Flavors of abstract interpretation

David Monniaux

CNRS / VERIMAG

April 9, 2024

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

Why abstract interpretation

Over-approximations of behavior of programs.
(And also under-approximations.)

- Prove that programs satisfy specifications.
- Study program behavior.
- Enable optimizations in compilers.

Abstract interpretation in a nutshell

What can happen in the program: R (undecidable as per Rice's theorem)
What we compute: $R^\sharp$
Soundness: $R \subseteq R^\sharp$

Program takes a step from R to R'
We compute: ${R^\sharp}'$

Limits of this tutorial

Vast topic

Will skim over many aspects

Focus on **numerical abstraction** because easier to visualize
(But will talk about other kinds of abstraction)

Will not cover underapproximations
Will not cover termination analysis

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

Plan

Introduction

Vanilla: finite lattices

- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

Rule of signs

Abstract integers into $\{\textbf{-}, \textbf{0}, \textbf{+}\}$.
To each $z \in \mathbb{Z}$, associate $s(z) = \textbf{+}$ if $z > 0$, $s(z) = \textbf{-}$ if $z < 0$, $s(0) = \textbf{0}$.

$\top = \{\textbf{-}, \textbf{0}, \textbf{+}\}$

x	y	$x + y$
-	**-**	**-**
-	**0**	**-**
-	**+**	$\top$
0	**-**	**-**
0	**0**	**0**
0	**+**	**+**
+	**-**	$\top$
+	**0**	**+**
+	**+**	**+**

Refinement for known constants

x	$x+1$
-	$\{$-$, 0\}$
0	+
+	+

From variable to state

Finite number of variables: abstract each variable separately.
Concrete state: $(x_1, \ldots, x_n)$
Abstract state: $(\alpha(x_1), \ldots, \alpha(x_n))$

Transform concrete $\to$ into abstract $\to^\sharp$
e.g. $(x, y) \xrightarrow{y:=x+y} (x', y')$ defined by $(x, y) \to (x, x + y)$

$(x^\sharp, y^\sharp) \xrightarrow{y:=x+y} \left(x^\sharp, y^{\sharp\prime}\right)$ for all $y^{\sharp\prime}$ in the plus abstract table.

e.g. $(\mathbf{+}, \mathbf{-}) \to^\sharp (\mathbf{+}, \mathbf{+})$
$(\mathbf{+}, \mathbf{-}) \to^\sharp (\mathbf{+}, \mathbf{0})$
$(\mathbf{+}, \mathbf{-}) \to^\sharp (\mathbf{+}, \mathbf{-})$

Verimag cnrs

Control locations

The control location is just another variable, often not abstracted.

If instruction from control location p to control location p' is
$y := x + y$;:
$(p, x, y) \to (p', x', y')$
and proceed as above

Simple data abstractions

Abstraction

To each $s \in \Sigma$ attach $\alpha(s) \in \Sigma^\sharp$.
To each $S \subseteq \Sigma$, define $\alpha(S) = \{\alpha(s) \mid s \in S\}$.

Replace $\mathcal{P}(\Sigma)$ (infinite) by $\mathcal{P}\left(\Sigma^\sharp\right)$ (finite).

Soundness

$\sigma \rightarrow \sigma' \implies \alpha(\sigma) \rightarrow^\sharp \alpha(\sigma')$

Most precise: $\sigma^\sharp \rightarrow^\sharp \sigma^{\sharp\prime}$ iff
$\exists \sigma, \sigma' \; \alpha(\sigma) = \sigma^\sharp \wedge \alpha(\sigma') = \sigma^{\sharp\prime} \wedge \sigma \rightarrow \sigma'$

Reachability analysis

Reachable states for $\rightarrow^\sharp$ are computable, because finite state, if $\rightarrow^\sharp$ is decidable.

Worklist graph traversal algorithm:

- start from initial state $\sigma_0^\sharp$, add $\sigma_0^\sharp$ to worklist
- until worklist empty, take $\sigma^\sharp$ from worklist, if not marked as "reached", mark it and add all its successors to worklist

Note on algorithmic results

Reachability in the abstract is uniquely defined.

The above algorithm computes the same set of abstract states regardless of worklist ordering.

Choices of ordering $\Longrightarrow$ cost of analysis issue only
(e.g. order worklist using **reverse postorder**)

Collecting by program point

We collect abstract states $(p, v_1^\sharp, \ldots, v_n^\sharp)$. We can group them by control location (program point) p.

For each program point, compute a set of reachable abstract states $(v_1^\sharp, \ldots, v_n^\sharp)$ where $v_1, \ldots, v_n \in \{\mathbf{-}, \mathbf{0}, \mathbf{+}\}$.

In other words, to each p, associate $R^\sharp(p) \subseteq \{\mathbf{-}, \mathbf{0}, \mathbf{+}\}^n$ similar to collecting $R(p)$ reachable program states at control location p.

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

Independent abstraction between variables

Instead of any $R^\sharp(p) \subseteq \{\textbf{-}, \textbf{0}, \textbf{+}\}^n$
consider only Cartesian products $\prod_{i=1}^{n} R^\sharp(p, i)$ where
$R^\sharp(p, i) \subseteq \{\textbf{-}, \textbf{0}, \textbf{+}\}$

In other words: $R^\sharp(p)$ is either

- $\bot$ "location is unreachable"
- a map from $1 \dots n$ to $\mathcal{P}(\{\textbf{-}, \textbf{0}, \textbf{+}\}) \setminus \{\}$

"Smashed bottom" = "if one variable cannot contain a value, then the instruction is unreachable"

Difference between dependent and independent abstraction

1: $y := x$
2: if $x = 0$:
3:     if $y \neq 0$:
4:         here

Variables: x, y

Dependent

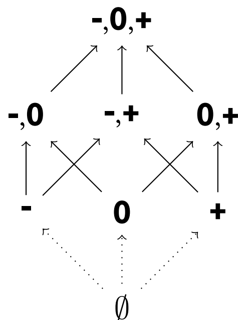
$R^\sharp(2) = \{(\mathbf{-}, \mathbf{-}), (\mathbf{0}, \mathbf{0}), (\mathbf{+}, \mathbf{+})\}$
$R^\sharp(3) = \{(\mathbf{0}, \mathbf{0})\}$
$R^\sharp(4) = \emptyset$

Independent

$R^\sharp(2) = \{\mathbf{-}, \mathbf{0}, \mathbf{+}\} \times \{\mathbf{-}, \mathbf{0}, \mathbf{+}\}$
$R^\sharp(3) = \{\mathbf{0}\} \times \{\mathbf{-}, \mathbf{0}, \mathbf{+}\}$
$R^\sharp(4) = \{\mathbf{0}\} \times \{\mathbf{-}, \mathbf{+}\}$

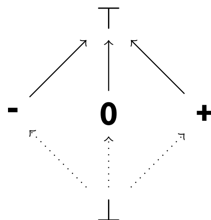
Powerset lattice

To each program location and each variable, attach a nonempty subset of $\{\mathbf{-}, \mathbf{0}, \mathbf{+}\}$.



A simpler lattice

To each program location and each variable, attach an element of this lattice $L^\sharp$



n variables, take "smashed bottom" product lattice $\left(L^\sharp\right)^n$ (one $\bot = \bot$ everywhere)

What this means, variable per variable

Concretization function

γ maps a lattice element to the values it represents.

$\gamma(\top) = \mathbb{Z} \quad \gamma(\textbf{-}) = (-\infty, 1] \quad \gamma(\textbf{0}) = \{0\} \quad \gamma(\textbf{+}) = [1, +\infty)$

Abstraction function

$\alpha(\emptyset) = \bot$

For $S \subseteq (-\infty, 1]$, $S \neq \emptyset$, $\alpha(S) = \textbf{-}$

For $S \subseteq [1, +\infty)$, $S \neq \emptyset$, $\alpha(S) = \textbf{+}$

$\alpha(\{0\}) = \textbf{0}$

$\alpha(S) = \top$ otherwise

Abstraction for a vector of variables

Concretization

$\gamma_v(l^\sharp_1, \dots, l^\sharp_n) = \{x_1, \dots, x_n \mid \forall i\ x_i \in \gamma(l^\sharp_i)\}$

Abstraction

$\alpha_v(A) = \{a^\sharp_1, \dots, a^\sharp_n\}$
$a^\sharp_i = \bigsqcup_{x_1,\dots,x_n \in A} \alpha(x_i)$
α, γ form a **Galois connection**.

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

An algorithm for inferring invariants

Worklist graph traversal algorithm:

- start from initial state $\left(p_0, \sigma_0^\sharp\right)$, set $R^\sharp(p_0) := \sigma_0^\sharp$, add p_0 to worklist
- until worklist empty, take p in worklist; for each transition $p \xrightarrow{op} p'$:
  - compute $y^\sharp := R^\sharp(p') \sqcup op^\sharp(R^\sharp(p))$
  - if $y^\sharp \neq R^\sharp(p')$, set $R^\sharp(p') := y^\sharp$ and add p' to the worklist

Termination and soundness

Termination

At every iteration, at least one $R^\sharp(p)$ increases, within a finite domain

Soundness

When it terminates, for any transition $p \xrightarrow{op} p'$: $op^\sharp(R^\sharp(p)) \sqsubseteq (R^\sharp(p'))$.
Consequence: if $(p, \sigma) \xrightarrow{op} (p', \sigma')$, $\alpha(\sigma) \in R^\sharp(p)$, then $\alpha(\sigma') \in R^\sharp(p')$.

In other words, the $R^\sharp(p)$ define inductive invariants.

Optimality

Assuming

- $op^\sharp$ is monotone
- $\sqcup$ computes least upper bound

Then this algorithm computes the least fixed point.
$R^\sharp(p)$ everywhere for least inductive invariant expressed by α.

Optimal only among inductive invariants

```
i = 6;
do { i -=2; } while(i $\neq$ 0);
```

$$p_1 \xrightarrow{i := 6} p_2 \xrightarrow{i = 0} p_3$$

(loop on p_2: $i := i - 2;\ i \neq 0$)

Concrete reachable states at p_2: $\{6, 4, 2\}$.
$\alpha(\{6, 4, 2\}) = \{\mathbf{+}\}$

Abstract reachable states at p_2: $\{\mathbf{-}, \mathbf{0}, \mathbf{+}\}$
(because $1 \to -1$, 1 abstractly reachable but not concretely)

Plan

Introduction

Vanilla: finite lattices

- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

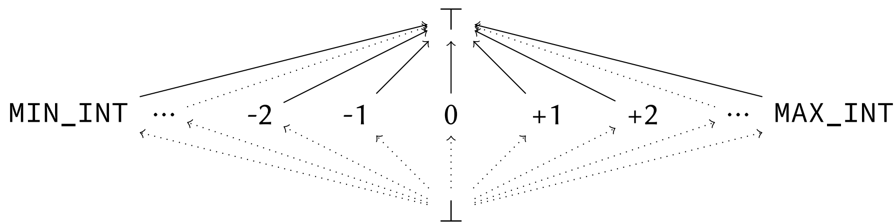
Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

Constant propagation



Example: CompCert

CompCert (2022 ACM System Software Award)

"Value analysis" computes fixed point in a (complicated) finite lattice with points-to analysis:

- constant propagation
- local strength reduction of instructions with known parameters

Distinguishes pointers

- points into local stackframe, at known or unknown offset
- points out of local stackframe
  - points into global variable, at known or unknown offset

"Is matched by"

CompCert has predicates:

- pointer p is matched by abstract pointer $p^\sharp$ [according to block classification C]
- pointer v is matched by abstract value $v^\sharp$ [according to block classification C]

$$\gamma(v^\sharp) = \{v \mid \mathit{vmatch}(C, v^\sharp, v)\}$$

Proofs that if $\forall i, \mathit{vmatch}(C, v_i, v_i^\sharp)$, $\mathit{vmatch}(C, \mathit{op}(v_1, \ldots, v_n), \mathit{op}^\sharp(v_1^\sharp, \ldots, v_n^\sharp))$

Fixed-point proof: if fixed-point iterations converge within N steps, then the result is inductive.

Example: forward dataflow analysis

Finite set $P_1, \ldots, P_n$ of predicates over program states = subsets of program states
Abstract element: $S^\sharp \subseteq \{1 \ldots n\}$
$\gamma(S^\sharp) = \bigcap_{i \in S^\sharp} P_i$

$S^\sharp \sqsubseteq S^{\sharp\prime}$ iff $S^{\sharp\prime} \subseteq S^\sharp$
Note: opposite direction, dataflow analysis usually presented with opposite ordering as abstract interpretation

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

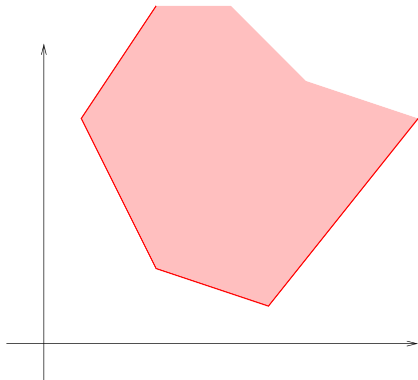
Coffee: paths and direction

Fudge: widenings

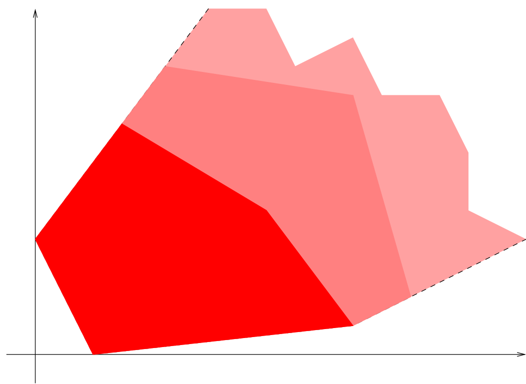
Blackcurrant: accelerated solving

Fiore di latte: conclusion

Convex polyhedra



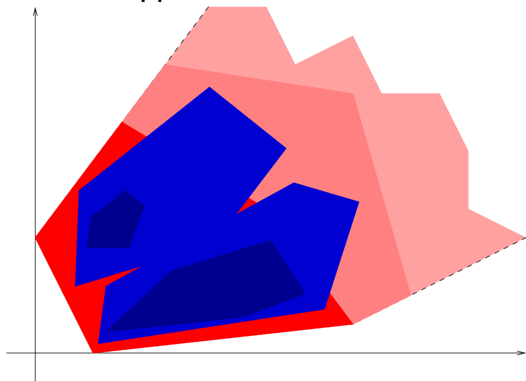
Convex polyhedra: widening



(Possibility: thresholds = linear inequalities found in program)

Convex polyhedra

Two overapproximations: the abstraction + the widening!



Note about Galois connections

$\alpha(S)$ is the **best overapproximation** of S in the abstract domain.

A disc has no best overapproximation as a convex polyhedron.

Cannot define α in general.

"Constructive" views of abstract interpretation often just define γ.

Absolute value

```
y = abs(x);
if (y >= 1) {
  assert(x != 0);
}
```

Intervals

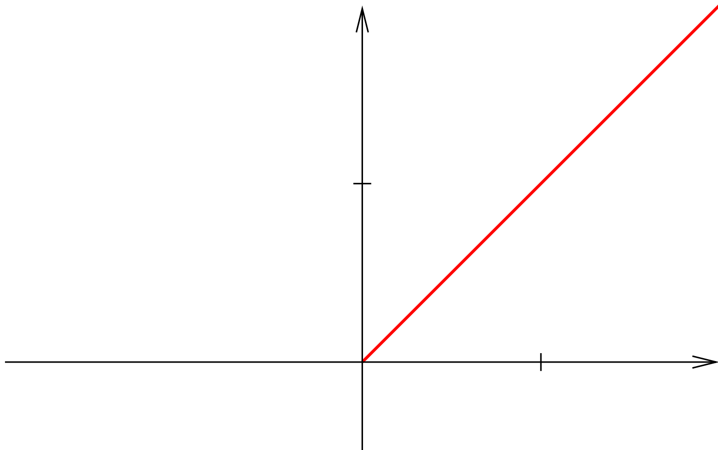
Intervals:

```
/* -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 !!! */
}
```

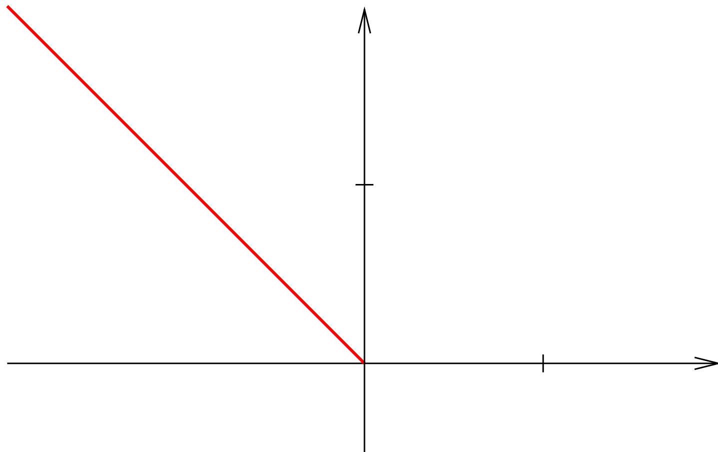
Polyhedra

Branch $x \geq 0$



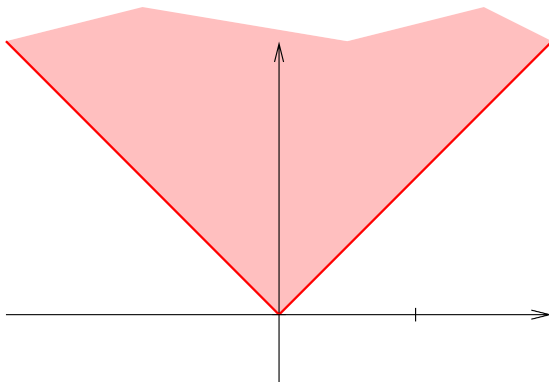
Other branch

Branch $x < 0$



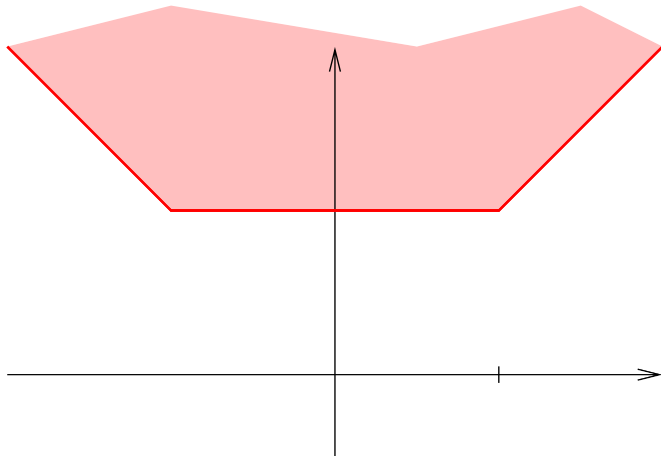
After first test

$y = |x|$ = union of the two red lines. Not a convex.
Convex hull = pink polyhedron



At second test

Note: includes $(x, y) = (0, 1)$.



Disjunction

Possible if we do a union of two polyhedra:

- $x \geq 0 \wedge y = x$
- $x < 0 \wedge y = -x$

But with n tests?

Sources of imprecision

- Need to distinguish **each path** and compute one polyhedron for each.
- But 2^n paths.
- **Too costly** if done naively.
- Use SMT-solving to distinguish individual paths (as e.g. PAGAI tool, see Henry's PhD thesis)

Forward analysis, reminder

Compute $I_p^\sharp$ at all position p in forward direction (next-state)
$\gamma(I_p^\sharp)$ contains all memory/variable states reachable at control position p

To prove that an undesirable control position p is unreachable: check $I_p^\sharp = \bot$

Forward / backward analysis

Compute $I_p^\sharp$ at all position p by combined forward/backward

We want:
$\gamma(I_p^\sharp)$ contains all memory/variable states at control position p reachable (from program start) and co-reachable from error location

Compute back from error location

```
/* false */
if (x >= 0)
  y = x;                /* x = 0 ∧ x ≥ 1 ≡ false */
else
  y = -x;               /* x = 0 ∧ −x ≥ 1 ≡ false */
if (y >= 1) {           /* x = 0 ∧ y ≥ 1 */
  assert(x != 0);       /* x = 0 */
}
```

Forward / backward

More generally: compute forward from program start, then backward from error location, possibly forward again.

Forward restricted to postcondition

$y^{\sharp\prime} = \text{forward}(op, x^\sharp, x^{\sharp\prime})$

$$\forall x, x',\ x \in \gamma(x^\sharp) \land x \xrightarrow{op} x' \land x' \in \gamma(x^{\sharp\prime}) \implies x' \in \gamma(y^{\sharp\prime})$$

Backward restricted to precondition

$y^\sharp = \text{backward}(op, x^{\sharp\prime}, x^\sharp)$

$$\forall x, x',\ x' \in \gamma(x^{\sharp\prime}) \land x \xrightarrow{op} x' \land x \in \gamma(x^\sharp) \implies x \in \gamma(x^\sharp)$$

Why restrictions to precondition/postcondition

(See optional parameter in e.g APRON)

$\text{backward}(op, x^{\sharp\prime}, x^{\sharp}) = \text{backward}(op, x^{\sharp\prime}) \sqcap x^{\sharp}$ would be valid.
But less precise!

Precondition: $x \in [0,3]$, postcondition $\top$, instruction: assume $x \le y$.

Backward analysis of assume $x \le y$ from $\top$ in the interval domain: $\top$, intersection with $x \in [0,3]$ is $x \in [0,3]$

Backward analysis knowing $x \in [0,3]$ yields $x \in [0,3] \wedge y \in [0,\infty)$

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

Bounded interval analysis

Elements of the lattice: pairs of integers (a, b), $a \leq b$, or $\perp$

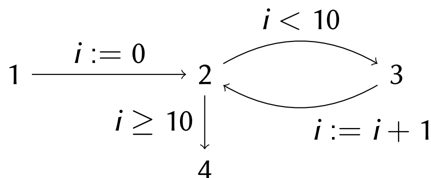
$$\alpha(S) = (\min S, \max S)$$
$$\gamma((a, b)) = a \dots b$$

$(a, b) \sqsubseteq (a', b')$ is $\leq a \leq b \leq b'$
(note: $\sqsubseteq$ a kind of decidable inclusion, we need
$l \sqsubseteq l' \implies \gamma(l) \subseteq \gamma(l')$)

Finite height lattice, largest [`MIN_INT`, `MAX_INT`]

On an example

```
i = 0;
while (i < 10) {
    i++;
}
```



First iteration: $[0, 0]$ goes through $i < 10$, $[1, 1]$ at line 3, $\sqcup$ at line 2 yields $[0, 1]$

Ensuing iterations at line 2: $[0, 2]$, $[0, 3]$, $[0, 4]$, ..., $[0, 10]$

Objection

What if we have to iterate to $\top = [\texttt{MIN_INT}, \texttt{MAX_INT}]$?

2^{31} or even 2^{63} iterations.

Need a way to accelerate!

Standard widening operator on intervals

Ascending right bounds $[0, 1]$, $[0, 2]$...try $[0, \texttt{MAX_INT}]$ (or $[0, +\infty)$).

$[0, \texttt{MAX_INT}]$ indeed an inductive invariant for

```
i = 0;
while(i < 10) {
  i++;
}
```

Obviously not the strongest! (which is $[0, 10]$)

Thresholds

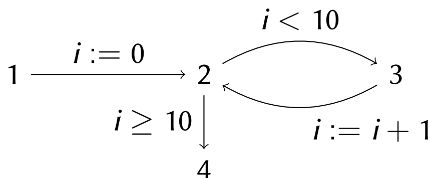
(Reinvented several times)

- Notice (syntactically or by dynamic recording) that there is a $i < 10 \equiv i \leq 9$ comparison.
- Widen to 9 then 10 instead of `MAX_INT`

Gets $i \in [0, 10]$

Narrowing step

```
i = 0;
while(i < 10) {
  i++;
}
```



If at location 2, we come from 1 or 3:

- either we start the loop, $i \in [0, 0]$
- either we have already gone through the loop, $2 \to 3 \to 2$, thus executing $i < 10$; $i := i + 1$ from $i \in [0, \texttt{MAX_INT}]$: getting $i \in [1, 10]$

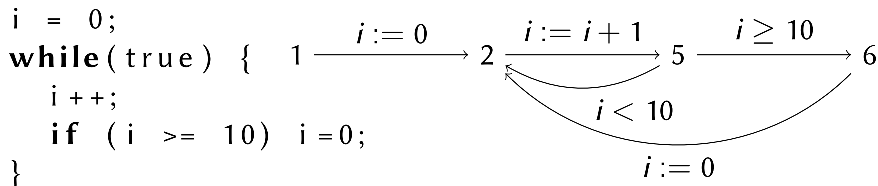
Thus at 2, i must be in $[0, 10]$!

A more mathematical view

We have an inductive invariant S: $f(S) \subseteq S$.

f (concrete semantics) is monotone (more states in precondition, more states in the outcome): $f(f(S)) \subseteq f(S)$
$f(S)$ is also an inductive invariant, and maybe $f(S) \subsetneq S$!

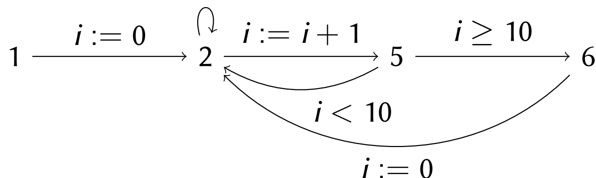
Narrowing works



Widening: $i \in [0, MAX_INT]$
Narrowing: $i \in [0, 9]$

Narrowing is foiled

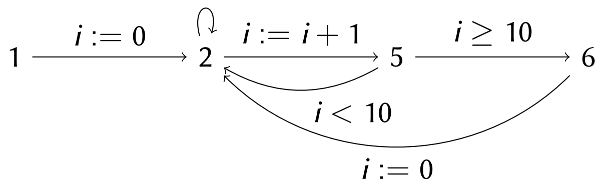
```
i = 0;
while(true) {
  if (*) {
    i++;
    if (i >= 10)
      i=0;
  }
}
```



Because of the self-loop, the "next iteration" operator satisfies $S \subseteq f(S)$ and thus narrowing never narrows.

Wider precondition

```
i = [0,9];
while(true) {
  if (*) {
    i++;
    if (i >= 10)
      i=0;
  }
}
```



No iterations needed, we have the invariant $[0, 9]$ straight from the start!

Non-monotonic behavior

Precondition $i = 0$: analysis computes $i \in [0, \texttt{MAX_INT}]$
Precondition $i \in [0, 9]$: analysis computes $i \in [0, 9]$

A more precise precondition leads to a less precise analysis result!
Counter-intuitive for end users.

(Other cause of non-monotonic behavior)

A long time ago in a galaxy far, far away.

(Other cause of non-monotonic behavior)

A long time ago in a galaxy far, far away.
In the **Astrée analyzer**.

Rewriting system + intervals $y \in [0, 10]$
$y \to x + 1$
$z \to 3 \times y$

Straight computation for $t := z + 1$ yield $\top$.
Full rewriting of $t := z + 1$ yields $t := 3(x + 1) + 1$, yields $\top$.
Partial rewriting (forget $y \to x + 1$) yields $t := 3y + 1$,
yields $t \in [1, 31]$.

Partial propagation of information for efficiency $\to$ **non-monotonic** behavior.

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

The problem on which narrowing failed

```
1  i = 0;
2  while(true) {
3    if (*) {
4      i++;
5      if (i >= 10)
6        i=0;
7    }
8  }
```

Write the interval analysis symbolically (forget handling of possibly empty intervals): $[-l_1, h_1] = [0, 0]$,
$[-l_2, h_2] = [-\max(l_1, l_8), \max(h_1, h_8)]$, $[-l_5, h_5] = [-(l_2 - 1), h_2 + 1]$,
$[-l_6, h_6] = [-l_5, \min(h_5, 9)]$, $[-l_7, h_7] = [-\max(l_6, 0), \max(h_6, 0)]$,
$[-l_8, h_8] = [-\max(l_2, l_7), \max(h_2, h_7)]$.

In a nutshell

$$l_2 = \max(0, \max(l_2, \max(l_2 - 1, 0)))$$
$$h_2 = \max(0, \max(h_2, \max(\min(h_2 + 1, 9), 0)))$$

(separated equations on this simple examples, in general not)

Any solution in (l_2, h_2) yields an inductive invariant in intervals.
How to solve such equations? (Outside of SMT-solving them.)

Descending policy iterations

(Many publications in E. Goubault's group, see also P.L. Garoche, P. Roux)

"$\min(a, b)$ must be equal to either a or b"

$h_2 = \max(0, \max(h_2, \max(\min(h_2 + 1, 9), 0)))$ can become

- $h_2 = \max(0, \max(h_2, \max(h_2 + 1, 0)))$: $h_2 = +\infty$ as only solution (no real solution)
- $h_2 = \max(0, \max(h_2, \max(9, 0)))$: $h_2 = 9$ as only solution

Thus $h_2 = 9$ as only solution!

Heuristic for descending iterations

$h_2 = \max(0, \max(h_2, \max(h_2 + 1, 0)))$ and
$h_2 = \max(0, \max(h_2, \max(9, 0)))$ correspond to the original program with one guard (test) over-approximated:
$i < 10$ means the interval for i

- either is the same (the bound has no effect, the test is always taken)
- or is truncated by 9

Heuristic: tests are likely to be useful, not always taken, thus try the second case first!

Solving the simplified system

Ordinary abstract interpretation

Run a regular abstract interpreter on a simplified program (simpler interpretation of guards/tests).

Exact solving

Least solution of $h_2 = \max(0, \max(h_2, \max(h_2 + 1, 0)))$: by monotonicity, least solution of

$$
\begin{aligned}
h_2 &\geq 0 \\
h_2 &\geq h_2 \\
h_2 &\geq h_2 + 1
\end{aligned}
$$

Solve by linear programming: no real solution.

Downward iterations

Assume we solve and get $h_2 = +\infty$.
Evaluate $\max(0, \max(h_2, \max(\min(h_2 + 1, 9), 0)))$ with $h_2 = +\infty$, get 9.
The solution of the simplified system is not a solution of the original system.

Flip the choice for min to a number yielding a lower value in the current solution!

Downward policy iteration

"Strategy" or "policy" iteration by similarity with approach for solving Markov decision processes and games.

- Pick argument for min or even inf occurring in the equation system (= simplify tests and reductions).
- Solve the simplified problem exactly or approximately.
- Replace the solution into the original problem, check if solution.
- If not solution, switch to other choices for min or inf and restart.

All intermediate systems over-approximate the original, thus their solved solutions over-approximate the least solution of the original system.
Can stop at any point and remain sound!

Treatment of relational abstract domains

$$x \leq A$$
$$y \leq B$$
$$x + y \leq C$$

can be reduced with e.g. $x + y \leq A + B$ thus $C' = \min(C, A + B)$

min or inf operations occur explicitly or implicity in bound computations (e.g. dual linear programming = take a minimum over Farkas witnesses)

Also to be treated by downward policy iteration!

Take-home message

Downward policy iteration

- computes downward sequence of simpler fixed-points
- sequence may be stopped at any time, producing a valid inductive invariant
- not guaranteed to converge to least fixed-point (= least inductive invariant in the abstract domain) but often does
- good heuristic choice of initial "policy" (choice of min-argument) matters

Max-policies

$h_2 = \max(0, h_2, \min(h_2 + 1, 9))$

Each max operator has value one of its arguments, add also $-\infty$

- $h_2 = -\infty$
- $h_2 = 0$
- $h_2 = h_2$
- $h_2 = \min(h_2 + 1, 0)$

Start with $h_2 = -\infty$.

Max-policy iterations

(Many publications from H. Seidl)

1. $h_2 = -\infty$ replaced in $\max(0, h_2, \min(h_2 + 1, 9))$:
   $\max(0, -\infty, \min(-\infty + 1, 9) = 0 > -\infty$, pick 0 (2nd argument) instead
2. $h_2 = 0$ replaced in $\max(0, h_2, \min(h_2 + 1, 9))$:
   $\max(0, 0, \min(0 + 1, 9)) = 1 > 0$, pick 1 (3rd argument) instead
3. $h_2 = \min(h_2 + 1, 9)$; solve for least solution and get $h_2 = 9$

Max-policy iterations in a nutshell

Replace a least fixed-point computation by an ascending sequence of fixed-point computations

Must go on until no “improvement” possible.

Converges to strongest inductive invariant in domain / least fixed point

Another example

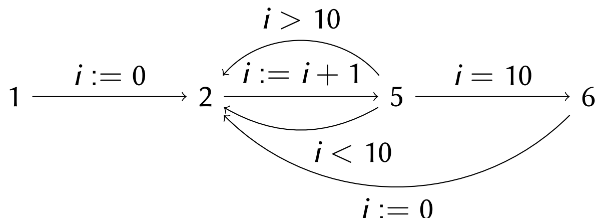
```
i = 0;
while(true) {
  i++;
  if (i == 10)
    i=0;
}
```

Widening to $+\infty$, narrowing does not help.

Guided static analysis

Idea: replace an invariant computation over the full program by a sequence of invariant computations over partial programs.
Partial program = subset of control-flow graph

```
i = 0;
while(true) {
  i++;
  if (i == 10)
    i=0;
}
```

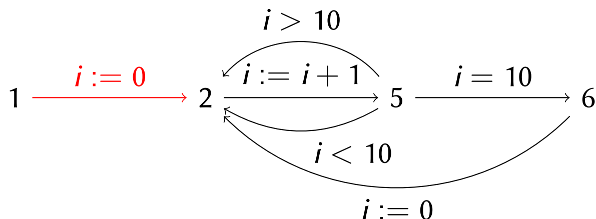


At node 2: $\perp$

Guided static analysis

Idea: replace an invariant computation over the full program by a sequence of invariant computations over partial programs.
Partial program = subset of control-flow graph

```
i = 0;
while(true) {
  i++;
  if (i == 10)
    i=0;
}
```

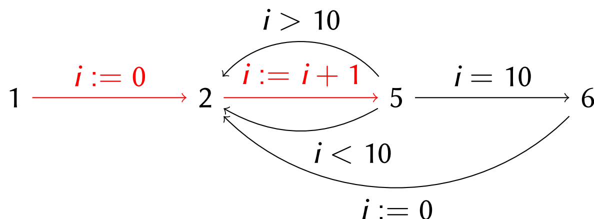


At node 2: $[0, 0]$

Guided static analysis

Idea: replace an invariant computation over the full program by a sequence of invariant computations over partial programs.
Partial program = subset of control-flow graph

```
i = 0;
while(true) {
  i++;
  if (i == 10)
    i=0;
}
```

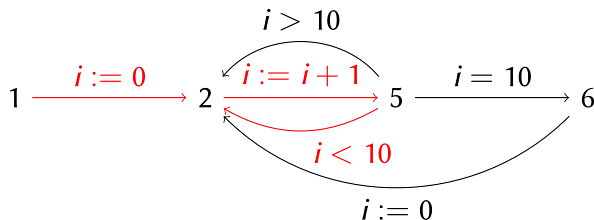


At node 2: $[0, 0]$

Guided static analysis

Idea: replace an invariant computation over the full program by a sequence of invariant computations over partial programs.
Partial program = subset of control-flow graph

```
i = 0;
while(true) {
  i++;
  if (i == 10)
    i=0;
}
```

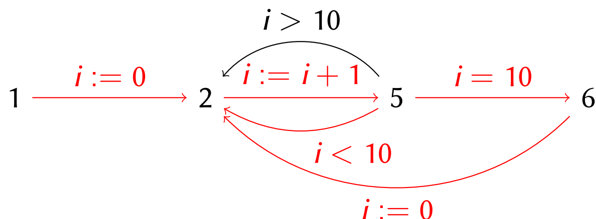


At node 2: $[0, 9]$

Guided static analysis

Idea: replace an invariant computation over the full program by a sequence of invariant computations over partial programs.
Partial program = subset of control-flow graph

```
i = 0;
while(true) {
  i++;
  if (i == 10)
    i=0;
}
```



At node 2: $[0, 9]$

Plan

Introduction

Vanilla: finite lattices
- Mapping to finite state
- Smaller lattices
- Invariant inference algorithm
- Examples from the real world

Coffee: paths and direction

Fudge: widenings

Blackcurrant: accelerated solving

Fiore di latte: conclusion

An intriguing problem

Given a class of programs (with unreachability assertions) and an abstract domain, is the existence of inductive invariants suitable for proving unreachability decidable?

E.g. for template polyhedra, intervals etc. decidable because existence of invariants expressible in a decidable arithmetic theory (real closed fields, Presburger...)

How about general convex polyhedra, for linear programs? (if nonlinear: undecidable)

More generally: relative completeness

Design methods that will not “lose” inductive invariants if they exist in the abstract domain.

E.g. certain analyses on abstractions of functions/maps/arrays can be expressed as syntactic transformation without losing completeness

Conclusion

- devil in the details
- widenings lead to non-monotonicity and brittleness
- rough lattices (intervals...) can regain precision by splitting along paths and/or using forward/backward
- exact methods in some cases