Probabilistic Programming From the Ground Up

Steven Holtzen
11 June 2024

Lecture 2: Conditioning Sampling

Example: Network packet reachability

Compute the probability of packets reaching the end of the network

\[
\text{route} \leftarrow \text{flip } 0.5; \quad \text{true: top path}
\]

\[
l_1 \leftarrow \text{flip } 0.99;
\]

\[
l_2 \leftarrow \text{flip } 0.99;
\]

\[
l_3 \leftarrow \text{flip } 0.99;
\]

\[
l_4 \leftarrow \text{flip } 0.99;
\]

\[
\text{return } \begin{cases} l_1 \land l_3 & \text{if route} \\ l_2 \land l_4 & \text{else} \end{cases}
\]
Imagine packet does not get to the end of the network. What is the probability that \( l_1 \) failed?

- This is an example of Bayesian conditioning.
- Given the packet did not reach the end of the network, compute the probability of failure.
- This inspires an extension to Tiny PPL, changing it to Tiny Cond.

\[
p := \text{true} \mid \text{false} \\
C := \text{flip } r \mid x := e \mid \text{return } p \mid \text{observe } p \mid x
\]

To compute the probability of link failure, packet not reaching the end of the network:

\[
\begin{align*}
\text{route} & \leftarrow \text{flip } 1/2 \\
l_1 & \leftarrow \text{flip } 0.99; \\
l_2 & \leftarrow \text{flip } 0.99; \\
l_3 & \leftarrow \text{flip } 0.99; \\
l_4 & \leftarrow \text{flip } 0.99; \\
\text{observe } & \left( \text{return } (\text{if route then } l_4 \land l_3 \text{ else } l_2 \land l_4) \right)
\end{align*}
\]
Smaller Example to describe observe

\[ x \leftarrow \text{flip } 1/2; \]
\[ y \leftarrow \text{flip } 1/2; \]

What is the probability that \( x \) is observed if \( x \) is heads?

\[ \text{return } x \]

Best way to make sense semantically of what conditioning is doing us a "possible worlds" semantics of this program.

If we get rid of flip, we would be left with a bunch of deterministic programs:

1. \( x \leftarrow \text{return true}; \)
2. \( x = \text{false}; \)
3. \( x = \text{true}; \)
4. \( x \leftarrow \text{false}; \)

\[ y \leftarrow \text{return false}; \]
\[ y = \text{true}; \]
\[ y = \text{false}; \]

observe \( x \land y \);

\[ \text{return } x \]

Probability: \( \frac{1}{4} \)

\( P = \frac{1}{4} \)  \( P = \frac{1}{4} \)  \( P = \frac{1}{4} \)

If we know \( x \) or \( y \) is true, that removes one possible world (FF).

Removing one world means that the probability distribution doesn't sum to 1. So we need to renormalize, dividing by the total of remaining probabilities.
Divide each of the remaining by $3/4$

This is an instance of an application of Bayes’ rule.

- Probability $x = \text{true}$ is $2/3$ (because we're peeling at the remaining worlds, the probabilities of the ones with $x = \text{true}$ adds up to $2/3$).

What is semantics of $g$?

$$
\begin{align*}
x &\leftarrow \text{flip 1/2;} \\
y &\leftarrow \text{flip 1/2;} \\
\text{observe x v y;} & \\
\text{return x}
\end{align*}
$$

Semantics of $\text{Tiny Cond}$ (will have two semantics):

1. Unnormalized semantics: (doesn’t sum to one).
2. Denotes subprobability distributions, with all characteristics of probability distributions except not summing to 1.

$$
\begin{align*}
\text{flip 1/2} &\mathbb{P}(P)(tt) = 1/2 \\
\text{true} &\mathbb{P}(P)(tt) = 1 \\
\text{observe p e} &\mathbb{P}(P)(v) = \begin{cases} 
1/2 & \text{if } P = \text{tt} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
$$
Normalized Semantics:

\[
\delta_{\mathcal{E}}(\varphi)(v) = \frac{\delta_{\mathcal{E}}u(\varphi)(v)}{\delta_{\mathcal{E}}u(\varphi)(tt) + \delta_{\mathcal{E}}u(\varphi)(ff)}
\]

Note: Justin Hsu has example of effective version of Semantics. OPLSS ’21 Reasoning about Probabilistic Programs.

Q. How is \( p \) deterministic given its probabilistic semantics?

A. \[ \Gamma \vdash e : \text{Dist Bool} \quad \Gamma \vdash x \mapsto B \quad \Gamma \vdash e_2 : \text{Bool} \]

\[ \Gamma \vdash x \leftarrow e_1, e_2 : \text{Dist Bool} \]

In a monadic language we might have a type like this; terms remain pure, due to its monadic structure.

Tricky to think about:

\[
\begin{align*}
    x & \leftarrow \text{flip } \frac{1}{2}; \\
    y & \leftarrow \text{flip } \frac{1}{2}; \\
    \text{observe } x \lor y; \\
    \text{return } x
\end{align*}
\]

\[
\begin{align*}
    x & \leftarrow \text{flip } \frac{2}{3}; \\
    y & \leftarrow \text{flip } \frac{2}{3}; \\
    \text{return } x
\end{align*}
\]

This is difficult to do because it’s a tricky bit of time travel.

Effects are non-local & they perturb the world.
1. Are the following programs the same?
   While t := y = observe false

A. Dexter Kozen says no.
   Could go either way, depends on whether termination is
   an "observation" or not.
   Loops are tricky to talk about denotationally.

Q. If observe x, y: return x is replaced by if x ≤ y: return
   is that a valid change?
A. No, it's an unsound choice to make because of leaking

Q. Are there different flavors of observe? like observe x given
A. Not in our language, there are choices one could make
   for the thing you can pass to observe.

Q. Does adding observe make it worst then # P-hard?
A. No, because we've taken the semantics of tiny cond
   and added a simple counting query, not significantly
   altering the asymptotic cost.

Q. Could we not commutate observe all the way up to the
   last assignment, do some local rewriting?
A. That requires globally looking at the program to see
   if that's okay, requiring non-local reasoning.

In general, it is useful to push observe "up" as far as possible.
Operational Sampling Semantics

So far semantics has not been really efficient, because we've been summing over all possible assignments to these variables.

Practically, we want a runtime so we don't have to resort to worst case exponential sums all the time. We want runtime that works on interesting examples but are tractable to compute.

Example:

```
Ex: 1= x ← flip 1/2; Sample
    y ← flip 1/2; return x ∨ y
```

\[ \text{Ex} \approx 2/4 \]

Every single probabilistic program used today makes use of the following trick to approximate semantics.

Operator: Expectation of a random variable

\[ E_{Pr} L[f] = \sum_{c \in \omega} Pr(c) \times f(c) \]

\[ \approx \frac{1}{n} \sum_{c \in \omega \cap Pr} \frac{1}{Pr} \cdot P(c) \]

\( \uparrow \text{Drawn from probability distribution} \)
Drawing from probability distribution estimates the probability using the expectation.

We're going to prove our mnemonic strategy bound by relating it to expectation and then we're going to use the sampling estimator to conclude that in some finite number of runs, this will converge to the right semantics.

\[ \text{expectation} = \square \]

Q. Does it become very slow, because we have to draw too many samples (ie. from the program a lot)

A. example a series of 1000 \( x \leftarrow \text{flip} \)

If we drew 10 samples we get a good estimate, 100 a very close one.

Concentration inequality tells us how many samples we need.

**runtime** (Big step semantics)

Judgement \( \sigma \vdash \langle e, p \rangle \downarrow v \)

in the context \( \Gamma \), this term evaluates to value \( v \).

Reference: Wolpepper & Cobb 2017 "Contextual Equivalence for a PPL with CRV & return."
V \vdash \sigma \rightarrow \text{flip } \frac{1}{2} \rightarrow V

\prod_i \sigma \rightarrow \langle e_1, \rho \rangle \rightarrow V \quad \prod_i \sigma \rightarrow \langle e_2, \rho \rangle \rightarrow V

\sigma \rightarrow \langle x, e_1, e_2, \rho \rangle \rightarrow V'

pure expression: \sigma \vdash \text{return } \langle p, s \rangle \rightarrow V

Q. Why do we need infinite stream (\sigma)?
- It is convenient, can make 2 infinite streams from one
- Can handle much richer semantics, unbounded number of coin tosses

Q. Does this let us reason about continuous probability?
A. To add reals make \sigma a countably infinite product

\sigma \in [0,1]^\mathbb{N} (Hilbert's cube)

Uniform sample:

\nu \vdash \sigma \rightarrow \text{uniform } \rightarrow V

We want to relate the runtime of a program to an expectation by computing

\mathbb{E}_{\sigma \sim \nu} \left[ \mathbb{E} \left( \text{eval } (e, \sigma, s) = \tau \right) \right] = \left[ e \right] (\rho)

\sigma \sim \text{uniform distribution } \mathbb{B}^V
We look at one case of the theorem: the flip case.

\[ \mathbb{E} \left[ \frac{1}{2} \text{eval} \left( \text{flip} \frac{1}{2}, 6, 8 \right) = tt \right] = \frac{1}{2} \]

We need 2 lemmas to prove this.

Lemma: case analysis

\[ \mathbb{E}_\theta \left[ f(\theta) \right] = \mathbb{E}_\theta \left[ \frac{1}{2} f(tt, v) + \frac{1}{2} f(66, v) \right] \]

Lemma: \[ \mathbb{E}_\theta \left[ k \right] = k \]

Using the case analysis lemma:

\[ \frac{1}{2} \mathbb{E} \left[ \frac{1}{2} \text{eval} \left( \text{flip} \frac{1}{2}, tt, v, 8 \right) = tt \right] \]
\[ + \frac{1}{2} \mathbb{E} \left[ \text{eval} \left( \text{flip} \frac{1}{2}, tt, v, 8 \right) = 66 \right] \]

\[ = \mathbb{E} \left[ \frac{1}{2} \right] = \frac{1}{2} \]

Q. Why is bind independent?
A. It is because of the monadic semantics of bind.

\[ \mathbb{E}_\theta \left[ f(\theta) \cdot g(\theta) \right] = \mathbb{E}_\theta \left[ f(\theta) \right] \cdot \mathbb{E}_\theta \left[ g(\theta) \right] \]