

EC3204: Programming Languages and Compilers

Lecture 11 — Semantic Analysis (2) *Introduction to Abstract Interpretation*

Sunbeom So
Fall 2024

- **Abstract Interpretation** is a cost-effective analysis technology.
- Many useful static (compile-time) analyzers are based on abstract interpretation.
 - ▶ Infer (Meta): a tool for detecting memory leaks in Android applications.
 - ▶ Astrée (Airbus) – a static analyzer for aircraft software.
- **Key idea:** Over-approximation; abstract interpretation is a framework for automatically computing over-approximations of program states.



We can prove correctness by obtaining a **conservative approximation**.

Abstract Interpretation Recipe

To use abstract interpretation, follow the steps below.

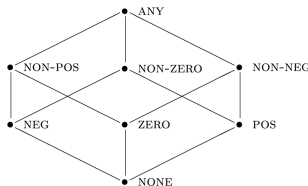
- ① **Abstract Domain** : define the abstract values that each variable can have (i.e., fixes “shape” of the invariants).
 - ▶ $c_1 \leq x \leq c_2$ (interval), $\pm x \pm y \leq c$ (octagon)
- ② **Abstract Semantics** (abstract transformers): define how to execute each statement in the chosen abstract domain.
- ③ **Fixed Point Computation**: Iteratively apply abstract transformers until you reach a fixed point.
 - ▶ The fixed point is an over-approximation of program states.
 - ▶ Sometimes done in abstract transformers.

Step 1: Abstract Domain

- Suppose we aim to infer invariants of the form $x \prec 0$ where $\prec \in \{>, \geq, <, \leq, =, \neq\}$.
- The abstract domain is defined as a pair **(Sign, \sqsubseteq)**:

$$\mathbf{Sign} = \{\top, \perp, \text{Pos}, \text{Neg}, \text{Zero}, \text{Non-Pos}, \text{Non-Neg}, \text{Non-Zero}\}$$

where $\top = \text{ANY}$, $\perp = \text{NONE}$, and the partial order (\sqsubseteq) is defined as:



Intuitively, $a \sqsubseteq b$ indicates b contains more information.

- A partially ordered set (poset) (D, \sqsubseteq) is **complete lattice**, iff every subset $Y \subseteq D$ has $\bigsqcup Y \in D$.

Step 1: Abstract Domain (Cont'd)

- The meaning of abstract domain (lattice) is defined by **abstraction** and **concretization** functions that relate concrete and abstract values.
- **Concretization function** (γ) maps each abstract value to sets of concrete elements.
 - ▶ $\gamma(\text{Pos}) = \{x \mid x \in \mathbb{Z} \wedge x > 0\}$
- **Abstraction function** (α) maps sets of concrete elements to values in the abstract domain.
 - ▶ $\alpha(\{0, 2, 10\}) = \text{Non-Neg}$
 - ▶ $\alpha(\{3, 114\}) = \text{Pos}$
 - ▶ $\alpha(\{-3, 2\}) = \text{Non-Zero}$

Step 1: Abstract Domain (Cont'd)

Formally, the abstraction of integers is defined as follows.

$$\begin{aligned}\alpha_{\text{Sign}} &: \mathcal{P}(\mathbb{Z}) \rightarrow \mathbf{Sign} \\ \alpha_{\text{Sign}}(Z) &= \sqcup_{z \in Z} \alpha'(z) \\ \text{where } \alpha'(z) &= \begin{cases} \text{Neg} & \dots z < 0 \\ \text{Zero} & \dots z = 0 \\ \text{Pos} & \dots z > 0 \end{cases}\end{aligned}$$

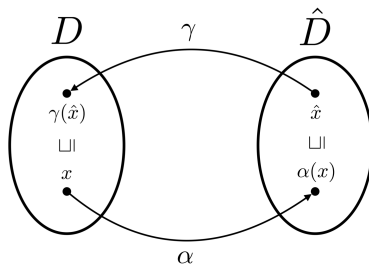
where join (\sqcup) is the least upper bound between two elements:

$$a \sqcup b = \begin{cases} a & \dots \text{if } b \sqsubseteq a \\ b & \dots \text{if } a \sqsubseteq b \\ \text{Non-Zero} & \dots \text{if } (a, b) = (\text{Neg}, \text{Pos}) \text{ or } (b, a) = (\text{Neg}, \text{Pos}) \\ \text{Non-Pos} & \dots \text{if } (a, b) = (\text{Neg}, \text{Zero}) \text{ or } (b, a) = (\text{Neg}, \text{Zero}) \\ \text{Non-Neg} & \dots \text{if } (a, b) = (\text{Zero}, \text{Pos}) \text{ or } (b, a) = (\text{Zero}, \text{Pos}) \\ \top & \dots \text{otherwise} \end{cases}$$

Step 1: Abstract Domain (Cont'd)

- **Important Requirement:** concrete domain D and abstract domain \hat{D} must be related through **Galois connection**:

$$\forall x \in D, \forall \hat{x} \in \hat{D}. \alpha(x) \sqsubseteq_A \hat{x} \iff x \sqsubseteq_C \gamma(\hat{x})$$



- α and γ respect the orderings in D and \hat{D} .
- The abstract value \hat{x} should capture all possibilities of the corresponding x .
 - ▶ Does $\alpha(\{2, 3\}) = \top$ and $\gamma(\top) = \mathbb{Z}$ satisfy Galois connection?

Step 1: Abstract Domain (Cont'd)

We can extend the lattice of abstract integers into that of abstract states.

- The complete lattice of abstract states ($\widehat{\mathbf{State}}, \sqsubseteq$):

$$\widehat{\mathbf{State}} = \mathit{Var} \rightarrow \mathbf{Sign}$$

with the pointwise ordering:

$$\hat{s}_1 \sqsubseteq \hat{s}_2 \iff \forall x \in \mathit{Var}. \hat{s}_1(x) \sqsubseteq \hat{s}_2(x).$$

- The least upper bound of $Y \subseteq \widehat{\mathbf{State}}$,

$$\bigsqcup Y = \lambda x. \bigsqcup_{\hat{s} \in Y} \hat{s}(x).$$

$$\text{i.e., } \hat{s}_1 \sqcup \hat{s}_2 = \lambda x. s_1(x) \sqcup s_2(x).$$

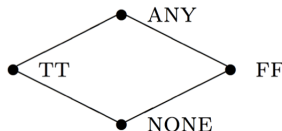
Step 1: Abstract Domain (Cont'd) - Abstract Booleans

The truth values $\mathbf{T} = \{true, false\}$ are abstracted by the complete lattice $(\widehat{\mathbf{T}}, \sqsubseteq)$:

$$\widehat{\mathbf{T}} = \{\top, \perp, \widehat{true}, \widehat{false}\}$$

where $\top = \text{ANY}$, $\perp = \text{NONE}$, $\widehat{true} = \text{TT}$, and $\widehat{false} = \text{FF}$.

$$\widehat{b}_1 \sqsubseteq \widehat{b}_2 \iff \widehat{b}_1 = \widehat{b}_2 \vee \widehat{b}_1 = \perp \vee \widehat{b}_2 = \top$$



Exercise) Define the abstraction function for the boolean lattice:

$$\alpha_{\widehat{\mathbf{T}}} : \mathcal{P}(\mathbf{T}) \rightarrow \widehat{\mathbf{T}}$$

Step 2: Abstract Semantics

- Given the abstract domain, we should define abstract transformers for each statement.
- A counter-part of concrete semantics.
 - ▶ In concrete execution, each statement changes concrete memory states.
 - ▶ In abstract execution, each statement changes abstract memory states.

We will consider the following language to define abstract semantics for our sign analysis.

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

Step 2: Abstract Semantics (Cont'd)

$$\widehat{\mathcal{A}} \llbracket a \rrbracket \quad : \quad \widehat{\mathbf{State}} \rightarrow \mathbf{Sign}$$

$$\widehat{\mathcal{A}} \llbracket n \rrbracket (\hat{s}) = \alpha_{\mathbf{Sign}}(\{n\})$$

$$\widehat{\mathcal{A}} \llbracket x \rrbracket (\hat{s}) = \hat{s}(x)$$

$$\widehat{\mathcal{A}} \llbracket a_1 + a_2 \rrbracket (\hat{s}) = \widehat{\mathcal{A}} \llbracket a_1 \rrbracket (\hat{s}) +_{\mathbf{Sign}} \widehat{\mathcal{A}} \llbracket a_2 \rrbracket (\hat{s})$$

$$\widehat{\mathcal{A}} \llbracket a_1 \star a_2 \rrbracket (\hat{s}) = \widehat{\mathcal{A}} \llbracket a_1 \rrbracket (\hat{s}) \star_{\mathbf{Sign}} \widehat{\mathcal{A}} \llbracket a_2 \rrbracket (\hat{s})$$

$$\widehat{\mathcal{A}} \llbracket a_1 - a_2 \rrbracket (\hat{s}) = \widehat{\mathcal{A}} \llbracket a_1 \rrbracket (\hat{s}) -_{\mathbf{Sign}} \widehat{\mathcal{A}} \llbracket a_2 \rrbracket (\hat{s})$$

Step 2: Abstract Semantics (Cont'd)

$+_S$	NONE	NEG	ZERO	POS	NON-POS	NON-ZERO	NON-NEG	ANY
NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
NEG	NONE	NEG	NEG	ANY	NEG	ANY	ANY	ANY
ZERO	NONE	NEG	ZERO	POS	NON-POS	NON-ZERO	NON-NEG	ANY
POS	NONE	ANY	POS	POS	ANY	ANY	POS	ANY
NON-POS	NONE	NEG	NON-POS	ANY	NON-POS	ANY	ANY	ANY
NON-ZERO	NONE	ANY	NON-ZERO	ANY	ANY	ANY	ANY	ANY
NON-NEG	NONE	ANY	NON-NEG	POS	ANY	ANY	NON-NEG	ANY
ANY	NONE	ANY	ANY	ANY	ANY	ANY	ANY	ANY

\star_S	NEG	ZERO	POS
NEG	POS	ZERO	NEG
ZERO	ZERO	ZERO	ZERO
POS	NEG	ZERO	POS

$-_S$	NEG	ZERO	POS
NEG	ANY	NEG	NEG
ZERO	POS	ZERO	NEG
POS	POS	POS	ANY

Step 2: Abstract Semantics (Cont'd)

$$\widehat{\mathcal{B}}[b] : \widehat{\text{State}} \rightarrow \widehat{\mathbf{T}}$$

$$\widehat{\mathcal{B}}[\text{true}](\hat{s}) = \widehat{\text{true}}$$

$$\widehat{\mathcal{B}}[\text{false}](\hat{s}) = \widehat{\text{false}}$$

$$\widehat{\mathcal{B}}[a_1 = a_2](\hat{s}) = \widehat{\mathcal{A}}[a_1](\hat{s}) =_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s})$$

$$\widehat{\mathcal{B}}[a_1 \leq a_2](\hat{s}) = \widehat{\mathcal{A}}[a_1](\hat{s}) \leq_{\text{Sign}} \widehat{\mathcal{A}}[a_2](\hat{s})$$

$$\widehat{\mathcal{B}}[\neg b](\hat{s}) = \neg_{\widehat{\mathbf{T}}} \widehat{\mathcal{B}}[b](\hat{s})$$

$$\widehat{\mathcal{B}}[b_1 \wedge b_2](\hat{s}) = \widehat{\mathcal{B}}[b_1](\hat{s}) \wedge_{\widehat{\mathbf{T}}} \widehat{\mathcal{B}}[b_2](\hat{s})$$

Step 2: Abstract Semantics (Cont'd)

$=_S$	NEG	ZERO	POS
NEG	ANY	FF	FF
ZERO	FF	TT	FF
POS	FF	FF	ANY

\leq_S	NEG	ZERO	POS
NEG	ANY	TT	TT
ZERO	FF	TT	TT
POS	FF	FF	ANY

\neg_T	
NONE	NONE
TT	FF
FF	TT
ANY	ANY

\wedge_T	NONE	TT	FF	ANY
NONE	NONE	NONE	NONE	NONE
TT	NONE	TT	FF	ANY
FF	NONE	FF	FF	FF
ANY	NONE	ANY	FF	ANY

Step 2: Abstract Semantics (Cont'd)

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

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

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

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

$$\widehat{\mathcal{C}}[\text{if } b \text{ } c_1 \text{ } c_2] = \widehat{\text{cond}}(\widehat{\mathcal{B}}[b], \widehat{\mathcal{C}}[c_1], \widehat{\mathcal{C}}[c_2])$$

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

$$\widehat{\text{cond}}(f, g, h)(\hat{s}) = \begin{cases} \perp & \dots f(\hat{s}) = \perp \\ g(\hat{s}) & \dots f(\hat{s}) = \widehat{\text{true}} \\ h(\hat{s}) & \dots f(\hat{s}) = \widehat{\text{false}} \\ g(\hat{s}) \sqcup h(\hat{s}) & \dots f(\hat{s}) = \top \end{cases}$$

Exercise: Abstract Semantics

Q1. Compute the final abstract state at the exit of the loop.

Q2. Is x always non-negative inside the loop?

```
1  x = 0;
2  y = 0;
3  while (y <= n) {
4      if (z == 0) {
5          x = x+1;
6      }
7      else {
8          x = x+y;
9      }
10     y= y+1;
11 }
```


Exercise: Abstract Semantics

Q1. Compute the final abstract state at the exit of the loop.

Q2. Is x always non-negative inside the loop?

```
1  x = 0;
2  y = 0;
3  while (y <= n) {
4      if (z == 0) {
5          x = x+1;
6      }
7      else {
8          x = x+y;
9      }
10     y= y+1;
11 }
```

	0	1	2
x	Zero	Non-Neg	Non-Neg
y	Zero	Non-Neg	Non-Neg
n	\top	\top	\top
z	\top	\top	\top

Important Requirement of Abstract Semantics

- To prove correctness, abstract semantics must be **sound** with respect to the concrete semantics (i.e., faithfully model the concrete semantics).
- Technically, the soundness of the abstract transformer \hat{F} means:

$$\forall x \in D, \forall \hat{x} \in \hat{D}. \alpha(x) \sqsubseteq \hat{x} \implies \alpha(F(x)) \sqsubseteq \hat{F}(\hat{x})$$

- If \hat{x} is an overapproximation of x , then $\hat{F}(\hat{x})$ is an over-approximation of $F(x)$.
 - ▶ The analysis result must be conservative with respect to actual program behaviors.

Summary

Abstract interpretation is a framework for automatically computing over-approximations of program states.

- ① **Abstract Domain** : define the abstract values that each variable can have (i.e., fixes “shape” of the invariants).
 - ▶ $c_1 \leq x \leq c_2$ (interval), $\pm x \pm y \leq c$ (octagon)
- ② **Abstract Semantics** (abstract transformers): define how to execute each statement in the chosen abstract domain.
- ③ **Fixed Point Computation**: Iteratively apply abstract transformers until you reach a fixed point.
 - ▶ The fixed point is an over-approximation of program states.
 - ▶ Sometimes done in abstract transformers.

Q. Does the fixed point computation always terminate?