Properties of Axiomatic Semantics
CS 4301/6371: Advanced Programming Languages

Kevin W. Hamlen

April 25, 2024
Goals of any axiomatic semantics:

- **Soundness:** If a Hoare triple $\{A\} c \{B\}$ is derivable, it is “true”.
- **Completeness:** If a Hoare triple $\{A\} c \{B\}$ is “true”, it is derivable.

Are our 6 axiomatic semantic rules sound and complete?

- Must first formally define what is meant by “true” in the above
- Typically we define this using... *denotational semantics*!
Denotations of Assertion Expressions

(1) Extend expression denotations $\mathcal{E}$ to include meta-variables $\bar{v}$:

- stores: $\Sigma : v \rightarrow \mathbb{Z}$
- interpretations: $\bar{\Sigma} : \bar{v} \rightarrow \mathbb{Z}$
- exp denotations: $\mathcal{E} : e \rightarrow \bar{\Sigma} \rightarrow \Sigma \rightarrow \mathbb{Z}$

\[
\begin{align*}
\mathcal{E}[n]\bar{\sigma}\sigma &= n \\
\mathcal{E}[v]\bar{\sigma}\sigma &= \sigma(v) \\
\mathcal{E}[\bar{v}]\bar{\sigma}\sigma &= \bar{\sigma}(\bar{v}) \\
\mathcal{E}[e_1 + e_2]\bar{\sigma}\sigma &= \mathcal{E}[e_1]\bar{\sigma}\sigma + \mathcal{E}[e_2]\bar{\sigma}\sigma \\
\mathcal{E}[e_1 - e_2]\bar{\sigma}\sigma &= \mathcal{E}[e_1]\bar{\sigma}\sigma - \mathcal{E}[e_2]\bar{\sigma}\sigma \\
\mathcal{E}[e_1 \ast e_2]\bar{\sigma}\sigma &= \mathcal{E}[e_1]\bar{\sigma}\sigma \cdot \mathcal{E}[e_2]\bar{\sigma}\sigma
\end{align*}
\]
(2) Define denotations $\mathcal{A}$ of assertions $A$:

assertion denotations \( \mathcal{A} : A \rightarrow \Sigma \rightarrow \Sigma \rightarrow \{T, F\} \)

\[
\begin{align*}
\mathcal{A}[T]\bar{\sigma}\sigma &= T \\
\mathcal{A}[F]\bar{\sigma}\sigma &= F \\
\mathcal{A}[e_1 \leq e_2]\bar{\sigma}\sigma &= \mathcal{E}[e_1]\bar{\sigma}\sigma \leq \mathcal{E}[e_2]\bar{\sigma}\sigma \\
\mathcal{A}[A_1 \Rightarrow A_2]\bar{\sigma}\sigma &= \mathcal{A}[A_1]\bar{\sigma}\sigma \Rightarrow \mathcal{A}[A_2]\bar{\sigma}\sigma \\
\mathcal{A}[\forall \bar{v}.A]\bar{\sigma}\sigma &= \forall i \in \mathbb{Z}, \mathcal{A}[A](\bar{\sigma}[\bar{v} \mapsto i])\sigma \\
\vdots
\end{align*}
\]
(3) Notations:
\[ \bar{\sigma}, \sigma \models A \text{ asserts } A[\bar{A}]\bar{\sigma}\sigma \]
\[ \sigma \models A \text{ asserts } \forall \bar{\sigma} \in \bar{\Sigma}, (\bar{\sigma}, \sigma \models A) \]
\[ \models A \text{ asserts } \forall \sigma \in \Sigma, (\sigma \models A) \]

Note: \( \models A \) is our notation from the Rule of Consequence.

(4) Hoare Triple Denotations: \( \models \{A\}c\{B\} \) asserts:

\[ \forall \bar{\sigma} \in \bar{\Sigma}, \forall \sigma, \sigma' \in \Sigma, (\bar{\sigma}, \sigma \models A) \land ((\sigma, \sigma') \in C[\cdot]) \Rightarrow (\bar{\sigma}, \sigma' \models B) \]

Note: \( C[\cdot] \) is the denotational semantics of the target programming language.
Proving Soundness

**Theorem (Soundness)**

If \( \{A\}c\{B\} \) is derivable then \( \models \{A\}c\{B\} \) holds.

**Proof**

Let \( \bar{\sigma} \in \bar{\Sigma} \) and \( \sigma, \sigma' \in \Sigma \) be given such that \( \bar{\sigma}, \sigma \models A \) and \( (\sigma, \sigma') \in C[c] \).

(Goal: Prove \( \bar{\sigma}, \sigma' \models B \).)
Proving Soundness

Theorem (Soundness)
If \( \{A\}c\{B\} \) is derivable then \( \models \{A\}c\{B\} \) holds.

Proof
Let \( \bar{\sigma} \in \bar{\Sigma} \) and \( \sigma, \sigma' \in \Sigma \) be given such that \( \bar{\sigma}, \sigma \models A \) and \( (\sigma, \sigma') \in C[c] \).
Let \( D \) be a derivation of \( \{A\}c\{B\} \). Proof is by structural induction over \( D \).

IH: If \( \{A_0\}c_0\{B_0\} \) has a derivation \( D_0 < D \), then \( \models \{A_0\}c_0\{B_0\} \) holds.

Case 1: Suppose \( D \) ends in Rule 1:

\[
D = \frac{\{A\}\text{skip}\{A\}}{(1)}
\]

Thus \( c = \text{skip} \) and \( B = A \).

(Goal: Prove \( \bar{\sigma}, \sigma' \models B \).)
Proving Soundness

Theorem (Soundness)

If \( \{A\}c\{B\} \) is derivable then \( \models \{A\}c\{B\} \) holds.

Proof

Let \( \bar{\sigma} \in \bar{\Sigma} \) and \( \sigma, \sigma' \in \Sigma \) be given such that \( \bar{\sigma}, \sigma \models A \) and \( (\sigma, \sigma') \in C[c] \).
Let \( D \) be a derivation of \( \{A\}c\{B\} \). Proof is by structural induction over \( D \).

IH: If \( \{A_{0}\}c_{0}\{B_{0}\} \) has a derivation \( D_{0} < D \), then \( \models \{A_{0}\}c_{0}\{B_{0}\} \) holds.

Case 1: Suppose \( D \) ends in Rule 1:

\[
D = \begin{array}{c}
\{A\}skip\{A\}
\end{array}
\]

(1)

Thus \( c = skip \) and \( B = A \). Since \( \sigma' = C[skip]\sigma = \sigma \) and \( B = A \), assumption \( \bar{\sigma}, \sigma \models A \) implies \( \bar{\sigma}, \sigma' \models B \).

\[
\cdots
\]

(Goal: Prove \( \bar{\sigma}, \sigma' \models B \).)
Recall: $\models \{A\}c\{B\}$ asserts

$$\forall \bar{\sigma} \in \bar{\Sigma}, \forall \sigma, \sigma' \in \Sigma, (\bar{\sigma}, \sigma \models A) \land ((\sigma, \sigma') \in C[c]) \Rightarrow (\bar{\sigma}, \sigma' \models B)$$

**Theorem (Completeness)**

If $\models \{A\}c\{B\}$ then $\{A\}c\{B\}$ is derivable.

**Proof**

Assume $\models \{A\}c\{B\}$.
Completeness

Recall: \( \models \{ A \} c \{ B \} \) asserts

\[
\forall \bar{\sigma} \in \bar{\Sigma}, \forall \sigma, \sigma' \in \Sigma, (\bar{\sigma}, \sigma \models A) \land ((\sigma, \sigma') \in C[c]) \Rightarrow (\bar{\sigma}, \sigma' \models B)
\]

Theorem (Completeness)
If \( \models \{ A \} c \{ B \} \) then \( \{ A \} c \{ B \} \) is derivable.

- Impossible! Recall our friend Kurt Gödel:
  
  No finite collection of axioms is both sound and complete.

- BUT... Stephen Cook\(^1\) (of P v. NP fame) comes to our rescue:
  - **Relative Completeness:** Given an oracle that (magically) derives the \( \models A \) premises in the Rule of Consequence (whenever they are true), Hoare logic is complete.
  - In essence, Hoare Logic is “as complete as possible” given the inherent incompleteness of mathematics in general.

Preconditions & Postconditions

Edsger Dijkstra’s idea: The strongest correctness assertions are those where

- the precondition is “weakest” (fewest assumptions)
- the postcondition is “strongest” (most conclusions)

Formally:

- We say “$D$ is (strictly) weaker than $C$” and “$C$ is (strictly) stronger than $D$” if $C \Rightarrow D$ (and $D \not\Rightarrow C$).
- A is a **weakest precondition** of program $c$ for postcondition $B$ iff every precondition $A_0$ satisfying $\{A_0\}c\{B\}$ implies $A$.
- B is a **strongest postcondition** of program $c$ for precondition $A$ iff $B$ implies every postcondition $B_0$ satisfying $\{A\}c\{B_0\}$.
Can Weakest Preconditions be Computed?

**Idea**

\[ wp(c, B) \] should return a weakest precondition \( A \) for command \( c \) with postcondition \( B \).

\[ wp(\text{skip}, B) = ? \]
Can Weakest Preconditions be Computed?

**Idea**

\( wp(c, B) \) should return a weakest precondition \( A \) for command \( c \) with postcondition \( B \).

\[
wp(\texttt{skip}, B) = B
\]
Can Weakest Preconditions be Computed?

**Idea**

wp\((c, B)\) should return a weakest precondition \(A\) for command \(c\) with postcondition \(B\).

\[
wp(\text{skip}, B) = B \\
wp(c_1 ; c_2, B) =
\]
Can Weakest Preconditions be Computed?

Idea

$wp(c, B)$ should return a weakest precondition $A$ for command $c$ with postcondition $B$.

\[
wp(\text{skip}, B) = B \\
wp(c_1 ; c_2, B) = wp(c_1, wp(c_2, B))
\]
Can Weakest Preconditions be Computed?

Idea

\( wp(c, B) \) should return a weakest precondition \( A \) for command \( c \) with postcondition \( B \).

\[
\begin{align*}
wp(\text{skip}, B) &= B \\
wp(c_1 ; c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(x := e, B) &=
\end{align*}
\]
Can Weakest Preconditions be Computed?

Idea

\( wp(c, B) \) should return a weakest precondition \( A \) for command \( c \) with postcondition \( B \).

- \( wp(\text{skip}, B) = B \)
- \( wp(c_1 ; c_2, B) = wp(c_1, wp(c_2, B)) \)
- \( wp(x := e, B) = B[e/x] \)
Can Weakest Preconditions be Computed?

**Idea**

\( wpc, B) \) should return a weakest precondition \( A \) for command \( c \) with postcondition \( B \).

\[
\begin{align*}
wp(\text{skip}, B) &= B \\
wp(c_1; c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(x := e, B) &= B[e/x] \\
wp(\text{if } b \text{ then } c_1 \text{ else } c_2, B) &=
\end{align*}
\]
## Can Weakest Preconditions be Computed?

### Idea

$\wp(c, B)$ should return a weakest precondition $A$ for command $c$ with postcondition $B$.

\[
\begin{align*}
\wp(\text{skip}, B) &= B \\
\wp(c_1 ; c_2, B) &= \wp(c_1, \wp(c_2, B)) \\
\wp(x := e, B) &= B[e/x] \\
\wp(\text{if } b \text{ then } c_1 \text{ else } c_2, B) &= (b \Rightarrow \wp(c_1, B)) \land (\neg b \Rightarrow \wp(c_2, B))
\end{align*}
\]
Can Weakest Preconditions be Computed?

Idea

$\wp(c, B)$ should return a weakest precondition $A$ for command $c$ with postcondition $B$.

\[
\begin{align*}
\wp(\text{skip}, B) &= B \\
\wp(c_1; c_2, B) &= \wp(c_1, \wp(c_2, B)) \\
\wp(x := e, B) &= B[e/x] \\
\wp(\text{if } b \text{ then } c_1 \text{ else } c_2, B) &= (b \implies \wp(c_1, B)) \land (\neg b \implies \wp(c_2, B)) \\
\wp(\text{while } b \text{ do } c, B) &=
\end{align*}
\]
Can Weakest Preconditions be Computed?

**Idea**

\( \text{wp}(c, B) \) should return a weakest precondition \( A \) for command \( c \) with postcondition \( B \).

\[
\begin{align*}
\text{wp}(\text{skip}, B) &= B \\
\text{wp}(c_1 ; c_2, B) &= \text{wp}(c_1, \text{wp}(c_2, B)) \\
\text{wp}(x := e, B) &= B[e/x] \\
\text{wp}(\text{if } b \text{ then } c_1 \text{ else } c_2, B) &= (b \Rightarrow \text{wp}(c_1, B)) \land (\neg b \Rightarrow \text{wp}(c_2, B)) \\
\text{wp}(\text{while } b \text{ do } c, B) &= \text{undecidable?}
\end{align*}
\]
Can Weakest Preconditions be Computed?

Idea

$wp(c, B)$ should return a weakest precondition $A$ for command $c$ with postcondition $B$.

$$
wp(\text{skip}, B) = B \\
wp(c_1 ; c_2, B) = wp(c_1, wp(c_2, B)) \\
wp(x := e, B) = B[e/x] \\
wp(\text{if } b \text{ then } c_1 \text{ else } c_2, B) = (b \Rightarrow wp(c_1, B)) \land (\neg b \Rightarrow wp(c_2, B)) \\
wp(\text{while } b \text{ do } c, B) = \forall \sigma \in \Sigma, \forall \bar{k}, \left(\forall i, (0 \leq i < \bar{k}) \Rightarrow C[c]^i \sigma \models b \right) \\
\Rightarrow \left(C[c]^\bar{k} \sigma \models b \lor B\right)
$$
Can Weakest Preconditions be Computed?

Idea

$\text{wp}(c, B)$ should return a weakest precondition $A$ for command $c$ with postcondition $B$.

$$
\text{wp}(\text{skip}, B) = B \\
\text{wp}(c_1 ; c_2, B) = \text{wp}(c_1, \text{wp}(c_2, B)) \\
\text{wp}(x := e, B) = B[e/x] \\
\text{wp}(\text{if } b \text{ then } c_1 \text{ else } c_2, B) = (b \Rightarrow \text{wp}(c_1, B)) \land (\neg b \Rightarrow \text{wp}(c_2, B)) \\
\text{wp}(\text{while } b \text{ do } c, B) = \forall \sigma \in \Sigma, \forall \bar{k}, (\forall i, (0 \leq i < \bar{k}) \Rightarrow C[c]^i \sigma \models b) \\
\Rightarrow (C[c]^\bar{k} \sigma \models b \lor B)
$$

Not supported by our assertion language (but turns out one can encode them):

- quantification over non-integers ($\forall \sigma \in \Sigma \ldots$)
- all of denotational semantics(!) ($C[c]$)
- function $n$-composition ($f^n$)
- axiomatic denotations ($\models$)
Exercise: Define an algorithm \( sp(A, c) \) that computes the strongest postcondition \( B \) for program \( c \) with precondition \( A \).

- Don’t worry about while-loops (hard!)
- Mostly similar to \( wp \) algorithm but assignment rule is messy

More (optional) topics:

- Read about *Dijkstra guarded commands*.
- Read “The Science of Programming” by David Gries (classic text).
- Read about *verification condition generators*.