Axiomatic Semantics
CS 4301/6371: Advanced Programming Languages

Kevin W. Hamlen

April 18, 2024
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. $\{x \leq 10\} \text{while } x \leq 10 \text{ do } x := x + 1 \{x = 11\}$
Examples

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

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

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

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

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

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

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

1. `{x ≤ 10} while x <= 10 do x := x + 1{x = 11}
2. `[x ≤ 10] while x <= 10 do x := x + 1[x = 11]
3. `[T] while x <= 10 do x := x + 1[x ≥ 11]
4. `[x = \overline{i}] while x <= 10 do x := x + 1[x = \max(11, \overline{i})]
5. `{T} while true do skip {F}
   - {any A} any non-terminating program {any B}
6. `{F} any program {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

\[
\{ A \} \text{skip}\{ ? \} 
\]
Skip Rule

\[
\{A\} \text{skip} \{A\} \tag{1}
\]
Sequence Rule

\[ \{A\}c_1 ; c_2 \{B\} \]
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

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

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

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

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

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

\[ \{ A \} c_1 \{ B \} \quad \{ A \} c_2 \{ B \} \]

\[ \frac{\{ A \} \textbf{if} b \textbf{ then } c_1 \textbf{ else } c_2 \{ B \}}{\{ A \} \textbf{if} b \textbf{ then } c_1 \textbf{ else } c_2 \{ B \}} \quad (3) \]
Conditional Rule

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

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

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

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

Solves completeness problem without sacrificing soundness:

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

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

\[ \{ A \} v := a \{ ? \} \]

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

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

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

Usage example:

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

equivalent \[\uparrow\]

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

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

Usage example:

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

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

\[\equiv\]

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

\[
\{B[a/v]\}v := a\{B\}
\]

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

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

\[
\begin{align*}
\{A \land b\} c; \textbf{while } b \textbf{ do } c \{B\} \quad &\quad \{A \land \neg b\} \textbf{skip} \{B\} \\
\{A\} \textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else } \textbf{skip} \{B\} \quad &\quad \{A\} \textbf{while } b \textbf{ do } c \{B\}
\end{align*}
\]

(3)
While Rule

\[
\begin{align*}
{A \land b} & \vdash c \{C\} \quad {C} \vdash \text{while } b \text{ do } c \{B\} \quad (2) \\
{A \land b} & \vdash c_1 ; \text{while } b \text{ do } c_2 \{B\} \quad \{A \land \neg b\} \vdash \text{skip} \{B\} \quad (3) \\
\{A\} & \vdash \text{if } b \text{ then } (c_1 ; \text{while } b \text{ do } c_2) \text{ else skip} \{B\} \quad (5) \\
\{A\} & \vdash \text{while } b \text{ do } c \{B\} 
\end{align*}
\]
While Rule

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

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

\[
\begin{align*}
\{ A \land b \} & c \{ A \} \\
\{ A \} & \textbf{while } b \textbf{ do } c \{ \neg b \land A \} \tag{5}
\end{align*}
\]
While Rule

\[
\{ I \land b \} \text{ } c \{ I \} \\
\{ I \} \text{while } b \text{ do } c \{ \neg b \land I \} \tag{5}
\]

\( I \) 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:

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

\[
\begin{align*}
| & \Rightarrow A \Rightarrow A' \quad \{ A' \} \ c \{ B' \} \\
\{ A \} \ c \{ B \} & \quad (6)
\end{align*}
\]

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

■ \ |= A \Rightarrow A' \quad \text{← Assumptions may be safely weakened}
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:

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

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

- \(\models A \Rightarrow A'\) \hspace{1em} ← Assumptions may be safely \textbf{weakened}

- \(\models B' \Rightarrow B\) \hspace{1em} ← Conclusions (goals) may be safely \textbf{strengthened}
Rule of Consequence Example

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

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

\[
\begin{align*}
\vdots & & \\
\models x > 10 \Rightarrow x + 1 > 11 & & \{x + 1 > 11\} \colon= x + 1\{x > 11\} & (4) & & \vdots \\
\{x > 10\} \colon= x + 1\{x > 11\} & & \models x > 11 \Rightarrow x > 11 & (6)
\end{align*}
\]

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

5.\[
\frac{\{I \land b\}c\{I\}}{\{I\text{while }b\text{ do }c\{\neg b \land I\}}}
\]

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