Probabilistic Programming from the Ground Up

Lecture 2: Conditioning, sampling

June 11, 2024

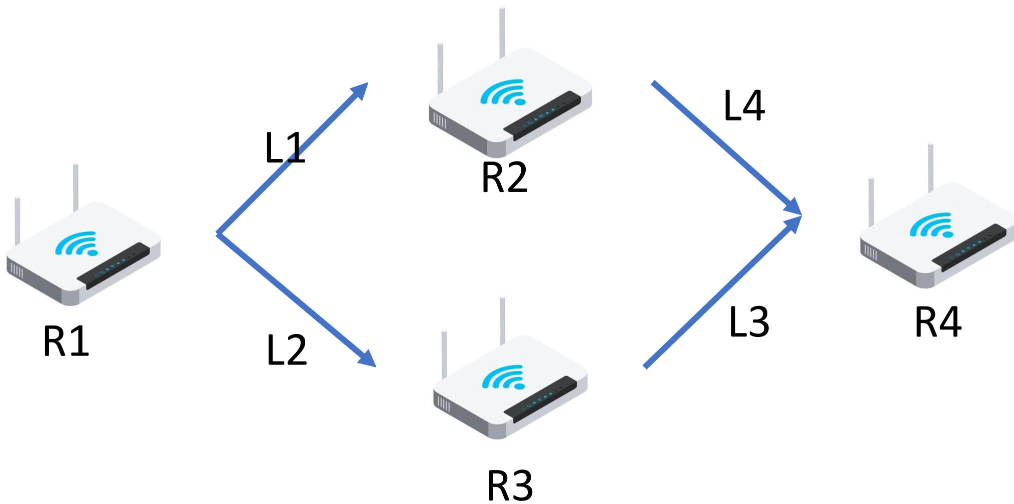
Steven Holtzen

s.holtzen@northeastern.edu

Oregon Programming Languages Summer School 2024

https://www.khoury.northeastern.edu/home/sholtzen/oplss24-ppl/

TinyPPL Exercise: Network Reliability



Suppose:

- each link fails independently with probably 1/50
- each router chooses which router to forward an incoming packet to with uniform probability.

What is the probability that an incoming packet reaches R4?

Network reliability

```
let network = tinyppl_e_of_string
  "(bind r2forward (flip 0.5)
   (bind l1fail (flip 0.02)
   (bind l2fail (flip 0.02)
   (bind l3fail (flip 0.02)
   (bind l4fail (flip 0.02)
   (return (if r2forward (and (not l1fail) (not l4fail))
                         (and (not l2fail) (not l3fail)))))))))”;;

> prob network StringMap.empty true ;;
- : float = 0.960399999999999809
```

TinyCond

Bayesian conditioning and observation

What is conditioning

- Conditioning *updates your beliefs about the world* given *observations*

- Classic example: medical diagnosis

- Your COVID-19 test comes back positive. Does that mean you have COVID?
  - Not necessarily!
  - The probability that you have COVID should *go up*, but by how much? This depends on the test, prevalence of COVID, etc.

COVID Test Motivating Example

- Query: What is the probability that I have COVID given the test came back positive?
- Required data:
  - **True positive probability**: the probability that the test will be positive if you do have COVID

$$\Pr(\text{ Test} = T \mid \text{Covid} = T\ ) = 0.99$$

  - **False positive probability**: the probability that the test will be positive if you do not have COVID

$$\Pr(\text{ Test} = T \mid \text{Covid} = F\ ) = 0.05$$

  - **Latent rate**: the probability that an average person has COVID

$$\Pr(\text{Covid} = T) = 0.01$$

Possible worlds

Check that probability of all worlds sums to 1

COVID	Test	Pr(COVID, Test)
T	T	0.01*0.99=0.0099
T	F	0.01*0.01=0.0001
F	T	0.99*0.05=0.0495
F	F	0.99*0.95=0.9405

Possible worlds after conditioning on Test = T

COVID	Test	
T	T	0.0099
T	F	0
F	T	0.0495
F	F	0

Called ***unnormalized probability*** *distribution*, since it does not sum to 1.

Possible worlds after renormalizing (Bayes’s Rule)

COVID	Test	Pr(Test | Covid = True)
T	T	0.0099/(0.0099+0.0495)=0.1666666
T	F	0.0001
F	T	0.0495/(0.0099+0.0495)=0.8333333
F	F	0.9405

Note: Even though the test came back positive, it is still more likely that we do not have COVID!

Probability can be surprisingly unintuitive.

Modeling the COVID Diagnosis Scenario in a PPL

```
has_covid <- flip 0.01;
test_pos_with_covid <- flip 0.99;
test_pos_no_covid <- flip 0.05;
test <- return if covid then test_pos_with_covid
               else test_pos_no_covid;
observe test;
return has_covid
```

TinyCond Grammar

Pure computations

```
<p> ::=
  | <ident>
  | true
  | false
  | if <p> then <p> else <p>
  | <p> && <p>
  | <p> || <p>
  | !<p>
```

Probabilistic computations

```
<e> ::=
  | flip <float>
  | <id> <- <e>; <e>
  | observe <e>; <e>
  | return <p>
```

Semantics of TinyCond

- **Unnormalized semantics**: denoted $[\![\mathrm{e}]\!]_U$ essentially the same as TinyPPL, but with added rule for observe:

$$[\![\texttt{observe}\, e_1; e_2]\!]_U\,(\rho)(v) = \begin{cases} [\![e_2]\!]\,(\rho)(v) & \textbf{if } [\![e_1]\!]\,(\rho) = \texttt{true} \\ 0 & \textbf{otherwise.} \end{cases}$$

- Simply assigns probability 0 to all executions that do not satisfy the observation.

Normalized semantics

- Then, we can compute the normalized semantics from the unnormalized semantics:

$$[\![\texttt{e}]\!](\rho)(v) = \frac{[\![\texttt{e}]\!](\rho)(v)}{[\![\texttt{e}]\!](\rho)(\texttt{tt}) + [\![\texttt{e}]\!](\rho)(\texttt{ff})}$$

Example Semantics of TinyCond

$$\left[\!\left[\begin{array}{l}\texttt{x <- flip 1/2;}\\ \texttt{y <- flip 1/2;}\\ \texttt{observe x || y;}\\ \texttt{return x}\end{array} \right]\!\right]_U (tt) = \frac{2}{4}$$

$$\left[\!\left[\begin{array}{l}\texttt{x <- flip 1/2;}\\ \texttt{y <- flip 1/2;}\\ \texttt{observe x || y;}\\ \texttt{return x}\end{array} \right]\!\right]_U (ff) = \frac{1}{4}$$

$$\left[\!\left[\begin{array}{l}\texttt{x <- flip 1/2;}\\ \texttt{y <- flip 1/2;}\\ \texttt{observe x || y;}\\ \texttt{return x}\end{array} \right]\!\right] (tt) = \frac{\frac{2}{4}}{\frac{2}{4}+\frac{1}{4}} = \frac{2}{3}$$

Bayesian Learning

To keep the code concise, we added in an effectful if

- Suppose I want to learn whether a coin is biased
  - If it's biased, then it lands heads with probability 0.9
- Initially you think there is a 50% chance the coin is biased; this is called your **prior**
- You observe three coin outcomes, True, True, False. Now you want to know **the posterior** probability of whether the coin is biased.

```
biased <- flip 0.5;
flip1 <- if biased then flip 0.9 else flip 0.5;
observe flip1;
flip2 <- if biased then flip 0.9 else flip 0.5;
observe flip2;
flip3 <- if biased then flip 0.9 else flip 0.5;
observe flip3;
return biased
```

Non-locality of Conditioning

```
x <- flip 0.5;
y <- flip 0.5;
observe x || y;
return x
```

$\equiv$

```
x <- flip 0.6666;
y <- flip 0.6666;
return x
```

Semantically, observation “reaches back in time” to affect previous probabilistic operations in the program!

Sampling semantics

Context

- Last time we discussed *semantics and modeling* in TinyPPL and TinySamp
- We gave small implementations of both these languages
- **Problem**: These implementations were **inefficient** and **inexpressive**

Challenge 1: Scalability

- What is the probability that this program returns true?

```
let big_program = tinyppl_e_of_string
  "(bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (bind x (flip 0.5)
   (return x))))))))))))))))))))))))"
```

Takes a long time to compute for TinyCond and TinyPPL, even though we can easily see it’s 0.5

Challenge 2: Expressivity

- TinyPPL and TinyCond are quite restrictive languages: very few language features.

- Limits ability to model realistic scenarios

Direct Sampling Semantics

- Simple intuition: sample all probabilistic quantities as they are encountered
  - Then you're left with a pure program you can run with standard semantics

```
x <- flip 0.5;
y <- flip 0.5;
return x || y;
```

→

```
x <- return true;
y <- flip 0.5;
return x || y;
```

→

```
x <- return true;
y <- return false;
return true || false;
```

↓

```
tt
```

Approximation Semantics Intuition

- We can approximate the semantics of a program by drawing many samples

```
x <- flip 0.5;
y <- flip 0.5;
return x || y;
```

Draw finite number of samples

Sampled value
tt
tt
ff
ff
tt

- Suppose we draw the above 5 samples; then we would conclude that program outputs T with probability approximately 3/5

Expectations

- Let Pr : $\Omega \to [0,1]$ be a **probability distribution** on a set Ω (we will assume countable sets)
  - Required that $\sum_{\omega \in \Omega} \Pr(\omega) = 1$.
- A **random variable** is a map out of the sample space; we will assume it is real-valued $f : \Omega \to real$
- Then, the **expectation of f with respect to Pr** is defined as:

$$\mathbf{E}_{\Pr}[f] = \sum_{\omega \in \Omega} \Pr(\omega) f(\omega).$$

Law of large numbers

- A few different theorems of the general shape:

$$\mathbf{E}_{\mathrm{Pr}}[f] = \lim_{N \to \infty} \frac{1}{N} \sum_{\omega_i \sim \mathrm{Pr}}^{N} f(\omega_i).$$

This notation means "draw N independent samples from Pr"

Expectation Estimator

- For a fixed finite N:

$$\mathbf{E}_{\mathrm{Pr}}[f] \approx \frac{1}{N} \sum_{\omega_i \sim \mathrm{Pr}}^{N} f(\omega_i).$$

- Lots of interesting theorems on bounding how quickly this estimator approaches the true expectation; see "concentration inequalities"

Direct Sampling Semantics

- Let's give a sampling semantics for TinyPPL with only fair coin flips

Pure computations

```
<p> ::=
  | <ident>
  | true
  | false
  | if <p> then <p> else <p>
  | <p> && <p>
  | <p> || <p>
  | !<p>
```

Probabilistic computations

```
<e> ::=
  | flip 1/2
  | <id> <- <e>; <e>
  | return <p>
```

High level goal

1. Give a semantics to TinyPPL that formalizes "drawing a sample". Call this the (direct) sampling semantics.

2. Relate the expectation of direct sampling semantics to the denotation of TinyPPL programs

3. Use this relationship to establish (asymptotic) correctness of sampling using expectation estimator

See Wand, Mitchell, et al. "Contextual equivalence for a probabilistic language with continuous random variables and recursion." Proceedings of the ACM on Programming Languages 2.ICFP (2018): 1-30.

Direct Sampling Semantics

- Is a big-step relation:

$$\sigma \vdash \langle \mathsf{e}, \rho \rangle \Downarrow v$$

Return value

Entropy space: an **infinite list** of Boolean values. Think of this like your random number generator.

A probabilistic TinyPPL term

An environment

- "In the context of the entropy space, the term e with environment ρ steps to value v"
- Will be defined inductively on terms

Sampling Semantics Definition

$$f :: \sigma \vdash \langle \texttt{flip}\ 1/2, \rho \rangle \Downarrow f$$

To handle coin flips, step to the first element of the entropy space.

Sampling Semantics Definition

$$\frac{\langle \mathtt{p}, \rho \rangle \Downarrow v}{\sigma \vdash \langle \mathtt{return\ \ p}, \rho \rangle \Downarrow v}$$

To handle returning pure values, run the pure term and return the value it runs to.

Sampling Semantics Definition

- To handle bind, we need to split up the entropy space into two independent streams of random values

- We do this with two auxiliary functions: π_L and π_R.
  - Define π_L to take all even-indexed elements of the entropy space, and π_R to take all odd-indexed elements.

Sampling Semantics Definition

$$\frac{\pi_L(\sigma) \vdash \langle \mathsf{e}_1, \rho \rangle \Downarrow v \qquad \pi_R(\sigma) \vdash \langle \mathsf{e}_2, \rho[x \mapsto v] \rangle \Downarrow v'}{\sigma \vdash \langle x \leftarrow \mathsf{e}_1; \mathsf{e}_2, \rho \rangle \Downarrow v'}$$

Evaluation function

- Auxiliary function called eval that runs a program to its returned value:

$$\text{eval}(\sigma, \rho, \mathtt{e}) = v \quad \text{if } \sigma \vdash \langle \mathtt{e}, \rho \rangle \Downarrow v$$

- This is a well-defined function because sampling semantics is deterministic (check!)
- It's a partial function: if the available entropy is too small, it will get stuck.

Theorem: Adequacy of sampling semantics

- Let e be a probabilistic TinyPPL program
- Let ρ be an environment that is well-typed for e (i.e., contains all free variables in e)
- Let σ be an entropy space that is "big enough" (i.e., eval cannot get stuck)
- Let $\mathrm{Pr}(\sigma)$ be a uniform probability distribution on entropy spaces (i.e., assigns probability $\frac{1}{|\sigma|}$ to every entropy space σ, where $|\sigma|$ is the length of the list

- Then,

$$\mathbf{E}_\sigma([\texttt{eval}(\texttt{e}, \rho, \sigma) = \texttt{tt}]) = [\![\texttt{e}]\!](\rho)(\texttt{tt})$$

Lemmas

Left as exercises for the curious

1. (Constant) For k constant: $\mathbf{E}_\sigma(k) = k$

2. (Splitting) $\mathbf{E}_\sigma(f(\sigma)) = \mathbf{E}_\sigma\left(\frac{1}{2}f(\texttt{tt} :: \sigma) + \frac{1}{2}f(\texttt{ff} :: \sigma)\right)$

3. (Independent) For f and g independent:

$$\mathbf{E}_\sigma(f(\sigma) \times g(\sigma)) = \mathbf{E}_\sigma(f(\sigma)) \times \mathbf{E}_\sigma(g(\sigma))$$

Proof for flip

$$
\begin{aligned}
&\mathbf{E}_\sigma\left([\texttt{eval}(\texttt{flip}, \sigma, \rho) = \texttt{tt}]\right) \\
&= \mathbf{E}_\sigma\left(1/2[\texttt{eval}(\texttt{flip}, \texttt{tt} :: \sigma, \rho) = \texttt{tt}] + 1/2[\texttt{eval}(\texttt{flip}, \texttt{ff} :: \sigma, \rho) = \texttt{tt}]\right) && \text{Split} \\
&= \mathbf{E}_\sigma\left(1/2\right) && \text{Def. of } \texttt{eval} \\
&= 1/2 && \text{Const} \\
&= [\![\texttt{flip}\ 1/2]\!](\rho)(\texttt{tt}).
\end{aligned}
$$

Proof for bind

$$
\begin{aligned}
&\mathbf{E}_\sigma\left([\mathtt{eval}(x \leftarrow \mathtt{e}_1; \mathtt{e}_2, \sigma, \rho) = \mathtt{tt}]\right) \\
&= \mathbf{E}_\sigma\left(\sum_{v'} [\mathtt{eval}(\mathtt{e}_1, \pi_L(\sigma), \rho) = v'] \times [\mathtt{eval}(\mathtt{e}_2, \pi_R(\sigma), \rho[x \mapsto v']) = \mathtt{tt}]\right) && \text{Def. of bind} \\
&= \sum_{v'} \mathbf{E}_\sigma\left([\mathtt{eval}(\mathtt{e}_1, \pi_L(\sigma), \rho) = v'] \times [\mathtt{eval}(\mathtt{e}_2, \pi_R(\sigma), \rho[x \mapsto v']) = \mathtt{tt}]\right) && \text{Linearity of } \mathbf{E} \\
&= \sum_{v'} \mathbf{E}_\sigma\left([\mathtt{eval}(\mathtt{e}_1, \pi_L(\sigma), \rho) = v']\right) \times \mathbf{E}_\sigma\left([\mathtt{eval}(\mathtt{e}_2, \pi_R(\sigma), \rho[x \mapsto v']) = \mathtt{tt}]\right) && \text{Indep} \\
&= \sum_{v'} [\![\mathtt{e}_1]\!](\rho)(v') \times [\![\mathtt{e}_2]\!](\rho[x \mapsto v'](\mathtt{tt}) && \text{I.H.} \\
&= [\![x \leftarrow \mathtt{e}_1; \mathtt{e}_2]\!](\rho)(\mathtt{tt}).
\end{aligned}
$$

TinySamp

```
(* some examples *)
let p1 = tinyppl_e_of_string "(bind x (flip 0.5) (return x))"

let p2 = tinyppl_e_of_string "(bind x (flip 0.5)
                              (bind y (flip 0.4)
                              (bind z (flip 0.6)
                              (return (if x y z)))))"

let () =
  assert (within_epsilon (estimate p1 true 10000) 0.5);
  assert (within_epsilon (estimate p2 true 10000) 0.5)
```

Rejection sampling

Adding observe

Pure computations

```
<p> ::=
  | <ident>
  | true
  | false
  | if <p> then <p> else <p>
  | <p> && <p>
  | <p> || <p>
  | !<p>
```

Probabilistic computations

```
<e> ::=
  | flip 1/2
  | <id> <- <e>; <e>
  | observe e; e
  | return <p>
```

Direct Sampling Semantics

- Simple intuition: just like sampling semantics, we sample all probabilistic quantities as they are encountered
  - If an observation is violated, **reject** the sample: do not count it towards the estimate

```
x <- flip 0.5;
y <- flip 0.5;
observe x || y;
return x;
```

→

```
x <- return false;
y <- flip 0.5;
observe x || y;
return x;
```

→

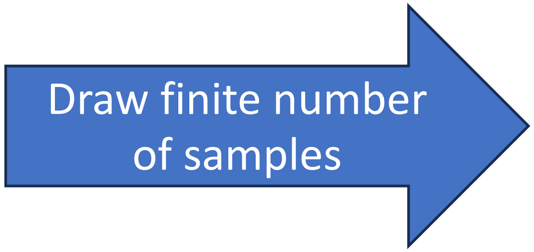
```
x <- return false;
y <- return false;
observe x || y;
return true;
```

↓

⊥

Rejection sampling

```
x <- flip 0.5;
y <- flip 0.5;
observe x || y;
return x;
```



Sampled value
tt
tt
⊥
⊥
ff

- Suppose we draw the above 5 samples; then we would conclude that program outputs T with probability approximately 1/3

Rejection Sampling Semantics

$$v :: \sigma \vdash \langle \texttt{flip}\,\theta, \rho\rangle \Downarrow v \qquad \frac{\langle p, \rho\rangle \Downarrow v}{\sigma \vdash \langle \texttt{return}\,p, \rho\rangle \Downarrow v}$$

$$\frac{\langle p, \rho\rangle \Downarrow \texttt{tt} \qquad \sigma \vdash \langle \mathrm{e}, \rho\rangle \Downarrow v}{\sigma \vdash \langle \texttt{observe}\,p; \mathrm{e}, \rho\rangle \Downarrow v} \qquad \frac{\langle p, \rho\rangle \Downarrow \texttt{ff}}{\sigma \vdash \langle \texttt{observe}\,p; \mathrm{e}, \rho\rangle \Downarrow \bot}$$

$$\frac{\pi_L(\sigma) \vdash \langle \mathrm{e}_1, \rho\rangle \Downarrow \bot}{\sigma \vdash \langle x \leftarrow \mathrm{e}_1; \mathrm{e}_2, \rho\rangle \Downarrow \bot} \qquad \frac{\pi_L(\sigma) \vdash \langle \mathrm{e}_1, \rho\rangle \Downarrow v \qquad v \neq \bot \qquad \pi_R(\sigma) \vdash \langle \mathrm{e}_2, \rho[x \mapsto v]\rangle \Downarrow v'}{\sigma \vdash \langle x \leftarrow \mathrm{e}_1; \mathrm{e}_2, \rho\rangle \Downarrow v'}$$