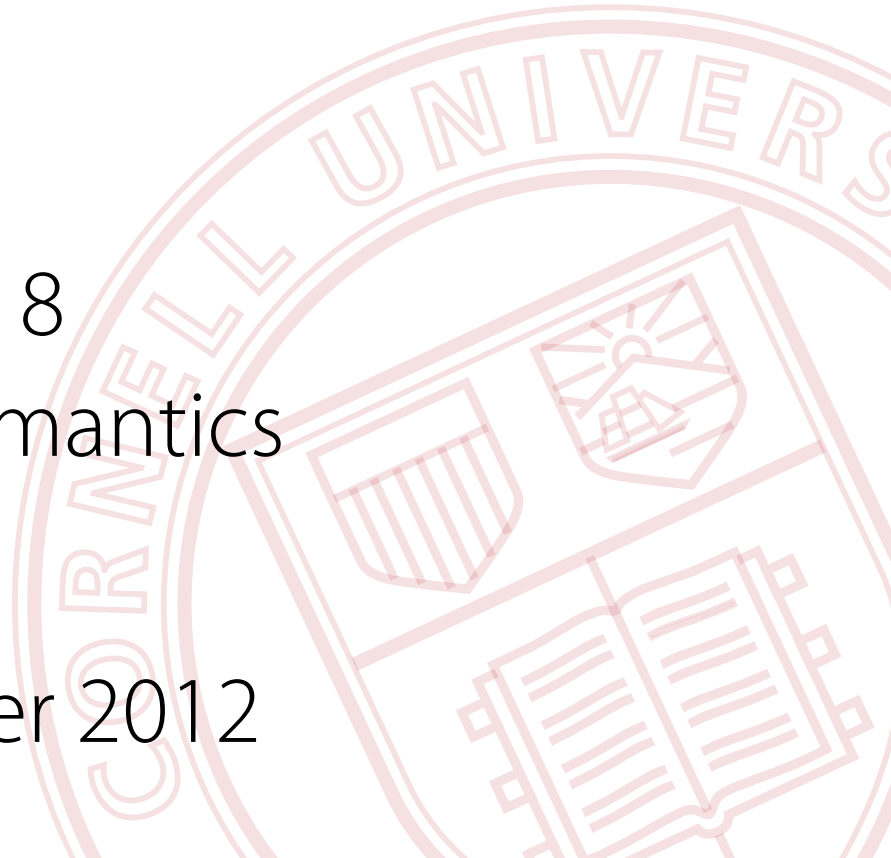
# CS 4110

# Programming Languages & Logics



## Lecture 8
## Axiomatic Semantics

14 September 2012

# Announcements

- Homework #3 due Monday at 11:59pm
- Foster office hours Monday 4-5pm in Upson 4137
- Rajkumar office hours Monday 5-6pm in 4135
- Around-the-clock help available on Piazza

# Review

## Operational Semantics

- Describes *how* programs compute
- Relatively easy to define
- Close connection to implementations

# Review

## Operational Semantics

- Describes *how* programs compute
- Relatively easy to define
- Close connection to implementations

## Denotational Semantics

- Describes *what* programs compute
- Solid mathematical foundation
- Simplifies many kinds of reasoning

# Review

## Operational Semantics

- Describes *how* programs compute
- Relatively easy to define
- Close connection to implementations

## Denotational Semantics

- Describes *what* programs compute
- Solid mathematical foundation
- Simplifies many kinds of reasoning

## Axiomatic Semantics

- A framework for reasoning about correctness
- History: Pioneered by Floyd & Hoare, refined by Djikstra & Gries

# Axiomatic Semantics

To define an axiomatic semantics we need:

- A language for expressing assertions
- Rules for establishing the validity of particular assertions with respect to specific programs

# Axiomatic Semantics

To define an axiomatic semantics we need:

- A language for expressing assertions
- Rules for establishing the validity of particular assertions with respect to specific programs

Assertions:

- The values of *x* and *y* are equal
- The values in a list *l* are sorted
- The program terminates

# Axiomatic Semantics

To define an axiomatic semantics we need:

- A language for expressing assertions
- Rules for establishing the validity of particular assertions with respect to specific programs

Assertions:

- The values of *x* and *y* are equal
- The values in a list *l* are sorted
- The program terminates

Assertion Languages:

- First-order logic: $\forall, \exists, \wedge, \vee, x = y, R(x), \ldots$
- Temporal or modal logic: $\Box, \diamond, \phi, \ldots$
- Special-purpose specification languages: Alloy, Z3, Sugar, etc.

# Applications

- Proving correctness
- Documentation
- Test generation
- Symbolic execution
- Translation validation
- Bug finding
- Malware detection

# Pre-Conditions and Post-conditions

Assertions often used (informally) in code

```
// Precondition: 0 <= i < A.length
// Postcondition: returns ith element of A
public int get(int i) {
    return A[i];
}
```

Very useful as documentation, but no guarantee they are correct.

Idea: make this rigorous by defining the semantics of the language in terms of pre-conditions and post-conditions!

# Partial Correctness

Recall the syntax of IMP:

$$
\begin{array}{lll}
a \in \textbf{Aexp} & a ::= & x \mid n \mid a_1 + a_2 \mid a_1 \times a_2 \\
b \in \textbf{Bexp} & b ::= & \textbf{true} \mid \textbf{false} \mid a_1 < a_2 \\
c \in \textbf{Com} & c ::= & \textbf{skip} \mid x := a \mid c_1 ; c_2 \\
 & & \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } b \textbf{ do } c
\end{array}
$$

A partial correctness statement is a triple:

$$\{P\}\ c\ \{Q\}$$

Meaning: If $P$ holds and executing $c$ terminates, then $Q$ holds after $c$

# Total Correctness

Note that partial correctness specifications don't ensure that the program will terminate—this is why they are called "partial"

Sometimes we need to know that the program will terminate

A total correctness statement is a triple:

$$[\,P\,]\;c\;[\,Q\,]$$

Meaning: if $P$ holds, then $c$ will terminate and $Q$ holds after $c$

We'll focus mostly on partial correctness.

# Example: Partial Correctness

$\{\text{foo} = 0 \wedge \text{bar} = i\}$

baz := 0;
**while** foo $\neq$ bar
**do**
    baz := baz $-$ 2;
    foo := foo $+$ 1

$\{\text{baz} = -2i\}$

Intuition: if we start with a store $\sigma$ that maps foo to 0 and bar to an integer $i$, and if the execution of the command terminates, then the final store $\sigma'$ will map baz to $-2i$

# Example: Total Correctness

$[\text{foo} = 0 \wedge \text{bar} = i \wedge i \geq 0]$

baz := 0;
**while** foo $\neq$ bar
**do**
    baz := baz − 2;
    foo := foo + 1

$[\text{baz} = -2i]$

Intuition: if we start with a store $\sigma$ that maps foo to 0 and bar to a non-negative integer $i$, then the execution of the command will terminate in a final store $\sigma'$ will map baz to $-2i$

# Another Example

$\{\text{foo} = 0 \wedge \text{bar} = i\}$

baz := 0;
**while** foo $\neq$ bar
**do**
    baz := baz + foo;
    foo := foo + 1

$\{\text{baz} = i\}$

Question: is this partial correctness statement valid?

# Assertions

We'll use the following language to write assertions:

$$i, j \in \textbf{LVar}$$

$$a \in \textbf{Aexp} ::= x \mid i \mid n \mid a_1 + a_2 \mid a_1 \times a_2$$

$$\begin{aligned} P, Q \in \textbf{Assn} ::= \; & \textbf{true} \mid \textbf{false} \\ \mid \; & a_1 < a_2 \\ \mid \; & P_1 \wedge P_2 \mid P_1 \vee P_2 \mid P_1 \Rightarrow P_2 \\ \mid \; & \neg P \mid \forall i.\, P \mid \exists i.\, P \end{aligned}$$

Note that every boolean expression $b$ is also an assertion.

# Satisfaction

Now we want to define what it means for a store $\sigma$ to satisfy an assertion

But before we can do this, we need an interpretion for the logical variables

$$I : \mathbf{LVar} \to \mathbf{Int},$$

# Satisfaction

Now we want to define what it means for a store $\sigma$ to satisfy an assertion

But before we can do this, we need an interpretion for the logical variables

$$I : \mathbf{LVar} \to \mathbf{Int},$$

$$\begin{aligned}
\mathcal{A}_i[\![n]\!](\sigma, I) &= n \\
\mathcal{A}_i[\![x]\!](\sigma, I) &= \sigma(x) \\
\mathcal{A}_i[\![i]\!](\sigma, I) &= I(i) \\
\mathcal{A}_i[\![a_1 + a_2]\!](\sigma, I) &= \mathcal{A}_i[\![a_1]\!](\sigma, I) + \mathcal{A}_i[\![a_2]\!](\sigma, I)
\end{aligned}$$

# Satisfaction

Next we define the satisfaction relation for assertions

## Definition (Assertation satisfaction)

| | |
|---|---|
| $\sigma \models_I \mathbf{true}$ | (always) |
| $\sigma \models_I a_1 < a_2$ | if $\mathcal{A}_i[\![a_1]\!](\sigma, I) < \mathcal{A}_i[\![a_2]\!](\sigma, I)$ |
| $\sigma \models_I P_1 \wedge P_2$ | if $\sigma \models_I P_1$ and $\sigma \models_I P_2$ |
| $\sigma \models_I P_1 \vee P_2$ | if $\sigma \models_I P_1$ or $\sigma \models_I P_2$ |
| $\sigma \models_I P_1 \Rightarrow P_2$ | if $\sigma \not\models_I P_1$ or $\sigma \models_I P_2$ |
| $\sigma \models_I \neg P$ | if $\sigma \not\models_I P$ |
| $\sigma \models_I \forall i.\, P$ | if $\forall k \in \mathit{Int}.\, \sigma \models_{I[i \mapsto k]} P$ |
| $\sigma \models_I \exists i.\, P$ | if $\exists k \in \mathit{Int}.\, \sigma \models_{I[i \mapsto k]} P$ |

# Satisfaction

Next we define what it means for a command $c$ to satisfy a partial correctness statement.

## Definition (Partial correctness statement satisfiability)

A partial correctness statement $\{P\}\ c\ \{Q\}$ is satisfied in store $\sigma$ and interpretation $I$, written $\sigma \vDash_I \{P\}\ c\ \{Q\}$, if:

$$\forall \sigma'.\ \text{if } \sigma \vDash_I P \text{ and } \mathcal{C}[\![c]\!]\sigma = \sigma' \text{ then } \sigma' \vDash_I Q$$

# Validity

## Definition (Assertion validity)

An assertion $P$ is valid (written $\vDash P$) if it is valid in any store, under any interpretation: $\forall \sigma, I.\ \sigma \vDash_I P$

## Definition (Partial correctness statement validity)

A partial correctness triple is valid (written $\vDash \{P\}\ c\ \{Q\}$), if it is valid in any store and interpretation: $\forall \sigma, I.\ \sigma \vDash_I \{P\}\ c\ \{Q\}$.

Now we know what we mean when we say "assertion $P$ holds" or "partial correctness statement $\{P\}\ c\ \{Q\}$ is valid."

# Proving Specifications

How do we show that $\{P\}\ c\ \{Q\}$ holds?

We know that $\{P\}\ c\ \{Q\}$ is valid if it holds for all stores and interpretations: $\forall \sigma, I.\ \sigma \vDash_I \{P\}\ c\ \{Q\}$.

Furthermore, showing that $\sigma \vDash_I \{P\}\ c\ \{Q\}$ requires reasoning about the denotation of $c$, as specified by the definition of satisfaction.

We can do this manually, but it turns out that there is a better way.

We can use a set of inference rules and axioms, called *Hoare rules*, to directly derive valid partial correctness statements without having to reason about stores, interpretations, and the execution of $c$.