The Reduction Theorem.

We choose our notation so that it can do some of the work for us; we just have to manipulate the expressions according to a small set of rules. But our notation has another major benefit, it allows us to identify and develop solutions which can be reused and applied in lots of ways.

It is always a good idea as a programmer, to look at every new solution you create and to see if it is possible to abstract from it to get a more useful generic solution. In this way you will build up a useful "toolbox" of generic reusable solutions. Let us see how we might do so.

Consider the postconditions for the following problems.

Compute the sum of the values in f[0..N) of int.

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

Compute the product of the values in f[20..100) of int.

$$p = \langle * j : 20 \le j < 100 : f.j \rangle$$

Determine the largest value in the array f[0..200) of int.

$$l = \langle \uparrow j : 0 \le j < 200 : f.j \rangle$$

Now, if we look at the shape of each of these postconditions, we notice that they are quite similar; they are all instances of a more abstract shape

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

Where $\oplus$ is an associative binary operator which has an identity value and α and β are the lower and upper bounds on the range.

Model the problem domain.

$$* \ (0) \ C.n = \langle \oplus j : \alpha \le j < n : f.j \rangle \quad , \alpha \le n \le \beta$$

Consider

$$\begin{aligned} & C.\alpha \\ = & \quad \{(0)\} \\ & \langle \oplus j : \alpha \le j < \alpha : f.j \rangle \\ = & \quad \{ \text{Empty Range} \} \\ & Id \oplus \end{aligned}$$

Which gives us

$$- \ (1) \ C.\alpha = Id \oplus$$

Consider

```
        C.(n+1)
=               {(0)}
        ⟨ ⊕ j  : α ≤ j < n+1 : f.j ⟩
=               { Split j = n term }
        ⟨ ⊕ j  : α ≤ j < n : f.j ⟩ ⊕ f.n
=               {(0)}
        C.n ⊕ f.n
```

Which gives us

- (2) C.(n+1) = C.n ⊕ f.n ,α ≤ n < β

Rewrite the postcondition using the model.

Post : r = C.β

Strengthen the postcondition.

Post : r = C.n ∧ n = β

Choose invariants.

P0: r = C.n
P1: α ≤ n ≤ β

Guard.

n ≠ β

Establish invariants.

n, r := α, Id⊕

Variant.

β - n

Loop body.

```
        (n, r := n+1, E).P0
=               { Text Substitution }
        E = C.(n+1)
=               {(2)}
        E = C.n ⊕ f.n
=               {P0}
        E = r ⊕ f.n
```

Finished algorithm.

```
n, r := α, Id⊕
;do n ≠ β →
        n, r := n+1, r ⊕ f.n
od
```

This is the generic shape of all reductions.