Reducing control traffic in a distributed implementation of mutual exclusion

The following has been prompted by [0], from whose introduction the statement of the problem to be solved has been taken almost verbatim.

"This note presents a scheme for achieving mutual exclusion in a distributed system that consists of a finite number of processes that communicate with one another by means of messages. Processes are permanent: they are not created, destroyed or faulty. Each process is in 1 of 3 states: (0) engaged in its noncritical section, (1) waiting to enter its critical section, (2) engaged within its critical section. The time spent by a process within its critical section on each entry into the critical section is finite. A mutual exclusion algorithm is one in which processes cooperate to ensure that

R0: only one process is engaged within its critical section at a time, and

R1: no process has to wait forever to enter its critical section."

Requirement R0 is easily met. Following [0], we introduce a single token, either residing at one of the processes or being sent from one process to another. Requirement R0 is then met by maintaining P0 given by

P0: a process engaged in its critical section holds the token.

Each process maintains P0 by (i) not entering its critical section unless in possession of the token, and (ii) not sending the token to another process while being engaged in its critical section.

The next invariant is P1 given by

P1: the process holding the token is not waiting to enter its critical section.

Each process maintains P1 by (i) skipping the waiting state and entering its critical section directly when it completes its noncritical activity while in possession of the token, and (ii) entering its critical section upon receipt of the token while in its waiting state.

The rest of this note deals with the question when and where to send the token. To this end the processes are arranged in a ring, the two circular directions in which are distinguished by "to the left" and "to the right" respectively. The token is sent to the left, so-called signals are sent to the right. A connection – the connection if the number of processes exceeds 2 – between two neighbours in the ring will in the sequel be treated as a three-valued variable: (i) it is unused, (ii) it is carrying a signal to the right, (iii) it is carrying the token to the left. The latter two states are postulated to last only a finite

period of time.

Furthermore, each process is either black or white. Initially, each process is engaged in its noncritical section, the token resides at an arbitrary process, all connections are unused and all processes are white.

Signals are sent to the right, starting at a white waiting process, to call the token to the left. As the signal is propagated processes become black; by sending the token to the left they become white again. (The black processes act like the breadcrumbs of Tom Thumb.) The following algorithm enjoys

P3: the white left-hand neighbour of a black process is waiting

P4: the recipient of a signal is white

P5: the recipient of the token is black or is waiting

P6: a process holding the token while in its noncritical section is white.

The elements of the algorithm are as follows

upon completion of the noncritical section:

```
if in possession of token → enter critical section
[] ¬ in possession of token →
        if black → skip
        [] white → send signal to the right
```

```
        fi; enter waiting state
    fi

upon completion of critical section:
    if white → skip
    [] black → become white and send
                  token to the left
    fi; enter noncritical section

upon receipt of a signal:
if in noncritical section →
     if in possession of token → send token to the left
     [] ¬ in possession of token → become black and send
                                   send signal to the right
     fi
[] ¬ in noncritical section → become black
fi

upon receipt of the token:
if waiting → enter critical section
[] in noncritical section ∧ black →
                become white and send
                token to the left
fi
```

The above has been written down, firstly because I liked the algorithm, and, secondly, because I was not sure how to present it. I still don't know how to do that and how to present its mathematics in an impeccable manner. That experiment is for later.

The point is that I have not given a genuine invariant characterizing all permissible states. The answer is probably coding process states – apart from the possession of the token 3*2 –, coding connection states –3– and enumerating the possible ring states as strings to be characterized by regular expressions. These strings could begin with the process holding or the connection carrying the token. But figuring out how to do that nicely takes more time than I have now.

[0] K. Mani Chandy, Department of Computer Sciences, University of Texas, Austin, Texas 78712, U.S.A., Private Communication, March 1, 1983.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

12 March 1983

prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow