An assertional proof of a program by G.L. Peterson.

Consider

program p: { $Lp=0 \wedge \neg inp$ }
do true → $Lp, inp := 1, true$; {P_1: $Lp=1 \wedge inp$}
$Lp, hold := 2, p$; {P_2: $Lp=2 \wedge Hp$}
if $\neg inq \rightarrow Lp := 3$
[] $hold \neq p \rightarrow Lp := 3$
fi ; {P_3: $Lp=3 \wedge Hp$}
Critical Section;
$Lp, inp := 0, false$
od

with $Hp = (hold=p \vee hold=q \wedge Lq=2) \wedge inp$
and the "mirror" of the above with (p, P) and (q, Q) interchanged. Programs p and q share inp, inq, and hold; Lp and Lq are auxiliary variables. The if-fi statement represents a delay until one of the guards is observed to be true.

We observe that the above program p cannot falsify $Hq: (hold=q \vee hold=p \wedge Lp=2) \wedge inq$ for the following reasons
1) it cannot falsify inq
2) when it falsifies $hold=q$, it "verifies" $hold=p \wedge Lp=2$
3) it cannot falsify $hold=p \wedge Lp=2 \wedge inq$, since its truth implies that both guards of $Lp:=3$ are false.
For reasons of symmetry, program q cannot falsify Hp, which program p establishes at $hold:=p$. This establishes the validity of the local assertions. Since

$P3 \wedge Q3 = (Lq=3 \wedge Hp) \wedge (Lp=3 \wedge Hq) \Rightarrow$
$(hold=p) \wedge (hold=q) = F$,
mutual exclusion is guaranteed; $(hold \neq p) \vee (hold \neq q)$ furthermore guarantees the absence of the danger of deadlock.

* *
*

The above solution appears in "Myths about the mutual exclusion problem" (submitted to IPL) by G.L. Peterson (Department of Computer Science, The University of Rochester, ROCHESTER, N.Y. 14627). I think it a very nice solution, much nicer than his justification for it. I quote from Peterson's paper

"The protocols of P_1 and P_2 are given in Figure 1. Q_1 and Q_2 are initially <u>false</u> and TURN may start as either 1 or 2.

/* trying protocol for P_1 */	/* trying protocol for P_2 */
Q1 := true	Q2 := true
TURN := 1	TURN := 2
wait until not Q2 or TURN=2	wait until not Q1 or TURN=1
Critical Section	Critical Section
/* exit protocol for P_1 */	/* exit protocol for P_2 */
Q1 := false	Q2 := false

A Simple Solution
Figure 1

As can be seen, the algorithm has a very simple structure. This results in an easy proof of correctness. First, neither process can be locked out. Consider P_1, it has only one wait loop, and assume it can be forced to remain there forever. After a finite amount of time, P_2 will be doing one of three general things: not trying to enter, waiting in its protocol, or repeatedly cycling through its protocols. In the first case, P_1 notes that Q_2 is false and proceeds. The second case is impossible due to turn being either 1 or 2. In the third case P_2 will quickly set TURN to 2 and never change it back to 1, allowing P_1 to proceed. For mutual exclusion, assume that both processes are in their critical sections at some point. They both could not have succeeded in their wait loops at the same time so one process had to be second. But the value of some variable must change between the tests, and the second process's last step before testing sets TURN to a value which causes it to wait." (End of quotation)

I did not demonstrate the absence of the danger of infinite overtaking. I should have added to that purpose "Since program p cannot falsify Hq, it cannot overtake program q in Q2, since $(Hq \wedge hold = p) \Rightarrow Lp = 2$."

Peterson continues his text with "Since the more complex algorithms naturally require more complex

proofs, one wonders whether the prevalent attitude on "formal" correctness arguments is based on poorly structured algorithms. Perhaps good parallel algorithms are not really all that hard to understand."

I am very much tempted to rephrase the last sentence as "Perhaps good parallel algorithms are not really all that hard to prove to be correct." (and then it could be observed that that statement could be interpreted as my definition of "good"). I must assume to be -like everyone else- biased by my past experience. On account of my past experience with operational arguments à la Peterson, I was most unhappy with his "justification" (which, even if I could reproduce it, I would not dream of presenting to my students). I liked Peterson's solution so much that I felt that it deserved an equally nice correctness proof à la Gries-Owicki, and I was very pleased when I found it. (I was badly out of training: it took me disgracefully long to design it!) It makes two things abundantly clear: firstly why the order of setting "in" and "hold" is relevant, and, secondly, why the order of testing "in" and "hold" -individually!- is irrelevant. Note that the one program "waits until" a condition which, when found true by the one program, may immediately be falsified by the other!

Plataanstraat 5
5671 AL NUENEN
The Netherlands

26 February 1981

prof.dr. Edsger W. Dijkstra
Burroughs Research Fellow.