Chapter 6: Calculating Programs.

In this chapter we introduce a style of programming which is called program calculation. What distinguishes this approach is that it deliberately tries to separate the modelling or understanding of the problem domain from the programming task. We will use the familiar example of summing the values in f[0..N) of int.

Write the postcondition

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

Strengthen the postcondition (if necessary)

We need to strengthen our postcondition to get the shape we usually require for making a loop.

Post' : $r = \langle + j : 0 \le j < n : f.j \rangle \wedge n = N$

Model the problem domain.

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

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

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.

Consider.

C.0

= { by definition (0) }

$\langle + j : 0 \le j < 0 : f.j \rangle$

= { empty or false range }

Id+

Which establishes the following theorem.

- (1) C.0 = Id+

Consider.

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 , for $0 \le n < N$

This completes our little domain model.

Rewrite the postcondition in terms of the model.

Post : r = C.N

Using Strengthening we can rewrite this as

Post' : r = C.n ∧ n = N

This is now the right shape for us to construct a loop program.

Invariants.

We choose as invariants

P0 : r = C.n

P1 : $0 \le n \wedge n \le N$.

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.

If we begin n at 0 and finish when n is equal to N, then a good choice for invariant is

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

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

Final program.

n, r := 0, Id+

; do n ≠ N -->

n, r := n+1, r + f.n

od

{ r = C.n ∧ n = N }