Abstract Interpretation

Part 2
How to compute such an upper bound?

- If abstract domain $\hat{D}$ is finite (i.e., all chains are finite), we can directly compute

$$\bigcup_{i \in \mathbb{N}} \hat{F}^i(\bot).$$

The computation always terminate.

- Otherwise, we compute a finite chain $\hat{X}_0 \subseteq \hat{X}_1 \subseteq \hat{X}_2 \subseteq \ldots$ such that

$$\bigcup_{i \in \mathbb{N}} \hat{F}^i(\bot) \subseteq \lim_{i \in \mathbb{N}} \hat{X}_i$$
Abstract Domain

\[ \text{gfp } \hat{F} = \left\{ x \in \hat{D} \mid x \supseteq \hat{F}(x) \right\} \]

\[ \text{fix}(\hat{F}) = \left\{ x \in \hat{D} \mid \hat{F}(x) = x \right\} \]

\[ \text{prefp}(\hat{F}) = \left\{ x \in \hat{D} \mid x \subseteq \hat{F}(x) \right\} \]

\[ \text{postfp}(\hat{F}) = \left\{ x \in \hat{D} \mid x \sqsupseteq \hat{F}(x) \right\} \]
Basic Upward/Downward Fixpoint Iteration

\[
\text{postfp}(\hat{F}) = \{ x \in \hat{D} \mid x \sqsupseteq \hat{F}(x) \}
\]

\[
\text{fix}(\hat{F}) = \{ x \in \hat{D} \mid \hat{F}(x) = x \}
\]

\[
\text{prefp}(\hat{F}) = \{ x \in \hat{D} \mid x \sqsubseteq \hat{F}(x) \}
\]
Widening: Overshooting via Extrapolation

\[
lfp \hat{F} = \hat{F}^w(\bot)
\]

\[
\hat{F}(\bot) \sqcup \hat{F}^2(\bot)
\]

\[
\text{prefp}(\hat{F}) = \{ x \in \hat{D} | x \sqsubseteq \hat{F}(x) \}
\]

\[
gfp \hat{F} = \hat{F}^w(\bot)
\]

\[
\text{postfp}(\hat{F}) = \{ x \in \hat{D} | x \sqsupseteq \hat{F}(x) \}
\]
Refining the Widened Result

\( \hat{F} \) is defined as the least fixed point:

\[
\hat{F} = \hat{F}^w(\bot)
\]

The postfp (postfixed point) is:

\[
\text{postfp}(\hat{F}) = \{ x \in \hat{D} \mid x \sqsupseteq \hat{F}(x) \}
\]

The lfp (least fixed point) is:

\[
\text{lfp} \hat{F} = \hat{F}^w(\bot)
\]

The gfp (greatest fixed point) is:

\[
\text{gfp} \hat{F} = \hat{F}^w(\hat{x})
\]

The prefp (prefixed point) is:

\[
\text{prefp}(\hat{F}) = \{ x \in \hat{D} \mid x \sqsubseteq \hat{F}(x) \}
\]
Narrowing

\[
\text{prefp}(\hat{F}) = \{x \in \hat{D} \mid x \sqsubseteq \hat{F}(x)\}
\]

\[
\text{postfp}(\hat{F}) = \{x \in \hat{D} \mid x \sqsupseteq \hat{F}(x)\}
\]
Widening

• We can define a finite chain with an widening operator \( \triangledown \)

\[
\begin{align*}
\hat{X}_0 &= \bot \\
\hat{X}_{i+1} &= \begin{cases} 
\hat{X}_i & \text{if } \hat{F}(\hat{X}_i) \subseteq \hat{X}_i \\
\hat{X}_i \triangledown \hat{F}(\hat{X}_i) & \text{o.w.}
\end{cases}
\end{align*}
\]

• Conditions on \( \triangledown \):

- \( \forall a, b \in D. \ (a \sqsubseteq a \triangledown b) \land (b \sqsubseteq a \triangledown b) \)
- For all increasing chains \( (x_i)_i \), the increasing chain \( (y_i)_i \) defined as

\[
y_i = \begin{cases} 
x_0 & \text{if } i = 0 \\
y_{i-1} \triangledown x_i & \text{if } i > 0
\end{cases}
\]

eventually stabilizes (i.e., the chain is finite).
Widening

• Then

• \( \hat{X}_0 \subseteq \hat{X}_1 \subseteq \cdots \subseteq \hat{X}_n \) is a finite chain.

• Its limit is correct:

\[
\bigcup_{i \in \mathbb{N}} (\hat{F}^i(\perp)) \subseteq \lim_{i \in \mathbb{N}} (\hat{X}_i).
\]

Theorem [widen’s safety]
Theorem (Widening’s Safety)

Let $\hat{D}$ be a CPO, $\hat{F} : \hat{D} \to \hat{D}$ a monotone function, $\triangledown : \hat{D} \times \hat{D} \to \hat{D}$ a widening operator. Then, chain $(\hat{X}_i)_i$ defined as (2) eventually stabilizes and

$$\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\bot}) \subseteq \lim_{i \in \mathbb{N}} \hat{X}_i.$$
Narrowing

- We can refine the widened result $\hat{A} \overset{\text{let}}{=} \lim_{i \in \mathbb{N}} (\hat{X}_i)$ with a narrowing operator $\triangle$.

- Compute chain $\{\hat{Y}_i\}_i$

\[
\hat{Y}_0 = \hat{A} \\
\hat{Y}_{i+1} = \hat{Y}_i \triangle \hat{F}(\hat{Y}_i)
\]
Narrowing

• Conditions
  
  - \( \forall a, b \in \hat{D} : a \sqsupseteq b \Rightarrow a \sqsubseteq (a \triangle b) \sqsupseteq b \)

  - \( \forall \) decreasing chain \( \{a_i\}_i : \) chain \( y_0 = a_0, y_{i+1} = y_i \triangle a_{i+1} \) is finite

• Then

  • \( \{\hat{Y}_i\}_i \) is a finite chain.

  • Its limit is still correct:
    \[
    \bigcup_{i \in \mathbb{N}} (\hat{F}^i (\bot)) \sqsubseteq \lim_{i \in \mathbb{N}} (\hat{Y}_i).
    \]

  Theorem [narrow’s safety]
Theorem (Narrowing’s Safety)

Let $\hat{D}$ be a CPO, $\hat{F} : \hat{D} \rightarrow \hat{D}$ a monotone function, $\triangle : \hat{D} \times \hat{D} \rightarrow \hat{D}$ a narrowing operator. Then, chain $(\hat{Y}_i)_i$ defined as (3) eventually stabilizes and

$$\bigcup_{i \in \mathbb{N}} \hat{F}^i(\bot) \subseteq \lim_{i \in \mathbb{N}} \hat{Y}_i.$$
Example

1: x := 0;
2: y := 0;
3: while (x < 10) {
4:   x := x + 1;
5:   y := y + 1;
6: }
7: print(x)
Example

• A fixpoint $\hat{A} : \mathbb{C} \rightarrow (\text{Var} \rightarrow \hat{\mathbb{Z}})$ should satisfy the followings

• For brevity, let $X_i = \hat{A}(c_i)(x)$

\[
\begin{align*}
X_1 &= [0, 0] \\
X_2 &= [0, 0] \\
X_3 &= (X_2 \sqcup X_5) \sqcap [-\infty, 9] \\
X_4 &= X_3 \hat{+}[1, 1] \\
X_5 &= X_4 \\
X_6 &= X_3 \sqcap [10, +\infty]
\end{align*}
\]
\[ \bigcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\cdot}) : \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1)</td>
<td>(\perp)</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td></td>
</tr>
<tr>
<td>(X_2)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td></td>
</tr>
<tr>
<td>(X_3)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([0,1])</td>
<td>([0,9])</td>
<td></td>
</tr>
<tr>
<td>(X_4)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>([1,1])</td>
<td>([1,1])</td>
<td>([1,1])</td>
<td>([1,10])</td>
<td></td>
</tr>
<tr>
<td>(X_5)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>([1,1])</td>
<td>([1,1])</td>
<td>([1,10])</td>
<td></td>
</tr>
<tr>
<td>(X_6)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>([0,0])</td>
<td>([0,0])</td>
<td>([10,10])</td>
<td></td>
</tr>
</tbody>
</table>
Widening/Narrowing for Interval

- A simple widening operator for the Interval domain:

  \[
  [a, b] \triangledown \bot = [a, b]
  \]
  \[
  \bot \triangledown [c, d] = [c, d]
  \]
  \[
  [a, b] \triangledown [c, d] = [(c < a? - \infty : a), (b < d? + \infty : b)]
  \]

- A simple narrowing operator:

  \[
  [a, b] \triangle \bot = \bot
  \]
  \[
  \bot \triangle [c, d] = \bot
  \]
  \[
  [a, b] \triangle [c, d] = [(a = -\infty ? c : a), (b = +\infty ? d : b)]
  \]
Widening Iterations

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>...</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>⊥</td>
<td>[0,0]</td>
<td></td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>$X_2$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>$X_3$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>$X_4$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td>[1,1]</td>
<td>[1,1]</td>
<td>[1,∞)</td>
<td>[1,∞)</td>
</tr>
<tr>
<td>$X_5$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td>[1,1]</td>
<td>[1,1]</td>
<td>[1,1]</td>
<td>[1,∞)</td>
</tr>
<tr>
<td>$X_6$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[10,∞)</td>
<td>[10,∞)</td>
</tr>
</tbody>
</table>

$X_3' = X_3 \vee ((X_2 \cup X_5) \cap [-\infty, 9])$
$= [0,0] \vee (([0,0] \cup [1,1]) \cap [-\infty, 9])$
$= [0,0] \vee [0,1] = [0,\infty]$
## Narrowing Iterations

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1)</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>(X_2)</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>(X_3)</td>
<td>[0,(+\infty)]</td>
<td>[0,9]</td>
<td>[0,9]</td>
<td>[0,9]</td>
</tr>
<tr>
<td>(X_4)</td>
<td>[1,(+\infty)]</td>
<td>[1,(+\infty)]</td>
<td>[1,10]</td>
<td>[1,10]</td>
</tr>
<tr>
<td>(X_5)</td>
<td>[1,(+\infty)]</td>
<td>[1,(+\infty)]</td>
<td>[1,(+\infty)]</td>
<td>[1,10]</td>
</tr>
<tr>
<td>(X_6)</td>
<td>[10,(+\infty)]</td>
<td>[10,(+\infty)]</td>
<td>[10,10]</td>
<td>[10,10]</td>
</tr>
</tbody>
</table>

\[X_3' = X_3 \triangle ((X_2 \cup X_5) \cap [-\infty, 9]) = [0,\infty] \triangle (([0, 0] \cup [1, \infty]) \cap [-\infty, 9]) = [0,\infty] \triangle [0, 9] = [0, 9]\]
Worklist Algorithm

• The fixpoint algorithm we use — “Tabulation algorithm”

  • at each iteration, abstract memory state at each node (i.e., program point) is repeatedly updated.

• “Worklist algorithm” — a node is visited only if at least one of its predecessors has been updated.

  • in the reverse topological order (i.e., program execution order)
Worklist Algorithm

\[ W \in Worklist = \emptyset(C) \]
\[ T \in C \rightarrow \hat{\mathcal{S}} \]
\[ \hat{f}_c \in \hat{\mathcal{S}} \rightarrow \hat{\mathcal{S}} \]

\[ W := C \]
\[ T := \lambda c. \bot \]

repeat
\[ c := \text{choose}(W) \]
\[ W := W - \{c\} \]
\[ s_{in} := \bigcup_{c' \rightarrow_c} \hat{f}_c'(T(c')) \]
if \( s_{in} \nsubseteq T(c) \)
\[ \text{if } c \text{ is a head of a flow cycle} \]
\[ s_{in} := T(c) \triangledown s_{in} \]
\[ T(c) := s_{in} \]
\[ W := W \cup \{c' \mid c \rightarrow c'\} \]
until \( W = \emptyset \)
Worklist Algorithm

\[ W \in Worklist = \emptyset(\mathbb{C}) \]
\[ T \in \mathbb{C} \rightarrow \hat{\mathbb{S}} \]
\[ \hat{f}_c \in \hat{\mathbb{S}} \rightarrow \hat{\mathbb{S}} \]

\[ W := \mathbb{C} \]
\[ T := \lambda c. \perp \]

repeat
  \[ c := \text{choose}(W) \]
  \[ W := W \setminus \{c\} \]
  \[ s_{in} := \bigsqcup_{c' \rightarrow c} \hat{f}_c'(T(c')) \]
  \[ \text{if } s_{in} \nsubseteq T(c) \]
    \[ \text{if } c \text{ is a head of a flow cycle} \]
      \[ s_{in} := T(c) \triangledown s_{in} \]
      \[ T(c) := s_{in} \]
    \[ W := W \cup \{c' \mid c \rightarrow c'\} \]
  until \[ W = \emptyset \]

† Efficient chaotic iteration strategies with widenings, Francois Bourdoncle
Actually, still inefficient ...

- The values of x and y are propagated to everywhere though they are not used at all at some points (e.g., x at [c2: y := 0])

- A remedy — sparse worklist algorithm
  - H. Oh et al., Design and Implementation of Sparse Global Analyses for C-like Languages. PLDI 2012
  - H. Oh et al., Global Sparse Analysis Framework. TOPLAS 2014
Sparse Analysis

Consider this program and suppose that we analyze the program with the initial abstract state.
Sparse Analysis

Conventional static analysis propagates the entire abstract states along control flows of the program.
Sparse Analysis

So, after the analysis is terminated, each program point is associated with the entire abstract states.
Sparse Analysis

Sparse analysis aims to optimize this conventional static analysis based on two observations.
Sparse Analysis

First, in the analysis of each statement, only a small subset of the state is actually used.

For example, only the value of `x` is necessary to analyze the first statement.
Sparse Analysis

So, in sparse analysis, we keep x here and remove other values.
Sparse Analysis

In this way, at each program point, sparse analysis stores only the values that will be used in the analysis.
Sparse Analysis

The second observation is that the semantic dependencies among statements are usually sparse.

For example, the value of x in the first statement is not used at the next statement but used at the third statement.
Sparse Analysis

So, sparse analysis propagates the value directly to the use point.
Sparse Analysis

Similarly, other values are also propagated along semantic dependencies of the program.
Sparse Analysis

These two "localizations" significantly improves the scalability of the non-sparse analysis.
Sparse Analysis

• can achieve speed-up of orders-of-magnitude while maintaining the same precision (formally proved)

• General for

  • programming languages, e.g., imperative (w/ pointer manipulations), functional, oop, etc.

  • various abstractions e.g., context- / path-sensitivity, etc.

• We’ve used this technique in our static analyzer for C and made it publicly available (https://github.com/ropas/sparrow)

• will be covered in detail at the end of the semester
Something bad happened to $y$

<table>
<thead>
<tr>
<th></th>
<th>Widening</th>
<th>Narrowing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y_1$</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>$Y_2$</td>
<td>⊥</td>
<td>[0,0]</td>
</tr>
<tr>
<td>$Y_3$</td>
<td>⊥</td>
<td>[0,+$\infty$]</td>
</tr>
<tr>
<td>$Y_4$</td>
<td>⊥</td>
<td>[1,+$\infty$]</td>
</tr>
<tr>
<td>$Y_5$</td>
<td>⊥</td>
<td>[1,+$\infty$]</td>
</tr>
<tr>
<td>$Y_6$</td>
<td>⊥</td>
<td>[0,+$\infty$]</td>
</tr>
</tbody>
</table>

0: ENTRY

1: $x := 0$

2: $y := 0$

3: $x < 10$

4: $x := x + 1$

5: $y := y + 1$

6: skip

N

Y
Why this happened?

- The conditional instruction \( x < 10 \) only could narrow down the value range of \( x \)

- since the interval analysis does not know that \( x \) is equal to \( y \) throughout the loop.
Numerical Analysis

• Non-relational analysis
  • captures numerical properties of each of variables
  • e.g., sign, interval, congruence, …

• Relational analysis
  • captures relations between values of variables
  • e.g., octagon, polyhedra, affine equalities, …
Numerical Analysis

- Well-known numerical domains
  - interval: \( x \in [l, u] \)
  - octagon: \( \pm x \pm y \leq c \)
  - polyhedra (affine inequalities): \( a_1 x_1 + \cdots + a_n x_n \leq c \)
  - Karr’s domain (affine equalities): \( a_1 x_1 + \cdots + a_n x_n = c \)
  - congruence: \( x \in a\mathbb{Z} + b \)
Various Abstractions

(a) Set of Points

(b) Interval Abstraction

\[ \begin{cases} x \in [3, 27] \\ y \in [4, 32] \end{cases} \]

(c) Octagon Abstraction

\[ \begin{aligned} 3 &\leq x \leq 7 \\ x + y &\leq 8 \\ 4 &\leq y \leq 5 \\ x - y &\leq 9 \end{aligned} \]

(d) Polyhedral Abstraction

\[ \begin{aligned} 7x + 3y &\leq 5 \\ 2x + 7y &\geq 0 \end{aligned} \]
Demo

- Interproc analyzer

  - http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
The Octagon Abstract Domain


- Why should we care?
  - As of now, the most practical abstract domain for numerical relational analysis
  - interesting aspects: data structure - DBMs, shortest-path algorithm for normalization (reduction)
Example

0: ENTRY
1: x := 0
2: y := 0
3: x < 10
4: x := x + 1
5: y := y + 1
6: skip

Interval Analysis

<table>
<thead>
<tr>
<th>x</th>
<th>[10, 10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>[0, +∞]</td>
</tr>
</tbody>
</table>

Octagon Analysis

<table>
<thead>
<tr>
<th>x</th>
<th>[10, 10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>[10, 10]</td>
</tr>
<tr>
<td>x - y</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>x + y</td>
<td>[20, 20]</td>
</tr>
</tbody>
</table>
Octagon

- A finite set of variables $\mathbf{V} = \{V_1, \ldots, V_n\}$
- An environment $\rho \in (\mathbf{V} \rightarrow \mathbb{I})$ ($\rho \in \mathbb{I}^n$), where $\mathbb{I}$ can be $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$.
- An octagonal constraint is a constraint of the form $\pm V_i \pm V_j \leq c$.
- An octagon is the set of points satisfying a conjunction of octagonal constraints.
A DBM (Difference Bound Matrix) $\mathbf{m}$ is a $n \times n$ square matrix, where $n$ is the number of program variables, with elements in $
abla = \mathbb{I} \cup \{+\infty\}$.

- $m_{ij} = \begin{cases} c & (V_i - V_j \leq c) \in C \\ +\infty & o.w. \end{cases}$
- $\text{DBM} = \nabla^{n \times n}$: the set of all DBMS.
- The potential set described by $\mathbf{m}$:

$$\gamma^{\text{Pot}}(\mathbf{m}) = \{(v_1, \ldots, v_n) \in \mathbb{I}^n \mid \forall i, j. v_j - v_i \leq m_{ij}\}$$
A potential constraint (i.e., difference constraint):

\[
\begin{align*}
V_i - V_j &\leq c \\
V_i &\leq V_j + c \\
V_i - V_j &\geq -c \\
V_i &\geq V_j - c
\end{align*}
\]

Let \( C \) be a set of potential constraints. \( C \) can be represented by a potential graph \( G = (V, A) \).

Assume that, for every \( V_i \rightarrow V_j \), there is at most one arc from \( V_i \) to \( V_j \).

A potential set of \( C \) is the set of points in \( \mathbb{R}^n \) that satisfy \( C \).

Difference graph

\( V_i \leftrightarrow_c V_j \iff (V_j - V_i \leq c) \)
Encoding Octagonal Constraints as Difference Constraints

• Octagonal constraints are of the form $\pm V_i \pm V_j \leq c$. whereas DBM can express constraints of the form $V_i - V_j \leq c$

• $V = \{V_1, \ldots, V_n\}$ : the set of program variables

• Define $\mathcal{V}' = \{V'_1, \ldots, V'_{2n}\}$, where each $V_i \in V$ has both a positive form $V'_{2n-i}$ and negative form $V'_{2n}$

• $V'_{2i-1} = V_i, V'_{2i} = -V_i$
Encoding Octagonal Constraints as Difference Constraints

- A conjunction of octagonal constraints on $V$ can be represented as a conjunction of difference constraints on $V'$

- $2n \times 2n$ DBM with elements in $\bar{I}$

- $\forall i, V'_{2i-1} = -V'_{2i}$ holds for any DBM encoding octagonal constraints.

<table>
<thead>
<tr>
<th>the constraint</th>
<th>is represented as</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_i - V_j \leq c$ ($i \neq j$)</td>
<td>$V'<em>{2i-1} - V'</em>{2j-1} \leq c$ and $V'<em>{2j} - V'</em>{2i} \leq c$</td>
</tr>
<tr>
<td>$V_i + V_j \leq c$ ($i \neq j$)</td>
<td>$V'<em>{2i-1} - V'</em>{2j} \leq c$ and $V'<em>{2j-1} - V'</em>{2i} \leq c$</td>
</tr>
<tr>
<td>$-V_i - V_j \leq c$ ($i \neq j$)</td>
<td>$V'<em>{2i} - V'</em>{2j-1} \leq c$ and $V'<em>{2j} - V'</em>{2i-1} \leq c$</td>
</tr>
<tr>
<td>$V_i \leq c$</td>
<td>$V'<em>{2i-1} - V'</em>{2i} \leq 2c$</td>
</tr>
<tr>
<td>$V_i \geq c$</td>
<td>$V'<em>{2i} - V'</em>{2i-1} \leq -2c$</td>
</tr>
</tbody>
</table>
Concretization

• Given a DBM $m$ of dimension $2n$, the octagon described by $m$ is defined as follows:

$$
\gamma^{Oct} : \text{DBM} \rightarrow \wp(V \rightarrow \mathbb{I})
$$

$$
\gamma^{Oct}(m) = \{ (v_1, \ldots, v_n) \in \mathbb{I}^n \mid (v_1, -v_1, \ldots, v_n, -v_n) \in \gamma^{Pot}(m) \}
$$
Lattice

• The set of DBMs forms a complete lattice

\[(DBM, \sqsubseteq, \sqcup, \sqcap, \bot, \top)\]

• \(\top\) is a DBM such that \(\top_{ij} = +\infty\)

• \(\bot\) is a new smallest element

• \(\forall m, n. m \sqsubseteq n \iff \forall i, j. m_{ij} \leq n_{ij} (m, n \neq \bot)\)

• \(\forall m, n. (m \sqcup n)_{ij} = \max(m_{ij}, n_{ij}) (m, n \neq \bot)\)

• \(\forall m, n. (m \sqcap n)_{ij} = \min(m_{ij}, n_{ij}) (m, n \neq \bot)\)
Normalization

- Different, incomparable DBMs may represent the same potential set:

\[
\begin{array}{c|ccc}
\hline
& 1 & 2 & 3 \\
\hline
1 & +\infty & 4 & 3 \\
2 & -1 & +\infty & +\infty \\
3 & -1 & 1 & +\infty \\
\hline
\end{array}
\]

\[
\begin{array}{c|ccc}
\hline
& 1 & 2 & 3 \\
\hline
1 & 0 & 5 & 3 \\
2 & -1 & +\infty & +\infty \\
3 & -1 & 1 & +\infty \\
\hline
\end{array}
\]

\[
\begin{array}{c|ccc}
\hline
& 1 & 2 & 3 \\
\hline
1 & 0 & 4 & 3 \\
2 & -1 & 0 & +\infty \\
3 & -1 & 1 & 0 \\
\hline
\end{array}
\]
Normalization

\[ \rho : \hat{D} \rightarrow \hat{D} \text{ s.t. } \rho(d) \sqsubseteq d \wedge \gamma(\rho(d)) = \gamma(d) \]

Ideally, \( \rho(d) = \alpha(\gamma(d)) \)
Shortest-Path Closure

- The shortest-path closure $m^*$ of $m$ is defined as follows:

$$
\begin{align*}
  m^*_{ii} &\overset{\text{def}}{=} 0 \\
  m^*_{ij} &\overset{\text{def}}{=} \min_{\text{all path from } i \text{ to } j} \sum_{k=1}^{m-1} m_{ik}m_{k+1} & \text{if } i \neq j \\
\end{align*}
$$

- The closure $m^*$ of $m$ is the smallest DBM representing $\gamma^{Pot}(m)$.

- Floyd-Warshall algorithm is used.

$$
\begin{align*}
  m^0 &\overset{\text{def}}{=} m \\
  m^k_{ij} &\overset{\text{def}}{=} \min(m^k_{ij}, m^k_{ik} + m^k_{kj}) & \text{if } 1 \leq i, j, k \leq n \\
  m^*_ij &\overset{\text{def}}{=} \begin{cases} 
    m^k_{ij} & \text{if } i \neq j \\
    0 & \text{if } i = j 
  \end{cases}
\end{align*}
$$
Shortest-Path Closure

• Computing the closure of a DBM makes implicit constraints explicit:

\[ V_j - V_k \leq c \land V_k - V_l \leq d \implies V_j - V_i \leq c + d \]
Abstract Transfer Functions

• The abstract operators are sound: \( F \circ \gamma \subseteq \gamma \circ \hat{F} \)

• union (join), intersection (meet)

• assignment

• test (guard)

• The shortest-path closure computation should be performed before / after applications of the abstract operators (except intersection).

• For more details, see the paper.
Other Applications of Abstract Interpretation

- program optimization (partial evaluation and program specialization)
- analysis for parallelization of sequential languages
- verification of reactive, real-time and hybrid systems
- semantic watermarking of software (to prevent copyright infringements)
- termination analysis
- time complexity and cost analysis
- ... more in Sec 3.3 in the paper: Patrick Cousot. Abstract Interpretation Based Formal Methods and Future Challenges.
Conclusion

- Abstract interpretation: a systematic method for sound approximations of concrete semantics

- Requirements:
  - Galois connection
  - Soundness of abstract semantic function

- Termination:
  - Widening / Narrowing

- Engineering:
  - Worklist algorithm
  - Sparse worklist algorithm (later)