A solution to an examination exercise

We consider a fixed set of (sequential) processes. Each process repeatedly performs an action called its *critical section*, possibly followed by other, noncritical, actions. The problem is to synchronise the processes in such a way that the following two requirements are met.

(0) At any moment in time, the number of processes engaged in their critical sections is at most 1.

(1) No process is engaged in its critical section for the (k+1)-th time before all processes have completed their critical sections for the k-th time, for all natural k.

We are required to solve this problem by means of the technique of the split binary semaphore.

The first step towards a solution is to formalise the specification. We only do this for (1). Requirement (0) is the well-known requirement of *mutual exclusion*. Its solution is well-known too; it follows from the (proper) use of split binary semaphores automatologically.

We number the processes $i : 0 \le i < N$, where N, $0 \le N$, is the (finite) number of processes. For each i, $0 \le i < N$, we introduce an auxiliary variable x_i with the following interpretation:

x_i = "the number of times process i has executed its critical section".

This interpretation is valid provided that initially $(\forall i :: x_i = 0)$ and x_i is increased by 1 each time process i performs its critical section.

Furthermore, we define k in terms of x by

$$k = (\text{MIN}\, i :: x_i) \quad .$$

By definition, we have

(2) $(\forall i :: k \leq x_i)$,

and requirement (1) can now be formalised as the required invariance of Q, with

Q : $(\forall i :: x_i \leq k+1)$.

The invariance of Q is guaranteed if we are able to see to it that $x_p < k+1$ is a precondition of $x_p := x_p + 1$, which is the one and only statement modifying x_p . (We are now developing a program for process p , $0 \leq p < N$; thus, we can use i as a dummy, such as in Q.) By (2), the condition $x_p < k+1$ is equivalent to $x_p = k$.

The following observations, I think, are relevant.

- $x_p < k+1$, and so also $x_p = k$, is stable under the actions of the other processes. So, we only need to worry about its local correctness.
- By (2) and Q we have $k \leq x_p \leq k+1$, which is the same as $x_p = k \lor x_p = k+1$. If $\neg(x_p = k)$ then $x_p = k+1$. So, when false $x_p = k$ can only be truthified by $k := k+1$ which happens when the

last x still equal to k is increased by 1. Formally: the precondition of $k := k+1$ is $(\forall i :: x_i = k+1)$; as a result $k := k+1$ establishes not only $x_p = k$ but also the stronger

(3) $\quad (\forall i :: x_i = k)$.

In a naive application of the technique, one might introduce a split binary semaphore with $N+1$ components: one "general" one and one for each of the preconditions $x_i = k$. The above observation shows that this is overdone: we need only one additional semaphore, namely for condition (3). Therefore, we introduce semaphores m and s, and integer variable b to be —according to the rules of the trade— associated with s, with the following invariants.

$$0 \le m \wedge 0 \le s \wedge m+s \le 1 \wedge 0 \le b \quad , \text{ and}$$
$$s = 0 \vee (b > 0 \wedge (\forall i :: x_i = k)) \quad , \text{ and}$$
$$m = 0 \vee b = 0 \vee \neg(\forall i :: x_i = k) \quad .$$

The remainder of the development of the program is completely standard. Observe that what only matters is the *difference* between x_p and k, and that this difference assumes only 2 values. These values can be represented by booleans. We introduce, therefore,

booleans c_i and d, and
integer n,

coupled to x and k by the following representation invariants. (Variable d is logically superfluous, but it enables us to encode $k := k+1$ as a simple statement.)

$$\begin{array}{l} (\forall i :: c_i \equiv d \equiv x_i = k) \quad , \text{ and} \\ n = (\# i :: c_i \equiv d) \quad . \end{array}$$

Thus, we obtain the following program.

<u>initial state</u> $m=1 \wedge s=0 \wedge b=0 \wedge n=N \wedge (\forall i :: c_i \equiv d)$

<u>process p</u> (read c_p for c: it is a local variable)

$$\begin{array}{l}
\quad\ P.m \\
;\ \underline{\text{if}}\ c \equiv d \rightarrow \text{skip} \\
\quad\ \square\ c \not\equiv d \rightarrow b := b+1\ ;\ V.m\ ;\ P.s\ ;\ b := b-1 \\
\qquad\qquad\qquad \{ (\forall i :: c_i \equiv d) \text{, hence also } c_p \equiv d \} \\
\qquad\qquad\qquad ;\ \underline{\text{if}}\ \ b>0 \rightarrow V.s\ ;\ \{ c_p \equiv d \}\ P.m \\
\qquad\qquad\qquad\quad \square\ \ b=0 \rightarrow \text{skip} \\
\qquad\qquad\qquad\quad \underline{\text{fi}} \\
\quad\ \underline{\text{fi}} \\
\{ c_p \equiv d \} \\
;\ \text{critical section} \\
;\ c, n := \neg c, n-1 \\
\{ c_p \not\equiv d \} \\
;\ \underline{\text{if}}\ n=0 \rightarrow \ d, n := \neg d, N \\
\qquad\qquad\qquad \{ (\forall i :: c_i \equiv d) \} \\
\qquad\qquad\qquad ;\ \underline{\text{if}}\ b>0 \rightarrow V.s\ \ \square\ b=0 \rightarrow V.m\ \ \underline{\text{fi}} \\
\quad\ \square\ n>0 \rightarrow \ \{ \neg(\forall i :: c_i \equiv d) \}\ V.m \\
\quad\ \underline{\text{fi}}
\end{array}$$

□

Eindhoven, 28 march 1991
Rob R. Hoogerwoord

appendix :
the raw code

```
  P.m
; if c ≡ d → skip
  ▯ c ≢ d → b := b+1 ; V.m ; P.s ; b := b-1
            ; if b > 0 → V.s ; P.m  ▯ b = 0 → skip fi
  fi
; critical section
; c, n := ¬c, n-1
; if n = 0 → d, n := ¬d, N
            ; if b > 0 → V.s  ▯ b = 0 → V.m  fi
  ▯ n > 0 → V.m
  fi
```