Axiomatic Semantics
CS 6371: Advanced Programming Languages

Kevin W. Hamlen

April 23, 2023
Operational Semantics
- large-step and small-step varieties
- formally defines the *operation* of a machine that executes the program

Denotational Semantics
- defines the mathematical object (i.e., function) that a program *denotes*

Static Semantics (Type Theory)
- a *static analysis* that prevents certain runtime errors ("stuck states")

Today: Axiomatic Semantics
Goal: We wish to prove complete correctness of mission-critical code.
- Type-theory too weak* (just proves soundness)
- Operational semantics requires us to step outside the derivation system to prove things about derivations. Non-derivation parts cannot be machine-checked.
- Denotational semantics creates a massive mathematical object that encodes all memory states (too hard to reason about).

Solution: Axiomatic Semantics
- inference rules that encapsulate the entire correctness proof into a derivation
- Derivation is fully machine-checkable, so no reliance on (error-prone) humans writing perfect proofs or perfectly checking proofs.

* Actually, advanced type systems like $\lambda_C$ encode an entire axiomatic semantics into the type system, but let’s classify that as type theory + axiomatic semantics.
Two Kinds of Correctness

- Partial Correctness
  - Notation: \( \{ A \} c \{ B \} \) (called a Hoare triple)
  - If \( A \) is true before executing \( c \), and if \( c \) terminates, then \( B \) is true after executing \( c \).
  - \( A \) is precondition, and \( B \) is postcondition

- Total Correctness
  - Notation: \([ A ] c [ B ]\)
  - If \( A \) is true before executing \( c \), then \( c \) eventually terminates and \( B \) is true once it does.
Examples

1. \{x \leq 10\} while x <= 10 do x := x + 1{?}
Examples

1 \{x \leq 10\} \textbf{while} x \leq 10 \textbf{do} x := x + 1\{x = 11\}
Examples

1. \{x \leq 10\} \textbf{while} x \leq 10 \textbf{ do } x := x + 1 \{x = 11\}

2. \{x \leq 10\} \textbf{while} x \leq 10 \textbf{ do } x := x + 1[?]
Examples

1. \( \{ x \leq 10 \} \text{while } x \leq 10 \text{ do } x := x + 1 \{ x = 11 \} \)

2. \( [ x \leq 10 ] \text{while } x \leq 10 \text{ do } x := x + 1 \{ x = 11 \} \)
Examples

1. \( \{ x \leq 10 \} \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{ x = 11 \} \)
2. \([ x \leq 10 ] \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{ x = 11 \} \)
3. \([T] \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{?\} \)
Examples

1. \( \{ x \leq 10 \} \textbf{while} \ x <= 10 \ \textbf{do} \ x := x + 1 \{ x = 11 \} \)

2. \([ x \leq 10 ]\textbf{while} \ x <= 10 \ \textbf{do} \ x := x + 1 \{ x = 11 \} \)

3. \([ T ]\textbf{while} \ x <= 10 \ \textbf{do} \ x := x + 1 \{ x \geq 11 \} \)
Examples

1. \{ x \leq 10 \} \textbf{while} x \leq 10 \textbf{do} x := x + 1 \{ x = 11 \}

2. \[ x \leq 10 \] \textbf{while} x \leq 10 \textbf{do} x := x + 1 \{ x = 11 \}

3. \[ T \] \textbf{while} x \leq 10 \textbf{do} x := x + 1 \{ x \geq 11 \}

4. \[ [ x = \bar{i} ] \] \textbf{while} x \leq 10 \textbf{do} x := x + 1 \{ ? \}

Any non-terminating program

Any program

Any non-terminating program

Any program
Examples

1. \{x \leq 10\} \textbf{while} x \leq 10 \textbf{ do } x := x + 1 \{x = 11\}

2. \[x \leq 10\] \textbf{while} x \leq 10 \textbf{ do } x := x + 1 \{x = 11\}

3. \[T\] \textbf{while} x \leq 10 \textbf{ do } x := x + 1 \{x \geq 11\}

4. \[x = \bar{i}\] \textbf{while} x \leq 10 \textbf{ do } x := x + 1 \{x = \max(11, \bar{i})\}
Examples

1. \{x \leq 10\} while x <= 10 do x := x + 1 \{x = 11\}
2. \[x \leq 10\] while x <= 10 do x := x + 1 \{x = 11\}
3. \[T\] while x <= 10 do x := x + 1 \{x \geq 11\}
4. \[x = \bar{i}\] while x <= 10 do x := x + 1 \{x = \text{max}(11, \bar{i})\}
5. \{T\} while true do skip \{F\}
Examples

1. \{x \leq 10\} \textbf{while} \; x \leq 10 \; \textbf{do} \; x := x + 1 \{x = 11\}
2. [x \leq 10] \textbf{while} \; x \leq 10 \; \textbf{do} \; x := x + 1 \{x = 11\}
3. [T] \textbf{while} \; x \leq 10 \; \textbf{do} \; x := x + 1 \{x \geq 11\}
4. [x = \bar{i}] \textbf{while} \; x \leq 10 \; \textbf{do} \; x := x + 1 \{x = \max(11, \bar{i})\}
5. \{T\} \textbf{while} \; \text{true} \; \textbf{do} \; \text{skip} \{F\}

- \{\text{any } A}\} \text{any non-terminating program}\{\text{any } B\}
Examples

1. \( \{x \leq 10\} \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{x = 11\} \)

2. \([x \leq 10] \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{x = 11\} \)

3. \([T] \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{x \geq 11\} \)

4. \([x = \bar{i}] \textbf{while } x \leq 10 \textbf{ do } x := x + 1 \{x = \max(11, \bar{i})\} \)

5. \(\{T\} \textbf{while } \text{true } \textbf{do } \text{skip} \{F\} \)
   - \(\{\text{any } A\}\text{any non-terminating program}\{\text{any } B\} \)

6. \(\{F\}\text{any program}\{\text{any } B\} \)
Language of Assertions

- First-order logic with arithmetic:

  arithmetic exps \( a ::= n \mid v \mid \bar{v} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \ast a_2 \)

  assertions \( A ::= T \mid F \mid a_1 = a_2 \mid a_1 \leq a_2 \mid A_1 \land A_2 \)

  \mid A_1 \lor A_2 \mid \neg A \mid A \Rightarrow A_2 \mid \forall \bar{v}.A \mid \exists \bar{v}.A

- **Meta-variables** (\( \bar{v} \)) are **mathematical** variables (not program variables) that have fixed (arbitrary) integer values across all assertions.

- From these one can construct all functions and logical operators, so we will freely use extensions to the above.

  - But if you write something extremely exotic, I reserve the right to challenge you on whether it can actually be expressed using the above.
Hoare Logic

- First published by Tony Hoare [1969]
  - First and most famous axiomatic semantics
  - “An axiomatic basis for computer programming”
  - Often cited as one of the greatest CS papers of all time (only 6 pages long!)
  - Optional: read the original paper (linked from course web site)

- Adaption to SIMPL consists of...
  - six axioms (rules) describing SIMPL programs
  - inference rules of first-order logic
  - axioms of arithmetic (e.g., Peano arithmetic)
Skip Rule

(1) \{A\} \text{skip}\{?\}
Skip Rule

\[ \{A\} \text{skip}\{A\} \]
Sequence Rule

\[
\{A\} c_1 ; c_2 \{B\} \quad (2)
\]
Sequence Rule

\[
\frac{\{A\}c_1\{C\} \quad \{C\}c_2\{B\}}{\{A\}c_1 ; c_2\{B\}} \tag{2}
\]
Conditional Rule

\[ \{ A \} \text{if } b \text{ then } c_1 \text{ else } c_2 \{ B \} \]
Conditional Rule

\[
\frac{\{A\}c_1\{B\}}{\{A\}\text{if } b \text{ then } c_1 \text{ else } c_2\{B\}}^{(3a)}
\]

\[
\frac{\{A\}c_2\{B\}}{\{A\}\text{if } b \text{ then } c_1 \text{ else } c_2\{B\}}^{(3b)}
\]

Problem: These rules can derive false assertions (unsound)!
**Conditional Rule**

\[
\begin{align*}
\{A\} & c_1 \{B\} \\
\{A\} & \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\} \quad (3a) \\
\{A\} & c_2 \{B\} \\
\{A\} & \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\} \quad (3b)
\end{align*}
\]

Problem: These rules can derive false assertions (unsound)!

\[
\begin{align*}
\{T\} & x := 0 \{x = 0\} \\
\{T\} & \text{if } x \leq 0 \text{ then } x := 0 \text{ else skip} \{x = 0\} \quad (3a)
\end{align*}
\]
Conditional Rule

\[
\begin{align*}
\{A\} & c_1 \{B\} & \quad \{A\} & c_2 \{B\} \\
\{A\} & \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\} & \quad (3)
\end{align*}
\]
Conditional Rule

\[
\frac{\{A\} c_1 \{B\} \quad \{A\} c_2 \{B\}}{\{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\}}^{(3)}
\]

Problem: This rule cannot derive some true assertions (incomplete)!

\[
\begin{align*}
\frac{\{T\} x := 0 \{x \geq 0\} \quad \{T\} \text{skip} \{x \geq 0\}}{\{T\} \text{if } x \leq 0 \text{ then } x := 0 \text{ else skip} \{x \geq 0\}^{(3)}
\end{align*}
\]
Conditional Rule

\[
\{ A \land b \} c_1 \{ B \} \quad \{ A \land \neg b \} c_2 \{ B \} \\
\{ A \} \text{if } b \text{ then } c_1 \text{ else } c_2 \{ B \} \tag{3}
\]

Solves completeness problem without sacrificing soundness:

\[
\{ T \land x \leq 0 \} x := 0 \{ x \geq 0 \} \quad \{ T \land \neg (x \leq 0) \} \text{skip} \{ x \geq 0 \} \\
\{ T \} \text{if } x \leq 0 \text{ then } x := 0 \text{ else skip} \{ x \geq 0 \} \tag{3}
\]
Assignment Rule

\[
\{ A \} v := a \{ ? \}^{(4)}
\]
Assignment Rule

\[
\{A\} v := a \{?\}^{(4)}
\]

Usage example:

\[
\{x > 10\} x := x + 1 \{x > 11\}
\]
Assignment Rule

\[ \{A\} v := a\{B\}^{(4)} \]

where \( B = A \) with all \( a \)'s replaced with \( v \)?

Usage example:

\[ \{x > 10\} x := x + 1 \{x > 11\} \]
Assignment Rule

\[ \{A\} v := a \{B\} \quad \text{(4)} \]

where \( B = A \) with all \( a \)'s replaced with \( v \).

Usage example:

\[ \{x > 10\} x := x + 1 \{x > 11\} \]

equivalent \( \iff \)

\[ \{x + 1 > 11\} x := x + 1 \{x > 11\} \]
Assignment Rule

\[
\{ ? \} v := a \{ B \}^{(4)}
\]

Usage example:

\[
\{ x > 10 \} x := x + 1 \{ x > 11 \}
\]

equivalent to

\[
\{ x + 1 > 11 \} x := x + 1 \{ x > 11 \}
\]
Assignment Rule

\[
\{B[a/v]\}v := a\{B\}^{(4)}
\]

Usage example:

\[
\{x > 10\}x := x + 1 \{x > 11\}
\]

equivalent

\[
\{x + 1 > 11\}x := x + 1 \{x > 11\}
\]
While Rule

\{ A \} \textbf{while} b \; \textbf{do} \; c \{ B \}
While Rule

\[
\{A\} \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip} \{B\} \quad (5)
\]

\[
\{A\} \text{while } b \text{ do } c \{B\}
\]
While Rule

$\{A \land b\} c; \text{while } b \text{ do } c \{B\}$

$\{A \land \neg b\} \text{skip}\{B\}$

$\{A\} \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}\{B\}$

$\{A\} \text{while } b \text{ do } c\{B\}$
While Rule

\[
\frac{\{A \land b\}c\{C\} \quad \{C\}\text{while } b \text{ do } c\{B\}}{\{A \land b\}c; \text{while } b \text{ do } c\{B\}} \quad (2)
\]

\[
\frac{\{A\}\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else } \text{skip}\{B\}}{\{A\}\text{while } b \text{ do } c\{B\}} \quad (3)
\]

\[
\frac{\{A\} \quad \{A\} \text{while } b \text{ do } c\{B\}}{\{A\} \quad \{A\} \text{while } b \text{ do } c\{B\}} \quad (5)
\]
While Rule

\[
\begin{align*}
\{A \land b\}c &\{C\} \quad \{C\} &\text{while } b \to c &\{B\} \quad \text{\text{(5)}} \\
\{A \land b\}c; &\text{while } b \to c &\{B\} &\quad \{A \land \neg b\} &\text{skip} &\{B\} \quad \text{\text{(2)}} \\
\{A\} &\text{if } b \text{ then } (c; &\text{while } b \to c) \quad \text{else } &\text{skip} &\{B\} &\quad \text{\text{(3)}} \\
\{A\} &\text{while } b \to c &\{B\} &\quad \text{\text{(5)}}
\end{align*}
\]
While Rule

\{A\} \textbf{while} \ b \ \textbf{do} \ c \{ \ ? \ \}
While Rule

\[
\frac{\{ A \land b \}\{ A \}}{\{ A \}\textbf{ while } b \textbf{ do } c\{ ? \}} \tag{5}
\]
While Rule

\[
\{ A \land b \} \mathbf{c} \{ A \}
\]

\[
\{ A \} \textbf{while } b \textbf{ do } c\{ A \}^{(5)}
\]
While Rule

\[
\{ A \land b \} \text{c}\{ A \} \\
\{A\} \text{while } b \text{ do } c\{ \neg b \land A \}^{(5)}
\]
While Rule

\[
\frac{\{I \land b\} c \{I\}}{\{I\} \textbf{while } b \textbf{ do } c \{\neg b \land I\}} (5)
\]

\( I \) is called a \textbf{loop invariant}
Rule of Consequence

Recall that we earlier needed a way to prove (derive) equivalence of assertions:

\[
\{ x > 10 \}\ x := x + 1 \{ x > 11 \}
\]

\[
\text{equivalent}
\]

\[
\{ x + 1 > 11 \}\ x := x + 1 \{ x > 11 \}
\]
Rule of Consequence

Recall that we earlier needed a way to prove (derive) equivalence of assertions:

\[
\{ x > 10 \} x := x + 1 \{ x > 11 \} \]

equivalent

\[
\{ x + 1 > 11 \} x := x + 1 \{ x > 11 \} \]

Rule of Consequence:

\[
\begin{align*}
\{A'\} & \{B'\} \\
\{A\} & \{B\} \\
\end{align*}
\]

(6)
Rule of Consequence

Recall that we earlier needed a way to prove (derive) equivalence of assertions:

\[
\{ x > 10 \} x := x + 1 \{ x > 11 \}
\]

equivalent

\[
\{ x + 1 > 11 \} x := x + 1 \{ x > 11 \}
\]

Rule of Consequence:

\[
\vdash A \Rightarrow A' \quad \{ A' \} c \{ B' \} \quad \{ A \} c \{ B \} \quad (6)
\]

\[\vdash\] with nothing to the left means implication is **universally true** (i.e., not merely true in this program or loop)

\[\vdash A \Rightarrow A' \quad \text{← Assumptions may be safely weakened}\]
Recall that we earlier needed a way to prove (derive) equivalence of assertions:

\[
\{x > 10\} x := x + 1 \{x > 11\}
\]

equivalent

\[
\{x + 1 > 11\} x := x + 1 \{x > 11\}
\]

Rule of Consequence:

\[
\frac{\models A \Rightarrow A'}{\models \{A\} c \{B\}} \quad \frac{\models B' \Rightarrow B}{\models \{A\} c \{B\}} \quad (6)
\]

\(\models\) with nothing to the left means implication is universally true (i.e., not merely true in this program or loop)

- \(\models A \Rightarrow A'\) ← Assumptions may be safely weakened
- \(\models B' \Rightarrow B\) ← Conclusions (goals) may be safely strengthened
Rule of Consequence Example

\[ \{ x > 10 \} \ x := x + 1 \{ x > 11 \} \]
Rule of Consequence Example

\[
\begin{align*}
\vdash x > 10 \Rightarrow x + 1 > 11 \\
\{x + 1 > 11\} \text{ } x := x + 1 \{x > 11\} \quad \text{(4)} \\
\{x > 10\} \text{ } x := x + 1 \{x > 11\}
\end{align*}
\]

\[
\vdash x > 11 \Rightarrow x > 11 \\
\{x > 10\} \text{ } x := x + 1 \{x > 11\} \quad \text{(6)}
\]
Rule of Consequence Example

\[
\vdots \\
\models x > 10 \Rightarrow x + 1 > 11 \\
\{x + 1 > 11\} x := x + 1 \{x > 11\} \tag{4} \\
\{x > 10\} x := x + 1 \{x > 11\} \\
\models x > 11 \Rightarrow x > 11 \tag{6}
\]

When you write axiomatic derivations in this class:

- You are **not** required to write out the derivations of consequence premises \(\models A\).
- I assume those are derivable using the laws of propositional logic and integer arithmetic.
- But make sure your implications \(X \Rightarrow Y\) are **universally true**!
**Axiomatic Semantics of SIMPL**

1. \[
\{A\} \text{skip} \{A\}
\]

2. \[
\frac{\{A\}c_1 \{C\} \quad \{C\}c_2 \{B\}}{\{A\}c_1 ; c_2 \{B\}}
\]

3. \[
\frac{\{A \land b\}c_1 \{B\} \quad \{A \land \neg b\}c_2 \{B\}}{\{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\}}
\]

4. \[
\frac{}{\{B[a/v]\}v := a\{B\}}
\]

5. \[
\frac{}{\{I \land b\}c\{I\}}
\]

6. \[
\frac{\models A \Rightarrow A' \quad \{A'\}c\{B'\}}{\models B' \Rightarrow B \quad \{A\}c\{B\}}
\]