AAA615: Formal Methods

Lecture 6 — Program Analysis

Hakjoo Oh

2017 Fall

Program Verification vs. Program Analysis

Essentially the same things with different trade-offs:

- Program verification
  - Pros: powerful to prove properties
  - Cons: hardly automated
- Program analysis
  - Pros: fully automatic
  - Cons: focus on rather weak properties

Contents

- Symbolic analysis
  - concrete, non-terminating
- Interval analysis
  - abstract, non-relational
- Octagon analysis
  - abstract, relational

Program Representation

Control-flow graph $(\mathbb{C}, \to)$

- $\mathbb{C}$: the set of program points in the program
- $(\to) \subseteq \mathbb{C} \times \mathbb{C}$: the control-flow relation
  - $c \to c'$: c is a predecessor of c'
- Each control-flow edge $c \to c'$ is associated with a command, denoted $\mathbf{cmd}(c \to c')$:

$$cmd \quad \to \quad v := e \mid \text{assume } c \mid cmd_1; cmd_2$$

Weakest Precondition

Weakest precondition transformer

$$\textbf{wp} : \text{FOL} \times \text{stmts} \rightarrow \text{FOL}$$

computes the most general precondition of a given postcondition and program statement:

- $\textbf{wp}(F, \text{assume } c) \iff c \rightarrow F$
- $\textbf{wp}(F[v], v := e) \iff F[e]$
- $\textbf{wp}(F, S_1; \ldots; S_n) \iff \textbf{wp}(\textbf{wp}(F, S_n), S_1; \ldots; S_{n-1})$

Strongest Postcondition

Strongest postcondition transformer

$$\textbf{sp} : \text{FOL} \times \text{stmts} \to \text{FOL}$$

computes the most specific postcondition of a given precondition and program statement:

- $\textbf{sp}(F, \text{assume } c) \iff c \wedge F$
- $\textbf{sp}(F[v], v := e[v]) \iff \exists v^0.\ v = e[v^0] \wedge F[v^0]$
- $\textbf{sp}(F, S_1; \ldots; S_n) \iff \textbf{sp}(\textbf{sp}(F, S_1), S_2; \ldots; S_n)$

Examples

$$\begin{aligned} & \mathsf{sp}(i \geq n, i := i + k) \\ \iff & \exists i^0.\ i = i^0 + k \land i^0 \geq n \\ \iff & i - k \geq n \end{aligned}$$

$$\begin{aligned} & \mathsf{sp}(i \geq n, \text{assume } k \geq 0;\ i := i + k) \\ \iff & \mathsf{sp}(\mathsf{sp}(i \geq n, \text{assume } k \geq 0), i := i + k) \\ \iff & \mathsf{sp}(i \geq n \land k \geq 0, i := i + k) \\ \iff & \exists i^0.\ i = i^0 + k \land i^0 \geq n \land k \geq 0 \\ \iff & i - k \geq n \land k \geq 0 \end{aligned}$$

Inductive Map

- The goal of static analysis is to find a map

$$T : \mathbb{C} \to \mathsf{FOL}$$

that stores inductive invariants for each program point and is implied by the precondition:

$$F_{pre} \implies T(c_0).$$

- If the result $T(c_{exit})$ implies the postcondition

$$T(c_{exit}) \implies F_{post}$$

the function obeys the specification.

Forward Symbolic Analysis Procedure

- Sets of reachable states are represented by formulas.
- Strongest postcondition (**sp**) executes statements over formulas.

$$
\begin{array}{l}
W := \{c_0\} \\
T(c_0) := F_{pre} \\
T(c) := \bot \text{ for } c \in \mathbb{C} \setminus \{c_0\} \\
\text{while } W \neq \emptyset \\
\quad c := \textbf{Choose}(W) \\
\quad W := W \setminus \{c\} \\
\quad \text{foreach } c' \in \textbf{succ}(c) \\
\quad\quad F := \textbf{sp}(T(c), \textbf{cmd}(c \to c')) \\
\quad\quad \text{if } F \not\Rightarrow T(c') \\
\quad\quad\quad T(c') := T(c') \vee F \\
\quad\quad\quad W := W \cup \{c'\} \\
\quad \text{done} \\
\text{done}
\end{array}
$$

Issues

- The implication checking

$$F \not\Rightarrow T(c')$$

  is undecidable in general. The underlying logic must be restricted to a decidable theory or fragment.
- Nontermination of loops.

Example

$$
\begin{array}{l}
@c_0 : i = 0 \wedge n \geq 0; \\
\text{while } @c_1 \\
(i < n) \ \{ \\
\qquad i := i + 1; \\
\} \\
@c_2 : i = n
\end{array}
$$

Initial map:

$$
\begin{aligned}
T(c_0) &\iff i = 0 \wedge n \geq 0 \\
T(c_1) &\iff \perp
\end{aligned}
$$

Following basic path $c_0 \to c_1$:

$$
\begin{aligned}
T(c_0) &\iff i = 0 \wedge n \geq 0 \\
T(c_1) &\iff T(c_1) \vee i = 0 \wedge n \geq 0 \iff i = 0 \wedge n \geq 0
\end{aligned}
$$

Example

Following basic path $c_1 \to c_1$:

1. Symbolic execution:

$$
\begin{aligned}
& \textsf{sp}(T(c_1), \textsf{assume } i < n; i := i + 1) \\
\iff & \textsf{sp}(i = 0 \land n \geq 0, \textsf{assume } i < n; i := i + 1) \\
\iff & \textsf{sp}(i < n \land i = 0 \land n \geq 0, i := i + 1) \\
\iff & \exists i^0.\, i = i^0 + 1 \land i^0 < n \land i^0 = 0 \land n \geq 0 \\
\iff & i = 1 \land n \geq 1
\end{aligned}
$$

2. Checking the implication:

$$i = 1 \land n \geq 1 \;\not\Longrightarrow\; i = 0 \land n \geq 0$$

3. Join the result:

$$T(c_1) \iff (i = 0 \land n \geq 0) \lor (i = 1 \land n \geq 1)$$

Example

At the end of the next iteration:

$$T(c_1) \iff (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee (i = 2 \wedge n \geq 2)$$

and at the end of kth iteration:

$$T(c_1) \iff (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \cdots \vee (i = k \wedge n \geq k)$$

This process does not terminate because

$$(i = k \wedge n \geq k) \not\Rightarrow (i = 0 \wedge n \geq 0) \vee \cdots \vee (i = k-1 \wedge n \geq k-1)$$

for any k. However,

$$0 \leq i \leq n$$

is an obvious inductive invariant that proves the postcondition:

$$0 \leq i \leq n \wedge i \geq n \implies i = n.$$

Addressing the Issues

- Unsound approach, e.g., unrolling loops for a fixed number
  - incapable of verifying properties but still useful for bug-finding
- Sound approach ensures correctness but cannot be complete.
- **Abstract interpretation** is a general method for obtaining sound and computable static analysis.
  - abstract domain
  - abstract semantics
  - widening and narrowing

1. Choose an Abstract Domain

The abstract domain D is a restricted subset of formulas; each member $d \in D$ represents a set of program states: e.g.,

- In the interval abstract domain D_I, a domain element $d \in D_I$ is a conjunction of constraints of the forms

$$c \leq x \qquad \text{and} \qquad x \leq c$$

- In the octagon abstract domain D_O, a domain element $d \in D_I$ is a conjunction of constraints of the forms

$$\pm x_1 \pm x_2 \leq c$$

- In the Karr's abstract domain D_K, a domain element $d \in D_K$ is a conjunction of constraints of the forms

$$c_0 + c_1 x_1 + \cdots c_n x_n = 0$$

2. Construct an Abstraction Function

The abstraction function:

$$\alpha_D : \text{FOL} \to D$$

such that $F \implies \alpha_D(F)$. For example, the assertion

$$F : i = 0 \land n \geq 0$$

can be represented in the interval abstract domain by

$$\alpha_{D_I}(F) : 0 \leq i \land i \leq 0 \land 0 \leq n$$

and in Karr's abstract domain by

$$\alpha_{D_K}(F) : i = 0$$

3. Define an Abstract Strongest Postcondition

Define an abstract strongest postcondition operator $\widehat{\mathbf{sp}}_D$, also known as abstract semantics or transfer function:

$$\widehat{\mathbf{sp}}_D : D \times \text{stmts} \to D$$

such that $\widehat{\mathbf{sp}}_D$ over-approximates $\mathbf{sp}$:

$$\mathbf{sp}(F, S) \implies \widehat{\mathbf{sp}}_D(F, S).$$

3. Define an Abstract Strongest Postcondition

For example, the strongest postcondition for assume:

$$\mathbf{sp}(F, \text{assume } c) \iff c \wedge F$$

is abstracted by

$$\widehat{\mathbf{sp}}(F, \text{assume } c) \iff \alpha_D(c) \sqcap_D F$$

where abstract conjunction $\sqcap_D : D \times D \to D$ is such that

$$F_1 \wedge F_2 \implies F_1 \sqcap_D F_2.$$

When the domain D consists of conjunctions of constraints of some form (e.g. interval domain), $\sqcap_D$ is exact and equals to the usual conjunction $\wedge$:

$$F_1 \wedge F_2 \iff F_1 \sqcap_D F_2.$$

4. Define Abstract Disjunction and Implication Checking

- Define abstract disjunction $\sqcup_D : D \times D \to D$ such that

$$F_1 \vee F_2 \implies F_1 \sqcup_D F_2$$

  Usually abstract disjunction is not exact.

- With a proper abstract domain, the implication checking

$$F \not\Longrightarrow T(c_k)$$

  can be performed by a custom solver without querying a full SMT solver.

5. Define Widening

A widening operator $\bigtriangledown_D$ is a binary operator

$$\bigtriangledown_D : D \times D \to D$$

such that

$$F_1 \vee F_2 \implies F_1 \bigtriangledown_D F_2$$

and the following property holds. For all increasing sequence $F_1, F_2, F_3, \ldots$ (i.e. $F_i \implies F_{i+1}$ for all i), the sequence G_i defined by

$$G_i = \begin{cases} F_1 & \text{if } i = 1 \\ G_{i-1} \bigtriangledown_D F_i & \text{if } i > 1 \end{cases}$$

eventually converges:

$$\text{for some } k \text{ and for all } i \geq k, G_i \iff G_{i+1}.$$

Abstract Interpretation Algorithm

$W := \{c_0\}$
$T(c_0) := \alpha_D(F_{pre})$
$T(c) := \bot$ for $c \in \mathbb{C} \setminus \{c_0\}$
while $W \neq \emptyset$
    $c :=$ **Choose**(W)
    $W := W \setminus \{c\}$
    foreach $c' \in$ **succ**(c)
        $F := \widehat{\mathbf{sp}}(T(c), \mathbf{cmd}(c \to c'))$
        if $F \not\Longrightarrow T(c')$
            if widening is needed
                $T(c') := T(c') \triangledown (T(c') \sqcup_D F)$
            else
                $T(c') := T(c') \sqcup_D F$
            $W := W \cup \{c'\}$
    done
done

Interval Analysis

The interval analysis uses the abstract domain D_I that includes $\bot, \top$ and conjunctions of constraints of the form

$$c \leq v \quad \text{and} \quad v \leq c$$

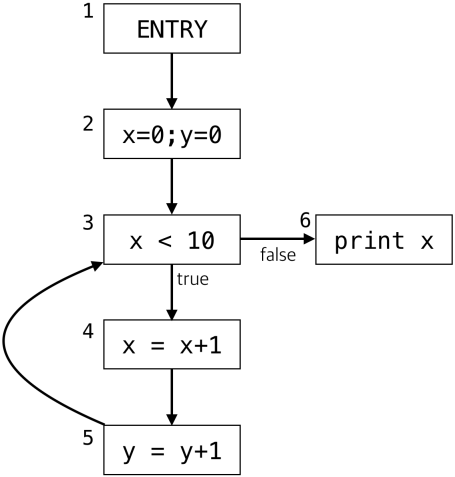
Equivalently, interval analysis computes intervals of program variables:

$$\{\bot\} \cup \{[a,b] \mid a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{+\infty\}, a \leq b\}$$

Consider the simple set of commands:

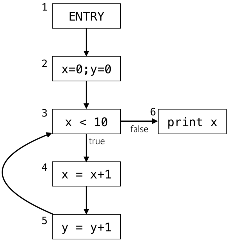
$$\begin{array}{rcl} cmd & \rightarrow & skip \mid x := e \mid x < n \\ e & \rightarrow & n \mid x \mid e + e \mid e - e \mid e * e \mid e/e \end{array}$$

How Interval Analysis Works



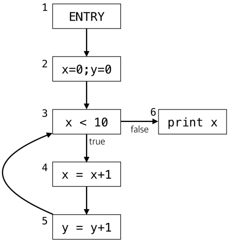
Node	Result
1	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto [0, 0]$ $y \mapsto [0, 0]$
3	$x \mapsto [0, 9]$ $y \mapsto [0, +\infty]$
4	$x \mapsto [1, 10]$ $y \mapsto [0, +\infty]$
5	$x \mapsto [1, 10]$ $y \mapsto [1, +\infty]$
6	$x \mapsto [10, 10]$ $y \mapsto [0, +\infty]$

Forward Propagation



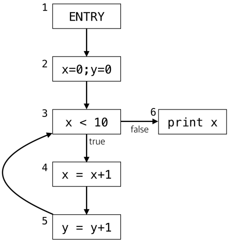
Node	initial	1	2	3	10	11	k	∞
1	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$
3	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,1]$ $y \mapsto [0,1]$	$x \mapsto [0,2]$ $y \mapsto [0,2]$	$x \mapsto [0,9]$ $y \mapsto [0,9]$	$x \mapsto [0,9]$ $y \mapsto [0,10]$	$x \mapsto [0,9]$ $y \mapsto [0,k-1]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$
4	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1,1]$ $y \mapsto [0,0]$	$x \mapsto [1,2]$ $y \mapsto [0,1]$	$x \mapsto [1,3]$ $y \mapsto [0,2]$	$x \mapsto [1,10]$ $y \mapsto [0,9]$	$x \mapsto [1,10]$ $y \mapsto [0,10]$	$x \mapsto [1,10]$ $y \mapsto [0,k-1]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$
5	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1,1]$ $y \mapsto [1,1]$	$x \mapsto [1,2]$ $y \mapsto [1,2]$	$x \mapsto [1,3]$ $y \mapsto [1,3]$	$x \mapsto [1,10]$ $y \mapsto [1,10]$	$x \mapsto [1,10]$ $y \mapsto [1,11]$	$x \mapsto [1,10]$ $y \mapsto [1,k]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$
6	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto [0,0]$	$x \mapsto \bot$ $y \mapsto [0,1]$	$x \mapsto \bot$ $y \mapsto [0,2]$	$x \mapsto [10,10]$ $y \mapsto [0,9]$	$x \mapsto [10,10]$ $y \mapsto [0,10]$	$x \mapsto [10,10]$ $y \mapsto [0,k-1]$	$x \mapsto [10,10]$ $y \mapsto [0,+\infty]$

Forward Propagation Widening



Node	initial	1	2	3
1	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto \bot$
2	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0, 0]$ $y \mapsto [0, 0]$	$x \mapsto [0, 0]$ $y \mapsto [0, 0]$	$x \mapsto [0, 0]$ $y \mapsto [0, 0]$
3	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [0, 0]$ $y \mapsto [0, 0]$	$x \mapsto [0, 9]$ $y \mapsto [0, +\infty]$	$x \mapsto [0, 9]$ $y \mapsto [0, +\infty]$
4	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1, 1]$ $y \mapsto [0, 0]$	$x \mapsto [1, 10]$ $y \mapsto [0, +\infty]$	$x \mapsto [1, 10]$ $y \mapsto [0, +\infty]$
5	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto [1, 1]$ $y \mapsto [1, 1]$	$x \mapsto [1, 10]$ $y \mapsto [1, +\infty]$	$x \mapsto [1, 10]$ $y \mapsto [1, +\infty]$
6	$x \mapsto \bot$ $y \mapsto \bot$	$x \mapsto \bot$ $y \mapsto [0, 0]$	$x \mapsto [10, +\infty]$ $y \mapsto [0, +\infty]$	$x \mapsto [10, +\infty]$ $y \mapsto [0, +\infty]$

Forward Propagation with Narrowing



Node	initial	1	2
1	$x \mapsto \perp$ $y \mapsto \perp$	$x \mapsto \perp$ $y \mapsto \perp$	$x \mapsto \perp$ $y \mapsto \perp$
2	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$	$x \mapsto [0,0]$ $y \mapsto [0,0]$
3	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$	$x \mapsto [0,9]$ $y \mapsto [0,+\infty]$
4	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [0,+\infty]$
5	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$	$x \mapsto [1,10]$ $y \mapsto [1,+\infty]$
6	$x \mapsto [10,+\infty]$ $y \mapsto [0,+\infty]$	$x \mapsto [10,10]$ $y \mapsto [0,+\infty]$	$x \mapsto [10,10]$ $y \mapsto [0,+\infty]$

Interval Domain

- Definition:

$$\mathbb{I} = \{\bot\} \cup \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \ \wedge \ l \leq u\}$$

- An interval is an abstraction of a set of integers:
  - $\gamma([1, 5]) =$
  - $\gamma([3, 3]) =$
  - $\gamma([0, +\infty]) =$
  - $\gamma([-\infty, 7]) =$
  - $\gamma(\bot) =$

Concretization/Abstraction Functions

- $\gamma : \mathbb{I} \to \wp(\mathbb{Z})$ is called *concretization function*:

$$\begin{array}{rcl} \gamma(\bot) & = & \emptyset \\ \gamma([a,b]) & = & \{z \in \mathbb{Z} \mid a \le z \le b\} \end{array}$$

- $\alpha : \wp(\mathbb{Z}) \to \mathbb{I}$ is *abstraction function*:
  - $\alpha(\{2\}) =$
  - $\alpha(\{-1,0,1,2,3\}) =$
  - $\alpha(\{-1,3\}) =$
  - $\alpha(\{1,2,\ldots\}) =$
  - $\alpha(\emptyset) =$
  - $\alpha(\mathbb{Z}) =$

$$\begin{array}{rcl} \alpha(\emptyset) & = & \bot \\ \alpha(S) & = & [\min(S), \max(S)] \end{array}$$

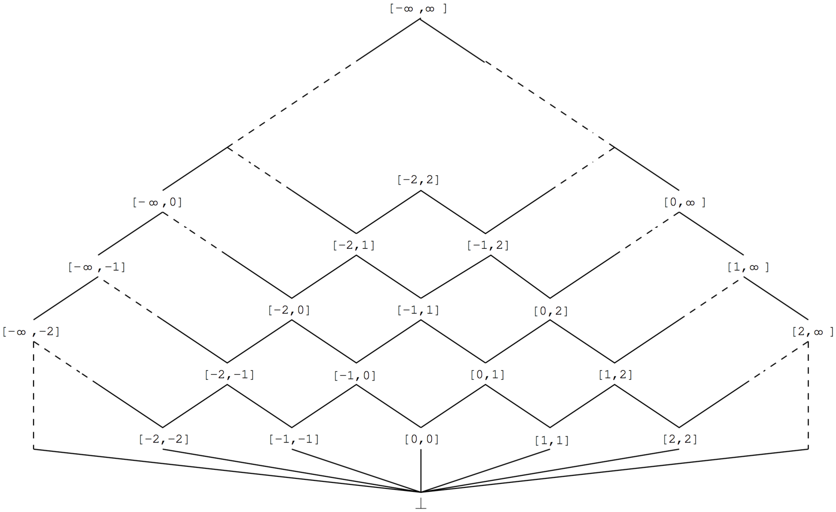
Partial Order ($\sqsubseteq$) $\subseteq \mathbb{I} \times \mathbb{I}$

- $\bot \sqsubseteq i$ for all $i \in \mathbb{I}$
- $i \sqsubseteq [-\infty, +\infty]$ for all $i \in \mathbb{I}$.
- $[1, 3] \sqsubseteq [0, 4]$
- $[1, 3] \not\sqsubseteq [0, 2]$

Definition:

$$i_1 \sqsubseteq i_2 \text{ iff } \begin{cases} i_1 = \bot \vee \\ i_2 = [-\infty, +\infty] \vee \\ (i_1 = [l_1, u_1] \ \wedge \ i_2 = [l_2, u_2] \ \wedge \ l_1 \geq l_2 \ \wedge \ u_1 \leq u_2) \end{cases}$$

Partial Order



Join $\sqcup$ and Meet $\sqcap$ Operators

- The join operator computes the *least upper bound*:
  - $[1,3] \sqcup [2,4] = [1,4]$
  - $[1,3] \sqcup [7,9] = [1,9]$
- The conditions of $i_1 \sqcup i_2$:
  1. $i_1 \sqsubseteq i_1 \sqcup i_2 \ \wedge\ i_2 \sqsubseteq i_1 \sqcup i_2$
  2. $\forall i.\ i_1 \sqsubseteq i \ \wedge\ i_2 \sqsubseteq i \implies i_1 \sqcup i_2 \sqsubseteq i$
- Definition:

$$
\begin{array}{rcl}
\bot \sqcup i & = & i \\
i \sqcup \bot & = & i \\
[l_1, u_1] \sqcup [l_2, u_2] & = & [\min(l_1, l_2), \max(l_1, l_2)]
\end{array}
$$

Join $\sqcup$ and Meet $\sqcap$ Operators

- The meet operator computes the *greatest lower bound*:
  - $[\mathbf{1},\mathbf{3}] \sqcap [\mathbf{2},\mathbf{4}] = [\mathbf{2},\mathbf{3}]$
  - $[\mathbf{1},\mathbf{3}] \sqcap [\mathbf{7},\mathbf{9}] = \bot$
- The conditions of $i_1 \sqcap i_2$:
  1. $i_1 \sqsubseteq i_1 \sqcup i_2 \ \wedge\ i_2 \sqsubseteq i_1 \sqcup i_2$
  2. $\forall i.\ i \sqsubseteq i_1 \ \wedge\ i \sqsubseteq i_2 \implies i \sqsubseteq i_1 \sqcap i_2$
- Definition:

$$\begin{array}{rcl} \bot \sqcap i & = & \bot \\ i \sqcap \bot & = & \bot \\ [l_1, u_1] \sqcap [l_2, u_2] & = & \begin{cases} \bot & \max(l_1, l_2) > \min(l_1, l_2) \\ [\max(l_1, l_2), \min(l_1, l_2)] & \text{o.w.} \end{cases} \end{array}$$

Widening and Narrowing

A simple widening operator for the Interval domain:

$$
\begin{array}{rcll}
[a, b] & \triangledown & \bot & = [a, b] \\
\bot & \triangledown & [c, d] & = [c, d] \\
[a, b] & \triangledown & [c, d] & = [(c < a ? -\infty : a), (b < d ? +\infty : b)]
\end{array}
$$

A simple narrowing operator:

$$
\begin{array}{rcll}
[a, b] & \triangle & \bot & = \bot \\
\bot & \triangle & [c, d] & = \bot \\
[a, b] & \triangle & [c, d] & = [(a = -\infty ? c : a), (b = +\infty ? d : b)]
\end{array}
$$

Abstract States

$$\mathbb{S} = \textbf{Var} \to \mathbb{I}$$

Partial order, join, meet, widening, and narrowing are lifted pointwise:

$$s_1 \sqsubseteq s_2 \text{ iff } \forall x \in \textbf{Var}.\ s_1(x) \sqsubseteq s_2(x)$$

$$s_1 \sqcup s_2 = \lambda x.\ s_1(x) \sqcup s_2(x)$$

$$s_1 \sqcap s_2 = \lambda x.\ s_1(x) \sqcap s_2(x)$$

$$s_1 \bigtriangledown s_2 = \lambda x.\ s_1(x) \bigtriangledown s_2(x)$$

$$s_1 \bigtriangleup s_2 = \lambda x.\ s_1(x) \bigtriangleup s_2(x)$$

The Abstract Domain

$$\mathbb{D} = \mathbb{C} \to \mathbb{S}$$

Partial order, join, meet, widening, and narrowing are lifted pointwise:

$$\boldsymbol{d_1 \sqsubseteq d_2} \text{ iff } \boldsymbol{\forall c \in \mathbb{C}.\ d_1(x) \sqsubseteq d_2(x)}$$

$$\boldsymbol{d_1 \sqcup d_2 = \lambda c.\ d_1(c) \sqcup d_2(c)}$$

$$\boldsymbol{d_1 \sqcap d_2 = \lambda c.\ d_1(c) \sqcap d_2(c)}$$

$$\boldsymbol{d_1 \bigtriangledown d_2 = \lambda c.\ d_1(c) \bigtriangledown d_2(c)}$$

$$\boldsymbol{d_1 \bigtriangleup d_2 = \lambda c.\ d_1(c) \bigtriangleup d_2(c)}$$

Abstract Semantics of Expressions

$$e \rightarrow n \mid x \mid e + e \mid e - e \mid e * e \mid e/e$$

$$\begin{array}{rcl}
eval & : & e \times \mathbb{S} \rightarrow \mathbb{I} \\
eval(n, s) & = & [n, n] \\
eval(x, s) & = & s(x) \\
eval(e_1 + e_2, s) & = & eval(e_1, s) \hat{+} eval(e_2, s) \\
eval(e_1 - e_2, s) & = & eval(e_1, s) \hat{-} eval(e_2, s) \\
eval(e_1 * e_2, s) & = & eval(e_1, s) \hat{*} eval(e_2, s) \\
eval(e_1 / e_2, s) & = & eval(e_1, s) \hat{/} eval(e_2, s)
\end{array}$$

Abstract Binary Operators

$$\begin{array}{rcl} i_1 \mathbin{\hat{+}} i_2 & = & \alpha(\{z_1 + z_2 \mid z_1 \in \gamma(i_1) \ \land \ z_2 \in \gamma(i_2)\}) \\ i_1 \mathbin{\hat{-}} i_2 & = & \alpha(\{z_1 - z_2 \mid z_1 \in \gamma(i_1) \ \land \ z_2 \in \gamma(i_2)\}) \\ i_1 \mathbin{\hat{*}} i_2 & = & \alpha(\{z_1 * z_2 \mid z_1 \in \gamma(i_1) \ \land \ z_2 \in \gamma(i_2)\}) \\ i_1 \mathbin{\hat{/}} i_2 & = & \alpha(\{z_1 / z_2 \mid z_1 \in \gamma(i_1) \ \land \ z_2 \in \gamma(i_2)\}) \end{array}$$

Implementable version:

$$\begin{array}{rcl} \bot \mathbin{\hat{+}} i & = & \\ i \mathbin{\hat{+}} \bot & = & \\ [l_1, u_1] \mathbin{\hat{+}} [l_2, u_2] & = & \\ [l_1, u_1] \mathbin{\hat{-}} [l_2, u_2] & = & \\ [l_1, u_1] \mathbin{\hat{*}} [l_2, u_2] & = & \\ [l_1, u_1] \mathbin{\hat{/}} [l_2, u_2] & = & \end{array}$$

Abstract Execution of Commands

$$f_c : \mathbb{S} \to \mathbb{S}$$

$$f_c(s) = \begin{cases} s & c = skip \\ [x \mapsto eval(e, s)]s & c = x := e \\ [x \mapsto s(x) \sqcap [-\infty, n-1]]s & c = x < n \end{cases}$$

Forward Propagation with Widening

$W := \{c_0\}$
$T(c_0) := \alpha_D(F_{pre})$
$T(c) := \bot$ for $c \in \mathbb{C} \setminus \{c_0\}$
while $W \neq \emptyset$
  $c :=$ **Choose**(W)
  $W := W \setminus \{c\}$
  foreach $c' \in$ **succ**(c)
    $s := f_{\mathsf{cmd}(c \to c')}(T(c))$
    if $s \not\sqsubseteq T(c')$
      if c' is a head of a flow cycle
        $T(c') := T(c') \bigtriangledown (T(c') \sqcup_D s)$
      else
        $T(c') := T(c') \sqcup_D F$
      $W := W \cup \{c'\}$
  done
done

Forward Propagation with Narrowing

$W := \mathbb{C}$
$T :=$ result from widening phase
while $W \neq \emptyset$
  $c :=$ choose(W)
  $W := W \setminus \{c\}$
  foreach $c' \in$ **succ**(c)
    $s := f_{\mathsf{cmd}(c \to c')}(T(c))$
    if $T(c') \not\sqsubseteq s$
      $T(c') := T(c') \bigtriangleup s$
      $W := W \cup \{c'\}$
done

Numerical Abstract Domains

Infer numerical properties of program variables: e.g.,

- division by zero,
- array index out of bounds,
- integer overflow, etc.

Well-known numerical domains:

- interval domain: $\boldsymbol{x \in [l, u]}$
- octagon domain: $\boldsymbol{\pm x \pm y \leq c}$
- polyhedron domain (affine inequalities): $\boldsymbol{a_1 x_1 + \cdots + a_n x_n \leq c}$
- Karr's domain (affine equalities): $\boldsymbol{a_1 x_1 + \cdots + a_n x_n = c}$
- congruence domain: $\boldsymbol{x \in a\mathbb{Z} + b}$

The octagon domain is a restriction of the polyhedron domain where each constraint involves at most two variables and unit coefficients.

Interval vs. Octagon

```
i = 0;
p = 0;

while (i < 12) {
    i = i + 1;
    p = p + 1;
}
assert(i==p)
```

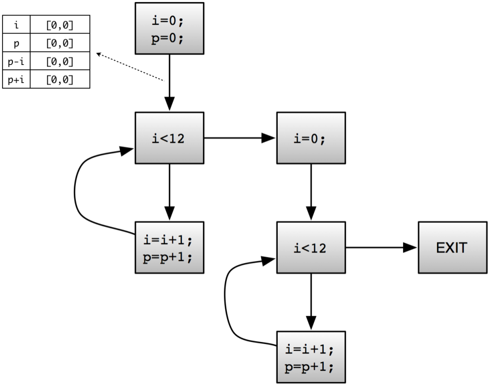
Interval analysis

i	[12,12]
p	[0,+oo]

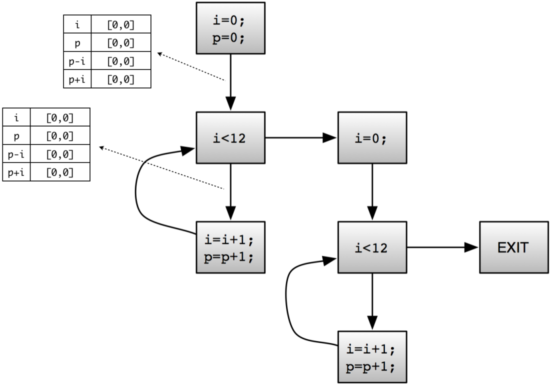
Octagon analysis

i	[12,12]
p	[12,12]
p-i	[0,0]
p+i	[24,24]

Example

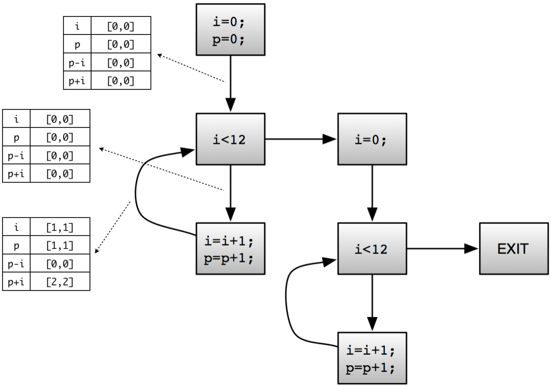


Example



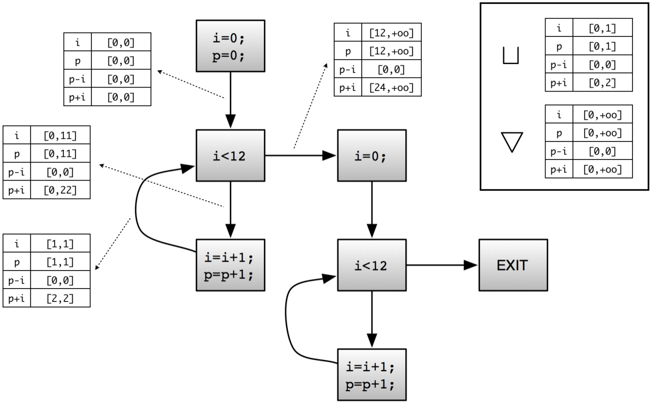
i [0,0]
p [0,0]
p-i [0,0]
p+i [0,0]
i [0,0]
p [0,0]
p-i [0,0]
p+i [0,0]
i=0;
p=0;
i<12
i=0;
i=i+1;
p=p+1;
i<12
EXIT
i=i+1;
p=p+1;

Example

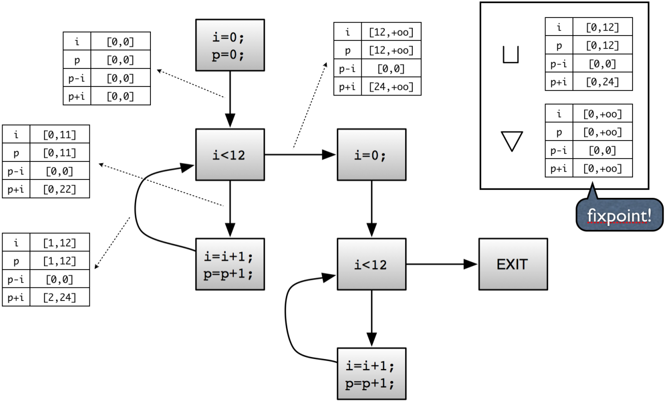


i [0,0]
p [0,0]
p-i [0,0]
p+i [0,0]
i [0,0]
p [0,0]
p-i [0,0]
p+i [0,0]
i [1,1]
p [1,1]
p-i [0,0]
p+i [2,2]
i=0;
p=0;
i<12
i=0;
i=i+1;
p=p+1;
i<12
EXIT
i=i+1;
p=p+1;

Example


i=0;
p=0;
i<12
i=i+1;
p=p+1;
i=0;
i<12
EXIT
i=i+1;
p=p+1;
i [0,0]
p [0,0]
p-i [0,0]
p+i [0,0]
i [0,11]
p [0,11]
p-i [0,0]
p+i [0,22]
i [1,1]
p [1,1]
p-i [0,0]
p+i [2,2]
i [12,+oo]
p [12,+oo]
p-i [0,0]
p+i [24,+oo]
⊔
i [0,1]
p [0,1]
p-i [0,0]
p+i [0,2]
▽
i [0,+oo]
p [0,+oo]
p-i [0,0]
p+i [0,+oo]

Example


i=0;
p=0;
i<12
i=i+1;
p=p+1;
i=0;
i<12
i=i+1;
p=p+1;
EXIT
i [0,0]
p [0,0]
p-i [0,0]
p+i [0,0]
i [0,11]
p [0,11]
p-i [0,0]
p+i [0,22]
i [1,12]
p [1,12]
p-i [0,0]
p+i [2,24]
i [12,+oo]
p [12,+oo]
p-i [0,0]
p+i [24,+oo]
⊔
i [0,12]
p [0,12]
p-i [0,0]
p+i [0,24]
▽
i [0,+oo]
p [0,+oo]
p-i [0,0]
p+i [0,+oo]
fixpoint!

Abstract Domain for Difference Constraints

We consider a restriction of the Octagon domain, which is able to discover invariants of the form

$$x - y \leq c \quad \text{and} \quad \pm x \leq c$$

where x, y are program variables and c is an integer.
Reference:

- Antoine Miné. A New Numerical Abstract Domain Based on Difference-Bound Matrices. PADO 2001.

Difference Constraints

- Let $\mathcal{V} = \{v_1, \ldots, v_n\}$ be the set of program variables and $\mathbb{I}$ be the set of integers.
- We are interested in constraints of the forms

$$v_j - v_i \leq c, \quad v_i \leq c, \quad v_i \geq c$$

- By fixing v_1 to be the constant 0, we can only consider potential/difference constraints of the form

$$v_j - v_i \leq c$$

  since $v_i \leq c$ and $v_i \geq c$ can be rewritten by $v_i - v_1 \leq c$ and $v_1 - v_i \leq -c$, respectively.
- $\mathbb{I}$ is extended to $\overline{\mathbb{I}} = \mathbb{I} \cup \{+\infty\}$.

Difference-Bound Matrices

- A set C of potential constraints over $\mathcal{V}$ can be represented by a $n \times n$ difference-bound matrix:

$$m_{ij} = \begin{cases} c & \text{if } (v_j - v_i \leq c) \in C \\ +\infty & o.w. \end{cases}$$

- A DBM can be represented by a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{A}, w)$, where $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ and $w \in \mathcal{A} \to \mathbb{I}$:

$$\begin{cases} (v_i, v_j) \notin \mathcal{A} & \text{if } m_{ij} = +\infty \\ (v_i, v_j) \in \mathcal{A} \text{ and } w(v_i, v_j) = m_{ij} & \text{if } m_{ij} \neq +\infty \end{cases}$$

- A path $\langle v_{i_1}, \ldots, v_{i_k} \rangle$ in $\mathcal{G}$ is a cycle if $i_1 = i_k$.

Domain of DBMs

- The $\mathcal{V}$-domain, denoted $D(m)$, of a DBM m is the set of points in $\mathbb{I}^n$ that satisfy all constraints in m:

$$D(m) = \{(x_1, \ldots, x_n) \in \mathbb{I}^n \mid \forall i, j.\ x_j - x_i \leq m_{ij}\}.$$

- Because v_1 is fixed to 0, we are interested in $v_2, \ldots, v_n$. The $\mathcal{V}^0$-domain, denoted $D^0(m)$, of a DBM m is defined by

$$D^0(m) = \{(x_2, \ldots, x_n) \in \mathbb{I}^{n-1} \mid (0, x_2, \ldots, x_n) \in D(m)\}.$$

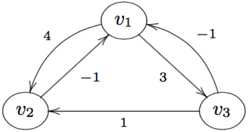
Example

(a)
$$\begin{cases} v_2 \leq 4 \\ -v_2 \leq -1 \\ v_3 \leq 3 \\ -v_3 \leq -1 \\ v_2 - v_3 \leq 1 \end{cases}$$

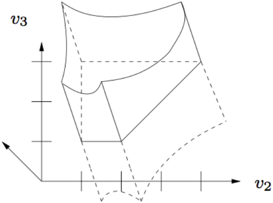
(b)

	v_1	v_2	v_3
v_1	$+\infty$	4	3
v_2	-1	$+\infty$	$+\infty$
v_3	-1	1	$+\infty$

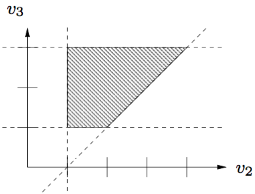
(c)



(d)



(e)



Partial Order

- The order between DBMs is defined as a point-wise extension of $\leq$ on $\bar{\mathbb{I}}$:

$$m \sqsubseteq n \iff \forall i, j.\ m_{ij} \leq n_{ij}.$$

- We have $m \sqsubseteq n \implies D^0(m) \subseteq D^0(n)$ but the converse is not true. For example, two different DBMs can represent the same domain (i.e. $D^0(m) = D^0(n) \not\Longrightarrow m = n$):

(a)

	v_1	v_2	v_3
v_1	$+\infty$	4	3
v_2	-1	$+\infty$	$+\infty$
v_3	-1	1	$+\infty$

(b)

	v_1	v_2	v_3
v_1	**0**	**5**	3
v_2	-1	$+\infty$	$+\infty$
v_3	-1	1	$+\infty$

(c)

	v_1	v_2	v_3
v_1	**0**	4	3
v_2	-1	**0**	$+\infty$
v_3	-1	1	**0**

- However, there is a normal form for any DBM and an algorithm to find it:

$$D^0(m) = D^0(n) \implies m^* = n^*$$

Emptiness Testing

Deciding unsatisfiability of potential constraints:

Theorem

A DBM has an empty $\mathcal{V}^0$-domain iff there exists, in its potential graph, a cycle with a strictly negative total weight.

Checking for cycles with a strictly negative weight can be done by running Bellman-Ford algorithm, which runs in $O(n^3)$.

Closure and Normal Form

Let m be a DBM with a non-empty $\mathcal{V}^0$-domain and $\mathcal{G}$ its potential graph. Since $\mathcal{G}$ has no cycle with a negative weight, we can compute its shortest path closure $\mathcal{G}^*$. The corresponding closed DBM m^* is defined by

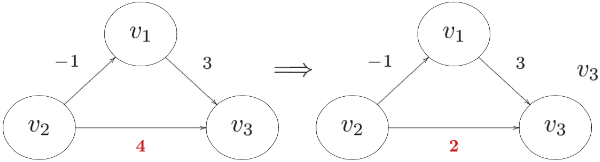
$$m^*_{ii} = 0$$

$$m^*_{ij} = \min_{\substack{\text{all path from } i \text{ to } j \\ \langle i = i_1, i_2, \ldots, i_N = j \rangle}} \sum_{k=1}^{N-1} m_{i_k i_{k+1}} \qquad \text{if } i \neq j$$

which can be computed with any shortest path algorithm (e.g. Floyd-Warshall, $O(n^3)$).

Example:

$$\begin{cases} v_3 - v_1 \leq 3 \\ v_1 - v_2 \leq -1 \end{cases}$$



$v_3 - v_2 \leq 2$

Properties

- $\boldsymbol{D^0(m^*) = D^0(m)}$
- $\boldsymbol{m^* = \min_{\sqsubseteq}\{n \mid D^0(n) = D^0(m)\}}$ (normal form)

Equality and Inclusion Testing

To check equality and inclusion, DMBs must be closed beforehand:

Theorem

If m and n have non-empty $\mathcal{V}^0$-domain,

1. $D^0(m) = D^0(n) \iff m^* = n^*$
2. $D^0(m) \subseteq D^0(n) \iff m^* \sqsubseteq n$

Projection

Given a DBM $\boldsymbol{m}$, we can get the interval value of variable $\boldsymbol{v_k}$ as follows:

Theorem

If $\boldsymbol{m}$ has a non-empty $\boldsymbol{\mathcal{V}^0}$-domain, then $\boldsymbol{\pi|_{v_k}(m) = [-m^*_{k1}, m^*_{1k}]}$.

Intersection and Least Upper Bound

Definition:

$$(\boldsymbol{m} \sqcap \boldsymbol{n})_{ij} = \min(\boldsymbol{m}_{ij}, \boldsymbol{n}_{ij})$$
$$(\boldsymbol{m} \sqcup \boldsymbol{n})_{ij} = \max(\boldsymbol{m}_{ij}, \boldsymbol{n}_{ij})$$

Properties:

- $D^0(\boldsymbol{m} \sqcap \boldsymbol{n}) = D^0(\boldsymbol{m}) \cap D^0(\boldsymbol{n})$ (exact)
- $D^0(\boldsymbol{m} \sqcup \boldsymbol{n}) \supseteq D^0(\boldsymbol{m}) \cup D^0(\boldsymbol{n})$ (exact)
- $\boldsymbol{m}^* \sqcup \boldsymbol{n}^* = \min_{\sqsubseteq}\{\boldsymbol{o} \mid D^0(\boldsymbol{o}) \supseteq D^0(\boldsymbol{m}) \cup D^0(\boldsymbol{n})\}$ (we have to close both arguments before join to get the most precise result)
- If $\boldsymbol{m}$ and $\boldsymbol{n}$ are closed, so is $\boldsymbol{m} \sqcup \boldsymbol{n}$.

Widening

A definition:

$$(m \bigtriangledown n)_{ij} = \begin{cases} m_{ij} & \text{if } n_{ij} \leq m_{ij} \\ +\infty & o.w. \end{cases}$$

Properties:

- $D^0(m \bigtriangledown n) \supseteq D^0(m) \cup D^0(n)$
- Finite chain property: For all m and $(n_i)_i$, the chain $(x_i)_i$

$$\begin{array}{rcl} x_0 & = & m \\ x_{i+1} & = & x_i \bigtriangledown n_i \end{array}$$

  eventually stabilizes.
- To improve precision, we can close m and n_i but not x_i.

Transfer Functions

Example definitions:

- $\left(\llbracket v_k :=?\rrbracket(m)\right)_{ij} = \begin{cases} m_{ij} & \text{if } i \neq k \wedge j \neq k \\ 0 & \text{if } i = j = k \\ \infty & o.w. \end{cases}$
- $\left(\llbracket v_{j_0} - v_{i_0} \leq c\rrbracket(m)\right)_{ij} = \begin{cases} \min(m_{ij}, c) & \text{if } i = i_0 \wedge j = j_0 \\ m_{ij} & o.w. \end{cases}$
- $\llbracket v_{i_0} := v_{j_0} + c\rrbracket(m) = \llbracket v_{j_0} - v_{i_0} \leq -c\rrbracket \circ \llbracket v_{i_0} - v_{j_0} \leq c\rrbracket \circ \llbracket v_{i_0} :=?\rrbracket(m)$ $(i_0 \neq j_0)$
- Otherwise, $\llbracket g\rrbracket(m) = m$ and $\llbracket v_{i_0} := e\rrbracket(m) = \llbracket v_{i_0} :=?\rrbracket(m)$

Program Analysis

Automated techniques for computing program invariants:

- Generic symbolic analysis procedure
- Abstraction examples: Interval and octagon analyses