Chapter 1: Specifications, Hoare Triples and Weakest Preconditions.

One of the main aims of Computing Science is to develop a scientific method of programming. Rather than relying on you natural cleverness and intuition to guide you in writing programs we would like to develop methods that didn't rely on individual cleverness or intuition. Such a method would allow us to construct programs in a step by step way, allow us to reason about the behaviour of programs and give us confidence in the correctness of what we have constructed.

Computing Science is still quite young and we haven't yet developed a complete model of programming, but we have made a lot of progress.

The very beginning of the programming task is to describe precisely what the program is supposed to achieve. We must be careful not to fall into the trap of describing what is to be achieved in terms of how it is to be achieved. These are two things here which need to be separated, and kept separate. **What** is to be achieved and **how** it is to be achieved are different and mixing them can lead to confusion.

What we want a program to achieve is usually called a **Specification** or a Program Specification. Generally it consists of a description of what you would like to be true when the program finishes, and what assumptions you can make before the program starts. An example might be like this. I want a program that adds up all the values in a 100 element array of integers. I will assume that when we start the array already has values in it.

Now suppose we have a specification and someone writes a program to solve the problem, how can we be sure that the program really does solve the problem. Another way to say this is to ask how we can be sure that the program is correct? **Correctness** is the most important quality that a program can have. How do we check for correctness? A common answer is to test the program. This is silly. If you test a program with some inputs and the result is wrong then you know the program is not correct. If you test it with some inputs and the result is correct it doesn't mean the program is correct, just that it correctly produced the answer for those particular inputs.

To prove it was correct by testing you would have to test it with all possible inputs and see if it gave the correct answer each time. But if you already knew the correct answer for each set of inputs why would you waste time writing a program? :-)

Hoare Triples.

Early attempts to try to model correctness in a mathematical way began in the early 1960s but really didn't achieve very much However, in the early 1970s a more useful

approach was developed by Tony Hoare. He proposed a notation which is now called called a **Hoare triple**.[1]

Here is an example of a Hoare triple.

$$\{P\}\ S\ \{Q\}$$

P and Q describe **States**. This means they tell you some[2] things that will be true at that point. S describes a mechanism, you can think of this as some instructions (or perhaps a program). P is called the **Precondition** and Q is called the **Postcondition**. Here are a few examples:

$$\{0 \leq x\}\ S\ \{y = x^2\}$$

Here the Precondition tells us that a variable x will have a value that is at least 0 at this point. The Postcondition tells us that at that point the variable y will have a value which is x squared.

$$\{x, y : nat\}\ S\ \{z = max(x, y)\}$$

Here the Precondition tells us that both x and y contain natural numbers and the Postcondition tells us that at that point z will contain the largest of the values x and y.

A Hoare triple is a Boolean Expression and it is true if we start in a state where P is true, perform the instruction S and finish, then we finish in a state where Q is true, Otherwise it is false.

Let us look at a few examples. In what follows we make use of an assignment of the form x := E, you can think of this operationally as a command which evaluates the expression E and places the result in the variable x. Later we will show you non-operational interpretations of assignment.

$$\{x = 7\}\ x := x + 4\ \{x = 11\}$$

$$\{x = 8\}\ x := x + 3\ \{x > 10\}$$

$$\{x > 5\}\ x := x + 1\ \{x > 6\}$$

---

[1] Named after C.A.R. Hoare who was once the professor of Computer Science in Queens University in Belfast before moving on to Oxford. He has since retired and joined Microsoft.

[2] We only focus on the true things that are of interest to us. Of course many other things may be true in a particular state, for example 1+1=2 :-) but it would be silly to write that down.

$\{x = 12\}\ x := x - 2\ \{ x > 11\}$

$\{x > 100\}\ x := x + 2\ \{x < 200\}$

The first 3 of these are valid, the final 2 are invalid.

Extreme cases of Hoare Triples.

There are four extreme cases which are interesting to look at. Note that you rarely will encounter cases like these.

{ True } S { True }
This is valid.

{ True } S { False }
This is invalid, the postcondition can never be made True.

{ False } S { True }
If you could start in a state where False was True then you could achieve anything you liked, so although this one looks strange it is in fact valid.

{ False } S { False }
Same as above, this is valid.

Some Laws of Hoare Triples.

Hoare was able to formulate a set of laws or rules for calculating with Hoare Triples. We list the main ones here. Note, in these examples the variables used are of type Integer.

$\{P\}\ S\ \{Q\} \wedge \{P\}\ S\ \{R\} \Rightarrow \{P\}\ S\ \{Q \wedge R\}$

An example of this would be the following

$\{ 0 \leq x \}\ x := 2*x\ \{ 0 \leq x \} \wedge$
$\{ 0 \leq x \}\ x := 2*x\ \{ \text{even.x} \} \Rightarrow \{ 0 \leq x \}\ x := 2*x\ \{ 0 \leq x \wedge \text{even.x} \}$

$\{P\}\ S\ \{Q\} \wedge \{R\}\ S\ \{Q\} \Rightarrow \{P \vee R\}\ S\ \{Q\}$

An example of this would be the following

$\{ 0 \leq x \}\ x := x*x\ \{ 0 \leq x \} \wedge$
$\{ x < 0 \}\ x := x*x\ \{ 0 \leq x \} \Rightarrow \{ 0 \leq x \vee x < 0 \}\ x := x*x\ \{ 0 \leq x \}$

$\{P\}\ S\ \{Q\} \wedge (Q \Rightarrow R) \quad \Rightarrow \quad \{P\}\ S\ \{R\}$ "weakening/strengthening post"

An example of this would be

$\{ x = 7 \wedge y = 12\}\ x := (x-1)*2\ \{ x = 12 \wedge y = 12 \} \wedge (x = 12 \wedge y = 12 \Rightarrow x = y)$
$\Rightarrow$
$\{ x = 7 \wedge y = 12\}\ x := (x-1)*2\ \{ x = y \}$

$\{P\}\ S\ \{Q\} \wedge (R \Rightarrow P) \quad \Rightarrow \quad \{R\}\ S\ \{Q\}$ "weakening/strengthening pre"

An example of this would be

$\{ x \leq 10\}\ x := x+1\ \{x \leq 11 \} \wedge (x \leq 6 \Rightarrow x \leq 10)$
$\Rightarrow$
$\{ x \leq 6 \}\ x := x+1\ \{x \leq 11 \}$

$\{P\}\ S0\ \{Q\} \wedge \{Q\}\ S1\ \{R\} \Rightarrow \{P\}\ S0;S1\ \{R\}$ "concatenation"

An example of this would be

$\{ x \leq 10 \}\ x := x+1\ \{ x \leq 11 \} \wedge \{ x \leq 11 \}\ x := x+2\ \{ x \leq 13 \}$
$\Rightarrow$
$\{ x \leq 10 \}\ x := x+1\ ;\ x := x+2\ \{ x \leq 13 \}$

Hoare introduced these laws in the early 1970's and used them to prove the correctness of programs which he had written. It was generally believed that the effort involved in proving even the simplest programs to be correct was a lot and this approach never really became popular outside of academic schools. Additionally, this approach does contain another flaw; what if the program that you are trying to prove correct isn't actually correct? We now know that proving a program to be correct after it has been written is indeed difficult if not impossible. [3]

However, in some parts of the world people do continue to try to prove existing programs to be correct. Nowadays, a lot of effort has gone into trying to automate some of the process. This also presents difficulties which you can read about yourselves.

However, there is a way in which this work has proved to be a useful basis for constructing correct programs and that is what we will start to look at now.

---

[3] After the fact proof is generally known as **program verification**. It is still popular and people devote lots of effort into developing tools to assist in the proof.

An introduction to the Weakest Precondition.

One of the world's greatest computing scientists, Edsger W Dijkstra, developed Hoare's work further and eventually developed the method of programming which we now use. His first major contribution to the field was the notion of the Weakest Precondition.

Consider the following Hoare triples

(0) $\{ 11 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(1) $\{ 10 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(2) $\{ 9 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(3) $\{ 8 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(4) $\{ 7 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(5) $\{ 6 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(6) $\{ 5 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(7) $\{ 4 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$
(8) $\{ 3 \leq x \}\ x := x + 1\ \{ 5 \leq x \}$

We note that (0) through (7) are all valid, (8) is not. In each of the examples the assignment and the postconditions remain the same, what changes is the precondition. They form a weakening chain

$$\{11 \leq x\} \Rightarrow \{10 \leq x\} \Rightarrow \ldots\ldots \Rightarrow \{4 \leq x\}$$

In this chain we say that $\{11 \leq x\}$ is the strongest and $\{4 \leq x\}$ is the weakest.

$\{4 \leq x\}$ is **the minimum which must be true** in order that performing the assignment $x := x + 1$ will bring us to the state $\{5 \leq x\}$. We say that $\{4 \leq x\}$ is the **Weakest Precondition.**

We write the Weakest Precondition as follows

$$WP.(x := x + 1).(5 \leq x)$$

The relationship between Hoare triples and WP.

We link these two as follows

$$\{P\}\ S\ \{Q\} \quad \equiv \quad P \Rightarrow WP.S.Q$$

For a particular postcondition Q and program S there may be many preconditions P which will make $\{P\}\ S\ \{Q\}$ a valid Hoare triple. However, there will only ever be one

Weakest Precondition. As we will see later, this Weakest Precondition will be something which we can calculate instead of having to try to guess it.

Weakest preconditions were first proposed by Dijkstra in the early 1970's. He used them to describe the semantics of a minimal language called the Guarded Command Language which we use.

The guarded command language has become the standard way in which Computing Scientists express algorithms. It has a very small number of commands and is machine independent which makes it an excellent vehicle for a clean and precise description of algorithms. Weakest Preconditions have proved to be even more useful and form the basis of the methods of program calculation which we shall explore.