CS655 Lecture 6 — Intro To Axiomatic Semantics — 7 February, 2006

Scribe: Daniel Dunbar — Lecturer: Wes Weimer

1 Introduction

The kinds of semantics covered so far in class have been

- **Operational Semantics.** For describing the semantics in terms of effects on a state (operations). This style of semantics is useful for translating into an interpeter or explaining operations intuitively.
- **Denotation Semantics.** A mathematical modeling of the semantics as state transition functions. Denotation semantics have the nice property of being composable but require more sophisticated mathematical machinery such as fixed point operators, and are less useful in practice.

These semantics have been used thus far for conducting proofs about the determinism of the language **IMP** or determining equivalence between statements, but not for reasoning about the meaning of a specific program (i.e. this code computes the least common denominator of an integer). In fact, it turns out that there is another semantic style specifically developed for this task by Hoare and Floyd. **Axiomatics semantics** is designed to reasons about a program using basic *axioms* to determine the *semantics* of a program (in terms of what is true after the program executes).

As Dijkstra noted, "program testing can be used to show the presence of bugs, but not their absence". Axiomatic semantics is the product of attempts to solve this problem by using formal methods to prove a program's correctness with respect to a (mathematical, logical) specification. The goals are to be able to prove the correctness of an algorithm or hardware implementation (or to find bugs in the same) by supplying a proof in axiomatic semantics.

However, despite proof techniques being in existence almost as long as programming itself, there has not been substantial progress towards being able to prove significant facts about real-world programs. In practice, this means that proving has not replaced testing or debugging in software development. Nevertheless, axiomatic semantics has still found applications in program analysis, for example to elimiante array bounds checking or for proof-carrying code.

2 Assertions

Axiomatic semantics consists of a language for making assertions about a program (and judging them!) and derivation rules for constructing proofs using those assertions. Assertions can be modeled in a variety of logics. First-order logic using quantifiers is the most common example, but other choices are available. Temporal logic might be used in order to express assertions such as "program P will at some time be deadlocked". In addition, program analysis tool like SLAM may define their own specification language such as *SLIC*.

Assertions take the form

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

which the following interpretation: if the *precondition* A holds and the command c is evaluated then the *postcondition* B will hold. These forms are called *Hoare triples* or *Hoare assertions*. The notation $\sigma \models A$ is a *judgment* that the assertion A holds (is true) in state σ. As stated the command c may diverge and this is called a *partial correctness assertion* and it is a valid assertion even if c does not terminate. The notation $[A]c[B]$ represents a total correctness assertion meaning that the assertion holds and the command does not diverge.

The formal meaning of a partial correctness assertion $\{A\}c\{B\}$ is

$$\forall_{\sigma\in\Sigma}\forall_{\sigma'\in\Sigma}(\sigma \models A \wedge \langle c, \sigma\rangle \Downarrow \sigma') \Rightarrow \sigma' \models B.$$

For a total correctness assertion $[A]c[B]$ then the formal meaning is

$$\{A\}b\{C\} \wedge \forall_{\sigma\in\Sigma}\exists_{\sigma'\in\Sigma}\ \langle c, \sigma\rangle \Downarrow \sigma.$$

2.1 Assertions in IMP

The language of assertions for **IMP** consists of first-order predicate logic along with normal expressions:

$$\begin{aligned} A ::= \quad & \mathit{true} \mid \mathit{false} \mid e_1 = e_2 \mid e_1 \geq e_2 \mid A_1 \wedge A_2 \\ & \mid A_1 \vee A_2 \mid A_1 \Rightarrow A_2 \mid \forall x.A \mid \exists x.A \end{aligned}$$

where, informally, the logical and program variables are taken to be distinct (this is a straightforward formalization, see Winskel Chapter 6).

The truth of the assertions (i.e. $\sigma \models A$) is defined inductively over the structure of assertions by the following rules:

$\sigma \models \mathit{true}$	always
$\sigma \models e_1 = e_2$	iff $[\![e_1]\!]\sigma = [\![e_2]\!]\sigma$
$\sigma \models e_1 \geq e_2$	iff $[\![e_1]\!]\sigma \geq [\![e_2]\!]\sigma$
$\sigma \models A_1 \wedge A_2$	iff $\sigma \models A_1$ and $\sigma \models A_2$
$\sigma \models A_1 \vee A_2$	iff $\sigma \models A_1$ or $\sigma \models A_2$
$\sigma \models A_1 \Rightarrow A_2$	iff $\sigma \models A_1$ implies $\sigma \models A_2$
$\sigma \models \forall x.A$	iff $\forall n \in \mathbb{Z}.\sigma[x := n] \models A$
$\sigma \models \exists x.A$	iff $\exists n \in \mathbb{Z}.\sigma[x := n] \models A$

3 Derivation Rules

In order to use these rules to construct proofs there are a set of derivation rules for working with assertions. The notation $\vdash A$ is used to mean that assertion A can be proved from the basic axioms and the given derivation rules. For **IMP** this means that the rules are those from first-order logic:

$$\frac{\vdash A \quad \vdash B}{\vdash A \wedge B} \qquad \frac{A \vdash B}{A \Rightarrow B} \qquad \frac{\vdash A \Rightarrow B \quad \vdash A}{\vdash B} \qquad \frac{\vdash [a/X]A \quad \textit{where a is fresh}}{\vdash \forall_x A}$$

$$\frac{\vdash \forall_x A}{[e/x]A} \qquad \frac{\vdash [e/x]A}{\exists_x A} \qquad \frac{\vdash \exists_x A \quad [a/x]A \vdash B}{\vdash B}$$

The notation $\vdash \{A\}c\{B\}$ is used whenever it is possible to derive the triple using derivation rules. These rules are known as (Floyd-) Hoare logic and there is one rule for each command in the language in addition to the rule of consequence:

$$\frac{}{\vdash \{A\}\mathsf{skip}\{A\}} \qquad \frac{}{\vdash \{[e/x]A\}x := e\{A\}} \qquad \frac{\vdash \{A\}c_1\{B\} \quad \vdash \{B\}c_2\{C\}}{\vdash \{A\}c_1;\ c_2\{C\}}$$

$$\frac{\vdash \{A \wedge b\}c_1\{B\} \quad \vdash \{A \wedge \neg b\}c_2\{B\}}{\vdash \{A\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\{B\}} \qquad \frac{\vdash \{A \wedge b\}c\{A\}}{\vdash \{A\}\ \mathsf{while}\ b\ \mathsf{do}\ c\{A \wedge \neg b\}}$$

$$\frac{\vdash A' \Rightarrow A \quad \vdash \{A\}c\{B\} \quad \vdash B \Rightarrow B'}{\vdash \{A'\}c\{B'\}} \text{ rule of consequence}$$

Some rules have alternate forms, for example the forward axiom for assignment

$$\frac{}{\vdash \{A\}x := e\{\exists_{x_0}[x_o/x]A \wedge x = [x_0/x]e\}}$$

or the rule for a while loop with the loop invariant explicit:

$$\frac{\vdash A \wedge b \Rightarrow C \quad \vdash \{C\}c\{A\} \quad \vdash A \wedge \neg b \Rightarrow B}{\vdash \{A\}\mathsf{while}\ b\ \mathsf{do}\ c\{B\}}$$

3.1 Examples

For example, suppose that x does not appear in e. By the assignment rule then

$$\overline{\{e = e\}x := e\{x = e\}}$$

because $[e/x](x = e) \rightarrow e = e$. Using this rule and consequence gives the derivation

$$\frac{\vdash \mathit{true} \Rightarrow e = e \quad \overline{\{e = e\}x := e\{x = e\}}}{\{\mathit{true}\}x := e\{x = e\}}$$

Note that for languages in which aliasing may be an issue then this form of the assignment rule requires revision.

The following example shows the application of the conditional rule:

$$\frac{D_1 :: \vdash \{\mathit{true} \wedge y \leq 0\}x := 1\{x > 0\} \quad D_2 :: \vdash \{\mathit{true} \wedge y > 0\}x := y\{x > 0\}}{\{\mathit{true}\}\textsf{if } y \leq 0 \textsf{ then } x := 1 \textsf{ else } x := y\ \{x > 0\}}$$

where D_1 and D_2 are obtained by consequence and assignment, for example D_1 has the form:

$$\frac{\vdash \{1 > 0\}x := 1\{x > 0\} \quad \vdash \mathit{true} \wedge y \leq 0 \Rightarrow 1 > 0}{\vdash \{\mathit{true} \wedge y \leq 0\}x := 1\{x > 0\}}$$

Finally, suppose that we wish to show that

$$\vdash \{x \leq 0\}\ \textsf{while } x \leq 5 \textsf{ do } x := x + 1\{x = 6\}.$$

Using the rule for while with the invariant $x \leq 6$ we obtain

$$D_1 :: \quad \frac{\dfrac{\vdash x \leq 6 \wedge x \leq 5 \Rightarrow x + 1 \leq 6 \quad \overline{\vdash \{x + 1 \leq 6\}x := x + 1\{x \leq 6\}}}{\vdash \{x \leq 6 \wedge x \leq 5\}x := x + 1\{x \leq 6\}}}{\vdash \{x \leq 6\}\ \textsf{while } x \leq 5 \textsf{ do } x := x + 1\{x \leq 6 \wedge x\rangle 5\}}$$

and apply the rule of consequence to conclude

$$\frac{\vdash x \leq 0 \Rightarrow x \leq 6 \quad \vdash x \leq 6 \wedge x\rangle 5 \Rightarrow x = 6 \quad D_1}{\vdash \{x \leq 0\}\ \textsf{while } x \leq 5 \textsf{ do } x := x + 1\{x = 6\}}$$

4 Using Hoare Rules

The Hoare rules are mostly syntax directed but there are three important questions that come up when doing derivations:

1. *What is the invariant for while?*

   This question can be approached using fixed point theory or widening, but this is the hardest problem for automatic application. In practice the current approach is to require the program to specify the invariants explicitly.

2. *When to apply consequence?*

3. *How to prove the first-order logic implication used by consequence?*

   In practice both of these problems can be solved automatically by a theorem prover.

5 Summary

Axiomatic semantics is a formal way of specifying and deriving assertions about a program. By introducting first-order logic into the language for assertions it is possible to write down a proof list for practical assertions that are not possible using only operation or denotational semantics.