Probabilistic Programming

Lecture #10: Conditioning

Joost-Pieter Katoen

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

RWTH Lecture Series on Probabilistic Programming 2018

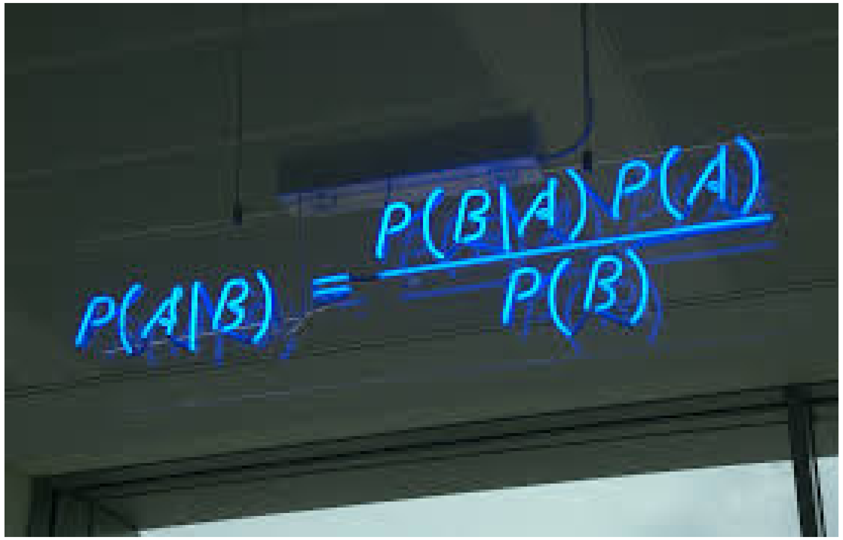
Overview

1. Motivation
2. Observe statements
3. Operational semantics
4. Conditional expected rewards
5. Program transformations

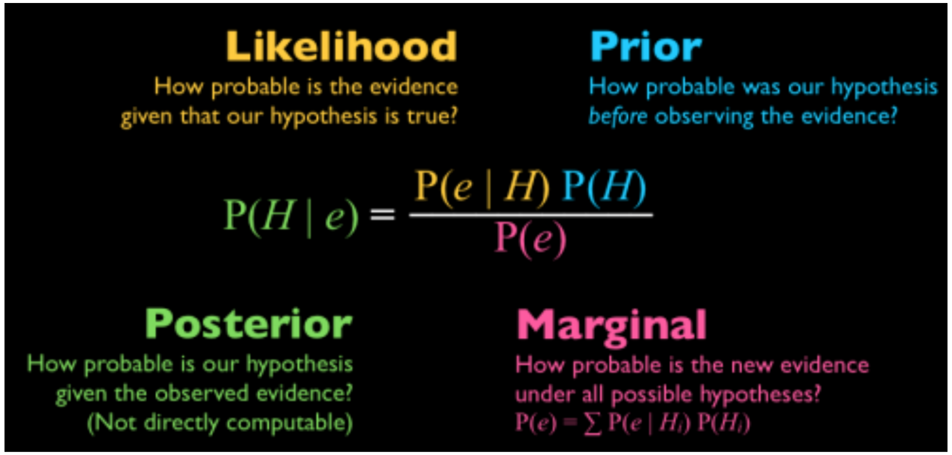
Overview

1. Motivation
2. Observe statements
3. Operational semantics
4. Conditional expected rewards
5. Program transformations

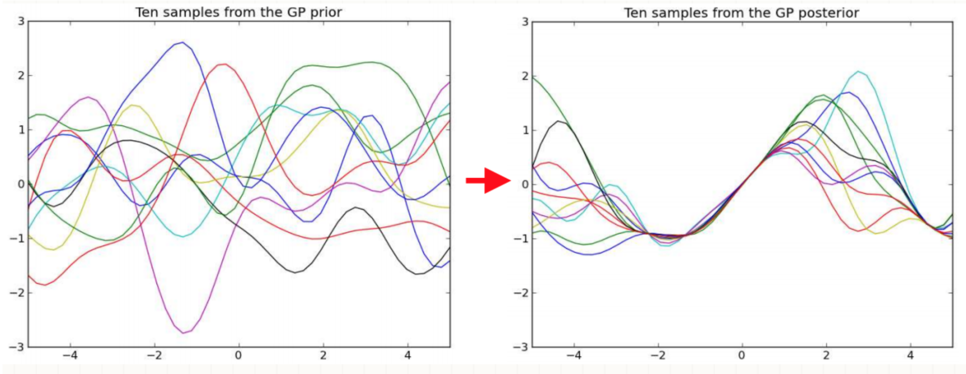
Bayes' rule



Bayes' rule explained



Conditioning = learning



Conditioning in `webPPL`

Overview

1. Motivation
2. Observe statements
3. Operational semantics
4. Conditional expected rewards
5. Program transformations

Conditional probabilistic GCL: cpGCL Syntax

- `skip` — empty statement
- `diverge` — divergence
- `x := E` — assignment
- `x :r= mu` — **random assignment** $(x :\approx \mu)$
- `observe (G)` — **conditioning**
- `prog1 ; prog2` — sequential composition
- `if (G) prog1 else prog2` — choice
- `prog1 [p] prog2` — **probabilistic choice**
- `while (G) prog` — iteration

Conditioning will be the key ingredient to be considered in this lecture.

Let's start simple

```
x := 0 [0.5] x := 1;
y := -1 [0.5] y := 0;
observe (x+y = 0)
```

This program blocks two runs as they violate `x+y = 0`. Outcome:

$$Pr[x=0, y=0] = Pr[x=1, y=-1] = 1/2$$

Observations thus normalize the probability of the "feasible" program runs

A loopy program

For $0 < p < 1$ an arbitrary probability:

```
bool c := true;
int i : = 0;
while (c) {
    i++;
    (c := false [p] c := true)
}
observe (odd(i))
```

The feasible program runs have a probability $\sum_{N \geq 0} (1-p)^{2N} \cdot p = \frac{1}{2-p}$

This program models the distribution:

$$Pr[i = 2N+1] = (1-p)^{2N} \cdot p \cdot (2-p) \quad \text{for } N \geq 0$$

$$Pr[i = 2N] = 0$$

A mathematician's perspective

A geometric distribution with $p = 1/2$, conditioned on "x is odd":

$$Pr(x = N \mid x \text{ is odd}) = \begin{cases} \frac{3}{2^{N+1}} & \text{if } N \text{ is odd} \\ 0 & \text{otherwise.} \end{cases}$$

A geometric distribution with $p = 1/3$, conditioned on "x is odd":

$$Pr(x = N \mid x \text{ is odd}) = \begin{cases} \frac{2^N \cdot 5}{3^{N+2}} & \text{if } N \text{ is odd} \\ 0 & \text{otherwise.} \end{cases}$$

Which program pairs are equivalent?

```
{ x := 0 [0.5] x := 1 };
observe(x = 1)
```

```
{ x := 0; observe(x = 1) }
[0.5]
{ x := 1; observe(x = 1) }
```

```
x := 1 [0.5] diverge
```

```
x := 1 [0.5] observe(false)
```

```
int x := 1;
while (x = 1) {
    x := 1
}
```

```
int x := 1;
while (x = 1) {
    x := 1 [0.5] x := 0;
    observe (x = 1)
}
```


Overview

1. Motivation
2. Observe statements
3. Operational semantics
4. Conditional expected rewards
5. Program transformations

Structural operational semantics: ingredients

- Variable valuation $s : Vars \to \mathbb{Q}$ maps each program variable onto a value (here: rational numbers)
- Expression valuation, let $[\![E]\!]$ denote the valuation of expression E
- Configuration (aka: state) $\langle P, s \rangle$ denotes that
  - program P is about to be executed (aka: program counter)
  - and the current variable valuation equals s.
- Transition rules for the execution of commands: $\langle P, s \rangle \longrightarrow \langle P', s' \rangle$

  transition rules are written as $\dfrac{\text{premise}}{\text{conclusion}}$

  where the premise is omitted if it is vacuously true.

Recall: Markov chains

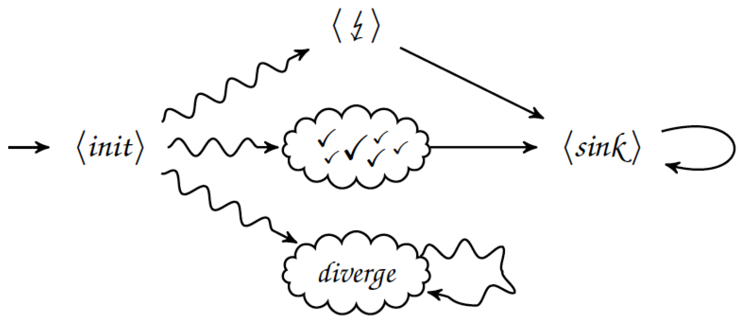
A Markov chain (MC) is a triple $(\Sigma, \sigma_I, \mathbf{P})$ with:

- Σ being a countable set of states
- $\sigma_I \in \Sigma$ the initial state, and
- $\mathbf{P} : \Sigma \to Dist(\Sigma)$ the transition probability function

where $Dist(\Sigma)$ is a discrete probability measure on Σ.

Operational semantics of conditional pGCL

Aim: Model the behaviour of a program P by the MC $[\![\, P \,]\!]$.



This can be defined using Plotkin's SOS-style semantics

Operational semantics

Aim: Model the behaviour of a conditional pGCL program P by MC $[\![\, P \,]\!]$.

Approach:

- Take states of the form
  - $\langle Q, s \rangle$ with program Q or $\downarrow$, and variable valuation $s : Vars \rightarrow \mathbb{Q}$
  - $\langle \lightning \rangle$ models the violation of an observation, and
  - $\langle sink \rangle$ models successful program termination
- Take initial state $\sigma_I = \langle P, s \rangle$ where s fulfils the initial conditions
- Transition relation $\rightarrow$ is the smallest relation satisfying the SOS rules on the next slides
  - Where transition probabilities equal to one are omitted

Transition rules for cpGCL (1)

$$\langle \texttt{skip}, s \rangle \rightarrow \langle \downarrow, s \rangle \qquad \langle \texttt{diverge}, s \rangle \rightarrow \langle \texttt{diverge}, s \rangle$$

$$\frac{s \models G}{\langle \texttt{observe}(G), s \rangle \rightarrow \langle \downarrow, s \rangle} \qquad \frac{s \not\models G}{\langle \texttt{observe}(G), s \rangle \rightarrow \langle \lightning \rangle}$$

$$\langle \downarrow, s \rangle \rightarrow \langle sink \rangle \qquad \langle \lightning \rangle \rightarrow \langle sink \rangle \qquad \langle sink \rangle \rightarrow \langle sink \rangle$$

$$\langle x := E, s \rangle \rightarrow \langle \downarrow, s[x := s([\![\, E \,]\!])] \rangle$$

$$\frac{\mu(s)(v) = a > 0}{\langle x :\approx \mu, s \rangle \xrightarrow{a} \langle \downarrow, s[x := v] \rangle}$$

$$\langle P[\, p \,]\, Q, s \rangle \rightarrow \mu \text{ with } \mu(\langle P, s \rangle) = p \text{ and } \mu(\langle Q, s \rangle) = 1{-}p$$

Transition rules for cpGCL (2)

$$\frac{\langle P, s \rangle \rightarrow \langle \lightning \rangle}{\langle P; Q, s \rangle \rightarrow \langle \lightning \rangle} \qquad \frac{\langle P, s \rangle \rightarrow \mu}{\langle P; Q, s \rangle \rightarrow \nu} \text{ with } \nu(\langle P'; Q', s' \rangle) = \mu(\langle P', s' \rangle) \text{ where } \downarrow; Q = Q$$

$$\frac{s \models G}{\langle \text{if } (G)\{P\} \text{ else } \{Q\}, s \rangle \rightarrow \langle P, s \rangle} \qquad \frac{s \not\models G}{\langle \text{if } (G)\{P\} \text{ else } \{Q\}, s \rangle \rightarrow \langle Q, s \rangle}$$

$$\frac{s \models G}{\langle \text{while}(G)\{P\}, s \rangle \rightarrow \langle P; \text{while } (G)\{P\}, s \rangle} \qquad \frac{s \not\models G}{\langle \text{while}(G)\{P\}, s \rangle \rightarrow \langle \downarrow, s \rangle}$$

Examples

The conditional distribution of a program

The conditional distribution $[\![P]\!]_\sigma \mid_{\neg \lightning}$ over terminal states of cpGCL program P when starting in state s is defined by:

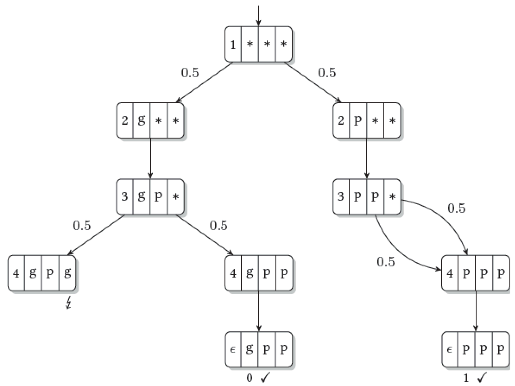
$$[\![P]\!]_\sigma \mid_{\neg \lightning}(\tau) \;=\; \begin{cases} 0 & \text{if } \tau = \lightning \text{ and } [\![P]\!]_\sigma(\lightning) < 1 \\ \dfrac{[\![P]\!]_\sigma(\tau)}{1 - [\![P]\!]_\sigma(\lightning)} & \text{if } \tau \neq \lightning \text{ and } [\![P]\!]_\sigma(\lightning) < 1 \\ \text{undefined} & \text{if } [\![P]\!]_\sigma(\lightning) = 1 \end{cases}$$

The piranha problem [Tijms, 2004]

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?

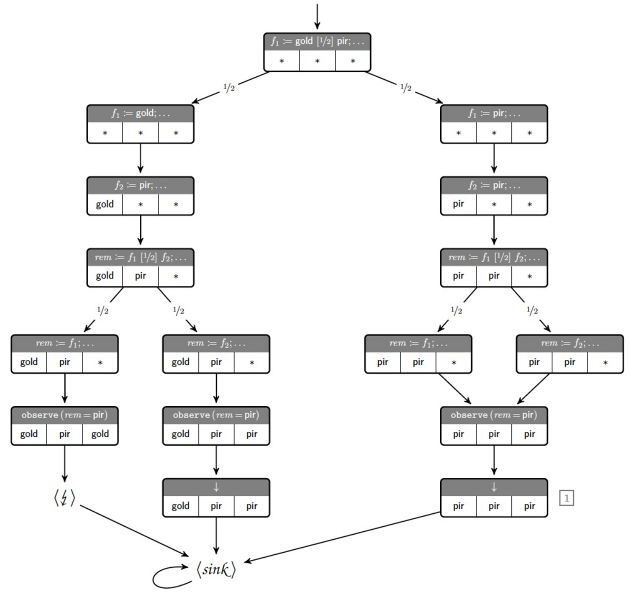
The piranha puzzle

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```



The full operational semantics

```
f1 := gf [0.5] f1 := pir
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```



Overview

1. Motivation
2. Observe statements
3. Operational semantics
4. Conditional expected rewards
5. Program transformations

Rewards

To reason about resource usage in MCs: use rewards.

MC with rewards

A reward MC is a pair (D, r) with D an MC with state space Σ and $r : \Sigma \to \mathbb{R}$ a function assigning a real reward to each state.

The reward $r(\sigma)$ stands for the reward earned on leaving state σ.

Cumulative reward for reachability

Let $\pi = \sigma_0 \ldots \sigma_n$ be a finite path in (D, r) and $G \subseteq \Sigma$ a set of target states with $\pi \in \Diamond G$. The cumulative reward along π until reaching G is:

$$r_G(\pi) = r(\sigma_0) + \ldots + r(\sigma_{k-1}) \text{ where } \sigma_i \notin G \text{ for all } i < k \text{ and } \sigma_k \in G.$$

If $\pi \notin \Diamond G$, then $r_G(\pi) = 0$.

Expected reward reachability

Expected reward for reachability

The expected reward until reaching $G \subseteq \Sigma$ from $\sigma \in \Sigma$ is:

$$\mathrm{ER}(\sigma, \Diamond G) = \sum_{\pi \models \Diamond G} Pr(\hat{\pi}) \cdot r_G(\hat{\pi})$$

where $\hat{\pi} = \sigma_0 \ldots \sigma_k$ is the shortest prefix of π such that $\sigma_k \in G$ and $\sigma_0 = \sigma$.

Conditional expected reward

Let $\mathrm{ER}(\sigma, \Diamond G \mid \neg\Diamond F)$ be the conditional expected reward until reaching G under the condition that no states in $F \subseteq \Sigma$ are visited.

Conditional expected reward

$\mathrm{ER}(\sigma, \Diamond G \mid \neg\Diamond F)$ is the expectation of random variable[1] $rv(\Diamond G \cap \neg\Diamond F)$ with respect to the conditional probability measure:

$$Pr(\Diamond G \mid \neg\Diamond F) \;=\; \frac{Pr(\Diamond G \cap \neg\Diamond F)}{Pr(\neg\Diamond F)}$$

Conditional expected reward

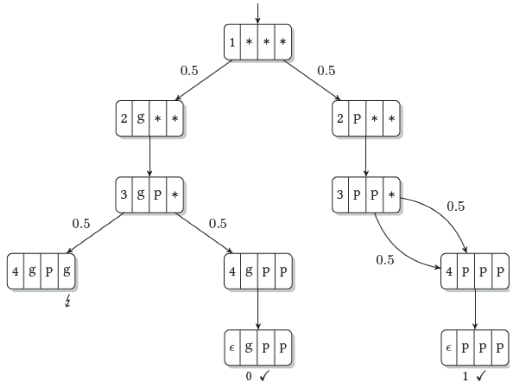
The conditional expected reward to reach $G \subseteq \Sigma$ while avoiding $F \subseteq \Sigma$ in Markov chain D is defined as:

$$\mathrm{ER}^D(\Diamond G \mid \neg\Diamond F) \;=\; \frac{\mathrm{ER}^D(\Diamond G \cap \neg\Diamond F)}{Pr(\neg\Diamond F)}$$

[1] This r.v. assigns to each path π of MC D the reward $r(\hat{\pi})$ where $\hat{\pi}$ is the shortest prefix of π such that the last state is in G and no previous state is in F.

The piranha puzzle

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```



What is the probability that the original fish in the bowl was a piranha?

Conditional expected reward of termination without violating any observe

$$\mathrm{ER}^{[\![P]\!]}(\sigma_I, \Diamond\langle sink\rangle \mid \neg\Diamond\langle \lightning \rangle) = \frac{1 \cdot {}^1\!/_2 + 0 \cdot {}^1\!/_4}{1 - {}^1\!/_4} = \frac{{}^1\!/_2}{{}^3\!/_4} = {}^2\!/_3.$$

A remark on divergence

Consider the two programs:

```
x := 1 [0.5] diverge
```

```
x := 1 [0.5] observe(false)
```

Q: What is the probability that `x = 1` on termination?

A: For the left program this is $1/2$; for the right one this is 1.

Divergence matters

```
diverge [0.5] {
   x := 0 [0.5] x := 1;
   y := 0 [0.5] y := 1;
   observe (x = 0 || y = 0)
}
```

Q: What is the probability that `y = 0` on termination?

A: $\frac{2}{7}$. Why?

Warning: This is a silly example. Typically divergence comes from loops.

Observations inside loops

Consider the following two "similar" programs:

```
int x := 1;
while (x = 1) {
    x := 1
}
```

- Certain divergence
- Conditional expected reward = 0

```
int x := 1;
while (x = 1) {
    x := 1 [0.5] x := 0;
    observe (x = 1)
}
```

- Divergence with probability zero
- Conditional expected reward = undefined

Our semantics does distinguish these programs.

Overview

1 Motivation
2 Observe statements
3 Operational semantics
4 Conditional expected rewards
5 Program transformations

Why formal semantics matters

- Unambiguous meaning to all programs
- Basis for proving correctness
  - of programs
  - of **program transformations**
  - of program equivalence
  - of static analysis
  - of compilers
  - ......

Program transformation to remove conditioning

- Idea: restart an infeasible run until all `observe`-statements are passed
- For program variable `x` use auxiliary variable `sx`
  - store initial value of `x` into `sx`
  - on each new loop-iteration restore `x` to `sx`
- Use auxiliary variable `flag` to signal observation violation:

```
flag := true; while(flag) { flag := false; mprog }
```

- Change `prog` into `mprog` by:
  - `observe(G)` $\rightsquigarrow$ `flag := !G || flag`
  - `abort` $\rightsquigarrow$ `if(!flag) abort`
  - `while(G) prog` $\rightsquigarrow$ `while(G && !flag) prog`

Resulting program

```
sx1,...,sxn := x1,...,xn; flag := true;
while(flag) {
   flag := false;
   x1,...,xn := sx1,...,sxn;
   modprog
}
```

In machine learning, this is known as rejection sampling.

Removal of conditioning

the transformation in action:

```
x := 0 [p] x := 1;
y := 0 [p] y := 1;
observe(x != y)
```

```
sx, sy := x, y; flag := true;
while(flag) {
  x, y := sx, sy; flag := false;
  x := 0 [p] x := 1;
  y := 0 [p] y := 1;
  flag := (x = y)
}
```

a simple data-flow analysis yields:

```
repeat {
  x := 0 [p] x := 1;
  y := 0 [p] y := 1
} until(x != y)
```


A dual program transformation

```
repeat
  a0 := 0 [0.5] a0 := 1;
  a1 := 0 [0.5] a1 := 1;
  a2 := 0 [0.5] a2 := 1;
  i := 4*a2 + 2*a1 + a0 + 1
until (1 <= i <= 6)
```

```
a0 := 0 [0.5] a0 := 1;
a1 := 0 [0.5] a1 := 1;
a2 := 0 [0.5] a2 := 1;
i := 4*a2 + 2*a1 + a0 + 1
observe (1 <= i <= 6)
```

Loop-by-observe replacement if there is "no data flow" between loop iterations

A third program transformation: Hoisting

Correctness of these transformations

This can be done by comparing conditional expected rewards in the Markov chains of the program before and after the program transformation.

Next lecture: prove the correctness using conditional weakest pre-expectations.