Semantics of Probabilistic Programs

Mark Goldstein

1 Intro

This document is concerned with the formal semantics of probabilistic programs used for probabilstic modeling and machine learning. One approach in this setting is that

- we want programs in our language to specify probabilistic generative models of data, and
- we want these programs to be interpreted as (un-normalized) posterior distributions over variables of interest in the model (latents and model parameters)

That is, we want our compilers/interpreters to do probabilstic inference. Once one specifies a programming language, including its terms, types, and syntax, one must specify how to interpret what a program means: in our case we must specify how a generative model maps to a posterior distribution. We will focus on denotational semantics, which associate a mathematical object with each fragment of program code in a compositional way. As such, each piece of our program will describe a distribution and the whole program will too.

We present a walkthrough of *Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints* [1, 2016] and *Commutative semantics for probabilistic programming* [2, 2017], mainly focusing on the presentation in [2]. We first briefly review measures and kernels because they show up in the semantics. We will later be interested in approximate inference but focus on the exact case here.

2 Measure Theory

A **σ-algebra** Σ_X on a set X is a collection of subsets of X that contains $\emptyset$ and is closed under complements and countable unions. A **measurable space** is a pair (X, Σ_X) of a set and a σ-algebra on it. The elements of Σ_X, which are themselves sets of elements in X, are called **measurable sets**. For example, the Borel sets are the smallest σ-algebra on $\mathbb{R}$ that contains the intervals. This is the usual σ-algebra for $\mathbb{R}$. For any countable set, you can always start off with the individual elements of the set: satisfying complements and unions this gives you the powerset σ-algebra.

A **measure** on a measurable space (X, Σ_X) is a function $\mu : \Sigma_X \to [0, \infty]$ into the set $[0, \infty]$ of extended non-negative reals that takes countable disjoint unions to sums, i.e. $\mu(\emptyset) = 0$ and $\mu(\cup_{n\in\mathbb{N}} U_n) = \Sigma_{n\in\mathbb{N}} \mu(U_n)$ for any $\mathbb{N}$-indexed sequence of disjoint measurable sets U_n. A **probability measure** is a measure μ such that $\mu(X) = 1$. For example, the Lebesgue measure λ on $\mathbb{R}$ is generated by $\lambda\Big((a, b)\Big) = b - a$. For any $x \in X$, the Dirac measure $\delta_x(U) = 1$ if $x \in U$ and 0 otherwise.

A **kernel** k from X to Y, notated $k : X \rightsquigarrow Y$, is a function $k : X \times \Sigma_Y \to [0, \infty]$ such that

- for fixed x, $k(x, -) : \Sigma_Y \to [0, \infty]$ is a measure
- for fixed dy, $k(-, dy) : X \to [0, \infty]$ is a measurable function.

An intuitive example of kernels at work is conditional probability. Take $k(x,-)$ to be a probability measure $\mu_Y : \Sigma_Y \to [0,1]$ on Y given a particular value $X = x$: $\sum_{dy \in \Sigma_y} k(x,dy) = 1$. Note that $\sum_{x \in X} k(x,dy) \neq 1$ in general.

3 Types and Semantics

The two papers [1,2] don't actually define the set of terms of the language, but they are implied to be the typical ones in a language like Haskell. They just define a set of types and typing rules. The 2018 POPL paper on semantics for higher-order languages [3] does formally introduce a kind and type system and a set of terms. We define our types as:

$$\mathbb{A}, \mathbb{B} ::= \mathbb{R} | P(\mathbb{A}) | 1 | \mathbb{A} \times \mathbb{B} | \sum_i \mathbb{A}_{i \in I}$$

where I is countable and non-empty. As we will see, types are to be interpreted as measurable spaces $[[\mathbb{A}]]$. We have sum and product types. We have a type $\mathbb{R}$ that denotes the Reals $[[\mathbb{R}]]$. We have a type $P(\mathbb{A})$ that denotes the set of probability measures $[[P(\mathbb{A})]]$ over a space denoted by $\mathbb{A}$. We have the type 1 the denotes a singleton set. For bools, we can just take $(1+1)$ and $P(1+1)$ is the type for distributions over bools. For the natural numbers, we can use $\sum_{i \in \mathbb{N}} 1$. Our terms are therefore elements of these sets, like elements of a σ-algebra whose probability we are interested in and density functions.

Typing judgements: $\Gamma \vdash_d$ for deterministic judgements and $\Gamma \vdash_p$ for probabilistic judgements. Having the two typing judgements is mainly for notational clarity: it helps us to define interpretation differently for deterministic and probabilistic terms.

3.1 Typing Judgements and Interpretations for Deterministic Terms

Deterministic terms $\Gamma \vdash_d t : \mathbb{A}$ are interpreted as measurable functions $[[t]] : [[\Gamma]] \to [[\mathbb{A}]]$, where $[[t]](\gamma) = x$ is an element of the underlying set $[[\mathbb{A}]]$ and not a measurable set in $\Sigma_{[[\mathbb{A}]]}$. **Note:** the following informally uses the notation $\Gamma :: (x : \mathbb{A}) :: \Gamma'$ to show a particular variable x of type $\mathbb{A}$ in the environment.

- **basic term** $x : \mathbb{A}$

$$\frac{}{\Gamma :: (x : A) :: \Gamma' \vdash_d x : \mathbb{A}} \qquad [[x]](\gamma :: d :: \gamma') = d$$

- **Disjoint Sum Type**

$$\frac{\Gamma \vdash_d t : \mathbb{A}_i}{\Gamma \vdash_d (i,t) : \sum_{i \in I} \mathbb{A}_i} \qquad [[(i,t)]](\gamma) = (i, [[t]](\gamma))$$

- **Deterministic `case`**

$$\frac{\Gamma \vdash_d t : \sum_{i \in I} \mathbb{A}_i \qquad (\Gamma :: (x : \mathbb{A}_i) \vdash_d u_i : \mathbb{B})_{i \in I}}{\Gamma \vdash_d (\texttt{case } t \texttt{ of } \{(i,x) \to u_i\}_{i \in I}) : \mathbb{B}}$$

$$[[\texttt{case}\ \ t\ \ \texttt{of}\ \ \{(i,x)\to u_i\}_{i\in I}]](\gamma) = [[u_i]](\gamma, d)\ \ \texttt{if}\ \ [[t]](\gamma) = (i,d)$$

- **Unit ()**

$$\frac{}{\Gamma \vdash_d () : 1} \qquad [[()]](\gamma) = ()$$

- **Product Type**

$$\frac{\Gamma \vdash_d t_0 : \mathbb{A}_0 \qquad \Gamma \vdash_d t_1 : \mathbb{A}_1}{\Gamma \vdash_d (t_0, t_1) : \mathbb{A}_0 \times \mathbb{A}_1} \qquad [[(t_0,t_1)]](\gamma) = ([[t_0]](\gamma), [[t_1]](\gamma))$$

- **Projection**

$$\frac{\Gamma \vdash_d t : \mathbb{A}_0 \times \mathbb{A}_1}{\Gamma \vdash_d \pi_j(t) : \mathbb{A}_j} \qquad [[\pi_j(t)]](\gamma) = d_j \text{ if } [[t]](\gamma) = (d_0, d_1)$$

where the `case` expression above has a deterministic continuation u_i. We will come back to the probabilistic continuation case. Now the semantics for sequencing.

3.2 Typing Judgements and Interpretations for Probabilistic Terms

Probabilistic terms $\Gamma \vdash_p t : \mathbb{A}$ are interpreted as s-finite kernels $[[t]] : [[\Gamma]] \rightsquigarrow [[\mathbb{A}]]$.

- `return`**(t)**

$$\frac{\Gamma \vdash_d t : \mathbb{A}}{\Gamma \vdash_p \texttt{return}(t) : \mathbb{A}} \qquad [[\texttt{return}(t)]](\gamma, da) = \delta_{[[t]](\gamma)}(da)$$

  this corresponds to a Dirac Delta measure sitting at a point $[[t]](\gamma)$.

- `let` **x = t** `in` **u**

$$\frac{\Gamma \vdash_p t : \mathbb{A} \quad \Gamma, x : \mathbb{A} \vdash_p u : \mathbb{B}}{\Gamma \vdash_p \texttt{let}\ \ x = t\ \ \texttt{in}\ \ u : \mathbb{B}} \qquad [[\texttt{let}\ \ x = t\ \ \texttt{in}\ \ u]](\gamma, db) = \int_{x,dx\in[[\mathbb{A}]]} [[u]]\Big(\gamma :: x, db\Big)[[t]]\Big(\gamma, dx\Big)$$

- **probabilistic** `case`

$$\frac{\Gamma \vdash_d t : \sum_{i\in I} \mathbb{A}_i \qquad (\Gamma, x : \mathbb{A}_i \vdash_p u_i : \mathbb{B})_{i\in I}}{\Gamma \vdash_p (\texttt{case}\ \ t\ \ \texttt{of}\ \ \{(i,x)\to u_i\}_{i\in I}) : \mathbb{B}}$$

$$[[\texttt{case}\ \ t\ \ \texttt{of}\ \ \{(i,x)\to u_i\}_{i\in I}]](\gamma, db) = [[u_i]](\gamma :: d, db)\ \ \texttt{if}\ \ [[t]](\gamma) = (i,d)$$

- `sample`**(t)**

$$\frac{\Gamma \vdash_d t : P(\mathbb{A})}{\Gamma \vdash_p \texttt{sample}(t) : \mathbb{A}} \qquad [[\texttt{sample}(t)]](\gamma, da) = \Big([[t]](\gamma)\Big)(da)$$

- `score`**(t)**

$$\frac{\Gamma \vdash_d t : \mathbb{R}}{\Gamma \vdash_p \texttt{score}(t) : 1}$$

$$[[\texttt{score}(t)]](\gamma, du) = \left\{ \begin{array}{lr} abs\Big([[t]](\gamma)\Big), & \text{if } du = \{()\} \\ 0, & \text{if } du = \emptyset \end{array} \right\}$$

3.3 Typing Judgements and Interpretation for Normalize

TODO

4 Example Programs

Beta-Bernoulli model: in the following, we derive that "the un-normalized posterior is a measure defined by integrating the likelihood with respect to the prior".

$$[[\texttt{let } x = \texttt{sample}(\texttt{Beta}(2,2)) \texttt{ in } \texttt{score}(\texttt{Bernoulli}(1;x)); \texttt{return}(x)]](db)$$

$$= \int_x [[\texttt{let } a = \texttt{score}(\texttt{Bernoulli}(1;x)) \texttt{ in } \texttt{return}(x)]](\gamma :: x, db) \quad [[\texttt{sample}(\texttt{Beta}(2,2))]](\gamma, dx)$$
$$= \int_x [[\texttt{let } a = \texttt{score}(\texttt{Bernoulli}(1;x)) \texttt{ in } \texttt{return}(x)]](\gamma :: x, db) \quad \texttt{Beta}(dx;2,2)$$
$$= \int_x \Big(\sum_{s \in \{\emptyset, \{()\}\}} [[\texttt{return}(x)]](\gamma :: x :: a, db) \quad [[\texttt{score}(\texttt{Bernoulli}(1;x))]](\gamma :: x, s) \Big) \quad \texttt{Beta}(dx;2,2)$$
$$= \int_x \Big([[\texttt{return}(x)]](\gamma :: x :: a, db) \quad [[\texttt{score}(\texttt{Bernoulli}(1;x))]](\gamma :: x, \{()\}) \Big) \quad \texttt{Beta}(dx;2,2)$$
$$= \int_x \Big([[\texttt{return}(x)]](\gamma :: x :: a, db) \quad \texttt{Bernoulli}(1;x) \Big) \quad \texttt{Beta}(dx;2,2)$$
$$= \int_x \Big(\mathbb{1}\big[x \in db\big] \quad \texttt{Bernoulli}(1;x) \Big) \quad \texttt{Beta}(dx;2,2)$$
$$= \texttt{UnnormalizedPosterior}(db) = \int_{x \in db} \texttt{Bernoulli}(1;x) \quad \texttt{Beta}(dx;2,2)$$
$$= \int_{x \in db} \Big(x^{2+1-1}(1-x)^{2-1} \Big)$$

This gives the expected result. We were able to interpret the program as an un-normalized posterior for a Beta-Bernoulli model.

Note on conjugacy: In this example, we consider a Beta-Bernoulli model. The Beta distribution is a continuous distribution over $[0,1]$ whose shape is determined by two parameters α and β so that the density is $\text{Beta}(x;\alpha,\beta) = \frac{1}{C(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta=1}$ where $C(\alpha,\beta)$ is just a normalizing constant. The Bernoulli distribution is a discrete distribution over $\{0,1\}$ with parameter $x \in [0,1]$ defined by mass function $\texttt{Bernoulli}(d;x) = x$ if $d = 1$ and $1-x$ if $d = 0$. The two distributions have the following property: if x is distributed $\texttt{Beta}(\alpha,\beta)$ and you observe data-points $d_i \in \{0,1\}$ with likelihood distribution $\prod_i \texttt{Bernoulli}(d_i;x)$ (Bernoulli with parameter x), then the posterior distribution over x is $\texttt{Beta}(\alpha + \#1, \beta + \#0)$, where there are $\#1$ $1's$ and $\#0$ $0's$ in the data. This is a result coming from the Beta and Bernoulli being a **conjugate pair**. If we were to wrap the entire program in a call to `normalize()`, we should get a measure with density $\texttt{Beta}(2+1,2)$

TODO: Definition of Density: if $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$, then: ...

TODO: For probability measure p with density g, recover the importance sampling algorithm for sampling from p by sampling from a Gaussian by showing:

$$\begin{aligned} [[\texttt{sample}(p)]] &= [[\texttt{let } x = \textit{lebesgue} \texttt{ in } \texttt{score}(p(x)); \texttt{return}(x)]] \\ &= [[\texttt{let } x = \textit{gauss}(0,1) \texttt{ in } \texttt{score}(1/f(x)); \texttt{score}(g(x)); \texttt{return}(x)]] \\ &= [[\texttt{let } x = \textit{gauss}(0,1) \texttt{ in } \texttt{score}(g(x)/f(x)); \texttt{return}(x)]] \end{aligned}$$

5 Higher Order

...

6 Citation

1. Staton et al. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. LICS 2016. https://arxiv.org/pdf/1601.04943.pdf

2. Staton. Commutative semantics for probabilistic programming. ESOP 2017. ‘ http://www.cs.ox.ac.uk/people/samuel.staton/papers/esop2017.pdf

3. Scibior et al. Denotational validation of Bayesian inference. POPL 2018. https://arxiv.org/pdf/1711.03219.pdf