Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 12, Part 4: Axiomatic Semantics

Axiomatic Semantics

- Axiomatic semantics provides a machine-independent formalism for quantifying semantics.
- The development of axiomatic semantics is more abstract than denotational semantics, and involves assertions and logical formulas from predicate logic.

Axiomatic Semantics Development

- Assume an imperative language.
- In axiomatic semantics, the program semantics reduces to the semantics of a statement (derived from `<command>`) which is itself usually a sequence of commands derived from `<command-seq>`.
- The statement semantics are described by the addition of assertions that are always TRUE when control of the program reaches the assertions.
- In this sense, *program meaning* = *program correctness* with respect to a specification are given by pre and post condition assertions.
- As with denotational semantics, the divide-and-conquer approach is fundamental.

Axiomatic semantics is commonly used for three primary objectives:

1. To prove programs 'correct';
2. To provide formal specifications from which programs may be derived; and
3. (Our interest:) to develop a formal and quantitative description of the semantics of syntactically correct program fragments.

The relation between an initial logical assertion and a final logical assertion following a code fragment captures the essence of the code fragment semantics.

A Sample Assertion

A typical assertion (where $\cap$ indicates conjunction or AND) might look like:

$$\{m < 5 \ \cap \ k = n^2\}$$

where m, n and k are *variables in predicate calculus which are related to program (fragment) variables.*

The truth value of the sample assertion depends upon the values of m, n and k.

We are concerned with two classes of assertions:

1. Assertions about entities that are TRUE just before execution of the code fragment. These are represented using a set of preconditions, $\{P\}$.
2. Assertions about entities that are TRUE just after the execution of the code fragment. These are represented using a set of postconditions, $\{Q\}$.

Using these yields an axiomatics semantic description of a code fragment of the form:

$$\{P\} \texttt{<code fragment>} \{Q\}$$

The program semantics are based on assertions about logical relationships that remain the same each time the program executes.

The basic representational strategy[a] is:

$$\{logical\ preconditions\} \cap \{syntactic\ fragment\} \rightarrow$$

$$\{logical\ postconditions\}$$

[a]Note this involves implication.

With respect to axiomatic semantics:

- Variables in the predicate calculus (logical description) correspond to program (fragment) variables. Note that variables in the logical description get values from the corresponding program variables.
- Function symbols (connectives) in the logical description include all the operations available in the programming language. This includes $<$, $>$, $*$ and so forth.
- The truth value of a statement in predicate logic is dependent upon the values of component variables.
- *Note a significant distinction is made between* $=$ *in logic* (equality test) *and the use of* = (assignment) *in the programming language.*

Implication Review

Implication is another important logical connective and plays a role in the rules which accompany axiomatic semantics.

Implication is not the same as equality. The truth table for implication is shown below:

p	q	$p \rightarrow q$	$\neg p \cup q$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

A very important equivalence is:

$$\{(p \rightarrow q) = (\neg p \cup q)\} = T$$

Ordered-Pair and Specification

The axiomatic semantics of `C` are given by the ordered pair

$$(\{P\}, \{Q\})$$

Alternately, $(\{P\}, \{Q\})$ provides a **specification** for code (fragment) `C`.

`minic` Example

A simple `minic` code fragment is shown below. Note the semicolon (`;`) is only shown to illustrate the termination of the statement. It is **not** part of the string derived from `<command>`.

```
x = 1 / y;
```

One axiomatic representation for this code fragment in the form $\{P\}$ `<code fragment>` $\{Q\}$ is

$$\{y \neq 0\} \quad \texttt{x = 1 / y} \; \{x = \frac{1}{y}\}$$

It is critical to note in the above example that `x` and `y` are **program variables** whereas x and y are variables in the predicate calculus.

Partial and Total Correctness

Given:

$$\{P\}\ \mathtt{C}\ \{Q\}$$

Partial Correctness: A program (fragment) is partially correct if the following compound statement in predicate logic is true:

$$\{P\}\ \cap\ \mathtt{C}\ \text{executes}\ \rightarrow \{Q\}$$

Total Correctness: Total correctness extends partial correctness with the additional requirement that the program (fragment) is guaranteed to terminate. In other words,

Partial Correctness + Termination = Total Correctness

Lack of Uniqueness of the Axiomatic Specification

Axiomatic representations are not unique, and this is perhaps the Achilles heel of the technique.

Axiomatic Descriptions Using `minic`

Consider the `minic` program fragment involving assignment and an expression:

```
x = y + 1;
```

Suppose further we are given an axiomatic specification corresponding to this fragment as

$$\{y = -3\}\ \texttt{x = y + 1}\ \{x < 0\} \tag{1}$$

Correctness of the Sample Specification

Given the specification $\{y = -3\}$ `x = y + 1` $\{x < 0\}$, note y and x are logical variables and `y` and `x` are program variables. We proceed in the following manner:

1. Assume $\{y = -3\}$ = TRUE. This requires that y (the logical variable) has the value -3. Note this is not assignment, but rather $=$ is used as a logical connective, in this case the test for equality. This also constrains the corresponding program variable, `y`.
2. Assume `x = y + 1` executes.
3. Since `y` is constrained by the value of y, use the substitution indicated in the program fragment to get the value of `x`, and consequently x. Note we are not done.
4. $\{y = -3\}$ and the substitution $\rightarrow \{x = -2\}$ (is TRUE).

5. Observe $\{x = -2\} \to \{x < 0\}$. Now we are done, except for the following important observation.

In the previous example, be careful to note

$$\{x = -2\} \to \{x < 0\}$$

This is implication, not equality, and the converse

$$\{x < 0\} \to \{x = -2\}$$

is not true.

Alternative Specifications for the Same Fragment

An alternate is

$$\{y = 3\}\ \texttt{x = y + 1}\ \{x > 0\} \tag{2}$$

Thus (so far), the axiomatic specification of program fragment `x = y + 1` is either the pair

$$(\{y = -3\}, \{x < 0\})$$

or

$$(\{y = 3\}, \{x > 0\})$$

Furthermore, *another* possible precondition *for the same code fragment and* $\{Q\}$ in (2) is

$$\{P\} = \{y \geq 0\} \tag{3}$$

Standardization: The Weakest Precondition

Formal Definition. Given C and $\{Q\}$, the weakest precondition, denoted $\{W\}$, is defined as follows. Suppose $\{P\}$ is any precondition for which

$$\{P\}\ \mathtt{C}\ \{Q\}$$

holds. If

$$\{P\} \rightarrow \{W\}$$

(note this is logical implication), then $\{W\}$ is the weakest precondition.

Axiom for Command Sequencing

This applies to a minic command sequence of the form `C1;C2`, where `C1` and `C2` are commands.

$$\frac{\{P\}\ \mathrm{C1}\ \{Q1\} \cap \{Q1\}\ \mathrm{C2}\ \{Q2\}}{\{P\}\ \mathrm{C1;C2}\{Q2\}} \tag{4}$$

For clarity or to avoid possible ambiguity we write this as:

$$\frac{[\{P\}\ \mathrm{C1}\ \{Q1\}] \cap [\{Q1\}\ \mathrm{C2}\ \{Q2\}]}{\{P\}\ \mathrm{C1;C2}\{Q2\}} \tag{5}$$

Axiom for Assignment

The semantics of syntactic constructs using = (assignment) must be carefully considered in order to avoid developing nonsensical axiomatic semantics descriptions.

Formulation #1.

$$\{true\}\ \texttt{x = x + 1}\ \{x = x + 1\} \tag{6}$$

Notice the direct mapping of program variables and connectives to logical variables and connectives in (6), most importantly the (erroneous) mapping of =. Note the postcondition *must be false*, i.e., there is no value of x such that $x = x + 1$ is TRUE in logic.

The attempt to capture the axiomatic semantics of assignment via logical variable x and the = connective in logic fails.

Formulation #2. We try to improve upon the previous axiomatic semantics description with the following specification:

$$\{x = A\}\ \texttt{x = x + 1}\ \{x = A + 1\}$$

Whereas this captures the intuitive notion of assignment in a logically correct manner, it is not very general.

The Axiom for Assignment

`v = e` is a common command in imperative languages.

To develop the pre/post-condition pairs, the following is useful:

$$\{[e/v]Q\}\ \texttt{v = e}\ \{Q\} \tag{7}$$

where `v` is derived from `<identifier>` and `e` is derived from `<expr>`. Furthermore, the notation $[e/v]$ means the substitution of e for all free occurrences of v in postcondition Q.

For example, consider the `minic` fragment `y = x+y`. An axiomatic semantics description might be:

$$\{x = 1, x + y > 2\}\ \texttt{y = x + y}\ \{x = 1, y > 2\}$$

where we have used $[x + y/y]\{x = 1, y > 2\}$ to produce $\{x = 1, x + y > 2\}$.

Using The Assignment Axiom

Consider a previous example:

$$\{x = A\}\ \texttt{x = x + 1}\ \{x = A + 1\}$$

Given {Q} = {x=A+1} we can derive {P} via:

$$[\frac{x+1}{x}]\ \{x = A + 1\} = \{x + 1 = A + 1\} = \{x = A\}$$