraction and application. This corollary gives a partial answer to a question by Park *et. al.* on the measurability of all λ-definable ground-type terms in probabilistic programs [25] (partial because our language does not include recursion).

8.3 Soundness

The same recipe as in Section 5.3 will show that the higher-order denotational semantics is sound and adequate with respect to the higher-order operational semantics. This needs Assumption 7.1.

A subtle point is that configuration spaces (5) involve uncountable sums: the set of terms of a given type is uncountable, but $\mathbf{y}$ only preserves countable sums. This is not really a problem because only countably many terms are reachable from a given program.

Definition 8.4. For a type $\mathbb{A}$, the binary *reachability* relation $\rightsquigarrow^*_{\mathsf{d}}$ on $\{(\Gamma, t) \mid \Gamma \vdash_{\mathsf{d}} t \colon \mathbb{A} \wedge \Gamma \,\text{canonical}\}$ is the least reflexive and transitive relation with $(\Gamma, t) \rightsquigarrow^*_{\mathsf{d}} (\Gamma', u)$ if $\langle \Gamma, t, \gamma \rangle \longrightarrow \langle \Gamma', u, \gamma' \rangle$ for $\gamma \in [\![\Gamma]\!]$, $\gamma' \in [\![\Gamma']\!]$. Similarly, $\rightsquigarrow^*_{\mathsf{p}}$ is the least reflexive and transitive relation on $\{(\Gamma, t) \mid \Gamma \vdash_{\mathsf{p}} t \colon \mathbb{A} \wedge \Gamma \,\text{canonical}\}$ with $(\Gamma, t) \rightsquigarrow^*_{\mathsf{p}} (\Gamma', u)$ if $\Pr\big(\langle \Gamma, t, \gamma \rangle \longrightarrow (\Gamma', u)\big) \neq 0$ for $\gamma \in [\![\Gamma]\!]$.

Proposition 8.5. *Let* $\mathsf{z} \in \{\mathsf{d}, \mathsf{p}\}$. *For any closed term* $\vdash_{\mathsf{z}} t \colon \mathbb{A}$, *the set of reachable terms* $\{(\Gamma, u) \mid (\emptyset, t) \rightsquigarrow^*_{\mathsf{z}} (\Gamma, u)\}$ *is countable.*

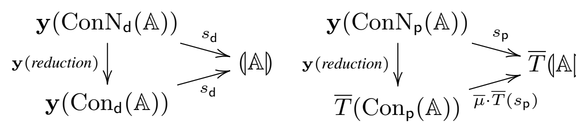
Proof. One-step reachability is countable by induction on terms. Since all programs terminate by Proposition 7.2, the reachable terms form a countably branching well-founded tree. □

We may thus restrict to the configurations built from a countable set $\mathcal{U}$ of terms that is closed under subterms and reachability. Extend the denotational semantics in $\overline{\mathbf{Meas}}$ to configurations by defining $s_{\mathsf{d}} \colon \mathbf{y}(\mathrm{Con}_{\mathsf{d}}(\mathbb{A})) \to (\!|\mathbb{A}|\!)$, $s_{\mathsf{p}} \colon \mathbf{y}(\mathrm{Con}_{\mathsf{p}}(\mathbb{A})) \to \overline{T}(\!|\mathbb{A}|\!)$, and $s_{V\mathsf{p}} \colon \mathbf{y}(\mathrm{ConV}_{\mathsf{p}}(\mathbb{A})) \to \overline{T}(\!|\mathbb{A}|\!)$; use the isomorphisms

$$\mathbf{y}(\mathrm{Con}_{\mathsf{z}}(\mathbb{A})) \cong \sum_{(\Gamma \vdash_{\mathsf{z}} t : \mathbb{A}) \in \mathcal{U}} (\!|\Gamma|\!)$$

to define $s_{\mathsf{d}}, s_{\mathsf{p}}, s_{V\mathsf{p}}$ by copairing the interpretation morphisms $(\!|\Gamma \vdash_{\mathsf{d}} t \colon \mathbb{A}|\!) \colon (\!|\Gamma|\!) \to (\!|\mathbb{A}|\!)$ and $(\!|\Gamma \vdash_{\mathsf{p}} u \colon \mathbb{A}|\!) \colon (\!|\Gamma|\!) \to \overline{T}((\!|\mathbb{A}|\!))$.

Proposition 8.6 (Soundness)**.** *The following diagrams commute.*



Adequacy It follows that the higher denotational semantics remains adequate, in the sense that for all probabilistic terms $\vdash_{\mathsf{p}} t \colon \mathbb{A}$,

$$(\!|t|\!)_1(*) = (\overline{T}(s_{V\mathsf{p}}))_1\big(\Pr(\langle \emptyset, t, * \rangle \Downarrow (-))\big).$$

Adequacy is usually only stated for first-order types. At first-order types $\mathbb{A}$ the function $s_{V\mathsf{p}}$ does very little, since global elements of $(\!|\mathbb{A}|\!)$ correspond bijectively with value configurations modulo weakening, contraction and exchange in the context. At higher types, the corollary still holds, but $s_{V\mathsf{p}}$ is not so trivial because we do not reduce under thunk or λ.

9. Continuous densities

Several higher-order probabilistic programming languages (such as Anglican) provide constructs to build probability distributions with *continuous densities*. The probability densities are given with respect to well-understood base measures, such as Lebesgue measure for $\mathbb{R}$ and counting measures on countable sets.[2] Our language easily extends to accommodate such distributions. Just add a collection *density types* to the syntax.

$$\mathbb{D} ::= \mathbb{R} \mid \mathsf{bool} \mid \mathbb{N} \mid 1 \mid \mathbb{D} \times \mathbb{D} \qquad \mathbb{A} ::= \cdots \mid \mathsf{D}(\mathbb{D})$$

The $\mathbb{D}$ in this grammar denotes a measurable space that: (i) carries a separable metrisable topology that generates the σ-algebra; and that (ii) comes with a chosen σ-finite measure μ. An example is $\mathbb{R}$ with its usual Euclidean topology and the Lebesgue measure.

The type $\mathsf{D}(\mathbb{D})$ denotes a measurable space $[\![\mathsf{D}(\mathbb{D})]\!]$ of continuous functions $f \colon [\![\mathbb{D}]\!] \to \mathbb{R}_{\geq 0}$ with $\int_X f \,\mathrm{d}\mu = 1$, where continuity and integration are taken with respect to the topology and the base

[2] A *measure* on a measurable space (X, Σ) is a function $\mu \colon \Sigma \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that $\mu(\bigcup_{i \in \mathbb{N}} U_i) = \sum_{i \in \mathbb{N}} \mu(U_i)$ for a disjoint sequence $U_1, U_2, \ldots$ of measurable sets. It is *σ-finite* when $X = \bigcup_{i \in \mathbb{N}} U_i$ with $U_i \in \Sigma$ and $\mu(U_i) < \infty$. The *Lebesgue measure* μ_L maps an interval to its size. The *counting measure* μ_C maps finite measurable sets to their cardinality, and infinite sets to ∞. A *density* for μ is a measurable function $f \colon X \to \mathbb{R}$ satisfying $\mu(U) = \int_U f \,\mathrm{d}\mu$.

measure μ of $[\![\mathbb{D}]\!]$. The σ-algebra of $[\![\mathsf{D}(\mathbb{D})]\!]$ is the least one making $\{f \mid f(x) \leq r\}$ measurable for all $(x, r) \in [\![\mathbb{D}]\!] \times \mathbb{R}_{\geq 0}$.

Insisting on continuity in the definition of $\mathsf{D}(\mathbb{D})$ ensures that

$$ev\colon [\![\mathsf{D}(\mathbb{D})]\!] \times [\![\mathbb{D}]\!] \to \mathbb{R}_{\geq 0} \qquad ev(f, x) = f(x)$$

is a measurable function. This would have been impossible if we had included noncontinuous densities, regardless of the σ-algebra [3]. Because of the measurability of ev, a common way to impose a soft constraint in probabilistic programming languages (such as Anglican) can be encoded in our first-order language as

$$\mathsf{score}(ev(f, x)),$$

where the datum x is observed with a probability distribution with density f. Thus the categorical machinery used to interpret higher-order functions is not needed for such soft constraints.

There is a limit to this use of continuity: probability measures produced by $\mathsf{norm}(t)$ need not have continuous density. For example, $\mathsf{norm}(\mathsf{return}(42.0))$ produces a discontinuous Dirac measure.

Density types can be incorporated into the higher order language straightforwardly. The only subtlety is that denotational semantics now needs the base category to contain $[\![\mathsf{D}(\mathbb{D})]\!]$.

Probability densities are often used by *importance samplers*. Let $dist_{\mathbb{D}}$ be the function that converts continuous densities f on $[\![\mathbb{D}]\!]$ to probability measures:

$$dist_{\mathbb{D}}(f)(U) \stackrel{\text{def}}{=} \int_U f \, \mathrm{d}\mu,$$

where μ is the base measure of δ. This function is measurable, so it is a constant term in our language. The importance sampler generates samples of $f \in [\![\mathsf{D}(\mathbb{R})]\!]$ by first sampling from a proposal distribution g where sampling is easy, and then normalizing those samples x from g according to their importance weight $f(x)/g(x)$. The following equivalence in our semantics expresses the correctness of this sampling strategy where we use the standard normal distribution as a proposal distribution.

$$[\![\mathsf{norm}(\mathsf{sample}(dist_{\mathbb{R}}(f)))]\!]$$
$$= \left[\!\!\left[\begin{array}{l}\mathsf{norm}(\mathsf{let}\ x = \mathsf{sample}(gauss(0.0, 1.0))\ \mathsf{in} \\ \qquad \mathsf{score}\big(f(x)/density\_gauss(x, (0.0, 1.0))\big); \mathsf{return}(x))\end{array}\right]\!\!\right]$$

10. Conclusion and future work

We have defined a metalanguage for higher-order probabilistic programs with continuous distributions and soft constraints, and presented operational and denotational semantics, together with useful program equations justified by the semantics. One interesting next step is to use these tools to study other old or new language features and concepts (such as recursion, function memoisation [30], measure-zero conditioning [5], disintegration [2, 31], and exchangeability [10, 20, 34]) that have been experimented with in the context of probabilistic programming. In particular, our tools may reveal new insights into how these features interact with higher-order functions. Another future direction is to formulate and prove the correctness of inference algorithms, especially those based on Monte Carlo simulation. In particular, it would be interesting to see whether our semantic techniques can enable some insights from [13] to be extended to higher-order languages.

References

[1] N. L. Ackerman, C. E. Freer, and D. M. Roy. Noncomputable conditional distributions. In *LiCS*, 2011.

[2] N. L. Ackerman, C. E. Freer, and D. M. Roy. On computability and disintegration, 2015. URL `http://arxiv.org/abs/1509.02992`.

[3] R. J. Aumann. Borel structures for function spaces. *Illinois Journal of Mathematics*, 5:614–630, 1961.

[4] S. K. Berberian. *Borel spaces*. World Scientific, 1988.

[5] J. Borgström, A. D. Gordon, M. Greenberg, J. Margetson, and J. van Gael. Measure transformer semantics for Bayesian machine learning. *LMCS*, 9(3):11, 2013.

[6] E.-E. Doberkat. *Stochastic Relations: Foundations for Markov Transition Systems*. Chapman & Hall, 2007.

[7] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[8] A. Edalat and M. H. Escardó. Integration in Real PCF. *Inf. Comput.* , 160:128–166, 2000.

[9] T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *POPL*, 2014.

[10] C. E. Freer and D. M. Roy. Computable de Finetti measures. *Ann. Pure Appl. Logic*, 163(5):530–546, 2012.

[11] M. Giry. A categorical approach to probability theory. *Categorical Aspects of Topology and Analysis*, 915:68–85, 1982.

[12] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *UAI*, 2008.

[13] C. Hur, A. V. Nori, S. K. Rajamani, and S. Samuel. A provably correct sampler for probabilistic programs. In *FSTTCS*, 2015.

[14] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *LiCS*, 1989.

[15] G. M. Kelly. *Basic concepts of enriched category theory*. CUP, 1982.

[16] A. Kock. Monads on symmetric monoidal closed categories. *Archiv der Mathematik*, XXI:1–10, 1970.

[17] D. Kozen. Semantics of probablistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[18] P. B. Levy. Call-by-push-value: A subsuming paradigm. In *TLCA'02*.

[19] P. B. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2), 2003.

[20] V. K. Mansinghka, D. Selsam, and Y. N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. 2014. URL `http://arxiv.org/abs/1404.0099`.

[21] T. Minka, J. Winn, J. Guiver, and D. Knowles. Infer.NET 2.4, Microsoft Research Cambridge, 2010.

[22] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1): 55–92, 1991.

[23] F. J. Oles. Type algebras, functor categories and block structure. In *Algebraic methods in semantics*. CUP, 1984.

[24] B. Paige and F. Wood. A compilation target for probabilistic programming languages. In *ICML*, 2014.

[25] S. Park, F. Pfenning, and S. Thrun. A probabilistic language based on sampling functions. *ACM TOPLAS*, 31(1):171–182, 2008.

[26] A. Patil, D. Huard, and C. J. Fonnesbeck. PyMC: Bayesian stochastic modelling in Python. *Journal of Statistical Software*, 35, 2010.

[27] A. M. Pitts. Categorical logic. In *Handbook of Logic in Computer Science*, volume 5. OUP, 2000.

[28] J. Power. Generic models for computational effects. *TCS*, 364(2): 254–269, 2006.

[29] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, 2002.

[30] D. M. Roy, V. Mansinghka, N. Goodman, and J. Tenenbaum. A stochastic programming perspective on nonparametric Bayes. In *ICML Workshop on Nonparametric Bayesian*, 2008.

[31] C.-C. Shan and N. Ramsey. Symbolic Bayesian inference by lazy partial evaluation, 2016.

[32] S. M. Srivastava. *A course on Borel sets*. Springer, 1998.

[33] Stan Development Team. Stan: A C++ library for probability and sampling, version 2.5.0, 2014. URL `http://mc-stan.org/`.

[34] F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *AISTATS*, 2014.

[35] H. Yang. Program transformation for probabilistic programs, 2015. Presentation at DALI.