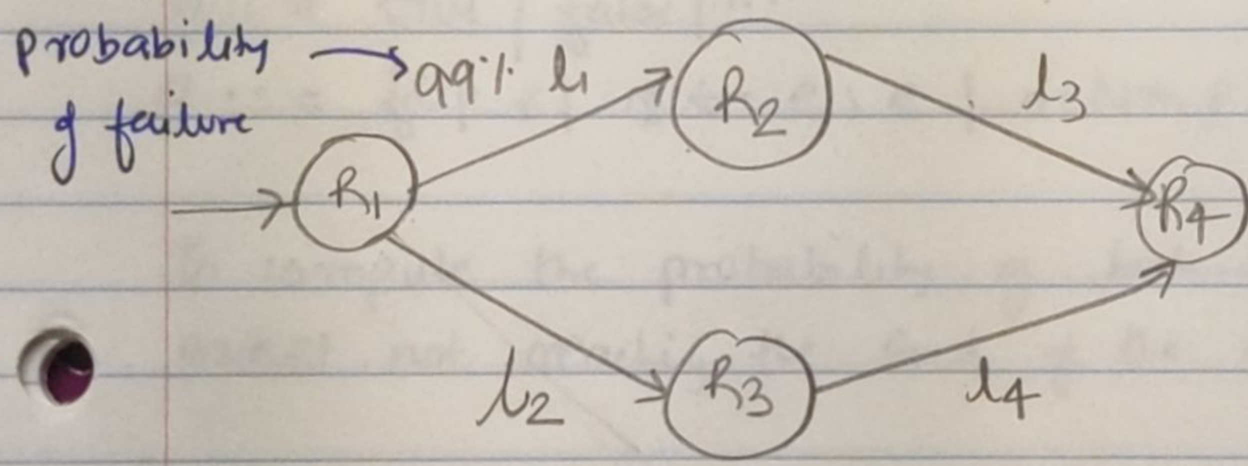
Probabilistic Programming From the Ground Up

Steven Holtzen

11 June 2024

Lecture 2: <u>Conditioning Sampling</u>

<u>Example</u>: Network packet reachability



Compute the probability of packets reaching the end of the network

```
route ← flip 1/2;      true; top path
l1 ← flip 0.99;
l2 ← flip 0.99;
l3 ← flip 0.99;
l4 ← flip 0.99;

return  if route
        then l1 ∧ l3
        else l2 ∧ l4
```

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 l_1 failing.
- This inspires an extension to Tiny PPL, changing it to
TinyCond.

$p ::= \text{true} \mid \text{false} \mid \cdots$
$e ::= \text{flip } r \mid x \leftarrow e; e \mid \text{return } p \mid \text{observe } p; e$

To compute the probability of ~~link l_1 failing~~ /
packet not reaching the end of the network

```
route ← flip 1/2
l1 ← flip 0.99;
l2 ← flip 0.99;
l3 ← flip 0.99;
l4 ← flip 0.99;
observe !(return (if route then l1 ∧ l3
                             else l2 ∧ l4))
```

Smaller Example to describe observe

```
x ← flip 1/2;
y ← flip 1/2;
observe x ∨ y;
return x
```

What is the probability that x is heads?

Best way to make sense semantically of what conditioning is doing is 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 ← return true;
y ← return false;
observe x ∨ y;
return x
```

2. x = F, y = T

3. x = T, y = F

4. ~~x = F, y = F~~

1.	2.	3.	4.
Probability: 1/4	P = 1/4	P = 1/4	~~P = 1/4~~

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

Removing one world means that the probability distribution doesn't sum to 1. So we need to <u>renormalize</u>, dividing by the total of remaining probabilities

Divide each of the remaining by 3/4
This is an instance of an application of Bayes's rule

$\therefore$ Probability x = true is $2/3$ (because in peeking at the remaining worlds, the probabilities of the ones with x = true adds up to $2/3$)

What is semantics of

$$\left[\!\left[\begin{array}{l} x \leftarrow \text{flip } 1/2; \\ y \leftarrow \text{flip } 1/2; \\ \text{observe } x \vee y; \\ \text{return } x \end{array}\right]\!\right] (tt) = \frac{2}{3}$$

<u>Semantics of TinyCond</u> (will have two semantics)

- Unnormalized Semantics : (doesn't sum to one).

Denotes sub probability distributions, with all characteristics of probability distributions except not summing to 1.

$$[\![\text{flip } 1/2]\!]_u (\rho)(tt) = 1/2$$

$$[\![\text{true}]\!]_u (\rho)(tt) = 1$$

$$\vdots$$

$$[\![\text{observe } p; e]\!]_u (\rho)(v) = \begin{cases} \text{if } [\![p]\!](\rho) = tt \text{ then } [\![e]\!]_u(\rho)(v) \\ 0, \text{ otherwise} \end{cases}$$

Normalized Semantics:

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

Note: Justin Hsu has examples of effectful versions of semantics OPLSS '21 Reasoning about Probabilistic Programs

Q. How is ρ deterministic given its probabilistic semantics?

A.

$$\frac{\Gamma \vdash e_1 : \text{Dist Bool} \quad \Gamma \cup \{x \mapsto B\} \vdash e_2 : \text{Bool}}{\Gamma \vdash x \leftarrow e_1; e_2 : \text{Dist Bool}}$$

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

Tricky to think about

$x \leftarrow$ flip 1/2;
$y \leftarrow$ flip 1/2;
observe $x \vee y$;
return x

$\equiv$

$x \leftarrow$ flip 2/3;
$y \leftarrow$ flip 2/3;
return x

This is difficult to do because its a tricky bit of time travel.

Effects are non-local & they perturb the world.

Q. Are the following programs the same?

while tt { y $\equiv$ observe ~~true~~ 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 $x \vee y$: return x is replaced by if $x \vee y$; return x 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 y

A Not in our language, those are choices one could make for the thing you can pass to observe.

Q. Does adding observe make it worse than # P-hard?

A No, because we've taken the semantics of binary cond and added a simple counting query, not significantly altering the asymptotic cost.

Q. Could we not commute 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 have 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:

Samples

$$ex := \begin{array}{l} x \leftarrow \text{flip } 1/2; \\ y \leftarrow \text{flip } 1/2; \\ \text{return } x \vee y \end{array}$$

T
T
f
f

$$[\![ex]\!] \approx 2/4$$

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

- **Operator:** Expectation of a random variable

$$\mathbb{E}_{Pr}[f] = \sum_{\omega} Pr(\omega) \times f(\omega)$$

$$\approx \frac{1}{N} \sum_{\omega \sim Pr}^{N} P(\omega)$$

Drawn from probability distribution

Drawing from probability distribution estimates the probability using this expectation.

We're going to prove our runtime strategy sound 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

$$\text{expectation} = [\![\cdot]\!]$$

Q. Doesn't this become very slow, because we have to draw too many samples (i.e. run the program a lot).

A example a series of 1000 $x \leftarrow$ flip ; m

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

Concentration inequalities tells us how many samples we need.

<u>Runtime</u> (Big step semantics)

Judgement $\sigma \vdash \langle e, p \rangle \Downarrow v$

$\in \mathbb{B}^N$ { in the context of Γ, this term evaluates to value v.

Reference: Culpepper & Cobb 2017 "Contextual equivalence for a PPL with CRV & recursion"

$$v :: \sigma \vdash \text{flip } 1/2 \Downarrow v$$

$$\frac{\pi_L(\sigma) \vdash \langle e_1, \rho \rangle \Downarrow v \qquad \pi_R(\sigma) \vdash \langle e_2, \rho[X \mapsto v] \rangle \Downarrow v'}{\sigma \vdash \langle X \leftarrow e_1 ; e_2, \rho \rangle \Downarrow v'}$$

pure expression: $\dfrac{}{\sigma \vdash \text{return } \langle \rho, \rho \rangle \Downarrow v}$

Q. Why do we need infinite stream (σ)?
- it is convinient, 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 probabilitis)
A To add reals make σ a countably infinite product

$$\sigma \in [0,1]^{\mathbb{N}} \quad (\text{Hilbert's Cube})$$

Uniform samble.

$$\frac{}{r :: \sigma \vdash \text{uniform} \Downarrow v}$$

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

$$\mathop{\mathbb{E}}_{\sigma \sim \Pr} \left[\mathbb{1}(\text{eval}(e, \sigma, \rho) = tt) \right] = [\![e]\!](\rho)(tt)$$

↳ Uniform Distribution $\mathbb{B}^{\mathbb{N}}$

We look at one case of the Theorem: the flip case

$$\mathbb{E}[\mathbb{1}(\text{eval}(\text{flip } 1/2, \sigma, \rho) = tt)] = 1/2$$

We need 2 Lemmas to prove this

Lemma: case analysis

$$\mathbb{E}_\sigma[f(\sigma)] = \mathbb{E}_\sigma\left[\frac{1}{2} f(tt :: \sigma) + \frac{1}{2} f(ff :: \sigma)\right]$$

Lemma: $\mathbb{E}_\sigma[k] = k$

Using the case analysis Lemma

$$\mathbb{E}\Big[\frac{1}{2}\mathbb{1}(\text{eval}(\text{flip } 1/2, tt :: \sigma, \rho) = tt)$$
$$+ \frac{1}{2}\underbrace{\cancel{\mathbb{1}(\text{eval}(\text{flip } 1/2, ff :: \sigma, \rho) = ff)}}_{0}\Big]$$

$$= \mathbb{E}[1/2] = 1/2$$

Q. Why is bind independent?

A. It's because of the monadic semantics of bind

$$\mathbb{E}_\sigma[f(\sigma) \cdot g(\sigma)] = \mathbb{E}_\sigma[f(\sigma)] \cdot \mathbb{E}_\sigma[g(\sigma)]$$