T-79.179 **Kevät 2002**
Rinnakkaiset ja hajautetut digitaaliset järjestelmät
Laskuharjoituksen 5 vastaukset
1.3.2002

1. TicketME-algoritmin Maria-kuvaus alla:

```
typedef unsigned(1..2) n_of_processes;
typedef enum {REM, ENTER, TEST, LEAVE_T, CRIT, RESET, LEAVE_E} process_state;
typedef struct {
  n_of_processes x,
  process_state y
} process_t;
typedef struct {
  n_of_processes x,
  n_of_processes y
} turn_t;
typedef struct {
  n_of_processes x,
  unsigned(0..#n_of_processes) y
} ticket_t;

place turn turn_t : {1,1};
place procs process_t : n_of_processes x : {x,REM};
place tickets ticket_t : n_of_processes x : {x,NIL};

trans try
   in  { place procs: {pid,REM}; }
   out { place procs: {pid,ENTER}; };

trans enter
   in  { place procs: {pid,ENTER}; place turn: {next,granted};
         place tickets: {pid,tick}; }
   out { place procs: (next == granted)?{pid,LEAVE_T}:{pid,TEST};
         place turn: {+next,granted};
         place tickets: {pid,next}; };

trans testtick
   in  { place procs: {pid,TEST}; place turn: {next,granted};
         place tickets: {pid,tick}; }
   out { place procs:
         (is n_of_processes tick == granted)?{pid,LEAVE_T}:{pid,TEST};
         place turn: {next,granted};
         place tickets: {pid,tick}; };

trans crit
   in  { place procs: {pid,LEAVE_T}; }
   out { place procs: {pid,CRIT}; };

trans exit
   in  { place procs: {pid,CRIT}; }
   out { place procs: {pid,RESET}; };

trans reset
   in  { place procs: {pid,RESET}; place turn: {next,granted};
         place tickets: {pid,tick}; }
   out { place procs: {pid,LEAVE_E}; place turn: {next,+granted};
         place tickets: {pid,NIL}; };

trans rem
```

```
in  { place procs: {pid,LEAVE_E}; }
out { place procs: {pid,REM}; };
```

(a) Olkoon yllä olevan Maria-kuvaus tallennettuna tiedostoon `tick.pn`. Esitetään ominaisuus LTL-kaavana, ominaisuus Marian LTL tai `reject`-kaavana, päteekö ominaisuus ja ajoesimerkki mallintarkastuksesta.

- Kaksi prosessia ei koskaan ole samaan aikaan kriittisessä lohkossa:

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

`[]!(is process_t ({1,CRIT},{2,CRIT}) subset place procs)`
tai, koska jonkin tilan saavuttamattomuus voidaan varmentaa myös `reject`-kaavalla
`reject is process_t ({1,CRIT},{2,CRIT}) subset place procs);`
Ominaisuus pätee. Tässä on käytetty `reject`-kaavaa, koska se on laskennallisesti huomattavasti kevyempi kuin LTL mallintarkastus.

- Prosessi 1 ei koskaan pääse kriitiseen lohkoon:

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

`[]!(is process_t {1,CRIT} subset place procs)`
tai `reject`-kaavana
`reject (is process_t {1,CRIT} subset place procs);`
Ominaisuus ei päde. Vastaesimerkki:

```
path 1 /lhome/jhonkola $./maria -b TicketME.pn
rejected state @8
rejected state @11
rejected state @16
rejected state @38
rejected state @47
rejected state @54
rejected state @61
rejected state @71
"TicketME.pn": 82 states, 164 arcs
@0$path @8
shortest path from @0 to @8 (4 nodes):
 @0 @1 @4 @8
@0$show @8
state (
 ids:
  1,2
 turn:
  {2,1}
 procs:
  {2,REM},{1,CRIT}
 tickets:
  {2,0},{1,1}
)
2 predecessors
2 successors
@0$exit
path 2 /lhome/jhonkola $
```

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

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

```
[]((is process_t {1,ENTER} subset place procs) => \
<> (is process_t {1,CRIT} subset place procs))
```

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

```
path 2 /lhome/jhonkola/ticket $./prop_3
./prop_3:5:constructing counterexample
./prop_3:5:counterexample path:
./prop_3:5: @0 @1 @3 @27 @56
@0$show @27
state (
 ids:
  1,2
 turn:
  {2,1}
 procs:
  {2,ENTER},{1,LEAVE_T}
 tickets:
  {2,0},{1,1}
)
3 predecessors
2 successors
@0$show @3
state (
 ids:
  1,2
 turn:
  {1,1}
 procs:
  {1,ENTER},{2,ENTER}
 tickets:
  {1,0},{2,0}
)
2 predecessors
2 successors
@0$show @56
state (
 ids:
  1,2
 turn:
  {1,1}
 procs:
  {2,TEST},{1,LEAVE_T}
 tickets:
  {1,1},{2,2}
)
3 predecessors
2 successors
@0$succ @56
transition testtick->@56
<
 pid:2
 next:1
 granted:1
 tick:2
>
```

```
transition crit->@57
<
 pid:1
>
@0$
```

- Jos prosessi 2 on kriitisessä lohkossa, se myös lopulta poistuu sieltä:

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

```
!<>[](is process_t {2,CRIT} subset place procs)
```

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

```
@0$!<>[](is process_t {2,CRIT} subset place procs)
(command line):2:constructing counterexample
(command line):2:counterexample path:
(command line):2: @0 @1 @3 @27 @56 @57 @44 @45 @34 @35 @18 @19 @50
@0$show @50
state (
 ids:
  1,2
 turn:
  {2,2}
 procs:
  {1,TEST},{2,CRIT}
 tickets:
  {1,1},{2,2}
)
3 predecessors
2 successors
@0$succ @50
transition testtick->@50
<
 pid:1
 next:2
 granted:2
 tick:1
>
transition exit->@51
<
 pid:2
>
@0$exit
path 3 /lhome/jhonkola/ticket $
```

(b) Jaetaan prosessien transition kahteen reiluusjoukkoon, toiseen joukkoon prosessin 1 siirtymät ja toiseen prosessin 2 siirtymät. Kaikkia *reiluusjoukkoja* kohdellaan reilusti. Käytetään vahvaa reiluutta, eli jos molemmissa reiluusjoukoissa on äärettömän usein vireessä olevia siirtymiä, niin molemmista *reiluusjoukoista* laukaistaan siirtymä äärettömän usein. Saman reiluusjoukon siirtymiä ei välttämättä kuitenkaan laukaista toisiinsa nähden reilusti.

```
strongly_fair trans try : pid == 1, trans enter : pid == 1,\
trans testtick : pid == 1, trans crit : pid == 1, trans exit : pid == 1,\
trans reset : pid == 1, trans rem : pid == 1;

strongly_fair trans try : pid == 2, trans enter : pid == 2,\
trans testtick : pid == 2, trans crit : pid == 2, trans exit : pid == 2,\
trans reset : pid == 2, trans rem : pid == 2;
```

Jos käytettävä työkalu ei mahdollista reiluusoletusten sisällyttämistä suoraan malliin, joudutaan reiluusominaisuudet mallittamaan LTL-kaavana joka lisätään jokaiseen verifioitavaan kaavaan erikseen. Olkoon reiluusoletus ϕ ja verifioitava kaava ψ. Tällöin reiluusoletuksen kanssa verifioitava kaava on muotoa $\phi \implies \psi$. Reiluusoletuksen lisääminen vaatii yleensä myös lisäyksiä malliin.

Tässä tapauksessa malliin joudutaan lisäämään skeduleri, joka määrää kumpi prosessi voi suorittaa siirtymiä. Reiluusoletuksena käytetään kaavaa

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

joka on Marian LTL kaavana:

```
[]<>(1 subset place scheduler) && []<> (2 subset place scheduler)
```

Kaava määrittää reiluiksi suorituksiksi sellaiset suoritukset joissa molemmat prosessit pääsevät suorittamaan siirtymiä äärettömän monta kertaa.

Nyt haluttu kaava saadaan mallintarkistettua reiluusoletus huomioon ottaen:

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

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

(c) Kysytty kaava pätee mallissa. Marian esimerkkiajo alla:

```
path 1 /lhome/jhonkola/ticket $cat prop_5
#!./maria
graph "TicketME";
translator "./lbt";
        ([](is process_t {1,ENTER} subset place procs) =>\
        (<>(is process_t {1,CRIT} subset place procs)));
path 2 /lhome/jhonkola/ticket $ls -la TicketME.rg?
-rw-rw-r--   1 jhonkola jhonkola     1124 Mar  1 07:30 TicketME.rga
-rw-rw-r--   1 jhonkola jhonkola     1032 Mar  1 07:31 TicketME.rgd
-rw-rw-r--   1 jhonkola jhonkola     2048 Mar  1 07:31 TicketME.rgh
-rw-rw-r--   1 jhonkola jhonkola     1312 Mar  1 07:31 TicketME.rgp
-rw-rw-r--   1 jhonkola jhonkola      492 Mar  1 07:31 TicketME.rgs
path 3 /lhome/jhonkola/ticket $./prop_5
./prop_5:5:property holds
@0$exit
path 4 /lhome/jhonkola/ticket $
```

Tässä esimerkkiajossa ei lisätty reiluusoletusta kaavaan, vaan käytettiin Marian sisäänrakennettuja reiluusominaisuuksia.

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!

Jos reiluusoletukset joudutaan lisäämään tarkastettavaan kaavaan, on siis oltava tarkkana, että ei käytetä sellaista reiluusoletusta jonka mukaan mikään ajo ei ole reilu.