EC4219: Software Engineering

Lecture 16 — Abstract Interpretation (1)

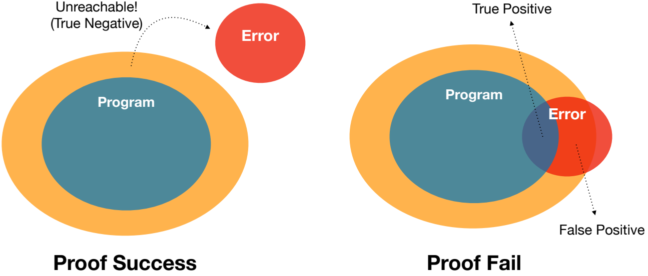
Sunbeom So
2024 Spring

Overview

- Deductive verifiers require annotations (e.g., loop invariants) from users.
- Fortunately, there are useful techniques that can automatically infer annotations (e.g., Houdini algorithm).
- **Abstract Interpretation** is a popular approach for this purpose.
- Many useful static 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

- In general, we cannot reason about exact program behaviors due to undecidability.
- However, we can still prove correctness by obtaining a **conservative approximation**.



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

Abstract Interpretation Recipe

To use abstract interpretation, follow the steps below.

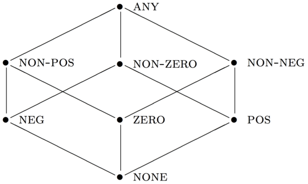
1. **Abstract Domain** : define the abstract values that each variable can have (i.e., fixes "shape" of the invariants).
   - $c_1 \leq x \leq c_2$ (interval), $\pm x \pm y \leq c$ (octagon)
2. **Abstract Semantics** (abstract transformers): define how to execute each statement in the chosen abstract 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 $x \prec \mathbf{0}$ where $\prec \in \{>, \geq, <, \leq, =, \neq\}$.
- The abstract domain is defined as a pair $(\mathbf{Sign}, \sqsubseteq)$:

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

where $\top$=ANY, $\bot$=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.
  - $\gamma(\text{Pos}) = \{x | x \in \mathbb{Z} \land x > 0\}$
- **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.

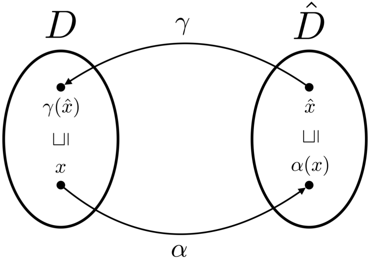
$$
\begin{array}{rcl}
\alpha_{\textsf{Sign}} & : & \mathcal{P}(\mathbb{Z}) \to \textbf{Sign} \\
\alpha_{\textsf{Sign}}(Z) & = & \sqcup_{z \in Z} \alpha'(z) \\
& & \text{where } \alpha'(z) = \begin{cases} \textsf{Neg} & \cdots\ z < 0 \\ \textsf{Zero} & \cdots\ z = 0 \\ \textsf{Pos} & \cdots\ z > 0 \end{cases}
\end{array}
$$

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

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

- **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 $(\widehat{\textbf{State}}, \sqsubseteq)$:

$$\widehat{\textbf{State}} = Var \rightarrow \textbf{Sign}$$

  with the pointwise ordering:

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

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

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

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

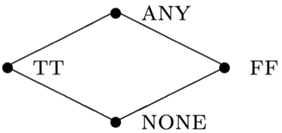
Step 1: Abstract Domain (Cont'd) - Abstract Booleans

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

$$\widehat{\mathsf{T}} = \{\top, \bot, \widehat{\boldsymbol{true}}, \widehat{\boldsymbol{false}}\}$$

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

$$\widehat{b_1} \sqsubseteq \widehat{b_2} \iff \widehat{b_1} = \widehat{b_2} \ \vee \ \widehat{b_1} = \bot \ \vee \ \widehat{b_2} = \top$$



Exercise) Define the abstraction function for the boolean lattice:

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

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.

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

Step 2: Abstract Semantics (Cont'd)

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

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)

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

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)

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

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

Exercise: Abstract Semantics

Q1. Compute the final abstract state at the exit of the loop. Q2. Is x always non-negative inside the loop?

```
x = 0;
y = 0;
while (y <= n) {
  if (z == 0) {
    x = x+1;
  }
  else {
    x = x+y;
  }
  y= y+1;
}
```

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:

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

- 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?