Probabilistic Programming

Lecture #11: Conditional Weakest Preconditions

Joost-Pieter Katoen

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

RWTH Lecture Series on Probabilistic Programming 2018

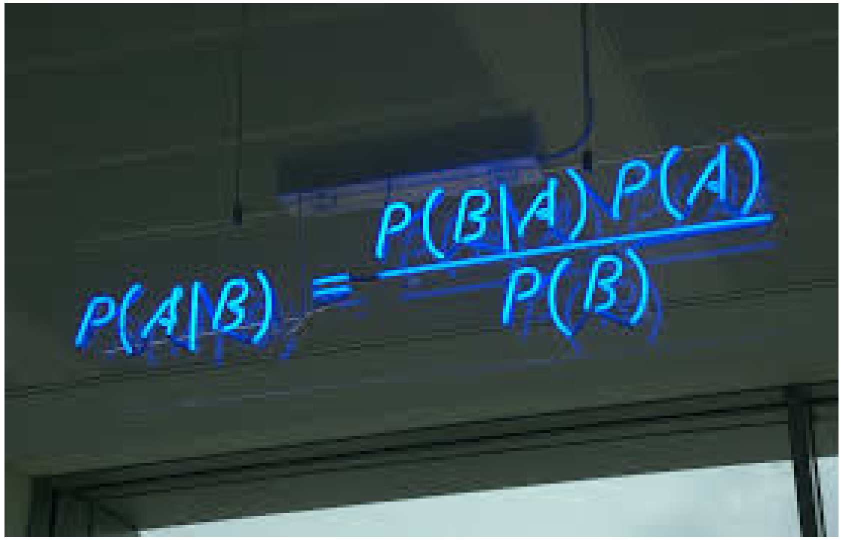
Overview

1. A short recap of conditioning
2. Extending weakest pre-expectations
3. Normalisation
4. Compatibility results
5. Program transformations

Overview

1. A short recap of conditioning
2. Extending weakest pre-expectations
3. Normalisation
4. Compatibility results
5. Program transformations

Bayes' rule



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:

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

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

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?

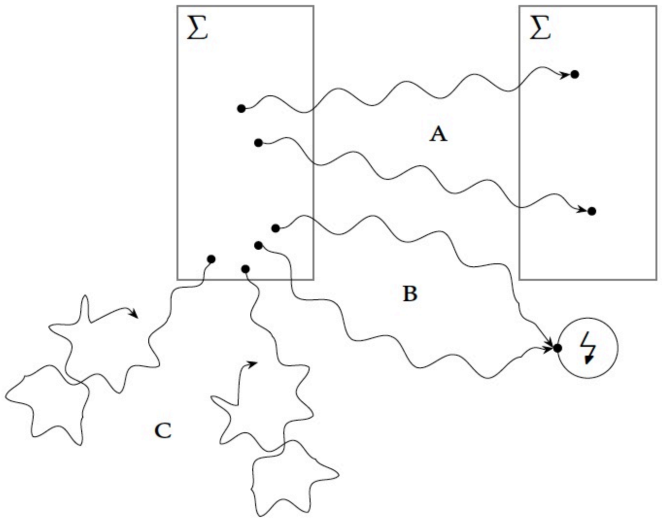
A: $\frac{2}{7}$. Why?

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

Overview

1. A short recap of conditioning
2. Extending weakest pre-expectations
3. Normalisation
4. Compatibility results
5. Program transformations

Possible outcomes of a cpGCL program



Expectations

Expectations

A expectation[1] (read: random variable) f maps program states onto non-negative reals extended with infinity, i.e., $f : \mathbb{S} \rightarrow \mathbb{R}_{\geq 0} \cup \{ \infty \}$.

Let $\mathbb{E}$ denote the set of all expectations and let $\sqsubseteq$ be defined for $f, g \in \mathbb{E}$ by:

$$f \sqsubseteq g \quad \text{if and only if} \quad f(s) \leq g(s) \quad \text{for all } s \in \mathbb{S}.$$

$(\mathbb{E}, \sqsubseteq)$ is a complete lattice.

[1] ≠ expectations in probability theory.

Weakest pre-expectations

Weakest precondition

For probabilistic program P and $e, f \in \mathbb{E}$, the expectation transformer $wp(P, \cdot) : \mathbb{E} \rightarrow \mathbb{E}$ is defined by $wp(P, f) = e$ iff e maps each (initial) state s to the expected value of f after executing P on s.

The characterising equation of a weakest pre-expectation is given by:

$$wp(P, f) \;=\; \lambda s. \int_{\mathbb{S}} f \, dP_s$$

where P_s is the distribution over the final states (reached on termination of P) when executing P on the initial state s.

Bounded expectations

Bounded expectations

The set of (one-)bounded expectations, denoted $\mathbb{E}_{\leq 1}$ is defined as:

$$\mathbb{E}_{\leq 1} \;=\; \{ f \in \mathbb{E} \mid f \sqsubseteq \mathbf{1} \}$$

$(\mathbb{E}_{\leq 1}, \sqsubseteq)$ is a complete lattice.

Proof.

Left as an exercise. The least element is $\lambda s.0$; the greatest element is $\lambda s.1$ and suprema are defined as for $\mathbb{E}$. □

Weakest liberal pre-expectations

Weakest liberal pre-expectation

For probabilistic program P and $e, f \in \mathbb{E}_{\leq 1}$, the expectation transformer $wlp(P, \cdot) : \mathbb{E}_{\leq 1} \rightarrow \mathbb{E}_{\leq 1}$ is defined by $wlp(P, f) = e$ such that e equals the expected value of f after executing P on s plus the probability that P diverges on s.

The characterising equation of a weakest liberal pre-expectation is given by:

$$wlp(P, f) \;=\; \lambda s. \int_{\mathbb{S}} f \, dP_s + \left(1 - \int_{\mathbb{S}} 1 \, dP_s \right)$$

where P_s is the distribution over the final states when executing P (reached on termination) on the initial state s.

Weakest liberal pre-expectation $wlp(P, f) \;=\;$ "$wp(P, f) + Pr[P \text{ diverges}]$".

Extending wp with conditioning

Syntax

- `skip`
- `diverge`
- `x := E`
- `observe (G)`
- `x :≈` μ
- `P1 ; P2`
- `if (G) P1 else P2`
- `P1 [p] P2`
- `while (G)P`

Semantics $wp(P, f)$

- f
- 0
- $f[x := E]$
- $[G] \cdot f$
- $\lambda s. \int_{\mathbb{Q}} (\lambda v. f(s[x := v]))\, d\mu_s$
- $wp(P_1, wp(P_2, f))$
- $[G] \cdot wp(P_1, f) + [\neg G] \cdot wp(P_2, f)$
- $p \cdot wp(P_1, f) + (1-p) \cdot wp(P_2, f)$
- $\text{lfp}\ X.\ ([G] \cdot wp(P, X) + [\neg G] \cdot f)$

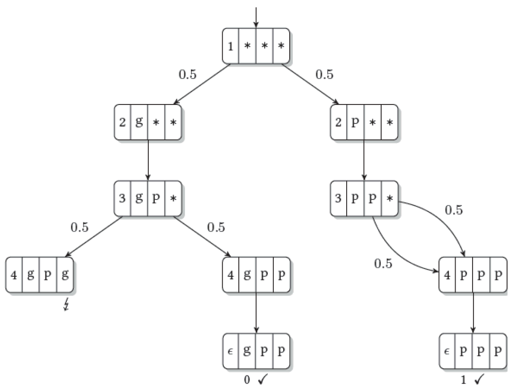
The wlp-semantics of pGCL can be extended analogously.
Normalisation is to be next. It is not covered here.

Overview

1. A short recap of conditioning
2. Extending weakest pre-expectations
3. Normalisation
4. Compatibility results
5. Program transformations

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

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

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 $cwp(P, f) = \dfrac{wp(P, f)}{wlp(P, \mathbf{1})}$. In fact $cwp(P, f) = (wp(P, f), wlp(P, \mathbf{1}))$.

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

Conditional expectations

Conditional expectations

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

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

$(f, g) \in \mathbb{C}$ represents the fraction $\frac{f}{g}$.
Beware: $(\mathbf{1}, \mathbf{1}) \neq (1/2, 1/2)$, and $(f, \mathbf{0})$ is a well-defined conditional expectation.

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

A partial order on conditional expectations

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

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

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

$(\mathbb{C}, \trianglelefteq)$ 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:

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

Operations on conditional expectations

- For $(f, g) \in \mathbb{C}$ and $c \in \mathbb{R}_{\geq 0}$, let $(c \cdot (f, g))(s) = (c \cdot f(s), c \cdot g(s))$
- For $(f, g), (f', g') \in \mathbb{C}$, let $(f, g) + (f', g') = (f + f', g + g')$.
- Multiplication and subtraction are defined analogously.
- For $(f, g) \in \mathbb{C}$, let $\pi_1(f, g) = f$ and $\pi_2(f, g) = g$.

Conditional weakest preconditions for `cpGCL`

Syntax	Semantics $cwp(P, f)$
`skip`	
`diverge`	
`x := E`	
`observe (G)`	
`x :≈ μ`	
`P1 ; P2`	
`if (G)P1 else P2`	
`P1 [p] P2`	
`while (G)P`	

Examples

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

These programs are mostly not distinguished as $wp(P_{left}, \mathbf{1}) = wp(P_{right}, \mathbf{1}) = \mathbf{0}$

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

- Certain divergence
- $(wp(P_{left}, f), wlp(P_{left}, \mathbf{1})) = (\mathbf{0}, \mathbf{1})$
- Conditional wp $= 0$

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

- Divergence with probability zero
- $\big(wp(P_{right}, f), wlp(P_{right}, \mathbf{1})\big) = (\mathbf{0}, \mathbf{0})$
- Conditional wp $=$ undefined

Our semantics do distinguish these programs.

Elementary properties of conditional wp

- Continuity: $cwp(P, z)$ is continuous on $(\mathbb{C}, \trianglelefteq)$
- Monotonicity: $z \trianglelefteq z'$ implies $cwp(P, z) \trianglelefteq cwp(P, z')$
- Decoupling: $cwp(P, (f, g)) \;=\; (wp(P, f), wlp(P, g))$
- Linearity: $cwp(P, (r \cdot f + g, g')) \;=\; \big(r \cdot wp(P, f) + wp(P, g), wlp(P, g')\big)$
- Strictness: $cwp(P, (\mathbf{0}, \mathbf{1})) = (\mathbf{0}, g)$ where $g = wlp(P, \mathbf{1})$

Feasibility

Feasibility of conditional wp

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

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

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

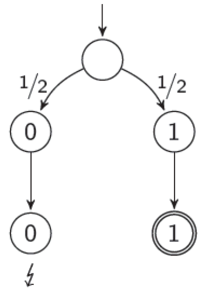
Proof.

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

Contextual equivalence?

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

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



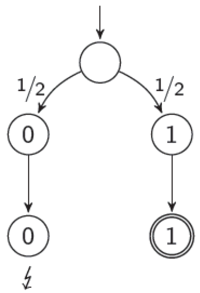
Of course

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

Contextual equivalence?

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

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



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. A short recap of conditioning
2. Extending weakest pre-expectations
3. Normalisation
4. Compatibility results
5. Program transformations

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 wp = conditional expected rewards

Compatibility theorem for conditional wp

For program P, input s and expectation f:

$$\frac{wp(P,f)(s)}{wlp(P,\mathbf{1})(s)} \;=\; ER^{[\![P]\!]}\big(s, (\Diamond\langle sink\rangle \mid \neg\Diamond\langle \lightning\rangle)\,\big)$$

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.)

For finite-state programs, conditional wp-reasoning can be done with model checkers such as PRISM and Storm (www.stormchecker.org).

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

Overview

1. A short recap of conditioning
2. Extending weakest pre-expectations
3. Normalisation
4. Compatibility results
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)` ⇝ `flag := !G || flag`
  - `abort` ⇝ `if(!flag) abort`
  - `while(G) prog` ⇝ `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)
```


Removal of conditioning

Correctness of transformation

For cpGCL 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 removing conditioning from 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 (G)P` is iid if and only if for any expectation f:

$$wp(P, [G] \cdot wp(P, f)) \quad = \quad wp(P, [G]) \cdot wp(P, f)$$

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

Correctness of transformation

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

$$cwp(\texttt{repeat P until (G)}, (f, g)) \quad = \quad cwp(\texttt{P ; observe (G)}, (f, g))$$

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

A third program transformation: Hoisting

Hoisting [Nori et al., 2014]

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

Correctness of hoisting

Correctness of hoisting

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

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

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