A refinement proof for Udding's semaphore program

Wim H. Hesselink (whh456), Mark IJbema, 2011

Dept. of Computing Science, University of Groningen
P.O.Box 407, 9700 AK Groningen, The Netherlands

1 Introduction

This technical report is a companion paper of [2], in which the three semaphore programs of [4,3,5] are shown to be implementations of a single more abstract program. As the treatment of Udding's algorithm [5] is largely similar to the treatment of Morris' program [4], we referred the formal proof for Udding's algorithm to this report. See [1] for the PVS proof script.

The abstract program uses the shared variables:

$$\textbf{var}\ \texttt{se}, \texttt{sm} : \mathbb{N}\ ,$$
$$\textbf{var}\ \texttt{ne}, \texttt{nm} : \mathbb{Z}\ .$$

The variables `se` and `sm` serve as semaphores that model the doors of the elevator at the two floors. The variables `ne` and `nm` count the number of processes outside and within the elevator. The initial conditions are

$$\texttt{ne} = \texttt{nm} = \texttt{sm} = 0\ \wedge\ \texttt{se} = 1\ .$$

We let p range over the processes. The code for process p is:

```
(0)      process (p) =
           loop
 9           NCS ;
10           ne++ ;
11           await se > 0 ; nm++ ; ne-- ;
             if ne = 0 then sm++ ; se-- endif ;
12           await sm > 0 ; sm-- ; nm-- ;
13           CS ;
14           if nm > 0 then sm++ else se++ endif
           endloop .
```

The numbered commands are regarded as atomic. The lines 10, 11, 12, and 14, are executed by the system, while the lines 9 and 13 are executed by the environment.

It is shown in [2] that this abstract algorithm guarantees mutual exclusion, is deadlock-free, and has bounded overtaking with bound 2. The problem with it is that it violates the principle of single critical reference: the shared variable `ne` is both read and written in line 10, and in each of the lines 11, 12, and 14 several shared variables are read and written.

2 Udding's algorithm

In [5], Udding derived an algorithm to establish starvation-free mutual exclusion. The algorithm uses a buffered semaphore `sb`, and two plain semaphores `sm` and `se`. The algorithm uses integer shared variables `ne` and `nm`. The initial values are:

$$\texttt{sb} = \texttt{se} = 1\ \wedge\ \texttt{sm} = \texttt{ne} = \texttt{nm} = 0\ .$$

The semaphores sb and se combine to take over the roles of the shared variable se of the abstract algorithm (0).

```
process (p) =
    loop
        NCS ;
        P(sb) ; ne++ ; V(sb) ;
        P(se) ; P(sb) ; nm++ ; ne-- ;
        if ne > 0 then V(sb) else V(sm) endif ;
        V(se) ;
        P(sm) ; nm-- ; CS ;
        if nm > 0 then V(sm) else V(sb) endif
    endloop .
```

In this case, the pair (sb, sm) forms a split binary semaphore, where the first one is buffered and the second one is plain. The third semaphore, se, is only needed to ensure bounded overtaking. We show this by a scenario in the next section.

2.1 Udding's transition system

The above algorithm is formalized in the transition system (1). The variable buf holds the buffer of the buffered semaphore sb. The lines 10, 11 and 16, 17 correspond to the actions **P**(sb). The actions **V**(sb) are contained in the lines 14, 22, and 28.

Just as with Morris' algorithm, we have replaced inspections of shared variables in the lines 22 and 28 by inspections of the private variable *tmp*.

Remark. If one removes the semaphore se and replaces the lines 15 and 23 by skip, individual starvation is possible. This is shown by the following scenario for three processes, q_0, q_1, and q_2.

The scenario starts with all three processes at 9. It proceeds in 9 stages.
1. Process q_0 goes to 14 and sets ne := 1.
2. The processes q_1 and q_2 go to line 11 and enter buf. Note the resulting state, we shall see it again.
3. Process q_0 executes 14, removes q_1 from buf, proceeds to 17, and enters buf.
4. Process q_1 goes from 11 to 14, removes q_0 from buf, goes to 17, and enters buf.
5. Process q_0 goes to 22, removes q_1 from buf, and starts waiting at 24.
6. Process q_1 goes to 22 with $tmp = 0$, increments sm, and arrives at 24. Now q_0 and q_1 are at 24, sm $= 1$, and nm $= 2$.
7. Process q_0 passes through *CS* and goes via 28, 9, and 10 to 11, and enters buf.
8. Process q_1 also passes through *CS* and goes via 28, where it removes q_0 from buf, and via 9 and 10 to 11, and enters buf.
9. Process q_0 goes to 13 and sets ne := 1.

Here, we are in the state after stage 2. Therefore, process q_2 can suffer individual starvation if semaphore se is removed. Semaphore sb precludes this scenario because it does not allow process q_1 in stage 4 to proceed to 17. □

The proofs of mutual exclusion and absence of deadlock in Udding's algorithm is largely similar to the case of Morris' algorithm. The invariants used are minor variations, see [2].

The invariants for the split binary semaphore (sb, sm) and the binary semaphore se are:

Iq0: $\#\{q \mid q \notin \texttt{buf} \ \wedge \ q \textbf{ in } \{11 \ldots 14, 17 \ldots 22, 25 \ldots 28\}\} = 1 - \texttt{sb} - \texttt{sm}$,
Iq1: $\#\{q \mid q \textbf{ in } \{16 \ldots 23\}\} = 1 - \texttt{se}$.

The buffered semaphore sb induces the invariants:

(1)
```
process ($p$) =
    var $tmp : \mathbb{Z}$ ;
    loop
 9      $NCS$ ;
10      if $\mathtt{sb} > 0$ then sb-- else
            $\mathtt{buf} := \mathtt{buf} \cup \{p\}$ ;
11          await $p \notin \mathtt{buf}$ endif ;
12      $tmp := \mathtt{ne} + 1$ ;
13      $\mathtt{ne} := tmp$ ;
14      if possible remove some $q$ from buf else sb++  endif ;
15      await $\mathtt{se} > 0$ ; se-- ;
16      if $\mathtt{sb} > 0$ then sb-- else
            $\mathtt{buf} := \mathtt{buf} \cup \{p\}$ ;
17          await $p \notin \mathtt{buf}$ endif ;
18      $tmp := \mathtt{nm} + 1$ ;
19      $\mathtt{nm} := tmp$ ;
20      $tmp := \mathtt{ne} - 1$ ;
21      $\mathtt{ne} := tmp$ ;
22      if $tmp > 0$ then
            if possible remove some $q$ from buf else sb++  endif ;
        else sm++ endif ;
23      se++ ;
24      await $\mathtt{sm} > 0$ ; sm-- ;
25      $tmp := \mathtt{nm} - 1$ ;
26      $\mathtt{nm} := tmp$ ;
27      $CS$ ;
28      if $tmp > 0$ then sm++
        elsif possible remove some $q$ from buf else sb++  endif ;
    endloop .
```

Iq2: $q \in \mathtt{buf} \;\Rightarrow\; q \textbf{ in } \{11, 17\}$,
Iq3: $\mathtt{sb} > 0 \;\Rightarrow\; \mathtt{buf} = \emptyset$.

Preservation of *Iq0* follows from *Iq2*. Mutual exclusion is implied by *Iq0*.

The variables ne and nm count the number of processes in the obvious regions:

Jq0: $\#\{q \mid q \textbf{ in } \{14 \ldots 21\} = \mathtt{ne}$,
Jq1: $\#\{q \mid q \textbf{ in } \{20 \ldots 26\} = \mathtt{nm}$.

The private variables *tmp* hold values related to ne or nm:

Jq2: $(q \textbf{ at } 13 \;\Rightarrow\; tmp.q = \mathtt{ne} + 1) \;\wedge\; (q \textbf{ at } 21 \;\Rightarrow\; tmp.q = \mathtt{ne} - 1)$,
Jq3: $(q \textbf{ at } 19 \;\Rightarrow\; tmp.q = \mathtt{nm} + 1) \;\wedge\; (q \textbf{ at } 26 \;\Rightarrow\; tmp.q = \mathtt{nm} - 1)$.

Preservation of *Jq0* follows from *Jq2*. Preservation of *Jq1* follows from *Jq3*. Preservation of *Jq2* and *Jq3* follows from *Iq0*.

The critical invariants against deadlock are:

Jq4: $\mathtt{sm} > 0 \;\Rightarrow\; \exists\, r : r \textbf{ in } \{23, 24\}$,
Jq5: $\mathtt{sb} > 0 \;\wedge\; q \textbf{ in } \{23, 24\} \;\Rightarrow\; \exists\, r : r \textbf{ in } \{15, 16\}$.

Preservation of *Jq4* at the lines 24 and 28 follows from *Iq0*, *Jq1*, and *Jq6* below. Preservation *Jq5* at lines 16, 22, and 28 follows from *Iq0*, *Iq2*, *Iq3*, *Jq0*, *Jq1*, and *Jq6*.

Jq6: $(q \textbf{ at } 22 \;\Rightarrow\; tmp.q = \mathtt{ne}) \;\wedge\; (q \textbf{ in } \{27, 28\} \;\Rightarrow\; tmp.q = \mathtt{nm})$.

Preservation of *Jq6* follows from *Iq0*.

Absence of immediate deadlock is proved as follows. Assume that all processes are idle or disabled. Then they are all at 9, 11, 15, 17 or 24. Those at 11 or 17 are elements of `buf`. By *Iq0*, it follows that $sb + \mathtt{sm} = 1$. If $\mathtt{sm} > 0$, *Jq4* implies there is a process r at 24, which is necessarily enabled. It follows that $\mathtt{sm} = 0$, and hence $\mathtt{sb} = 1$. From this it follows that `buf` is empty by *Iq3*. Therefore, all processes are at 9, 15, or 24. By *Iq1*, it follows that $\mathtt{se} = 1$. Therefore, no processes are disabled at 15. This proves that all processes are at 9 or 24. Now, *Jq5* implies that there are no processes at 24. This proves that all processes are idle.

If there are only finitely many processes, the absence of immediate deadlock implies deadlock-freedom, i.e., whenever there are competing processes, eventually one of these will reach *CS* and exit. This is proved in the same way as for the abstract algorithm in [2, Section 4].

2.2 Udding's refinement

We show that Udding's algorithm (1) implements the abstract algorithm (0) in such a way that the environment actions and the doorways of both algorithms correspond. This implies that it also has bounded overtaking with bound 2. We prove it by constructing a simulation in the same way as for Morris' algorithm in [2].

As the environment actions and the doorways must correspond, the lines 9, 10, and 11 of both algorithms must correspond, and the lines 27 and 28 of (1) must correspond with the lines 13 and 14 of (0). This leaves the question where line 12 of (0) should be located in (1).

To answer this question, we consider a simple scenario. Assume that process t_0 proceeds from line 9 to line 18. Then process t_1 enters and arrives at line 11 with $t_1 \in \mathtt{buf}$. Process t_0 proceeds, and observes $tmp.t_0 = 0$ at line 22. It therefore enters *CS* and exits. In the abstract version, this means that process t_0 completes line 11 before process t_1 executes line 10. This means that the abstract step 11 must be associated with a concrete step not later than 16.

On the other hand, when process t_1 enters when process t_0 is still at 16, then process t_1 will increments `ne`, and it can pass t_0 at line 24. We therefore decide to associate the end of the abstract step 11 with the concrete step 16.

We thus define the abstract program counter on the concrete state space by

$$\begin{aligned} apc.q = (&pc.q \leq 10 \;?\; pc.q \\ &:\; pc.q \leq 16 \;?\; 11 \\ &:\; pc.q \leq 26 \;?\; 12 \;\; : pc.q - 14)\ . \end{aligned}$$

Now that we have localized the abstract steps in the concrete algorithm, we can determine how that abstract shared variables `ne`, `nm`, `se`, and `sm` must be modified in the concrete algorithm. We introduce shared integer ghost variables `qne`, `qnm`, `qse`, and `qsm` in the concrete algorithm to plays the roles aof the abstract shared variables. They have the initial values of their abstract counterparts:

$$\mathtt{qne} = \mathtt{qnm} = \mathtt{qsm} = 0 \;\wedge\; \mathtt{qse} = 1\ .$$

In the concrete algorithm, they are modified at the lines 10, 16, 26, and 28, in the same way as their abstract counterparts in the abstract lines 10, 11, 12, and 14:

```
10a   qne++ ;
16a   qne-- ; qnm++ ;
      if sb > 0 ∧ ne = 1 then qse-- ; qsm++ endif ;
26a   qnm-- ; qsm-- ;
28a   if tmp > 0 then qsm++ else qse++ endif .
```

Strictly speaking, the extension of algorithm (1) with these ghost variables is a forward simulation (or history extension). The next step is to form a refinement function from the extended algorithm towards the abstract algorithm (0). This function forgets the implementation variables and promotes the ghost variables to their abstract counterparts. Just as for Morris' algorithm in [2], it can be described by

$$\begin{aligned}&f(x) = (\#\\&\quad \mathtt{ne} := x.\mathtt{qne}\ ,\ \mathtt{nm} := x.\mathtt{qnm}\ ,\\&\quad \mathtt{se} := x.\mathtt{qse}\ ,\ \mathtt{sm} := x.\mathtt{qsm}\ ,\\&\quad pc := x.apc\ \#)\ .\end{aligned}$$

In order to prove that this is a refinement function, we obtain with PVS the following proof obligations:

PO0: $q\ \mathbf{at}\ 16\ \Rightarrow\ (\mathtt{qne} = 1 \equiv (\mathtt{sb} > 0\ \wedge\ \mathtt{ne} = 1))\ ,$
PO1: $q\ \mathbf{at}\ 28\ \Rightarrow\ (tmp.q > 0 \equiv \mathtt{qnm} > 0)\ ,$
PO2: $q\ \mathbf{at}\ 16\ \Rightarrow\ \mathtt{qse} > 0\ ,$
PO3: $q\ \mathbf{at}\ 26\ \Rightarrow\ \mathtt{qsm} > 0\ .$

Proof obligation *PO0* follows from *Iq0*, *Iq1*, *Iq3*, *Jq0*, and the new invariants:

Kq0: $\#\{q \mid q\ \mathbf{in}\ \{11 \ldots 16\}\ \} = \mathtt{qne}\ ,$
Kq1: $q\ \mathbf{in}\ \{15, 16\}\ \wedge\ \mathtt{sb} = 0$
$\Rightarrow\ (\exists\, r : r\ \mathbf{in}\ \{11 \ldots 14\}\ \vee\ (r\ \mathbf{in}\ \{17 \ldots 22\}\ \wedge\ r \notin \mathtt{buf}))\ .$

Here, *Kq0* is obvious, but *Kq1* is proved using *Iq2*, *Jq0* and *Jq6* at the lines 14, 16, and 22.

The proof of *PO0* goes as follows. If $\mathtt{sb} > 0$, then *Iq3* and *Iq0* imply the absence of processes in the regions 11–14 and 17–22. Therefore, *Jq0* and *Kq0* give $\mathtt{ne} = \mathtt{qne}$. Otherwise $\mathtt{sb} = 0$, and *Kq1* and *Iq1* together imply the existence of a thread in region 11–14; as q is at 16, it follows from *Kq0* that $\mathtt{qne} > 1$.

Proof obligation *PO1* is strengthened to

$$q\ \mathbf{at}\ 28\ \Rightarrow\ tmp.q = \mathtt{qnm}\ .$$

This follows from *Jq6*, *Jq1*, and the new invariants

Kq2: $\#\{q \mid q\ \mathbf{in}\ \{17 \ldots 26\}\ \} = \mathtt{qnm}\ ,$
Kq3: $q\ \mathbf{in}\ \{11 \ldots 22\}\ \wedge\ r\ \mathbf{in}\ \{25 \ldots 28\}\ \Rightarrow\ q\ \mathbf{at}\ 11\ \wedge\ q \in \mathtt{buf}\ .$

Again, *Kq2* is straightforward, but the proof of *Kq3* is rather delicate and needs the additional invariant:

Kq4: $q\ \mathbf{in}\ \{11 \ldots 22\}\ \wedge\ \mathtt{sm} > 0\ \Rightarrow\ q\ \mathbf{at}\ 11\ \wedge\ q \in \mathtt{buf}\ .$

Proof obligation *PO2* follows from *Iq2* and the new invariant:

Kq5: $q\ \mathbf{in}\ \{11 \ldots 16\}\ \wedge\ q \notin \mathtt{buf}\ \Rightarrow\ \mathtt{qse} > 0\ .$

In order to prove this we also need to prove the invariants

Kq6: $\mathtt{sb} \leq \mathtt{qse}\ ,$
Kq7: $q\ \mathbf{in}\ \{21 \ldots 22\}\ \wedge\ tmp.q > 0\ \Rightarrow\ \mathtt{qse} > 0\ ,$
Kq8: $q\ \mathbf{in}\ \{18 \ldots 21\}\ \wedge\ \mathtt{ne} > 1\ \Rightarrow\ \mathtt{qse} > 0\ ,$
Kq9: $q\ \mathbf{at}\ 17\ \Rightarrow\ \mathtt{qse} > 0\ .$

The invariant *Iq1*, which is associated with the semaphore `se`, serves heavily in the proofs of the last three invariants.

The final proof obligation *PO3* is generalized in the invariant:

Lq0: q **in** $\{25, 26\} \Rightarrow \texttt{qsm} > 0$.

In order to prove this, we also need to prove the invariants

Lq1: $\texttt{sm} \leq \texttt{qsm}$,
Lq2: q **in** $\{21, 22\} \wedge \mathit{tmp} \leq 0 \Rightarrow \texttt{qsm} > 0$,
Lq3: q **in** $\{18 \ldots 21\} \wedge \texttt{ne} = 1 \Rightarrow \texttt{qsm} > 0$,
Lq4: q **at** $17 \wedge q \notin \texttt{buf} \Rightarrow \texttt{ne} > 1$.

This completes the proof that f is a refinement function from system (1) extended with ghost variables to system (0).

The composition of the forward simulation from system (1) to its extension with ghost variables with the refinement function to system (0) gives us a simulation from Udding's transition system (1) to the abstract system (0). This simulation respects the processes and their doorways and leaves the environment actions unchanged. If some process p in some execution of (1) overtakes another process k times, the simulation gives us an execution of (0) in which process p overtakes the other process k times. It follows that $k \leq 2$. Therefore system (1) has bounded overtaking with bound 2.

References

1. W.H. Hesselink. Starvation-free mutual exclusion with semaphores. `http://wimhesselink.nl/mechver/fairMXsema.html`, 2011.
2. W.H. Hesselink and M. IJbema. Starvation-free mutual exclusion with semaphores. *Formal Aspects of Comput.*, 25:947–969, 2013. doi: 10.1007/s00165-011-0219-y.
3. A.J. Martin and J.R. Burch. Fair mutual exclusion with unfair P and V operations. *Inf. Process. Lett.*, 21:97–100, 1985.
4. J.M. Morris. A starvation-free solution to the mutual exclusion problem. *Inf. Process. Lett.*, 8:76–80, 1979.
5. J.T. Udding. Absence of individual starvation using weak semaphores. *Inf. Process. Lett.*, 23:159–162, 1986.