Chapter. Calculating Programs.

In this chapter we introduce a style of programming which is called program calculation. I want you to pay particular attention to the steps which we perform, you will find these steps are involved in many of the programs we will construct from now on. Essentially, we are describing **a method of programming**.

We will use the familiar example of summing the values in f[0..N) of int.

Write the postcondition

Post : r = $\langle + j : 0 \leq j < N : f.j \rangle$

Model the problem domain.

We name the quantified expression. For n in the range, $0 \leq n \wedge n \leq N$ we define

* (0) C.n = $\langle + j : 0 \leq j < n : f.j \rangle$, $0 \leq n \leq N$

As all of the operators we use in Quantified expressions are associative and have identities we use these properties to develop some theorem about the problem domain.

Observe.

```
        C.0
=               { by definition (0) }
        ⟨ + j : 0 ≤ j < 0 : f.j ⟩
=               { empty or false range }
        Id+
```

Which establishes the following theorem.

\- (1) C.0 = Id+

Observe.

```
        C.(n+1)
=               { by definition (0) }
        ⟨ + j : 0 ≤ j < n +1 : f.j ⟩
=               { split off j = n term }
        ⟨ + j : 0 ≤ j < n : f.j ⟩ + f.n
=               { by definition (0) }
        C.n + f.n
```

Which establishes the following theorem.

- (2) C.(n+1) = C.n + f.n , $0 \leq n < N$

Rewrite the postcondition in terms of the model.

Post : r = C.N

Using Strengthening[1] we can rewrite this to get the right shape to construct a loop.

Post' : r = C.n ∧ n = N

Invariants.

P0 : r = C.n
P1 : $0 \leq n \wedge n \leq N$.

Establish invariants.

This is done by looking at our model. Law (1) tells us that if the argument of C is 0 the the value of C.n is Id+. Setting n to 0 also establishes P1.

n, r := 0, Id+

Guard.

$n \neq N$

Variant.

N-n

Loop body.

We know that a standard way to decrease the variant is to increase n by 1. Now we calculate the assignment to r which will keep P0 true.

(n, r := n+1, E). P0
= { text substitution }
E = C.(n+1)
= { (2) from model }
E = C.n + f.n
= { P0 binds C.n to a variable }
E = r + f.n

---

[1] We could have done this strengthening step just after writing the initial postcondition. In some other examples I will do so.

So now we know what value to assign to r so as to keep P0 true as we decrease vf.

Finished algorithm.

$$
\begin{array}{l}
\quad n, r := 0, Id+ \\
\quad ; \text{do } n \neq N \rightarrow \\
\qquad\quad n, r := n+1, r + f.n \\
\quad \text{od} \\
\quad \{ r = C.n \wedge n = N \}
\end{array}
$$

An aside on variants and how to decrease them.

Note that if we start n at 0 and it finishes at N then a suitable variant (vf) would be N– n. We now want to construct the loop body. This involves decreasing the variant. How shall we do so? Certainly we will have to change n, but by how much and why?

The invariant P1 : $0 \leq n \ \wedge n \leq N$ is there to ensure that when we refer to f.n in our loop we never breach the array bounds. So changes to n must maintain P1. We calculate how it might be maintained within the loop body. We do this with the following little calculation.

$$
\begin{array}{ll}
 & 0 \leq n \ \wedge n \leq N \wedge n \neq N \\
= & \quad \{ \text{arithmetic} \} \\
 & 0 \leq n \ \wedge n < N \\
= & \quad \{ \text{arithmetic} \} \\
 & 0 \leq n \ \wedge n+1 \leq N \\
\Rightarrow & \quad \{ \text{arithmetic} \} \\
 & 0 \leq n+1 \wedge n+1 \leq N \\
= & \quad \{ \text{WP} \} \\
 & \text{WP.}(n := n + 1). (0 \leq n \ \wedge n \leq N)
\end{array}
$$

This justifies how we decrease the invariant. At this point we have shown how to establish P1 and to maintain it within the loop. In future, where we P1 is similar to this and where we are decreasing it like this, I will not bother to include this standard calculation in the text of the example.

End of aside.