Misra's proof for Lamport's mutual exclusion

Consider for $i \geq 1$ -legenda will follow the program text- program i :

```
do true → {¬ s.i}
  ; b.i := true {¬ s.i ∧ b.i}
  ; x := i {¬ s.i ∧ b.i}
  ; if y = 0 → s.i := true fi {s.i ∧ b.i}
  ; y := i {s.i ∧ b.i ∧ (y ≠ 0 ∨ x ≠ i)}
  ; if x = i → z := i {P.i}
    [] true →
            b.i := false {s.i}
          ; (∀h :: if ¬ b.h → skip fi) {s.i ∧ (W ∨ y ≠ i)}
          ; if y = i → skip fi {P.i}
    fi {P.i} ; critical section {P.i}
  ; y, s.i := 0, false {¬ s.i}
  ; b.i := false {¬ s.i}
od .
```

Here x and y are shared integers and the b.i are shared booleans, one for each program; z is an integer thought variable, the s.i are boolean thought variables, one for each program. The setting of a thought variable occurs in the same atomic action as the assignment or (successful) test written in the same line. Statements "if $B \rightarrow S$ fi" represent operationally a possible delay.

For the annotation we introduce the abbreviations W, H, and P, for all i given by

$$W \equiv (\underline{A}k :: \neg(b.k \wedge s.k)) \tag{0}$$

$$H.i \equiv b.i \wedge (x=i \vee \neg s.x) \tag{1}$$

$$P.i \equiv s.i \wedge y \neq 0 \wedge ((z=i \wedge H.i) \vee (y=i \wedge W)) \tag{2}$$

From (0) $W \wedge s.i \Rightarrow \neg b.i$, hence from (1) $W \wedge s.i \Rightarrow \neg H.i$, hence from (2)

$$P.i \wedge P.j \wedge W \Rightarrow y=i \wedge y=j \; ; \tag{3}$$

from (2) all by itself

$$P.i \wedge P.j \wedge \neg W \Rightarrow z=i \wedge z=j \; . \tag{4}$$

As the right-hand sides of (3) and (4) imply $i=j$ we conclude

$$P.i \wedge P.j \Rightarrow i=j \; . \tag{5}$$

Furthermore we observe .

$$\begin{array}{ll} & s.i \wedge b.i \wedge x=i \\ \Rightarrow & \{\text{from (0)}\} \\ & \neg W \wedge x=i \wedge s.i \\ = & \{\text{pred.calc}\} \\ & \neg W \wedge x=i \wedge s.x \\ \Rightarrow & \{(1)\} \\ & (\underline{A}k : k \neq i : \neg W \wedge \neg H.k) \\ \Rightarrow & \{(2)\} \\ & (\underline{A}k : k \neq i : \neg P.k) \end{array} \tag{6}$$

We now check the validity of the annotations. The programs are defined to start with $\neg$ s.i . The postconditions of the next three statements are trivially established and not falsified by other programs. Let $k \neq i$.

The postcondition of y:=i is established by this assignment because $i \neq 0$. We can rewrite (6) as

$$s.i \wedge b.i \wedge P.k \Rightarrow x \neq i \quad ;$$

so, if program k falsifies $y \neq 0$, it does so while $x \neq i$. Because $k \neq i$, program k never falsifies $x \neq i$.

The assignment z:=i is executed under $s.i \wedge b.i \wedge y \neq 0 \wedge x = i$, hence establishes P.i . Next we have to verify that program k does not falsify P.i .

(i) Program k does not falsify s.i since $k \neq i$.

(ii) Program k does not falsify $y \neq 0$ on account of (5) and $k \neq i$.

(iii) Program k does not falsify z=i , since its z:=k is executed under $s.k \wedge b.k \wedge x = k$, which, on account of (6) implies $\neg$ P.i since $i \neq k$.

(iv) Program k does not falsify H.i because, firstly, it does not falsify b.i - since $i \neq k$ - , and, secondly, falsification of $(x = i \vee \neg s.x)$ does not occur: x:=k is harmless since it establishes $\neg$ s.x , and truthification of s.x is ruled out under $y \neq 0$.

(v) Program k does not falsify $y=i \wedge W$ since, firstly, the precondition of $y:=k$ implies $\neg W$ and, secondly, W is only falsified under $y=0$.

Statement $(\underline{A}h:: \underline{if} \neg b.h \rightarrow skip \underline{fi})$ establishes for each program h (i included) in some irrelevant order $\neg b.h$ and hence the weaker $\neg(b.h \wedge s.h) \vee y \neq i$. Falsification of the first disjunct only occurs under $y=0$, which implies the second disjunct, while no program k falsifies $y \neq i$. Hence the quantified statement establishes the stable $W \vee y \neq i$.

From the annotation we now see that the test $y=i$ establishes $(s.i \wedge (W \vee y \neq i)) \wedge y=i$, i.e. $s.i \wedge W \wedge y=i$, which on account of (2) and $i \neq 0$ implies $P.i$.

On account of (5), this concludes the proof of mutual exclusion of critical sections.

* * *

The above program has been extracted from

```
start: ⟨b[i] := true⟩;
       ⟨x := i⟩;
       if ⟨y ≠ 0⟩ then ⟨b[i] := false⟩;
                       await ⟨y = 0⟩;
                       goto start fi;
       ⟨y := i⟩;
       if ⟨x ≠ i⟩ then ⟨b[i] := false⟩;
                       for j := 1 to N do await ⟨¬b[j]⟩ od;
                       if ⟨y ≠ i⟩ then await ⟨y = 0⟩;
                                       goto start fi fi;
       critical section;
       ⟨y := 0⟩;
       ⟨b[i] := false⟩
```

as quoted from "A Fast Mutual Exclusion Algorithm", Op.71 of Leslie Lamport. An ingenious solution!

One Tuesday afternoon I presented this extract – but still with the guard $x \neq i$ instead of true – to the ATAC, and we looked at it without much result; and privately I have done so.

At a next session Jayadev Misra presented to the ATAC the outline of the above assertional proof. He and K. Mani Chandy – who may have been involved in its design, I don't know – strongly felt that their experience in "desequentialization" of algorithms should provide the guidance for the introduction of the appropriate thought variables. And so it did! (The introduction of z, however, is food for further thought.)

At that session a (slightly weaker) simplification for the original $P.i$ was suggested, Misra noted that the guard $x \neq i$ could probably be replaced by true and the completion of the proof was discussed.

It then took me another five hours of tying all loose ends together and smoothing the arguments to write down the above. (In the mean time I have learned to accept as quite normal that, after the core of the work has been done, the manuscript still requires somewhat over an hour per page: on the average no more than a few words or a formula per minute is a quite respectable output rate. People that have never actually observed their output rate usually have an entirely erroneous idea of the ergonomics of manuscript production and firmly believe that the mechanics of getting the characters on paper plays a major rôle. Quod non.)

Finally we note that the tests for $x=i$ and $y=i$ are used quite similarly. They share the property that their negations $x \neq i$ and $y \neq i$ are not falsified by program k; in both cases we find the appropriate negation as disjunct in the precondition of the test, and the condition itself as conjunct in the postcondition — be it that $x=i$ is replaced by $z=i$ —.

Austin, 20 November 1985

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 - 1188
USA