Inductive Invariant Synthesis

Using Convex Programming and Satisfiability Modulo Theory

George Egor Karpenkov

VERIMAG

March 29, 2017

Outline

Motivation

Need for Reliable Systems

- Only a couple of decades ago:
  - Computers are separate, stationary machines
- Now:
  - Hard to find a device which is not a computer
- Computerized systems can:

Motivation

Need for Reliable Systems

- Only a couple of decades ago:
  - Computers are separate, stationary machines
- Now:
  - Hard to find a device which is not a computer
- Computerized systems can:
  - Crash

Motivation

Need for Reliable Systems

- Only a couple of decades ago:
  - Computers are separate, stationary machines
- Now:
  - Hard to find a device which is not a computer
- Computerized systems can:
  - Crash
  - Have bugs

Motivation

Need for Reliable Systems

- Only a couple of decades ago:
  - Computers are separate, stationary machines
- Now:
  - Hard to find a device which is not a computer
- Computerized systems can:
  - Crash
  - Have bugs
  - Have security exploits

Motivation

Need for Reliable Systems

- Only a couple of decades ago:
  - Computers are separate, stationary machines
- Now:
  - Hard to find a device which is not a computer
- Computerized systems can:
  - Crash
  - Have bugs
  - Have security exploits
  - ...

Motivation

Need for Reliable Systems

- Only a couple of decades ago:
  - Computers are separate, stationary machines
- Now:
  - Hard to find a device which is not a computer
- Computerized systems can:
  - Crash
  - Have bugs
  - Have security exploits
  - ...
- Many exploits: shellshock, heartbleed, etc.

Goal

Increasing software reliability

Static Analysis

Main Ideas

- Analyze program *without* running it

Static Analysis

Main Ideas

- Analyze program *without* running it
- Sound safety proofs
  - Overflows
  - Null-pointer derefs
  - ...

Static Analysis

Main Ideas

- Analyze program *without* running it
- Sound safety proofs
  - Overflows
  - Null-pointer derefs
  - . . .
- Not complete (Turing, Church)

Static Analysis

Main Ideas

- Analyze program *without* running it
- Sound safety proofs
  - Overflows
  - Null-pointer derefs
  - ...
- Not complete (Turing, Church)

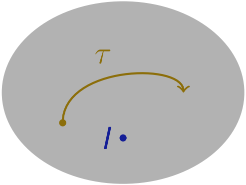
Overall Framework

Abstract interpretation unifies static analyses

Inductive Invariant

Proving Properties

- Infinite-State System
  - Proof: by induction
- Find inductive invariant
  - True by initiation
  - Holds by consecution
- Complete proof method



Everyone Loves Inductive Invariants

Verification, bug hunting, compiler optimizations, ...

Outline

Contributions

- Theoretical
  - Policy Iteration
    - Local Policy Iteration Algorithm
      (LPI, Chapter III, VMCAI'16)
    - Template Generation Approaches
      (Chapter IV, To be Published)
    - Summary Generation Using Policy Iteration
      (Chapter V, To be Published)
  - Inductive Invariants from Preconditions
    (Chapter VI, "Formula Slicing", HVC'16)
- Engineering
  - LPI Implementation in CPAchecker
    (Chapter VII, TACAS'16)
  - Library for Utilizing Satisfiability Modulo Theory Solvers
    (Chapter VIII, JavaSMT, VSTTE'16)

Contributions

- Theoretical
  - Policy Iteration
    - Local Policy Iteration Algorithm (LPI, Chapter III, VMCAI'16)
    - Template Generation Approaches (Chapter IV, To be Published)
    - Summary Generation Using Policy Iteration (Chapter V, To be Published)
  - Inductive Invariants from Preconditions (Chapter VI, "Formula Slicing", HVC'16)
- Engineering
  - LPI Implementation in CPAchecker (Chapter VII, TACAS'16)
  - Library for Utilizing Satisfiability Modulo Theory Solvers (Chapter VIII, JavaSMT, VSTTE'16)

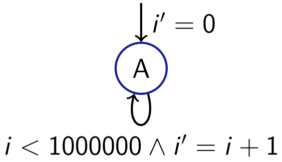
Outline

Control Flow Automaton

Program Formalization

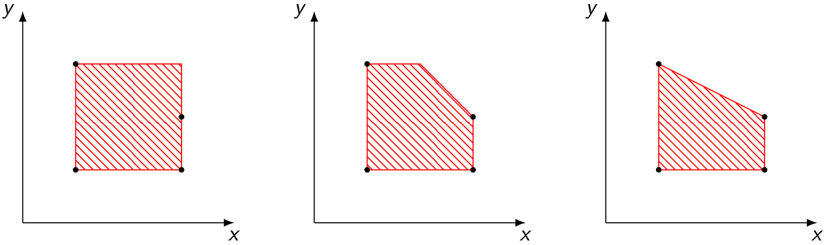
- Over program variables $\mathbf{x}$
- Transitions: associated with edges, first order formulas over
  - $\mathbf{x}$ input variables
  - $\mathbf{x}'$ output variables
- Invariants: associated with nodes, predicates over $\mathbf{x}$

```
int i=0;
while (i<1000000) {
    i++;
}
```



Abstract domains

- Domain is a lattice: set with a partial order
  - Given by inclusion
- Usual domains: intervals, octagons, polyhedra
- For a program with two variables $\mathbf{x} \equiv \{x, y\}$
  - Abstracting 4 states:



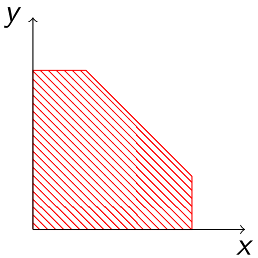
(a) intervals: $\pm x$

(b) octagons: $\pm x, \pm x \pm y$

(c) polyhedra: arbitrary convex shape $P\mathbf{x} \leq a$

Template Constraints Domain

- Polyhedra Domain:
  - most expressive
  - not a complete lattice
  - exponential runtime
- Configurable compromise: template constraints domains
  - directions fixed in advance
  - complete lattice
- For templates $T \equiv (-x, -y, x, y, x + y)$
  - State $a_0 \equiv (0, 0, 3, 3, 4)$
  - Concretizes to $0 \leq x \leq 3 \wedge 0 \leq y \leq 3 \wedge x + y \leq 4$



Template Constraints Domain

Strongest Postcondition

- Abstract semantics: transition relation in the abstract domain
- Template constraints domain: linear programming

Template Constraints Domain

Strongest Postcondition

- Abstract semantics: transition relation in the abstract domain
- Template constraints domain: linear programming

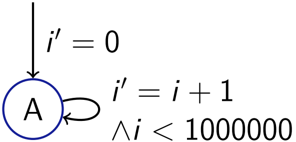
Example (Abstract Semantics)

- Template x, transition $x' = x + 1$, previous element $x \leq 5$
- New element given by $\max x'$ s. t. $x' = x + 1 \wedge x \leq 5$

Abstract Interpretation

Example Analysis in Intervals Domain

```
int i=0;
while (i < 1000000) {
    i++;
}
```

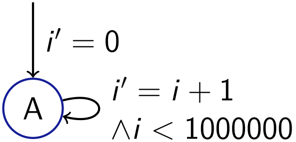


- Candidate invariants at A:
  - $i \in [0, 0]$ (abstraction of $\{i : 0\}$)

Abstract Interpretation

Example Analysis in Intervals Domain

```
int i=0;
while (i < 1000000) {
    i++;
}
```

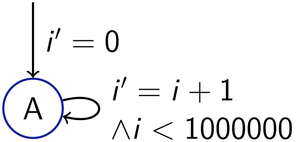


- Candidate invariants at A:
  - $i \in [0, 0]$ (abstraction of $\{i : 0\}$)
  - $i \in ([0, 0] \sqcup [1, 1]) = [0, 1]$

Abstract Interpretation

Example Analysis in Intervals Domain

```
int i=0;
while (i < 1000000) {
    i++;
}
```

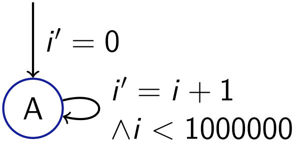


- Candidate invariants at A:
  - $i \in [0, 0]$ (abstraction of $\{i : 0\}$)
  - $i \in ([0, 0] \sqcup [1, 1]) = [0, 1]$
  - $i \in ([0, 1] \sqcup [1, 2]) = [0, 2]$

Abstract Interpretation

Example Analysis in Intervals Domain

```
int i=0;
while (i < 1000000) {
    i++;
}
```

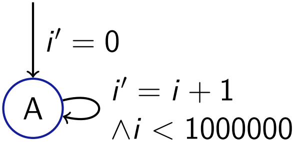


- Candidate invariants at A:
  - $i \in [0, 0]$ (abstraction of $\{i : 0\}$)
  - $i \in ([0, 0] \sqcup [1, 1]) = [0, 1]$
  - $i \in ([0, 1] \sqcup [1, 2]) = [0, 2]$
  - $\ldots$

Abstract Interpretation

Example Analysis in Intervals Domain

```
int i=0;
while (i < 1000000) {
    i++;
}
```

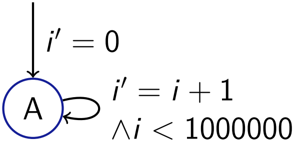


- Candidate invariants at A:
  - $i \in [0, 0]$ (abstraction of $\{i : 0\}$)
  - $i \in ([0, 0] \sqcup [1, 1]) = [0, 1]$
  - $i \in ([0, 1] \sqcup [1, 2]) = [0, 2]$
  - ...
  - Widening: $i \in [0, +\infty)$

Abstract Interpretation

Example Analysis in Intervals Domain

```
int i=0;
while (i < 1000000) {
    i++;
}
```



- Candidate invariants at A:
  - $i \in [0, 0]$ (abstraction of $\{i : 0\}$)
  - $i \in ([0, 0] \sqcup [1, 1]) = [0, 1]$
  - $i \in ([0, 1] \sqcup [1, 2]) = [0, 2]$
  - $\ldots$
  - Widening: $i \in [0, +\infty)$
  - Narrowing: $i \in [0, 1000000]$

Policy Iteration

Motivation

```
int i=0;
while (input()) {
    i++;
    if (i == 1000000) {
        break;
    }
}
```

- Slightly modified program

Policy Iteration

Motivation

```
int i=0;
while (input()) {
    i++;
    if (i == 1000000) {
        break;
    }
}
```

- Slightly modified program
- Narrowing is fragile

Policy Iteration

Motivation

```
int i=0;
while (input()) {
    i++;
    if (i == 1000000) {
        break;
    }
}
```

- Slightly modified program
- Narrowing is fragile
- Narrowing: $i \in [0, \infty)$

Max-Policy Iteration

Finding *Least* Inductive Invariant

- Game-theoretic technique
- Used in artificial intelligence field
- E.g. solving poker

Max-Policy Iteration

Properties

- Generate smallest inductive invariant in the abstract domain
- Certain restriction on an abstract domain
- Exponential runtime
- Formulate as an optimization problem
- Solve non-convex optimization problem
  - By iteration over convex under-approximations (policies)

Guarantees

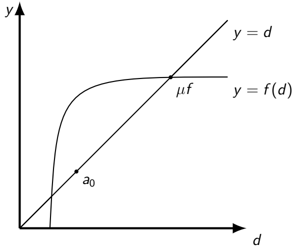
Least inductive invariant in the domain, not least invariant in general!

Max-Policy Iteration

Optimization Problem

- Simple program: one node, one initial condition, one transition
- Template Constraints Domain T, $n \equiv \|T\|$
  - Abstract state: $\mathbf{a} \in \mathbb{R}^n$
  - Abstract monotone transformer: $f : \mathbb{R}^n \to \mathbb{R}^n$
  - Initial condition: $\mathbf{a}_0 \in \mathbb{R}^n$
  - Tarski: least fixpoint of f exists in $(\mathbb{R} \cup \{+\infty, -\infty\})^n$
- Least inductive invariant definition:

$$\min \mathbf{a} \text{ s.t. } \mathbf{a} \succeq f(\mathbf{a}) \wedge \mathbf{a} \succeq \mathbf{a}_0$$



Max-Policy Iteration

Towards Convex Optimization Problem

- Convex optimization problems are (generally) feasible

Max-Policy Iteration

Towards Convex Optimization Problem

- Convex optimization problems are (generally) feasible
- Least fixed point

$$\min \mathbf{a} \text{ s.t. } \mathbf{a} \succeq f(\mathbf{a})$$

Max-Policy Iteration

Towards Convex Optimization Problem

- Convex optimization problems are (generally) feasible
- Least fixed point

$$\min \mathbf{a} \text{ s.t. } \mathbf{a} \succeq f(\mathbf{a})$$

- Towards convexity: suppose f is concave
- Then *greatest* fixed point optimization problem is convex:

$$\max \mathbf{a} \text{ s.t. } \mathbf{a} \preceq f(\mathbf{a})$$

Max-Policy Iteration

Towards Convex Optimization Problem

- Convex optimization problems are (generally) feasible
- Least fixed point

$$\min \mathbf{a} \text{ s.t. } \mathbf{a} \succeq f(\mathbf{a})$$

- Towards convexity: suppose f is concave
- Then *greatest* fixed point optimization problem is convex:

$$\max \mathbf{a} \text{ s.t. } \mathbf{a} \preceq f(\mathbf{a})$$

Theorem (Fixed Point Uniqueness)

For monotone, concave f, where for initial condition a_0, $f(a_0) \succ a_0$ post-a_0 fixed point is unique!

Max-Policy Iteration

Introducing Policies

- What's concave?
  - Template constraints domain transfer function

Max-Policy Iteration

Introducing Policies

- What's concave?
  - Template constraints domain transfer function
- When it stops being concave?
  - Disjunctions: multiple incoming edges
  - Each conjunct is concave
  - Transition: point-wise maximum over incoming transitions

Max-Policy Iteration

Introducing Policies

- What's concave?
  - Template constraints domain transfer function
- When it stops being concave?
  - Disjunctions: multiple incoming edges
  - Each conjunct is concave
  - Transition: point-wise maximum over incoming transitions

Idea

Iterate over concave under-approximations of f (policies), find the value of each one

Max-Policy Iteration

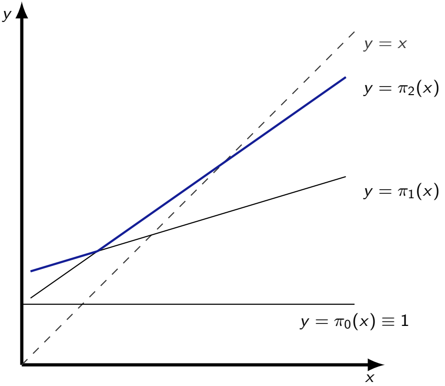
Example

```
double x = 1;
while (input()) {
  if (input()) {
     x=0.3*x+1.5;
  } else {
     x=0.7*x+1;
  }
}
```

- Find: inductive upper bound a on x
- Initial condition: $\pi_0 = \lambda d.1$
- Two policies:
  - $\pi_1 \equiv \lambda d.\max x'$ s.t. $x' = 0.3x + 1.5 \wedge x \leq d$
  - $\pi_2 \equiv \lambda d.\max x'$ s.t. $x' = 0.7x + 1 \wedge x \leq d$
- $f \equiv \lambda d.\max\{\pi_0(d), \pi_1(d), \pi_2(d)\}$
- Inductive upper bound is:
  - $\min d$ s.t. $d \geq f(d)$

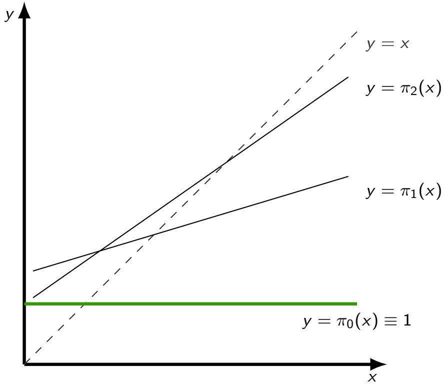
Max-Policy Iteration Visualization

$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



Max-Policy Iteration Visualization

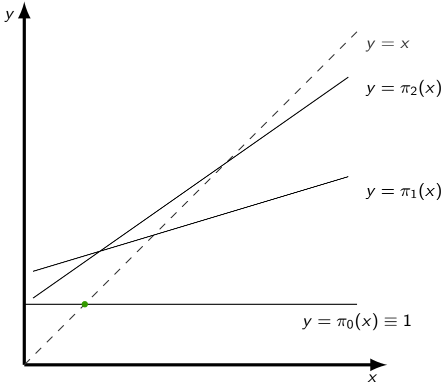
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



- Initial condition

Max-Policy Iteration Visualization

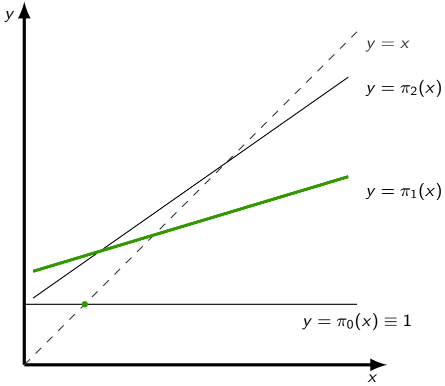
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



- Initial condition
- Value evaluates to $d = 1$: $\max d$ s.t. $d \leq x' \wedge x' = 1$

Max-Policy Iteration Visualization

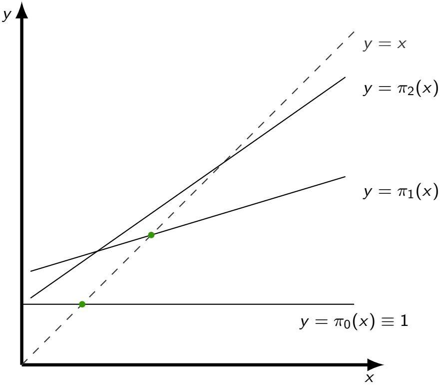
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



- Initial condition
- Value evaluates to $d = 1$: $\max d \text{ s.t. } d \leq x' \wedge x' = 1$
- Not inductive: $f(1) > 1$

Max-Policy Iteration Visualization

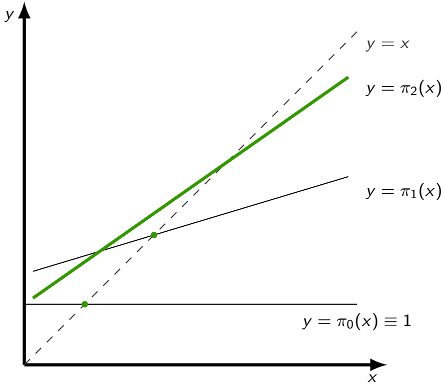
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



- Initial condition
- Value evaluates to $d = 1$: $\max d \text{ s.t. } d \leq x' \wedge x' = 1$
- Not inductive: $f(1) > 1$
- Value evaluates to 1.8: $\max d \text{ s.t. } d \leq \pi_2(d)$

Max-Policy Iteration Visualization

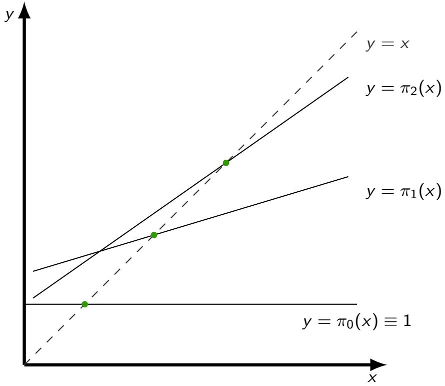
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



- Initial condition
- Value evaluates to $d = 1$: $\max d$ s.t. $d \leq x' \wedge x' = 1$
- Not inductive: $f(1) > 1$
- Value evaluates to 1.8: $\max d$ s.t. $d \leq \pi_2(d)$
- Not inductive: $f(1.8) > 1.8$

Max-Policy Iteration Visualization

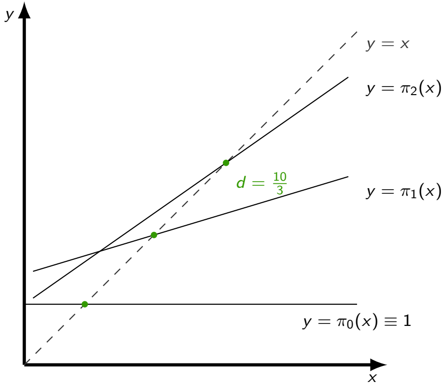
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \wedge x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \wedge x \leq d \end{cases}$$



- Initial condition
- Value evaluates to $d = 1$: $\max d$ s.t. $d \leq x' \wedge x' = 1$
- Not inductive: $f(1) > 1$
- Value evaluates to 1.8: $\max d$ s.t. $d \leq \pi_2(d)$
- Not inductive: $f(1.8) > 1.8$
- Value evaluates to $\frac{10}{3}$: $\max d$ s.t. $d \leq \pi_3(d)$

Max-Policy Iteration Visualization

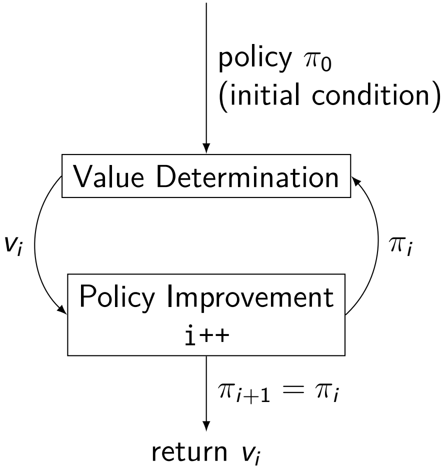
$$d \geq \max \begin{cases} \max x' \text{ s.t. } x' = 1 \\ \max x' \text{ s.t. } x' = 0.3x + 1.5 \land x \leq d \\ \max x' \text{ s.t. } x' = 0.7x + 1 \land x \leq d \end{cases}$$



- Initial condition
- Value evaluates to $d = 1$: $\max d$ s.t. $d \leq x' \land x' = 1$
- Not inductive: $f(1) > 1$
- Value evaluates to 1.8: $\max d$ s.t. $d \leq \pi_2(d)$
- Not inductive: $f(1.8) > 1.8$
- Value evaluates to $\frac{10}{3}$: $\max d$ s.t. $d \leq \pi_3(d)$
- Inductive!: $f(\frac{10}{3}) = \frac{10}{3}$

Policy Iteration

Iteration Algorithm



- Iterate on policies
- Find value for each
- Terminate on inductiveness

Max-Policy Iteration

Larger Programs

- Multiple templates, multiple nodes:
  - Choice of incoming transition per template, per node

Max-Policy Iteration

Larger Programs

- Multiple templates, multiple nodes:
  - Choice of incoming transition per template, per node
- Policy Improvement: SMT query
- Value Determination: LP query

Outline

Problems of Policy Iteration

Motivation for our work

Policy Iteration is under-determined

Which policy to improve? When? How?

- Arising issues:
  - Scalability: solving global equation system
  - Iteration Order: not defined in the algorithm
  - Cooperability: doesn't fit into existing frameworks

Local Policy Iteration

Integrating Abstract Interpretation

- Our work: LPI (Local Policy Iteration)
  - Exploits existing iteration strategies
  - Avoids solving the global equation at each step
  - Unifies policy iteration: precise widening operator

Idea

Bring results from abstract interpretation back into policy iteration (iteration order, locality, communication)

Abstract Interpretation Formulation

Required Ingredients

Abstract Interpretation Formulation

Required Ingredients

- Abstract domain: $\mathcal{D}$, partial order $\sqsubseteq$
- Join operator: $\sqcup : \mathcal{D} \rightarrow \mathcal{D} \rightarrow \mathcal{D}$
- Postcondition operator: $\rightsquigarrow: \mathcal{D} \rightarrow \tau(\mathbf{x} \cup \mathbf{x}') \rightarrow \mathcal{D}$
- Widening $\nabla : \mathcal{D} \rightarrow \mathcal{D} \rightarrow \mathcal{D}$

Abstract Interpretation Formulation

Required Ingredients

- Abstract domain: $\mathcal{D}$, partial order $\sqsubseteq$
- Join operator: $\sqcup : \mathcal{D} \to \mathcal{D} \to \mathcal{D}$
- Postcondition operator: $\leadsto: \mathcal{D} \to \tau(\mathbf{x} \cup \mathbf{x}') \to \mathcal{D}$
- Widening $\nabla : \mathcal{D} \to \mathcal{D} \to \mathcal{D}$

Aim

Express policy iteration as classical Kleene iteration

LPI Abstract Domain $\mathcal{D}$

Idea

Set of reachable abstract states stores policy implicitly

LPI Abstract Domain $\mathcal{D}$

Idea

Set of reachable abstract states stores policy implicitly

Definition (LPI State)

- Template constraints domain state + policy information.
- Map from templates to tuples
- (bound $d \in \mathbb{R}$, policy $\pi : \mathbb{R}^n \to \mathbb{R}$, **backpointer** $a \in \mathcal{D}$)

LPI Abstract Domain $\mathcal{D}$

Idea

Set of reachable abstract states stores policy implicitly

Definition (LPI State)

- Template constraints domain state + policy information.
- Map from templates to tuples
- (bound $d \in \mathbb{R}$, policy $\pi : \mathbb{R}^n \to \mathbb{R}$, **backpointer** $a \in \mathcal{D}$)

- Abstract state example: $\{i : (0, i' = 0, \mathbf{A})\}$
- Partial order: component-wise comparison on bounds

LPI Abstract Domain $\mathcal{D}$

Idea

Set of reachable abstract states stores policy implicitly

Definition (LPI State)

- Template constraints domain state + policy information.
- Map from templates to tuples
- (bound $d \in \mathbb{R}$, policy $\pi : \mathbb{R}^n \to \mathbb{R}$, **backpointer** $a \in \mathcal{D}$)

- Abstract state example: $\{i : (0, i' = 0, \mathbf{A})\}$
- Partial order: component-wise comparison on bounds

- For mapping $s \equiv \{t : (d, \pi, a)\}$
  - Concretization γ: $\{\mathbf{x} \mid t^\top \mathbf{x} \leq d\}$
  - Invariant: $d = \pi(\gamma(a))$

LPI Postcondition Computation

$$a_0 \equiv \{x : (4, \ldots, \ldots)\} \xrightarrow{x' = x + 1 \lor x' = 2x} \; ?$$

- Record the policy and the backpointer along with the bound
- Policy: concave under-approximation of transition relation
- Computed using optimization modulo SMT:
  - Successor is $a_1 \equiv \{x : (8, \lambda d.\max x' \text{ s.t. } x' = 2x \land x \le d, a_0)\}$

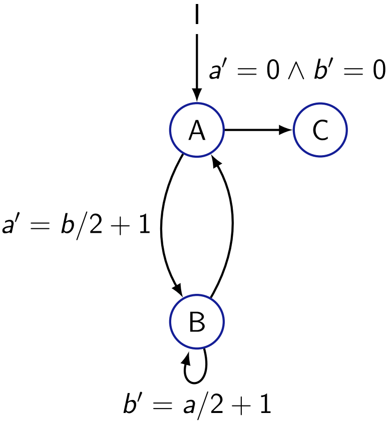
Policy Improvement

Implicitly chooses best policy locally

Example

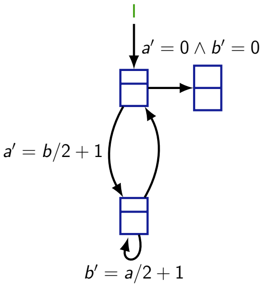
Applying LPI algorithm

```
double a = 0, b = 0;
while (input()) {
    a = b / 2 + 1;
    while (input()) {
        b = a / 2 + 1;
    }
}
```



Local Policy Iteration

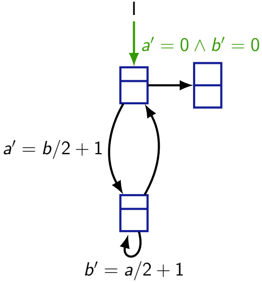
Algorithm Example



1. Start with $\top$ state a_0

Local Policy Iteration

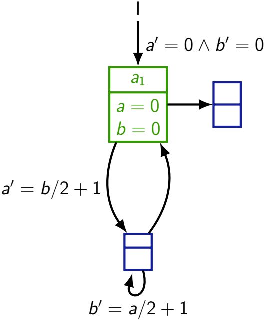
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1

Local Policy Iteration

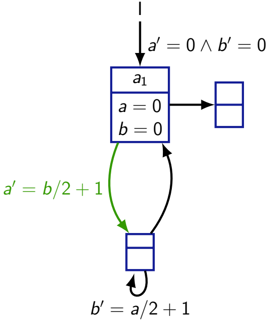
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A

Local Policy Iteration

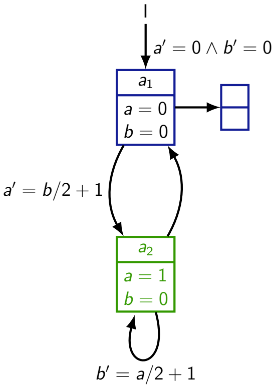
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2

Local Policy Iteration

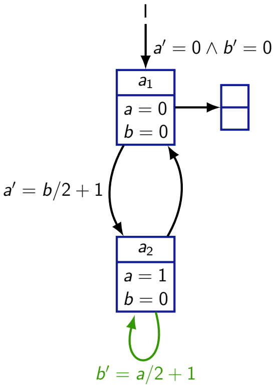
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B

Local Policy Iteration

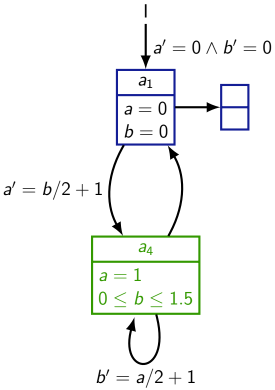
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3

Local Policy Iteration

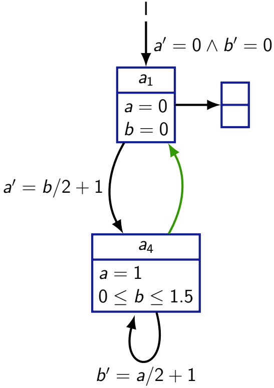
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B

Local Policy Iteration

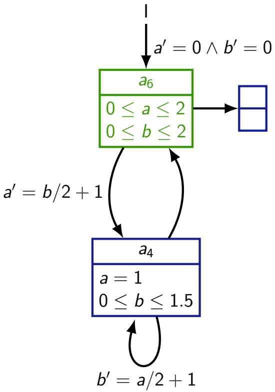
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B
8. Postcondition generates a_5

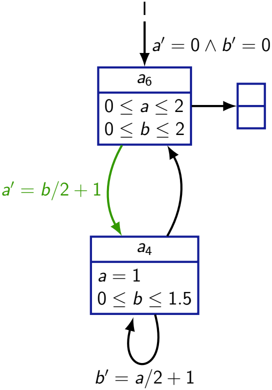
Local Policy Iteration

Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B
8. Postcondition generates a_5
9. Join a_5 and a_1, subsequent value determination generates a_6, associated with A

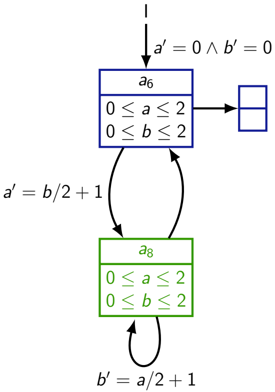
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B
8. Postcondition generates a_5
9. Join a_5 and a_1, subsequent value determination generates a_6, associated with A
10. ...

Local Policy Iteration

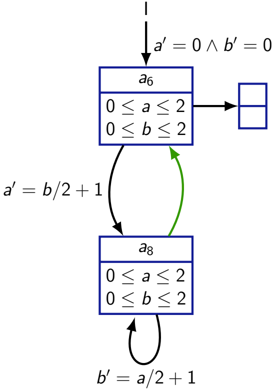
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B
8. Postcondition generates a_5
9. Join a_5 and a_1, subsequent value determination generates a_6, associated with A
10. ...

Local Policy Iteration

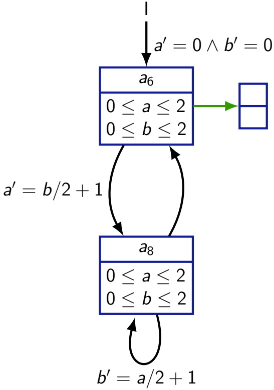
Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B
8. Postcondition generates a_5
9. Join a_5 and a_1, subsequent value determination generates a_6, associated with A
10. ...

Local Policy Iteration

Algorithm Example



1. Start with $\top$ state a_0
2. Postcondition generates new state a_1
3. Associate a_1 with node A
4. Postcondition generates new state a_2
5. Associate a_2 with node B
6. Postcondition generates a_3
7. Join a_2 and a_3, run subsequent value determination, associate result a_4 with B
8. Postcondition generates a_5
9. Join a_5 and a_1, subsequent value determination generates a_6, associated with A
10. ...

LPI Join Operator

- Joining states a_0 (previous), and a_1 (new)
- For each template $t \in T$:
  - Keep $a_0[t]$, unless bound in $a_1[t]$ is strictly larger
  - Guarantees feasibility
- If variable dependencies form a strongly connected component:
  - Launch value determination

Result

Together with postcondition simulates policy improvement

LPI Value Determination

- On updated templates:

LPI Value Determination

- On updated templates:
  - Reconstruct the equation system using the recorded policies
  - Recover the strongly connected component of variable dependencies
  - Solve LP to find the policy value

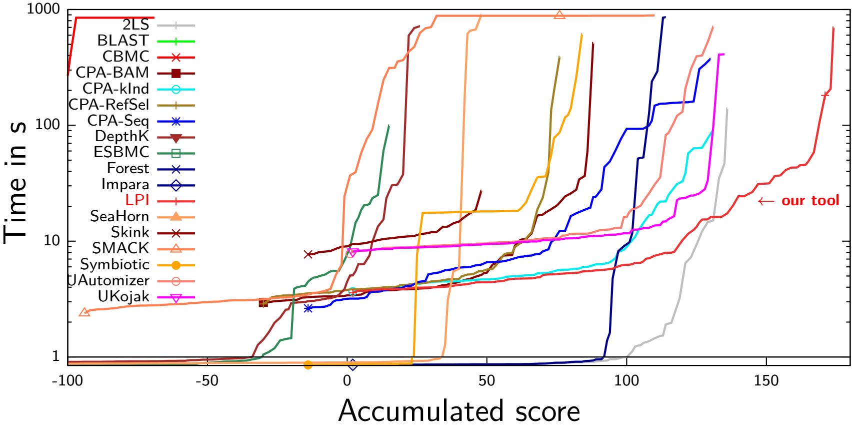
Value Determination Problem

Potentially size of the largest loop

SV-COMP Results 2016

Evaluation

First in Loops category!



Conclusion

LPI Features

- Local updates
- Update frontier
- Iteration order
  - Can use existing results
- Fits into existing frameworks
- Can be run in parallel with other analyses

LPI Formulation

Precise widening operator converging in finite number of steps

Outline

Template Generation Strategies

Annotations defining domain *shape*

- Combinatorial Synthesis
- Abstract reachability tree generation
  - Enables counterexample traces
  - Enables interpolation
- Synthesis using polyhedral analysis

  Generating templates using convex hull and projection
  - Offline
    - Generate templates using convex hull, use after restart
  - Online
    - Value determination before widening in polyhedral abstract interpretation

Underlying Theme

Refine template size on failed analysis

Template Generation Strategies

Annotations defining domain *shape*

- Combinatorial Synthesis
- Abstract reachability tree generation
  - Enables counterexample traces
  - Enables interpolation
- Synthesis using polyhedral analysis
  Generating templates using convex hull and projection
  - Offline
    - Generate templates using convex hull, use after restart
  - Online
    - Value determination before widening in polyhedral abstract interpretation

Underlying Theme

Refine template size on failed analysis

Combinatorial Enumeration

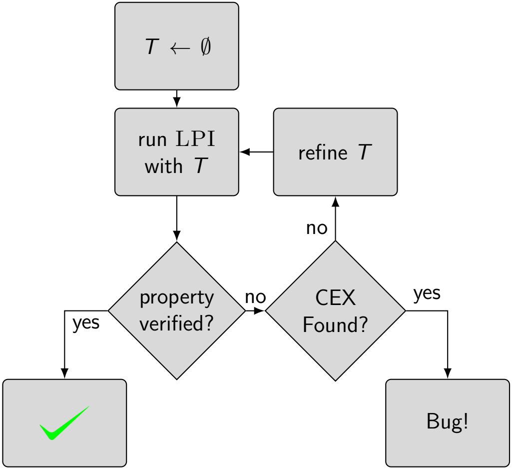
- Defining set of templates
  - Fix occurring constants (say, 1)
  - Fix expression size
  - E.g. vars: `int x, y`, size: 1, constants: $\{1, 0\}$
    - Generates $\{x, y\}$
- Refinement
  - raise the expression size
  - allow more constants
- Upper size bound: # of variables

Example

- For two variables x, y:

$\emptyset, \{x, y\}, \{x + y, x - y, -x - y, y - x\}, \ldots$

Combinatorial Enumeration



- Liveness & redundancy filtering
- Good results in practice

Abstract Reachability Tree Generation

- Goal: construct using abstract interpretation

Abstract Reachability Tree Generation

- Goal: construct using abstract interpretation
- Trick:
  - Do not join

Abstract Reachability Tree Generation

- Goal: construct using abstract interpretation
- Trick:
  - Do not join
  - Postcondition: if exists ancestor for same node, return join result
  - Ancestor: previous state, same branch, same node

Abstract Reachability Tree Generation

- Goal: construct using abstract interpretation
- Trick:
  - Do not join
  - Postcondition: if exists ancestor for same node, return join result
  - Ancestor: previous state, same branch, same node
  - For state s, ancestor a, transition τ
  - $\rightsquigarrow_t \equiv [\![\tau]\!]^\sharp(s) \sqcup a$ if a exists
  - $\rightsquigarrow_t \equiv [\![\tau]\!]^\sharp(s)$ otherwise

Abstract Reachability Tree Generation

- Goal: construct using abstract interpretation
- Trick:
  - Do not join
  - Postcondition: if exists ancestor for same node, return join result
  - Ancestor: previous state, same branch, same node
  - For state s, ancestor a, transition τ
  - $\rightsquigarrow_t \equiv [\![\tau]\!]^\sharp(s) \sqcup a$ if a exists
  - $\rightsquigarrow_t \equiv [\![\tau]\!]^\sharp(s)$ otherwise
- Templates from interpolants:
  - Mining for linear expressions
  - Combinatorial synthesis from occurring

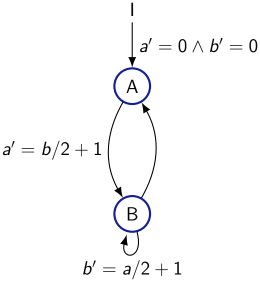
Abstract Reachability Tree Generation

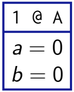
- Goal: construct using abstract interpretation
- Trick:
  - Do not join
  - Postcondition: if exists ancestor for same node, return join result
  - Ancestor: previous state, same branch, same node
  - For state s, ancestor a, transition τ
  - $\rightsquigarrow_t \equiv [\![\tau]\!]^\sharp (s) \sqcup a$ if a exists
  - $\rightsquigarrow_t \equiv [\![\tau]\!]^\sharp (s)$ otherwise
- Templates from interpolants:
  - Mining for linear expressions
  - Combinatorial synthesis from occurring

Tree Construction

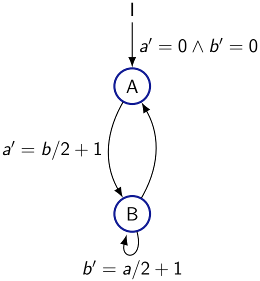
- Enables counterexample traces
- Enables interpolation

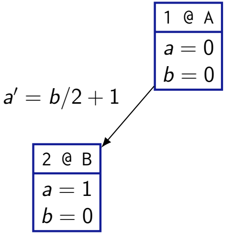
ART Generation Example



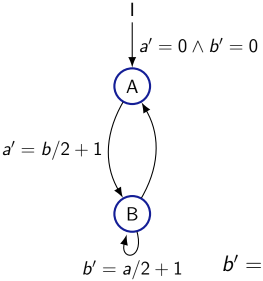


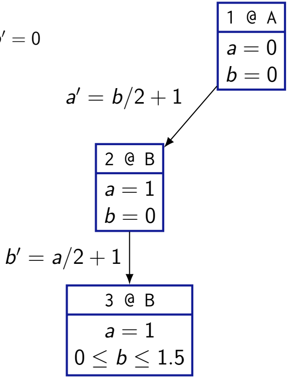
ART Generation Example



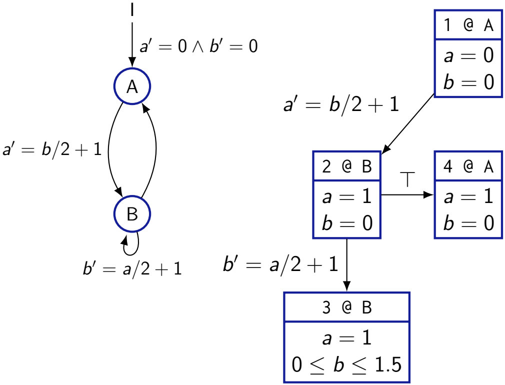


ART Generation Example

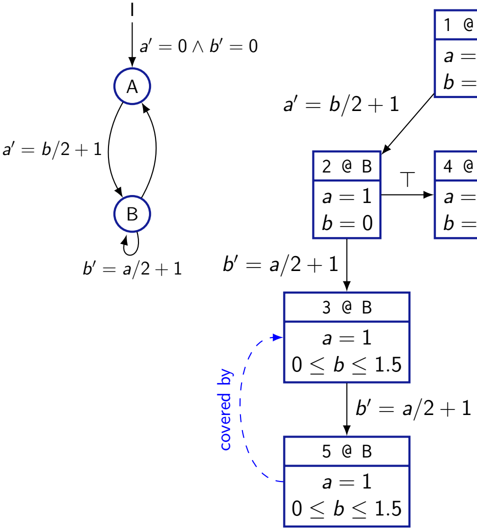




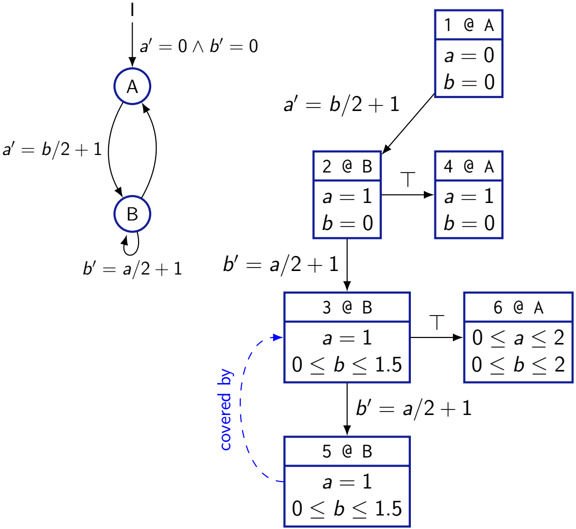
ART Generation Example



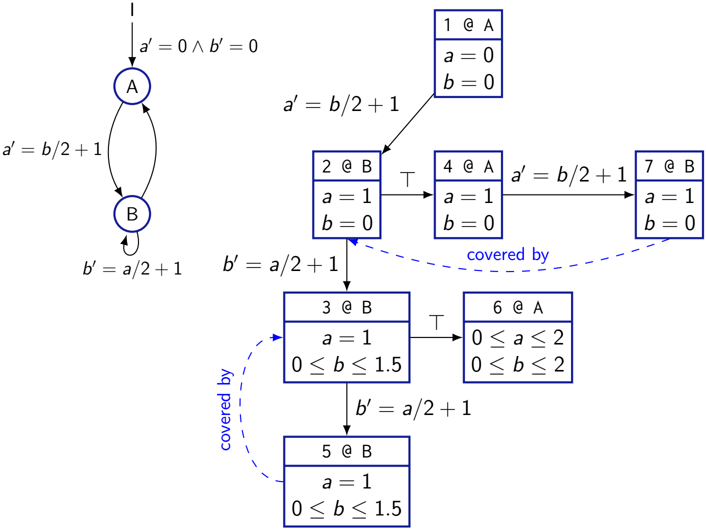
ART Generation Example



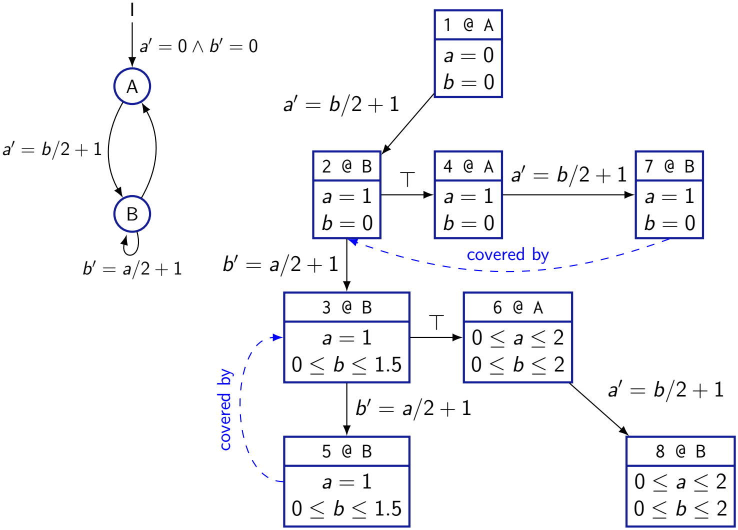
ART Generation Example



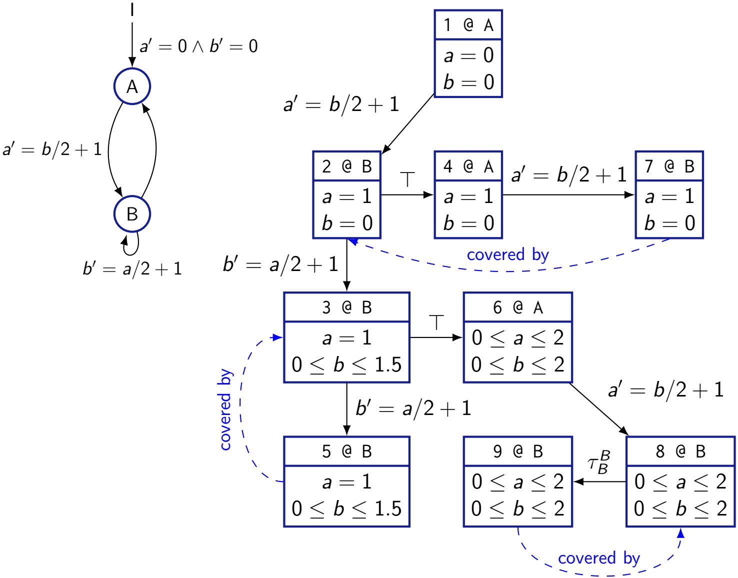
ART Generation Example



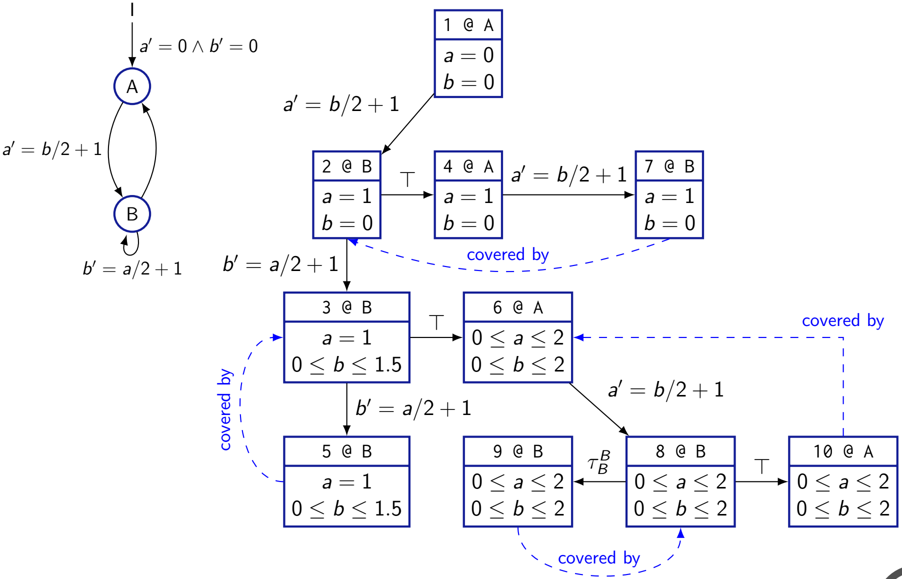
ART Generation Example



ART Generation Example



ART Generation Example



Outline

Summary Generation

- One-state invariants $P(\mathbf{x})$:
  - Inlining is exponential
  - No recursion support
- Solution: summarize functions
  - Invariants $S(\mathbf{x} \cup \mathbf{x}')$
  - Possible transitions within the function...
  - ...with valid calling context

Summary Equations

$$
\begin{aligned}
\text{Program Initiation: } & I_{f_m}^{m} = \top \\
\text{Consecution: } & \text{for all } (a, \texttt{OPS}, b) \in \mathit{edges}: \\
& [\![\texttt{OPS}]\!]^{\sharp}(I_f^a) \preceq I_f^b \\
\text{Function Call: } & \text{for all } (g, n_{call}, n_{ret}, \mathbf{x}_p, \mathbf{x}_o) \in \mathit{calledges}: \\
& I_f^{n_{call}}|_{\mathbf{x}_p}[\mathbf{x}_p / \mathbf{x}_i^g] \preceq I_f^{n_{en}} \\
\text{Summary Coverage: } & I_f^{n_{ex}}|_{\mathbf{x}_i \cup \mathbf{x}_r} \preceq S_f \\
\text{Function Application: } & \text{for all } (g, n_{call}, n_{ret}, \mathbf{x}_p, \mathbf{x}_o) \in \mathit{calledges}: \\
& I_f^{n_{call}}|_{\mathbf{x} \setminus \mathbf{x}_o} \sqcap S_g[\mathbf{x}_i^g / \mathbf{x}_p][\mathbf{x}_r^g / \mathbf{x}_o] \preceq I_f^{n_{ret}}
\end{aligned}
$$

Contribution

Generating least inductive summaries using policy iteration

Inductive Invariants from Preconditions

Formula Slicing

- Verification: loop-free program fragments can be exactly encoded as formulas

Inductive Invariants from Preconditions

Formula Slicing

- Verification: loop-free program fragments can be exactly encoded as formulas
- Reachability: SMT query

Inductive Invariants from Preconditions

Formula Slicing

- Verification: loop-free program fragments can be exactly encoded as formulas
- Reachability: SMT query
- Problem: loops
- Usual solution:
  - convex abstraction
  - very coarse

Inductive Invariants from Preconditions

Formula Slicing

- Verification: loop-free program fragments can be exactly encoded as formulas
- Reachability: SMT query
- Problem: loops
- Usual solution:
  - convex abstraction
  - very coarse
- Common pattern: long initialization, short loop

```
struct vmxnet3_adapter *adapter = netdev_priv(netdev);
u32 *buf = p;
int i = 0, j = 0;
memset(p, 0, vmxnet3_get_regs_len(netdev));
regs->version = 2;
buf[j++] = VMXNET3_READ_BAR1_REG(adapter, REG_VRRS);
buf[j++] = VMXNET3_READ_BAR1_REG(adapter, REG_UVRS);
buf[j++] = VMXNET3_READ_BAR1_REG(adapter, REG_DSAL);
// ...
buf[j++] = adapter->intr.num_intrs;
for (i = 0; i < adapter->intr.num_intrs; i++) {
    buf[j++] = VMXNET3_READ_BAR0_REG(adapter, ...);
}
```

Inductive Invariants from Preconditions

Formula Slicing

- Verification: loop-free program fragments can be exactly encoded as formulas
- Reachability: SMT query
- Problem: loops
- Usual solution:
  - convex abstraction
  - very coarse
- Common pattern: long initialization, short loop

```
struct vmxnet3_adapter *adapter = netdev_priv(netdev);
u32 *buf = p;
int i = 0, j = 0;
memset(p, 0, vmxnet3_get_regs_len(netdev));
regs->version = 2;
buf[j++] = VMXNET3_READ_BAR1_REG(adapter, REG_VRRS);
buf[j++] = VMXNET3_READ_BAR1_REG(adapter, REG_UVRS);
buf[j++] = VMXNET3_READ_BAR1_REG(adapter, REG_DSAL);
// ...
buf[j++] = adapter->intr.num_intrs;
for (i = 0; i < adapter->intr.num_intrs; i++) {
    buf[j++] = VMXNET3_READ_BAR0_REG(adapter, ...);
}
```

Our Solution

For program trace find inductive over-approximation over the loop

JavaSMT Library

- Satisfiability modulo theories solvers:
  - Ubiquitous in program analysis
- SMT-LIB initiative: often limited
- Solver API: vendor lock-in
- Solution:
  - Common API for using SMT solvers
  - Proper types, introspection, performance, etc.

Getting the Library

`https://github.com/sosy-lab/javasmt`

Outline

Conclusion

Contributions Overview

- New algorithms for inductive invariant synthesis
- LPI: unifying policy iteration and abstract interpretation
  - More accessible to engineers
- For all contributions:
  - Implementation in CPAchecker
  - Evaluation

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)
  - Suitable for overflow, array bounded-ness checks

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)
  - Suitable for overflow, array bounded-ness checks
- Non-convex invariants

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)
  - Suitable for overflow, array bounded-ness checks
- Non-convex invariants
  - E.g. via splitting states

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)
  - Suitable for overflow, array bounded-ness checks
- Non-convex invariants
  - E.g. via splitting states
- Integration with model-checking approaches

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)
  - Suitable for overflow, array bounded-ness checks
- Non-convex invariants
  - E.g. via splitting states
- Integration with model-checking approaches
  - Invariants for predicate analysis

Future Work

Charting the landscape

- Evaluation outside of SV-COMP: towards system verification
  - Verfication of modules (e.g. Kernel code)
  - Suitable for overflow, array bounded-ness checks
- Non-convex invariants
  - E.g. via splitting states
- Integration with model-checking approaches
  - Invariants for predicate analysis
  - Guiding model checking tools

Questions?

Thank you for your time!