Program Construction

Roland Backhouse

January 2001

Outline

- Program Specification
- Assignments
- Conditional Statements
- Sequential Composition
- Loops

Program Specification

Comments

When writing computer programs it is a very good idea to comment them thoroughly in order to explain what is going on.

Comments can also be almost useless. The comment

```
increment i by 1
```

immediately preceding the C/Java statement

```
i++
```

is completely useless to the experienced programmer who can be expected to know that "`i++`" means "increment `i` by one" in C/Java idiom.

Useless comments simply repeat in natural language what is stated simply and precisely in the program statements. They are *operational*.

Good comments, on the other hand, should have added value. They should supplement the program text with explanations of the program's function and why the code that is used achieves that function.

Assertions

Here, comments will be indicated by enclosing them in curly brackets — "{" and "}".

They state formal properties of the program variables at a particular point in the execution of the program.

For example, the text of a program may look like

$$\ldots \quad \{\mathtt{i}=0\} \quad \ldots$$

where the dots represent some arbitrary program statements.

The intended meaning is that, when execution of the program has reached the point in the program text where the comment appears, the value of the variable $\mathtt{i}$ is guaranteed be zero.

Such comments are called *assertions*, *conditions* or *properties*.

Bracketed Statements

When a program statement is bracketed by two comments, as for example in

$$\{\ 0<\mathtt{i}\}\ \mathtt{i} := \mathtt{i}-1\{\ 0\leq\mathtt{i}\}\quad ,$$

we reason about the correctness of the program statement on the basis that the first comment acts as an assumption.

That is, we understand the comments as claiming that *if* $0<\mathtt{i}$ before the statement $\mathtt{i} := \mathtt{i}-1$ is executed *then* $0\leq\mathtt{i}$ after the assignment has been executed.

Hoare Triples

An expression of the form

$$\{\ P\}\ S\{\ Q\}\quad ,$$

where P and Q are properties of the program variables and S is a program statement (some portion of the program text), is called a *Hoare triple*.

The property P is called the *precondition* and the property Q is called the *postcondition* of the statement S.

We read such a triple as the claim that if the property P holds of the program variables before execution of statement S then execution of S is guaranteed to terminate and afterwards the program variables will satisfy property Q.

A Hoare triple thus denotes a boolean value; if the value is true then we say the triple is valid, and if it is false we say the triple is invalid.

Valid Triples

$$\{\, i=0 \,\}\; i := i+1 \{\, i=1 \,\} \quad ,$$

$$\{\, i+j=0 \,\}\; i := i+1 \,;\, j := j-1 \{\, i+j=0 \,\} \quad ,$$

$$\{\, \mathsf{true} \,\}\; i := 1 \{\, i=1 \,\} \quad .$$

Invalid Triples

$$\{\, i=1 \,\}\; i := i+1 \{\, i=0 \,\} \quad ,$$

$$\{\, i+j\neq 0 \,\}\; i := i+1 \,;\, j := j-1 \{\, i+j=0 \,\} \quad ,$$

$$\{\, \mathsf{true} \,\}\; i := 1 \{\, i=0 \,\} \quad .$$

Exercise 1 Using your current knowledge say which of the following is a valid Hoare triple. (Shortly we show how to validate Hoare triples formally.)

(a) $\{\ i=1\ \}\ j := i\ \{\ i=j=1\ \}$

(b) $\{\ i=1\ \}\ i := j\ \{\ i=j=1\ \}$

(c) $\{\ 0\leq i<N\ \}\ i := i+1\ \{\ 0<i\leq N\ \}$

(d) $\{\ \mathsf{true}\ \}\ i := j+1\ \{\ i<j\ \}$

(e) $\{\ i=1\ \}\ i := 0\ \{\ \mathsf{true}\ \}$

(f) $\{\ i=0\ \}\ i := 1\ \{\ \mathsf{false}\ \}$

(g) $\{\ \mathsf{false}\ \}\ i := 1\ \{\ i=0\ \}$

□

Pre and Post Conditions

The specification of a program, in its simplest form, is a *relation* between *input* values and *output* values.

It is important to note that specifications are by nature *nondeterministic*.

There is usually some latitude in what is acceptable output for given input.

In mathematical terms, specifications are truly *relations* and not *functions*.

A program S is specified by stating a *precondition* P and a *postcondition* Q and requiring that S be constructed to satisfy

$$\{ P\}\ S\{ Q\} \quad .$$

If so, we say that S *establishes* (postcondition) Q under the assumption of precondition P.

Problems

Four main problems with the use of Hoare triples are

(a) we are forced to name the variables to be used in the program (whereas the names are irrelevant to the specification),

(b) there is no way of saying which variables may be altered in the course of execution of the program and which should remain constant (that is, there is no distinction between input and output variables),

(c) there is no way of limiting the mechanisms for updating the values of the output variables,

(d) an artificial mechanism (so-called "ghost" or "rigid" variables) sometimes needs to be employed to relate the input values of variables to their desired output values.

Output Variables

The second problem is illustrated by a very simple example. If we require that program S satisfies

$$\{\ \mathsf{true}\}\ S\{\ \mathtt{i}=\mathtt{j}\}\ .$$

then this can be achieved by the assignments

$$\mathtt{i} := \mathtt{j}$$

and

$$\mathtt{j} := \mathtt{i}\ ,$$

there being no way to distinguish between the two variables.

In reality one of $\mathtt{i}$ and $\mathtt{j}$ would be the input value and the other the output value, and the requirement would be to assign a value to the output variable so as to meet the specification leaving the value of the input variable unchanged.

The problem is resolved informally — we state which are the input and which are the ouput variables in the text accompanying the formal specification.

Ghost Variables

Suppose we want to specify that the sum of two variables $\mathtt{i}$ and $\mathtt{j}$ should remain constant.

We specify this by introducing a *ghost* variable C.

This variable should not be used anywhere else in the program; to distinguish ghost variables from normal program variables we use a sans-serif type.

Then the program S is specified by

$$\{\ \mathtt{i}+\mathtt{j}=\mathsf{C}\}\ \mathsf{S}\{\ \mathtt{i}+\mathtt{j}=\mathsf{C}\}\quad .$$

This says that if the sum of $\mathtt{i}$ and $\mathtt{j}$ has the value C before execution of statement S then execution of statement S is guaranteed to terminate in a state in which the sum of $\mathtt{i}$ and $\mathtt{j}$ still has the value C.

Ghost variables are treated just like ordinary program variables but the program code may not refer to them in any way.

Assignment

It is convenient to allow *simultaneous assignments*. In a simultaneous assignment, the left side is a list of variables and the right side is a list of expressions of the same length as the list of variables.

A simultaneous assignment to three variables x, y, and z is, for example,

$$x,y,z \;:=\; 2\times y\,,x+y\,,3\times z \;\;.$$

A simultaneous assignment

$$x_0\,,x_1\,,\ldots,x_n \;:=\; e_0\,,e_1\,,\ldots,e_n$$

is executed by evaluating all of the expressions $e_0\,,e_1\,,\ldots,e_n$ and then, for each i, updating the value of the variable x_i to the value obtained for expression e_i.

The assignment

$$x,y \;:=\; y,x$$

has the effect of swapping the values stored in variables x and y.

Restrictions

The variables on the left side of a simultaneous assignment should be pairwise distinct. For example, the assignment

$$x,x \;:=\; 0,1$$

doesn't make sense and is disallowed.

Very occasionally it is useful to relax this requirement. The statement

$$a[i]\,,a[j] \;:=\; a[j]\,,a[i]$$

swaps the ith and jth values in the array a.

When i and j are equal the statement means "do nothing".

Assignment Axiom

$$\{\ Q[x := e]\}\ x := e\{\ Q\}$$

Example

Application of the assignment axiom gives

$$\{\ 0=0\}\ i := 0\{\ i=0\}\ \ .$$

Of course, $0=0$ simplifies to true. So the conclusion is:

$$\{\ \mathsf{true}\}\ i := 0\{\ i=0\}\ \ .$$

Example

Application of the assignment axiom gives

$$\{\ 2\times i<10\}\ i := 2\times i\{\ i<10\}\quad .$$

Again, the precondition can be simplified, this time to $i<5$. So we conclude that:

$$\{\ i<5\}\ i := 2\times i\{\ i<10\}\quad .$$

Assignment Axiom (More Than One LHS Variable)

If x is the list $x_0, x_1, \ldots, x_n$ and e is the list $e_0, e_1, \ldots, e_n$ then $Q[x := e]$ denotes the *simultaneous* substitution of e_0 for x_0, e_1 for x_1, and so on.

Example

Consider the postcondition $i+j=C$ and the simultaneous assignment $i,j := i+1, j-1$. Then, simultaneously substituting "$i+1$" for "i" and "$j-1$" for "j", application of the assignment axiom gives

$$\{ (i+1)+(j-1)=C \}\ i,j := i+1, j-1 \{ i+j=C \} .$$

Simplifying the precondition we get

$$\{ i+j=C \}\ i,j := i+1, j-1 \{ i+j=C \} .$$

Exercise 2 Perform the following substitutions. Be careful with parenthesisation and remove unnecessary parentheses. (A raised infix dot denotes multiplication. Multiplication has precedence over addition)

1. $x[x := x+2]$
2. $(y \cdot x)[x := x+y]$
3. $(x+y)[x := x+y]$
4. $(x+1)[y := x]$
5. $x[x, y := 0\,, x+2]$
6. $(x + y \cdot x)[x, y := x-y\,, x+y]$
7. $(x+y)[x, y := x \cdot y\,, x \cdot y]$

□

Exercise 3 Using the assignment axiom , determine preconditions for the following statements and postconditions. Simplify the preconditions you obtain.

	Statement	Postcondition
(a)	$x := x+1$	$x+y<10$
(b)	$x := x-1$	$x^2+2\cdot x=0$
(c)	$x,y := x-y, x+y$	$x\cdot y=1$
(d)	$x,y,z := z,x,y$	$x=0 \vee y=1 \vee z=2$

□

Calculating Assignments

Suppose the requirement is to maintain the value of the sum $j+k$ constant whilst incrementing k by 1. Our task is to calculate an expression X such that

$$\{\, j+k=C\}\ j,k \ :=\ X, k+1\{\, j+k=C\} \ .$$

Applying the assignment axiom, we get

$$\{\, X+k+1=C\}\ j,k \ :=\ X, k+1\{\, j+k=C\} \ .$$

Comparing the precondition so obtained with the given precondition, the specification is met if

$$j+k=C \ \Rightarrow\ X+k+1=C \ .$$

Now,

$$
\begin{array}{ll}
 & \mathsf{j{+}k} \\
= & \quad \{ \quad \text{arithmetic — introducing “k+1”} \quad \} \\
 & \mathsf{j{+}k{+}1{-}1} \\
= & \quad \{ \quad \text{rearranging} \quad \} \\
 & \mathsf{(j{-}1){+}k{+}1} \quad .
\end{array}
$$

It thus follows that a suitable value of X is $\mathsf{j{-}1}$. That is,

$$\{\, \mathsf{j{+}k{=}C} \,\}\, \mathsf{j,k} \;:=\; \mathsf{j{-}1}\,,\mathsf{k{+}1}\{\, \mathsf{j{+}k{=}C} \,\} \quad .$$

Suppose variables s and n satisfy the property

$$s = n^2$$

and we want to increment n by 1 whilst maintaining this relationship between s and n.

Our goal is to calculate an expression X involving only addition such that

$$\{\ s = n^2\}\ s,n\ :=\ s+X, n+1\{\ s = n^2\}\ .$$

Applying the assignment axiom we get

$$\{\ s+X = (n+1)^2\}\ s,n\ :=\ s+X, n+1\{\ s = n^2\}\ .$$

Comparing with the specification we calculate X so that

$$s = n^2\ \Rightarrow\ s+X = (n+1)^2\ .$$

Now,

$$
\begin{array}{lll}
 & (n+1)^2 & \\
= & \{ \quad \text{arithmetic — introducing “}n^2\text{”} \quad \} & \\
 & n^2+2n+1 \quad . &
\end{array}
$$

That is,

$$s=n^2 \;\Rightarrow\; s+2n+1=(n+1)^2 \quad .$$

In this way we have calculated the required assignment statement:

$$\{\, s=n^2 \,\}\; s,n \;:=\; s+2n+1 \;,\; n+1 \{\, s=n^2 \,\} \quad .$$

Exercise 4 Suppose there are three program variables n, s and t. Calculate assignments to s and t that maintain invariant the relationship

$$s = n^2 \wedge t = n^3 \quad .$$

In other words, calculate X and Y such that

$$\{\, s = n^2 \wedge t = n^3 \,\}\ s,t,n \ :=\ s+X, t+Y, n+1 \{\, s = n^2 \wedge t = n^3 \,\} \quad .$$

The assignments to s and t should involve additions only. Multiplications are not allowed.

□

Complications

Division by zero, overflow and underflow errors, out-of-bound errors in array indexing, etc. are catered for by the more complete rule:

$$\{\ \text{“}e\text{”}\ \text{is well-defined} \wedge Q[x := e]\}\ x := e\{\ Q\}\quad .$$