Linear-invariant generation for probabilistic programs

JP Katoen$^a$ AK McIver$^b$ LA Meinicke$^b$ CC Morgan$^c$

$^a$ RWTH Aachen University, Germany
$^b$ Macquarie University, Australia
$^c$ University of New South Wales, Australia
Overview: Invariant generation

Inductive invariants may be used to verify iterative programs (Floyd, 1967; Hoare 1969; Dijkstra, 1971).

Automatically generating these invariants is possible for invariants of restricted forms for restricted types of programs.

Methods include

- **iterative fixed-point methods** like abstract interpretation (Cousot and Cousot, 1977), and

- **constraint-based approaches** (e.g., Colón et al., 2003; Podelski and Rybalchenko, 2004; Cousot, 2005; Monniaux, 2000; Gulwani et al., 2008).
Overview: Constraint-based approaches

Invariants are found by

- constructing verification conditions that are sufficient to show that
- predicates of a given parameterised formula (containing only first-order unknowns) are invariant, and then

- solving them for all possible parameters of the formula using off-the-shelf constraint solvers.
Overview: Constraint-based approaches

In Colón et al. (2003) parameterised formulas are

- linear constraints on program variables,

and the programs themselves must be

- linear

- with real-valued variables.

Methods that require weaker restrictions on the form of the program and invariant have been investigated. (E.g., Sankaranarayanan et al., 2004; Cousot, 2005; Kapur, 2005.)
Overview: Probabilistic Programs

Choices may be made both qualitatively ($S \cap T$) and quantitatively ($S_p \oplus T$).

Inductive quantitative invariants can be used in probabilistic program verification (Morgan, 1996; McIver and Morgan, 2005).

No methods exist for automatically generating quantitative invariants ...
Overview: Goals

The development of an automated assistant for quantitative invariant discovery to augment interactive proofs.
Overview: Goals

So far we have defined a constraint-based method for automatically generating quantitative invariants of

- linear probabilistic programs with

- real-valued variables, in which

- the parameterised invariants are structures built from linear terms.
Outline

• Qualitative and quantitative invariants.

• Constraint-solving for qualitative and quantitative invariants.

• An example.

• Comparison to other automated approaches.
Qualitative invariants: notation and semantics

Programs are interpreted using a weakest-liberal-precondition semantics:

- \( wlp.S.Q \) denotes the largest set of states from which \( S \) is guaranteed to either not terminate or terminate in a state satisfying \( Q \).

- \( S \) satisfies specification \( [P, Q] \) when \( P \Rightarrow wlp.S.Q \).

- \( S \sqsubseteq T \triangleq (\forall Q \cdot wlp.S.Q \Rightarrow wlp.T.Q) \).

- Specifications are treated as programs.
Qualitative invariants: inductive invariants

An inductive invariant $I$ of

$$loop \triangleq \text{while } G \text{ do } S \text{ od },$$

is a predicate on the state space of the program that is preserved by iterations of the loop.

I.e., $G \land I \Rightarrow wlp.S.I$.

If $I$ is an inductive invariant of $loop$, we have that $loop$ satisfies the specification $[I, \neg G \land I]$. 
Qualitative invariants: inductive invariant maps

A valid inductive invariant map of a program

\[ \text{loop}_1 \triangleq \text{while } G_1 \text{ do } S_1 \text{ od} \]

containing \( J \) program loops

\[ \text{loop}_j \triangleq \text{while } G_j \text{ do } S_j \text{ od} \]

is a set of predicates, \( \{I_j \cdot j \in [1..J]\} \), such that

\[ I_j \text{ is an invariant of while } G_j \text{ do } S'_j \text{ od} \]

where \( S_j \) is the same as \( S_j \) except that each of its inner loops \( \text{loop}_i \) (if any) has been replaced by \([I_i, \neg G_i \land I_i]\).
Qualitative invariants: inductive invariant maps

For example, \( \{I_1, I_2\} \) is a valid inductive invariant map of

\[
\begin{align*}
\text{loop}_1 : & \quad \text{while } G_1 \text{ do} \\
& \quad S_1; \\
\text{loop}_2 : & \quad \text{while } G_2 \text{ do } S_2 \text{ od} \\
& \quad \text{od}
\end{align*}
\]

(in which programs \( S_1 \) and \( S_2 \) do not contain loops) if

\[
\begin{align*}
I_2 \land G_2 & \Rightarrow \text{wlp}.S_2.I_2 \quad \text{and} \\
I_1 \land G_1 & \Rightarrow \text{wlp}.(S_1; [I_2, \neg G_2 \land I_2]).I_1 .
\end{align*}
\]
Qualitative invariants: inductive invariant maps

If \( \{I_j \cdot j \in [1..J]\} \) is a valid inductive invariant map of a loop \( \text{loop}_1 \), then each \( I_j \) is an invariant of \( \text{loop}_j \).

Generating valid inductive invariant maps for a given qualitative loop involves (by definition) finding solutions to a set of second-order constraints.
Quantitative invariants: notation and semantics

Probabilistic programs are given a meaning in terms of a weakest-liberal-precondition semantics (McIver and Morgan, 2005).

Predicates are generalised to non-negative, one-bounded, real-valued functions, referred to as expectations.

- \( \text{wlp}.S.\text{expt}.\sigma \) denotes the least expected value of \( \text{expt} \) that may be witnessed by executing \( S \) from initial state \( \sigma \).

- \( S \) satisfies specification \([\text{expt}_1, \text{expt}_2]\) when \( \text{expt}_1 \leq \text{wlp}.S.\text{expt}_2 \).

- \( S \sqsubseteq T \triangleq (\forall \text{expt} \cdot \text{wlp}.S.\text{expt} \leq \text{wlp}.T.\text{expt}) \).

- Specifications are treated as programs.
Quantitative invariants

A quantitative invariant $I$ of

$$loop \triangleq \text{while } G \text{ do } S \text{ od}$$

is an expectation whose expected value does not decrease after iteration of the body of the loop.

I.e., $[G] \times I \leq wlp.S.I$.

If $I$ is a quantitative invariant of $loop$ then $loop$ satisfies the specification $[I, [-G] \times I]$.
Quantitative invariants: Binomial update example

We have that

\[ I \triangleq [0 \leq x \leq n \leq N] \times (x/N - pn/N + p) \]

is an invariant of:

\[
\begin{align*}
\text{init} & : \quad x, n := 0, 0; \\
\text{loop} & : \quad \text{while } n < N \text{ do} \\
\text{body} & : \quad (x := x + 1_p \oplus \text{skip}); n := n + 1 \\
& \quad \text{od}
\end{align*}
\]

since \( I \leq wlp.body.I \).
Quantitative invariants: Binomial update example

So we can conclude that

\[
\text{wlp.}(\text{init}; \text{loop}).(x/N) \\
= \text{wlp.init.}(\text{wlp.loop.}(x/N)) \quad \{\text{definition of sequential composition}\} \\
\geq \text{wlp.init.I} \\
= [0 \leq N] \times p \quad \{\text{calculate}\} \\
= p. \\
\]

\{\text{assuming that } N \text{ is positive}\}

And so \( pN \leq \text{wlp.}(\text{init}; \text{loop}).x \).
Quantitative invariants: inductive invariant maps

We define a valid quantitative inductive invariant map to be a valid qualitative inductive invariant map in which expectations take the place of predicates.

If \( \{I_j \cdot j \in [1..J]\} \) is a valid quantitative inductive invariant map of a probabilistic loop \( \text{loop}_1 \), then each \( I_j \) is a quantitative invariant of \( \text{loop}_j \).
Constraint-solving for qualitative invariants

An overview of how the constraint-solving method of Colón et al. (2003) may be applied to find valid inductive invariant maps for any "linear program"

\[ \text{loop}_1 \triangleq \text{while } G_1 \text{ do } S_1 \text{ od} \]

with real-valued program variables \( x_1, \ldots, x_X \), containing \( J \) loops

\[ \text{loop}_j \triangleq \text{while } G_j \text{ do } S_j \text{ od} . \]
Constraint-solving for qualitative invariants: parameterisation

Each $I_j$ is first parameterised using an $(M, N)$-linear predicate

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

with free real-valued variables $\alpha_{(j,mn,x)}$ and $\beta_{(j,mn)}$, where $\approx$ may be instantiated with either comparison operator $\leq$ or $\lt$. 
Constraint-solving for qualitative invariants: parameterisation

Parameterising each $I_j$ in the definition of an inductive invariant map reduces each proof obligation

$$G_j \land I_j \Rightarrow \text{wl}p.S'_j.\dot{I}_j$$

to a first-order constraint on the free real-valued variables in the parametric representations.
Constraint-solving for qualitative invariants:
simplification and solving

Next we:

- Evaluate the weakest-liberal precondition expressions, representing each proof obligation

\[ G_j \land I_j \Rightarrow \text{wlp}.S'_j.I_j \]

as a finite Boolean expression on linear constraints.

- Translate these universally quantified Boolean expressions to existentially quantified polynomial constraints using Motzkin’s Transposition Theorem (Motzkin, 1936).
• Solve the resulting constraints using off-the-shelf constraint solvers.
Constraint-solving for quantitative invariants: parameterisation

We parameterise each $I_j$ with an $(M, N)$-linear expression

$$
\sum_{m \in [1..M]} [\bigwedge_{n \in [1..N]} \alpha(j,mn,1)x_1 + \ldots + \alpha(j,mn,X)x + \beta(j,mn) \approx 0] \\
\times \left( \gamma(j,m,1)x_1 + \ldots + \gamma(j,m,X)x + \delta(j,m) \right)
$$

containing free real-valued variables $\alpha(j,mn,x)$, $\beta(j,mn)$, $\gamma(j,m,x)$ and $\delta(j,m)$.

We impose the additional constraint that, for each $j \in [1..J]$, $I_j$ is bounded.

I.e., $0 \leq I_j$ and $I_j \leq 1$. 
Constraint-solving for quantitative invariants: parameterisation

Parameterisation reduces the constraints on each inductive invariant $I_j$ in the quantitative inductive invariant map to the following universally quantified constraints on first-order unknowns:

$$0 \leq I_j,$$
$$I_j \leq 1 \quad \text{and}$$
$$[G_j] \times I_j \leq wlp(S'_j).I_j.$$
Constraint-solving for quantitative invariants: simplification and solving

Next we

- Evaluate the weakest-liberal precondition expressions, and translate each constraint

\[ 0 \leq I_j , \quad (1) \]

\[ I_j \leq 1 \quad \text{and} \quad (2) \]

\[ [G'_j] \times I_j \leq wlp.S'_j I_j . \quad (3) \]

to a finite **Boolean expression on linear constraints**

- Apply Motzkin’s Transposition theorem (as for qualitative case).
• Solve the resulting constraints (as for qualitative case).
Constraint-solving for quantitative invariants: simplification and solving

We have shown that the first of these steps is possible since:

(i) Conditions (1-3) may be written as inequalities between some \((M,N)\)-linear and \((K,L)\)-linear expressions.

(ii) Each inequality between a \((M,N)\)-linear and \((K,L)\)-linear expression may be represented as a finite Boolean expression over linear constraints.
Example: Binomial Update

Given that $I_1 \triangleq [0 \leq x \leq n \leq N]$ is invariant, we search for quantitative invariants

$$I \triangleq I_1 \times (\alpha x + \beta n + \gamma).$$
Example: Binomial Update

The constraints that any such \( I \triangleq I_1 \times (\alpha x + \beta n + \gamma) \) must satisfy are:

\[
0 \leq I_1 \times (\alpha x + \beta n + \gamma) \quad (4)
\]

\[
I_1 \times (\alpha x + \beta n + \gamma) \leq 1 \quad (5)
\]

\[
[n < N] \times I_1 \times (\alpha x + \beta n + \gamma) \leq \text{wl}p.body.(I_1 \times (\alpha x + \beta n + \gamma)) \quad (6)
\]
Example: Binomial Update

Using the fact that $I_1$ is invariant, these are equivalent to the following Boolean constraints

$$I_1 \Rightarrow (0 \leq \alpha x + \beta n + \gamma) \quad (7)$$

$$I_1 \Rightarrow (\alpha x + \beta n + \gamma \leq 1) \quad (8)$$

$$n < N \land I_1 \Rightarrow (\alpha x + \beta n + \gamma) \leq \text{wl}_p.body.(\alpha x + \beta n + \gamma) \quad (9)$$
Example: Binomial Update

Evaluating \( wlp \) expression:

\[
I_1 \Rightarrow (0 \leq \alpha x + \beta n + \gamma) \quad (10)
\]
\[
I_1 \Rightarrow (\alpha x + \beta n + \gamma \leq 1) \quad (11)
\]
\[
n < N \land I_1 \Rightarrow (\alpha x + \beta n + \gamma) \leq \alpha x + \beta n + p\alpha + \beta + \gamma \quad (12)
\]

Translating (12) to a set of existentially quantified constraints and solving reveals that parameters \( \alpha, \beta \) and \( \gamma \) must satisfy

\[
p\alpha + \beta \geq 0,
\]

i.e., variable \( x \) grows at most \( p \) times the rate of \( n \).

(The other constraints may be similarly solved.)
Alternative automated methods: comparison

Using proof-based methods in conjunction with automatic quantitative invariant generation methods we can verify programs with parameters.

(Such as the Binomial program with parameters $p$ and $N$.)
Alternative automated methods: comparison

Probabilistic model checkers for MDP’s, e.g.,

- **PRISM:**
  PCTL model checking allowing calculation of, e.g., maximal reachability probabilities.

- **LiQuor:**
  Maximal probability that MDP M satisfies LTL formula $\varphi$.

Can be applied to instances of probabilistic programs.

E.g., for the Binomial distribution program it can be checked:

“whether or not the probability that $x = 0$ is $(1 - p)^N$, for a fixed value of $p$ and $N$”. 
Alternative automated methods: comparison

Automated abstraction-refinement, e.g.,

- **PASS**: A SAT-based extension of PRISM. Can be used to check maximal reachability properties.

- **SAT-based PRISM**: lower and upper bounds of maximal reachability probabilities or of expected values.

  can be used to verify properties of programs with unbounded unknowns.
Alternative automated methods: comparison

Other tools include

- **APEX**: checks language equivalence between probabilistic programs over finite integer datatypes but in addition allows open programs, i.e., programs in which the value of certain variables is not fixed.

- **Abstract interpretation methods** for probabilistic programs: As for non-probabilistic abstract interpretation methods, these might only produce “approximate answers”.

None of these other methods produce quantitative invariants for probabilistic programs.
Conclusion: what we’ve done

We have defined a sound constraint-based method for generating linear quantitative invariants for linear probabilistic programs with real-valued variables.
Conclusion: what we’d like to do

We would like to

- build tool support for this approach

- extend our approach to generate polynomial forms of quantitative invariants (as in Sankaranarayanan et al. (2004), Cousot (2005) and Kapur (2005)).