Semantics of Higher-order Probabilistic Programs

Extended Abstract

Sam Staton, Univ. of Oxford
Hongseok Yang, Univ. of Oxford
Chris Heunen, Univ. of Edinburgh
Ohad Kammar, Univ. of Cambridge
Frank Wood, Univ. of Oxford

1. Introduction

Probabilistic programming is the idea of using programs for expressing probabilistic models. It aims at enabling scientists to develop sophisticated probabilistic models easily. Most probabilistic programming languages blend usual programming constructs with special probabilistic primitives, so that scientists can use both and express advanced models succinctly. Also, these languages come with generic inference algorithms, and relieve scientists of the nontrivial task of designing custom inference algorithms for their models. Several probabilistic programming languages are available by now. Some languages such as Infer.net and Stan restrict the form of programs but provide powerful inference algorithms, while others such as Church, Venture and Anglican have less-performant inference algorithms but support continuous and other advanced distributions and include powerful features of general-purpose programming languages, such as higher-order functions and Lisp-style meta-programming facilities.

Our work focuses on the latter group of probabilistic programming languages that in particular support full higher-order functions. Our goal is to answer a foundational question: *What are the formal semantics of such a language?* In this extended abstract, we will describe our current status on developing denotational semantics of a typed call-by-value higher-order probabilistic programming language, which is motivated by the Anglican language [5].

The combination of higher-order functions and continuous distributions puts a nontrivial challenge for giving denotational (or even operational) semantics that satisfies expected program equivalences and basic properties from probability theory. This challenge originates from the absence of well-behaved function spaces in the standard setting of probability theory: the evaluation function

$$\mathrm{ev}\colon \mathbb{R}^{\mathbb{R}} \times \mathbb{R} \to \mathbb{R}, \quad \mathrm{ev}(f,x) = f(x)$$

is not measurable regardless of the σ-algebra chosen for the set $\mathbb{R}^{\mathbb{R}}$ of measurable functions on $\mathbb{R}$ [1]. Because of this technical difficulty, as far as we are aware, no denotational semantics have been developed for a programming language with higher-order functions and continuous probabilistic choices. Existing semantics exclude higher-order functions or consider only discrete distributions [2, 4].

Our idea is to overcome this absence of well-behaved function spaces by embedding the standard semantic universe of probability theory $\mathbf{M}$ into a better one with more structure, $\mathsf{PSh}^{\times}(\mathbf{M})$. Informally, the guiding idea is that, since higher-order functions abstract parts of the type context, the semantic universe should take the type context seriously. We make this precise using category theory. A function space B^A in a category $\mathbf{M}$ comes with a natural bijection $\mathbf{M}(-\times A, B) \cong \mathbf{M}(-, B^A)$ ('currying'). If no such function space B^A actually exists in $\mathbf{M}$, we can use the functor $\mathbf{M}(-\times A, B)\colon \mathbf{M}^{\mathrm{op}} \to \mathbf{Set}$ in its place.

Precisely, then, we fix a small category $\mathbf{M}$ of measurable spaces, and consider the category $\mathsf{PSh}^{\times}(\mathbf{M})$ of product-preserving functors $\mathbf{M}^{\mathrm{op}} \to \mathbf{Set}$. There is a full embedding of $\mathbf{M}$ into $\mathsf{PSh}^{\times}(\mathbf{M})$, called the Yoneda functor. Furthermore, the Yoneda embedding preserves the structure of $\mathbf{M}$ that is commonly used to interpret first-order probabilistic programs: it preserves products and coproducts. The so-called Giry monad G on $\mathbf{M}$ used for interpreting probabilistic choices can be extended to a monad on $\mathsf{PSh}^{\times}(\mathbf{M})$ along the embedding. Our denotational semantics interprets call-by-value higher-order probabilistic programs using the category $\mathsf{PSh}^{\times}(\mathbf{M})$.

Although our semantics uses abstract tools from category theory, it is motivated to address practical concerns raised during the development of Anglican. In the rest of this extended abstract, we will present key aspects of our denotational semantics and connect them with these motivations.

Preliminaries We review a few basic concepts from probability theory. A *σ-algebra* on a set A is a collection $\mathcal{F}$ of subsets of A such that (i) $\mathcal{F}$ contains $\emptyset$; and (ii) it is closed under countable union and set complement. A pair of a set A and a σ-algebra $\mathcal{F}$ on A forms a *measurable space*; in such a case, subsets in $\mathcal{F}$ are called *measurable sets*. A function $f\colon (A,\mathcal{F}) \to (B,\mathcal{G})$ between measurable spaces is a *measurable function* if the inverse image of f preserves measurable sets ($f^{-1}(X) \in \mathcal{F}$ for all $X \in \mathcal{G}$).

A *measure* μ on a measurable space $(A,\mathcal{F})$ is a function from $\mathcal{F}$ to the non-negative reals or ∞ such that (i) $\mu(\emptyset) = 0$; and (ii) $\mu(\bigcup A_{n\in\mathbb{N}}) = \sum_{n\in\mathbb{N}} \mu(A_i)$ for every *disjoint countable* family $\{A_n\}_{n\in\mathbb{N}}$ in $\mathcal{F}$. When $\mu(A) = 1$, we call μ a *probability measure*. A measurable space $(A,\mathcal{F})$ with a measure μ on it forms a *measure space*.

A standard example is the measure space of reals with the Lebesgue measure, $(\mathbb{R}, \mathcal{F}_{\mathbb{R}}, \lambda)$. Here $\mathcal{F}_R$ is the smallest σ-algebra that contains all the open intervals $(r,r') \subseteq \mathbb{R}$, and λ is the unique measure that maps each open interval (r,r') to its length $r'-r$. Restricting $\mathcal{F}_{\mathbb{R}}$ and λ to a measurable subset of $\mathbb{R}$, such as the set $\mathbb{R}_{\geq 0}$ of non-negative, makes that subset into a measure space.

2. Programming Language

We use a typed variant of Anglican [5]. This is a call-by-value simply-typed lambda calculus extended with types and primitives for describing probabilistic models. The language has two kinds of types:

$$\begin{array}{rl} \text{Measure Type} & \delta ::= \texttt{bool} \mid \texttt{nat} \mid \texttt{real} \mid \texttt{unit} \mid \delta \times \delta \\ \text{General Type} & \tau ::= \delta \mid \texttt{dist}[\delta] \mid \tau \times \tau \mid \tau \to \tau \end{array}$$

Measure types specify spaces on which we can create distributions using terms in the language. Such a distribution has type $\texttt{dist}[\delta]$ for some δ, and can be manipulated by the following primitives:

$$\texttt{sample}_\delta\colon \texttt{dist}[\delta] \to \delta, \qquad \texttt{observe}_\delta\colon \texttt{dist}[\delta] \times \delta \to \texttt{unit}$$

The first primitive samples a δ-typed value from the distribution, and the second expresses an observation of a sample from the distribution; operationally, it alters the score (called importance weight) associated with each execution by multiplying the likelihood score of the observation.

The reason that the language allows distributions only over measure types is that this restriction lets us interpret $\texttt{observe}_\delta$. In our semantics, a measure type δ denotes a particular kind of measure space $(A, \mathcal{F}, \mu)$ where it is possible to say when a measurable function f from A to $\mathbb{R}_{\geq 0}$ is *continuous*. Note the presence of the default measure μ. In our language, a distribution of type $\mathtt{dist}[\delta]$ can be created only if it can be defined by altering the default measure μ of δ via some continuous function f from A to $\mathbb{R}_{\geq 0}$, called *density*. Our interpretation of $\mathtt{observe}_\delta\langle d, v\rangle$ uses the density f of the distribution d, computes the (likelihood) score $f(v)$ of the observed value v, and alters the total score of each execution by multiplying $f(v)$.

Terms of our language are mostly standard. The only exceptions are $\mathtt{sample}_\delta$ and $\mathtt{observe}_\delta$ from above and primitives for constructing distributions. We list three of these primitives below:

$$\begin{aligned}\mathtt{Normal}&\colon \mathtt{real} \times \mathtt{real} \to \mathtt{dist}[\mathtt{real}]\\ \mathtt{Poisson}&\colon \mathtt{real} \to \mathtt{dist}[\mathtt{nat}]\\ \mathtt{ProdD}&\colon \mathtt{dist}[\delta] \times \mathtt{dist}[\delta'] \to \mathtt{dist}[\delta \times \delta']\end{aligned}$$

Here $\mathtt{Normal}(r_1, r_2)$ creates a Gaussian distribution with mean r_1 and standard deviation r_2, and $\mathtt{Poisson}(r)$ a Poisson distribution with rate r. Finally $\mathtt{ProdD}$ combines two probability distributions by cartesian product.

3. Semantics

As we already mentioned in the introduction, the major challenge in defining the semantics of our language is that a natural setting for the semantics, namely, the category $\mathbf{M}$ of "small" measurable spaces and measurable functions, does not have a structure for interpreting function types; it is not cartesian closed.

Our idea to address this challenge is to embed $\mathbf{M}$ into a category with more structure using the following Yoneda functor:

$$\mathsf{Yoneda}\colon \mathbf{M} \to \mathsf{PSh}^{\times}(\mathbf{M})$$
$$\mathsf{Yoneda}(A) = \mathbf{M}(-, A) \qquad \mathsf{Yoneda}(f) = \mathbf{M}(-, f)$$

Here $\mathsf{PSh}^{\times}(\mathbf{M})$ is a full subcategory of the presheaf category on $\mathbf{M}$. It consists of functors preserving finite or countable products.[1] This embedding has three nice properties. First, the target category $\mathsf{PSh}^{\times}(\mathbf{M})$ has enough structure to interpret all types in our language: it has limits and colimits and is cartesian closed. Second, the Yoneda functor preserves finite or countable products and coproducts, which are often used to interpret first-order probabilistic programs. Third, for every strong monad M on $\mathbf{M}$, there exists a strong monad $\widehat{M}$ on $\mathsf{PSh}^{\times}(\mathbf{M})$ such that

$$\mathsf{Yoneda} \circ M = \widehat{M} \circ \mathsf{Yoneda}$$

In particular, this means that we can lift the following variant G' of the Giry monad G on $\mathbf{M}$ to a strong monad $\widehat{G'}$ on $\mathsf{PSh}^{\times}(\mathbf{M})$.

$$G'\colon \mathbf{M} \to \mathbf{M}, \quad G'(A) = G(\mathbb{R}_{\geq 0} \times A), \quad G'(f) = G(\mathrm{id}_{\mathbb{R}_{\geq 0}} \times f).$$

Here $G'(A)$ consists of probability measures on pairs $(r, a) \in \mathbb{R}_{\geq 0} \times A$. On a measurable function $f\colon A \to B$, the monad maps such a probability measure $\mu \in G'(A)$ to $\mu \circ (\mathrm{id} \times f)^{-1} \in G'(B)$. Intuitively, $\mu \in G'(A)$ defines a distribution on weighted samples on A, and it implicitly represents a renormalised version of this distribution using these associated weights. This monad G' captures the informal importance-sampling semantics advocated by van de Meent and Wood, two key designers of Anglican.

Our semantics interprets measure types δ as measure spaces $\langle\!\langle \delta \rangle\!\rangle = (A, \mathcal{F}, \mu)$ with certain properties. The reader does not need to know all these properties of $(A, \mathcal{F}, \mu)$ and even the details of our interpretation, as long as she or he understands the following two consequences. First, the properties make it possible to identify a default topology on A and to say when a measurable function from A to $\mathbb{R}_{\geq 0}$ is continuous. Second, they ensure that the collection of such measurable continuous functions f with $\int_A f d\mu = 1$ forms a measurable space $(A \Rightarrow_{c,\mu} \mathbb{R}_{\geq 0})$ where the evaluation map is measurable [1]:

$$\mathrm{ev}\colon (A \Rightarrow_{c,\mu} \mathbb{R}_{\geq 0}) \times A \to \mathbb{R}_{\geq 0}, \qquad \mathrm{ev}(f, x) = f(x).$$

These consequences enable us to interpret the distribution type $\mathtt{dist}[\delta]$ and primitives $\mathtt{sample}_\delta$ and $\mathtt{observe}_\delta$ in our language.

The rest of our interpretation is carried out in the category $\mathsf{PSh}^{\times}(\mathbf{M})$. We interpret general types as objects in $\mathsf{PSh}^{\times}(\mathbf{M})$:

$$\begin{aligned}[\![\delta]\!] &= \mathsf{Yoneda}(A) && ((A, \mathcal{F}, \mu) = \langle\!\langle \delta \rangle\!\rangle)\\ [\![\mathtt{dist}[\delta]]\!] &= \mathsf{Yoneda}((A \Rightarrow_{c,\mu} \mathbb{R}_{\geq 0}) \times \{\mu\}) && ((A, \mathcal{F}, \mu) = \langle\!\langle \delta \rangle\!\rangle)\\ [\![\tau \times \tau']\!] &= [\![\tau]\!] \times [\![\tau']\!] \qquad\qquad [\![\tau \to \tau']\!] = [\![\tau]\!] \Rightarrow \widehat{G'}([\![\tau']\!])\end{aligned}$$

where the σ-algebra of $\{\mu\}$ is $\{\emptyset, \{\mu\}\}$. We interpret terms as morphisms in $\mathsf{PSh}^{\times}(\mathbf{M})$. It is mostly standard, and we just show the two most interesting cases:

$$\begin{aligned}&[\![\mathtt{sample}_\delta]\!] : [\![\mathtt{dist}[\delta]]\!] \to \widehat{G'}([\![\delta]\!])\\ &[\![\mathtt{sample}_\delta]\!] = \mathsf{Yoneda}(\lambda(f, \mu).\, \lambda X.\, \int_{\{a \mid (1,a) \in X\}} f \,\mathrm{d}\mu)\\ &[\![\mathtt{observe}_\delta]\!] : [\![\mathtt{dist}[\delta] \times \delta]\!] \to \widehat{G'}([\![\mathtt{unit}]\!])\\ &[\![\mathtt{observe}_\delta]\!] = \mathsf{Yoneda}(\lambda((f, \mu), v).\, \lambda X.\\ &\qquad\qquad \text{if } ((f(v), *) \in X) \text{ then } 1 \text{ else } 0)\end{aligned}$$

where $*$ is the unique element in the terminal object of $\mathbf{M}$.

Our semantics answers a variant of the Church-Turing thesis often raised by Anglican users: can we only express measurable functions on reals in Anglican despite all the higher-order functions there? For a similar higher-order probabilistic programming language, Park, Pfenning and Thrun conjectured a positive answer [3], but no formal proofs have been given. Our semantics confirms the conjecture and fill in the gap:

Theorem 3.1. *For all terms $x\colon \mathtt{real} \vdash e\colon \mathtt{real}$, there exists a measurable function f such that $[\![e]\!] = \mathsf{Yoneda}(f)$.*

The initial motivation for developing our semantics is to validate optimisation techniques that some of us have developed for Anglican. Our semantics indeed validates program transformations that marginalise internally used random variables. However, to validate other advanced optimisations, we have to pose program equivalence after normalisation, which is defined in terms of a natural transformation of the following type:

$$\mathrm{norm}\colon G(\mathbb{R}_{\geq 0} \times (-)) \to (\mathbb{R}_{>0} \times G(-) + 1 + 1)$$

where G is the Giry monad, 1 a terminal object in $\mathbf{M}$, $\mathbb{R}_{>0}$ the measurable space of positive numbers, and $+$ the coproduct in $\mathbf{M}$. If this abstract gets accepted, we will explain program equivalence in detail during the talk.

Selected References

[1] R. J. Aumann. Borel structures for function spaces. *Illinois J. Math.*, 5(4):614–630, 12 1961.

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

[3] S. Park, F. Pfenning, and S. Thrun. A probabilistic language based on sampling functions. *TOPLAS*, 31(1), 2008.

[4] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL'02*, 2002.

[5] F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *AISTATS'14*, 2014.

[1] Since these are functors from $\mathbf{M}^{\mathrm{op}}$ to $\mathbf{Sets}$, they map finite or countable coproducts in $\mathbf{M}$ to products in $\mathbf{Sets}$.