The superfluity of the general semaphore.

Consider a set of sequential processes that are mutually synchronized by means of the operations V(s) and P(s) on a so-called "general semaphore" s , that is a semaphore that can take on any nonnegative value. We shall show the derivation of a set of equivalent programs in which the semaphores used are binary, i.e. restricted to the values 0 and 1 . This is done by replacing the operations P(s) and V(s) by the text fragments "ps" and "vs" respectively; they operate in a shared environment of integer variables and binary semaphores.

In the following the reader is assumed to be familiar with EWD703 "A tutorial on the split binary semaphore.". The solution to the above problem can be viewed as a further demonstration of the applicability of the methods of EWD703, as their straightforward application leads to the following solution.

The shared environment consists of

```
integer   bm   (initially = the number of processes, which is assumed to be > 0)
integer   bt   (initially = 0)
integer   s    (initially = the initial value of the general semaphore to be
                            simulated)
semaphore  m  (initially = 1)
semaphore  t  (initially = 0)   .
```

The program fragments are

```
ps:    P(m); bm:= bm - 1; bt:= bt + 1; Q;
       P(t); bt:= bt - 1; s:= s - 1; bm:= bm + 1; Q

vs:    P(m); s:= s + 1; Q
```

with Q being short for

```
Q:     if s > 0 and bt > 0 → V(t) [] bm > 0 → V(m) fi   .
```

* * *

Next we have to investigate whether the alternative construct Q , with its guards as derived, can lead to abortion. On account of the invariant

$$bt + bm = \text{number of processes } (> 0)$$

the falsity of the second guard --i.e. bm = 0-- implies

$$bt = \text{number of processes } (> 0) \quad ;$$

if also the first guard is to be false we must, therefore, have $s = 0$. In other words, all processes have performed the prefix of ps and the "semaphore" s equals zero. In the original set of sequential processes this situation would correspond to deadlock. Under the assumption that the original set of sequential processes is free from the danger of deadlock, the danger of abortion in Q is absent.

Our next concern is the potential nondeterminacy of Q : when both guards are true, do we really not care which of the two alternatives is selected? I propose that we do care and wish to give absolute priority to the first alternative because not doing so would have two consequences, both of which I regard as unpleasant. Firstly, it could delay processes in the middle of ps without any reason as far as the current value of s is concerned, and from a strategic point of view that seems hardly attractive. Secondly, the absence of such an unnecessary delay has been shown to have a logical significance for the prevention of individual starvation when the P- and V-operations have been implemented weakly --i.e. in a way that does not exclude unbounded overtaking-- . Hence I propose to strengthen the second guard of Q with the negation of the first:

$$(s = 0 \ \underline{\text{or}}\ bt = 0) \ \underline{\text{and}}\ bm > 0 \rightarrow V(m) \quad .$$

Note. The only effect of this way of strengthening a guard is the reduction --in this case even the removal-- of nondeterminacy; it does not introduce the danger of abortion. (End of Note.)

I furthermore assume that we are not interested in deadlock detection (by abortion or otherwise): if the danger of deadlock is present in the original set of sequential processes, we allow exactly the same form of deadlock in our equivalent programs. That means dropping the term $bm > 0$ from the second guard. Thus our new (and last) Q becomes

```
Q:   if $s > 0$ and $bt > 0$ → {$s > 0$ and $bt > 0$} V(t)
      ▯ $s = 0$ or $bt = 0$ → {$s = 0$ or $bt = 0$} V(m)
     fi
```

* * *

The last phase of our development is one of (more or less) systematic optimization. With our final Q the value of bm is no longer relevant; hence the statements modifying its value can be removed, and so can bm itself. Furthermore, by substituting Q we may be able to simplify its guards differently in the different substitutions. This can be done by associating an assertion with each component of the split binary semaphore. Such an assertion can be taken as postcondition of the corresponding P-operation, provided it is implied by the preconditions of all V-operations on that component (and for the component initialized at 1 also by the initial state). In the following annotation we shall not repeat all the time the general invariant $s \geq 0$ and $bt \geq 0$.

Applying this process to vs we get (when using the precondition of V(m) from the above Q)

```
vs:  P(m) {$s = 0$ or $bt = 0$};
     $s := s + 1$ {$s = 1$ or $bt = 0$};
     if $bt > 0$ → {$s = 1$ and $bt > 0$} V(t)
      ▯ $bt = 0$ → {$s = 0$ or $bt = 0$} V(m)
     fi
```

Note. For the simplification of the guards we have used the general invariant $s \geq 0$. The precondition of V(m) has been weakened from $bt = 0$ to $s = 0$ or $bt = 0$, i.e. the original one. (End of Note.)

Applying the same techniques to ps we get

```
ps:  P(m) {s = 0 or bt = 0};
     bt:= bt + 1 {s = 0 or bt = 1};
     if s > 0 → { s > 0 and bt = 1} V(t)
      ▯ s = 0 → { s = 0 or bt = 0} V(m)
     fi;
     P(t) {(s = 1 and bt > 0) or (s > 0 and bt = 1)};
     bt:= bt - 1; s:= s - 1 {s = 0 or bt = 0};
     V(m)
```

__Note__. For the postcondition of P(t) we have chosen __provisionally__ the disjunction of the preconditions of V(t) from vs and from ps , three lines higher. Since the second Q from ps now reduces to V(m) , we are done! (End of Note.)

We can further shorten the texts

1) by moving the initial increases of s and bt into the alternatives

2) by moving the three statements following the alternative construct in ps into both alternatives

3) by replacing in the first alternative of ps
"bt:= bt + 1; V(t); P(t); bt:= bt - 1"
by a skip

4) by removing from the second alternative of ps the
"bt:= bt - 1; s:= s - 1"
following the only P(t) left and inserting them in front of the only V(t) left

5) by replacing in the first alternative of vs
"s:= s + 1; bt:= bt - 1; s:= s - 1"
by "bt:= bt - 1".

The result of these five successive transformations is

```
ps:  P(m);
     if s > 0 → s:= s - 1 ▯ s = 0 → bt:= bt + 1; V(m); P(t) fi;
     V(m)
```

```
vs:   P(m);
      if bt > 0 → bt:= bt - 1; V(t) ▯ bt = 0 → s:= s + 1; V(m) fi
```

We can shorten our texts still further by first introducing a redundant variable k satisfying

$$k = s - bt$$

a relation that we can keep invariant by inserting into ps: k:= k - 1 and into vs: k:= k + 1 . The values of the guards can then be derived from k , and after removal of the then redundant s and bt we get

```
ps:   P(m); k:= k - 1;
      if k ≥ 0 → skip ▯ k < 0 → V(m); P(t) fi;
      V(m)

vs:   P(m); k:= k + 1;
      if k ≤ 0 → V(t) ▯ k > 0 → V(m) fi
```

Acknowledgment. The presentation of the last transformation is the result of a considerable improvement suggested by A.J.Martin. (End of Acknowledgment.)

Concluding remarks. Firstly I would like to stress that our discovery of the assertion to be associated with component t was just a stroke of luck. If the last Q of ps had not reduced to V(m) and the strongest precondition for the remaining V(t) would have failed to imply the assumed assertion for component t , we would have been in trouble; the assertion has then to be weakened. Apart from this piece of luck, that part of the derivation is entirely satisfactory.

I am less satisfied with the final transformations. They are trivial and easily performed with pencil and rubber or with chalk and eraser. They are, however, most laborious to record in written form. I leave to my reader the choice where to put the blame: on the whole approach that calls for the transformations or on the constraints of the written word. (End of Concluding Remarks.)

Plataanstraat 5
5671 AL NUENEN
The Netherlands

11 April 1980
prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow