Chapter 2 : The Guarded Command Language.

In which we describe our simple language.

The guarded command language was invented by Dijkstra as a way to express algorithms. It is a minimal, imperative language. It has a number of nice features, one of them is that it doesn't "run" on any electronic digital computer. This means that we do not have to concern ourselves with the physical limitations of such machines. When we have a finished algorithm it is a trivial job to translate it into a standard imperative language and compile it.

Because it has such a small set of statements it makes it easier to learn and because the semantics of the language are formally described using WP there is no ambiguity or vagueness, we know precisely what the statements mean. The language is as follows.

(0) Skip.

$$WP.skip.Q \quad \equiv \quad Q$$

Informally, you might think of skip as "do nothing". This may seem strange at first but remember lots of people found the concept of zero difficult to handle too. In fact some still do!

(1) Concatenation.

$$WP.(R;S).Q \quad \equiv \quad WP.R.(WP.S.Q)$$

Concatenation is used to build a sequence of statements. Some implementation languages also use ";" as a statement terminator. Please do not get confused by them. For us ";" has a precise meaning.

We note that ";" is an algebraic operator. It has some nice properties

$$(A; B); C \quad = \quad A; (B; C) \qquad \text{"; is associative"}$$

$$A; skip \quad = \quad A \qquad \text{"skip is the identity of ;"}$$

(2) Assignment.

$$WP.(x := E).Q \quad \equiv \quad Q(\text{all } x \text{ replaced by } E)$$

And we note $\{P\}\ x := E\ \{Q\}$

means $P \Rightarrow WP.(x := E).Q$

According to this definition we can calculate the weakest precondition of an assignment by textual substitution. We also allow for what is called parallel assignment, where we can have more than one variable on the left of the ":=" for example

$x, y := x+1, y-2$

Or even

$x, y := y, x$

(3) Selection (if..fi).

$\{P\}$
if B0 → S0
[] B1 → S1
fi
$\{Q\}$

means

$P \Rightarrow B0 \vee B1$
$\wedge$
$\{P \wedge B0\}\ S0\ \{Q\} \wedge \{P \wedge B1\}\ S1\ \{Q\}$

So the precondition must imply at least one of the guards is true and both $\{P \wedge B0\}\ S0\ \{Q\}$ and $\{P \wedge B1\}\ S1\ \{Q\}$ are valid Hoare triples.

We are not limited to 2 branches, we can have as many guarded branches Bi → Si as we need.

Unlike most implementation languages, there is no ordering in evaluating the guards. Each of the guards are evaluated at the same time and from those which evaluate to true, one is selected and the corresponding statement is performed. For example

{x, y have integer values}
if $x \leq y \rightarrow z := x$
[] $y \leq x \rightarrow z := y$
fi
$\{z = x \uparrow y\}$

This little program fragment establishes that z is the maximum of x and y. In the case that x = y we do not know which branch was performed, and we do not care.

(4) Repetition (do..od).

{R}
do B → S od
{Q}

means

$R \Rightarrow P$
$\wedge$
$\{P \wedge B \wedge vf = VF\}\ S\ \{P \wedge vf < VF\}$
$\wedge$
$P \wedge B \Rightarrow 0 < vf$
$\wedge$
$P \wedge \neg B \Rightarrow Q$

Where P is called the **loop invariant** and vf is an integer function, which is bounded below by 0, called the **variant,** which you can think of as the amount of work still to do.

Going through this line by line, we require that

The precondition implies the truth of the loop invariant

Each execution of the loop body decreases the variant vf but maintains the truth of P.

While the loop invariant and the guard are true we have not finished

The loop invariant is true and the guard is false implies the postcondition

In our work it is more usual for us to use the following template

"establish P"
{P}
do B → {P ∧ B}

"Decrease vf and maintain P"

{P}
od
{P ∧ ¬B}

Conclusions.

So that is our simple language. We are almost ready to begin. But not quite yet. We need to learn some notation which will allow us to specify our problems. We will use standard logical notation and some notation to express mathematical ideas. Sadly, a lot of mathematical notation is simply a shorthand, it was designed to be interpreted rather than manipulated. We require a notation which has nice manipulation laws so that we can let the notation do a lot of the work for us. Soon we will introduce some nice notation that does just that.

But before we do so, we will present an informal example which will help you to understand the concepts introduced in this chapter.