CS150 APL: Abstract Interpretation

Guannan Wei
guannan.wei@tufts.edu

Nov 18, 2025

Tufts University

Last time

- Abstract interpretation
- Lattice and abstract domains
- Soundness condition: galois connection
- Analyzing expressions and sequential control-flow with sign domains

Today

- Conditional
- Loops
- Fixpoints

Syntax

Syntax

$$
\begin{array}{lcll}
e & ::= & n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 / e_2 \mid \texttt{input}() & \text{(expressions)} \\
s & ::= & x \leftarrow e \mid \texttt{skip} \mid s_1 ; s_2 \mid \texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \mid \texttt{while}\ (e)\ s & \text{(statements)}
\end{array}
$$

- Analysis: range of numeric variables

Abstract domain

- $\langle D, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$
  - $\sqsubseteq$: $D \times D \to \mathbb{B}$: partial order
  - $\sqcup : D \times D \to D$: least upper bound (join)
  - $\sqcap : D \times D \to D$: greatest lower bound (meet)
  - $\bot \in D$: least element
  - $\top \in D$: greatest element
- Galois connection
  - $\alpha : \mathbb{Z} \to D$: abstraction function
  - $\gamma : D \to \mathcal{P}(\mathbb{Z})$: concretization function

Abstract semantics

- Expression and statement semantics take a $\widehat{Store} = Var \to D$

$$
\begin{aligned}
[\![e]\!] &: \widehat{Store} \to D \\
[\![n]\!]\hat{\sigma} &= \alpha(n) \\
[\![x]\!]\hat{\sigma} &= \hat{\sigma}(x) \\
[\![e_1 + e_2]\!]\hat{\sigma} &= [\![e_1]\!]\hat{\sigma} \mathbin{\hat{+}} [\![e_2]\!]\hat{\sigma} \\
[\![e_1 - e_2]\!]\hat{\sigma} &= [\![e_1]\!]\hat{\sigma} \mathbin{\hat{-}} [\![e_2]\!]\hat{\sigma} \\
[\![e_1 / e_2]\!]\hat{\sigma} &= [\![e_1]\!]\hat{\sigma} \mathbin{\hat{/}} [\![e_2]\!]\hat{\sigma} \\
[\![\texttt{input}()]\!]\hat{\sigma} &= \top_D
\end{aligned}
$$

Abstract semantics: Statements

- Statement semantics

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= ?
\end{aligned}
$$

Abstract semantics: Conditionals

- Need to cover both branches

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma}
\end{aligned}
$$

Abstract semantics: Conditionals

- Need to cover both branches

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma}
\end{aligned}
$$

- Have we defined $\sqcup$ for two abstract stores?

Detour: Point-wise lattice operations

- Given a lattice $\langle D, \sqsubseteq_D, \sqcup_D, \sqcap_D, \bot_D, \top_D \rangle$ and a set X
- $X \to D$ forms a lattice $\langle X \to D, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ where
  - $f_1 \sqsubseteq f_2$ iff $\forall x \in X. f_1(x) \sqsubseteq_D f_2(x)$
  - $(f_1 \sqcup f_2) = \lambda x. f_1(x) \sqcup_D f_2(x)$, $f_n(x) = \bot_D$ if $x \notin \mathsf{dom}(f_n)$
  - $(f_1 \sqcap f_2) = \lambda x. f_1(x) \sqcap_D f_2(x)$, $f_n(x) = \top_D$ if $x \notin \mathsf{dom}(f_n)$
  - $\bot = \lambda x. \bot_D$
  - $\top = \lambda x. \top_D$

Abstract semantics: Conditionals

- Need to cover both branches

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma}
\end{aligned}
$$

- Example: `if0 e then { x <- -5, y <- 3 } else { x <- 7 }`
  $\hat{\sigma_{\mathsf{thn}}} = \{x \mapsto \mathsf{neg}, y \mapsto \mathsf{pos}\}$, $\hat{\sigma_{\mathsf{els}}} = \{x \mapsto \mathsf{pos}\}$

Abstract semantics: Conditionals

- Need to cover both branches

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma}
\end{aligned}
$$

- Example: `if0 e then { x <- -5, y <- 3 } else { x <- 7 }`
  $\hat{\sigma_{\text{thn}}} = \{x \mapsto \text{neg}, y \mapsto \text{pos}\}$, $\hat{\sigma_{\text{els}}} = \{x \mapsto \text{pos}\}$
- Can we do better?

Abstract semantics: Conditionals

- Need to cover both branches
- Examine the condition if it can only be 0
  - Generic way using γ but not always computable
  - Could use domain-specific knowledge

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= \begin{cases} [\![s_1]\!]\hat{\sigma} & \text{if } \gamma([\![e]\!]\hat{\sigma}) = \{0\} \\ [\![s_2]\!]\hat{\sigma} & \text{if } 0 \notin \gamma([\![e]\!]\hat{\sigma}) \\ [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma} & \text{otherwise} \end{cases}
\end{aligned}
$$

While loops

- What about while loops?

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma} \\
[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} &= ?
\end{aligned}
$$

While loops

- What about while loops?

$$
\begin{aligned}
[\![s]\!] &: \widehat{Store} \to \widehat{Store} \\
[\![x \leftarrow e]\!]\hat{\sigma} &= \hat{\sigma}[x \mapsto [\![e]\!]\hat{\sigma}] \\
[\![\texttt{skip}]\!]\hat{\sigma} &= \hat{\sigma} \\
[\![s_1; s_2]\!]\hat{\sigma} &= [\![s_2]\!]([\![s_1]\!]\hat{\sigma}) \\
[\![\texttt{if0}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2]\!]\hat{\sigma} &= [\![s_1]\!]\hat{\sigma} \sqcup [\![s_2]\!]\hat{\sigma} \\
[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} &= ?
\end{aligned}
$$

Intuitively:

$$
[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} = [\![\texttt{if0}\ e\ \texttt{then}\ (s; \texttt{while}\ (e)\ s)\ \texttt{else skip}]\!]\hat{\sigma}
$$

While loops

Intuitively:

$$
\begin{aligned}
[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} &= [\![\texttt{if0}\ e\ \texttt{then}\ (s; \texttt{while}\ (e)\ s)\ \texttt{else}\ \texttt{skip}]\!]\hat{\sigma} \\
&= [\![s; \texttt{while}\ (e)\ s]\!]\hat{\sigma} \sqcup [\![\texttt{skip}]\!]\hat{\sigma} \\
&= [\![\texttt{while}\ (e)\ s]\!]([\![s]\!]\hat{\sigma}) \sqcup [\![\texttt{skip}]\!]\hat{\sigma} \\
&= [\![\texttt{if0}\ e\ \texttt{then}\ (s; \texttt{while}\ (e)\ s)\ \texttt{else}\ \texttt{skip}]\!]([\![s]\!]\hat{\sigma}) \sqcup \hat{\sigma} \\
&= ([\![s; \texttt{while}\ (e)\ s]\!][\![s]\!]\hat{\sigma} \sqcup [\![\texttt{skip}]\!]([\![s]\!]\hat{\sigma})) \sqcup \hat{\sigma} \\
&= ([\![\texttt{while}\ (e)\ s]\!]([\![s]\!]([\![s]\!]\hat{\sigma})) \sqcup ([\![s]\!]\hat{\sigma})) \sqcup \hat{\sigma} \\
&= \ldots
\end{aligned}
$$

While loops

$$[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} \quad = \text{fix}\ (F)(\hat{\sigma}) \text{ where } F(\hat{\sigma}') = [\![s]\!]\hat{\sigma}'$$

- Given a monotonic function $F : D \to D$, $x \in D$ is a fixpoint of F if $F(x) = x$
- fix $(F) : D \to D$ finds a fixpoint of F

While loops

$$[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} \quad = \text{fix}\ (F)(\hat{\sigma}) \text{ where } F(\hat{\sigma}') = [\![s]\!]\hat{\sigma}'$$

- Given a monotonic function $F : D \to D$, $x \in D$ is a fixpoint of F if $F(x) = x$
- fix $(F) : D \to D$ finds a fixpoint of F

Algorithmically:

```
def fix(f: D -> D, init: D) -> D:
  val next = f(init)
  if (next == init) next
  else fix(f, next)
```

Kleene's fixpoint theorem

$$[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} \quad = \text{fix}\ (F)(\hat{\sigma}) \text{ where } F(\hat{\sigma}') = [\![s]\!]\hat{\sigma}'$$

Does it always terminate?

Kleene's fixpoint theorem

$$[\![\texttt{while}\ (e)\ s]\!]\hat{\sigma} \quad = \text{fix}\ (F)(\hat{\sigma}) \text{ where } F(\hat{\sigma}') = [\![s]\!]\hat{\sigma}'$$

Does it always terminate?

- If D is a complete lattice and $F : D \to D$ is Scott-continous (implying monotonic), then F has a least fixpoint
  - fix $F = \bigsqcup_{i \geq 0} F^i(\bot) = \bigsqcup\{\bot, F(\bot), F(F(\bot)), F(F(F(\bot))), ...\}$
- Scott-continuous: $F(\bigsqcup C) = \bigsqcup\{F(c) \mid c \in C\}$

Next time

- More expressive abstract domains
- Widening for termination