Collecting Semantics
Instructor: Hossein Hojjat

November 8, 2017
• So far we abstracted the value of expressions
• Now we want to abstract the state at each point in the program
• First we define the concrete semantics that we are abstracting
• We will use a **collecting** semantics
Collecting Semantics

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly

Example of a program annotated with its collecting semantics:

\[
\begin{align*}
\ell_0 &: \text{ int } x := 0; \\
\ell_1 &: \text{ while } (x < 3) \\
\ell_2 &: \quad x := x + 2; \\
\ell_3 &:
\end{align*}
\]
Collecting Semantics

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly

Example of a program annotated with its collecting semantics:

\begin{align*}
\ell_0 : \text{int } x &:= 0; \\
\ell_1 : \text{while } (x < 3) \\
\ell_2 : & \quad x := x + 2; \\
\ell_3 : &
\end{align*}
Collecting Semantics

- Collecting semantics collects together all the states that can arise at each program point.
- Neither big nor the small-step semantics express this information directly.

Example of a program annotated with its collecting semantics:

\[ \ell_0 : \text{int } x := 3; \]
\[ \ell_1 : \textbf{while } (x \geq 3) \]
\[ \ell_2 : \quad x := x + 2; \]
\[ \ell_3 : \]

- Annotations can also be infinite sets of states.
Collecting semantics collects together all the states that can arise at each program point.

Neither big nor the small-step semantics express this information directly.

We can approximate sets of integers by "abstract values".

- e.g. intervals \([\text{low}, \text{high}]\)

Example of a program annotated with its collecting semantics:

\[
\begin{align*}
\ell_0 &: \text{int } x := 0; \\
\ell_1 &: \textbf{while } (x < 3) \\
\ell_2 &: \quad x := x + 2; \\
\ell_3 &:
\end{align*}
\]
Collecting Semantics Abstraction

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly
- We can approximate sets of integers by “abstract values”
  - e.g. intervals \([\text{low}, \text{high}]\)

Example of a program annotated with its collecting semantics:

\[
\begin{align*}
\ell_0 &: \text{int } x := 3; \\
\ell_1 &: \text{while } (x \geq 3) \\
\ell_2 &: \quad x := x + 2; \\
\ell_3 &: 
\end{align*}
\]

- We lose some precision but the annotations have become finitely representable
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

\[ \ell_0: \text{int } x := 0; \]
\[ \ell_1: \textbf{while } (x < 3) \]
\[ \ell_2: \quad x := x + 2; \]
\[ \ell_3: \]

\[ \ell_0 \quad \ell_1 \quad \ell_2 \quad \ell_3 \]

\[ x = \{0\} \]
\[ x := x + 2 \]
\[ [x < 3] \]
\[ x = \{\} \]

\[ [x \geq 3] \]
\[ x = \{\} \]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

$$\ell_0: \text{int } x := 0;$$
$$\ell_1: \textbf{while} \ (x < 3)$$
$$\ell_2: \quad x := x + 2;$$
$$\ell_3:$$
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

$\ell_0$: int $x := 0$

$\ell_1$: while $(x < 3)$

$\ell_2$: $x := x + 2$

$\ell_3$: 

Diagram:

- $\ell_0$: $x = \{0\}$
- $\ell_1$: $x = \{0, 2\}$
- $\ell_2$: $x = \{\}$
- $\ell_3$: $x = \{\}$
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

\[
\ell_0: \text{int } x := 0; \\
\ell_1: \textbf{while} \ (x < 3) \\
\ell_2: \quad x := x + 2; \\
\ell_3: \\
\]

\[x = \{0, 2\} \quad \ell_0 \quad x := 0 \quad \ell_1 \quad [x \geq 3] \quad \ell_3 \]

\[x = \{0, 2\} \quad \ell_2 \quad [x < 3] \quad \ell_1 \]

\[x = \{\} \quad x := x + 2 \quad \ell_2 \]

\[x = \{\} \quad \ell_3 \]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

\[
\begin{align*}
\ell_0: & \quad \text{int } x := 0; \\
\ell_1: & \quad \text{while } (x < 3) \\
\ell_2: & \quad x := x + 2; \\
\ell_3: & \\
\end{align*}
\]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

\[\ell_0: \text{int } x := 0;\]
\[\ell_1: \textbf{while } (x < 3)\]
\[\ell_2: \quad x := x + 2;\]
\[\ell_3: \]

Diagram:
- \(\ell_0\): \(x := 0\)
- \(\ell_1\): \((x < 3)\)
- \(\ell_2\): \(x := x + 2\)
- \(\ell_3\): 

\[x = \{0, 2, 4\}\]
\[\ell_1\]
\[x \geq 3\]
\[x = \{4\}\]
\[x := x + 2\]
\[x < 3\]
\[x = \{0, 2\}\]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

\[\ell_0: \text{int } x := 0;\]
\[\ell_1: \textbf{while } (x < 3)\]
\[\ell_2: \quad x := x + 2;\]
\[\ell_3: \]

\[
\begin{align*}
x &= [0, 0] & [x \geq 3] \\
x &= x + 2 & [x < 3]
\end{align*}
\]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

\[ \ell_0: \text{int } x := 0; \]
\[ \ell_1: \text{while } (x < 3) \]
\[ \ell_2: \quad x := x + 2; \]
\[ \ell_3: \]
• Start from an unannotated program
• Iterate abstract execution of the program on intervals until the annotations stabilize

\[\ell_0: \text{int } x := 0;\]
\[\ell_1: \text{while } (x < 3)\]
\[\ell_2: \quad x := x + 2;\]
\[\ell_3: \]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

\[
\ell_0: \text{int } x := 0; \\
\ell_1: \textbf{while} \ (x < 3) \\
\ell_2: \quad x := x + 2; \\
\ell_3: \\
\]

\begin{align*}
\ell_0 &: \quad x := 0; \\
\ell_1 &: \quad \textbf{while} \quad (x < 3) \\
\ell_2 &: \quad x := x + 2; \\
\ell_3 &: \\
\end{align*}
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

\[\ell_0: \text{int } x := 0;\]
\[\ell_1: \text{while } (x < 3)\]
\[\ell_2: \quad x := x + 2;\]
\[\ell_3: \]

\[x = [0, 4]\]
\[\ell_1\]
\[x = [0, 2]\]
\[\ell_2\]
\[x := x + 2\]
\[\ell_3\]
Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

\[ \ell_0: \text{int } x := 0; \]
\[ \ell_1: \textbf{while} (x < 3) \]
\[ \ell_2: \quad x := x + 2; \]
\[ \ell_3: \]
Compute Annotations

- Let $[\tau] : \text{State} \to \text{State}$ be the transfer function of transition $\tau$ in the control-flow graph
  - (strongest post-condition)
- Let $\mathcal{X}_i$ be the set of values at $i$ of CFG

\[ \mathcal{X}_0 = \{ x : \mathbb{Z} \} \]
\[ \mathcal{X}_1 = [x := 0] \mathcal{X}_0 \cup [x := x + 2] \mathcal{X}_2 \]
\[ \mathcal{X}_2 = [x < 3] \mathcal{X}_1 \]
\[ \mathcal{X}_3 = [x \geq 3] \mathcal{X}_1 \]
We can associate a system of semantic equations to the collecting semantics of the form

\[ \mathcal{X} = F(\mathcal{X}) \quad \text{with} \quad \mathcal{X} \in (\mathcal{P}(\text{Var} \rightarrow \mathbb{Z}))^n \]

where \( n \) is the number of states in CFG

Collecting semantics is the fixpoint solution of the semantic equation

Tarski’s theorem guarantees fixpoints in complete lattices

- But the above proof does not say how to find them.

It is always computable when \( F \) is omega continuous

\[
\text{lfp}(F) = \bigcup_{n \geq 0} F^n(\emptyset, \ldots, \emptyset)
\]
We can formulate analogous constraints in abstract domain

Approximate Abstraction:
\[ \alpha(lfp(F)) \leq lfp(F^A) \]

Exact Abstraction:
\[ \alpha(lfp(F)) = lfp(F^A) \]
Abstract Transfer Functions

For $[\tau] : C \rightarrow C$ we define the best abstract transfer function $[[\tau]]^A : A \rightarrow A$

- Let $\sigma^A$ maps variables to interval lattice elements

  $[[x := y + z]]^A(\sigma^A) = \sigma^A[x \mapsto [l, h]]$ where $l = \sigma^A(y).\text{low} + \sigma^A(z).\text{low}$ and $h = \sigma^A(y).\text{high} + \sigma^A(z).\text{high}$

  $[[x := y + z]]^A(\sigma^A) = \sigma^A$ where $\sigma^A(y) = \bot \lor \sigma^A(z) = \bot$

- (Recall $[a, b] +^A [c, d] = [a + c, b + d]$)
Abstract Semantic Function

- Let $\mathcal{X}^A_i$ be the abstraction if set of values at $i$ of CFG

  $\mathcal{X}^A_0 = \{x : [-\infty, +\infty]\}$
  
  $\mathcal{X}^A_1 = [x := 0]^A \mathcal{X}^A_0 \sqcup [x := x + 2]^A \mathcal{X}^A_2$
  
  $\mathcal{X}^A_2 = [x < 3]^A \mathcal{X}^A_1$
  
  $\mathcal{X}^A_3 = [x \geq 3]^A \mathcal{X}^A_1$

- Fixpoint of $F^A$ is an over-approximate of fixpoint of $F$