The longest plateau and other stories

For a sequence $f(i: 0 \leq i < N)$ we call

$$f(i: x \leq i < x+k) \quad \text{with} \quad 0 \leq x \leq x+k \leq N$$

a segment of length k. (Note that the empty sequence - i.e. $N=0$ - contains a segment of length 0.) We are interested in programs msl - for maximum segment length - that determine the maximum lenghth of a segment satisfying some criterion $\mathcal{B}$.

With the abbreviation b defined by

$$b.x.y \equiv \mathcal{B}.f(i: x \leq i < y) \vee x = y$$

the functional specification of msl is

$$[\![N: \text{int} ; f(i: 0 \leq i < N): \underline{\text{array}} \ \{N \geq 0\}$$
$$; [\![c: \text{int}$$
$$; \text{msl} \ \{R: \ c = (\underline{\text{MAX}}\ x,y: 0 \leq x \leq y \leq N \wedge b.x.y: y-x)\}$$
$$]\!]$$
$$]\!] \ .$$

Note. In the definition of b, the disjunct $x=y$ has been added so as to ensure that in R the right-hand side is defined. From $c=0$ we can conclude that no non-empty segment satisfies $\mathcal{B}$. (End of Note.)

For our first approximation we obtain the invariant P_0 by replacing in R the constant N by a variable and restricting its range:

$$P_0: \quad c = (\underline{\text{MAX}}\ x,y: 0 \leq x \leq y \leq n \wedge b.x.y: y-x) \wedge 0 \leq n \leq N$$

with the relevant properties $[(c,n) = (0,0) \Rightarrow P_0]$ and $[P_0 \wedge n = N \Rightarrow R]$. This leads to our first approximation of msl:

$\|[n: \text{int} ; c,n := 0,0 \{P_0\}$

; do $n \neq N \rightarrow \{P_0 \wedge n \neq N\}\ c,n := E, n+1 \{P_0\}$ od

$\{P_0 \wedge n = N$, hence$\}$

$]\| \{R\}$

where E should be such as to justify the annotation.

Remark Without apology we omit the justification for our decision to increase n each time by 1. (End of Remark.)

In order to determine E we observe under the validity of $P_0 \wedge n \neq N$

$wp(\text{"}c,n := E, n+1\text{"}, P_0)$

$= \{$Axiom of Assignment, definition of $P_0\}$

$E = (\underline{MAX}\ x,y: 0 \leq x \leq y \leq n+1 \wedge b.x.y: y-x) \wedge 0 \leq n+1 \leq N$

$= \{$properties of MAX$\}$

$E = (\underline{MAX}\ x,y: 0 \leq x \leq y \leq n \wedge b.x.y: y-x)\ \underline{max}$

$(\underline{MAX}\ x: 0 \leq x \leq n+1 \wedge b.x.(n+1): n+1-x)$

$\wedge\ 0 \leq n+1 \leq N$

$= \{P_0 \wedge n \neq N$; properties of MAX and MIN$\}$

$E = c\ \underline{max}\ n+1 - (\underline{MIN}\ x: 0 \leq x \leq n+1 \wedge b.x.(n+1): x)$

$= \{$since $[b.(n+1).b(n+1)]\}$

$E = c\ \underline{max}\ n+1 - (\underline{MIN}\ x: 0 \leq x \wedge b.x.(n+1): x)$.

This suggests a program with a further variable, h say, to record the value of the above MIN-expression, and with the invariant $P0 \wedge P1$, where $P1$ is given by

$P1$: $h = (\underline{MIN}\ x: 0 \leq x \wedge b.x.n: x)$.

Our second approximation thus becomes - $P1$ being initialized by virtue of b.0.0 -

$[\![n,h : int ;\ c,n,h := 0,0,0 \ \{P0 \wedge P1\}$

$;\ \underline{do}\ n \neq N \rightarrow \{P0 \wedge P1 \wedge n \neq N\}\ h := E1$

$\{P0 \wedge P1^{n}_{n+1} \wedge n \neq N\}$

$;\ c,n := c\ \underline{max}\ n+1-h\ ,\ n+1\ \{P0 \wedge P1\}$

$\underline{od}$

$]\!]$

in which $E1$ is defined as the solution of $h: P1^{n}_{n+1}$ or, equivalently, as the smallest natural solution of $x: b.x.(n+1)$.

We observe that, with $P2$ given by

$P2$: $n-h \leq c$

(i) $wp(\text{"}c,n := c\ \underline{max}\ n+1-h\ ,\ n+1\text{"}, P2)$

$= \{$ Axiom of Assignment, definition of $P2 \}$

$n+1-h \leq c\ \underline{max}\ n+1-h$

$= \{$ properties of $\underline{max} \}$

true

(ii) $[(c,n,h) = (0,0,0) \Rightarrow P2]$;

hence, the invariant $P_0 \wedge P_1$ of our second approximation may be replaced by $P_0 \wedge P_1 \wedge P_2$. Note that, in general, P_2 is falsified by $h := E_1$.

In view of the definition of E_1 as the smallest natural solution of $x: b.x.(n+1)$ the Linear Search Theorem tells us to implement $h := E_1$ by

$$h := 0 \; ; \; \underline{\text{do}} \; \neg b.h.(n+1) \rightarrow h := h+1 \; \underline{\text{od}} \quad ,$$

which, not counting the evaluations of b, would lead to a quadratic algorithm. If, however, E_1 is an ascending function of n, $h := E_1$ could be implemented by

$$\underline{\text{do}} \; \neg b.h.(n+1) \rightarrow h := h+1 \; \underline{\text{od}}$$

which, not counting the evaluations of b, would lead to a linear algorithm. This observation is a strong suggestion to explore the consequences of the

Monotonicity Assumption The smallest natural solution of $x: b.x.n$ is an ascending function of n. (End of Monotonicity Assumption.)

as this might lead to a class of fast algorithms. In the rest of this note, the Monotonicity Assumption is made.

Under the monotonicity assumption, $h := E_1$ never decreases h; hence $h := E_1$ maintains P_2! So, let us investigate what the precondition P_2, i.e. $n-h \leq c$ allows us to conclude about the adjustment of c, viz.

$$c := c \; \underline{\max} \; n+1-h \quad .$$

The adjustment leaves c unchanged if initially $n+1-h \leq c$ or, equivalently, $n-h<c$. In view of the initial validity of P2, $n-h=c$ is the only remaining initial possibility, in which case c is increased by 1. Hence the adjustment could be coded

(0)
```
if n-h<c → skip
[] n-h=c → c:=c+1
fi
```

It is nice to know that c will be increased by 1 at a time. Let us now turn to a possible optimization: suppose we can think of a Q such that

(i) Q is stable, i.e. is maintained by the repeated statements and, hence, when once true, remains true, and

(ii) is such that Q implies in conjunction with with the precondition of (0) that $n-h<c$ holds.

In that case, c has reached its final value as soon as Q holds and, hence, we can strengthen the guard $n \neq N$ to $n \neq N \wedge \neg Q$, thereby possibly causing termination after fewer iterations.

From the precondition, $P0 \wedge n \neq N$ implies $n < N$, hence Q defined by

Q: $N-h \leq c$

meets requirement (ii). It also meets requirement (i) since, the repeated statements don't decrease c and under the monotonicity assumption don't decrease h and, hence, don't falsify Q. (The stability conclusion re Q is our last exploitation

of the monotonicity assumption.) So, without influencing the final value of c, we may replace the guard by $n \neq N \wedge N-h > c$. But this can now be simplified: in conjunction with invariant P_2 —i.e. $n-h \leq c$— $N-h>c$ implies $n \neq N$, which conjunct can therefore be omitted. (This is our last usage of P_2.)

Collecting the above, we come —under the monotonicity assumption— to our third approximation for msl:

```
|[ n, h: int ; c, n, h := 0, 0, 0
; do N-h>c → h:= E1
          ; if n-h<c → skip
            [] n-h=c → c:= c+1
          fi
          ; n:= n+1
  od
]|
```

Comment I have hesitated about the order in which to introduce P_2 and the Monotonicity Assumption, because it is only under the latter that the former becomes of interest. Because P_2 does not depend on the monotonicity assumption I have finally decided to introduce it first. (End of Comment.)

* * *

The monotonicity assumption is now interesting enough to be explored a little further, in particular we would like to have manageable criteria for determining whether it is satisfied. Fortunately we have a strong theorem, easily expressed in terms of our original predicate $\mathcal{B}$ on strings - now with $\mathcal{B}.\phi$ -

Theorem 0 The Monotonicity Assumption $\equiv$

$$(\underline{A}\, X,Y: \mathcal{B}.(XY): \mathcal{B}.X) \quad .$$

Proof 0

(i) LHS $\Leftarrow$ RHS

In the following, quantifications are constrained by $0 \leq x < k \leq n < m \leq N$.

The Monotonicity Assumption
$=$ {definition}
the minimal natural solution of $x: b.x.n$ is an ascending function of n
$=$ {definition of minimal natural solution}
$(\underline{A}\, k,n,m :: (\underline{A}x :: \neg\, b.x.n) \wedge b.k.n \Rightarrow (\underline{A}x :: \neg\, b.x.m))$
$\Leftarrow$ {predicate calculus}
$(\underline{A}\, k,n,m :: (\underline{A}x :: \neg\, b.x.n) \Rightarrow (\underline{A}x :: \neg\, b.x.m))$
$\Leftarrow$ {predicate calculus}
$(\underline{A}\, k,n,m :: (\underline{A}x :: \neg\, b.x.n \Rightarrow \neg\, b.x.m))$
$=$ {predicate calculus}
$(\underline{A}\, k,n,m :: (\underline{A}x :: b.x.m \Rightarrow b.x.n))$
$\Leftarrow$ {definition of $\mathcal{B}$}
$(\underline{A}\, X,Y: \mathcal{B}.(XY): \mathcal{B}.X)$

(ii) $\neg$ LHS $\Leftarrow$ $\neg$ RHS

From $\neg$ RHS we conclude that we can choose an X and Y such that $\neg\, \mathcal{B}.X$ and $\mathcal{B}.(XY)$.

Choose $n0$, $n1$ and f such that

$$f(i: 0 \leq i < n0) = X \quad \text{and} \quad f(i: n0 \leq i < n1) = Y .$$

From $\neg B.X \wedge B.(XY)$ we conclude - on account of the definition of b - $\neg b.0.n0 \wedge b.0.n1$, and that Y is not empty, i.e. $n0 < n1$. That is, the minimal natural solution of $x: b.x.n0$ exceeds 0, that of $x: b.x.n1$ equals 0. Hence the Monotonicity Assumption is not satisfied. (End of Proof 0.)

Comment I don't like the above proof, which caused me a lot of trouble to write down. The fact that I had to prove an equivalence by mutual implication does not bother me, as this is not unusual when dealing with minimal solutions. What somehow annoys me very much is that the two halves are written in such very different styles. (End of Comment.)

Though I have reservations about the above proof, I like the theorem very much, because (i) it expresses an equivalence, in particular also tells us when the Monotonicity Assumption is not satisfied, and (ii) its right-hand side

$$(1) \qquad (\underline{A}\, X, Y: B.(XY): B.X)$$

is so neat that there must be some nice theory about predicates satisfying (1), nice enough to be useful.

Let us call predicates B satisfying (1) "right-strengthening". Then we can formulate

Theorem 1. Disjunction and conjunction of two

right-strengthening predicates are both right-strengthening.

Proof 1

$$(\underline{A}X,Y: B0.(XY): B0.X) \wedge (\underline{A}X,Y: B1.(XY): B1.X)$$
$$\Rightarrow \{\text{predicate calculus}\}$$
$$(\underline{A}X,Y: B0.(XY) \vee B1.(XY): B0.X \vee B1.X) \quad \wedge$$
$$(\underline{A}X,Y: B0.(XY) \wedge B1.(XY): B0.X \wedge B1.X)$$

(End of Proof 1)

Remark. A segment is called a "slope" if it is ascending or descending. Everybody who ever made a mess of it when trying to solve W.H.J. Feijen's problem of determining the maximum slope length – and there are many such people – will appreciate the insight captured by the above theorem. (End of Remark.)

For right-strengthening B, string X and "character" y we conclude from (1)

$$B.(Xy) \Rightarrow B.X \quad ,$$

that is, there exists a boolean function d of a string and a character such that

$$B.(Xy) \equiv B.X \wedge d.X.y$$

– the weakest d being

$$d.X.y \equiv B.(Xy) \vee \neg B.X \qquad - \quad .$$

By induction on the length of the string we conclude

Theorem 2. Consider

(2) $(\underline{A}X:: B.X \equiv (\underline{A}\,YyZ: YyZ = X: d.Y.y) \vee X = \emptyset)$;

then (i) for any given d, there exists a right-strenthening $\mathcal{B}$ satisfying (2),

and (ii) for any given right-strengthening $\mathcal{B}$, there exists a d satisfying (2).

Also Theorem 2 gives much satisfaction; it is useful to know because, in case of freedom, it makes sense to choose one's d most conveniently.

* * *

At last we should be ready for the famous Problem of the Longest Plateau. [I must confess to observe with great pleasure how long I have managed to postpone this stage.]

A plateau is defined as a segment all whose elements are equal. We recommend for d

$$d.Y.y \equiv y = head.(Yy) \quad ,$$

i.e. we write $b.x.y$ as

$$b.x.y \equiv (\underline{A}j: x \leq j < y: f.x = f.j) \quad .$$

From our third approximation we derive

```
|[ n,h: int ; c,n,h := 0,0,0
; do N-h>c
    → if f.h=f.n → skip [] f.h≠f.n → h:= n fi
    ; if n-h<c → skip [] n-h=c → c:= c+1 fi
    ; n:= n+1
  od
]|
```

Let us study our solution a bit further: what is the initial condition under which an iteration increases c ? To this purpose we evaluate

$$wp(\text{"}\underline{\text{if}}\ f.h=f.n \to \text{skip} \ [] \ f.h \neq f.n \to h:=n \ \underline{\text{fi}}\text{"}, n-h=c)$$
$$= \{ \text{semantics} \}$$
$$(f.h=f.n \wedge n-h=c) \vee (f.h \neq f.n \wedge 0=c) \ .$$

The condition $0=c$ is so strong that the second disjunct never occurs: in fact, P_3 —to all intents and purposes its negation—

$$P_3: \quad f.h=f.n \vee 0<c$$

is a further invariant. (P_3 is implied by

$$(n,h,c)=(0,0,0) \vee 0<c \ ,$$

the invariance of which is straightforward to establish.)

The invariance of P_3 allows the annotation

$$f.h \neq f.n \to \{0<c\} \ h:=n \ \{n-h<c\} \ ,$$

and hence the option of adjusting c can be confined to the first alternative:

```
|[ n,h: int ; c,n,h := 0,0,0
; do N-h>c → if f.h = f.n → if n-h < c → skip
                           [] n-h = c → c := c+1
                           fi
             [] f.h ≠ f.n → h := n
             fi ; n := n+1
  od
]|
```

The condition for increase of c

$$f.h = f.n \;\wedge\; n-h = c$$

is equivalent to

$$f.(n-c) = f.n \;\wedge\; n-h = c$$

and we could ask ourselves under which circumstance the first conjunct -which does not contain h - implies the second, for then the test $n-h=c$ is no longer needed and we may be able to dispense with h altogether. In view of P_2 it suffices to demonstrate the weaker

(3) $$f.(n-c) = f.n \;\Rightarrow\; n-h \geq c$$

The right-hand side is equivalent to $h \leq n-c$, and h being the smallest natural solution of 'x: b.x.n', we can prove (3) provided we can show

$$f.(n-c) = f.n \;\Rightarrow\; b.(n-c).n$$

- $n-c \geq 0$ being trivially invariant - , i.e.

$$f.(n-c) = f.n \;\Rightarrow\; (\underline{A}j: n-c \leq j < n: f.(n-c) = f.j) \quad .$$

This implication holds for an f in which equal values are nowhere separated by different ones. Let us call such an f "equality compact".

Replacing the guard $N-h>c$ by the original $n \neq N$, we can eliminate for equality-compact f the variable h altogether:

```
|[ n: int ; c, n := 0, 0
; do n ≠ N → if f.(n-c) = f.n → c, n := c+1, n+1
             [] f.(n-c) ≠ f.n → n := n+1
             fi
  od
]|
```

This is the program David Gries gives on p.203 of "The Science of Programming". (He restricts himself to ordered f -ascending or descending-, which is a special case of equality-compact f; for unclear reasons he has chosen to restrict himself to the case $N \geq 1$.)

So much for the problem of The Longest Plateau.

In retrospect I should have defined the function g by

g.n = the smallest natural solution of x: b.x.n ;

and instead of "h:= E1" rather "h:= g.(n+1)". I could then have used g in arguments as well. (End of In retrospect.)

Wasn't this fun?

Austin, 10 October 1985

prof.dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
USA