EC4219: Software Engineering

Lecture 17 — Abstract Interpretation (2)

Sunbeom So
2024 Spring

Fixed Point Computation May Not Terminate

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

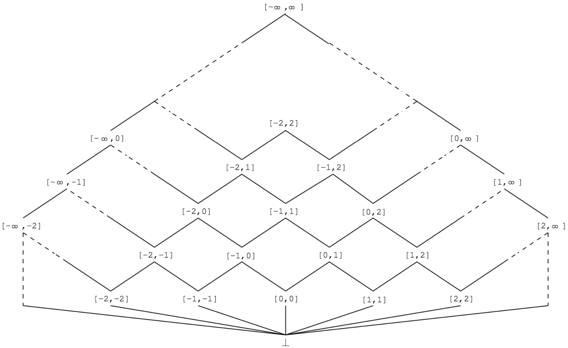
Fixed Point Computation May Not Terminate

- We compute fixed points to obtain sound over-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.

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

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

- 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 entry (head) of the loop?

```
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 entry (head) of the loop?

```
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 force 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 entry of the loop 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

Refine the post-fixed point at the entry of the loop using narrowing.

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

Step 1. Interval Domain

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

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

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

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

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

Concretization/Abstraction Functions

- $\gamma : \mathbb{I} \rightarrow \mathcal{P}(\mathbb{Z})$ is a concretization function.
  - $\gamma([1, 5]) =$
  - $\gamma([3, 3]) =$
  - $\gamma([-\infty, 7])$

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

- $\alpha : \mathcal{P}(\mathbb{Z}) \rightarrow \mathbb{I}$ is an abstraction function.
  - $\alpha(\{2\}) =$
  - $\alpha(\{-1, 0, 1, 2, 3\}) =$
  - $\alpha(\{-1, 3\}) =$
  - $\alpha(\{1, 2, \cdots\}) =$
  - $\alpha(\emptyset) =$
  - $\alpha(\mathbb{Z}) =$

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

Step 2. Abstract Semantics

Recall our imperative language for the sign analysis:

$$\begin{array}{lcl} a & \to & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\ b & \to & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ c & \to & x := a \mid \texttt{skip} \mid c_1; c_2 \mid \texttt{if}\ b\ c_1\ c_2 \mid \texttt{while}\ b\ c \end{array}$$

Abstract semantics for the arithmetic expressions:

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

Step 2. Abstract Semantics (Cont'd)

Abstract arithmetic operators:

- $\bot \mathbin{\hat{+}} i =$
- $i \mathbin{\hat{+}} \bot =$
- $[l_1, u_1] \mathbin{\hat{+}} [l_2, u_2] =$
- $[l_1, u_1] \mathbin{\hat{-}} [l_2, u_2] =$
- $[l_1, u_1] \mathbin{\hat{\star}} [l_2, u_2] =$

Step 2. Abstract Semantics (Cont'd)

Abstract semantics for the boolean expressions:

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

Step 2: Abstract Semantics (Cont'd)

$$
\begin{array}{rcl}
\hat{\mathcal{C}}[\![c]\!] & : & \widehat{\textbf{State}} \to \widehat{\textbf{State}} \\
\hat{\mathcal{C}}[\![x := a]\!] & = & \lambda \hat{s}.\hat{s}[x \mapsto \hat{\mathcal{A}}[\![a]\!](\hat{s})] \\
\hat{\mathcal{C}}[\![\texttt{skip}]\!] & = & \textbf{id} \\
\hat{\mathcal{C}}[\![c_1; c_2]\!] & = & \hat{\mathcal{C}}[\![c_2]\!] \circ \hat{\mathcal{C}}[\![c_1]\!] \\
\hat{\mathcal{C}}[\![\texttt{if}\ b\ c_1\ c_2]\!] & = & \widehat{\textbf{cond}}(\hat{\mathcal{B}}[\![b]\!], \hat{\mathcal{C}}[\![c_1]\!], \hat{\mathcal{C}}[\![c_2]\!]) \\
\hat{\mathcal{C}}[\![\texttt{while}\ b\ c]\!] & = & \mathit{fix} \hat{F} \\
 & & \text{where } \hat{F}(g) = \widehat{\textbf{cond}}(\hat{\mathcal{B}}[\![b]\!], g \circ \hat{\mathcal{C}}[\![c]\!], \textbf{id})
\end{array}
$$

$$
\widehat{\textbf{cond}}(f, g, h)(\hat{s}) = \begin{cases} \bot & \cdots f(\hat{s}) = \bot \\ g(\hat{s}') & \cdots f(\hat{s}) = \widehat{\mathit{true}} \\ h(\hat{s}') & \cdots f(\hat{s}) = \widehat{\mathit{false}} \\ g(\hat{s}') \sqcup h(\hat{s}') & \cdots f(\hat{s}) = \top \end{cases}
$$

$$
\text{where } \hat{s}' = \bigsqcup \{\hat{s}'' | \hat{s}'' \sqsubseteq \hat{s}', \hat{s}'' \models p\}
$$

Widening and Narrowing

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

A simple widening operator:

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

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 recover precision.