Copyright Notice

A new elephant built from mosquitos humming in harmony.

In an earlier document --EWD456-- I mentioned a problem, suggesting that it boiled down to forming a transitive closure. M.Rem pointed out to me that suggestion was wrong; this report deals with the problem in question.

We consider a non-deterministic finite state automaton with N states, each state being either a terminal or a non-terminal state. We can associate each state with a different node of a directed graph --and vice versa-- in which each node has at least one outgoing arc. Terminal nodes --i.e. nodes corresponding to a terminal state-- are the nodes whose only outgoing arc leads back into themselves: the only outgoing arc of a terminal node is also one of its incoming arcs. For each node the outgoing arcs point to the set of permissible "successor nodes", a node with only one outgoing arc is a deterministic node and all directed paths along the graph correspond to a possible computation of the machine.

Let R be a set of terminal nodes. We can then ask for the set V of nodes v , such that any directed path, starting at a node v will arrive after a finite number of arcs in a node from R. (This is asking for the weakest pre-condition for the finite state automaton.) After reducing the given graph by removing from each node from R its only outgoing arc, with respect to that reduced graph we can also define the set V as all the points v , such that each directed path starting at v is finite.

The following sequential program would do the job. Assuming the nodes to be consecutively numbered, we introduce an array nia --i.e. "number of ill-directed arcs-- that (after the removal of the outgoing arcs from nodes r in R) count for each node the number of its outgoing arcs that lead to a node outside V.

"initialize nia such that nia(r) = 0 for r in R and nia(n)= number of node n's outgoing arcs for any node n not in R;
C:= R; V:= empty;
do C ≠ empty → transfer an arbitrary node c from C to V;
PC:= predecessor set of c;
do PC ≠ empty → remove an arbitrary node pc from PC;

if nia(pc) > 1 → nia:(pc)= nia(pc) - 1
[] nia(pc) = 1 → nia:(pc)= 0; C:= C + pc
fi

od

od" .

And this sequential program demonstrates the ugliness of the problem quite nicely: for the initialization of nia we need for each node outside R (the size of its) successor set; thereafter we need for each node c its predecessor set.

The following "program" is a little bit less sequential: it manipulates the connection matrix. Let con(i, j) = 1 if the is an arc from i to j , otherwise con(i, j) = 0. (To each terminal node corresponds a 1 on the diagonal which is the only 1 in its row.) The array con will be broken down as the computation proceeds:

"C:= R; V:= empty;
do C ≠ empty → V:= V + C;
make all columns corresponding to the elements of
C equal to all zeros;
C:= all elements outside V to which correspond
all-zero rows
od .

Here the "ugliness" observed above is reflected by the repeatable statement itself, in which the connectivity matrix is accessed either by rows or by columns. In its second form the algorithm reflects, however, the potential parallellism, because each time all columns or all rows respectively can be treated concurrently.

One and a half year ago I designed a number of so-called "elephants built from mosquitos". The idea was that we should have a large set of micro-computers --mosquitos--with only very few input legs and output legs (and possibly some antennae for synchronization). According to a fixed pattern, input and output legs would be paired, each pair thus providing a directed communication link between two mosquitos. The question was whether we could design powerful special-purpose elephants built from such mosquitos, harmo-

niously humming together. (The hyper-fast Fourier elephant was the most spectacular output of that effort, but it turned out to be known.) The remainder of this report deals with the design of an elephant solving the problem posed above. It is reasonable to wish to design an elephant for this task: the modifications to which the matrix con is subjected is strictly monotonic and that should simplify the problems, otherwise present in elephant design, considerably. We are not interested in a one-mosquite elephant, not in an N^2-mosquito elephant either, and we are heading for an N-mosquito elephant, and we shall try to come away with the simplest strongly connected arrangement I can think of: a cyclic arrangement with traffic in one direction only, with a mosquito associated with each node.

We consider the nodes and the associated mosquitos numbered from 0 through N-1 . In order to do away with superfluous subscripts, each machine j refers to machine (j+1) mod N as "its right-hand neighbour". If all machine have a variable called "x" , transmission of information to one's right-hand neighbour will be coded as "xR:= ...." (We are heading for fully synchronized mosquitos.)

We shall now describe mosquito nr j . It is primarily the manager of the j-th column of the matrix con . We shall represent it as a boolean vector (with "true" for "1", i.e. the presence of an incoming arc for node j)

arc(i) means: from node i leads (still) an arc to node j.

Furthermore, we observe that in an arrangement like this, it does not seem to do any harm if a mosquito, once in set V , continues to set its vector "arc" to all elements false (for the time not bothering about termination). We introduce for each mosquito j a boolean

out means: node i is (still) outside set V.

We initialize V:= R , i.e. out = false for all target nodes and still true for all the others.

Consider what will happen if all machines j are now, after this initialization, simultaneously started on a synchronous execution of the following program:

```
mosquito j:     arc:(j)= arc(j) and out; xR:= arc(j);
                i:= (j - 1) mod N;
                do i ≠ j → arc:(i)= arc(i) and out;
                           xR:= x or arc(i);
                           i:= (i - 1) mod N
                od;
                out:= out and x
```

Each row is inspected starting at the diagonal and then towards the right. Each mosquito starts updating its column at the diagonal and then upwards. Each time a mosquito has updated element arc(i), x means "in row i a 1 (or true) occurs to the left of column j up to and including the diagonal element of row i", and updating and confrontation take place in complete synchronism. The above program should be repeated as many times as necessary. The following program will see to that with the same initialization

```
mosquito j:     new:= non out; act:= true;
                do act →
                      goonR:= new;
                      arc:(j)= arc(j) and out; xR:= arc(j);
                      i:= (j - 1) mod N;
                      do i ≠ j →
                            goonR:= goon or new;
                            arc:(i)= arc(i) and out;
                            xR:= x or arc(i);
                            i:= (i - 1) mod N
                      od;
                      new:= out and non x;
                      out:= out and x;
                      act:= goon
                od .
```

All mosquitos will terminate simultaneously. (The local boolean "act" is not strictly necessary: we could have done it with "goon" itself.)

* * *

Time-wise, the above elephant is not very spectacular. Perhaps this is not too surprising: it has been remarked before --for instance by Hopcroft and Tarjan in print-- that algorithms manipulating graphs in terms of the

connection matrix tend to be relatively poor. This elephant has been recorded for a few other reasons.

Firstly --with the exception of the hyper-fast Fourier elephant-- very little has been documented about our earlier efforts at elephant design.

Secondly, this is the first time that I have been able to solve a problem from graph theory with an elephant whose internal connection pattern between the mosquitos does not depend on the structure of the graph. (If it does, the elephant is such a very special-purpose one to be hardly interesting.) In view of the remark by Hopcroft and Tarjan it remains questionable whether much may be expected from such elephants, but that is still an open question.

Thirdly, it has been recorded as "a reminder", viz. a reminder of the fact that we do not have any systematic methodology for elephant design as we now seem to have for the design of sequential programs. The latter we can now usually present as the "natural" outcome of a number of stepwise refinements. The reader who has seen a number of such program developments will have noticed the completely different presentation of the above elephant. I can only say: "Well, here it is." and the reader, at the moment of understanding it, is expected to react with: "Ain't that cute!". But this is, of course, very unsatisfactory, for it just means that we have not understood yet the problems involved in elephant design. (The interlocking of updating the columns and scanning the rows is, of course, "cute" and there is no point in denying that I show it with some pride!)

Fourthly, the way in which simultaneous termination of mosquito activity is controlled --although not "deep" in any sense-- seems to have the virtue of generality, and therefore, deserves recording.

Fifthly, the solution seems remarkable for its very low demands on the facilities for inter-mosquito communication.

28th November 1974
Plataanstraat 5
NUENEN - 4565
The Netherlands

prof.dr. Edsger W.Dijkstra
Burroughs Research Fellow