CS 411 Lecture 10 Introduction to Axiomatic Semantics 22 Sep 2003
Lecturer: Greg Morrisett

Introduced by Floyd (on state charts) and generalized by Hoare as a logical or algebraic way of constructing and reasoning about programs. Gries, Dijkstra, and others really pushed on this idea, not just for simple imperative programming languages but for other purposes. Today, the ideas show up in projects such as proof-carrying code, SPLint, model checkers, etc.

Basic idea: we want to show that a given program c computes something. We can describe the something using a logical formula or assertion. For instance, we might write:

$$c \triangleq (x := 0; s := 0; \mathsf{while}\ (s \leq p)\ \mathsf{do}\ (x := x + s; s := s + 1))$$

Assuming that p is a positive integer, then what do we get as an output? A state where $s = p + 1$ and $x = \Sigma_{i=1}^{p+1} i$.

We might write this as:

$$\{p \geq 0\}\ c\ \{s = p + 1\ \wedge\ x = \Sigma_{i=1}^{p+1} i\}$$

Here, the assertion $\{p \geq 0\}$ is a pre-condition describing which states are valid for running the command. The assertion $\{s = p + 1\ \wedge\ x = \Sigma_{i=1}^{p+1} i\}$ is a post-condition specifying what the state looks like after the command completes.

In general, when we have:

$$\{\mathsf{Pre}\}\ c\ \{\mathsf{Post}\}$$

then we read this as:

> "If we start in a state s such that Pre is true, and if we run command c in state s and get an output state s', then Post will be true for s'."

Note that this is a *partial correctness* statement, as it doesn't say anything about a non-terminating command. In particular:

$$\{\mathsf{true}\}\ \mathsf{while\ true\ do\ skip}\ \{\mathsf{Post}\}$$

is a valid statement for any assertion Post, because the command does not terminate.

A *total correctness assertion*, sometimes written:

$$[\mathsf{Pre}]\ c\ [\mathsf{Post}]$$

says:

> "If we start in a state s satisfying Pre, then if we run c in state s, we will get an output state s' satisfying Post."

Pre/post-conditions are often written (informally) on interfaces as the contract between a client and a library designer. For instance, we might have a lookup procedure that expects its data to be in sorted order. That could be specified as a pre-condition. We might also have a sorting routine which would have as a post-condition that it leaves data in sorted order.

What we're going to do with axiomatic semantics:

- Specify language of assertions carefully.
- Define what we mean by a state satisfying an assertion.
- Specify pre/post-condition rules for IMP that are as general as possible. These are built using something called weakest pre-conditions (wp).

Weakest pre-conditions are a very powerful tool. They let us take an arbitrary post condition Post (a specification) and a command c, and automatically calculate a pre-condition Pre such that:

$$\{\mathsf{Pre}\}\ c\ \{\mathsf{Post}\}$$

Furthermore, Pre will be the weakest possible pre-condition, meaning it will put as few requirements on the input state as is possible to run the command and get a state satisfying the post-condition. Note that the ideal pre-condition is true.

$$\begin{array}{rcl} e \in \mathsf{Exp} & ::= & \ldots \mid n \qquad \longleftarrow \text{a logical variable} \\ A \in \mathsf{Assn} & ::= & \mathsf{true} \mid \mathsf{false} \mid e_1 \leq e_2 \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \implies A_2 \mid \neg A \mid \forall n.\ A \mid \exists n.\ A \end{array}$$

Assertions are like boolean expressions, except that we've added in $\forall$ and $\exists$ quantifiers for integers. These are useful for expressing facts at the logical level that correspond to loops. For instance, if we want to specify that all of the elements in an array M are sorted, we might write:

$$\begin{array}{c} \forall i.\ 0 \leq i < \mathsf{size}(M). \\ \forall j.\ i < j < \mathsf{size}(M).\ M[i] \leq M[j] \end{array}$$

Or if we wanted to assert that x divides y evenly, we might have something like:

$$\exists z.\ x * z = y$$

We now need to define when a state satisfies a given assertion. We write:

$$s \vDash_I A$$

to mean that state s satisfies assertion A, under interpretation I for the logical variables. Here, I is just a value for the logical variables in the assertion A. Its role will become apparent when we see what happens with quantifiers. We can define satisfaction as follows:

$s \vDash_I \mathsf{true}$ (always)	
$s \vDash_I e_1 \leq e2$	if $\mathcal{E}[\![\mathsf{subst}(I, e_1)]\!]s \leq \mathcal{E}[\![\mathsf{subst}(I, e_2)]\!]s$
$s \vDash_I A_1 \wedge A_2$	if $s \vDash_I A_1 \wedge s \vDash_I A_2$
$s \vDash_I A_1 \vee A_2$	if $s \vDash_I A_1 \vee s \vDash_I A_2$
$s \vDash_I A_1 \implies A_1$	if $s \nvDash_I A_1 \vee s \vDash_I A_2$
$s \vDash_I \neg A$	if $s \nvDash_I A$
$s \vDash_I \forall n.\ A$	if $\forall i.\ s \vDash_{I[i/n]} A$
$s \vDash_I \exists n.\ A$	if $\exists i.\ s \vDash_{I[i/n]} A$
$\emptyset \vDash_I A$	

Note that the definition relies upon an auxiliary function $\mathsf{subst}(I, e)$ which is meant to substitute away the logical variables in an expression. In particular:

$$\begin{array}{rcl} \mathsf{subst}(I, n) & = & I(n) \\ \mathsf{subst}(I, i) & = & i \\ \mathsf{subst}(I, x) & = & x \\ \mathsf{subst}(I, e_1\ \mathsf{op}\ e_2) & = & \mathsf{subst}(I, e_1)\ \mathsf{op}\ \mathsf{subst}(I, e_2) \end{array}$$

Now we can define:

$$s \vDash_I \{A_1\}\ c\ \{A_2\}$$

as meaning:

$$\text{If } s \vDash_I A_1 \text{ and } (s, s') \in \mathcal{C}[\![c]\!], \text{ then } s' \vDash_I A_2.$$

We write:

$$\vDash_I \{A_1\}\ c\ \{A_2\}$$

and say that this is a valid partial correctness assertion with respect to I, if for all states s, $s \vDash_I A_1$, if $(s, s') \in \mathcal{C}[\![c]\!]$, then $s' \vDash_I A_2$.

Finally, we write:

$$\vDash \{A_1\}\ c\ \{A_2\}$$

and say the partial correctness assertion is valid if for all interpretations I, $\vDash_I \{A_1\}\ c\ \{A_2\}$.

Proof rules for determining partial correctness assertions:

$$\textsf{skip}\quad \{A\}\ \textsf{skip}\ \{A\}$$

$$\textsf{assign}\quad \{A[e/x]\}\ x := e\ \{A\}$$

$$\textsf{seq}\quad \frac{\{A_1\}\ c_1\ \{A_2\} \{A_2\}\ c_2\ \{A_3\}}{\{A_1\}\ c_1; c_2\ \{A_3\}}$$

$$\textsf{if}\quad \frac{\{A_1 \wedge b\}\ c_1\ \{A_2\} \qquad \{A_1 \wedge \neg b\}\ c_2\ \{A_2\}}{\{A_1\}\ \textsf{if}\ b\ \textsf{then}\ c_1\ \textsf{else}\ c_2\ \{A_2\}}$$

$$\textsf{while}\quad \frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \textsf{while}\ b\ \textsf{do}\ c\ \{A \wedge \neg b\}}$$

$$\textsf{consequence}\quad \frac{\vDash A_1 \implies A_2 \qquad \{A_2\}\ c\ \{A_3\} \qquad \vDash A_3 \implies A_4}{\{A_1\}\ c\ \{A_4\}}$$