Programming Calculus

“Proof of Correctness”
Dijkstra and Gries

Notation

• Assignment \( \{P[e \to x]\} \ x := e \ {P} \)
• Composition
  \( \{P_1\} S_1 \ {P_2}, \ {P_2} S_2 \ {P_3} \)
  \( \{P_1\} S_1 S_2 \ {P_3} \)
• Alternation B_1 or … or B_n
  \( \{P ^ B_1\} \ \text{Si} \ \{R\} \ 1 \leq i < n \)
  \( \{P\} \ \text{if} \ B_1 \to S_1 \ldots B_n \to S_n \ \text{fi} \ \{R\} \)
  -one Bi must be true, Si is executed
Notation 2

- Iteration \{P \text{ and } B\} S \{P\}
  
P is invariant relation (does not change)
  \{P \land B \land t \leq t_0 + 1\} S \{t \leq t_0\}
  S must reduce \( t \) by at least 1
  \(P \land B \Rightarrow t > 0\)
  \{P\} do B \rightarrow S or \{P \land \lnot B\}
  When \( t \leq 0 \) then \( B \leftarrow \text{false} \)

Notation 3

\(WP (S,R)\)

Statement \( S \) – give a rule for developing the weakest precondition for which execution of \( s \) will established the post condition \( R \)

- must derive the precondition \( P \) from \( S \) and \( R \)
- \( S \) is a predicate transformer which transforms \( P \) to \( R \)
• Step 1 – Input and Output Assertions
  – pre and post condition
• Step 2 – Loops
  – develop invariant relation
  – initialize, loop body, termination
• Step 3 – Think in terms of “While” loop not For
• Step 4 – Work backwards – start at post condition
• Step 5 – Limit nested statements
• Step 6 – Document program

Let Bi be initial values in b[i]
Input assertion
(3) s ≥ 0 ^ b[i] = Bi 1 ≤ i ≤ n

Form of line (w1 – word 1)

Output form
Restrictions $p, q, t$

$1 \leq t \leq n$

$p \geq 0, q \geq 0$

$p(t-1) + q(n-t) = s$

$(5) \ ((\text{odd } z) \land q=p+1) \lor ((\text{even } Z) \land p=q+1)$

from 3

$(6) \ \{ \begin{array}{ll}
    b[i] = B_i + p(i-1) & 1 \leq i \leq t \\
    b[i] = B_i + p(t-1) + q(i-t) & t < i \leq n \\
\end{array} \}$

post condition – 6

Algorithm

$\{ (3) \} \$

Calculate $p, q, t$ from $(5)$

$\{ (3) \land (5) \}$

Calculate $b[1:t]$ from $(6)$

$\{ (5) \land b[i] = B_i + p(i-1) \ 1 \leq i \leq t \}$

$\land b[i] = B_i \ t < i \leq n$

Calculate new $b[t+1:n]$ from $(6)$

$(7) \ \{ (5) \land (6) \}$
Algorithm 2

Refine Calculations b[1:t]
• need loop counter k
• invariant relation P1
P1: 1 \leq k \leq t\; \text{incr} = p \cdot (k-1) (5)

\[ \text{b}[i] = \text{B}_i + p \cdot (i-1) \quad 1 \leq i \leq k \]
(8) \quad \text{b}[i] = \text{B}_i \quad k < i \leq n

Algorithm 3

I initial k = 1; incr = 0
loop S: k=k+1; incr = incr + P; b[k] = b[k] + incr
Choose B \in P1 \land B \text{ is postcondition}
\neg B \equiv \text{incr} = p \cdot (t-1) \quad \text{since}
use B \equiv \text{incr} < p \cdot (t-1) \quad \text{p may be 0}
loop:
{calculate b[1:t] from (6)}
k := 1
incr = 0
While incr <> p * (t-1)
    k := k + 1
    incr := incr + p
    b[k] := b[k] + incr

P2  t < k ≤ n  incr = p * (t-1) + q * (k-t)  (5)
    b[i] = Bi + p * (i-1)  1 ≤ i < t
    b[i] = Bi + p * (t-1) + q * (i-t)  t < i ≤ k
    b[i] = Bi  k < i ≤ n

{Calculate b[t+1:n]}
loop2  k = t
While k <> n do
    k := k + 1
    incr := incr + q
    b[k] := b[k] + incr

{Calculate p,q,t}
even (z) → p = q + 1
(q+1) * (t-1) + q * (n-t) = s  (5)
0 ≤ t-1 ≤ n-1

    q = s div (n-1)
(10) \{  
    t = s + 1 - q * (n-1)
    p = q + 1
\}
Derive Wp execution of (10) will establish

\[ \text{even } (z) \wedge s \geq o \wedge (s \div (n-1)) \geq 0 \]

(must have at least 2 words)

if \( n \leq 1 \rightarrow \text{skip} \)
else if \( n > 1 \rightarrow \text{if even } z \rightarrow \)
\[
q = s \div (n-1) \\
t = s + 1 - q \ast (n-1) \\
p = q + 1
\]

if (odd z) \rightarrow
\[
p = s \div (n-1) \\
q = p + 1 \\
t = p \ast (n-1) + n - s
\]

loop 1
loop 2
fi