

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

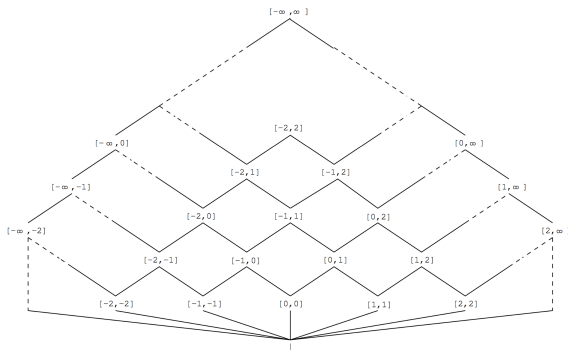
- We compute fixed points to obtain sound over-approximations.
- Q. Does this computation always terminate?
- A. Yes if the abstract domain (lattice) is finite. Otherwise, it may not.
- Unfortunately, many useful domains have infinite heights. To ensure the termination, we need **widening** operators.

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 entry (head) of the loop?

```
1 x = 0;  
2 y = 0;  
3 while (x < 10) {  
4     x = x+1;  
5     y = y+1;  
6 }
```

Example: Non-Terminating Fixed Point Computation

Q. What is the resulting abstract state at the entry (head) of the loop?

```
1  x = 0;
2  y = 0;
3  while (x < 10) {
4      x = x+1;
5      y = y+1;
6  }
```

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 ∇ to force convergence.
- ② **Narrowing:** After finding a post-fixed point (using widening), we have a second pass using a narrowing operator Δ .

Example: Fixed Point Computation with Widening

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

```
1 x = 0;
2 y = 0;
3 while (x < 10) {
4     x = x+1;
5     y = y+1;
6 }
```

	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.

```
1 x = 0;  
2 y = 0;  
3 while (x < 10) {  
4     x = x+1;  
5     y = y+1;  
6 }
```

- With widening:

	0	1	2
<i>x</i>	$[0, 0]$	$[0, \infty]$	$[0, \infty]$
<i>y</i>	$[0, 0]$	$[0, \infty]$	$[0, \infty]$

- With narrowing:

	0	1	2
<i>x</i>	$[0, \infty]$	$[0, 10](= [0, \infty] \triangle [0, 10])$	$[0, 10]$
<i>y</i>	$[0, \infty]$	$[0, \infty](= [0, \infty] \triangle [0, \infty])$	$[0, \infty]$

Step 1. Interval Domain

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

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

- $\mathbb{I} = \{\perp\} \cup \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \wedge l \leq u\}$
- How to define \sqsubseteq ?
 - ▶ $\perp \sqsubseteq i$ 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 $(\mathbb{I}, \sqsubseteq)$.

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

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

Concretization/Abstraction Functions

- $\gamma : \mathbb{I} \rightarrow \mathcal{P}(\mathbb{Z})$ is a concretization function.

- ▶ $\gamma([1, 5]) =$
- ▶ $\gamma([3, 3]) =$
- ▶ $\gamma([-\infty, 7])$

$$\begin{aligned}\gamma(\perp) &= \emptyset \\ \gamma([a, b]) &= \{z \in \mathbb{Z} \mid a \leq z \leq b\}\end{aligned}$$

- $\alpha : \mathcal{P}(\mathbb{Z}) \rightarrow \mathbb{I}$ is an abstraction function.

- ▶ $\alpha(\{2\}) =$
- ▶ $\alpha(\{-1, 0, 1, 2, 3\}) =$
- ▶ $\alpha(\{-1, 3\}) =$
- ▶ $\alpha(\{1, 2, \dots\}) =$
- ▶ $\alpha(\emptyset) =$
- ▶ $\alpha(\mathbb{Z}) =$

$$\begin{aligned}\alpha(\emptyset) &= \perp \\ \alpha(S) &= [\min(S), \max(S)]\end{aligned}$$

Step 2. Abstract Semantics

Recall our imperative language for the sign analysis:

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

Abstract semantics for the arithmetic expressions:

$$\begin{aligned}\widehat{\mathcal{A}}[a] &: \widehat{\text{State}} \rightarrow \mathbb{I} \\ \widehat{\mathcal{A}}[n](\hat{s}) &= \alpha(\{n\}) \\ \widehat{\mathcal{A}}[x](\hat{s}) &= \hat{s}(x) \\ \widehat{\mathcal{A}}[a_1 + a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) \hat{+} \widehat{\mathcal{A}}[a_2](\hat{s}) \\ \widehat{\mathcal{A}}[a_1 \star a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) \hat{\star} \widehat{\mathcal{A}}[a_2](\hat{s}) \\ \widehat{\mathcal{A}}[a_1 - a_2](\hat{s}) &= \widehat{\mathcal{A}}[a_1](\hat{s}) \hat{-} \widehat{\mathcal{A}}[a_2](\hat{s})\end{aligned}$$

Step 2. Abstract Semantics (Cont'd)

Abstract arithmetic operators:

$$\perp \hat{+} i =$$

$$i \hat{+} \perp =$$

$$[l_1, u_1] \hat{+} [l_2, u_2] =$$

$$\perp \hat{-} i =$$

$$i \hat{-} \perp =$$

$$[l_1, u_1] \hat{-} [l_2, u_2] =$$

$$\perp \hat{\star} i =$$

$$i \hat{\star} \perp =$$

$$[l_1, u_1] \hat{\star} [l_2, u_2] =$$

Step 2. Abstract Semantics (Cont'd)

Abstract arithmetic operators:

$$\perp \hat{+} i = \perp$$

$$i \hat{+} \perp = \perp$$

$$[l_1, u_1] \hat{+} [l_2, u_2] = [l_1 + l_2, u_1 + u_2]$$

$$\perp \hat{-} i = \perp$$

$$i \hat{-} \perp = \perp$$

$$[l_1, u_1] \hat{-} [l_2, u_2] = [l_1 - u_2, u_1 - l_2]$$

$$\perp \hat{\star} i = \perp$$

$$i \hat{\star} \perp = \perp$$

$$[l_1, u_1] \hat{\star} [l_2, u_2] = [\min(l_1 \star l_2, l_1 \star u_2, u_1 \star l_2, u_1 \star u_2), \max(l_1 \star l_2, l_1 \star u_2, u_1 \star l_2, u_1 \star u_2)]$$

Step 2. Abstract Semantics (Cont'd)

Abstract semantics for the boolean expressions:

$$\widehat{\mathcal{B}} \llbracket b \rrbracket : \widehat{\text{State}} \rightarrow \widehat{\mathbf{T}}$$

$$\widehat{\mathcal{B}} \llbracket \text{true} \rrbracket(\hat{s}) = \widehat{\text{true}}$$

$$\widehat{\mathcal{B}} \llbracket \text{false} \rrbracket(\hat{s}) = \widehat{\text{false}}$$

$$\widehat{\mathcal{B}} \llbracket a_1 = a_2 \rrbracket(\hat{s}) = \widehat{\mathcal{A}} \llbracket a_1 \rrbracket(\hat{s}) \hat{=} \text{Sign} \widehat{\mathcal{A}} \llbracket a_2 \rrbracket(\hat{s})$$

$$\widehat{\mathcal{B}} \llbracket a_1 \leq a_2 \rrbracket(\hat{s}) = \widehat{\mathcal{A}} \llbracket a_1 \rrbracket(\hat{s}) \hat{\leq} \text{Sign} \widehat{\mathcal{A}} \llbracket a_2 \rrbracket(\hat{s})$$

$$\widehat{\mathcal{B}} \llbracket \neg b \rrbracket(\hat{s}) = \neg \widehat{\mathcal{B}} \llbracket b \rrbracket(\hat{s})$$

$$\widehat{\mathcal{B}} \llbracket b_1 \wedge b_2 \rrbracket(\hat{s}) = \widehat{\mathcal{B}} \llbracket b_1 \rrbracket(\hat{s}) \hat{\wedge} \widehat{\mathcal{B}} \llbracket b_2 \rrbracket(\hat{s})$$

Step 2: Abstract Semantics (Cont'd)

$$\widehat{\mathcal{C}}[c] : \widehat{\text{State}} \rightarrow \widehat{\text{State}}$$

$$\widehat{\mathcal{C}}[x := a] = \lambda \hat{s}. \hat{s}[x \mapsto \widehat{\mathcal{A}}[a](\hat{s})]$$

$$\widehat{\mathcal{C}}[\text{skip}] = \text{id}$$

$$\widehat{\mathcal{C}}[c_1; c_2] = \widehat{\mathcal{C}}[c_2] \circ \widehat{\mathcal{C}}[c_1]$$

$$\widehat{\mathcal{C}}[\text{if } b \text{ } c_1 \text{ } c_2] = \begin{cases} \perp & \dots \widehat{\mathcal{B}}[b](\hat{s}) = \perp \\ \widehat{\mathcal{C}}[c_1](\hat{s}) & \dots \widehat{\mathcal{B}}[b](\hat{s}) = \widehat{\text{true}} \\ \widehat{\mathcal{C}}[c_2](\hat{s}) & \dots \widehat{\mathcal{B}}[b](\hat{s}) = \widehat{\text{false}} \\ \widehat{\mathcal{C}}[c_1](\text{filter}(b)(\hat{s})) & \dots \widehat{\mathcal{B}}[b](\hat{s}) = \top \\ \sqcup \widehat{\mathcal{C}}[c_2](\text{filter}(\neg b)(\hat{s})) & \end{cases}$$

$$\widehat{\mathcal{C}}[\text{while } b \text{ } c] = \lambda \hat{s}. \text{filter}(\neg b)(\text{fix}(\lambda \hat{x}. \hat{s} \sqcup \widehat{\mathcal{C}}[c](\text{filter}(b)(\hat{x}))))$$

Step 2: Abstract Semantics (Cont'd)

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

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

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

Other cases can be defined in similar ways.

Widening and Narrowing

During analyzing while-loop, replace \sqcup with ∇ and Δ in sequence (possibly after some iterations).

A simple widening operator:

$$\begin{aligned}[a, b] \nabla \perp &= [a, b] \\ \perp \nabla [c, d] &= [c, d] \\ [a, b] \nabla [c, d] &= [(c < a ? -\infty : a), (b < d ? \infty : b)]\end{aligned}$$

A simple narrowing operator:

$$\begin{aligned}[a, b] \Delta \perp &= \perp \\ \perp \Delta [a, b] &= \perp \\ [a, b] \Delta [c, d] &= [(a < -\infty ? c : a), (b = \infty ? d : b)]\end{aligned}$$

Exercise

Describe the result of the interval analysis at the entry of the loop.

- 1 without widening/narrowing
- 2 with widening/narrowing

```
1 x = 0;  
2 y = 0;  
3 while (x != 10) {  
4     x = x+1;  
5     y = y+1;  
6 }
```

Exercise

Describe the result of the interval analysis at the entry of the loop.

- 1 without widening/narrowing
- 2 with widening/narrowing

```
1 x = 0;  
2 y = 0;  
3 while (x != 10) {  
4     x = x+1;  
5     y = y+1;  
6 }
```

Lesson: Narrowing does not guarantee to regain precision!

Summary

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