Tik-79.179 **Kevät 2001**
Rinnakkaiset ja hajautetut digitaaliset järjestelmät
Laskuharjoituksen 5 vastaukset
2.3.2000

1. Muodostetaan verkon insidenssi-matriisi (transitioiden muutokset paikkojen merkintöihin pystyvektoreita):

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

Jokin pystyvektori ι on invariantti, joss $C^T \cdot \iota = 0$, jossa 0 on pystyvektori.

a) Tehtävässä tulee näyttää invarianttien avulla: $M(p_4) \leq 2$.
Arvataan yhden invariantin olevan $\iota = M(p_2) + M(p_4)$. Tästä saadaan ι-vektori, jossa molempien paikkojen kerroin on 1. Tarkistetaan että kyseinen vektori todellakin on invariantti:

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

Siis $\iota = M(p_2) + M(p_4)$ on invariantti. Nyt lasketaan alkumerkinnästä M_0, mikä on invariantin lukuarvo kaikissa saavutettavissa merkinnöissä.

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

Siis $M(p_2) + M(p_4) = 2 \Rightarrow M(p_4) \leq 2$. Täten korkeintaan 2 prosessoria voi samanaikaisesti käyttää muistia.

b) Tehtävässä osoitetaan ensiksi $\iota = M(p_1) + M(p_3) + M(p_4) + M(p_5)$ olevan invariantti. Tarkistetaan se:

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

Sitten lasketaan invariantin lukuarvo alkumerkinnästä:

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

Näin on osoitettu invariantiksi: $M(p_1) + M(p_3) + M(p_4) + M(p_5) = 5$.

2. TicketME-algoritmin `PROD`-kuvaus alla:

```
#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) Olkoon yllä olevan `PROD`-kuvaus tallennettuna tiedostoon `tick.net`. Esitetään: ominaisuus LTL-kaavana, sama `PROD` verify kaavana, päteekö ominaisuus, esimerkki mallintarkastus ajo.

- Kaksi prosessia ei koskaan ole samaan aikaan kriittisessä sektiossa:

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

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

  Ominaisuus pätee.

  ```
  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>
  ```

- Prosessi 1 ei koskaan pääse kriitiseen sektioon:

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

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

  Ominaisuus ei päde.

  ```
  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>
```

- Jos prosessi 1 haluaa päästä kriittiseen sektioon, se myös lopulta sinne pääsee:

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

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

Ominaisuus ei päde mallissa. `PROD` antaa vasta-esimerkin, jossa prosessi 2 jatkuvasti suorittaa transitiota `testtick`, ja prosessi 1 ei pääse koskaan etenemään:

```
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#
```

- Jos prosessi 2 on kriitisessä sektiossa, se myös lopulta poistuu kriittisestä sektiosta:

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

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

Ominaisuus ei päde. `PROD` antaa vasta-esimerkin, jossa prosessi 2 on kriittisessä sektiossa, ja prosessi 1 suorittaa jatkuvasti `testick` transitiota:

```
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) Lisätään PROD malliin skeduleri tiedostoon `ticket.net`. Täydellinen PROD-kuvaus alla:

```
#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
```

Reiluus-oletus joudutaan sisällyttämään jokaiseen verifioitavaan kaavaan erikseen. Oletus on siis seuraava:

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

joka on `PROD` verify-kaavana:

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

Nyt halutut kaavat saadaan mallintarkistettua reiluus-oletus huomioon ottaen:

- Jos prosessi 1 haluaa päästä kriittiseen sektioon, se myös lopulta sinne pääsee:

$$\begin{aligned}&(\Box\Diamond scheduler(\langle 1\rangle) \wedge \Box\Diamond scheduler(\langle 2\rangle)) \implies \\ &(\Box(procs(\langle 1, ENTER\rangle) \implies \Diamond procs(\langle 1, CRIT\rangle)))\end{aligned}$$

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

  Ominaisuus pätee reiluusoletuksilla:

```
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) Kysytty kaava pätee mallissa. `PROD` esimerkki-ajo alla:

```
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>
```

Myös seuraava kaava pätee mallissa:

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

Kaava on esimerkki reiluus-oletuksesta, joka ei päde järjestelmän yhdellekään suoritukselle. Se sanoo että reiluja ajoja ovat sellaiset ajot, joissa lopulta sekä prosessin 1 että 2 skedulointi lopetetaan. Tällaisia ajoja ei mallissa ole!