A note on the Kaldewaij Schemes for longest and shortest segments

First in [0] and later in [1], Anne Kaldewaij has published two beautiful program schemes. One of them deals with the computation of a *longest* array segment satisfying a condition so and so, and the other with the computation of a *shortest* segment. However, the mysterious — i.e. ununderstood — thing is that the two schemes are very similar, but not entirely similar. Could this difference be explained from the difference between "longest" and "shortest"? The answer is that both schemes apply to both categories, which was overlooked by Kaldewaij. This note is meant to fill in the gap.

* * *

We consider boolean function $C.p.q$ and integer function $t.p.q$ to be defined for all p, q satisfying $0 \leq p \leq q \leq N$. Now we define $G.x.y$ for $0 \leq x \leq y \leq N$ by

$$G.x.y \;=\; (\underline{\max}\; p,q : \; x \leq p \leq q \;\wedge\; y \leq q \leq N \;\wedge\; C.p.q \\ : \; t.p.q \;)$$

We envisage a program that establishes

R: $\quad G.0.0 = r$

on the basis of the invariant $P0 \wedge P1$ given by

$P0$: $\quad G.0.0 = r \;\underline{\max}\; G.x.y$

$P1$: $\quad 0 \leq x \leq y \leq N$.

As a first approximation for such a program we choose

$r, x, y := +\infty, 0, 0$
$\{\text{Inv } P0 \wedge P1\}$ $\{\text{Bnd } 2*N - (x+y)\}$
; <u>do</u> guard $\rightarrow$ "increase $x+y$" $\{P0 \wedge P1\}$ <u>od</u>

By construction, the number of steps of the repetition is linear in N at worst. Now the question is this: can we complete the program so that its total "time-complexity" remains linear in N at worst? It turns out that we can at the expense of some simple requirements to be imposed on C and t.

* * *

As always with these kinds of problems, we investigate increments of x and y by 1, in the following way

- $G.x.y$

$=$ { isolate $p = x$ }

$(\underline{\max}\ q : x \le q \wedge y \le q \le N \wedge C.x.q : t.x.q)$
$\underline{\max}$
$G.(x+1).y$

$=$ { use $x \le y$ – from $P1$ – }

$(\underline{\max}\ q : y \le q \le N \wedge C.x.q : t.x.q)\ \underline{\max}\ G.(x+1).y$

$=$ { • definition of A }

$A\ \underline{\max}\ G.(x+1).y$

Thus, we maintain $P0$ by $r, x := r\ \underline{\max}\ A\ ,\ x+1$.

- $G.x.y$

$=$ { isolate $q = y$ }

$$(\underline{\max}\ p:\ x \le p \le y \wedge C.p.y :\ t.p.y)\ \underline{\max}\ G.x.(y+1)$$
$=\quad \{\circ$ definition of $B\}$
$$B\ \underline{\max}\ G.x.(y+1)$$

Thus, we maintain P0 by $r, y := r\ \underline{\max}\ B,\ y+1$.

The question that remains is how to determine — at a bargain — the values of A and B given by

$$A = (\underline{\max}\ q:\ y \le q \le N \wedge C.x.q :\ t.x.q)$$
$$B = (\underline{\max}\ p:\ x \le p \le y \wedge C.p.y :\ t.p.y) \quad .$$

In view of the fact that x, y, and N are program variables the (only) potential values for A that can be computed at once are $t.x.N$, $t.x.y$, and $+\infty$. For B these are $t.y.y$, $t.x.y$, and $+\infty$.

<u>Re A</u>

$A = t.x.N$: not useful for the rest of the derivation

(A0) $A = t.x.y$
$\Leftarrow$
$C.x.y \wedge t.x.q$ descending in q

(A1) $A = +\infty$
$\Leftarrow$
$\neg C.x.y \wedge C.x.q$ strengthening in q

<u>Re B</u>

$B = t.y.y$: not useful for the rest of the derivation

(B0) $B = t.x.y$
$\Leftarrow$
$C.x.y \wedge t.p.y$ descending in p

(B1) $B = +\infty$
$\Leftarrow$
$\neg C.x.y \wedge C.p.y$ strengthening in p

(<u>End</u> Re.)

By the above analysis we may take as the body of the repetition

<u>if</u> $C.x.y \rightarrow$ {$t.x.q$ descending in q}
$r, x := r$ <u>max</u> $t.x.y$, $x+1$ — A0 —

▯ $\neg C.x.y \rightarrow$ {$C.x.q$ strengthening in q}
$r, x := r$ <u>max</u> $+\infty$, $x+1$ — A1 —

▯ $C.x.y \rightarrow$ {$t.p.y$ descending in p}
$r, y := r$ <u>max</u> $t.x.y$, $y+1$ — B0 —

▯ $\neg C.x.y \rightarrow$ {$C.p.y$ strengthening in p}
$r, y := r$ <u>max</u> $+\infty$, $y+1$ — B1 —

<u>fi</u> ,

provided —of course— the plugged in assertions are satisfied. (If they are not, those alternatives are to be removed.)

In case alternatives A0 and A1 apply, variable y can be dispensed with altogether, and presumably should not have been introduced in the first place. Similarly, the combination B0, B1 renders x superfluous. And thus we are left with the combinations A0, B1 and B0, A1, which give rise to the two schemes announced before

<u>Scheme 0</u> (A0, B1)

```
     r, x, y := +∞, 0, 0
     {Inv P0 ∧ P1} {Bnd 2*N - (x+y)}
   ; do guard0 →
          if C.x.y → r, x := r max t.x.y , x+1
          [] ¬C.x.y → y := y+1
          fi
     od ,
     provided   t.p.q  descending in q
                C.p.q  strengthening in p
```

<u>Scheme 1</u> (B0, A1)

```
     r, x, y := +∞, 0, 0
     {Inv P0 ∧ P1 } {Bnd 2*N - (x+y) }
   ; do guard1 →
          if C.x.y → r, y := r max t.x.y , y+1
          [] ¬C.x.y → x := x+1
          fi
     od ,
     provided   t.p.q  descending in p
                C.p.q  strengthening in q
```

What remains to be done is to determine the guards and to settle the invariance of

$P1: \quad 0 \le x \le y \le N$.

We shall carry this out for Scheme 0 .

Statement $x := x+1$ may violate P1. We preclude this danger by requiring that its guard $C.x.y$ satisfy

$$C.x.y \Rightarrow x \neq y \quad .$$

This requirement is met by imposing on C the additional constraint

$$\neg C.p.p \quad , \quad \text{for all } p \quad .$$

Statement $y := y+1$ may also violate P1 . We preclude this danger by requiring that its guard $\neg C.x.y$ satisfy

$$\neg C.x.y \Rightarrow y \neq N ,$$

i.e. $\quad C.x.y \vee y \neq N ,$

and it is precisely this condition that we shall take as the guard of the repetition .

Upon termination we now have P0 and the negation of the guard, implying

$$G.0.0 = r \ \underline{max} \ G.x.N \qquad \wedge \quad \neg C.x.N$$

Because we have

$\quad G.x.N$

$= \quad \{ \text{def. of } G \}$

$$(\underline{\max}\ p,q:\ x \le p \le q\ \wedge\ N \le q \le N\ \wedge\ C.p.q\ :\ t.p.q)$$

= { one-point rule }

$$(\underline{\max}\ p:\ x \le p \le N\ \wedge\ C.p.N\ :\ t.p.N)$$

= { $\neg C.x.N$ holds, and $C.x.N$ is strengthening in x }

$+\infty$,

the postcondition implies

R: $\quad G.0.0 = r$,

as required.

Summarizing, we have

<u>Scheme 0</u>

```
r, x, y := +∞, 0, 0
; do C.x.y ∨ y ≠ N →
      if C.x.y → r, x := r max t.x.y , x+1
      [] ¬C.x.y → y := y+1
      fi
  od {R} .
  provided   t.p.q   descending in q
             C.p.q   strengthening in p
             ¬C.p.p
```

And in a similar way we obtain

Scheme 1

```
   r, x, y := +∞, 0, 0
;  do ¬C.x.y ∨ y ≠ N →
        if  C.x.y → r, y := r max t.x.y , y+1
        [] ¬C.x.y → x := x + 1
        fi
   od
;  r := r max t.x.N   {R}  ,

   provided   t.p.q   descending in p
              C.p.q   strengthening in q
              C.p.p
```

* * *

The above schemes – although nice or even beautiful in their own right – should, I think, not be learned by heart, simply on the ground that they represent far too complicated theorems. Moreover, if we vary the monotonicity properties of C and t, other schemes emerge.

Our interest is not in the Schemes themselves, but in their derivations which act as model derivations for a rather big class of programming problems. Each time we encounter the problem of computing

$$(\underline{\max}\ p,q : 0 \le p \le q \le N \wedge C.p.q : t.p.q) ,$$

where C and t have monotonicity properties, it pays to define a suitable function $G.x.y$ to be plugged into a tail invariant ala $P0$.

(While writing the above "suitable", it occurred to me that it should not be difficult to enumerate the suitable functions G for the various monotonicity patterns. But that's for later, because I wish to call it a day - it is 10 PM -)

WHJ Feijen

Sterksel,
2 December 1992
(my father's birthday)

[0] Anne Kaldewaij,
Shortest and Longest Segments,
in: Beauty is our Business, a birthday salute to Edsger W. Dijkstra
Springer Verlag 1990

[1] Anne Kaldewaij,
Programming: The Derivation of Algorithms,
Prentice Hall 1990