EC3204: Programming Languages and Compilers

Lecture 13 — Semantic Analysis (4)
Interval Analysis

Sunbeom So
Fall 2024

Fixed Point Computation May Not Terminate

- We compute fixed points to obtain safe approximations.
- Q. Does this computation always terminate?

Fixed Point Computation May Not Terminate

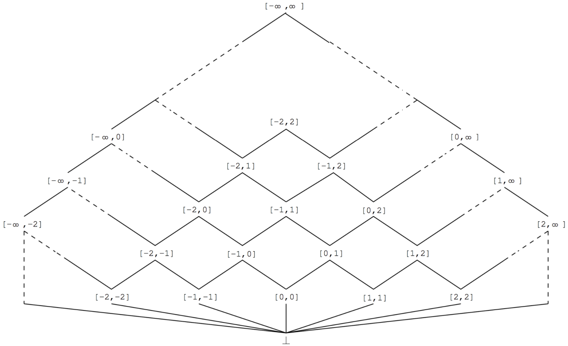
- We compute fixed points to obtain safe approximations.
- Q. Does this computation always terminate?
- A. Yes if the abstract domain (lattice) is finite. Otherwise, it may not.
- Unfortunately, many useful domains have infinite heights. To ensure the termination, we need **widening** operators.

Example: Interval Domain

- The interval domain $\mathbb{I}$ has an infinite height.

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

- Abstract values are expressed by lower and upper bounds: $[l, u]$
  - If the abstract value of x is $[1, 3]$ at some program point p, $1 \leq x \leq 3$ is an invariant at p.



Example: Non-Terminating Fixed Point Computation

Q. What is the resulting abstract state at the loop entry?

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

Example: Non-Terminating Fixed Point Computation

Q. What is the resulting abstract state at the loop entry?

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

A. You cannot obtain it, because computation does not terminate (i.e., we cannot reach a fixed point).

	0	1	2	...	9	10	11	12	...	k
x	$[0,0]$	$[0,1]$	$[0,2]$	...	$[0,9]$	$[0,10]$	$[0,10]$	$[0,10]$	...	$[0,10]$
y	$[0,0]$	$[0,1]$	$[0,2]$	...	$[0,9]$	$[0,10]$	$[0,11]$	$[0,12]$	...	$[0,k]$

Fixed Point Computation with Widening and Narrowing

Two staged fixed point computations:

1. **Widening**: If the abstract domain does not have the finite-height property, we need a widening operator $\triangledown$ to enforce convergence.
2. **Narrowing**: After finding a post-fixed point using widening, we have a second pass using a narrowing operator $\triangle$.

Example: Fixed Point Computation with Widening

Find a post-fixed point at the loop entry using a widening operator.

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

- Initially: $\widehat{m}_0 = \{x \mapsto [0,0], y \mapsto [0,0]\}$
- After the 1st iteration:
  - $\widehat{m}_1 = \{x \mapsto [0,1], y \mapsto [0,1]\} = \widehat{m}_0 \sqcup \{x \mapsto [1,1], y \mapsto [1,1]\}$.
  - Apply widening:
  $$\widehat{m}_1 \leftarrow \widehat{m}_0 \bigtriangledown \widehat{m}_1$$
  where the resulting $\widehat{m}_1 = \{x \mapsto [0,+\infty], y \mapsto [0,+\infty]\}$.
  - Since $\widehat{m}_1 \not\sqsubseteq \widehat{m}_0$ (i.e., not a fixed point), we attempt the 2nd iteration.

Example: Fixed Point Computation with Widening

Find a post-fixed point at the loop entry using a widening operator.

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

- After the 2nd iteration:
  - $\widehat{m}_2 = \{x \mapsto [0, 10], y \mapsto [0, +\infty]\} = \widehat{m}_0 \sqcup \{x \mapsto [1, 10], y \mapsto [1, +\infty]\}$.
  - Apply widening:

    $$\widehat{m}_2 \leftarrow \widehat{m}_1 \bigtriangledown \widehat{m}_2$$

    where the resulting $\widehat{m}_2 = \{x \mapsto [0, +\infty], y \mapsto [0, +\infty]\}$.
  - Since $\widehat{m}_2 \sqsubseteq \widehat{m}_1$ (i.e., a fixed point), we terminate.

Example: Fixed Point Computation with Widening

Find a post-fixed point at the loop entry using a widening operator.

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

	0	1	2
x	$[0, 0]$	$[0, \infty]$	$[0, \infty]$
y	$[0, 0]$	$[0, \infty]$	$[0, \infty]$

Example: Fixed Point Computation with Narrowing

Find a post-fixed point at the loop entry using a widening operator.

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

- Initially: $\widehat{m}'_0 = \{x \mapsto [0, +\infty], y \mapsto [0, +\infty]\}$
- After the 1st iteration:
  - $\widehat{m}_1 = \{x \mapsto [0, 10], y \mapsto [0, +\infty]\} = \widehat{m}_0 \sqcup \{x \mapsto [1, 10], y \mapsto [1, +\infty]\}$.
  - Apply narrowing:

    $$\widehat{m}_1 \leftarrow \widehat{m}'_0 \bigtriangleup \widehat{m}_1$$

    where the resulting $\widehat{m}_1 = \{x \mapsto [0, 10], y \mapsto [0, +\infty]\}$.
  - Since $\widehat{m}'_0 \not\sqsubseteq \widehat{m}_1$ (i.e., not a fixed point), we attempt the 2nd iteration.

Example: Fixed Point Computation with Narrowing

Find a post-fixed point at the loop entry using a widening operator.

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

- After the 2nd iteration:
  - $\widehat{m_2} = \{x \mapsto [0, 10], y \mapsto [0, +\infty]\} = \widehat{m_0} \sqcup \{x \mapsto [1, 10], y \mapsto [1, +\infty]\}$.
  - Apply narrowing:
    $$\widehat{m_2} \leftarrow \widehat{m_2} \bigtriangleup \widehat{m_1}$$
    where the resulting $\widehat{m_2} = \{x \mapsto [0, 10], y \mapsto [0, +\infty]\}$.
  - Since $\widehat{m_1} \sqsubseteq \widehat{m_2}$ (i.e., a fixed point), we terminate.

Example: Fixed Point Computation with Narrowing

Find a post-fixed point at the loop entry using a widening operator.

```
x = 0;
y = 0;
while (x < 10) {
  x = x+1;
  y = y+1;
}
```

- With widening:

	0	1	2
$\boldsymbol{x}$	$[0, 0]$	$[0, \infty]$	$[0, \infty]$
$\boldsymbol{y}$	$[0, 0]$	$[0, \infty]$	$[0, \infty]$

- With narrowing:

	0	1	2
$\boldsymbol{x}$	$[0, \infty]$	$[0, 10] (= [0, \infty] \triangle [0, 10])$	$[0, 10]$
$\boldsymbol{y}$	$[0, \infty]$	$[0, \infty] (= [0, \infty] \triangle [0, \infty])$	$[0, \infty]$

Step 1. Interval Domain

Plan: formally define the widening/narrowing operators for the interval domain.

The interval domain is a pair of $(\mathbb{I}, \sqsubseteq)$.

- $\mathbb{I} = \{\bot\} \cup \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \land l \leq u\}$
- How to define $\sqsubseteq$?
  - $\bot \sqsubseteq i$ for all $i \in \mathbb{I}$
  - $i \sqsubseteq [-\infty, \infty]$ for all $i \in \mathbb{I}$
  - $[1, 3] \sqsubseteq [0, 4]$
  - $[1, 3] \not\sqsubseteq [0, 2]$

Step 1. Interval Domain

Plan: formally define the widening/narrowing operators for the interval domain.

The interval domain is a pair of $(\mathbb{I}, \sqsubseteq)$.

- $\mathbb{I} = \{\bot\} \cup \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \land l \leq u\}$
- How to define $\sqsubseteq$?
  - $\bot \sqsubseteq i$ for all $i \in \mathbb{I}$
  - $i \sqsubseteq [-\infty, \infty]$ for all $i \in \mathbb{I}$
  - $[1, 3] \sqsubseteq [0, 4]$
  - $[1, 3] \not\sqsubseteq [0, 2]$

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

Step 2. Abstract Semantics

Abstract semantics for the arithmetic expressions:

$$
\begin{array}{rcl}
\widehat{\mathcal{A}}[\![\ a\ ]\!] & : & \widehat{\textsf{State}} \to \mathbb{I} \\
\widehat{\mathcal{A}}[\![\ n\ ]\!](\hat{s}) & = & \alpha(\{n\}) \\
\widehat{\mathcal{A}}[\![\ x\ ]\!](\hat{s}) & = & \hat{s}(x) \\
\widehat{\mathcal{A}}[\![\ a_1 + a_2\ ]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![\ a_1\ ]\!](\hat{s}) \mathbin{\hat{+}} \widehat{\mathcal{A}}[\![\ a_2\ ]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![\ a_1 \star a_2\ ]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![\ a_1\ ]\!](\hat{s}) \mathbin{\hat{\star}} \widehat{\mathcal{A}}[\![\ a_2\ ]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![\ a_1 - a_2\ ]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![\ a_1\ ]\!](\hat{s}) \mathbin{\hat{-}} \widehat{\mathcal{A}}[\![\ a_2\ ]\!](\hat{s})
\end{array}
$$

where $\alpha : \mathbb{Z} \to \mathbb{I}$ abstracts the integer constants:

$$\alpha(z) = [z, z]$$

Step 2. Abstract Semantics

Abstract arithmetic operators:

$$
\begin{aligned}
\bot \mathbin{\widehat{+}} i &= \\
i \mathbin{\widehat{+}} \bot &= \\
[l_1, u_1] \mathbin{\widehat{+}} [l_2, u_2] &= \\
\\
\bot \mathbin{\widehat{-}} i &= \\
i \mathbin{\widehat{-}} \bot &= \\
[l_1, u_1] \mathbin{\widehat{-}} [l_2, u_2] &= \\
\\
\bot \mathbin{\widehat{\star}} i &= \\
i \mathbin{\widehat{\star}} \bot &= \\
[l_1, u_1] \mathbin{\widehat{\star}} [l_2, u_2] &=
\end{aligned}
$$

Step 2. Abstract Semantics

Abstract arithmetic operators:

$$
\begin{array}{rcl}
\bot \mathbin{\widehat{+}} i & = & \bot \\
i \mathbin{\widehat{+}} \bot & = & \bot \\
[l_1, u_1] \mathbin{\widehat{+}} [l_2, u_2] & = & [l_1 + l_2, u_1 + u_2] \\
\\
\bot \mathbin{\widehat{-}} i & = & \bot \\
i \mathbin{\widehat{-}} \bot & = & \bot \\
[l_1, u_1] \mathbin{\widehat{-}} [l_2, u_2] & = & [l_1 - u_2, u_1 - l_2] \\
\\
\bot \mathbin{\widehat{\star}} i & = & \bot \\
i \mathbin{\widehat{\star}} \bot & = & \bot \\
[l_1, u_1] \mathbin{\widehat{\star}} [l_2, u_2] & = & [\min(l_1 \star l_2, l_1 \star u_2, u_1 \star l_2, u_1 \star u_2), \\
 & & \max(l_1 \star l_2, l_1 \star u_2, u_1 \star l_2, u_1 \star u_2)]
\end{array}
$$

Step 2. Abstract Semantics

Abstract semantics for the boolean expressions:

$$
\begin{array}{rcl}
\widehat{\mathcal{B}}[\![\ b\ ]\!] & : & \widehat{\textsf{State}} \rightarrow \widehat{\textsf{T}} \\
\widehat{\mathcal{B}}[\![\ \texttt{true}\ ]\!](\hat{s}) & = & \widehat{\mathit{true}} \\
\widehat{\mathcal{B}}[\![\ \texttt{false}\ ]\!](\hat{s}) & = & \widehat{\mathit{false}} \\
\widehat{\mathcal{B}}[\![\ a_1 = a_2\ ]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![\ a_1\ ]\!](\hat{s}) \mathrel{\widehat{=}} \widehat{\mathcal{A}}[\![\ a_2\ ]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\ a_1 \leq a_2\ ]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![\ a_1\ ]\!](\hat{s}) \mathrel{\widehat{\leq}} \widehat{\mathcal{A}}[\![\ a_2\ ]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\ \neg b\ ]\!](\hat{s}) & = & \widehat{\neg}\widehat{\mathcal{B}}[\![\ b\ ]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\ b_1 \land b_2\ ]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![\ b_1\ ]\!](\hat{s}) \mathrel{\widehat{\land}} \widehat{\mathcal{B}}[\![\ b_2\ ]\!](\hat{s})
\end{array}
$$

Step 2: Abstract Semantics (Cont'd)

$$
\begin{array}{rcl}
\widehat{\mathcal{C}}[\![c]\!] & : & \widehat{\textbf{State}} \to \widehat{\textbf{State}} \\
\widehat{\mathcal{C}}[\![x := a]\!] & = & \lambda \widehat{s}.\widehat{s}[x \mapsto \widehat{\mathcal{A}}[\![a]\!](\widehat{s})] \\
\widehat{\mathcal{C}}[\![\texttt{skip}]\!] & = & \textbf{id} \\
\widehat{\mathcal{C}}[\![c_1; c_2]\!] & = & \widehat{\mathcal{C}}[\![c_2]\!] \circ \widehat{\mathcal{C}}[\![c_1]\!] \\
\widehat{\mathcal{C}}[\![\texttt{if}\ b\ c_1\ c_2]\!] & = & \begin{cases} \bot & \cdots \widehat{\mathcal{B}}[\![b]\!](\widehat{s}) = \bot \\ \widehat{\mathcal{C}}[\![c_1]\!](\widehat{s}) & \cdots \widehat{\mathcal{B}}[\![b]\!](\widehat{s}) = \widehat{true} \\ \widehat{\mathcal{C}}[\![c_2]\!](\widehat{s}) & \cdots \widehat{\mathcal{B}}[\![b]\!](\widehat{s}) = \widehat{false} \\ \widehat{\mathcal{C}}[\![c_1]\!](\textbf{filter}(b)(\widehat{s})) & \cdots \widehat{\mathcal{B}}[\![b]\!](\widehat{s}) = \top \\ \sqcup\ \widehat{\mathcal{C}}[\![c_2]\!](\textbf{filter}(\neg b)(\widehat{s})) & \end{cases} \\
\widehat{\mathcal{C}}[\![\texttt{while}\ b\ c]\!] & = & \lambda \widehat{s}.\textbf{filter}(\neg b)(\mathit{fix}(\lambda \widehat{x}.\widehat{s} \sqcup \widehat{\mathcal{C}}[\![c]\!](\textbf{filter}(b)(\widehat{x}))))
\end{array}
$$

Step 2: Abstract Semantics

filter$(p)(\hat{s})$ returns the abstract state $\hat{s}'$ that can make p *true*. Let $\hat{s}(x) = [l, u]$.

$$\textbf{filter}(x < n)(\hat{s}) = \begin{cases} \lambda y \in \mathbb{X}.\bot & \text{if } l \geq n \\ \hat{s}[x \mapsto [l, n-1]] & \text{if } l < n \leq u \\ \hat{s} & \text{if } u < n \end{cases}$$

$$\textbf{filter}(x \leq n)(\hat{s}) = \begin{cases} \lambda y \in \mathbb{X}.\bot & \text{if } l > n \\ \hat{s}[x \mapsto [l, n]] & \text{if } l \leq n < u \\ \hat{s} & \text{if } u \leq n \end{cases}$$

Other cases can be defined in similar ways.

Widening and Narrowing

During analyzing `while`-loop, replace $\bigsqcup$ with $\triangledown$ and $\triangle$ in sequence (possibly after some iterations).

A simple widening operator:

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

A simple narrowing operator:

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

Summary

- Fixed point computations may not terminate.
- Widening ensures convergence and narrowing helps to regain precision.