Chapter 21: Transforming Recursion to Iteration (loop programs).

Often we are asked to construct a program to compute the value of some function, where the function definition has a shape like this.

```
* (0) f.n        =    c.n              <=    Not.(b.n)
* (1) f.n        =    h.n ⊕ (f.(g.n))  <=    b.n
```

Aside.

e.g. Sum the digits in an integer.

```
f.n    =    0                          <=    n=0
f.n    =    (n mod 10)  +  f.(n div 10)    <=    n <> 0
```

End of aside.

Given that we don't allow recursion in the guarded command language we need to find a way to construct a loop program to compute f.N

We propose using the following invariants.

Invariants.

P0: $r \oplus f.n = f.N$
P1: $0 \leq n \leq N$

Establish P0 & P1

r, n := Id ⊕, N

Termination.

consider

```
        P0  & Not.(b.n)
=               {P0 }
        r ⊕ f.n = f.N & Not.(b.n)
=               {(0)}
        r ⊕ c.n = f.N
```

So we can compute f.N in this case without any recursion.

This suggests the following

Guard.

b.n

Vf.

In the particular case we need to satisfy ourselves that b.n will eventually become false.

Loop body.

P0
= {defn.}
r ⊕ f.n = f.N
= { guard b.n, (1) }
r ⊕ ((h.n) ⊕ (f.(g.n))) = f.N
= { is associative }
(r ⊕ (h.n)) ⊕ (f.(g.n)) = f.N
= { WP.}
(r, n := (r ⊕ (h.n)), g.n).P0

Finished Algorithm.

r, n := Id ⊕, N
;do b.n —>
r, n := (r ⊕ (h.n)), g.n
od
; r := r ⊕ c.n
{ r = f.N }