CS611 Lecture 14 **Axiomatic Semantics** 29 September 2003

Scribe: Karen McKeen, Jonathan Moon Lecturer: Radu Rugina

# 1 Axiomatic Semantics

The goal of axiomatic semantics is to prove properties of programs. We will do this in terms of preconditions $\{P\}$ and postconditions $\{Q\}$. These conditions are referred to as assertions.

## 1.1 Example

$$\begin{array}{l} \textbf{while } \neg(X = 0) \textbf{ do} \\ \quad Y := Y + 2; \\ \quad X := X - 1 \end{array}$$

If this program is considered to be the command $c$ then $P' \equiv (X = n) \wedge (Y = 0)$ is a possible precondition for $c$ and $Q' \equiv \; Y = 2 * n$ is a possible postcondition.

## 1.2 Types of Assertions

A *partial correctness assertion* (PCA) has the form $\{P\}\ c\ \{Q\}$ and means that if $P$ holds prior to the execution of $c$, and $c$ terminates, then $Q$ holds after termination. However, a PCA does not guarantee that $c$ terminates. In the above example, we have $\{P'\}\ c\ \{Q'\}$

In contrast, a *total correctness assertion*, denoted $[P]\ c\ [Q]$, does guarantee that c terminates, in addition to $P$ holding prior to execution and $Q$ holding afterwards. In the above example, if we changed $P'$ to $(X = n) \wedge (n \geq 0) \wedge (Y = 0)$, then $[P]\ c\ [Q]$.

## 1.3 Definition of Assertions

**Assn** is the domain of assertions. Let $a_0$ and $a_1$ be arithmetic expressions. Assertions are defined recursively by the following grammar:

$$A ::= \textbf{true} \;|\; \textbf{false} \;|\; a_0 = a_1 \;|\; a_0 \leq a_1 \;|\; A_0 \wedge A_1 \;|\; A_0 \vee A_1 \;|\; \neg A_0 \;|\; A_0 \Rightarrow A_1 \;|\; \forall i.\ A \;|\; \exists i.\ A$$

Note: We need $\forall$ and $\exists$ to be able to describe conditions such as divisibility ($\exists i.\ m = i * n$) and test for prime numbers ($\forall i.\ [i \leq 1 \vee i \geq p \vee (\forall j.\ p \neq i * j)]$).

## 1.4 Alternate Definition of Assertions

Equivalently, assertions can be represented by the following grammar:

$$A ::= \textbf{false} \;|\; a_0 \leq a_1 \;|\; A_0 \Rightarrow A_1 \;|\; \forall i.\ A$$

The remaining parts of the first definition can be represented as follows:

- $\neg A_0 \equiv A_0 \Rightarrow \textbf{false}$
- $a_0 = a_1 \equiv (a_0 \leq a_1) \wedge (a_1 \leq a_0)$
- $\textbf{true} \equiv \neg\textbf{false}$
- $A_0 \vee A_1 \equiv \neg A_0 \Rightarrow A_1$
- $A_0 \wedge A_1 \equiv \neg(\neg A_0 \vee \neg A_1)$

- $\exists i.\ A \equiv \neg \forall i.\ \neg A$

We will also allow in assertions the inclusion of variables not found in the program. The domain of these variables is **Intvar**, which is distinct from **Loc**.

The domain of arithmetic expressions in assertions is **Aexpv**, and is defined recursively for $a,\ a_0,\ a_1 \in$ **Aexpv**, $X \in$ **Loc**, $n \in \mathbb{Z}$, $i \in$ **Intvar** by the following:

$$a ::= X \ \mid\ n \ \mid\ i \ \mid\ a_0 \oplus a_1 \qquad (\oplus \in \{+, -, *\})$$

We also define $FV(A)$ as the set of free variables (variables not quantified existentially or universally in $A$) of an assertion $A$. A formal definition by structural induction is offered in the Winskel text on page 81.

Substitution of an arithmetic expression $a$ for an integer variable $i$ (or a location $X$) is also defined for assertions, and is denoted $A[i \mapsto a]$ (or $A[X \mapsto a]$). A formal definition can be found in the Winksel text on page 82.

### 1.5 Semantics of Assertions

Before we can evaluate the validity of an assertion, we need to have values for both the variables used in the program and those introduced in the assertion. Those from the program are contained in the state, $\sigma$. The other variables come from the evaluation of an *interpretation function* $I$. This function assigns an integer to each integer variable:

$$I : \mathbf{Intvar} \to \mathbb{Z}$$

Then the evaluation function of an assertion expression is $\mathcal{A}v[\![a]\!] : \Sigma \to (\mathbf{Intvar} \to \mathbb{Z}) \to \mathbb{Z}$ and is defined as:

$$\begin{array}{rcl} \mathcal{A}v[\![X]\!]\sigma I & = & \sigma(X) \\ \mathcal{A}v[\![n]\!]\sigma I & = & n \\ \mathcal{A}v[\![i]\!]\sigma I & = & I(i) \\ \mathcal{A}v[\![a_0 \oplus a_1]\!]\sigma I & = & \mathcal{A}v[\![a_0]\!]\sigma I \oplus \mathcal{A}v[\![a_1]\!]\sigma I \qquad (\oplus \in \{+, -, *\}) \end{array}$$

### 1.6 Validity of Assertions

To say that an assertion $A$ is valid in state $\sigma$ under interpretation $I$ we write:

$$\sigma \models^I A$$

We define this relation inductively on the structure of $A$. We use the simpler grammar for assertions, which consists of **false**, $\leq$, $\Rightarrow$, and $\forall$:

$$\begin{array}{lcll} \sigma & \models^I & \mathbf{false} & \text{never holds} \\ \sigma & \models^I & A_0 \Rightarrow A_1 & \text{if } \neg(\sigma \models^I A_0) \vee (\sigma \models^I A_1) \\ \sigma & \models^I & \forall i\ .\ A & \text{if } \forall n \in \mathbb{N}\ .\ \sigma \models^{I[i \mapsto n]} A \\ \sigma & \models^I & a_0 \leq a_1 & \text{if } \mathcal{A}v[\![a_0]\!]\sigma I \leq \mathcal{A}v[\![a_1]\!]\sigma I \end{array}$$

And finally, for all $A$ and $I$,

$$\bot \ \ \models^I A$$

Notationally, we say $\sigma \not\models^I A$ if $\sigma \models^I A$ doesn't hold.

### 1.7 Results

The following results follow from the definition of the validity relation and from the semantics of expressions. The proofs involve a straightforward structural induction.

1. $\mathcal{A}$ and $\mathcal{A}v$ agree on expressions in **Aexp**.

$$\forall a \in \mathbf{Aexp}.\ \forall \sigma, I.\quad \mathcal{A}[\![a]\!]\sigma \ = \ \mathcal{A}v[\![a]\!]\sigma I$$

2. The denotational semantics of boolean expressions agrees with the validity relations on assertions.

$$\begin{array}{l} \forall b \in \mathbf{Bexp}.\ \forall \sigma, I.\quad \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \iff \ \sigma \models^I b \\ \forall b \in \mathbf{Bexp}.\ \forall \sigma, I.\quad \mathcal{B}[\![b]\!]\sigma = \mathbf{false} \iff \ \sigma \not\models^I b \end{array}$$

### 1.8 Validity of Partial Correctness Assertions

We define the validity of a PCA as:

$$\sigma \models^I \{P\}\ c\ \{Q\} \overset{def}{\iff} [(\sigma \models^I P \wedge \mathcal{C}[\![c]\!]\sigma \neq \perp) \Rightarrow (\mathcal{C}[\![c]\!]\sigma \models^I Q)]$$

But if $\mathcal{C}[\![c]\!]\sigma = \perp$ then $Q$ is trivially valid, so we can rewrite the antecedent as:

$$\sigma \models^I \{P\}\ c\ \{Q\} \iff [(\sigma \models^I P) \Rightarrow (\mathcal{C}[\![c]\!]\sigma \models^I Q)]$$

We can eliminate I, then $\sigma$ from this definition:

$$\begin{array}{rcl} \sigma \models \{P\}\ c\ \{Q\} & \overset{def}{\iff} & \forall I.\ \ \sigma \models^I \{P\}\ c\ \{Q\} \\ \models \{P\}\ c\ \{Q\} & \overset{def}{\iff} & \forall \sigma.\ \ \sigma \models \{P\}\ c\ \{Q\} \end{array}$$

When $\models \{P\}\ c\ \{Q\}$ we say that the PCA is *valid*.

How can we check a PCA for validity? We could compute the state resulting from the execution of the command $c$ using denotational (or operational) semantics. A simpler method involves the use of Hoare Logic.

## 2 Hoare Logic

Hoare logic ia a proof system, that is, a collection of syntax-directed proof rules that generates partial correctness assertions.

$$\frac{}{\{A\}\ \mathbf{skip}\ \{A\}}$$

$$\frac{}{\{A[X \mapsto a]\}\ X := a\ \{A\}}$$

$$\frac{\{A\}\ c_0\ \{C\} \quad \{C\}\ c_1\ \{B\}}{\{A\}\ c_0; c_1\ \{B\}}$$

$$\frac{\{A \wedge b\}\ c_0\ \{B\} \quad \{A \wedge \neg b\}\ c_1\ \{B\}}{\{A\}\ \mathbf{if}\, b\, \mathbf{then}\, c_0\ \mathbf{else}\, c_1\ \{B\}}$$

$$\frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \mathbf{while}\, b\, \mathbf{do}\, c\ \{A \wedge \neg b\}}$$

$$\frac{A \Rightarrow A' \quad \{A'\}\ c\ \{B'\} \quad B' \Rightarrow B}{\{A\}\ c\ \{B\}}$$

This final rule is known as the *Rule of Consequence.* We refer to the partial correctness assertions derived using these rules as *theorems*. We want to know if the set of theorems derived using Hoare logic is the same as the set of valid PCA's.