Two Applications of the Split Binary Semaphore

0 Introduction

In this note we present two applications of the technique of the split binary semaphore. The first example is a straightforward application of the recipe; in addition, we show how the resulting program can be *cleaned up* in a systematic way. The second example is a not-so-standard application of the technique.

Recall that to reason about semaphore operations in a formal way we use the *Rules of Import and Export*; for semaphore s and predicate Q not containing s these rules are:

import: $\{\ s{=}0 \vee Q\ \}\ \ \mathsf{P}{\cdot}s\ \{\ Q\ \}$

export: $\{\ Q\ \}\ \ \mathsf{V}{\cdot}s\ \{\ s{=}0 \vee Q\ \}$

The rule of import states that Q is a correct postassertion of $\mathsf{P}{\cdot}s$ provided that $s{=}0 \vee Q$ is a correct preassertion of $\mathsf{P}{\cdot}s$; the correctness of the latter is often obtained by turning it into a global invariant. For the invariance of $s{=}0 \vee Q$ we need the rule of export: to guarantee this invariance every $\mathsf{V}{\cdot}s$ must have Q as a preassertion.

1 The Bounded Buffer

This examples pertains to the synchronisation needed by "producers" and "consumers" communicating by means of a (single) bounded buffer. We consider a (possibly large) collection of components performing two kinds of statement, namely $n := n{+}1$ and $n := n{-}1$. The problem is to synchronise these statements in such a way that this relation is invariant:

$$0 \leq n\ \ \wedge\ \ n \leq N\quad ,$$

where $N, 1 \leq N$, is a given constant.

According to the rules of the technique, we calculate the additional preconditions for these statements, and for each precondition thus obtained we introduce a semaphore with an associated integer variable. In this way we obtain:

statement	precondition	semaphore	invariant
$n := n+1$	$n<N$	s, b	$s=0 \ \vee \ (b>0 \wedge n<N)$
$n := n-1$	$0<n$	t, c	$t=0 \ \vee \ (c>0 \wedge 0<n)$

Also, we introduce an additional semaphore m, the neutral component, with invariant:

$$m=0 \ \vee \ ((b=0 \vee n=N) \wedge (c=0 \vee 0=n)) \quad .$$

Semaphores s, t, and m now form a split binary semaphore satisfying:

$$0 \leq s+t+m \leq 1 \quad .$$

For the sake of the initial validity of these invariants we must choose as precondition for the whole program:

$$s=0 \ \wedge \ b=0 \ \wedge \ t=0 \ \wedge \ c=0 \ \wedge \ m=1 \quad .$$

By instantiation of the general scheme, as provided by the standard technique, we obtain the following programs for $n := n+1$ and $n := n-1$:

```
    P·m
;   if  n<N  →  skip
    []  n=N  →  b := b+1 ; V·m ; P·s ; b := b-1
    fi
;  { n<N }
    n := n+1
;   if   b>0 ∧ n<N                      →  V·s
    []   c>0 ∧ 0<n                      →  V·t
    []  (b=0 ∨ n=N) ∧ (c=0 ∨ 0=n)       →  V·m
    fi
```

and:

```
    P·m
;   if  0<n  →  skip
    []  0=n  →  c := c+1 ; V·m ; P·t ; c := c-1
    fi
;  { 0<n }
    n := n-1
;   if   b>0 ∧ n<N                      →  V·s
    []   c>0 ∧ 0<n                      →  V·t
    []  (b=0 ∨ n=N) ∧ (c=0 ∨ 0=n)       →  V·m
    fi
```

We now try to clean up these programs, by simplifying the guards of the V-operations as much as possible. First, we observe that $n := n+1$ admits $0<n$ as a postassertion, and so we may omit conjuncts $0<n$, because they are true, and disjuncts $0=n$, because they are false, from the guards. Similarly, $n := n-1$ admits $n<N$ as a postassertion, and so we may omit conjuncts $n<N$ and disjunts $n=N$. Doing so we obtain the following programs:

```
  P·m
; if  n<N  →  skip
  [] n=N  →  b := b+1 ; V·m ; P·s ; b := b-1
  fi
; { n<N }
  n := n+1
; { 0<n }
  if  b>0 ∧ n<N              →  V·s
  []  c>0                    →  V·t
  []  (b=0 ∨ n=N) ∧ c=0      →  V·m
  fi
```

and:

```
  P·m
; if  0<n  →  skip
  [] 0=n  →  c := c+1 ; V·m ; P·t ; c := c-1
  fi
; { 0<n }
  n := n-1
; { n<N }
  if  b>0                    →  V·s
  []  c>0 ∧ 0<n              →  V·t
  []  b=0 ∧ (c=0 ∨ 0=n)      →  V·m
  fi
```

Second, and this is less simple, we observe that $n := n+1$ never truthifies the condition $n<N$ and, more important, that when initially $b>0 \wedge n=N$ then $n := n-1$ establishes $b>0 \wedge n+1=N$, which is *stronger than* the old $b>0 \wedge n<N$. On account of this I suspect that it should be possible to completely eliminate the alternative $b>0 \wedge n<N \rightarrow$ V·s from the first program. Formally, this elimination requires that $b=0 \vee n=N$ be a preassertion of the selection statement, which in turn requires that $b=0 \vee n+1=N$ be a preassertion of $n := n+1$. Because of the preceding operation P·s, this will only be

feasible if $\mathsf{P}{\cdot}s$ admits $n{+}1{=}N$ as a postassertion; by the rule of import this requires that we strengthen the invariant for semaphore s to:

$$s{=}0 \;\vee\; (b{>}0 \wedge n{+}1{=}N) \quad .$$

In view of the above observations this seems to be a viable proposal and we try to incorporate this into the programs, as follows:

$$\begin{array}{l}
\quad \mathsf{P}{\cdot}m\,\{\; b{=}0 \vee n{=}N \;\} \\
;\;\; \mathsf{if}\;\; n{<}N \;\rightarrow\; \{\; b{=}0 \;\}\; \mathsf{skip} \\
\quad [\!]\;\; n{=}N \;\rightarrow\;\; b := b{+}1 \;;\; \mathsf{V}{\cdot}m \;;\; \mathsf{P}{\cdot}s \;;\; b := b{-}1 \\
\qquad\qquad\qquad \{\; n{+}1{=}N \;\} \\
\quad \mathsf{fi} \\
;\{\; n{<}N \;\}\{\; b{=}0 \vee n{+}1{=}N \;\} \\
\quad n := n{+}1 \\
;\{\; 0{<}n \;\}\{\; b{=}0 \vee n{=}N \;\} \\
\quad \mathsf{if}\;\; c{>}0 \;\rightarrow\;\; \mathsf{V}{\cdot}t \\
\quad [\!]\;\; c{=}0 \;\rightarrow\;\; \mathsf{V}{\cdot}m \\
\quad \mathsf{fi}
\end{array}$$

and:

$$\begin{array}{l}
\quad \mathsf{P}{\cdot}m \\
;\;\; \mathsf{if}\;\; 0{<}n \;\rightarrow\;\; \mathsf{skip} \\
\quad [\!]\;\; 0{=}n \;\rightarrow\;\; c := c{+}1 \;;\; \mathsf{V}{\cdot}m \;;\; \mathsf{P}{\cdot}t \;;\; c := c{-}1 \\
\quad \mathsf{fi} \\
;\{\; 0{<}n \;\} \\
\quad n := n{-}1 \\
;\{\; n{<}N \;\} \\
\quad \mathsf{if}\;\; b{>}0 \qquad\qquad\qquad\quad \rightarrow \{\; n{+}1{=}N \,??\; \}\; \mathsf{V}{\cdot}s \\
\quad [\!]\;\; c{>}0 \wedge 0{<}n \qquad\quad \rightarrow\;\; \mathsf{V}{\cdot}t \\
\quad [\!]\;\; b{=}0 \wedge (c{=}0 \vee 0{=}n) \;\rightarrow\;\; \mathsf{V}{\cdot}m \\
\quad \mathsf{fi}
\end{array}$$

exercise: Verify the correctness of all assertions in these programs, except the one labelled with '??'.
□

The stronger invariant for s requires that every $\mathsf{V}{\cdot}s$ has $b{>}0 \wedge n{+}1{=}N$ as its preassertion, here this is the $\mathsf{V}{\cdot}s$ in the second program. The assertion $n{+}1{=}N$ in the second program is correct provided that $b{=}0 \vee n{+}1{=}N$ is a valid preassertion of the selection, which in turn requires that $b{=}0 \vee n{=}N$

be a valid preassertion of $n := n-1$. For the case $0<n$ in the preceding selection the invariant associated with semaphore m does the job, but for the preceding $\mathsf{P}{\cdot}t$ we can only admit $b=0 \vee n=N$ as a postassertion provided that we strengthen the invariant for semaphore t with this condition:

$$t=0 \;\vee\; (c>0 \wedge 0<n \wedge (b=0 \vee n=N)) \quad .$$

This stronger invariant, in turn, brings about new proof obligations with respect to the operations $\mathsf{V}{\cdot}t$: they must have $b=0 \vee n=N$ as an additional preassertion; the annotation in the first program shows that this additional requirement is met, whereas in the second program we shall, later on, eliminate $\mathsf{V}{\cdot}t$ altogether:

$$\begin{array}{l}
\quad \mathsf{P}{\cdot}m\,\{\ b=0 \vee n=N\ \} \\
;\ \mathsf{if}\ \ n<N \ \rightarrow\ \{\ b=0\ \}\ \mathsf{skip} \\
\quad [\!]\ \ n=N \ \rightarrow\ \ b := b+1\ ;\ \mathsf{V}{\cdot}m\ ;\ \mathsf{P}{\cdot}s\ ;\ b := b-1 \\
\qquad\qquad\qquad \{\ n+1=N\ \} \\
\quad \mathsf{fi} \\
;\ \{\ n<N\ \}\,\{\ b=0 \vee n+1=N\ \} \\
\quad n := n+1 \\
;\ \{\ 0<n\ \}\,\{\ b=0 \vee n=N\ \} \\
\quad \mathsf{if}\ \ c>0 \ \rightarrow\ \ \mathsf{V}{\cdot}t \\
\quad [\!]\ \ c=0 \ \rightarrow\ \ \mathsf{V}{\cdot}m \\
\quad \mathsf{fi}
\end{array}$$

and:

$$\begin{array}{l}
\quad \mathsf{P}{\cdot}m\,\{\ b=0 \vee n=N\ \} \\
;\ \mathsf{if}\ \ 0<n \ \rightarrow\ \ \mathsf{skip} \\
\quad [\!]\ \ 0=n \ \rightarrow\ \ c := c+1\ ;\ \mathsf{V}{\cdot}m\ ;\ \mathsf{P}{\cdot}t\ ;\ c := c-1 \\
\qquad\qquad\qquad \{\ b=0 \vee n=N\ \} \\
\quad \mathsf{fi} \\
;\ \{\ 0<n\ \}\,\{\ b=0 \vee n=N\ \} \\
\quad n := n-1 \\
;\ \{\ n<N\ \}\,\{\ b=0 \vee n+1=N\ \} \\
\quad \mathsf{if}\ \ b>0 \qquad\qquad\qquad\quad \rightarrow \{\ n+1=N\ \}\ \mathsf{V}{\cdot}s \\
\quad [\!]\ \ c>0 \wedge 0<n \qquad\qquad \rightarrow \{\ ??\ \}\ \mathsf{V}{\cdot}t \\
\quad [\!]\ \ b=0 \wedge (c=0 \vee 0=n) \ \rightarrow\ \ \mathsf{V}{\cdot}m \\
\quad \mathsf{fi}
\end{array}$$

All this served to eliminate the alternative $b>0 \wedge n<N \rightarrow \mathsf{V}{\cdot}s$ from the first program. In an entirely similar way we can eliminate the alternative

$c{>}0 \wedge 0{<}n \rightarrow \mathsf{V}{\cdot}t$ from the second program. This brings about a similar need to strengthen the invariants even further, yielding:

$$t{=}0 \;\vee\; (c{>}0 \wedge 1{=}n \wedge (b{=}0 \vee n{=}N)) \quad , \text{ and:}$$

$$s{=}0 \;\vee\; (b{>}0 \wedge n{+}1{=}N \wedge (c{=}0 \vee 0{=}n)) \quad ,$$

whereas the invariant associated with m remains what it was:

$$m{=}0 \;\vee\; ((b{=}0 \vee n{=}N) \wedge (c{=}0 \vee 0{=}n)) \quad .$$

In this way we obtain our final version of the programs; with all assertions omitted these programs are quite simple:

```
  P·m
; if  n<N  →  skip
  [] n=N  →  b := b+1 ; V·m ; P·s ; b := b−1
  fi
; n := n+1
; if  c>0  →  V·t
  [] c=0  →  V·m
  fi
```

and:

```
  P·m
; if  0<n  →  skip
  [] 0=n  →  c := c+1 ; V·m ; P·t ; c := c−1
  fi
; n := n−1
; if  b>0  →  V·s
  [] b=0  →  V·m
  fi
```

exercise: To train your ability verify the validity of the above three invariants by (re)constructing, from scratch, all necessary assertions in the programs.
□

This example shows that programs constructed by means of the standard technique can be cleaned up in a very systematic way, but it also shows that this transformation can be quite laborious. Moreover, it is not always clear

at the beginning that the process of repeatedly strengthening the invariants will converge at all. It may help, though, to look at the programs with an operational eye, as this may give a clue to what simplifications should be possible.

2 The Leader and the Gang

We consider a collection of named components called *gangsters* and one additional component called *the leader*. The leader has the shape $*[\ y := y+1\ ]$, where y is a shared variable; gangster p has the shape $*[\ x_p := x_p+1\ ]$, where x_p is a private variable of gangster p, for every p from the collection. The problem is to synchronise these components in such a way that the gangsters follow the leader but do not overtake it, that is the problem is to guarantee the invariance of:

$Q:\quad (\forall i :: x_i \leq y)$.

We assume that Q holds initially. Statement $y := y+1$ does not violate Q and the additional precondition for $x_p := x_p+1$ is, of course, $x_p < y$. Hence, different gangsters have different preconditions and, therefore, in a naive application of the technique we would introduce as many semaphores as there are gangsters; more precisely, for every ganster p we would introduce semaphore s_p with additional boolean variable b_p, with invariant:

$$s_p=0 \ \vee \ (b_p \wedge x_p<y) \quad .$$

remark: I have used boolean variables instead of integers because the operation $\mathsf{P}{\cdot}s_p$ will occur in only one component, namely gangster p: the integer variables would assume the values 0 and 1 only. Instead of $b=0$ and $b>0$ we now shall use $\neg b$ and b respectively, and $b := b+1$ and $b := b-1$ will become $b := \mathsf{true}$ and $b := \mathsf{false}$.

□

Next, we introduce a neutral component semaphore m, as usual; the invariant associated with m is obtained by taking the conjunction of the negations of the conditions $b_p \wedge x_p < y$, entirely along the lines of the standard technique:

$$m=0 \ \vee \ (\forall i :: \neg b_i \vee x_i=y) \quad .$$

In this way we would obtain programs with very many V-operations. We can, however, do better: because of Q the statement $y := y+1$ establishes

$(\forall i :: x_i{<}y)$ and this implies $x_p{<}y$ for every p. This gives rise to the idea that a single semaphore might be sufficient, instead of many semaphores. Formally, the argument runs as follows. We could strengthen the invariants for the semaphores s_p to:

$$s_p{=}0 \;\vee\; (b_p \wedge (\forall i :: x_i{<}y)) \quad ,$$

but that is too strong: $(\forall i :: x_i{<}y)$ is violated by any of the $x_q := x_q{+}1$. In the presence of conjunct b_p, however, we may weaken this relation as follows:

$$s_p{=}0 \;\vee\; (b_p \wedge (\forall i :: \neg b_i \vee x_i{<}y)) \quad .$$

This weaker invariant is still sufficient to conclude the correctness of the postassertion in: $\mathsf{P}{\cdot}s_p\,\{\; b_p \wedge x_p{<}y \;\}$; moreover, it is viable because the offending statements $x_q := x_q{+}1$ have $\neg b_q$ as their precondition (as we will see).

Next, we combine all semaphores x_p into a single new semaphore, that is, we "unsplit" the semaphores s_p; calling the new semaphore r we obtain as invariants for r and m:

$$r{=}0 \;\vee\; ((\exists i :: b_i) \wedge (\forall i :: \neg b_i \vee x_i{<}y))$$

$$m{=}0 \;\vee\; (\forall i :: \neg b_i \vee x_i{=}y) \quad .$$

With the aid of these invariants the following programs can be constucted quite easily. Notice that, in the program for gangster p, I have distributed the statement $x_p := x_p{+}1$ and the ensuing V-operation over the alternatives of the preceding selection; the annotation shows why this was a good idea. (This has to do with the fact that, as we have strengthened the invariant, the conditions associated with r and m are not complementary anymore.)

gangster p:

```
  P·m { ¬b_p } { (∀i :: ¬b_i ∨ x_i=y) }
; if  x_p<y  →  x_p := x_p+1 ; { (∀i :: ¬b_i ∨ x_i=y) }  V·m
  []  x_p=y  →  b_p := true ; V·m ; P·r ; b_p := false
                ; { ¬b_p } { (∀i :: ¬b_i ∨ x_i<y) }
                  x_p := x_p+1
                ; { (∀i :: ¬b_i ∨ x_i<y) }
                  if  (∃i :: b_i)   →  V·r
                  []  (∀i :: ¬b_i)  →  V·m
                  fi
  fi
```

the leader:

$$
\begin{array}{ll}
 & \mathsf{P}{\cdot}m\,\{\ \neg b_p\ \}\,\{\ (\forall i :: \neg b_i \vee x_i{=}y)\ \} \\
; & y := y{+}1 \\
; & \{\ (\forall i :: x_i{<}y)\ \} \\
 & \mathsf{if}\ \ (\exists i :: b_i) \ \ \rightarrow\ \ \mathsf{V}{\cdot}r \\
 & [\!]\ \ (\forall i :: \neg b_i)\ \rightarrow\ \ \mathsf{V}{\cdot}m \\
 & \mathsf{fi}
\end{array}
$$

Finally, the boolean variables b_p can be reduced to auxiliary variables, and eliminated from the program, at the expense of one additional integer variable c coupled to the booleans by:

$$c \ = \ (\#i :: b_i) \quad .$$

* * *

An alternative solution to this problem is obtained if we exploit the fact that the condition $x_p{<}y$ is stable, that is, globally correct: this enables us to place the synchronisation, establishing the condition, and the action proper, $x_p := x_p{+}1$, into two separate critical sections. This means that the first part of the program for gangster p only implements $\mathsf{if}\ x_p{<}y \rightarrow \mathsf{skip}\ \mathsf{fi}$; as a result we can now use the stronger invariants —the ones we had to reject above—. I leave it as an exercise to verify that this gives rise to the following invariants and programs. These invariants can be formulated right from the start, that is, without the detour via a semaphore per gangster:

invariants:

$$r{=}0 \ \vee \ (c{>}0 \wedge (\forall i :: x_i{<}y))$$

and:

$$m{=}0 \ \vee \ c{=}0 \ \vee \ (\exists i :: x_i{=}y)$$

ganster p:

```
    P·m { c=0 ∨ (∃i :: x_i=y) }
;   if  x_p<y  →  V·m
    []  x_p=y  →  c := c+1 ; V·m ; P·r ; c := c−1
                  ; { (∀i :: x_i<y) } { hence: x_p<y }
                    if  c>0  →  V·r
                    []  c=0  →  V·m
                    fi
    fi
; { x_p<y }
    P·m ; x_p := x_p+1 ; V·m
```

the leader:

```
    P·m
;   y := y+1
; { (∀i :: x_i<y) }
    if  c>0  →  V·r
    []  c=0  →  V·m
    fi
```

Eindhoven, 8 february 1995

Rob R. Hoogerwoord
department of mathematics and computing science
Eindhoven University of Technology
postbus 513
5600 MB Eindhoven