Invariant Generation for Linear Probabilistic Programs

Joost-Pieter Katoen

Software Modeling and Verification Group
RWTH Aachen University

joint work with Larissa Meinicke, Annabelle McIver and Carroll Morgan

9. Juli 2010

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0;
  int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; // probabilistic choice
     n := n + 1;
  }
  return x;
}
```

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0;
  int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; // probabilistic choice
     n := n + 1;
  }
  return x;
}
```

Claim:

The value of x equals $k \in [0, N)$ according to a binomial distribution with parameter p.

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0;
  int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; // probabilistic choice
     n := n + 1;
  }
  return x;
}
```

Claim:

The value of $x = k$ with probability $\binom{N}{k} p^k \cdot (1-p)^{N-k}$.

Turning a fair coin into a biased one

```
int fair2biased(float p) { // 0 < p < 1
  int x := p;
  bool b := true;
  while (b) {
     b := false [1/2] true; // flip a coin
     if (b) {
          x := 2*x;
          if (x >= 1) x := x - 1; else skip;
     }
     else if (x >= 1/2) x := 1; else x := 0;
  }
  return x;
}
```

Turning a fair coin into a biased one

```
int fair2biased(float p) { // 0 < p < 1
  int x := p;
  bool b := true;
  while (b) {
     b := false [1/2] true; // flip a coin
     if (b) {
          x := 2*x;
          if (x >= 1) x := x - 1; else skip;
     }
     else if (x >= 1/2) x := 1; else x := 0;
  }
  return x;
}
```

Claim: [Hurd, 2002]

The value of x equals one with probability p.

Uniform distribution

```
int uniform(int N) {
  int n := 1;
  int g := N;
  while (g >= N) {
      g := 0;
      n := 1;
      while (n < N) {
          n := 2*n;
          g := 2*g [1/2] 2*g + 1; // flip a coin
      }
  }
  return g;
}
```

Uniform distribution

```
int uniform(int N) {
  int n := 1;
  int g := N;
  while (g >= N) {
      g := 0;
      n := 1;
      while (n < N) {
          n := 2*n;
          g := 2*g [1/2] 2*g + 1; // flip a coin
      }
  }
  return g;
}
```

Claim: [Chor et al., 1998]

The probability that $g = k$ for $k \in [0, N)$ equals $\frac{1}{N}$.

Correctness of probabilistic programs

Question:

How can we verify the correctness of such programs? In an automated way?

Correctness of probabilistic programs

Question:

How can we verify the correctness of such programs? In an automated way?

Apply model checking!

- Apply MDP model checking. LiQuor, PRISM
  - ⇒ works for program instances, but no general solution.
- Use abstraction-refinement techniques. PASS, PRISM
  - ⇒ loop analysis with real variables does not work well.
- Check language equivalence. APEX
  - ⇒ cannot deal with parameterised probabilistic programs.
- Apply parameterised probabilistic model checking. PARAM
  - ⇒ deals with fixed-sized probabilistic programs.

Correctness of probabilistic programs

Question:

How can we verify the correctness of such programs? In an automated way?

Apply deductive verification!

[McIver & Morgan]

- Use Floyd-Hoare style reasoning for probabilistic programs.
  - allowing for backward post- pre-condition reasoning.
- Quantitative loop invariants are pivotal to this approach.
  - . . . . . . but are much harder to find than qualitative loop invariants.
- Finding such loop invariants typically requires human ingenuity.

Correctness of probabilistic programs

Question:

How can we verify the correctness of such programs? In an automated way?

Our approach:

Automated loop-invariant generation for probabilistic programs.

Correctness of probabilistic programs

Question:

How can we verify the correctness of such programs? In an automated way?

Our approach:

Automated loop-invariant generation for probabilistic programs.

Our achievement:

A sound and complete constraint-based method for generating linear quantitative invariants for linear probabilistic programs with real-valued variables.

Qualitative loop invariants

Weakest liberal precondition [Dijkstra 1976]

Let P and Q boolean predicates over program variables in `prog`. Then:

$$\{ P \} \; prog \; \{ Q \} \quad \text{or} \quad P \Rightarrow wlp(prog, Q)$$

denotes that: whenever the precondition P holds before the execution of `prog`, the postcondition Q holds after provided `prog` terminates.

Qualitative loop invariants

Weakest liberal precondition [Dijkstra 1976]

Let P and Q boolean predicates over program variables in `prog`. Then:

$$\{ P \} \; prog \; \{ Q \} \quad \text{or} \quad P \Rightarrow wlp(prog, Q)$$

denotes that: whenever the precondition P holds before the execution of `prog`, the postcondition Q holds after provided `prog` terminates.

Loop invariants

Predicate I is a loop invariant of **while** $G \; \{ \; prog \; \}$ if it is preserved by loop iterations, i.e., $G \wedge I \Rightarrow wlp(prog, I)$.

Linear invariant generation [Colón et al., 2002]

Linear programs

A program is linear program whenever all guards are linear constraints, and updates are linear expressions (in the real program variables).

Linear invariant generation [Colón et al., 2002]

Linear programs

A program is linear program whenever all guards are linear constraints, and updates are linear expressions (in the real program variables).

Approach by Colón et al.

1. Speculatively annotate a program with linear boolean expressions:

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \leqslant 0$$

where a_i is a parameter and x_i a program variable.

Linear invariant generation [Colón et al., 2002]

Linear programs

A program is linear program whenever all guards are linear constraints, and updates are linear expressions (in the real program variables).

Approach by Colón et al.

1. Speculatively annotate a program with linear boolean expressions:

$$a_1{\cdot}x_1 + \ldots + a_n{\cdot}x_n + a_{n+1} \ \leqslant\ 0$$

   where a_i is a parameter and x_i a program variable.

2. Express verification conditions as inequality constraints over a_i, x_i.

Linear invariant generation [Colón et al., 2002]

Linear programs

A program is linear program whenever all guards are linear constraints, and updates are linear expressions (in the real program variables).

Approach by Colón et al.

1. Speculatively annotate a program with linear boolean expressions:

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \leqslant 0$$

   where a_i is a parameter and x_i a program variable.
2. Express verification conditions as inequality constraints over a_i, x_i.
3. Transform these inequality constraints into polynomial constraints (Farkas lemma).

Linear invariant generation [Colón et al., 2002]

Linear programs

A program is linear program whenever all guards are linear constraints, and updates are linear expressions (in the real program variables).

Approach by Colón et al.

1. Speculatively annotate a program with linear boolean expressions:

   $$a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \leqslant 0$$

   where a_i is a parameter and x_i a program variable.
2. Express verification conditions as inequality constraints over a_i, x_i.
3. Transform these inequality constraints into polynomial constraints (Farkas lemma).
4. Use off-the-shelf constraint-solvers to solve them (e.g., REDLOG).

Linear invariant generation [Colón et al., 2002]

Linear programs

A program is linear program whenever all guards are linear constraints, and updates are linear expressions (in the real program variables).

Approach by Colón et al.

1. Speculatively annotate a program with linear boolean expressions:

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \leqslant 0$$

   where a_i is a parameter and x_i a program variable.
2. Express verification conditions as inequality constraints over a_i, x_i.
3. Transform these inequality constraints into polynomial constraints (Farkas lemma).
4. Use off-the-shelf constraint-solvers to solve them (e.g., REDLOG).
5. Exploit resulting assertions to infer program correctness.

Quantitative invariants

Weakest liberal precondition [McIver and Morgan, 2001]

Let P and Q be expectations (i.e., real-valued functions) over program variables in `prog`. Then:

$$\{ P \} \; prog \; \{ Q \} \quad \text{or} \quad P \leqslant wlp(prog, Q)$$

denotes that: if `prog` takes some initial state σ to a final distribution μ on states, then the expected value of postexpectation Q over μ is at least the (actual) value of pre-expectation P over σ.

Quantitative invariants

Weakest liberal precondition **[McIver and Morgan, 2001]**

Let P and Q be expectations (i.e., real-valued functions) over program variables in `prog`. Then:

$$\{\, P \,\}\ prog\ \{\, Q \,\} \quad \text{or} \quad P \ \leqslant\ wlp(prog, Q)$$

denotes that: if `prog` takes some initial state σ to a final distribution μ on states, then the expected value of postexpectation Q over μ is at least the (actual) value of pre-expectation P over σ.

In absence of probability, quantitative wlp is isomorphic to qualitative wlp.

Quantitative invariants

Weakest liberal precondition [McIver and Morgan, 2001]

Let P and Q be expectations (i.e., real-valued functions) over program variables in `prog`. Then:

$$\{ P \} \; prog \; \{ Q \} \quad \text{or} \quad P \leqslant wlp(prog, Q)$$

denotes that: if `prog` takes some initial state σ to a final distribution μ on states, then the expected value of postexpectation Q over μ is at least the (actual) value of pre-expectation P over σ.

In absence of probability, quantitative wlp is isomorphic to qualitative wlp.

Loop invariants

Predicate I is a loop invariant of **while** $G \; \{ \; prog \; \}$ if it is preserved by loop iterations, i.e., $[G] \times I \leqslant wlp(prog, I)$.

A simple slot machine

```
void flip {
  d1 := ♡ [1/2] ♢;
  d2 := ♡ [1/2] ♢;
  d3 := ♡ [1/2] ♢;
}
```

A simple slot machine

```
void flip {
  d1 := ♡ [1/2] ♢;
  d2 := ♡ [1/2] ♢;
  d3 := ♡ [1/2] ♢;
}
```

Example weakest liberal preconditions

Let $all(x) \equiv x = d_1 = d_2 = d_3$.

A simple slot machine

```
void flip {
  d1 := ♡ [1/2] ♢;
  d2 := ♡ [1/2] ♢;
  d3 := ♡ [1/2] ♢;
}
```

Example weakest liberal preconditions

Let $all(x) \equiv x = d_1 = d_2 = d_3$.

- If $Q = all(\heartsuit)$, then $wlp(flip, Q) = \frac{1}{8}$.

A simple slot machine

```
void flip {
  d1 := ♡ [1/2] ♢;
  d2 := ♡ [1/2] ♢;
  d3 := ♡ [1/2] ♢;
}
```

Example weakest liberal preconditions

Let $all(x) \equiv x = d_1 = d_2 = d_3$.

- If $Q = all(\heartsuit)$, then $wlp(flip, Q) = \frac{1}{8}$.
- If $Q' = 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$, then:

$$wlp(flip, Q') = 6 \times \frac{1}{8} \times 0 + 1 \times \frac{1}{8} \times 1 + 1 \times \frac{1}{8} \times \frac{1}{2} = \frac{3}{16}.$$

Play the game

```
void playGame {
  flip; // init
  while ¬(all(♡) ∨ all(◊)) { // loop
     flip;
  }
}
```

Play the game

```
void playGame {
  flip; // init
  while ¬(all(♡) ∨ all(♢)) { // loop
     flip;
  }
}
```

Example loop invariant

Let $Q' = 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$

Play the game

```
void playGame {
  flip; // init
  while ¬(all(♡) ∨ all(♢)) { // loop
     flip;
  }
}
```

Example loop invariant

Let $Q' = 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$

- Invariant $I = \frac{3}{4} \times [\neg all(\heartsuit) \wedge \neg all(\diamondsuit)] + 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$.

Play the game

```
void playGame {
  flip; // init
  while ¬(all(♡) ∨ all(♢)) { // loop
     flip;
  }
}
```

Example loop invariant

Let $Q' = 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$

- Invariant $I = \frac{3}{4} \times [\neg all(\heartsuit) \wedge \neg all(\diamondsuit)] + 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$.
- As $Q' = [all(\heartsuit) \vee all(\diamondsuit)] \times I$ we have $\{ I \}$ *loop* $\{ Q' \}$

Play the game

```
void playGame {
  flip; // init
  while ¬(all(♡) ∨ all(♢)) { // loop
     flip;
  }
}
```

Example loop invariant

Let $Q' = 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$

- Invariant $I = \frac{3}{4} \times [\neg all(\heartsuit) \wedge \neg all(\diamondsuit)] + 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$.
- As $Q' = [all(\heartsuit) \vee all(\diamondsuit)] \times I$ we have $\{ \, I \, \} \; loop \; \{ \, Q' \, \}$
- It follows $wlp(init, I) = \frac{3}{4}$.

Play the game

```
void playGame {
  flip; // init
  while ¬(all(♡) ∨ all(♢)) { // loop
     flip;
  }
}
```

Example loop invariant

Let $Q' = 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$

- Invariant $I = \frac{3}{4} \times [\neg all(\heartsuit) \wedge \neg all(\diamondsuit)] + 1 \times [all(\heartsuit)] + \frac{1}{2} \times [all(\diamondsuit)]$.
- As $Q' = [all(\heartsuit) \vee all(\diamondsuit)] \times I$ we have $\{ I \}$ *loop* $\{ Q' \}$
- It follows $wlp(init, I) = \frac{3}{4}$.
- In 50% the loop terminates with all ♡, in 50% with all ♢.

Probabilistic programs

Syntax

- `skip`
- `x := E`
- `prog1 ; prog2`
- **if** `(G) prog1`
  else `prog2`
- `prog1 [] prog2`
- `prog1 [p] prog2`
- **while** `(G)prog`

Probabilistic programs

Syntax

- `skip`
- `x := E`
- `prog1 ; prog2`
- `if (G) prog1`
  `else prog2`
- `prog1 [] prog2`
- `prog1 [p] prog2`
- `while (G)prog`

Semantics $wlp(prog, P)$

- $wlp(skip, P) = P$
- $wlp(x := E, P) = P[x/E]$
- $wlp(prog_1, wlp(prog_2, P))$
- $[G] \times wlp(prog_1, P)$
  $+[\neg G] \times wlp(prog_2, P)$
- $\min\left(wlp(prog_1, P), wlp(prog_2, P)\right)$
- $p * wlp(prog_1, P) + (1-p) * wlp(prog_2, P)$
- $\mu X. [G] \times wlp(prog, X) + [\neg G] \times P$

$*$ is scalar multiplication, $\times$ denotes multiplication of expectations.

Our approach

Main steps

Our approach

Main steps

1. Speculatively annotate a program with linear expressions:

$$[a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \ll 0] \times (b_1 \cdot x_1 + \ldots + b_n \cdot x_n + b_{n+1})$$

with real parameters a_i, b_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.

Our approach

Main steps

1. Speculatively annotate a program with linear expressions:

$$[a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \ \ll \ 0] \times (b_1 \cdot x_1 + \ldots + b_n \cdot x_n + b_{n+1})$$

   with real parameters a_i, b_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.

2. Transform these inequalities into boolean predicates.

Our approach

Main steps

1. Speculatively annotate a program with linear expressions:

$$[a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \ll 0] \times (b_1 \cdot x_1 + \ldots + b_n \cdot x_n + b_{n+1})$$

   with real parameters a_i, b_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.
2. Transform these inequalities into boolean predicates.
3. Transform these constraints into polynomial constraints (using Motzkin's theorem)

Our approach

Main steps

1. Speculatively annotate a program with linear expressions:

$$[a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \ll 0] \times (b_1 \cdot x_1 + \ldots + b_n \cdot x_n + b_{n+1})$$

   with real parameters a_i, b_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.
2. Transform these inequalities into boolean predicates.
3. Transform these constraints into polynomial constraints (using Motzkin's theorem)
4. Use off-the-shelf constraint-solvers to solve them (e.g., REDLOG).

Our approach

Main steps

1. Speculatively annotate a program with linear expressions:

$$[a_1 \cdot x_1 + \ldots + a_n \cdot x_n + a_{n+1} \ll 0] \times (b_1 \cdot x_1 + \ldots + b_n \cdot x_n + b_{n+1})$$

   with real parameters a_i, b_i, program variable x_i, and $\ll \in \{ <, \leqslant \}$.
2. Transform these inequalities into boolean predicates.
3. Transform these constraints into polynomial constraints (using Motzkin's theorem)
4. Use off-the-shelf constraint-solvers to solve them (e.g., REDLOG).
5. Exploit resulting assertions to infer program correctness.

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0;
  int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip;
     n := n + 1;
  }
  return x;
}
```

Template for quantitative invariant

$I \;=\; [0 \leqslant x \leqslant n \leqslant N] \times (a_1 \cdot x + a_2 \cdot n + a_3).$

Invariant templates

Qualitative setting [Colón et al., 2002]

Parameterised version of the j-th invariant I_j has the following shape:

$$\bigwedge_{m \in [1..M]} \left(\bigvee_{n \in [1..N]} \alpha_{(j,mn,1)} \cdot x_1 + \ldots + \alpha_{(j,mn,K)} \cdot x_k + \beta_{(j,mn)} \leqslant 0 \right)$$

Invariant templates

Qualitative setting [Colón et al., 2002]

Parameterised version of the j-th invariant I_j has the following shape:

$$\bigwedge_{m\in[1..M]} \left(\bigvee_{n\in[1..N]} \alpha_{(j,mn,1)} \cdot x_1 + \ldots + \alpha_{(j,mn,K)} \cdot x_k + \beta_{(j,mn)} \leqslant 0 \right)$$

Quantitative setting [Katoen et al., 2010]

Parameterised version of the j-th invariant I_j has the following shape:

$$\sum_{m\in[1..M]} \left[\bigwedge_{n\in[1..N]} \alpha_{(j,mn,1)} \cdot x_1 + \ldots + \alpha_{(j,mn,K)} \cdot x_k + \beta_{(j,mn)} \approx 0 \right] \times \left(\gamma_{(j,m,1)} x_1 + \ldots + \gamma_{(j,m,K)} x_K + \delta_{(j,m)} \right)$$

with the additional constraints $0 \leqslant I_j$ and $I_j \leqslant 1$.

Constructing machine-solvable constraints (1)

Lemma

For any loop-free probabilistic program *prog*, and linear expressible expectation P, $wlp(prog, P)$ is expressible as linear expression.

Constructing machine-solvable constraints (2)

Theorem

Let Q_{MN} be a linear expression with equivalent DNF (M, N)-linear expression

$$[P_1] \times Q_1 + \ldots + [P_M] \times Q_M$$

and Q'_{KL} be a linear expression with equivalent DNF (K, L)-linear expression

$$[P'_1] \times Q'_1 + \ldots + [P'_K] \times Q'_K.$$

Constructing machine-solvable constraints (2)

Theorem

Let Q_{MN} be a linear expression with equivalent DNF (M, N)-linear expression

$$[P_1] \times Q_1 + \ldots + [P_M] \times Q_M$$

and Q'_{KL} be a linear expression with equivalent DNF (K, L)-linear expression

$$[P'_1] \times Q'_1 + \ldots + [P'_K] \times Q'_K.$$

Then:

$$Q_{MN} \leqslant Q'_{KL}$$

if and only if for all $m \in [1..M]$, and $n \in [1..K]$:

$$P_m \wedge P'_k \Rightarrow (Q_m - Q'_k \leqslant 0)$$

$$P_m \wedge \left(\bigwedge_{k \in [1..K]} \neg P'_k\right) \Rightarrow Q_m \leqslant 0$$

Obtaining existentially quantified FO-formulas

Motzkin's transposition theorem is one of the deepest results in the part of mathematics dealing with linear inequalities

[Nemirovski & Roos, Encyclopedia of Optimization, 2009]

Motzkin's transposition theorem (1936)

Let A, A' be matrices, b, b' column vectors, and x a column vector of variables.

If $A \cdot x \leqslant b$ and $A' \cdot x < b'$ is unsatisfiable, then there exist row vectors λ, λ' with:

$$\lambda \geqslant 0 \text{ and } \lambda' \geqslant 0 \text{ and } \lambda \cdot A + \lambda' \cdot A' = 0$$

and either

1. $\lambda \cdot b + \lambda' \cdot b' > 0$, or
2. some entry of λ' is strictly positive and $\lambda \cdot b + \lambda' \cdot b' \geqslant 0$.

(λ and λ' form a witness of $A \cdot x \geqslant b$ and $A' \cdot x > b'$ being unsatisfiable.)

Motzkin's transposition theorem

$$\left[\begin{array}{ccccccccc} a_{(1,1)}x_1 & + & \ldots & + & a_{(1,n)}x_n & + & b_1 & \leqslant & 0 \\ & & \ldots & & & & & & \\ a_{(m,1)}x_1 & + & \ldots & + & a_{(m,n)}x_n & + & b_m & \leqslant & 0 \end{array}\right]$$

and

$$\left[\begin{array}{ccccccccc} a_{(m+1,1)}x_1 & + & \ldots & + & a_{(m+1,n)}x_n & + & b_{m+1} & < & 0 \\ & & \ldots & & & & & & \\ a_{(m+k,1)}x_1 & + & \ldots & + & a_{(m+k,n)}x_n & + & b_{m+k} & < & 0 \end{array}\right]$$

has no solution in $x_1, \ldots, x_n$

Motzkin's transposition theorem

iff there exist $\lambda_0, \lambda_1, \ldots, \lambda_{m+k} \in \mathbb{R}_{\geqslant 0}$ such that:

$$
\begin{array}{ll}
\lambda_1 & \\
 & \left[\begin{array}{ccccccccc} a_{(1,1)}x_1 & + & \ldots & + & a_{(1,n)}x_n & + & b_1 & \leqslant & 0 \\ & & \ldots & & & & & & \\ a_{(m,1)}x_1 & + & \ldots & + & a_{(m,n)}x_n & + & b_m & \leqslant & 0 \end{array}\right] \\
\lambda_m &
\end{array}
$$

and

$$
\begin{array}{ll}
\lambda_{m+1} & \\
 & \left[\begin{array}{ccccccccc} a_{(m+1,1)}x_1 & + & \ldots & + & a_{(m+1,n)}x_n & + & b_{m+1} & < & 0 \\ & & \ldots & & & & & & \\ a_{(m+k,1)}x_1 & + & \ldots & + & a_{(m+k,n)}x_n & + & b_{m+k} & < & 0 \end{array}\right] \\
\lambda_{m+k} &
\end{array}
$$

So: the inequalities can be linearly combined to get either $0 > 0$ or $0 \geqslant 1$.

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0; int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; n := n + 1; // body
  }
  return x;
}
```

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0; int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; n := n + 1; // body
  }
  return x;
}
```

Template for quantitative invariant

Given that $I_1 = 0 \leqslant x \leqslant n \leqslant N$ is invariant, we suggest the parameterised quantitative invariant:

$$I \ = \ [I_1] \times (a_1{\cdot}x + a_2{\cdot}n + a_3).$$

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0; int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; n := n + 1; // body
  }
  return x;
}
```

Constraints

1. $0 \leqslant I$
2. $I \leqslant 1$
3. $[n < N] \times I \leqslant wlp(body, I)$

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0; int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; n := n + 1; // body
  }
  return x;
}
```

Due to our theorems, this reduces to:

1. $0 \leqslant [I_1] \times (a_1{\cdot}x + a_2{\cdot}n + a_3)$
2. $[I_1] \times (a_1{\cdot}x + a_2{\cdot}n + a_3) \leqslant 1$
3. $[n < N \wedge I_1] \times (a_1{\cdot}x + a_2{\cdot}n + a_3) \leqslant wlp(body, a_1{\cdot}x + a_2{\cdot}n + a_3)$.

Binomial update

```
int binUpdate(float p, int N) { // 0 < p < 1
  int x := 0; int n := 0;
  while (n < N) {
     x := (x + 1) [p] skip; n := n + 1; // body
  }
  return x;
}
```

Due to invariance of I_1, this simplifies to:

1. $0 \leqslant [I_1] \times (a_1 \cdot x + a_2 \cdot n + a_3)$
2. $[I_1] \times (a_1 \cdot x + a_2 \cdot n + a_3) \leqslant 1$
3. $[n < N] \times (a_1 \cdot x + a_2 \cdot n + a_3) \leqslant wlp(body, a_1 \cdot x + a_2 \cdot n + a_3)$.

Derivation

For $wlp(body, a_1{\cdot}x + a_2{\cdot}n + a_3)$ we derive:

$$
\begin{aligned}
& wlp(x := x{+}1 \;{}_p{\oplus}\; \textbf{skip}; n := n{+}1, a_1{\cdot}x + a_2{\cdot}n + a_3) && \mid \text{ wlp for ;} \\
= \; & wlp(x := x{+}1 \;{}_p{\oplus}\; \textbf{skip}, wlp(n := n{+}1, a_1{\cdot}x + a_2{\cdot}n + a_3)) && \mid \text{ wlp for :=} \\
= \; & wlp(x := x{+}1 \;{}_p{\oplus}\; \textbf{skip}, a_1{\cdot}x + a_2{\cdot}n + a_2 + a_3) && \mid \text{ wlp for } {}_p{\oplus} \\
= \; & p * (a_1{\cdot}x + a_1 + a_2{\cdot}n + a_2 + a_3) \\
+ \; & (1{-}p) * (a_1{\cdot}x + a_2{\cdot}n + a_2 + a_3) && \mid \text{ simplify} \\
= \; & a_1{\cdot}x + a_2{\cdot}n + p{\cdot}a_1 + a_2 + a_3.
\end{aligned}
$$

Derivation

For $wlp(body, a_1 \cdot x + a_2 \cdot n + a_3)$ we derive:

$$
\begin{aligned}
& wlp(x := x{+}1 \;{}_p{\oplus}\; \textbf{skip}; n := n{+}1, a_1 \cdot x + a_2 \cdot n + a_3) && \mid \text{wlp for ;} \\
= \; & wlp(x := x{+}1 \;{}_p{\oplus}\; \textbf{skip}, wlp(n := n{+}1, a_1 \cdot x + a_2 \cdot n + a_3)) && \mid \text{wlp for :=} \\
= \; & wlp(x := x{+}1 \;{}_p{\oplus}\; \textbf{skip}, a_1 \cdot x + a_2 \cdot n + a_2 + a_3) && \mid \text{wlp for } {}_p{\oplus} \\
= \; & p * (a_1 \cdot x + a_1 + a_2 \cdot n + a_2 + a_3) \\
+ \; & (1{-}p) * (a_1 \cdot x + a_2 \cdot n + a_2 + a_3) && \mid \text{simplify} \\
= \; & a_1 \cdot x + a_2 \cdot n + p \cdot a_1 + a_2 + a_3.
\end{aligned}
$$

Thus:

$[n < N] \times (a_1 \cdot x + a_2 \cdot n + a_3) \leqslant wlp(body, a_1 \cdot x + a_2 \cdot n + a_3)$ reduces to

$[n < N] \times (a_1 \cdot x + a_2 \cdot n + a_3) \leqslant a_1 \cdot x + a_2 \cdot n + p \cdot a_1 + a_2 + a_3$, that is

$[n < N] \times 0 \leqslant p \cdot a_1 + a_2$.

From inequalities to matrices

Linear expressions

1. $0 \leqslant [0 \leqslant x \leqslant n \leqslant N] \times a_1 \cdot x + a_2 \cdot n + a_3$
2. $[0 \leqslant x \leqslant n \leqslant N] \times a_1 \cdot x + a_2 \cdot n + a_3 \leqslant 1$
3. $[n < N] \times (a_1 \cdot x + a_2 \cdot n + a_3) \leqslant wlp(body, a_1 \cdot x + a_2 \cdot n + a_3)$.

From inequalities to matrices

Linear expressions

1. $0 \leqslant [0 \leqslant x \leqslant n \leqslant N] \times a_1{\cdot}x + a_2{\cdot}n + a_3$
2. $[0 \leqslant x \leqslant n \leqslant N] \times a_1{\cdot}x + a_2{\cdot}n + a_3 \leqslant 1$
3. $[n < N] \times 0 \leqslant p{\cdot}a_1 + a_2$.

From inequalities to matrices

Linear expressions

1. $0 \leqslant [0 \leqslant x \leqslant n \leqslant N] \times a_1 \cdot x + a_2 \cdot n + a_3$
2. $[0 \leqslant x \leqslant n \leqslant N] \times a_1 \cdot x + a_2 \cdot n + a_3 \leqslant 1$
3. $[n < N] \times 0 \leqslant p \cdot a_1 + a_2$.

Inequalities

1. $-a_1 \cdot x - a_2 \cdot n - a_3 \leqslant 0 \lor -n + N < 0 \lor n - x < 0 \lor x < 0$
2. $a_1 \cdot x + a_2 \cdot n + a_3 - 1 \leqslant 0 \lor -n + N < 0 \lor n - x < 0 \lor x < 0$
3. $-p \cdot a_1 - a_2 < 0 \lor -n + N < 0$

Applying Motzkin's theorem

⇒ we obtain FO-formulas:

- $\exists \lambda_0, \ldots, \lambda_4 : \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 + \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 + \lambda_4 \beta \\ \wedge & 1 = \lambda_1(-M) + \lambda_4 \gamma - \lambda_0 \end{array} \right) \vee \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & \lambda_4 \neq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 + \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 + \lambda_4 \beta \\ \wedge & 0 = \lambda_1(-M) + \lambda_4 \gamma - \lambda_0 \end{array} \right)$
- $\exists \lambda_0, \ldots, \lambda_4 : \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 - \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 - \lambda_4 \beta \\ \wedge & 1 = -\lambda_1 M + \lambda_4(\gamma + 1) - \lambda_0 \end{array} \right) \vee \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & \lambda_4 \neq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 - \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 - \lambda_4 \beta \\ \wedge & 0 = -\lambda_1 M + \lambda_4(\gamma + 1) - \lambda_0 \end{array} \right)$
- $\exists \lambda_0, \ldots, \lambda_2 : \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_2 \geq 0 \\ \wedge & 0 = \lambda_2 \\ \wedge & 1 = \lambda_1(\beta p + \alpha) - \lambda_2 M - \lambda_0 \end{array} \right) \vee \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_2 \geq 0 \\ \wedge & \lambda_1, \lambda_2 \neq 0 \\ \wedge & 0 = \lambda_2 \\ \wedge & 0 = \lambda_1(\beta p + \alpha) - \lambda_2 M - \lambda_0 \end{array} \right)$

Applying Motzkin's theorem

⇒ we obtain FO-formulas:

- $\exists \lambda_0, \ldots, \lambda_4 : \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 + \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 + \lambda_4 \beta \\ \wedge & 1 = \lambda_1(-M) + \lambda_4 \gamma - \lambda_0 \end{array} \right) \vee \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & \lambda_4 \neq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 + \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 + \lambda_4 \beta \\ \wedge & 0 = \lambda_1(-M) + \lambda_4 \gamma - \lambda_0 \end{array} \right)$
- $\exists \lambda_0, \ldots, \lambda_4 : \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 - \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 - \lambda_4 \beta \\ \wedge & 1 = -\lambda_1 M + \lambda_4(\gamma + 1) - \lambda_0 \end{array} \right) \vee \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & \lambda_4 \neq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 - \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 - \lambda_4 \beta \\ \wedge & 0 = -\lambda_1 M + \lambda_4(\gamma + 1) - \lambda_0 \end{array} \right)$
- $\exists \lambda_0, \ldots, \lambda_2 : \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_2 \geq 0 \\ \wedge & 0 = \lambda_2 \\ \wedge & 1 = \lambda_1(\beta p + \alpha) - \lambda_2 M - \lambda_0 \end{array} \right) \vee \left(\begin{array}{ll} & \lambda_0, \ldots, \lambda_2 \geq 0 \\ \wedge & \lambda_1, \lambda_2 \neq 0 \\ \wedge & 0 = \lambda_2 \\ \wedge & 0 = \lambda_1(\beta p + \alpha) - \lambda_2 M - \lambda_0 \end{array} \right)$

solving with REDLOG obtains valid parameter constraints:

$[0 \leqslant x \leqslant n \leqslant N] \times (a_1 \cdot x - p \cdot a_1 \cdot n + p \cdot a_1 \cdot N)$

is invariant if N is positive and $0 < a_1 \leqslant \frac{1}{N}$.

Applying Motzkin's theorem

⇒ we obtain FO-formulas:

- $\exists \lambda_0, \ldots, \lambda_4 : \begin{pmatrix} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 + \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 + \lambda_4 \beta \\ \wedge & 1 = \lambda_1(-M) + \lambda_4 \gamma - \lambda_0 \end{pmatrix} \vee \begin{pmatrix} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & \lambda_4 \neq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 + \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 + \lambda_4 \beta \\ \wedge & 0 = \lambda_1(-M) + \lambda_4 \gamma - \lambda_0 \end{pmatrix}$
- $\exists \lambda_0, \ldots, \lambda_4 : \begin{pmatrix} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 - \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 - \lambda_4 \beta \\ \wedge & 1 = -\lambda_1 M + \lambda_4(\gamma + 1) - \lambda_0 \end{pmatrix} \vee \begin{pmatrix} & \lambda_0, \ldots, \lambda_4 \geq 0 \\ \wedge & \lambda_4 \neq 0 \\ \wedge & 0 = \lambda_1 - \lambda_2 - \lambda_4 \alpha \\ \wedge & 0 = \lambda_2 - \lambda_3 - \lambda_4 \beta \\ \wedge & 0 = -\lambda_1 M + \lambda_4(\gamma + 1) - \lambda_0 \end{pmatrix}$
- $\exists \lambda_0, \ldots, \lambda_2 : \begin{pmatrix} & \lambda_0, \ldots, \lambda_2 \geq 0 \\ \wedge & 0 = \lambda_2 \\ \wedge & 1 = \lambda_1(\beta p + \alpha) - \lambda_2 M - \lambda_0 \end{pmatrix} \vee \begin{pmatrix} & \lambda_0, \ldots, \lambda_2 \geq 0 \\ \wedge & \lambda_1, \lambda_2 \neq 0 \\ \wedge & 0 = \lambda_2 \\ \wedge & 0 = \lambda_1(\beta p + \alpha) - \lambda_2 M - \lambda_0 \end{pmatrix}$

It follows that a lower bound of the least expected value of x is $p \cdot N$.

Epilogue

Achievements:

- Generating loop invariants using constraint solving.
- Applied to linear probabilistic programs.
- Has potential for automated probabilistic program analysis.
- Prototypical tool-support under development.

Epilogue

Achievements:

- Generating loop invariants using constraint solving.
- Applied to linear probabilistic programs.
- Has potential for automated probabilistic program analysis.
- Prototypical tool-support under development.

Further work:

- Non-linear probabilistic programs.
- Average time-complexity analysis.
- Combination with model-checking approaches.
- Case studies.