Semantics of Probabilistic Programs
Mark Goldstein

1 Intro

A walkthrough of [1] and [2], mainly focusing on the presentation in [2].

2 Measures

...

3 Kernels

We need to briefly review kernels because they show up in the semantics. 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 on Σ_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.

Let X, Y, Z be measurable spaces and let $k^1 : X \times Y \rightsquigarrow Z$ and $k^2 : X \rightsquigarrow Y$ be s-finite kernels (**TODO**: explain s-finiteness). Then we can define the composition $(k^1 \star k^2) : X \rightsquigarrow Z$ as

$$(k^1 \star k^2)(x, U) = \int_Y k^2(x, dy) k^1(x, y, U)$$

For intuition, consider $k(x, -) : \Sigma_Y \to [0, 1]$ to be the probability measure that tells you how likely it is to start off at location x and end up in the interval dy. Then the composition $(k^1 \star k^2)(x, dz) = \int_Y k^2(x, dy) k^1(x, y, dz)$ can be taken to represent the probability of starting off at x and ending up in the interval dz, where we take a step in-between to land on y, but average out across all intervals dy that we can land in when jumping from x.

4 Types and Semantics

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

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

where I is countable and non-empty. Sum and product types. Reals. Distributions over $\mathbb{A}$. $(1 + 1)$ is the type of bools, $P(1 + 1)$ is the type for distributions over bools, and $\sum_{i \in \mathbb{N}} 1$ is the type for natural numbers.

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.

Types are interpreted as measurable spaces $[[\mathbb{A}]]$.

4.1 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 $[[\mathbb{A}]]$ rather than a measurable set in $\Sigma_{[[\mathbb{A}]]}$.

- $[[x]](\gamma :: d :: \gamma') = d$
- $[[(i,t)]](\gamma) = (i, [[t]](\gamma))$
- $[[\texttt{case}\ \ t\ \ \texttt{of}\ \ \{(i,x) \to u_i\}_{i \in I}]](\gamma) = [[u_i]](\gamma, d)$ if $[[t]](\gamma) = (i, d)$
- $[[()]](\gamma) = ()$
- $[[(t_0, t_1)]](\gamma) = ([[t_0]](\gamma), [[t_1]](\gamma))$
- $[[\pi_j(t)]](\gamma) = d_j$ 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.

4.2 Typing Judgements and Interpretations for Probabilstic 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` $\mathbf{x = t}$ `in` $\mathbf{u}$**

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

  This one takes careful reading. Consider the kernel composition definition above. Take k_1 to be $[[u]]$ and take k_2 to be $[[t]]$. Then $(k^1 \star k^2)(\gamma, db) = ([[u]] \star [[t]])(\gamma, db)$

- **probabilstic `case`**

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

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

- $\texttt{sample}(\mathbf{t})$

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

- $\texttt{score}(\mathbf{t})$

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

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

4.3 Typing Judgements and Interpretation for Normalize

$$\frac{\Gamma \vdash_p t : \mathbb{A}}{\Gamma \vdash_d \texttt{normalize}(t) : \mathbb{R} \times P(\mathbb{A}) + 1 + 1}$$

To give it a semantics, we must find the normalizing constant to divide by. Consider $\Gamma \vdash_p t : \mathbb{A}$ and let $\texttt{evidence}_t = [[t]]_{\gamma, [[\mathbb{A}]]}$. Then:

$$[[\texttt{normalize}(t)]](\gamma) = \left\{ \begin{array}{lr} (0, (\texttt{evidence}_t, \frac{[[t]](\gamma, (-))}{\texttt{evidence}_t})), & \text{if } \texttt{evidence}_t \in (0, \infty) \\ (1, ()), & \text{if } \texttt{evidence}_t = 0 \\ (2, ()), & \text{if } \texttt{evidence}_t = \infty \end{array} \right\}$$

5 Example Programs

...

6 Commutativity

...

7 Higher Order

...

8 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