Properties of Axiomatic Semantics

CS 4301/6371: Advanced Programming Languages

Kevin W. Hamlen

April 25, 2024

Motivation

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

$$
\begin{aligned}
\mathcal{E}[\![n]\!]\bar{\sigma}\sigma &= n \\
\mathcal{E}[\![v]\!]\bar{\sigma}\sigma &= \sigma(v) \\
\mathcal{E}[\![\bar{v}]\!]\bar{\sigma}\sigma &= \bar{\sigma}(\bar{v}) \\
\mathcal{E}[\![e_1 \texttt{+} e_2]\!]\bar{\sigma}\sigma &= \mathcal{E}[\![e_1]\!]\bar{\sigma}\sigma + \mathcal{E}[\![e_2]\!]\bar{\sigma}\sigma \\
\mathcal{E}[\![e_1 \texttt{-} e_2]\!]\bar{\sigma}\sigma &= \mathcal{E}[\![e_1]\!]\bar{\sigma}\sigma - \mathcal{E}[\![e_2]\!]\bar{\sigma}\sigma \\
\mathcal{E}[\![e_1 \texttt{*} e_2]\!]\bar{\sigma}\sigma &= \mathcal{E}[\![e_1]\!]\bar{\sigma}\sigma \cdot \mathcal{E}[\![e_2]\!]\bar{\sigma}\sigma
\end{aligned}
$$

Denotations of Assertions

(2) Define denotations $\mathcal{A}$ of assertions A:

assertion denotations $\quad \mathcal{A} : A \to \bar{\Sigma} \to \Sigma \rightharpoonup \{T, F\}$

$$
\begin{aligned}
\mathcal{A}[\![T]\!]\bar{\sigma}\sigma &= T \\
\mathcal{A}[\![F]\!]\bar{\sigma}\sigma &= F \\
\mathcal{A}[\![e_1 \le e_2]\!]\bar{\sigma}\sigma &= \mathcal{E}[\![e_1]\!]\bar{\sigma}\sigma \le \mathcal{E}[\![e_2]\!]\bar{\sigma}\sigma \\
\mathcal{A}[\![A_1 \Rightarrow A_2]\!]\bar{\sigma}\sigma &= \mathcal{A}[\![A_1]\!]\bar{\sigma}\sigma \Rightarrow \mathcal{A}[\![A_2]\!]\bar{\sigma}\sigma \\
\mathcal{A}[\![\forall \bar{v}.A]\!]\bar{\sigma}\sigma &= \forall i \in \mathbb{Z}, \mathcal{A}[\![A]\!](\bar{\sigma}[\bar{v} \mapsto i])\sigma \\
&\vdots
\end{aligned}
$$

Denotations of Judgments

(3) Notations:

$$\bar{\sigma}, \sigma \models A \text{ asserts } \mathcal{A}[\![A]\!]\bar{\sigma}\sigma$$
$$\sigma \models A \text{ asserts } \forall \bar{\sigma} \in \bar{\Sigma}, (\bar{\sigma}, \sigma \models A)$$
$$\models A \text{ asserts } \forall \sigma \in \Sigma, (\sigma \models A)$$

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

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

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

Note: $\mathcal{C}[\![c]\!]$ 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 $(\sigma, \sigma') \in \mathcal{C}[\![c]\!]$.

(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 $(\sigma, \sigma') \in \mathcal{C}[\![c]\!]$.
Let $\mathcal{D}$ be a derivation of $\{A\}c\{B\}$. Proof is by structural induction over $\mathcal{D}$.

IH: If $\{A_0\}c_0\{B_0\}$ has a derivation $\mathcal{D}_0 < \mathcal{D}$, then $\models \{A_0\}c_0\{B_0\}$ holds.

Case 1: Suppose $\mathcal{D}$ ends in Rule 1:

$$\mathcal{D} = \frac{}{\{A\}\mathbf{skip}\{A\}}(1)$$

Thus $c = \mathbf{skip}$ 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 $(\sigma, \sigma') \in \mathcal{C}[\![c]\!]$.
Let $\mathcal{D}$ be a derivation of $\{A\}c\{B\}$. Proof is by structural induction over $\mathcal{D}$.

IH: If $\{A_0\}c_0\{B_0\}$ has a derivation $\mathcal{D}_0 < \mathcal{D}$, then $\models \{A_0\}c_0\{B_0\}$ holds.

Case 1: Suppose $\mathcal{D}$ ends in Rule 1:

$$\mathcal{D} = \frac{}{\{A\}\mathbf{skip}\{A\}}(1)$$

Thus $c = \mathbf{skip}$ and $B = A$. Since $\sigma' = \mathcal{C}[\![\mathbf{skip}]\!]\sigma = \sigma$ and $B = A$, assumption $\bar{\sigma}, \sigma \models A$ implies $\bar{\sigma}, \sigma' \models B$.

...

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

Completeness

Recall: $\models \{A\}c\{B\}$ asserts

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

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

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

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.

[1] S.A. Cook, *"Soundness and Completeness of an Axiom System for Program Verification,"* SIAM J. Comput. 7(1):70–90, Feb. 1978.

Preconditions & Postconditions

$$\{A\}c\{B\}$$

$$\underset{\text{precondition}}{\uparrow} \qquad \underset{\text{postcondition}}{\uparrow}$$

- 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 $D \nRightarrow C$).
  - 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.

$$wp(\texttt{skip}, B) = \text{?}$$

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.

$$wp(\texttt{skip}, B) = B$$
$$wp(c_1\,; c_2, B) =$$

Can Weakest Preconditions be Computed?

Idea

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

$$wp(\mathbf{skip}, B) = B$$
$$wp(c_1\,;c_2, B) = wp(c_1, wp(c_2, B))$$

Can Weakest Preconditions be Computed?

Idea

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

$$wp(\textbf{skip}, B) = B$$
$$wp(c_1\,;c_2, B) = wp(c_1, wp(c_2, B))$$
$$wp(\mathtt{x:=}\,e, B) =$$

Can Weakest Preconditions be Computed?

Idea

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

$$
\begin{aligned}
wp(\texttt{skip}, B) &= B \\
wp(c_1\,;c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(\texttt{x:=}\,e, B) &= B[e/x]
\end{aligned}
$$

Can Weakest Preconditions be Computed?

Idea

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

$$\begin{aligned} wp(\mathbf{skip}, B) &= B \\ wp(c_1\,\mathtt{;}\,c_2, B) &= wp(c_1, wp(c_2, B)) \\ wp(\mathtt{x}\,\mathtt{:=}\,e, B) &= B[e/x] \\ wp(\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2, B) &= \end{aligned}$$

Can Weakest Preconditions be Computed?

Idea

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

$$\begin{aligned} wp(\texttt{skip}, B) &= B \\ wp(c_1 \,\texttt{;}\, c_2, B) &= wp(c_1, wp(c_2, B)) \\ wp(\texttt{x:=}\, e, B) &= B[e/x] \\ wp(\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, B) &= (b \Rightarrow wp(c_1, B)) \wedge (\neg b \Rightarrow wp(c_2, B)) \end{aligned}$$

Can Weakest Preconditions be Computed?

Idea

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

$$
\begin{aligned}
wp(\mathbf{skip}, B) &= B \\
wp(c_1\,;c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(\mathtt{x:=}e, B) &= B[e/x] \\
wp(\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2, B) &= (b \Rightarrow wp(c_1, B)) \wedge (\neg b \Rightarrow wp(c_2, B)) \\
wp(\mathbf{while}\ b\ \mathbf{do}\ c, B) &=
\end{aligned}
$$

Can Weakest Preconditions be Computed?

Idea

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

$$
\begin{aligned}
wp(\texttt{skip}, B) &= B \\
wp(c_1\,;c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(\texttt{x:=}\,e, B) &= B[e/x] \\
wp(\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, B) &= (b \Rightarrow wp(c_1, B)) \land (\neg b \Rightarrow wp(c_2, B)) \\
wp(\texttt{while}\ b\ \texttt{do}\ c, B) &= \text{undecidable?}
\end{aligned}
$$

Can Weakest Preconditions be Computed?

Idea

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

$$
\begin{aligned}
wp(\textbf{skip}, B) &= B \\
wp(c_1\,\textbf{;}\,c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(\texttt{x:=}\,e, B) &= B[e/x] \\
wp(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, B) &= (b \Rightarrow wp(c_1, B)) \wedge (\neg b \Rightarrow wp(c_2, B)) \\
wp(\textbf{while } b \textbf{ do } c, B) &= \forall \sigma \in \Sigma, \forall \bar{k}, \Big(\forall \bar{i}, (0 \leq \bar{i} < \bar{k}) \Rightarrow \mathcal{C}[\![c]\!]^{\bar{i}}\sigma \models b\Big) \\
&\qquad \Rightarrow (\mathcal{C}[\![c]\!]^{\bar{k}}\sigma \models b \vee B)
\end{aligned}
$$

Can Weakest Preconditions be Computed?

Idea

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

$$
\begin{aligned}
wp(\textbf{skip}, B) &= B \\
wp(c_1\,\mathtt{;}\,c_2, B) &= wp(c_1, wp(c_2, B)) \\
wp(\mathtt{x := } e, B) &= B[e/x] \\
wp(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, B) &= (b \Rightarrow wp(c_1, B)) \wedge (\neg b \Rightarrow wp(c_2, B)) \\
wp(\textbf{while } b \textbf{ do } c, B) &= \forall \sigma \in \Sigma, \forall \bar{k}, \big(\forall \bar{i}, (0 \leq \bar{i} < \bar{k}) \Rightarrow \mathcal{C}[\![c]\!]^{\bar{i}} \sigma \models b\big) \\
&\qquad \Rightarrow (\mathcal{C}[\![c]\!]^{\bar{k}} \sigma \models b \vee B)
\end{aligned}
$$

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(!) ($\mathcal{C}[\![c]\!]$)
- function n-composition (f^n)
- axiomatic denotations ($\models$)

Exercises and Supplemental Topics

- 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*.