EFFICIENCY (1)

1. Try to reduce variant function as quickly as possible.
2. On a binary computer $\dot{-}2$ and mod 2 are efficient.

EXAMPLE

Using only + - *2, $\dot{-}2$ and mod 2, not changing X, Y, write a program

that achieves $z = X*Y$ .... r

choose invariant $p = X*Y = x*y + z$

$\bar{b} = y = 0$

variant y

check $p \cap \bar{b} \Rightarrow r$, $x,y,z := X, Y, 0$ a p

try $y := y \dot{-} 2$

check $b \cap p \Rightarrow y := y \dot{-} 2$ dec y

we need x' s.t $x*y + z = x' *(y \dot{-} 2) + z$

try $x' = x*2$. This works if $y \bmod 2 = 0$

If $y \bmod 2 \neq 0$, we use the usual slow method

$x,y,z := X,Y,0$ a p
do $y \neq 0 \rightarrow$ if $y \bmod 2 = 0 \rightarrow x := x*2;\ y := y \dot{-} 2,$

$z := z + x;\ y := y - 1$

fi
od

EXAMPLE (2)

Using only +1, -1, *2, $\dot{-}2$, mod 2, and not changing $X,Y > 0$ write a

program that achieves $q = X \dot{-} Y,\ x = X \bmod Y$

choose $p \triangleq X = q* dd + x \ \&\ dd = Y*2^n > x$

$\bar{b} \triangleq dd = Y$

$t \triangleq dd$

note $p \cap b \Rightarrow dd := dd \dot{-} 2$ dec dd

to reestablish $X = q*dd + x$ we also need.

n := n - 1; q := q * 2; make $x < dd$,

keeping $X = q * dd + x$

changing x & q only.

but we already know how to do that!

It remains only to establish p.

choose $p' = X = q*dd + x \;\&\; dd = Y * 2^n$

$\overline{b}' = dd > x$ $\qquad t = 2*x - dd$

clearly (q := 0; x := X; dd := Y; n := 0) a p'

$p' \cap b' \Rightarrow$ (dd := dd*2; n := n+1) a p'

" dec $2*x - dd$

SOLUTION (3)

$\{ Y>0 \;\&\; X \geq 0 \}$

q, x, dd := 0, X, Y;

n := 0; $\{X = q*dd + x \;\&\; dd = Y*2^n \quad \ldots p^1\}$

do $dd \leq x \rightarrow$ dd := dd*2; n := n+1 od; $\{p^1 \;\&\; dd > x\}$

do $dd \neq Y \rightarrow$ n := n - 1; q := q*2; dd := dd $\div$ 2; $\{p^1\}$

do $x \geq dd \rightarrow x := x - dd;\ q := q+1$ od $\{p^1 \;\&\; dd > x\}$

od $\{p^1 \;\&\; dd > x \;\&\; dd = Y \ldots\} \Rightarrow \{X = q*Y + x \;\&\; Y > x\}$

note. all operations on n can be omitted.

EXERCISE (1) same as above, but using $\div$ 10 and *10 instead of $\div$ 2 and *2

(2) Using *, $\div$ 2, mod 2, write a program that achieves $z = X^Y$.

SQUARE ROOT (4)

Write a program that assigns to a an approximation to the square root of $n \geq 0$

postcondition : $\underbrace{a^2 \leq n}_{p} \underbrace{< (a+1)^2}_{\overline{b}}$

note $a := 0$ $\underline{a}$ p

note $b = (a+1)^2 \leq n \Rightarrow a := a + 1$ $\underline{a}$ p

$\underline{dec}$ $n - a^2$

$\therefore$ $a := 0$; $\underline{do}$ $(a+1) * (a+1) \leq n \Rightarrow a := a + 1$ $\underline{od}$

now make this a bit faster.

postcondition: $p = a^2 \leq n < (a+c)^2$ & $c = 2^i$

$\overline{b} \equiv c = 1$ variant = c

$p \cap b \Rightarrow c := c \div 2;\ i := i - 1$ $\underline{dec}$ c

to restore, if $(a+c)^2 \leq n$ we must increase a in just this case,

$a := a + c$ will do it!

$\{n \geq 0\}$

$a := 0;$ $\boxed{i := 0;}$ $c := 1;$ (5)

$\{a^2 \leq n$ & $c = 2^i\}$ ... p^0

$\underline{do}$ $c^2 \leq n \rightarrow c := 2*c;$ $\boxed{i := i + 1}$ $\underline{od};$

$\{p^0$ & $n < (a+c)^2\}$ ... p

$\underline{do}$ $c \neq 1 \rightarrow c := c \div 2;$ $\boxed{i := i - 1}$

$\underline{if}$ $(a+c)^2 \leq n \rightarrow a := a + c,$ skip $\underline{fi}$

$\{p\}$

$\underline{od}$ $\{p \cap c = 1\}$

$\{a^2 \leq n < (a+1)^2\}$

$\boxed{\text{operations on } i \text{ can be omitted}}$

check that the invariant p is preserved.

$p \cap c \neq 1 \Rightarrow \underline{if}$ $(a + c \div 2)^2 \leq n \rightarrow (a + c \div 2)^2 \leq n < (a + c \div 2 + c \div 2)$,

$a \leq n < (a + c \div 2)$

$\underline{fi}$

Exercise. Rewrite the square root program using $*4$, $*2$, $\div 4$, $\div 2$ instead of squaring. we have to get rid of $c^2 \leq n$ and $(a + c)^2 \leq n$ (6)

$\therefore$ we introduce $q = c^2$, $p = a*c$, $r = n - a^2$ I

thus $c^2 \leq n \equiv q \leq n$

$(a + c)^2 \leq n \equiv 2*p + q \leq r$

To preserve the invariant I, we must accompany

$c := 2*c$ by $q := 4*q$; $p := p*2$

$c := c \div 2$ by $q := q \div 4$; $p := p \div 2$

$a := a + c$ by $p := p + q$; $r := r - (2*p + q)$

$a := 0$ by $p := 0$; $r := n$

$c := 1$ by $q := 1$

also $q = 1 \equiv p = a$

$c = 1 \equiv q = 1$

$\boxed{a := 0;}\ p := 0;\ r := n;$

$\boxed{c := 1;}\ q := 1;$

$\underline{do}\ q \leq n \rightarrow \boxed{c := 2*c;}\ q := 4*q;\ p := 2*p\ \underline{od}\ ;$

$\underline{do}\ q \neq 1 \rightarrow \boxed{c := c \div 2;}\ q := q \div 4;\ p := p \div 2;$

$\underline{if}\ 2*p + q \leq r \rightarrow \boxed{a := a + c;}\ p := p + q;$

$r := r - (2*p + q),$

skip

$\underline{fi}$

$\underline{od}\ \{p = a\}$

$a := p$

all the operations of the original program can be omitted!

They are operations on "abstract" or "ghost" variables, which have been more efficiently represented by the "concrete" variables p,q,r.

We take lower subscript bound always zero.*-

If T is a type, as is T*

(the set of finite sequences whose items are in T).

Axioms:

(1) T* empty is a T* (abbreviated to $\langle\rangle$)

(2) If a is a T* and x is a T then

$a.^\frown x$ is a T* (a extended by x)

(3) These are all the elements of T*

(i.e. T* is least set closed wr to (1) and (2))

(4) $a.^\frown x = b.^\frown y \iff a = b \;\&\; x = y.$

$\langle\rangle.^\frown x.^\frown y.^\frown \ldots .^\frown z$ is abbreviated to $\langle x, y, \ldots, z\rangle$

dot associates to left, i.e. $a.^\frown x.^\frown y = (a.^\frown x).^\frown y$

*-This is simpler than Dijkstra's treatment

"shift" is not relevant, and "lob" is redundant.

In the following, (9)

$a,b : T^*;$

$x,y : T;$

$i,j : NN;$ (natural number)

LENGTH : $\#\langle\rangle = 0$ $\#(a.^\frown x) = \#a + 1$

HIGH: $(a.^\frown x).\,high = x$ $\langle\rangle.\,high$ is not defined.

SUBSCRIPT: $(a.^\frown x)(\#a) = x$ $\langle\rangle(i)$ is not defined.

$i < \#a \Rightarrow (a.^\frown x)(i) = a(i)$

ALTER: $(a.^\frown x).\,alt(\#a,y) = a.^\frown y$ $\langle\rangle.alt(i,y)$ is not defined.

$i < \#a \Rightarrow (a.^\frown x).alt(i,y) = a.alt(i,y).^\frown x$

SWAP: $a.swap(i,j) = a.alt(i,a(j)).alt(j,a(i))$

PREFIX: $x \hat{}. \langle\rangle = \langle\rangle. \hat{} x$

$x \hat{}.(a. \hat{} y) = (x \hat{}.a). \hat{} y$

REMAINDER: $(a. \hat{} x).\text{hirem} = a$ $\quad\quad \langle\rangle$. hirem is not defined

$(x \hat{}.a).\text{lorem} = a$ $\quad\quad \langle\rangle$. lorem is not defined

CATENATE: $a \hat{} \langle\rangle = a$ (10)

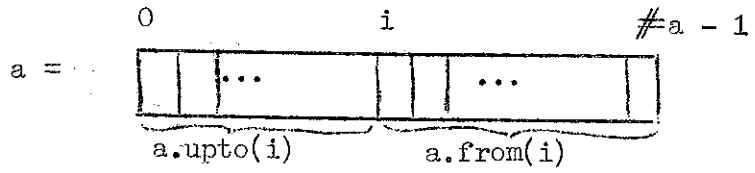
$a \hat{} (b. \hat{} x) = (a \hat{} b). \hat{} x$

INITIAL SEG: $a.\ \text{upto}(0) = \langle\rangle$

$a.\ \text{upto}(i+1) = a.\ \text{upto}(i). \hat{} a(i)$

FINAL SEG: $a.\ \text{from}(0) = a$

$a.\ \text{from}(i+1) = a.\ \text{from}(i).\text{lorem}$



THEOREMS: $(a \hat{} b) \hat{} c = a \hat{} (b \hat{} c)$. $\quad a.\text{upto}(\#a) = a$

$i < \#a \Rightarrow (a.\text{upto}(i)) \hat{} (a.\text{from}(i)) = a$

ABBREVIATIONS: a:(i) := e $\quad$ for a := a.alt(i,e)

For any xxx, a: xxx(...) $\quad$ for a := a.xxx(...)

x,a: hipop $\quad$ for x := a.high; a:hirem

x,a:lopop $\quad$ for x := a(0); a:lorem

SORTED (11)

Given a type T and a function key: T $\rightarrow$ NN

we define key sorted: T* $\rightarrow \{ \underline{T}, \underline{F} \}$

$a.\ \text{keysorted} = \forall i,j:NN\ (i \leq j < \#a \Rightarrow a(i).\text{key} \leq a(j).\text{key})$

= **if** $\#a \leq 1 \rightarrow \underline{T}$

, $a(0).\text{key} \leq a(1).\text{key} \rightarrow a.\text{lorem}.\ \text{keysorted}$

, $\underline{F}$

fi

$= \#a \leq 1$

$\forall \exists$ k:NN a. partitioned at (k)

& a.upto(k). keysorted

& a. from(k). keysorted

where a. partitioned at (k) $= 0 \leq k < \#a$

$\& \forall i,j:NN\ i<k\leq j<\#a \Rightarrow a(i).key \leq a(j).key$

If T = NN and $\forall$ i i.key = i,

we write a.sorted for a.keysorted

SEARCHING (12)

Precondition $f(0) \leq x < f.high$ ... precond

find m s.t $f(m) \leq x < f(m+1)$ ... result

introduce n:NN; define

$p = f(m) \leq x < f(n) \quad \& \ m<n$

$\overline{b} = n \leq m+1$

$t = n - m$

check that $p \ \& \ \overline{b} \Rightarrow$ result

precond $\Rightarrow$ (m := 0; n := #f - 1) wp p

we must find C s.t. $p \ \& \ b \Rightarrow C$ wp $p \ \& \ C$ dec t

obviously, we must change n or m. $\therefore$ try (for arb.d).

$p \ \& \ b \Rightarrow$ (m := d wp p) & (m := d dec t)

i.e. $p \ \& \ b \Rightarrow f(d) \leq x \ \& \ m<d<n$

similarly $p \ \& \ b \Rightarrow$ (n := d wp p) & (n := d dec t)

reduces to $p \ \& \ b \Rightarrow x<f(d) \ \& \ m<d<n$

$\therefore$ C = setd; if $f(d) \leq x \rightarrow m := d$

$[] \ x<f(d) \rightarrow n := d$

fi

where $p \;\&\; b \Rightarrow$ setd $\underline{wp}$ $m < d < n$ (13)

and setd changes only d, and b is $n > m + 1$

ALTERNATIVES:

setd = $d := m + 1$

$d := n - 1$

$d := (m + n) \div 2$

the third alternative is most efficient.

given f is sorted & $\exists\, m$ $f(m) = x$ & precondition

prove that our algorithm achieves the result $f(m) = x$.

what happens when several values of m satisfy $f(m) = x$?

MERGING (14)

z. merge $(<>,<>) \equiv z = <>$

z. merge $(x,y) \equiv$ z. hiext(1).merge (x.hiext(1), y) ...(1)

z. merge(x,y) $\equiv$ z. merge (y,x)

Following Dijkstra, we define a.dom = #a, a.hiext(x) = $a.\hat{}x$

Write a program (not changing X,Y) with preconditions X.sorted and Y.sorted and postcondition Z.sorted & Z.merge (X,Y)

introduce variables m,n and split postcondition

$p = m \leq X.dom \;\&\; n \leq Y.dom \;\&$ Z.sorted &

Z.merge (X.upto(m), Y.upto (n))

$\overline{b} = m = X.dom \;\&\; n = Y.dom.$

variant expression : X.dom + Y.dom - m - n.

consider:

$(b \;\&\; p \Rightarrow m := m + 1 \;\underline{wp}\; p) \equiv b \;\&\; p \Rightarrow m + 1 \leq X.dom$

& Z.merge(X.upto (m).hiext (X(m)), Y)

upto (m + 1)

the last clause indicates that Z must be extended by X(m) see(1)

Try again (15)

b & p $\Rightarrow$ (Z : hiext (X(m)); m := m + 1 wp p)

i.e. b & p $\Rightarrow$ Z. hiext (X(m)).sorted & m + 1 $\leq$ X.dom

i.e. b & p $\Rightarrow$ (Z.dom = 0 $\vee$ Z.high $\leq$ X(m)) & m + 1 $\leq$ X.dom

introduce abbreviation Z.ok(1) = Z.dom = 0 $\vee$ Z.high $\leq$ 1 and construct guarded commands:

m < X.dom & Z.ok(X(m)) $\rightarrow$ Z: hiext (X(m)); m := m + 1

n < Y.dom & Z.ok(Y(n)) $\rightarrow$ Z: hiext (Y(n)); n := n + 1

(second one is constructed by symmetry from the first)

BUT what do we do when the guards are false?

ALSO we haven't used the fact that X & Y are sorted.

SOLUTION strengthen invariant to (16)

new p = p & (m < X.dom $\Rightarrow$ Z.ok (X(m)))

& (n < Y.dom $\Rightarrow$ Z.ok (Y(n)))

b & new p $\Rightarrow$ Z: hiext (X(m)); m := m + 1 wp new p

= b & new p $\Rightarrow$ Z.ok (X(n)) & m + 1 $\leq$ X.dom (from b & p, as before) (1)

& (m + 1 < X.dom $\Rightarrow$ Z. hiext (X(m)).ok (X(m+1))) (2)

& (n < Y.dom $\Rightarrow$ Z.hiext (X(m)).ok (Y(n))) (3)

line (1) (copied from previous argument) reduces to m < X.dom

line (2) follows from Z.ok(X(m)) & X.sorted

line (3) reduces to n < Y.dom $\Rightarrow$ X(m) $\leq$ Y(n)

do m < X.dom & (n < Y.dom $\Rightarrow$ X(m) $\leq$ Y(n)) $\rightarrow$ Z.hiext(X(m)); m := m + 1

$\square$ n < Y.dom & (m < X.dom $=$ Y(n) $\leq$ X(m)) $\rightarrow$ Z.hiext(Y(n)); n := n + 1

od

This time, the negation of the conditions is.

m $\geq$ X.dom & n $\geq$ Y.dom, which, with p, implies our result.

Initialisation: X.sorted & Y.sorted $\Longrightarrow$

m := 0; n := 0; Z := T*.empty wp new p

0 $\leq$ X.dom & 0 $\leq$ Y.dom & T*.empty.sorted & T*.empty.merge(X.upto(0),Y.upto(0)) & ...

FINITE SETS (17)

If T is a data type, then so is T**

(the set of all finite subsets of T).

(1) T**.empty is a T** (abbreviated to $\{\}$)

(2) if t is a T** and x is a T, t.ext(x) is a T**

(3) all elements of T** are formed as described above.

$\neg\, x \notin T^{**}.empty, \quad x \in t.ext(x)$ $\quad \left\{ \begin{array}{l} \text{for } x,y \quad T \\ s,t \quad T^{**} \end{array} \right\}$

$x \neq y \Rightarrow (x \in t.ext(y) \equiv x \in t)$

(4) $\forall\, x(x \in t \equiv x \in s) \Longleftrightarrow t = s.$ "

$s \subseteq t \Longleftrightarrow \forall x(x \in s \Rightarrow x \in t)$ "

$s \cup T^{**}.empty = s, \quad s \cup t.ext(x) = (s \cup t).ext(x)$ "

$x \in (s \cap t) \equiv x \in s \;\&\; x \in t$ "

$x \in (s - t) \equiv x \in s \;\&\; \neg\, x \in t$ "

$T^{**}.empty.size = 0, \quad x \,\tilde{\in}\, s \Rightarrow s.ext(x).size = s.size + 1$

ABBREVIATIONS (18)

$$\{x_1, x_2, \ldots, x_n\} = T^{**}.empty.ext(x_1).ext(x_2) \ldots\ldots ext(x_n)$$

$x :\in t \;\underline{wp}\; r(x) = t \neq T^{**}.empty$

$\&\, \forall x(x \in t \Rightarrow r(x))$ (x not in t)

$x :\in \{e\} \equiv x := e$ (e defined).

FINITE MAPPINGS (19)

If D and R are data types, then so is $D \xrightarrow{Fin} R$

Fin will be omitted inconsistently! (D into R)

(1) $(D \rightarrow R).empty$ is a $D \rightarrow R$

(2) If a is a $(D \rightarrow R)$, d is D, r is an R, (extend

then $a.ext(d,r)$ is a $(D \rightarrow R)$ or alter)

(3) All elements of $(D \rightarrow R)$ can be formed as described above.

$(D \rightarrow R).empty.dom = D^{**}.empty$ $\quad dom: (D \xrightarrow{Fin} R) \rightarrow D^{**}$

$a.ext(d,r).dom = a.dom.ext(d)$

$a.ext(d,r).val(d) = r$ (not defined for $(D \rightarrow R).empty$)

$d' \neq d \Rightarrow a.ext(d,r).val(d') = a.val(d')$.

(4) $a.dom = b.dom \;\&\; \forall d\,(d \in a.dom \Rightarrow a.val(d) = b.val(d))$

$\Rightarrow a = b.$

$(D \rightarrow R).empty.del(d) = (D \rightarrow R).empty$

$d \neq d' \Rightarrow a.ext(d,r).del(d') = a.del(d').ext(d,r)$

$\neg d \in a.dom \Rightarrow a.ext(d,r).del(d) = a.$

PRODUCTS AND SUMS (20)

If $T_1, T_2, \ldots, T_n$ are types, then so are

$PROD = T_1 \times T_2 \times \ldots \times T_n$ (cartesion product)

$UNION = T_1 + T_2 + \ldots + T_n$ (discriminated union)

(1) $x_1 \in T_1, \ldots, x_n \in T_n \Rightarrow (x_1, x_2, \ldots, x_n) \in PROD.$ (constructor)

(2) all elements of PROD can be formed as in (1)

(3) $(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \Rightarrow x_1 = y_1 \;\&\; \ldots \;\&\; x_n = y_n$

Define $(x_1, \ldots, x_n).\, i = x_i$ for numeral i (selector)

(1) $(x_1 \in T_1 \Rightarrow UNION.1(x_1) \in UNION) \;\&\; \ldots \;\&\; (x_n \in T_n \Rightarrow UNION.n(x_n) \in UNION)$

(2) all elements of UNION can be formed as in (1)

(3) $UNION.i\,(x) = UNION.j(y) \Rightarrow i = j \;\&\; x = y \in T_i$

transaction = key X (amendment + deletion + insertion) (21)

given. master : key $\xrightarrow{Fin}$ record

tr : transaction*

report : error message*

error : transaction → error message

upd : (record X transaction) → record

content: transaction → record.

y. is insertion $=_{df}$ y.2.sort = 3

etc

$y.key = \bar{y}.1$

A random - access file updating program is:

report:= empty; y: transaction;

do tr. dom ≠ 0 → y, tr: lopop;

if y.key $\tilde{\in}$ master.dom & y. is insertion

→ master: ext (y.key, y.content)

, y.key ∈ master.dom & y. is deletion

→ master: del (y.key)

, y.key ∈ master.dom & y. is amendment

→ master: ext(y.key, upd (master(y.key), y))

, report: hiext (error (y))

fi

od

SEQUENTIAL FILE UPDATING (22)

The master is represented by oldmaster: (key X record)*

s.t. oldmaster. keysorted & oldmaster.consistent

& $\forall$ k: key (k ∈ master.dom $\Leftrightarrow$

$\exists$ i oldmaster(i) = (k, master (k)))

} oldmaster.ok

where x.keysorted $=_{df} \forall i,j\ (i \leq j < x.dom \Rightarrow x(i).key \leq x(j).key)$

x.consistent $=_{df} \forall\ i,j(x(i).key = x(j).key \Rightarrow i = j)$

given some data as previous slide. and newmaster: (X)* and precondition

oldmaster.ok & tr. keysorted

rewrite previous program to establish^{o} result, additional result

newmaster.ok (i.e. wrto new value of master)

use only these operations on newmaster & oldmaster:

z, oldmaster: lopop, newmaster := empty, newmaster: hiext

MAXIMAL STRONG COMPONENTS (23)

(due to Derek Moore after EWD)

A finite relation between nodes (a digraph) can be represented by a mapping:

SUCCESSORS: NODE $\rightarrow$ NODE**

The reflexive transitive closure of this is

DESC: NODE $\rightarrow$ NODE** , defined by

$$DESC(n) =_{df} \bigcup_{i=0}^{\infty} D_i$$

where $D_o = \{n\}$

$$D_{i+1} = \bigcup_{n \in D_i} \text{SUCCESSORS}(n)$$

If $SUCC(s) =_{df} \bigcup_{n \in s} \text{SUCCESSORS}(n)$

then $DESC(n) = \bigcap \{S \mid n \in S \ \& \ SUCC(s) \subseteq S\}$

$= \text{l.f.p of } S = \{n\} \cup SUCC(S)$

$ANC(n) = \{m \mid n \in DESC(m)\}$

The maximal strong component containing n is

$$STR(n) = DESC(n) \cap ANC(n)$$

$m \leq n \quad =_{df} \quad m \in DESC(n)$ (24)

$\leq$ is a preorder (reflexive & transitive)

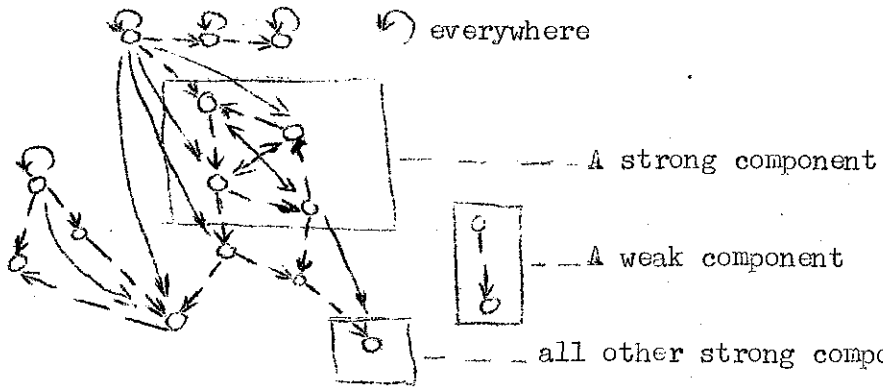
$m \equiv n \quad =_{df} \quad m \leq n \quad \& \; n \leq m$

$\equiv$ is an equivalence relation.

The equivalence classes are the max strong components of the graph defined by SUCCESSORS.

$M.strong = \forall n \quad n \in M \Rightarrow M = STR(n)$

$M.strong \;\&\; N.strong \;\&\; M \neq N \Rightarrow M \cap N = empty$



--> SUCCESSORS.

--> DESC (examples)

Let MAXSCS: (NODE**)** (25)

Write a program with postcondition $MAXSCS = \{ M \mid M.strong \}$

$\cup$ MAXSCS = ALLNODES $\quad \overline{b}$

& $\forall M.\ M \in MAXSCS \Rightarrow M.\ strong \quad p$

MAXSCS := (NODE**)**.empty

do MAXSCS $\neq$ ALLNODES --> "extend MAXSCS" od

where $p \cap b \Rightarrow$ "extend MAXSCS" wp p

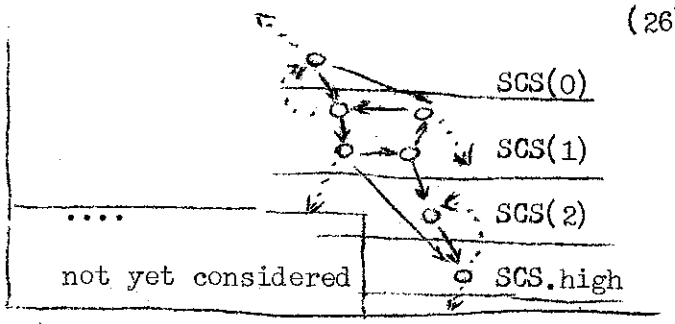
& " dec t

where t = ALLNODES.size − MAXSCS.size.

"extend MAXSCS"

Introduce a new variable

SCS: (NODE**)**



define p^1 $\forall i\ i < SCS.dom \Rightarrow \forall n \in SCS(i) \Rightarrow SCS(i) \subseteq STR(n)$

$p^2.\forall i\ 0 < i < SCS.dom \Rightarrow (SCS(i) \cap SUCC(SCS(i-1))).size \neq 0$

$p^3.\forall i\ i < SCS.dom \ \&\ M \in MAXSCS \Rightarrow (M \setminus SCS(i)).size = 0$

$p = p^1 \ \&\ p^2 \ \&\ p^3$

$REM =_{df} SUCC(SCS.high) - \bigcup MAXSCS - SCS.high$

$\bar{b} = REM.size = 0$

check $p \ \&\ \bar{b} \Rightarrow SCS.high.strong \ \&\ SCS.high \tilde{\in} MAXSCS$

$\&\ SCS.dom > 0$ (from p^3)

(Dijkstra them 2A)

Proof. let $n \in SCS.high$ (exists by p^2_3) let $m \in STR(n)$

(by $\ni$) $n \in STR(m)$. Hence $m \tilde{\in} UMAXSCS$ (otherwise n would be too, contradicting p^3). Now let m be a <u>closest</u> descendent of n, i.e. $m \in SUCCESSORS(n) \subseteq SUCC(SCS.high)$ (since $n \in SCS.high$)

Thus $m \in SUCC(SCS.high) - UMAXSCS \subseteq SCS.high$ (since $REM = \emptyset$)

By induction, this is true of all $m \in STR(n)$

hence $\forall n \in SCS.high$, $STR(n) \subseteq SCS.high$. equality follows from p^1

(27)

Precondition ALLNODES - UMAXSCS $\neq \emptyset$

Initialisation = N: NODE;

N: $\in$ ALLNODES - UMAXSCS;

SCS := (N.unitset).unitseq

check $N \in ALLNODES - UMAXSCS \Rightarrow$

(p^1) $\forall n \;\; n \in N.unitset \Rightarrow N.unitset \subseteq STR(n)$ (27)

(p^2) vacuous because SCS.dom = 1

(p^3) $\forall M \;\; M \in MAXSCS \Rightarrow (M \cap N.unitset).size = 0$

Extend MAXSCS = Initialisation; Loop

where Loop = do REM $\neq \emptyset \rightarrow$ { extend SCS
                                                { decrease REM od;

MAXSCS: ext (SCS.high);

SCS: hirem

More efficient: Initialisation; do SCS.dom $\neq 0$ — Loop od

check that SCS.hirem satisfies p.

It remains to write the body of the lpp (28)

do REM $\neq \emptyset \rightarrow$ { extend SCS
                                  { decrease REM od

preserving the invariant p. and \variant REM.size)

Select arbitrary $n \in REM$. Define USCS = $\bigcup_{i < SCS.dom} SCS(i)$

Lemma 1 if $n \tilde{\in}$ USCS then SCS.hiext (n.unitset) satisfies p

Proof p^1. $\{n\} \subseteq STR(n)$ always, so new element of SCS satisfies p^1

p^2. from $n \in REM$, $n \in SUCC(SCS.high)$

so $\{n\} \cap SUCC(SCS.high) \neq \emptyset$

p^3. from $n \in REM$, $n \tilde{\in} UMAXSCS$

hence $\{n\} \cap UMAXSCS = \emptyset$

extend SCS

decrease REM = $n :\in REM$;

[$n \tilde{\in}$ USCS $\rightarrow$ SCS:hiext(n.unitset)

[] $n \in$ USCS $\rightarrow$ see next slide.

]

Given $n \in REM \cap USCS$, write code to preserve p (and reduce REM') (29)

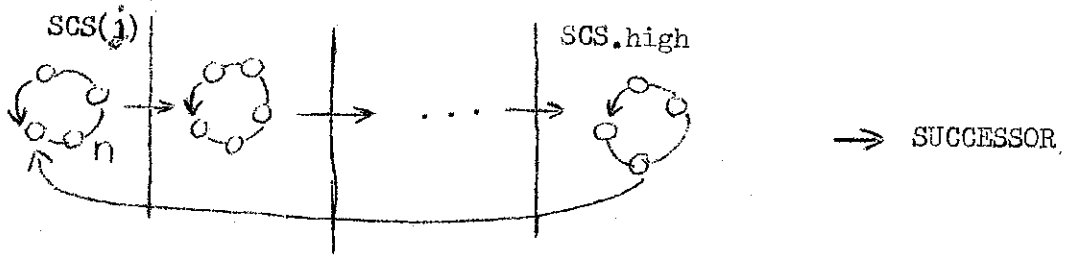
(Can't be done, so we will reduce SCS.dom)

Let $n \in SCS(j)$ for some j. then by $n \in REM$, $SCS(j) \neq SCS.high$

Therefore there is a cycle (because $n \notin SUCC(SCS.high)$)

$$SCS(j) \longrightarrow SCS(j+1) \longrightarrow \ldots \longrightarrow SCS.high \longrightarrow SCS(j)$$

where neighbouring items satisfy p^2



consequently any pair of elements from this picture is connected by a cyclic path i.e. by $ANC \cap DESC$, hence the union $\bigcup = \bigcup_{k=j}^{SCS.dom-1} SCS(k)$ is contained in STR(m) for all $m \in \bigcup$. (property p^1). Dijkstra thm. 1A

So try do $n \tilde{\in} SCS.high \longrightarrow SCS(SCS.dom - 2) :\cup SCS.high;$ SCS.hirem od

MAXSCS: (NODE**)** = empty; UMAXSCS: NODE** = empty (30)

do UMAXSCS $\neq$ ALLNODES $\longrightarrow$ N: NODE;

N: $\in$ ALLNODES — UMAXSCS;

SCS: (NODE**)* = N.unitset.unitseq; USCS: NODE** = SCS(0);

do SCS.dom $\neq$ 0 $\longrightarrow$ REM := SUCC(SCS.high) — UMAXSCS — SCS.high;

do REM.size $\neq$ 0 $\longrightarrow$ N: REM;

if N $\tilde{\in}$ USCS $\longrightarrow$ SCS:hiext (N.unitset); USCS: ext(N);

$\prod$ N $\in$ USCS $\longrightarrow$ do N $\tilde{\in}$ SCS.high $\longrightarrow$

SCS(SCS.dom − 2) :$\cup$ SCS.high; SCS: hirem

od

fi; REM := SUCC(SCS.high) − UMAXSCS − SCS.high

od; MAXSCS: ext (SCS.high); UMAXSCS: $\cup$SCS.high; SCS: hirem

od

od