## Week 8 Review.

I am going to set an assignment this week so before that let us have a quick review of where we are in the material.

**Specification**.

Before we start to construct a program we need to know exactly what we want the program to do. The way we write what it is to do is called a Specification. A Specification has 2 parts, a Precondition (what we can assume before we start) and a Postcondition (what we want to be true when the program finishes).

In general, the Postcondition is the more important part and it usually guides us in constructing the program. The Precondition often is much simpler.

**Notation**.

We have learned a new notation called the Quantified notation. This allows us to describe using a binary operator on a collection of values (note the binary operator needs to be associative and have an identity value). Expressions written in the Quantified notation have a number of rules for manipulation which are important to learn. The ones we need to know about are the following.

*The empty range law (allowed because ⊕ has an identity element)*

$$
\begin{array}{ll}
 & \langle \oplus j : 0 \le j < 0 : f.j \rangle \\
= & \quad \{\text{empty range}\} \\
 & ID_{\oplus}
\end{array}
$$

*The one-point rule*

$$
\begin{array}{ll}
 & \langle \oplus j : j = i : f.j \rangle \\
= & \quad \{\text{one-point}\} \\
 & f.i
\end{array}
$$

*The split off a term law (allowed because ⊕ is associative)*

$$
\begin{array}{ll}
 & \langle \oplus j : 0 \le j < i+1 : f.j \rangle \quad \text{where } 0 \le i < N \\
= & \quad \{\text{split off } j = i \text{ term}\} \\
 & \langle \oplus j : 0 \le j < i : f.j \rangle \ \oplus \ f.i
\end{array}
$$

*The Strengthening law.*

$$
\begin{array}{ll}
 & \langle \oplus j : 0 \le j < i : f.j \rangle \ \wedge \ i = N \\
\Rightarrow & \quad \{\text{Leibniz}\} \\
 & \langle \oplus j : 0 \le j < N : f.j \rangle
\end{array}
$$

## Guarded Command Language.

We learned a very simple language called the Guarded Command Language which we will use for constructing and expressing our programs. It has the following commands

*(0) Skip.*

WP.skip.Q ≡ Q

*(1) Concatenation.*

WP.(R;S).Q ≡ WP.R.(WP.S.Q)

*2) Assignment.*

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

*(3) Selection (if..fi).*

{P}
if B0 → S0
[] B1 → S1
fi
{Q}

*(4) Repetition (Do..Od)*

"establish P"
{P}
do B → {P ∧ B ∧ vf = VF }

"Decrease vf and maintain P"

{P ∧ vf < VF}
od
{P ∧¬B}

You can read the details of these in your notes.

**Method of Program Construction.**

We have learned a method which we can use to construct programs. The method has a set of standard steps. In some cases not all of the steps will be required, however, I am listing all of them so you can learn them.

(0) Write Specification. Precondition and Postcondition.

(1) Strengthen Postcondition to introduce this shape "∧"

(2) Model the problem domain

(3) Rewrite Postcondition using model

(4) Choose Invariants

(5) Choose loop guard

(6) Establish Loop Invariants

(7) Choose Variant vf

(8) Loop body (decrease variant and keep invariants true)

**Problem types.**

We used this method on a number of different problems and we began to see patterns in both the Postconditions and the finished algorithms. This led us to talk about problem categories. So far we have seen 4 such problem categories. Their postconditions have different shapes.

When we write a postcondition for a new problem we should look at the shape and check to see if it matches one of our problem categories. If it does, then we immediately know how to solve it. The following are the shapes we have seen so far.

*Reductions.*

Post: $r = \langle \oplus j : \alpha \le j < \beta : f.j \rangle$

*Partitioned Reductions.*

Post: $r = \langle \oplus j : \alpha \le j < \beta : g.(f.j) \rangle$

where

$$\begin{array}{llll} g.x & = & h.x & \Leftarrow \quad Q.x \\ g.x & = & Id \oplus & \Leftarrow \quad not.(Q.x) \end{array}$$

*Linear Searches.*

Pre: $\langle \exists j : \alpha \le j < \beta : \neg Q.(f.j) \rangle$

Post: $\langle \forall j : \alpha \le j < n : \neg Q.(f.j) \rangle \wedge Q.(f.n)$

*Bounded Linear Searches.*

There are 2 important variations of the Bounded Linear Search. We illustrate them below. Given a finite, non-empty, ordered domain f[α..β) and a predicate Q defined on the elements of f.

Does Q hold true *everywhere*

$$\langle\, \forall\, j : \alpha \le j < i : Q.(f.j) \,\rangle \wedge ((\neg Q.(f.i) \;\wedge \text{"no it doesn't"}) \vee (\, Q.(f.i) \;\wedge i = \beta - 1 \wedge \text{"yes it does"} \;))$$

Does Q hold true *anywhere*

$$\langle\, \forall\, j : \alpha \le j < i : \neg Q.(f.j) \,\rangle \wedge ((Q.(f.i) \;\wedge \text{"yes it does"}) \vee (\, \neg Q.(f.i) \;\wedge i = \beta - 1 \wedge \text{"no it doesn't"} \;))$$