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: $\forall, \exists, \land, \lor, x = y, R(x), \ldots$
- 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

```java
// 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:

\[
\begin{align*}
    a & \in \mathbf{Aexp} & a ::= x \mid n \mid a_1 + a_2 \mid a_1 \times a_2 \\
    b & \in \mathbf{Bexp} & b ::= \text{true} \mid \text{false} \mid a_1 < a_2 \\
    c & \in \mathbf{Com} & c ::= \text{skip} \mid x ::= a \mid c_1 ; c_2 \\
    & & \quad | \quad \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c
\end{align*}
\]

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

```
{foo = 0 ∧ bar = i}
baz := 0;
while foo ≠ bar
  do
    baz := baz − 2;
    foo := foo + 1
  {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

[foo = 0 ∧ bar = i ∧ i ≥ 0]
baz := 0;
while foo ≠ bar
do
    baz := baz − 2;
    foo := foo + 1
[baz = −2i]

Intuition: if we start with a store σ 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 σ′ will map baz to −2i
Another Example

{foo = 0 \land bar = i}
baz := 0;
while foo \neq bar
  do
    baz := baz + foo;
    foo := foo + 1
{baz = i}

Question: is this partial correctness statement valid?
Assertions

We’ll use the following language to write assertions:

\[ i, j \in \text{LVar} \]

\[ a \in \text{Aexp} ::= x \mid i \mid n \mid a_1 + a_2 \mid a_1 \times a_2 \]

\[ P, Q \in \text{Assn} ::= \text{true} \mid \text{false} \]

\[ \mid a_1 < a_2 \]

\[ \mid P_1 \land P_2 \mid P_1 \lor P_2 \mid P_1 \Rightarrow P_2 \]

\[ \mid \neg P \mid \forall i. \ P \mid \exists i. \ P \]

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 interpretation for the logical variables:

$$\ell : \text{LVar} \rightarrow \text{Int},$$
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 interpretation for the logical variables:

\[ l : \text{LVar} \rightarrow \text{Int}, \]

\[ A_i[n](\sigma, l) = n \]
\[ A_i[x](\sigma, l) = \sigma(x) \]
\[ A_i[i](\sigma, l) = l(i) \]
\[ A_i[a_1 + a_2](\sigma, l) = A_i[a_1](\sigma, l) + A_i[a_2](\sigma, l) \]
Next we define the satisfaction relation for assertions

**Definition (Assertion satisfaction)**

<table>
<thead>
<tr>
<th>Logical Form</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma \models_I \text{true}$</td>
<td>(always)</td>
</tr>
<tr>
<td>$\sigma \models_I a_1 &lt; a_2$</td>
<td>if $A_i[a_1](\sigma, l) &lt; A_i[a_2](\sigma, l)$</td>
</tr>
<tr>
<td>$\sigma \models_I P_1 \land P_2$</td>
<td>if $\sigma \models_I P_1$ and $\sigma \models_I P_2$</td>
</tr>
<tr>
<td>$\sigma \models_I P_1 \lor P_2$</td>
<td>if $\sigma \not\models_I P_1$ or $\sigma \models_I P_2$</td>
</tr>
<tr>
<td>$\sigma \models_I P_1 \rightarrow P_2$</td>
<td>if $\sigma \not\models_I P_1$ or $\sigma \models_I P_2$</td>
</tr>
<tr>
<td>$\sigma \models_I \neg P$</td>
<td>if $\sigma \not\models_I P$</td>
</tr>
<tr>
<td>$\sigma \models_I \forall i. P$</td>
<td>if $\forall k \in \text{Int. } \sigma \models_{[i \mapsto k]} P$</td>
</tr>
<tr>
<td>$\sigma \models_I \exists i. P$</td>
<td>if $\exists k \in \text{Int. } \sigma \models_{[i \mapsto k]} P$</td>
</tr>
</tbody>
</table>
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 $\sigma \models_I \{P\} \ c \ \{Q\}$, if:

$$\forall \sigma'. \text{ if } \sigma \models_I P \text{ and } C[c]\sigma = \sigma' \text{ then } \sigma' \models_I Q$$
Validity

Definition (Assertion validity)
An assertion $P$ is valid (written $\models P$) if it is valid in any store, under any interpretation: $\forall \sigma, I. \sigma \models I P$

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

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: $\forall \sigma, I. \ \sigma \models_I \ \{P\} \ c \ \{Q\}$.

Furthermore, showing that $\sigma \models_I \ \{P\} \ c \ \{Q\}$ 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 $c$. 