Shiloach's Algorithm, taken as an exercise in Presenting Programs ,

by A.J.M. van Gasteren and W.H.J. Feijen .

The reason for writing this report is about twofold.
Firstly, though programs are nowadays developed and published at a great rate and sometimes even embody marvellous designs of high intellectual density, they are rarely presented in a fashion conducive to excitement. This is a pity.
Secondly, the deplorable state of the art in the field of presenting programs and justifying their correctness might very well be the reason why computing science as a challenging new branch of applied mathemetics has largely been ignored by the mathematical community as a whole. This is a pity too.

We wish to demonstrate how a program can be presented clearly, shortly, and in all its relevant detail by a suitable arrangement of the considerations that might lead to its development.
As an example we have chosen an ingenious algorithm, invented by Yossi Shiloach in 1979 (that late!) [0], which checks the equivalence of two circular lists.
For the techniques we use in reasoning about programs the reader may consult [1].

*

The problem we describe is symmetric in the integer sequences
$(A[0], A[1], \ldots, A[N-1])$ and $(B[0], B[1], \ldots, B[N-1])$, $N \geq 1$,
and so will be the solution.

+

In terms of $(A[0], A[1], \ldots, A[N-1])$ we define the set of so-called A-sequences A_i , viz. for all natural i (i.e. $i \geq 0$)

A_i: $(A[i], A[i+1], \ldots, A[i+N-1])$, with indices reduced modulo N .

Note that $A_i = A_{i+N}$, so that the set of A-sequences contains at most N elements.
It is requested to write a program solving the equation

R: $eq = (\underline{E}\ i,j :\ A_i = B_j)$.

(This is an equation in a single unknown eq which can take on the values true and false .)

+

(R is easily solved by comparing each A-sequence with each B-sequence, but in our trade we refuse to do so: no specific property of A- and B-sequences is then exploited.)

Either the sets of A- and B-sequences are disjoint, in which case eq = false solves R , or they are not, in which case eq = true solves R . If they are not disjoint they are equal: if $A_i = B_j$ then $A_{i+1} = B_{j+1}$ (see the definitions of A_i and B_j), and hence $A_{i+k} = B_{j+k}$ for all k .
The sets being either disjoint or equal, they can be compared by comparing two corresponding members.

We define

AA: the lexicographically first A-sequence ,

and rephrase equation R :

R: $eq = (AA = BB)$.

+

(R can be solved by computing AA and BB , but in our trade we refuse to do so: when solving R , only their equality or difference matters.)

We propose to discover the solution eq = true by identifying a pair (i,j) such that $A_i = B_j$. To that end we introduce the weaker relation

P: $0 \leq h \ \wedge\ (\underline{A}\ k:\ 0 \leq k < h:\ A[i+k] = B[j+k])$.

It derives its importance from (the universal validity of)

(0): $(P \wedge h \geq N \wedge eq) \Rightarrow R$.

We propose to discover the solution $eq = false$ by observing $AA \neq BB$. To that end we introduce the weaker relation

QA: $0 \leq i \wedge (\underline{A}\ k: 0 \leq k < i: A_k \succ BB)$,

in which " $\succ$ " is read "comes lexicographically after" .
It derives its importance from (the universal validity of)

(1): $(QA \wedge i \geq N \wedge \neg eq) \Rightarrow R$:

$$
\begin{array}{l}
(QA \quad i \geq N \wedge \neg eq) \\
\Rightarrow\ ((\underline{A}\ k: 0 \leq k < N: A_k \succ BB) \wedge \neg eq) \\
\Rightarrow\ (AA \succ BB \wedge \neg eq) \\
\Rightarrow\ (AA \neq BB \wedge \neg eq) \\
\Rightarrow\ R \ .
\end{array}
$$

Note that the relation P couples the name i with the name A , and j with B , so that we propose that

(2): the discussion is symmetric in the pairs (A,i) and (B,j) .

+

The following program establishes R

```
begin h, i, j: int
    ; h, i, j:= 0, 0, 0
      {P ∧ QA     QB: (loop-)invariant}
    ; do h < N ∧ i < N ∧ j < N
      → {h + i + j ≤ 3.N - 3}
        "increase h + i + j (,maintaining the invariant)"
      od
      {P ∧ QA ∧ QB ∧ (h ≥ N ∨ i ≥ N ∨ j ≥ N)}
```

```
    ; if h ≥ N → {P ∧ h ≥ N} eq:= true {R, see (0)}
      [] i ≥ N → {QA ∧ i ≥ N} eq:= false {R, see (1)}
      [] j ≥ N → eq:= false {R, see (2) and the preceding alternative}
      fi {R}
end .
```

+

Our remaining obligation consists in detailing "increase h + i + j (,maintaining the invariant)" .

If $A[i+h] = B[j+h]$, an increase of h by 1 does the job.
If $A[i+h] \neq B[j+h]$, we conclude $A_i \neq B_j$ and more specifically - on account of the meaning of lexicographic order - $(P \wedge A[i+h] > B[j+h]) \Rightarrow A_i \succ B_j$.
An even stronger conclusion is possible, viz. if $(A_i \succ B_j \wedge A[i] = B[j])$ then $A_{i+1} \succ B_{j+1}$, and hence (by mathematical induction)

$$(P \wedge A[i+h] > B[j+h]) \Rightarrow (\underline{A}\ k:\ 0 \leq k < h+1:\ A_{i+k} \succ B_{j+k}) ,$$

from which we deduce

(3): $$(P \wedge QA \wedge A[i+h] > B[j+h]) \Rightarrow QA(i:= i+h+1) :$$

(If R is a predicate, R(x:= E) is a copy of R in which each free occurrence of x is replaced by (E) .)

$$\begin{aligned} &(P \wedge QA \wedge A[i+h] > B[j+h]) \\ \Rightarrow\ &(QA \wedge (\underline{A}\ k:\ 0 \leq k < h+1:\ A_{i+k} \succ B_{j+k})) \\ \Rightarrow\ &(QA \wedge (\underline{A}\ k:\ 0 \leq k < h+1:\ A_{i+k} \succ BB)) \\ \Rightarrow\ &QA(i:= i+h+1) \end{aligned}$$

And now we are ready to present "increase ..." :

```
if A[i+h] = B[j+h] → h:= h + 1
 [] A[i+h] > B[j+h] → i:= i + h + 1 {QA, see (3)}; h:= 0 {P}
 [] B[j+h] > A[i+h] → j:= j + h + 1; h:= 0 {see (2) and the preceding alternative}
fi .
```

This completes the treatment of our example.

*

Final Remarks.

We intentionally did not assemble the parts into a complete program text, though many programmers would have liked it. The only valid reason for such an assembly is to feed the text into a machine so that it can be executed. What then happens is not a programmer's concern.

In our reasoning and arguing we have aimed at a level of formal detail in which each mathematician should be able to bridge the gaps. The choice of this level indicates the kind of mathematician we address, and we would like to be.

We conclude that the treatment of the above programming problem in an elementary course on computer programming should present no difficulties whatsoever.

(End of Final Remarks.)

Acknowledgements.
J.J.A.M. Brands, H.C.A. van Tilborg, and the members of the Tuesday Afternoon Club are acknowledged for their instructive and illuminating comments.
(End of Acknowledgements.)

References.

[0] Shiloach, Yossi, "A fast equivalence-checking algorithm for circular lists", Information Processing Letters 8 (5) (1979) 236-238 .

[1] Dijkstra, Edsger W., "A Discipline of Programming", Prentice-Hall, 1976 .

(End of References.)

(End of Shiloach's Algorithm, taken as an exercise in Presenting Programs ,)

19 June 1981

drs. A.J.M. van Gasteren
BP Venture Research Fellow

and

W.H.J. Feijen

Department of
Mathematics and Computing Science,
University of Technology,
5600 MB EINDHOVEN,
The Netherlands.