Probabilistic Programming

Lecture #12: Conditioning

Joost-Pieter Katoen

RWTH Lecture Series on Probabilistic Programming 2022-23

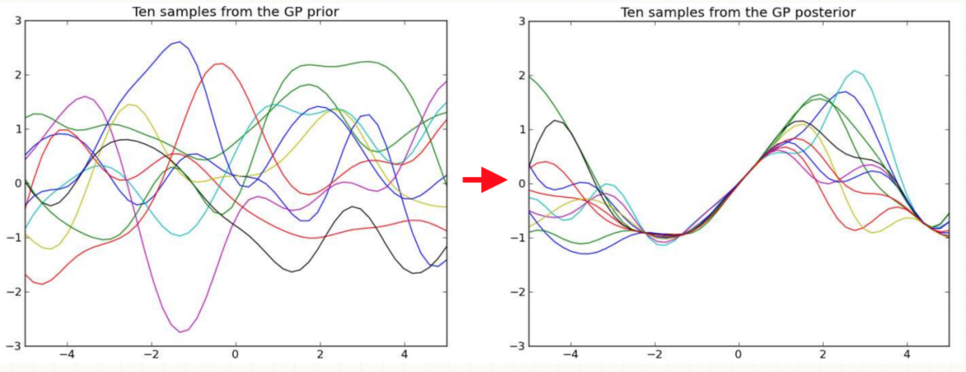
Overview

Overview

Bayes' rule

$$\underbrace{Pr(X \mid D)}_{\substack{\text{posterior of } X \\ \text{given } D}} = \frac{\overbrace{Pr(D \mid X)}^{\substack{\text{likelihood of } X \\ \text{under } D}} \cdot \overbrace{Pr(X)}^{\text{prior of } X}}{\underbrace{Pr(D)}_{\text{marginal}}}$$

Conditioning = learning



Observations change the distribution over data

Conditional probabilistic GCL

- `skip` — empty statement
- `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

A simple example

```
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 $\sum_{N \geqslant 0} (1-p)^{2N} \cdot p = \dfrac{1}{2-p}$

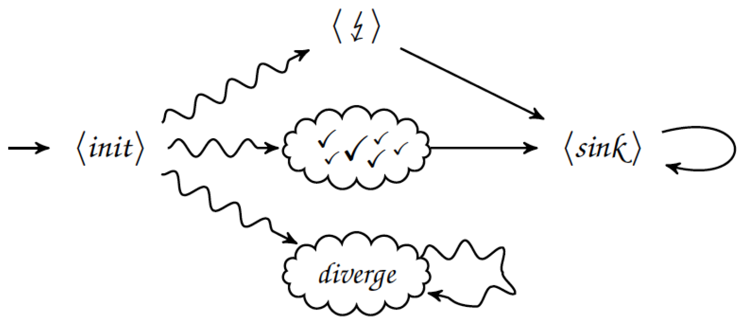
This program models the distribution:

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

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

Operational semantics

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



This can be defined using structured operational semantics

Operational semantics

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

Approach:

- Take states of the form
  - $\langle Q, s\rangle$ with program Q or $\downarrow$, and variable valuation $s : Var \to \mathbb{Q}$
  - $\langle \lightning \rangle$ models the violation of an observation, and
  - $\langle sink\rangle$ models program termination (successful or violated observation)
- Take initial state $\langle P, s\rangle$ where s fulfils the initial conditions
- Take transition relation $\to$ as smallest relation satisfying the transition rules

Transition rules (1)

$$\langle \texttt{skip}, s\rangle \to \langle \downarrow, s\rangle$$

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

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

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

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

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

Transition rules (2)

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

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

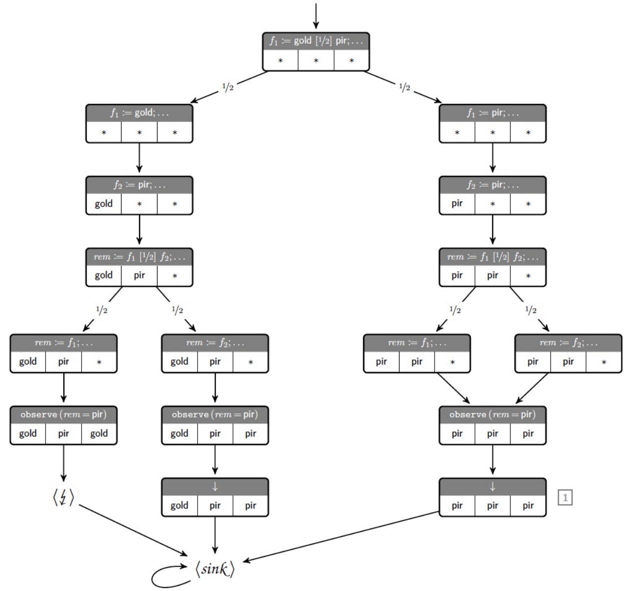
$$\frac{s \models G}{\langle \text{while}(G)\{P\}, s\rangle \to \langle P; \text{while } (G)\{P\}, s\rangle} \qquad \frac{s \not\models G}{\langle \text{while}(G)\{P\}, s\rangle \to \langle \downarrow, s\rangle}$$

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?

Example

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



The conditional distribution of a program

Consider the operational semantics $[\![P]\!]_s$ of cpGCL program P

The conditional distribution $[\![P]\!]_s |_{\neg ↯}$ over final states of cpGCL program P when starting in state s is defined by:

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

The normalisation factor $1 - [\![P]\!]_s(↯)$ includes diverging runs.

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.

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

Extending wp (and similarly wlp) with conditioning

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$\mathtt{observe}(\varphi)$	$[\varphi] \cdot f$
$x :\approx \mu$	$\lambda s.\ \int_{\mathbb{Q}} (\lambda v. f(s[x := v]))\, d\mu_s$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$
$\mathtt{if}\ (\varphi)\ P\ \mathtt{else}\ Q$	$[\varphi] \cdot wp[\![P]\!](f) + [\neg\varphi] \cdot wp[\![Q]\!](f)$
$P\,[p]\,Q$	$p \cdot wp[\![P]\!](f) + (1-p) \cdot wp[\![Q]\!](f)$
$\mathtt{while}\ (\varphi)\ \{P\}$	$\mathrm{lfp}\, X.\ \underbrace{(([\varphi] \cdot wp[\![P]\!](X)) + [\neg\varphi] \cdot f)}_{\text{loop characteristic function } \Psi_f(X)}$

where lfp is the least fixed point wrt. the ordering $\sqsubseteq$ on $\mathbb{E}$.

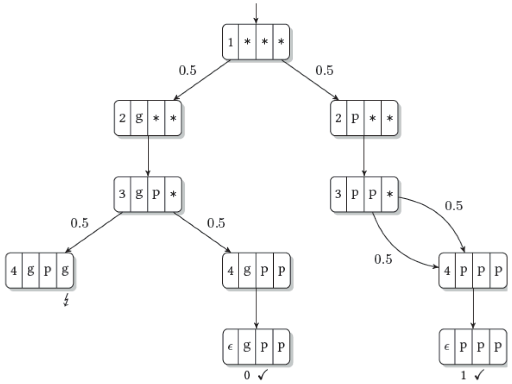
Normalisation?

The semantics so far treats `observe` as an `assert` statement.

It does not cover normalisation.

Flash back: 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

$$\mathrm{ER}^{[\![P]\!]}(\sigma_I, \Diamond\langle sink\rangle \mid \neg\Diamond\langle \lightning \rangle) = \frac{1 \cdot 1/2 + 0 \cdot 1/4}{1 - 1/4} = \frac{1/2}{3/4} = 2/3.$$

The piranha program – a wp perspective

```
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?

$$\mathbb{E}(\texttt{f1 = pir} \mid \text{“feasible” run}) = \frac{1 \cdot 1/2 + 0 \cdot 1/4}{1 - 1/4} = \frac{1/2}{3/4} = \frac{2}{3}.$$

Let $cwp[\![P]\!](f) = \dfrac{wp[\![P]\!](f)}{wlp[\![P]\!](\mathbf{1})}$. We will define: $\boxed{cwp[\![P]\!](f) = (wp[\![P]\!](f), wlp[\![P]\!](\mathbf{1}))}$.

Note: $wlp[\![P]\!](\mathbf{1}) = 1 - Pr[P$ violates an observation$]$. This includes diverging runs.

Overview

Conditional expectations

Conditional expectations

A conditional expectation is a pair (f, g) with $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leqslant 1}$.

Let $\mathbb{C} = \mathbb{E} \times \mathbb{E}_{\leqslant 1}$ denote the set of conditional expectations.

$(f, g) \in \mathbb{C}$ represents the fraction $\frac{f}{g}$.

$$(f, g) \text{ is interpreted (in the end) as } \lambda s. \begin{cases} \dfrac{f(s)}{g(s)} & \text{if } g(s) \neq 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Beware: $(\mathbf{1}, \mathbf{1}) \neq (1/2, 1/2)$, and $(f, \mathbf{0})$ is a well-formed conditional expectation.

A partial order on conditional expectations

Let $\trianglelefteq \subseteq \mathbb{C} \times \mathbb{C}$ be defined by:

$$(f, g) \trianglelefteq (f', g') \text{ if and only if } f \sqsubseteq f' \text{ and } g \sqsupseteq g'.$$

The "fractional interpretation": $(f, g) \trianglelefteq (f', g')$ implies $\frac{f(s)}{g(s)} \leqslant \frac{f'(s)}{g'(s)}$.

$(\mathbb{C}, \trianglelefteq)$ is a complete lattice.

Proof.

Straightforward. The least element is $(\mathbf{0}, \mathbf{1})$ and the greatest element is $(\infty, \mathbf{0})$. The supremum of a subset S in $\mathbb{C}$ is given point-wise by the pair:

$$\sup_{\trianglelefteq} S = \left(\sup_{\leqslant} \{ f \mid (f, g) \in S \}, \inf_{\leqslant} \{ g \mid (f, g) \in S \} \right).$$

$\square$

Overview

Conditional weakest preconditions for cpGCL

$$cwp[\![P]\!](f) \;=\; \underbrace{(wp[\![P]\!](f), wlp[\![P]\!](\mathbf{1}))}_{\text{conditional expectation}}$$

Note: $wlp[\![P]\!](\mathbf{1}) = 1 - Pr[P \text{ violates an observation}]$.

This includes diverging runs.

Finally interpret this as $\dfrac{wp[\![P]\!](f)}{wlp[\![P]\!](\mathbf{1})}$ provided $wlp[\![P]\!](\mathbf{1}) \neq \mathbf{0}$

Example: the piranha problem

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.

Observations inside loops

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

- Certain divergence
- $(wp[\![P_1]\!](f), wlp[\![P_1]\!](\mathbf{1})) = (\mathbf{0}, \mathbf{1})$
- Conditional wp $= \frac{\mathbf{0}}{\mathbf{1}} = 0$

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

- Divergence with probability zero
- $(wp[\![P_2]\!](f), wlp[\![P_2]\!](\mathbf{1})) = (\mathbf{0}, \mathbf{0})$
- Conditional wp $= \frac{\mathbf{0}}{\mathbf{0}} =$ undefined

The cwp-semantics does distinguish these programs.
Note that: $wp[\![P_1]\!](\mathbf{1}) = wp[\![P_2]\!](\mathbf{1}) = \mathbf{0}$

Feasibility

Recall feasibility for probabilistic wp w/o conditioning.

Feasibility of conditional wp

For cpGCL program P, $f \in \mathbb{E}$ and $g \in \mathbb{E}_{\leqslant 1}$, it holds:

$$\forall s \in \mathbb{S}. \, g(s) > 0 \;\Rightarrow\; \frac{f(s)}{g(s)} \leqslant k \quad \text{and} \quad cwp[\![P]\!]((f, g)) = (f', g')$$

$$\text{implies} \quad (\forall s \in \mathbb{S}. \, g'(s) > 0 \;\Rightarrow\; f'(s) \leqslant k).$$

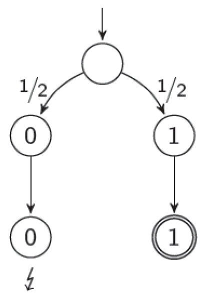
Proof.

By structural induction on P. The non-trivial case is probabilistic choice. □

Compositionality?

$$P: \quad \{x := 0\} \; [1/2] \; \{x := 1\}; \; observe(x = 1)$$

$$Q: \quad \{x := 0; \; observe(x = 1)\} \; [1/2] \; \{x := 1; \; observe(x = 1)\}$$



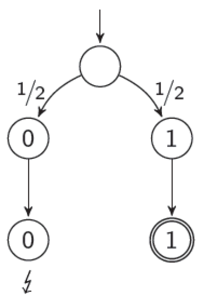
Of course

$$\frac{wp(P, [x = 1])}{wlp(P, 1)} = \frac{wp(Q, [x = 1])}{wlp(Q, 1)} = \frac{1/2}{1/2} = 1$$

Compositionality?

$P: \quad \{x := 0\}\ [1/2]\ \{x := 1\};\ \mathit{observe}(x = 1)$

$Q: \quad \underbrace{\{x := 0;\ \mathit{observe}(x = 1)\}}_{Q_1}\ [1/2]\ \underbrace{\{x := 1;\ \mathit{observe}(x = 1)\}}_{Q_2}$



Of course

$$\frac{wp(P, [x = 1])}{wlp(P, 1)} = \frac{wp(Q, [x = 1])}{wlp(Q, 1)} = \frac{1/2}{1/2} = 1$$

but we cannot decompose

$$\frac{wp(Q, [x = 1])}{wlp(Q, 1)} \neq 0.5 \frac{wp(Q_1, [x = 1])}{wlp(Q_1, 1)} + 0.5 \frac{wp(Q_2, [x = 1])}{wlp(Q_2, 1)}$$

This all motivates that we deal with pairs rather than fractions.

Overview

1. Conditioning
2. Observe statements in w(l)p
3. Conditional expectations
4. Conditional weakest preconditions
5. **Program transformations**
6. Compatibility results

Program transformation: removal of 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)` $\mapsto$ `flag := !G || flag`
  - `while(G) prog` $\mapsto$ `while(G && !flag) prog`

Resulting program

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

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)
```


Removal of conditioning

Correctness of transformation

For conditional pGCL program P that has at least one feasible run and expectation f:

$$cwp[\![P]\!](f, \mathbf{1}) \;=\; wp[\![\widehat{P}]\!](f).$$

where $\widehat{P}$ is the result of replacing conditioning in P by a loop.

Proof: straightforward by structural induction on P.

Remark

Due to this result, observe-statements are equivalent to loops.

They are thus syntactic sugar.

But: they are practically very handy and

do not require loop invariants or fixed points.

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

Independent and identically distributed loops

iid-Loop

Loop `while` $(\varphi)\{P\}$ is iid if and only if for any expectation f:

$$wp[\![P]\!]([\varphi] \cdot wp[\![P]\!](f)) \quad = \quad wp[\![P]\!]([\varphi]) \cdot wp[\![P]\!](f)$$

Event that φ holds after P is independent of the expected value of f after P.

Correctness of transformation

For iid-loop `repeat P until` (φ) and expectations f, g we have:

$$cwp[\![\texttt{repeat P until } (\varphi)]\!]((f,g)) \quad = \quad cwp[\![\texttt{P ; observe } (\varphi)]\!]((f,g))$$

Loop-free programs are easier to reason about — no loop invariants.

An alternative program transformation: Hoisting

This transformation "pushes" `observe`-statements "to the top".

Hoisting **[Nori et al., AAAI 2014]**

$$\begin{array}{rcl}
T(\texttt{skip}, f) & = & (\texttt{skip}, f) \\
T(x := E, f) & = & (x := E, f[x := E]) \\
T(\texttt{observe}(\varphi), f) & = & (\texttt{skip}, [\varphi] \cdot f) \\
T(P_1; P_2, f) & = & (Q_1; Q_2, h) \text{ where } (Q_2, g) = T(P_2, f) \\
 & & \text{ and } (Q_1, h) = T(P_1, g) \\
T(\texttt{if}(\varphi)\, P_1 \texttt{ else } P_2, f) & = & (\texttt{if}(\varphi)\, Q_1 \texttt{ else } Q_2, [\varphi]\cdot g + [\neg\varphi]\cdot h) \text{ where} \\
 & & (Q_1, g) = T(P_1, f) \text{ and } (Q_2, h) = T(P_2, f) \\
T(P_1[p]P_2, f) & = & (Q_1[q]Q_2, p\cdot g + (1-p)\cdot h) \text{ where } (Q_1, g) = T(P_1, f) \\
 & & \text{ and } (Q_2, h) = T(P_2, f) \text{ and } q = \frac{p\cdot g}{p\cdot g + (1-p)\cdot h} \\
T(\texttt{while}(\varphi)P, f) & = & (\texttt{while}(\varphi)Q, g) \text{ where } g = \text{gfp}\, H \text{ with} \\
 & & H(h) = [\varphi]\cdot(\pi_2 \odot T)(P, h) + [\neg\varphi]\cdot f \\
 & & \text{ and } (Q, -) = T(P, g)
\end{array}$$

Correctness of hoisting

Correctness of hoisting

For any conditional pGCL program P with at least one feasible run and $f \in \mathbb{E}$:

$$cwp[\![P]\!]((f, \mathbf{1})) \;=\; wp[\![Q]\!](f) \quad \text{with} \quad T(P, \mathbf{1}) = (Q, h).$$

The component h represents the probability that P satisfies all its observe-statements.

Overview

1. Conditioning
2. Observe statements in w(l)p
3. Conditional expectations
4. Conditional weakest preconditions
5. Program transformations
6. Compatibility results

Backward compatibility

We have seen earlier:

McIver's wp-semantics is a conservative extension of Dijkstra's wp-semantics.

For any ordinary (aka: GCL) program P and predicate F:

$$\underbrace{wp[\![P]\!]([F])}_{\text{McIver}} \;=\; \underbrace{[\,wp[\![P]\!](F)\,]}_{\text{Dijkstra}}$$

The cwp-semantics is a conservative extension of McIver's wp-semantics.

For any observe-free pGCL program P and expectation f:

$$cwp[\![P]\!]((f,\mathbf{1})) \;=\; (f',g') \text{ implies } \frac{f'}{g'} \;=\; wp[\![P]\!](f).$$

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:

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

The conditional expected reward until reaching G while avoiding $F \subseteq \Sigma$ is:

$$\mathrm{CER}(\sigma, \Diamond G \mid \neg\Diamond F) \;=\; \frac{\mathrm{ER}(\sigma, \Diamond G \cap \neg\Diamond F)}{Pr(\sigma \models \neg\Diamond F)}$$

[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.

Conditional wp = conditional expected rewards

Compatibility theorem for conditional wp

For cpGCL program P, input s and expectation f:

$$\underbrace{\frac{wp[\![P]\!](f)(s)}{wlp[\![P]\!](\mathbf{1})(s)}}_{\text{conditional wp of } P} \;=\; \underbrace{\mathrm{CER}^{[\![P]\!]}\big(s, (\Diamond\langle sink\rangle \mid \neg\Diamond\langle \lightning \rangle)\big)}_{\text{conditional expected reward in MC } [\![P]\!]}\;.$$

The ratio of $wp[\![P]\!](f)$ over $wlp[\![P]\!](\mathbf{1})$ for input s equals[2] the conditional expected reward to reach the terminal state $\langle sink\rangle$ while satisfying all observations in P's MC when starting with s. (The rewards in MC $[\![P]\!]$ are defined as before.)

[2] Either both sides are equal or both sides are undefined.

Take-home messages

- Conditioning changes the probability distribution
- Conditioning is semantically treated in two steps:
  1. A simple extension of weakest preconditions with observe
  2. Conditional expectations: pairs of weakest (liberal) preconditions
- $cwp[\![P]\!]((f, \mathbf{1})) \;=\; \dfrac{wp[\![P]\!](f)}{wlp[\![P]\!](\mathbf{1})}$ provided $wlp[\![P]\!](\mathbf{1}) \neq \mathbf{0}$
- Conditioning can be removed at the expense of a loop
- Or, can be pushed backwards through the program

Next lecture: recursion theory

Next lecture

Thursday Dec 1, 16:30