1 Axiomatic Semantics

So far we have focused on *operational semantics*, which are natural for modeling computation or talking about how state changes from one step of the computation to the next. In operational semantics, there is a well-defined notion of *state*. We take great pains to say exactly what a state is and how it is manipulated by a program.

In *axiomatic semantics*, on the other hand, we do not so much care what the states actually are, but only the properties that we can observe about them. This approach emphasizes the relationship between the properties of the input (preconditions) and properties of the output (postconditions). This approach is useful for specifying what a program is supposed to do and talk about a program's correctness with respect to that specification.

2 Preconditions and Postconditions

The *preconditions* and *postconditions* of a program say what is true before and after the program executes, respectively. Often the correctness of the program is specified in these terms. Typically this is expressed as a contract: as long as the caller guarantees that the initial state satisfies some set of preconditions, then the program will guarantee that the final state will satisfy some desired set of postconditions. Axiomatic semantics attempts to say exactly what preconditions are necessary for ensuring a given set of postconditions.

3 An Example

Consider the following program to compute x^p:

```
$y = 1;$
$q = 0;$
while $(q < p)$ {
    $y = y\,x;$
    $q = q + 1;$
}
```

The desired postcondition we would like to ensure is $y = x^p$; that is, the final value of the program variable y is the pth power of x. We would also like to ensure that the program halts. One essential precondition needed to ensure halting is $p \geq 0$, because the program will only halt and compute x^p correctly if that holds. Note that $p > 0$ will also guarantee that the program halts and produces the correct output, but this is a stronger condition (is satisfied by fewer states, has more logical consequences).

$$\underbrace{p > 0}_{\text{stronger}} \quad \Rightarrow \quad \underbrace{p \geq 0}_{\text{weaker}}$$

The weaker precondition is better because it is less restrictive of the possible starting values of p that ensure correctness. Typically, given a postcondition expressing a desired property of the output state, we would like to know the *weakest precondition* that guarantees that the program halts and satisfies that postcondition upon termination.

4 Weakest Preconditions

Given a program S and a postcondition φ, the weakest property of the input state that guarantees that S halts in a state satisfying φ is called the *weakest precondition* of S and φ and is denoted $\mathsf{wp}\ S\ \varphi$. This says that

- $\mathsf{wp}\ S\ \varphi$ implies that S terminates in a state satisfying φ ($\mathsf{wp}\ S\ \varphi$ is a precondition of S and φ),
- if ψ is any other condition that implies that S terminates in a state satisfying φ, then $\psi \Rightarrow \mathsf{wp}\ S\ \varphi$ ($\mathsf{wp}\ S\ \varphi$ is the *weakest precondition* of S and φ).

As in the λ-calculus, juxtaposition represents function application, so wp can be viewed as a higher-order function that takes a program S and a postcondition φ and returns the weakest precondition of S and φ. The function wp can also be viewed as taking a program and returning a function that maps postconditions to preconditions. For this reason, axiomatic semantics is sometimes known as *predicate transformer semantics.*

5 Guarded Commands

Dijkstra introduced the Guarded Command Language (GCL) with the grammar

$$\begin{aligned} S \quad ::= \quad & \mathsf{skip} \mid x := E \mid S_1;\ S_2 \\ & \mid \mathsf{if}\ B_1 \to S_1 \mathbin{[\!]} B_2 \to S_2 \mathbin{[\!]} \cdots \mathbin{[\!]} B_n \to S_n\ \mathsf{fi} \\ & \mid \mathsf{do}\ B_1 \to S_1 \mathbin{[\!]} B_2 \to S_2 \mathbin{[\!]} \cdots \mathbin{[\!]} B_n \to S_n\ \mathsf{od} \end{aligned}$$

where the B_i are Boolean expressions. The B_i are called *guards* because they guard the corresponding statements S_i. The symbol $[\!]$ is the *nondeterministic choice operator* and is not to be confused with $|$. In if and do statements, a clause $B_i \to S_i$ is said to be *enabled* if its guard B_i is true.

Informally, when executing the if statement, at least one of its clauses must be enabled, otherwise it is a runtime error. One of the enabled clauses $B_i \to S_i$ is chosen nondeterministically and the corresponding statement S_i is executed. The do statement works similarly, except that there is no requirement that at least one clause be enabled. If none are enabled, execution just falls through to the following statement. If at least one is enabled, then one of the enabled clauses is chosen nondeterministically for execution. After the clause is executed, the guards are reexamined, and the process is repeated. This process repeats until all guards become false.

6 Weakest Preconditions in GCL

We now show how to determine the weakest preconditions for each part of GCL, as well as provide generic examples and special cases of the wp function.

Skip

Since skip does not do anything, we have $\mathsf{wp}\ \mathsf{skip}\ \varphi \ \Leftrightarrow\ \varphi$. Examples:

- $\mathsf{wp}\ \mathsf{skip}\ (x = 1) \quad \Leftrightarrow \quad x = 1$
- $\mathsf{wp}\ \mathsf{skip}\ \mathsf{false} \quad \Leftrightarrow \quad \mathsf{false}$.

Assignments

For assignments $x := E$,

$$\mathsf{wp}\ (x := E)\ \varphi \quad \Leftrightarrow \quad \varphi\{E/x\}.$$

Here $\varphi\{E/x\}$ denotes safe substitution of E for x in the formula φ, in which variables bound by quantifiers $\forall x$ or $\exists x$ are renamed if necessary to avoid capture. Note that the mapping $\varphi \mapsto \varphi\{E/x\}$ goes right-to-left; that is, $\varphi\{E/x\}$ is the precondition that must hold of the input state in order to ensure that the postcondition φ holds of the output state. Examples:

- $\mathsf{wp}\ (x := 1)\ (x = 1) \ \Leftrightarrow\ \mathsf{true}$. In words, $x = 1$ is true after $x := 1$ no matter what holds before execution.
- $\mathsf{wp}\ (y := 1)\ (x = 1) \ \Leftrightarrow\ x = 1$.
- $\mathsf{wp}\ (x := y)\ (x = 1) \ \Leftrightarrow\ (x = 1)\{y/x\} \ \Leftrightarrow\ y = 1$.
- $\mathsf{wp}\ (x := x + 1)\ (x = 3)$

  $\Leftrightarrow\ (x = 3)\{x + 1/x\}$ by definition of assignment

  $\Leftrightarrow\ x + 1 = 3$ by substitution

  $\Leftrightarrow\ x = 2$ by arithmetic,

  so $\mathsf{wp}\ (x := x + 1)\ (x = 3) \ \Leftrightarrow\ x = 2$.

Sequential Composition

To determine the weakest precondition for which φ holds after executing $S_1; S_2$, we first find the weakest precondition for which φ holds after the execution of S_2, and then determine the weakest precondition that ensures that property after S_1:

$$\mathsf{wp}\ (S_1; S_2)\ \varphi \quad \Leftrightarrow \quad \mathsf{wp}\ S_1\ (\mathsf{wp}\ S_2\ \varphi).$$

Examples:

- $\mathsf{wp}\ (x := 1;\ y := 2)\ (x = 1 \wedge y = 2)$

  $\Leftrightarrow\ \mathsf{wp}\ (x := 1)\ (\mathsf{wp}\ (y := 2)\ (x = 1 \wedge y = 2))$ by definition of ;

  $\Leftrightarrow\ \mathsf{wp}\ (x := 1)\ (x = 1 \wedge 2 = 2)$ by definition of assignment

  $\Leftrightarrow\ 1 = 1 \wedge 2 = 2$ by definition of assignment

  $\Leftrightarrow\ \mathsf{true}$ by predicate calculus.

- $\mathsf{wp}\ (x := x + 1;\ y := y - 1)\ (x \le y)$

  $\Leftrightarrow\ \mathsf{wp}\ (x := x + 1)\ (\mathsf{wp}\ (y := y - 1)\ (x \le y))$ by definition of ;

  $\Leftrightarrow\ \mathsf{wp}\ (x := x + 1)\ (x \le y - 1)$ by definition of assignment

  $\Leftrightarrow\ x + 1 \le y - 1$ by definition of assignment

  $\Leftrightarrow\ y - x \ge 2$ by arithmetic.

If

In an if statement, at least one guard B_i must be true. This condition is expressed by the disjunction $B \stackrel{\triangle}{=} \bigvee_i B_i$. The S_i that is chosen for execution is chosen nondeterministically among all enabled clauses, and in order to guarantee the postcondition φ, all enabled clauses had better guarantee φ. Thus

$$\mathsf{wp}\ (\mathsf{if}\ B_1 \to S_1 \ [\!]\ \cdots\ [\!]\ B_n \to S_n\ \mathsf{fi})\ \varphi \quad \Leftrightarrow \quad B \wedge \bigwedge_i (B_i \Rightarrow \mathsf{wp}\ S_i\ \varphi).$$

Example: The following program computes the maximum of two numbers.

$$MAX \quad \stackrel{\triangle}{=} \quad \mathsf{if}\ x \geq y \to z := x \ \ [\!]\ \ y \geq x \to z := y\ \mathsf{fi}$$

To prove that the program halts and correctly computes the maximum of x and y regardless of input state, it suffices to show that true is the weakest precondition corresponding to the postcondition $z = \max x, y$.

$$\begin{array}{ll}
\mathsf{wp}\ MAX\ (z = \max x, y) & \\
\quad \Leftrightarrow\ x \geq y \vee y \geq x & \\
\qquad \wedge\ (x \geq y \Rightarrow (\mathsf{wp}\ (z := x)\ (z = \max x, y)) & \\
\qquad \wedge\ (y \geq x \Rightarrow (\mathsf{wp}\ (z := y)\ (z = \max x, y)) & \text{by definition of } \mathsf{if} \\
\quad \Leftrightarrow\ \mathsf{true} & \\
\qquad \wedge\ (x \geq y \Rightarrow x = \max x, y) & \\
\qquad \wedge\ (y \geq x \Rightarrow y = \max x, y) & \text{by predicate calculus and the definition of assignment} \\
\quad \Leftrightarrow\ \mathsf{true} & \text{by predicate calculus.}
\end{array}$$

Do

Since do has the complication that it may not terminate, it is difficult to formalize its weakest precondition. In fact, over arbitrary structures, first-order predicate logic is not sufficiently expressive to formulate weakest preconditions for this construct. However, we can use infinitary logic (logic with infinite conjunctions and disjunctions). We can write

$$\mathsf{wp}\ (\mathsf{do}\ B_1 \to S_1 \ [\!]\ \cdots\ [\!]\ B_n \to S_n\ \mathsf{od})\ \varphi \quad \Leftrightarrow \quad \bigvee_k P_k,$$

where informally P_k is the weakest precondition ensuring that the do statement terminates in exactly k iterations and satisfies φ upon termination. Formally, let E be the body of the do statement (thus the statement is $\mathsf{do}\ E\ \mathsf{od}$), and let $B = \bigvee_i B_i$ as above. Define inductively

$$P_0 \quad \stackrel{\triangle}{=} \quad \neg B \wedge \varphi, \tag{1}$$

$$P_{k+1} \quad \stackrel{\triangle}{=} \quad B \wedge \mathsf{wp}\ (\mathsf{if}\ E\ \mathsf{fi})\ P_k. \tag{2}$$

The basis condition (1) says that no clause is enabled and φ is true of the input state, which is equivalent to the condition that the body E of the do statement is executed exactly 0 times and terminates in a state satisfying φ. The inductive condition (2) says that B is true, thus the body E of the do statement is executed at least once, and after executing the body once, P_k will hold, implying that the do statement will execute exactly k more times and satisfy φ upon termination.

If you don't like infinitary logic, you can do the same thing with the μ-calculus predicate

$$\mu P.\, (\neg B \wedge \varphi) \vee (B \wedge \mathsf{wp}\ (\mathsf{if}\ E\ \mathsf{fi})\ P),$$

which denotes the least fixpoint of the monotone map $\lambda P.\, (\neg B \wedge \varphi) \vee (B \wedge \mathsf{wp}\ (\mathsf{if}\ E\ \mathsf{fi})\ P)$ on predicates.

7 Refinement

For nondeterministic programs S and T, we say that S *refines* T iff for any starting state, the set of possible final states of S is a (not necessarily strict) subset of the set of possible final states of T.

Consider for example

$$\begin{aligned} \mathsf{S} &\triangleq \mathsf{if}\ x = 1 \to y := 1\ [\!]\ x \neq 1 \to \mathsf{skip}\ \mathsf{fi} \\ \mathsf{T} &\triangleq \mathsf{if}\ x = 1 \to y := 1\ [\!]\ x = 1 \to y := 2\ [\!]\ x \neq 1 \to \mathsf{skip}\ \mathsf{fi} \end{aligned}$$

For input state $(x = a, y = b)$, the only possible final states of S are $(x = 1, y = 1)$ if $a = 1$ and $(x = a, y = b)$ if $a \neq 1$, whereas the possible final states of T are $\{(x = 1, y = 1), (x = 1, y = 2)\}$ if $a = 1$ and just $(x = a, y = b)$ if $a \neq 1$, therefore S refines T.

The refinement relation is usually used only with programming languages with some form of nondeterministic choice as a relative measure of how nondeterministic a program is. A correctness specification might be written using nondeterministic choice, since often we may be happy with any one of a range of outcomes. Any deterministic program that refines the specification is considered correct.

8 Weakest Liberal Preconditions

Recall that the weakest precondition of a program S and a postcondition φ is the weakest precondition that guarantees that S halts and satisfies φ upon halting.

The *weakest liberal precondition* (wlp) of a program S and a postcondition φ is the weakest precondition that guarantees that if S halts, then it satisties φ upon halting. The weakest liberal precondition of S and φ is denoted $\mathsf{wlp}\ S\ \varphi$.

The difference between $\mathsf{wp}\ S\ \varphi$ and $\mathsf{wlp}\ S\ \varphi$ is that $\mathsf{wp}\ S\ \varphi$ implies that S terminates, whereas $\mathsf{wlp}\ S\ \varphi$ does not. Since $\mathsf{wlp}\ S\ \varphi$ is weaker than $\mathsf{wp}\ S\ \varphi$, it is presumably easier to establish.

Recall that weakest preconditions of the do construct of GCL are not necessarily expressible in first-order logic. In fact, the same will be true of the weakest liberal preconditions. However, we may be able to find *some* precondition that will be sufficient to establish correctness, even though that precondition may not necessarily be the weakest. In other words, we will find ψ such that $\psi \Rightarrow \mathsf{wlp}\ S\ \varphi$.

Recall that if and do statements look like

$$\begin{gathered} \mathsf{if}\ B_1 \to S_1\ [\!]\ B_2 \to S_2\ [\!] \cdots [\!]\ B_n \to S_n\ \mathsf{fi} \\ \mathsf{do}\ B_1 \to S_1\ [\!]\ B_2 \to S_2\ [\!] \cdots [\!]\ B_n \to S_n\ \mathsf{od} \end{gathered}$$

Define

$$B \triangleq \bigvee_{i=1}^{n} B_i.$$

We will look for a property ψ such that

(i) $\psi \wedge B \quad \Rightarrow \quad \mathsf{wlp}\ (\mathsf{if}\ B_1 \to S_1\ [\!]\ B_2 \to S_2\ [\!] \cdots [\!]\ B_n \to S_n\ \mathsf{fi})\ \psi$

(ii) $\psi \wedge \neg B \quad \Rightarrow \quad \varphi.$

Property (i) says that ψ is a *loop invariant*: if it holds before execution of the body of the do loop, and if at least one clause in the body is enabled, then it holds after one execution of the body. It follows by induction that if ψ holds before execution of the do loop, then it will hold after any number of iterations of the loop. Property (ii) says that if ψ holds and no clause of the loop is enabled (so that the loop will fall through), then the postcondition φ is satisfied.

These observations say that the following proof rule is valid:

$$\frac{\psi \wedge B \;\Rightarrow\; \mathsf{wlp}\ (\mathsf{if}\ B_1 \to S_1 \,[\!]\, B_2 \to S_2 \,[\!]\, \cdots \,[\!]\, B_n \to S_n\ \mathsf{fi})\ \psi \qquad \psi \wedge \neg B \;\Rightarrow\; \varphi}{\psi \;\Rightarrow\; \mathsf{wlp}\ (\mathsf{do}\ B_1 \to S_1 \,[\!]\, B_2 \to S_2 \,[\!]\, \cdots \,[\!]\, B_n \to S_n\ \mathsf{do})\ \varphi}.$$

Note that a loop invariant need not hold continuously throughout the execution of the body of the loop. It is enough that it holds when the loop iteration is complete.

For example, consider the following program, which for some function $f : \mathsf{Int} \to \mathsf{Bool}$ finds the least x such that $f(x)$ (assume that such an x exists, so that the program will terminate).

```
$x := 0$;
do $\neg f(x) \to x := x + 1$ od
```

An appropriate postcondition that specifies what it means for the program to be correct is

$$\varphi \;\overset{\triangle}{\iff}\; f(x) \;\wedge\; \forall y\ \ 0 \le y < x \;\Rightarrow\; \neg f(y).$$

(read as "f holds of x and does not hold of any number smaller than x"). One method of finding a good loop invariant is to look at ways of weakening the postcondition, thereby allowing more states to satisfy the predicate. In this case, we can eliminate the conjunct asserting that we have already found a good x. This yields the invariant

$$\psi \;\overset{\triangle}{\iff}\; \forall y\ \ 0 \le y < x \;\Rightarrow\; \neg f(y).$$

One can show that this is indeed an invariant and satisfies the two premises of the proof rule above with $B = \neg f(x)$, therefore

$$\psi \;\Rightarrow\; \mathsf{wlp}\ (\mathsf{do}\ \neg f(x) \to x := x + 1\ \mathsf{od})\ \varphi.$$

The definitions of wlp S φ for the other basic constructs of GLC are the same as wp S φ except for if, which is

$$\mathsf{wlp}\ (\mathsf{if}\ B_1 \to S_1 \,[\!]\, \cdots \,[\!]\, B_n \to S_n\ \mathsf{fi})\ \varphi \;\overset{\triangle}{=}\; \bigwedge_{i=1}^{n} (B_i \Rightarrow \mathsf{wlp}\ S_i\ \varphi).$$

Note that we no longer require B as with wp, since we do not have to ensure halting.