Chapter. The Guarded Command 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 ≡ 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 ≡ 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	=	A; (B; C)	"; is associative"
A; skip	=	A	"skip is the identity of ;"

2) Assignment.

WP.(x := E).Q ≡ Q(all x replaced by E)

So, to calculate the Weakest Precondition of the assignment x := E to achieve the state Q we simply copy Q and then replace every occurrence of the variable x with the expression E.

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

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

This shows the link between Hoare Triples and Weakest Preconditions.

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
```

The number of expressions on the right of the := symbol must equal the number of variables on the left side. The corresponding variables and expressions must have the same types.

(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 $x \leq y \rightarrow z := y$
[] $y \leq x \rightarrow z := x$
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 $B \rightarrow S$ od
{Q}
```

The textbook definition of what it means is as follows

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

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

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. So at the start of an iteration of the loop body the variant vf has some particular value that we call VF and at the end of the loop body it must have a value that is smaller than the value at the start

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

The loop invariant is true and the guard is false implies Q, the postcondition of the loop

Note that we have a single statement to express repetition and not the multitude which normally occur in implementation languages. All the repetitions we might ever wish to write can be written using the do..od statement.

The above is the full formal definition of the repetition which you will find in textbooks. However, when constructing repetitions we more commonly just use the following template to guide us. This template is worth remembering.

```
“establish P”
$\{P\}$
do B $\rightarrow$ $\{P \wedge B \wedge vf = VF\}$

                    “Decrease vf and maintain P”

            $\{P \wedge vf < VF\}$
od
$\{P \wedge \neg B\} \Rightarrow \{Q\}$
```