Probabilistic Programming

Lecture #11: Reasoning About Loops

Joost-Pieter Katoen

RWTH Lecture Series on Probabilistic Programming 2022-23

Overview

Overview

Motivation

- Reasoning about loops is the hardest task in program verification
- Why?
  - Weakest preconditions of loops are defined as fixed points
  - They can be approximated iteratively
  - But: Recognise a pattern to yield a closed-form formula for taking a loop k times
  - Taking the limit yields the required fixed point

  These last two steps are the source of undecidability
- "Practical" approach: capture the effect of a loop by a loop invariant

A loop invariant is a property of a program loop
that is true before (and after) each loop iteration.

Overview

Loop invariants à la Dijkstra

Recall that for while-loops we have for $F \in \mathbb{P}$:

$$wlp[\![\text{while}(\varphi)\{P\}]\!](F) \;=\; \text{gfp}\, X.\, (\varphi \wedge wlp[\![P]\!](X) \vee (\neg\varphi \wedge F))$$

To determine the effect of a while-loop, one exploits an "invariant" $I \in \mathbb{P}$

Loop invariant

Predicate $I \in \mathbb{P}$ is a loop invariant w.r.t. postcondition $F \in \mathbb{P}$ if it satisfies:

1. $\varphi \Rightarrow I$
2. $(\neg\varphi \wedge I) \Rightarrow F$, and
3. $(\varphi \wedge I) \Rightarrow wlp[\![P]\!](I)$.

Satisfaction of I is invariant under (guarded) iteration of the loop body P.

Example

Characteristic functions for probabilistic loops

Let P be a probabilistic program in pGCL

Recall for expectation $f \in \mathbb{E}$:

$$wp[\![\text{while } (\varphi)\,\{\,P\,\}]\!](f) \;=\; \text{lfp}\, X.\;\underbrace{([\varphi]\cdot wp[\![P]\!](X) + [\neg\varphi]\cdot f)}_{\text{characteristic function } \Phi_f(X) \text{ for wp}}$$

and

$$wlp[\![\text{while } (\varphi)\,\{\,P\,\}]\!](f) \;=\; \text{gfp}\, X.\;\underbrace{([\varphi]\cdot wlp[\![P]\!](X) + [\neg\varphi]\cdot f)}_{\text{characteristic function } \Psi_f(X) \text{ for wlp}}$$

Loop invariants à la Dijkstra for pGCL

For $I, F \in \mathbb{P}$ and probabilistic loop while$(\varphi)\{P\}$ it holds:

$$((\neg\varphi \wedge I) \Rightarrow F \text{ and } (\varphi \wedge I) \Rightarrow wlp[\![P]\!](I)) \quad \text{iff} \quad [I] \sqsubseteq \Psi_{[F]}([I])$$

where $\Psi_{[F]}$ is the wlp-characteristic function of the probabilistic loop for postcondition $[F]$.

Proof.
On the black board. □

Overview

Probabilistic invariants

Let Φ_f be the wp-characteristic function of $P' = \text{while}(\varphi)\{P\}$ with respect to post-expectation $f \in \mathbb{E}$ and let $I \in \mathbb{E}$. Then:

1. I is a wp-superinvariant of P' w.r.t. f iff $\Phi_f(I) \sqsubseteq I$.
2. I is a wp-subinvariant of P' w.r.t. f iff $I \sqsubseteq \Phi_f(I)$.

Sub- and superinvariants for wlp are defined analogously (but are bounded, i.e., then $I \in \mathbb{E}_{\leqslant 1}$, and Φ_f is replaced by Ψ_f.)

Lemma
$[\varphi] \cdot I \sqsubseteq wp[\![P]\!](I)$ iff $I \sqsubseteq \Phi_{[\neg\varphi] \cdot I}(I)$ for all $I \in \mathbb{E}$ and pGCL program P.

Proof.
Left as an exercise. □

Duelling cowboys: when does A win?

```
int cbDuel(float a, b) {
  int t := A;  // cowboy A starts
  int c := 1;
  while (c = 1) {
    if (t = A) {
      (c := 0 [a] t := B);
    } else {
      (c := 0 [b] t := A);
    }
  }
  return t;
}
```

Probabilistic loop invariant w.r.t. postcondition $[t = A]$

$$I = [t = A \wedge c = 0] \cdot \mathbf{1} + [t = A \wedge c = 1] \cdot \frac{a}{a + b - a \cdot b} + [t = B \wedge c = 1] \cdot \frac{(1-b)a}{a + b - a \cdot b}$$

Overview

Induction for upper bounds on wp

Recall:

$$wp[\![\text{while } (\varphi)\{ P \}]\!](f) \;=\; \text{lfp}\, X.\, \underbrace{([\varphi]\cdot wp[\![P]\!](X) + [\neg\varphi]\cdot f)}_{\Phi_f(X)}$$

Park's lemma: let $(D, \sqsubseteq)$ be a complete lattice and $\Phi : D \to D$ continuous. Then:

$$\forall d \in D. \quad \Phi(d) \sqsubseteq d \quad \text{implies} \quad \text{lfp}\, \Phi \sqsubseteq d.$$

Corollary: upper bounds on weakest pre-expectations

For while$(\varphi)\{P\}$ and expectations f and I we have:

$$\underbrace{\Phi_f(I) \sqsubseteq I}_{\text{wp-superinvariant}} \quad \text{implies} \quad \underbrace{wp[\![\text{while}(\varphi)\{P\}]\!](f)}_{\text{lfp}\,\Phi_f} \sqsubseteq I$$

Pictorially

Example

while(c = 0) { x++ [p] c := 1 }

Claim: $I \;=\; x + [c = 0] \cdot \dfrac{p}{1-p}$ is a super-invariant of P w.r.t. $f = x$.

Proof

Induction for lower bounds on wlp

Recall:

$$wlp[\![\text{while } (\varphi)\{P\}]\!](f) \;=\; \text{gfp}\, X.\, \underbrace{([\varphi]\cdot wlp[\![P]\!](X) + [\neg\varphi]\cdot f)}_{\Psi_f(X)}$$

Park's lemma: let $(D, \sqsubseteq)$ be a complete lattice and $\Psi : D \to D$ continuous. Then:

$$\forall d \in D. \quad d \sqsubseteq \Psi(d) \quad \text{implies} \quad d \sqsubseteq \text{gfp}\, \Psi.$$

Corollary: lower bounds on weakest liberal pre-expectations

For while$(\varphi)\{P\}$ and bounded expectations f and I we have:

$$\underbrace{I \sqsubseteq \Psi_f(I)}_{\text{wlp-subinvariant}} \quad \text{implies} \quad I \sqsubseteq \underbrace{wlp[\![\text{while}(\varphi)\{P\}]\!](f)}_{\text{gfp}\,\Psi_f}$$

Verifying loops

The following procedure for induction (and co-induction):

1. Guess an appropriate loop invariant I
2. Push I through the characteristic function of the loop once, i.e., compute $\Phi(I)$
3. Check whether this took us down or up in the partial order $\leqslant$:
   - 3.1 $\Phi(I) \sqsubseteq I$, for induction (upper bound to wp), or
   - 3.2 $I \sqsubseteq \Psi(I)$, for co-induction (lower bound to wlp).

The key difficulty is to find an appropriate invariant I. This is undecidable.

Overview

Aiming for lower bounds on wp (= lfp)

The following result does not hold:

$$I \sqsubseteq \Phi_f(I) \quad \textit{implies} \quad I \sqsubseteq \text{lfp}\,\Phi_f$$

Pictorially:

Counterexample

Conclusion

Let $(D, \sqsubseteq)$ be a complete lattice and $\Phi : D \to D$ continuous. Then

$$\forall d \in D. \quad d \sqsubseteq \Phi(d) \quad \text{does not imply} \quad d \sqsubseteq \text{lfp}\,\Phi.$$

Co-induction for lower bounds on wp of loops is unsound.

Induction on upper bounds on wlp of loops also fails:

$$\forall d \in D. \quad \Phi(d) \sqsubseteq d \quad \text{does not imply} \quad \text{gfp}\,\Phi \sqsubseteq d.$$

Induction for upper bounds on wlp of loops is unsound.

A proof rule for lower bounds on wp

Proof rule for lower bounds

$$\big(I \sqsubseteq \Phi_f(I) \wedge \text{side conditions}\big) \quad \textit{implies} \quad I \sqsubseteq \text{lfp}\,\Phi_f$$

where the side conditions for pGCL program while$(\varphi)\{P\}$ are:

1. while$(\varphi)\{P\}$ terminates in finite expected time, and
2. for any $s \models \varphi$, $\underbrace{wp[\![P]\!](|I(s) - I|)(s) \leqslant c}_{\text{conditional difference boundedness}}$ for some given $c \in \mathbb{R}_{\geqslant 0}$.

Example

A simpler proof rule for lower bounds

Guard strengthening for lower bounds

Let $P_{loop} = \text{while}(\varphi)\{P\}$ and $P'_{loop} = \text{while}(\varphi')\{P\}$, and expectations f and I. Then it holds:

$$\left(\varphi' \Rightarrow \varphi \;\wedge\; I \sqsubseteq \underbrace{wp[\![P'_{loop}]\!]([\neg\varphi]\cdot f)}_{\text{lfp}\,\Phi'_{[\neg\varphi]\cdot f}}\right) \quad \text{implies} \quad I \sqsubseteq \underbrace{wp[\![P_{loop}]\!](f)}_{\text{lfp}\,\Phi_f}$$

- This rule is more general (e.g., applicable to divergent loops)
- The tightness of the lower bound depends on φ' approximating φ
- Algorithmically in case P'_{loop} has finitely many states

Proof

Random walks

Let $\oplus$ abbreviate a uniform distribution. 1D-symmetric random walk on $\mathbb{Z}$:

$$\text{while } (x \neq 0) \; \{ \; x := x{+}1 \;\oplus\; x := x{-}1 \}$$

2D-symmetric random walk on $\mathbb{Z}^2$:

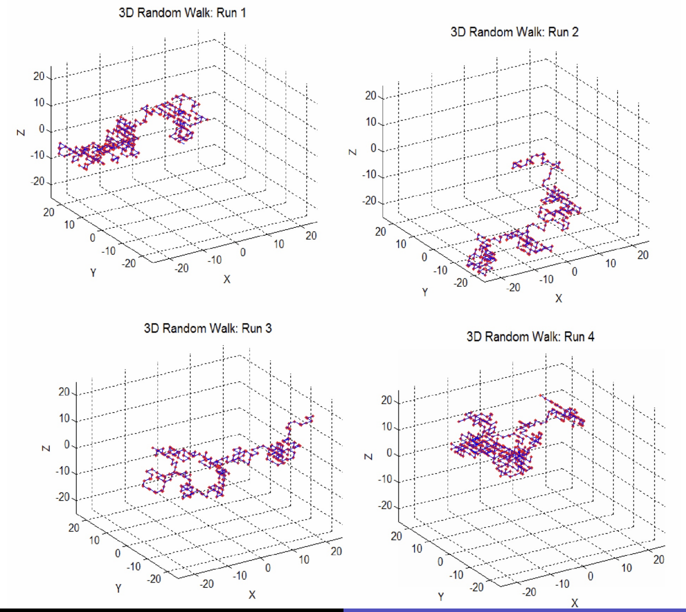
$$\text{while } (x \neq 0 \vee y \neq 0) \; \{$$
$$x := x{+}1 \;\oplus\; x := x{-}1 \;\oplus\; y := y{+}1 \;\oplus\; y := y{-}1 \;\}$$

3D-symmetric random walk on $\mathbb{Z}^3$:

$$\text{while } (x \neq 0 \vee y \neq 0 \vee z \neq 0) \; \{$$
$$x := x{+}1 \;\oplus\; x := x{-}1 \;\oplus$$
$$y := y{+}1 \;\oplus\; y := y{-}1 \;\oplus$$
$$z := z{+}1 \;\oplus\; z := z{-}1 \;\}$$

The 1D and 2D random walks reach the origin with probability one.
The 3D random walks does not.

Example: 3D symmetric random walk on $\mathbb{Z}^3$



Example: 3D symmetric random walk on $\mathbb{Z}^3$

George Pólya (1887-1985)

Pólya's analysis result

The termination probability of the 3D symmetric random walk starting from any neighbour location of the origin $(0,0,0)$ equals

$$1 - \left(\frac{3}{(2\pi)^3} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \frac{dx\, dy\, dz}{3 - \cos x - \cos y - \cos z} \right)^{-1} = 0.3405373296\ldots$$

Can we obtain a tight lower bound on the termination probability?

Obtaining a lower bound

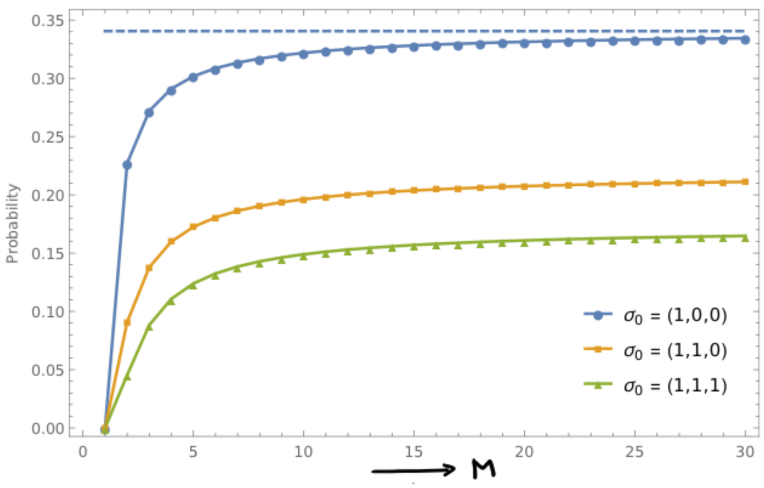
$$\begin{array}{l} \text{while } (x \neq 0 \vee y \neq 0 \vee z \neq 0) \ \{ \\ \quad x := x{+}1 \ \oplus \ x := x{-}1 \ \oplus \\ \quad y := y{+}1 \ \oplus \ y := y{-}1 \ \oplus \\ \quad z := z{+}1 \ \oplus \ z := z{-}1 \ \} \end{array}$$

Bound the positions (x, y, z) to a cube of side length $2{\cdot}M$ for $M \in \mathbb{N}_{>0}$:

$$\begin{array}{l} \text{while } ((x \neq 0 \vee y \neq 0 \vee z \neq 0) \wedge |x| \leqslant M \wedge |y| \leqslant M \wedge |z| \leqslant M) \ \{ \\ \quad x := x{+}1 \ \oplus \ x := x{-}1 \ \oplus \\ \quad y := y{+}1 \ \oplus \ y := y{-}1 \ \oplus \\ \quad z := z{+}1 \ \oplus \ z := z{-}1 \ \} \end{array}$$

As the resulting program P' is finite state, $wp[\![P']\!](\mathbf{1})$ can be computed algorithmically. By increasing M, we aproximate Pólya's result arbitrarily closely.

Verification results



Overview

Playing with geometric distributions

- X is a random variable, geometrically distributed with parameter p
- Y is a random variable, geometrically distributed with parameter q

Q: generate a sample x, say, according to the random variable $X - Y$

```
int XminY1(float p, q){ // 0 <= p, q <= 1
  int x := 0;
  bool flip := false;
  while (not flip) { // take a sample of X to increase x
    (x +:= 1 [p] flip := true);
  }
  flip := false;
  while (not flip) { // take a sample of Y to decrease x
    (x -:= 1 [q] flip := true);
  }
  return x; // a sample of X-Y
}
```


An alternative program

```
int XminY2(float p, q){
  int x := 0;
  bool flip := false;
  (flip := false [0.5] flip := true); // flip a fair coin
  if (not flip) {
    while (not flip) { // sample X to increase x
      (x +:= 1 [p] flip := true);
    }
  } else {
    flip := false; // reset flip
    while (not flip) { // sample Y to decrease x
      x -:= 1;
      (skip [q] flip := true);
    }
  }
return x; // a sample of X-Y
}
```

Program equivalence: $X - Y$

```
int XminY1(float p, q){
  int x, c := 0, 1;
  while (c) {
    (x +:= 1 [p] c := 0);
  }
  c := 1;
  while (c) {
    (x -:= 1 [q] c := 0);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x := 0;
  (c := 0 [0.5] c := 1);
  if (c) {
    while (c) {
      (x +:= 1 [p] c := 0);
    }
  } else {
    c := 1;
    while (c) {
      x -:= 1;
      (skip [q] c := 0);
    }
  }
return x;
}
```

For which p and q are the expected outcomes for $f = x$ equal?

Program equivalence: $X - Y$

```
int XminY1(float p, q){
  int x, f := 0, 0;
  while (f = 0) {
    (x++ [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x-- [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x++ [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x--;
      (skip [q] f := 1);
    }
  }
  return x;
}
```

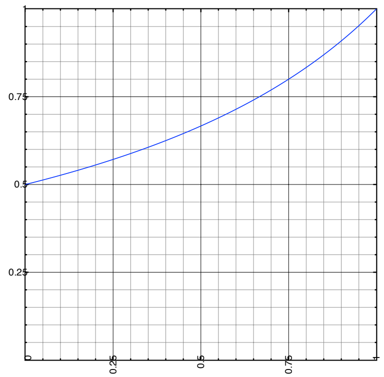
Using template $I = x + [f = 0] \cdot \alpha$ we find:

$\alpha_{11} = \frac{p}{1-p}$, $\alpha_{12} = -\frac{q}{1-q}$, $\alpha_{21} = \alpha_{11}$ and $\alpha_{22} = -\frac{1}{1-q}$.

Expected value of x is $\frac{p}{1 \ldots} - \frac{q}{1 \ldots}$ and $\frac{p}{2(1 \ldots)} - \frac{1}{2(1 \ldots)}$.

Program equivalence

Using wp, one can prove that the expectations of $f = x$ coincide if and only if $q = \frac{1}{2-p}$.



Take-home messages

- Loop invariants allow to reason about (unbounded) loops
- Probabilistic invariants are expectations
- That either are lower or upper bounds of loops
- Upper bounds can be obtained by Park's lemma
- Lower bounds can be obtained by strengthening the loop guard
- Or by difference bounded loop bodies and finite termination

Next lecture: conditioning

Next lecture

Thursday Nov 24, 16:30