Tik-79.231 **Autumn 2000**
Parallel and distributed digital systems
Answers to Tutorial 5
Nov 3, 2000

1. Let us create the incidence matrix of the net (the columns indicate the changes in the place markings by transitions are)

$$C = \begin{pmatrix} -1 & 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

The vector representation of the column ι is an invariant iff $C^T \cdot \iota = 0$, where 0 is a [column] vector.

a) Our task is to prove the following with invariants: $M(p_4) \leq 2$.
Let us make the guess that one of the invariants is $\iota = M(p_2) + M(p_4)$. Hence, we obtain the ι vector, where the multipliers for both places are 1. Let us check that our vector really is an invariant:

$$C^T \cdot \iota = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

So $\iota = M(p_2) + M(p_4)$ really is an invariant. Then we will compute, using the initial marking M_0, the value of the invariant.

$$\iota^T \cdot M_0 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 2$$

The result was $M(p_2) + M(p_4) = 2 \Rightarrow M(p_4) \leq 2$, which means that at most 2 processors can access the memory concurrently.

b) First we will prove that $\iota = M(p_1) + M(p_3) + M(p_4) + M(p_5)$ is an invariant. Let us check it:

$$C^T \cdot \iota = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

As the next step we will compute the value of the invariant, again basing on the initial marking:

$$\iota^T \cdot M_0 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 5$$

Hence we have proved the following formula to be the invariant: $M(p_1) + M(p_3) + M(p_4) + M(p_5) = 5$.

2. A PROD description of the TicketME algorithm:

```
#ifndef nprocs
#define nprocs 2
#endif nprocs

#define NIL 0

#enum REM, ENTER, TEST, LEAVE_T, CRIT, RESET, LEAVE_E

#define ADDMOD(x) (1 + ((x) % (nprocs)))


#place ids lo(<.1.>) hi(<.nprocs.>) mk(<.1..nprocs.>)

#place turn lo(<.1,1.>) hi(<.nprocs,nprocs.>) mk(<.1,1.>)

#place procs lo(<.1,LEAVE_E.>) hi(<.nprocs,REM.>) \
mk(<.1..nprocs,REM.>)

#place tickets lo(<.1,NIL.>) hi(<.nprocs,nprocs.>) \
mk(<.1..nprocs,NIL.>)


#trans try
   in  { procs: <.pid,REM.>; }
   out { procs: <.pid,ENTER.>; }
#endtr

#trans enter
   in  { procs: <.pid,ENTER.>; turn: <.next,granted.>;
         tickets: <.pid,tick.>; }
   out { procs: (next == granted)<.pid,LEAVE_T.> +
                (next != granted)<.pid,TEST.>;
         turn: <.ADDMOD(next),granted.>;
         tickets: <.pid,next.>; }
#endtr

#trans testtick
   in  { procs: <.pid,TEST.>; turn: <.next,granted.>;
         tickets: <.pid,tick.>; }
   out { procs: (tick == granted)<.pid,LEAVE_T.> +
                (tick != granted)<.pid,TEST.>;
         turn: <.next,granted.>;
         tickets: <.pid,tick.>; }
#endtr

#trans crit
   in  { procs: <.pid,LEAVE_T.>; }
   out { procs: <.pid,CRIT.>; }
#endtr

#trans exit
   in  { procs: <.pid,CRIT.>; }
   out { procs: <.pid,RESET.>; }
#endtr

#trans reset
   in  { procs: <.pid,RESET.>; turn: <.next,granted.>;
         tickets: <.pid,tick.>; }
   out { procs: <.pid,LEAVE_E.>; turn: <.next,ADDMOD(granted).>;
```

```
        tickets: <.pid,NIL.>; }
#endtr

#trans rem
   in  { procs: <.pid,LEAVE_E.>; }
   out { procs: <.pid,REM.>; }
#endtr
```

(a) Let the above PROD description be in the file `tick.net`. Let us present:

- The property as an LTL formula,
- The property as a PROD verify formula,
- Is the property valid?
- An example of a model checking run.

- Two processes are never in the critical section concurrently:

$$\Box(\neg(procs(\langle 1, CRIT\rangle) \wedge procs(\langle 2, CRIT\rangle)))$$

```
#verify henceforth(!(procs >= (<.1,CRIT.> + <.2,CRIT.>)));
```

The property holds.

```
birdy:~/ticket> cat formula1
#verify henceforth(!(procs >= (<.1,CRIT.> + <.2,CRIT.>)));
birdy:~/ticket> cat tick.net formula1 > tick_formula1.net
birdy:~/ticket> prod tick_formula1.init
birdy:~/ticket> ./tick_formula1 -.
........
birdy:~/ticket>
```

- The process 1 never makes it to the critical section:

$$\Box(\neg(procs(\langle 1, CRIT\rangle)))$$

```
#verify henceforth(!(procs >= <.1,CRIT.>));
```

The property does not hold.

```
birdy:~/ticket> cat tick.net formula2 > tick_formula2.net
birdy:~/ticket> prod tick_formula2.init
birdy:~/ticket> ./tick_formula2 -.

Illegal infinite path found
Loop 11 [0> 11
For more information, start "probe ./tick_formula2"
and look at the set %1
birdy:~/ticket> probe tick_formula2
0#build volatile verbose %1
PATH
Node 11
  ids: <.1.> + <.2.>
  turn: <.1,1.>
  procs: <.1,CRIT.> + <.2,TEST.>
  tickets: <.1,1.> + <.2,2.>
  t_e_s_t_e_r: <.1.>
Arrow 0: transition testtick, precedence class 0
  pid = 2
  next = 1
  granted = 1
  tick = 2
```

```
Node 11
  ids: <.1.> + <.2.>
  turn: <.1,1.>
  procs: <.1,CRIT.> + <.2,TEST.>
  tickets: <.1,1.> + <.2,2.>
  t_e_s_t_e_r: <.1.>
------------------------------------------------
1 paths
------------------------------------------------
0#quit
birdy:~/ticket>
```

- If the process 1 wants access to the critical section, it will make it there:

$$\Box(procs(\langle 1, ENTER\rangle) \implies \Diamond procs(\langle 1, CRIT\rangle))$$

```
#verify henceforth((procs >= <.1,ENTER.>) implies
                   (eventually (procs >= <.1,CRIT.>)));
```

The property does not hold in the model. PROD gives a counter example where the process 2 is continuously firing the transition `testtick` and the process 1 cannot make any progress.

```
birdy:~/ticket> ./tick_formula3 -.

Illegal infinite path found
Loop 11 [0> 11
For more information, start "probe ./tick_formula3"
and look at the set %1
birdy:~/ticket> probe tick_formula3
0#build volatile verbose %1
PATH
Node 11
  ids: <.1.> + <.2.>
  turn: <.1,1.>
  procs: <.1,LEAVE_T.> + <.2,TEST.>
  tickets: <.1,1.> + <.2,2.>
  t_e_s_t_e_r: <.3.>
Arrow 0: transition testtick, precedence class 0
  pid = 2
  next = 1
  granted = 1
  tick = 2
Node 11
  ids: <.1.> + <.2.>
  turn: <.1,1.>
  procs: <.1,LEAVE_T.> + <.2,TEST.>
  tickets: <.1,1.> + <.2,2.>
  t_e_s_t_e_r: <.3.>
------------------------------------------------
1 paths
------------------------------------------------
0#
```

- If the process 2 is in the critical section, it will eventually leave the critical section:

$$\neg\Diamond\Box(procs(\langle 2, CRIT\rangle))$$

```
#verify not eventually(henceforth(procs >= <.2,CRIT.>));
```

The property does not hold. PROD gives a counter example where the process 2 is in the critical section and the process 1 is continuously firing the transition `testick`.

```
birdy:~/ticket> ./tick_formula4

Illegal infinite path found
Loop 33 [0> 33
For more information, start "probe ./tick_formula4"
and look at the set %1
birdy:~/ticket> probe tick_formula4
0#build volatile verbose %1
PATH
Node 33
  ids: <.1.> + <.2.>
  turn: <.2,2.>
  procs: <.1,TEST.> + <.2,CRIT.>
  tickets: <.1,1.> + <.2,2.>
  t_e_s_t_e_r: <.4.>
Arrow 0: transition testtick, precedence class 0
  pid = 1
  next = 2
  granted = 2
  tick = 1
Node 33
  ids: <.1.> + <.2.>
  turn: <.2,2.>
  procs: <.1,TEST.> + <.2,CRIT.>
  tickets: <.1,1.> + <.2,2.>
  t_e_s_t_e_r: <.4.>
------------------------------------------------
1 paths
------------------------------------------------
0#
```

(b) Let us add a schedulet to the PROD description file `ticket.net`, listed below:

```
#ifndef nprocs
#define nprocs 2
#endif nprocs

#define NIL 0

#enum REM, ENTER, TEST, LEAVE_T, CRIT, RESET, LEAVE_E

#define ADDMOD(x) (1 + ((x) % (nprocs)))


#place turn lo(<.1,1.>) hi(<.nprocs,nprocs.>) mk(<.1,1.>)

#place procs lo(<.1,LEAVE_E.>) hi(<.nprocs,REM.>) \
mk(<.1..nprocs,REM.>)

#place tickets lo(<.1,NIL.>) hi(<.nprocs,nprocs.>) \
mk(<.1..nprocs,NIL.>)

#place ids lo(<.1.>) hi(<.nprocs.>) mk(<.1..nprocs.>)

#place scheduler lo(<.NIL.>) hi(<.nprocs.>) mk(<.NIL.>)


#trans schedule
   in  { ids: <.pid.>; scheduler: <.NIL.>; }
   out { ids: <.pid.>; scheduler: <.pid.>; }
#endtr

#trans try
   in  { procs: <.pid,REM.>; scheduler: <.pid.>; }
   out { procs: <.pid,ENTER.>; scheduler: <.NIL.>; }
#endtr

#trans enter
   in  { procs: <.pid,ENTER.>; turn: <.next,granted.>;
         tickets: <.pid,tick.>; scheduler: <.pid.>; }
   out { procs: (next == granted)<.pid,LEAVE_T.> +
                (next != granted)<.pid,TEST.>;
```

```
        turn: <.ADDMOD(next),granted.>;
        tickets: <.pid,next.>; scheduler: <.NIL.>; }
#endtr

#trans testtick
   in  { procs: <.pid,TEST.>; turn: <.next,granted.>;
        tickets: <.pid,tick.>; scheduler: <.pid.>; }
   out { procs: (tick == granted)<.pid,LEAVE_T.> +
               (tick != granted)<.pid,TEST.>;
        turn: <.next,granted.>;
        tickets: <.pid,tick.>; scheduler: <.NIL.>; }
#endtr

#trans crit
   in  { procs: <.pid,LEAVE_T.>; scheduler: <.pid.>; }
   out { procs: <.pid,CRIT.>; scheduler: <.NIL.>; }
#endtr

#trans exit
   in  { procs: <.pid,CRIT.>; scheduler: <.pid.>; }
   out { procs: <.pid,RESET.>; scheduler: <.NIL.>; }
#endtr

#trans reset
   in  { procs: <.pid,RESET.>; turn: <.next,granted.>;
        tickets: <.pid,tick.>; scheduler: <.pid.>; }
   out { procs: <.pid,LEAVE_E.>; turn: <.next,ADDMOD(granted).>;
        tickets: <.pid,NIL.>; scheduler: <.NIL.>; }
#endtr

#trans rem
   in  { procs: <.pid,LEAVE_E.>; scheduler: <.pid.>; }
   out { procs: <.pid,REM.>; scheduler: <.NIL.>; }
#endtr
```

The fairness assumption has to be included separately in each verifiable formula. The assumption is as follows:

$$(\Box\Diamond scheduler(\langle 1\rangle) \wedge \Box\Diamond scheduler(\langle 2\rangle))$$

which results in the following PROD verify formula:

```
((henceforth (eventually (scheduler >= <.1.>))) and
 (henceforth (eventually (scheduler >= <.2.>))))
```

Now we will be able to do the model checking of our formulas w.r.t. the fairness assumption:

- If the process 1 wants access to the critical section , it will eventually make it there:

$$(\Box\Diamond scheduler(\langle 1\rangle) \wedge \Box\Diamond scheduler(\langle 2\rangle)) \implies$$
$$(\Box(procs(\langle 1, ENTER\rangle) \implies \Diamond procs(\langle 1, CRIT\rangle)))$$

```
#verify (((henceforth (eventually (scheduler >= <.1.>))) and
          (henceforth (eventually (scheduler >= <.2.>))))
        implies
        (henceforth((procs >= <.1,ENTER.>)
                    implies
                    (eventually (procs >= <.1,CRIT.>)))));
```

  The property holds with the fairness assumption:

```
birdy:~/ticket> cat formula5
#verify (((henceforth (eventually (scheduler >= <.1.>))) and
          (henceforth (eventually (scheduler >= <.2.>))))
        implies
        (henceforth((procs >= <.1,ENTER.>)
                    implies
                    (eventually (procs >= <.1,CRIT.>)))));
birdy:~/ticket> cat ticket.net formula5 > ticket_formula5.net
birdy:~/ticket> prod ticket_formula5.init
birdy:~/ticket> ./ticket_formula5 -.
........................
........................
```

```
........................
........................
........................
........................
........................
....
birdy:~/ticket>
```

(c) The formula is valid in the model. A PROD example run is given below:

```
birdy:~/ticket> cat formula9
#verify (((eventually (henceforth (!(scheduler >= <.1.>)))) and
          (eventually (henceforth (!(scheduler >= <.2.>)))))
         implies
         (eventually(procs >= (<.1,CRIT.> + <.2,CRIT.>))));
birdy:~/ticket> cat ticket.net formula9 > ticket_formula9.net
birdy:~/ticket> prod ticket_formula9.init
birdy:~/ticket> ./ticket_formula9 -.
........................
........................
........................
........................
........................
........................
........................
........................
..........
birdy:~/ticket>
```

Also the following formula holds in the model:

$$(\Diamond\Box\neg scheduler(\langle 1\rangle) \wedge \Diamond\Box\neg scheduler(\langle 2\rangle)) \implies (1=0)$$

The formula is an example of a fairness assumption which does not hold for any execution. It claims that fair runs are the ones in which the scheduling of both the process 1 and 2 is terminated. Such runs do not exist in the model!