A Few Variations on Policy Iteration

David Monniaux

VERIMAG

October 5, 2015
Real programs

Not just numeric variables!

- arrays
- recursive data structures
- pointers, pointer arithmetic
- procedures
- exceptions
- objects, closures
- parallelism

(We won’t cover that here.)
Approaches

**Bounded model checking** search for bugs up to length $n$

**Predicate abstraction** from proofs of absence of bugs on traces of length 1, 2, \ldots, derive an inductive argument

**Abstract interpretation** compute over-approximations of successor states in a fixed abstract domain (intervals, convex polyhedra)
Kleene Iterations with Widening

```plaintext
x = 0;
y = 1;
while (x < 1000) {
    x = x + 1;
    y = y + 2;
}
```

Compute polyhedra after $n$ iterations:

$x = 0 \land y = 1$
Kleene Iterations with Widening

\[ x = 0; \]
\[ y = 1; \]
\[ \textbf{while} \ (x < 1000) \{ \]
\[ \quad x = x + 1; \]
\[ \quad y = y + 2; \]
\[ \}\]

Compute polyhedra after \( n \) iterations:

\[ x = 0 \land y = 1 \quad \leadsto \quad x = 1 \land y = 3 \]
Kleene Iterations with Widening

```c
x = 0;
y = 1;
while (x < 1000) {
    x = x + 1;
y = y + 2;
}
```

Compute polyhedra after $n$ iterations:

$x = 0 \land y = 1$ $\leadsto x = 1 \land y = 3$

$0 \leq x \leq 1 \land y = 2x + 1$
Kleene Iterations with Widening

\[ x = 0; \]
\[ y = 1; \]
\[ \textbf{while} \ (x < 1000) \ { \}
\[ \quad x = x + 1; \]
\[ \quad y = y + 2; \]
\[ \}

Compute polyhedra after \( n \) iterations:

\[ x = 0 \land y = 1 \quad \leadsto \quad x = 1 \land y = 3 \]
\[ 0 \leq x \leq 1 \land y = 2x + 1 \quad \leadsto \quad 1 \leq x \leq 2 \land y = 2x + 1 \]
Kleene Iterations with Widening

x = 0;
y = 1;
while (x < 1000) {
    x = x + 1;
y = y + 2;
}

Compute polyhedra after $n$ iterations:

$x = 0 \land y = 1$  $\leadsto x = 1 \land y = 3$

$0 \leq x \leq 1 \land y = 2x + 1$  $\leadsto 1 \leq x \leq 2 \land y = 2x + 1$

$0 \leq x \leq 2 \land y = 2x + 1$
Kleene Iterations with Widening

\[ x = 0; \]
\[ y = 1; \]
\[ \textbf{while} \ (x < 1000) \{ \]
\[ \quad x = x + 1; \]
\[ \quad y = y + 2; \]
\[ \}

Compute polyhedra after \( n \) iterations:

\[ x = 0 \land y = 1 \quad \leadsto \quad x = 1 \land y = 3 \]
\[ 0 \leq x \leq 1 \land y = 2x + 1 \quad \leadsto \quad 1 \leq x \leq 2 \land y = 2x + 1 \]
\[ 0 \leq x \leq 2 \land y = 2x + 1 \quad \leadsto \quad 1 \leq x \leq 3 \land y = 2x + 1 \]
Kleene Iterations with Widening

x = 0;
y = 1;
while (x < 1000) {
    x = x + 1;
y = y + 2;
}

Compute polyhedra after $n$ iterations:

\[
\begin{align*}
x = 0 & \land y = 1 \quad \Leftrightarrow \quad x = 1 & \land y = 3 \\
0 \leq x \leq 1 & \land y = 2x + 1 \quad \Leftrightarrow \quad 1 \leq x \leq 2 & \land y = 2x + 1 \\
0 \leq x \leq 2 & \land y = 2x + 1 \quad \Leftrightarrow \quad 1 \leq x \leq 3 & \land y = 2x + 1 \\
\vdots
\end{align*}
\]

\[0 \leq x \land y = 2x + 1\]

\[\downarrow \text{widenin}\]

inductive invariant
Kleene Iterations with Widening

\[
x = 0; \\
y = 1; \\
\textbf{while} \ (x < 1000) \ { \\
\quad x = x + 1; \\
\quad y = y + 2; \\
}\}
\]

Compute polyhedra after \( n \) iterations:

\[
x = 0 \land y = 1 \quad \leadsto \quad x = 1 \land y = 3 \\
0 \leq x \leq 1 \land y = 2x + 1 \quad \leadsto \quad 1 \leq x \leq 2 \land y = 2x + 1 \\
0 \leq x \leq 2 \land y = 2x + 1 \quad \leadsto \quad 1 \leq x \leq 3 \land y = 2x + 1 \\
\vdots \\
0 \leq x \land y = 2x + 1 \\
0 \leq x \leq 1000 \land y = 2x + 1 \quad \text{inductive invariant} \\
0 \leq x \leq 1000 \land y = 2x + 1 \quad \text{other inductive invariant}
\]
Widening

Iterate $u_{n+1} = u_n \triangledown f(u_n)$ instead of $u_{n+1} = f(u_n)$. Enforce convergence in finite time to $f(u) \subseteq u$.

Examples:
- On polyhedra: drop all unstable constraints.
- On regular languages: enforce arbitrary limit on the number of states of the defining automata.

Once $f(u) \subseteq u$, $f(f(u)) \subseteq f(u)$, narrow down the invariant.
Overshooting

```c
int x = 0, y = 1;
while (true) {
    if (x >= 1000) break;
    if (nondeterministic()) {
        x = x + 1;
        y = y + 2;
    }
}
```

Now $u \subseteq f(u)$ for all $u$, narrowing cannot work. Invariant obtained $0 \leq x \land y = 2x + 1$. 
Non-Monotonicity

```c
int x = 0;
while(true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

$x = 0$

$x \rightarrow x = 1 \rightarrow x = 2 \rightarrow \ldots \ldots$

# widening

0 $\rightarrow$ 1 $\rightarrow$ ...

# inductive invariant

# narrowing

0 $\rightarrow$ 1 $\rightarrow$ ...

stay the same
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[ x = 0 \Rightarrow x = 1 \]
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

$x = 0 \implies x = 1$

$0 \leq x \leq 1$
Non-Monotonicity

```cpp
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[ x = 0 \quad \mapsto \quad x = 1 \]
\[ 0 \leq x \leq 1 \quad \mapsto \quad 1 \leq x \leq 2 \]
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

- $x = 0 \rightsquigarrow x = 1$
- $0 \leq x \leq 1 \rightsquigarrow 1 \leq x \leq 2$
- $0 \leq x \leq 2$
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[
x = 0 \implies x = 1
\]
\[
0 \leq x \leq 1 \implies 1 \leq x \leq 2
\]
\[
0 \leq x \leq 2 \implies 1 \leq x \leq 3
\]
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[ x = 0 \implies x = 1 \]
\[ 0 \leq x \leq 1 \implies 1 \leq x \leq 2 \]
\[ 0 \leq x \leq 2 \implies 1 \leq x \leq 3 \]
\[ \vdots \]
\[ 0 \leq x \quad \text{inductive invariant} \]

\[ \downarrow \text{widening} \]
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

$x = 0 \rightsquigarrow x = 1$

$0 \leq x \leq 1 \rightsquigarrow 1 \leq x \leq 2$

$0 \leq x \leq 2 \rightsquigarrow 1 \leq x \leq 3$

\[\vdots\]

$0 \leq x$ \hspace{1cm} inductive invariant \hspace{1cm} $\downarrow$ narrowing

$\downarrow$ widening
Non-Monotonicity

```c
int x = 0;
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[
\begin{align*}
x &= 0 & \implies x &= 1 \\
0 &\leq x \leq 1 & \implies 1 &\leq x \leq 2 \\
0 &\leq x \leq 2 & \implies 1 &\leq x \leq 3 \\
\vdots & & \vdots \\
0 &\leq x & \text{inductive invariant} \\
0 &\leq x & \text{stay the same}
\end{align*}
\]

\[\downarrow \text{widening}\]
\[\downarrow \text{narrowing}\]
Non-Monotonicity

```c
int x = [0, 9];
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

0 ≤ x ≤ 9
Non-Monotonicity

```c
int x = [0, 9];
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

0 ≤ x ≤ 9  inductive invariant
Non-Monotonicity

```c
int x = [0, 9];
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[0 \leq x \leq 9\quad \text{inductive invariant} \quad \downarrow \text{narrowing}\]
Non-Monotonicity

```java
int x = [0, 9];
while (true) {
    x = x + 1;
    if (x == 10) x = 0;
}
```

\[
\begin{align*}
0 & \leq x \leq 9 \quad \text{inductive invariant} \\
0 & \leq x \leq 9 \quad \text{stay the same}
\end{align*}
\]

Knowing that \( x = 0 \), more precise than \( 0 \leq x \leq 9 \), leads to worse results.

\[
[0, 0] \mapsto [0, +\infty) \\
[0, 9] \mapsto [0, 9]
\]
Solutions

- Extra trickery in widening (“widening with thresholds” or “widening up-to”, etc.).
- Extra trickery in iterations (“guided static analysis”, path-focused iterations, etc.).
- Acceleration: for some classes of input/output relations, the exact behavior of the loop is computable (see Radu).
- Exact least fixed point computation.
Interval bounds as unknowns

```c
int x = 0;
while (x < 1000) { /* -l ≤ x ≤ u */
    if (nondeterministic()) {
        x = x + 1;
    }
}
```

Two edges in loop:

- $x < 1000 \land x' = x + 1,$
Interval bounds as unknowns

```c
int x = 0;
while (x < 1000) { /* -l \leq x \leq u */
    if (nondeterministic()) {
        x = x + 1;
    }
}
```

Two edges in loop:

- \( x < 1000 \land x' = x + 1, \]
  \([-l, u] \mapsto [-(l - 1), \min(u, 999) + 1] \]
- \( x < 1000 \land x' = x, \]
  \([-l, u] \mapsto [-l, u] \)
Least Fixed Point Equation

Least interval containing 0 stable by:

- $[-l, u] \mapsto [-(l - 1), \min(u, 999) + 1]$
- $[-l, u] \mapsto [-l, u]$

Thus system for least solution:

$$l = \max(0, l - 1, l)$$
$$u = \max(0, \min(u, 999) + 1, u)$$

(In general, the system is not separable.)
Systems of Min-Max Equations

Equations $lhs = rhs$ where $rhs$ built from min, max and monotone linear affine forms.

(Resembles problem of finding values of 2-player Markov games. If you don’t know what this is, don’t worry.)

Simple idea: any solution of such a system induces a min-policy (resp. max-policy): choice of argument for each min (resp. max) operator.
Min-Policy Iteration

\[ u = \max(0, \min(u, 999) + 1, u) \]

\[ u = \max(0, u + 1, u) \]

Find least solution of \( u = \max(e_1, e_2, \ldots) = \)
find least solution of \( u \geq e_1 \land u \geq e_2 \ldots \)

Here \( u \geq 0 \land u \geq u + 1 \), least solution \( u = +\infty \) (in general, use linear programming).
Replace into min: \( \min(\infty, 999) = 999 \), policy is obviously bad.
Min-Policy Iteration (2)

\[ u = \max(0, \min(u, 999) + 1, u) \]

\[ \downarrow \]

\[ u = \max(0, 999 + 1, u) \]

Find least solution of \( u \geq 0 \land u \geq 1000 \land u \geq u \), least solution \( u = 1000 \).

\[ u = 1000 \] is a fixed point of \( u = \max(0, \min(u, 999) + 1, u) \), terminate.
Some results on Min-Policy Iteration

(See works by Goubault, Gaubert etc.)

- Converges to some fixed point...
- in at most $2^n$ iterations where $n$ is the number of “min”.
- Produces a sequence of inductive invariants (can safely stop at any time).
- Converges to least fixed point under some conditions on mapping.
- Can be extended to constraints $AX \leq B$, $A$ fixed matrix, $B$ vector of unknown bounds.
- Can be extended to some nonlinear constraints.
Max-Policy Iteration

\[ u = \max(0, \min(u, 999) + 1, u) \]
\[ \downarrow \]
\[ u = 0 \]

Is this stable?
Max-Policy Iteration

\[ u = \max(0, \min(u, 999) + 1, u) \]

\[ \downarrow \]

\[ u = 0 \]

Is this stable? No! \( x = 0 \in [0, 0] \mapsto x = 1 \notin [0, 0] \).

\[ u = \max(0, \min(u, 999) + 1, u) \]

\[ \downarrow \]

\[ u = \min(u, 999) + 1 \]

**Greatest finite fixed point:** largest \( u \) s.t. \( u \leq u + 1 \wedge u \leq 1000 \). Thus \( u = 1000 \).
Some results on Max-Policy Iteration

- Converges to the **least** fixed point...
- in at most $2^n$ iterations where $n$ is the number of “min”.
- Can be extended to constraints $AX \leq B$, $A$ fixed matrix, $B$ vector of unknown bounds.
- Can be extended to **implicit paths**.
Max-Policy iteration with implicit paths

In the original formulation, abstract at every program point
What’s wrong with abstracting everywhere

```c
if (abs(x) >= 0.1) {
    y = sin(x) / x - 1 + x;
} else {
    y = x - x*x/6 + x*x*x*x*x / 120;
}
```

$|x| \geq 0.1$ in first branch proves no division by zero. But it is non-convex.
In more detail

```c
if (x >= 0) {
    y = x;
} else {
    y = -x;
}
if (y >= 0.1) {
    y = sin(x)/x - 1 + x;
} else {
    y = x - x*x/6 + x*x*x*x/120;
}
```

|x| \(\geq 0.1\) in first branch proves no division by zero. But it is **non-convex**.
Distinguish paths

If there are $n$ tests, distinguish $2^n$ paths.
In policy iteration

\[ p \text{ incoming paths to a node} \leadsto p + 1 \text{ arguments to “max”}. \]
In policy iteration

\( p \) incoming paths to a node \( \rightsarrow p + 1 \) arguments to “max”.
\( 2^n \) incoming paths to a node \( \rightsarrow 2^n + 1 \) arguments to “max”.

Keep this exponential set of paths/arguments **implicit**.
Rephrase questions

“Is the currently selected argument to “max” truly the largest?”
Rephrase questions

“Is the currently selected argument to “max” truly the largest?”

SMT-solving query “Is there a transition from the current candidate invariant to outside of it.” (see Julien Henry)

If so, improve the policy using the corresponding path in code (extracted from the Booleans in the SMT-solving query).
Decision problem

Given:
- template $A$
- initial state
- transition relation (including $\lor$, $\exists$)
- state to be excluded

"Is there any inductive invariant of the form $AX \leq B$?"
Complexity

Obviously NP-hard

```plaintext
choose(a); choose(b); ...
if (propositional_formula(a, b, ...)) {
    x = 1;
} else {
    x = 0;
}
/* B */
```

Interval at $B$ is $[0, 0]$ or $[0, 1]$ depending on satisfiability.
Complexity

Obviously NP-hard

```c
choose(a); choose(b); ... 
if (propositional_formula(a, b, ...)) {
    x = 1;
} else {
    x = 0;
}
/* B */
```

Interval at $B$ is $[0, 0]$ or $[0, 1]$ depending on satisfiability.

In fact, $\Sigma_2^P$-complete (NP-complete with a co-NP-complete oracle).
Ongoing work

Instead of abstracting as a single polyhedron for a given control point, abstract as many polyhedra according to the values of

- Boolean variables
- given list of arbitrary predicates

Exponential blowup again! But using equivalence classes of Boolean states with respect to constraints.

Submitted to CAV 2013.
Executive summary

Max-policy iteration computes least inductive invariants in a domain.

Carries over to implicitly represented (SMT) sets of paths.