Invariant inference by abstract interpretation

David Monniaux

École polytechnique

3 février 2012
1. Intervals

2. Extrapolation

3. Backward / forward
Inductive vs non-inductive invariants

Reachable states
Least invariant as product of intervals
Least invariant as convex polyhedron
Inductive vs non-inductive invariants

Reachable states

Least invariant as product of intervals \textit{not inductive}

Least invariant as convex polyhedron
Inductive vs non-inductive invariants

Reachable states
Least invariant as product of intervals not inductive
Least invariant as convex polyhedron inductive
Best invariant in domain not computable

P();
\ x=0;

Best invariant at end of program, as interval?
P();
x = 0;

Best invariant at end of program, as interval?

[0, 0] iff P() terminates
∅ iff P() does not terminate

Entails solving the halting problem.
Recall the idea

Try to compute an interval for each variable at each program point using *interval arithmetic*:

```c
assume(x >= 0 && x<= 1);
assume(y >= 2 && y= 3);
assume(z >= 3 && z= 4);
t = (x+y) * z;
```

Interval for `z`?
Recall the idea

Try to compute an interval for each variable at each program point using \textit{interval arithmetic}:

\begin{verbatim}
assume(x >= 0 && x<= 1);
assume(y >= 2 && y= 3);
assume(z >= 3 && z= 4);
t = (x+y) * z;
\end{verbatim}

Interval for $z$? $[6, 16]$
Why is this interesting?

Let $t(0..10)$ an array.
Program writes to $t(i)$.

We must know whether $0 \leq i \leq 10$, thus know an interval over $i$. 
\begin{verbatim}
assume(x >= 0 && x <= 1);
y = x;
z = x - y;
\end{verbatim}

- The human (intelligent) sees $z = 0$ thus interval $[0, 0]$, taking into account $y = x$.
- Interval arithmetic does not see $z = 0$ because it does not take $y = x$ into account.
How to track relations

Using **relational domains**.

E.g. : keep

- for each variable an interval
- for each pair of variables \((x, y)\) an information \(x - y \leq C\).
- (One obtains \(x = y\) by \(x - y \leq 0\) and \(y - x \leq 0\).)

How to **compute** on that?
Bounds on differences
Suppose $x - y \leq 4$, computation is $z = x + 3$, then we know $z - y \leq 7$.

Suppose $x - z \leq 20$, that $x - y \leq 4$ and that $y - z \leq 6$, then we know $x - z \leq 10$.

We know how to **compute** on these relations (transitive closure / shortest path).

On our example, obtain $z = 0$. 
Why this is useful

Let $t(0..n)$ an array in the program. The program writes $t(i)$.

Need to know whether $0 \leq i \leq n$, otherwise said find bounds on $i$ and on $n - i$...
Can we do better?

How about tracking relations such as $2x + 3y \leq 6$?

At a given program point, a set of **linear inequalities**.

In other words, a **convex polyhedron**.
Example of polyhedron
Caveat

(In general) The more precise we are, the higher the costs. For each line of code:

- Intervals: algorithms $O(n)$, $n$ number of variables.
- Differences $x - y \leq C$: algorithms $O(n^3)$
- Octagons $\pm x \pm y \leq C$ (Miné): algorithms $O(n^3)$
- Polyhedra (Cousot / Halbwachs): algorithms often $O(2^n)$.

On short examples with few variables, ok... But in general?
Even linear may not be fast enough

Fly-by-wire control code from Airbus:

- Main control loop
- Number of tests linear in length $n$ of code
- Number of variables linear in length $n$ of code (global state)
- Complexity of naive convex hull on products of intervals linear in number of variables
Fly-by-wire control code from Airbus:

- Main control loop
- Number of tests linear in length $n$ of code
- Number of variables linear in length $n$ of code (global state)
- Complexity of naive convex hull on products of intervals linear in number of variables

$\Rightarrow$ Cost per iteration in $n^2$
Absolute value

\[ y = \text{abs}(x); \quad /* \text{valeur absolue} */ \]
\[ \text{if (} y \geq 1 \text{)} \{ \]
\[ \quad \text{assert}(x \neq 0); \]
\[ \} \]
Interval expansion

Intervals:

```c
/* -1000 <= x <= 2000 */
if (x < 0) y = -x; /* 0 <= y <= 1000 */
else y = x; /* 0 <= y <= 2000 */

if (y >= 1) {
    /* 1 <= y <= 2000 */
    assert(x != 0); /* -1000 <= x <= 2000 */
}
```
Branch $x \geq 0$
Other branch

Branch $x < 0$
After first test

\[ y = |x| = \text{union of the two red lines. Not a convex.} \]

Convex hull = pink polyhedron
Note: includes $(x, y) = (0, 1)$.
Disjunction

Possible if we do a union of two polyhedra:
- $x \geq 0 \land y = x$
- $x < 0 \land y = -x$

But with $n$ tests?
Two tests

```c
if (x >= 0) y = x; else y = -x;
if (y >= 1) z = y + 1; else z = y;
```

4 polyhedra = costly computations
Two tests, convex hull

More imprecise:
Sources of imprecision

- Need to distinguish each path and compute one polyhedron for each.
- But $2^n$ paths.
- Too costly if done naively.
- In current tools, not implemented.
- $\Rightarrow$ explains some imprecisions.
Current research

In the last few years articles propose methods distinguishing paths. Use of SMT-solving techniques to cut the exponential cost: Only look at “useful” paths.
1. Intervals

2. Extrapolation

3. Backward / forward
Loops?

Push intervals / polyhedra forward...

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

Loop iterations [0, 0], [0, 1], [0, 2], [0, 3],...

How? $\phi(X) = \text{état initial} \sqcup \text{post}(X)$, thus $\phi([a, b]) = \{0\} \sqcup [a + 1, \min(b, 999) + 1]$

**When do we stop?** Wait 1000 iterations? No.
One solution...

Extrapolation!

\([0, 0], [0, 1], [0, 2], [0, 3] \rightarrow [0, +\infty)\)

Push interval:

```java
int x=0; /* [0, 0] */
while /* [0, +infty) (x<1000) */ {
    /* [0, 999] */
    x=x+1;
    /* [1, 1000] */
}
```

Yes! \([0, \infty[\) is stable!
Mediocre results

Expected : 

Obtained : 

Run one more iteration of the loop : 

Obtain
Mediocre results

Expected : $[0, 999]$. Obtained $[0, +\infty)$.

Run one more iteration of the loop :

$[0, +\text{inf}ty) (x<1000)$

/* $[0, 999]$ */

$x=x+1$;

/* $[1, 1000]$ */

Obtain $\{0\} \sqcup [1, 1000] = [0, 1000]$. 
Narrowing

```c
int x=0; /* [0, 0] */
while /* [0,1000] (x<1000) {
    /* [0, 999] */
    x=x+1;
    /* [1, 1000] */
}

Yes! [0,1000] is an inductive invariant!
```
Stabilization

Look for a set (polyhedron, intervals)
- Containing initial values for the loop.
- **Inductive**: if valid at one iteration, valid at the next.

Look for $X$ such that $\phi(X) \subseteq X$ with
$\phi(X) = \text{états initiaux} \cup \text{post}(X)$
$\text{post}(X) = \text{states reachable from } X \text{ in one loop iteration}$

**Any** inductive invariant. (Not necessarily the least one.)
Computing the inductive invariant

We don’t know how to compute \( \text{post}(P) \) with \( P \) interval / polyhedron in general.
(The loop body may be complex, with tests...) Replace computation by simpler over-approximation
\( \text{post}(X) \subseteq \text{post}^\#(X) \).

Cannot do \( \cup \) over polyhedra, do \( \sqcup \) (convex hull)
Thus computation : \( \phi^\#(X) = \text{initial states } \sqcup \text{post}^\#(X) \)
Instead of \( \phi(X) \subseteq X \) with \( \phi(X) = \text{initial states } \cup \text{post}(X) \)
All the time, over-approximation

\[ \phi(X) \subseteq \phi^\#(X) \text{ so } \text{lfp } \phi \subseteq \text{lfp } \phi^\# \]

(work out the math, using \( \text{lfp } \psi = \inf \{ X \mid \psi(X) \subseteq X \} \))

In the end, **over-approximation** of the least fixed point of \( \phi \).
Graphical vision

Dark blue = concrete reachable states after $\leq 1$ loop iteration
Light blue = concrete reachable states after $\leq 2$ loop iterations
Dark red = over-approximated states after $\leq 1$ loop iteration
Light red = over-approximated states after $\leq 2$ loop iterations
Extrapolation
Where to extrapolate?

Extrapolation needed for termination: avoid iterating infinitely on cycles in control flow graph.

Need to extrapolate only at a limited set of points that break all cycles.

Choice of minimal set NP-complete. Minimal does not necessarily mean better precision.

Simple method: depth-first search for cycles.
Depth-first search:
init → loop1 → loop2 → init
backtrack to loop2, then loop2 → loop1
Mark init, loop1 as widening nodes
Minimal set

\[ \text{init} \rightarrow \text{loop1} \rightarrow \text{loop2} \]

\[ \text{init} \rightarrow \text{loop1} \rightarrow \text{loop2} \]
A bad invariant

```java
int i = 0;
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}
```

Analysis using widening will yield...
i = 0;
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}

Analysis using widening will yield
[0, 0], [0, 1], [0, 2], ..., [0, +\infty)
A bad invariant

```java
i = 0;
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}
```

Analysis using widening will yield

\([0, 0], [0, 1], [0, 2], \ldots, [0, +\infty)\)

Narrowing yields
A bad invariant

```java
i = 0;
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}
```

Analysis using widening will yield

\[ [0, 0], [0, 1], [0, 2], \ldots, [0, +\infty) \]

Narrowing yields

\[ [0, +\infty) \]
A bigger precondition

```java
i = [0, 99];
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}
```

Analysis using widening will yield

Note: with larger precondition, smaller inferred invariant.

Analysis with widening is non monotonic.

David Monniaux (École polytechnique)
Invariant inference by abstract interpretation
A bigger precondition

```java
i = [0, 99];
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}
```

Analysis using widening will yield
[0, 99], fixpoint reached

Note: with larger precondition, smaller inferred invariant.
Analysis with widening is non monotonic.
Workaround: widening with thresholds

Syntactic detection of comparisons

```java
i = 0;
while (true) {
    if (random()) {
        i = i + 1;
        if (i >= 100) i = 0;
    }
}
```

Detect $i \geq 100$, so 99 "magic value".

Widening: $[0, 0], [0, 1], \ldots, [0, 99]$

Applicable to intervals, octagons, polyhedra.
Consequences

- Over-approximate during computations (even without loops).
- Over-approximation during widening.
- Thus obtain super-set of reachable states.
- This super-set is an **inductive invariant** (cannot exit from it).
Practical consequences

- Cannot prove that a problem truly happens. Example: interval $i \in [0, 20]$ for access $t(0..10)$, is the interval exact?
- Yet sure that all potential problems are detected (over-approximation of problems).
- Let $B$ be the set of bad states. $X^\# \cap B \neq \emptyset$: “ORANGE”
- If $X^\# \subseteq B$, “RED”.
- What do orange vs red mean?
1 Intervals

2 Extrapolation

3 Backward / forward
Simple “avoid zero” example

```plaintext
if (x >= 0) {
    y = x;
} else {
    y = -x;
}
if (y >= 1) {
    assert(x != 0);
}
```

Forward analysis with polyhedra:

- \( P_2 = \{ x \geq 0 \land y = x \} \)
- \( P_4 = \{ x < 0 \land y = x \} \)
- \( P_5 = P_2 \sqcup P_4 = \{ y \geq x \land y \geq -x \} \)
- \( P_6 = P_5 \cap \{ y \geq 1 \} \)
Backward analysis

Move backward from $x = 0$ “bad state”, intersect each time with analysis result from forward.
Idea

Reachable = reachable from start
Co-reachable = co-reachable from a certain error condition

Forward : compute superset of reachable states
Forward then backward : compute superset of reachable $\cap$ co-reachable

and then
Forward then backward then forward etc.
Downwards iterations (every time, intersect with preceding).
Backward analysis over intervals

\[ z = x - y; \]

If you know \( z \in [0, 3] \) at the end, what do you get over \( x \) and \( y \)?
Backward analysis over intervals

\[ z = x - y; \]

If you know \( z \in [0, 3] \) at the end, what do you get over \( x \) and \( y \)? Nothing.
Forward-backward analysis over intervals

\[ z = x - y; \]

If you know \( z \in [0, 3] \) at the end, and \( x \in [0, 2] \), what do you get over \( y \)?
Forward-backward analysis over intervals

\[ z = x - y; \]

If you know \( z \in [0, 3] \) at the end, and \( x \in [0, 2] \), what do you get over \( y \)?

\[ y = x - z \text{ thus } y \in [-3, 2] \]
Forward / backward

Backward analysis alone: hardly usable on intervals, better for relational domains
Much better if preceded by forward analysis

Forward analysis first: don’t worry about states obviously unreachable
Backward analysis first: don’t worry about states obviously not co-reachable

In general, forward then backward.