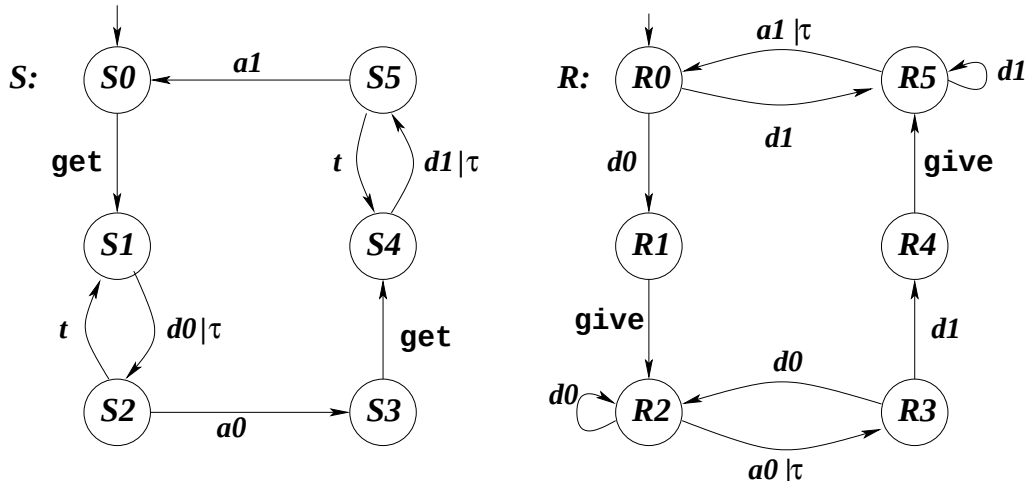
Spesifioinnin ja verifioinnin perusteet

Harjoitus 6, 22.2.2008

1. Spesifioi Lotoksella AB-protokollan toimintaa kuvaava systeemi $S|[d0, d1, a0, a1]|R$, kun S ja R ovat seuraavat siirtymäsysteemit:



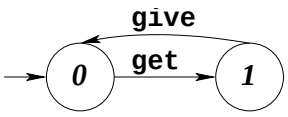
Alla on ratkaisu, jossa on jo valmiiksi piilotettu tapahtumat d0, d1, a0, a1, t seuraavaa tehtävää ajatellen.

```
specification AB [get, give, d0, d1, a0, a1, t] : noexit behaviour

hide d0, d1, a0, a1, t in

        S[get, d0, d1, a0, a1, t]
         |[d0, d1, a0, a1]|
        R[give, d0, d1, a0, a1]

where

process S [get, d0, d1, a0, a1, t] : noexit :=
        get;S1[get, d0, d1, a0, a1, t]
endproc

process S1 [get, d0, d1, a0, a1, t] : noexit :=
        d0;S2[get, d0, d1, a0, a1, t]   []
        i;S2[get, d0, d1, a0, a1, t]
endproc
```

```
process S2 [get, d0, d1, a0, a1, t] : noexit :=
        t;S1[get, d0, d1, a0, a1, t]   []
        a0;get;S4[get, d0, d1, a0, a1, t]
endproc

process S4 [get, d0, d1, a0, a1, t] : noexit :=
        d1;S5[get, d0, d1, a0, a1, t]   []
        i;S5[get, d0, d1, a0, a1, t]
endproc

process S5 [get, d0, d1, a0, a1, t] : noexit :=
        t;S4[get, d0, d1, a0, a1, t]   []
        a1;S[get, d0, d1, a0, a1, t]
endproc

process R [give, d0, d1, a0, a1] : noexit :=
        d0;give;R2[give, d0, d1, a0, a1] []
        d1;R5[give, d0, d1, a0, a1]
endproc

process R2 [give, d0, d1, a0, a1] : noexit :=
        d0;R2[give, d0, d1, a0, a1] []
        a0;R3[give, d0, d1, a0, a1] []
        i;R3[give, d0, d1, a0, a1]
endproc

process R3 [give, d0, d1, a0, a1] : noexit :=
        d0;R2[give, d0, d1, a0, a1] []
        d1;give;R5[give, d0, d1, a0, a1]
endproc

process R5 [give, d0, d1, a0, a1] : noexit :=
        d1;R5[give, d0, d1, a0, a1] []
        a1;R[give, d0, d1, a0, a1] []
        i;R[give, d0, d1, a0, a1]
endproc

endspec
```

2. Muodosta edellisen tehtävän yhteistilaverkko Aldebarania käyttäen. Toteuttaako tämä AB-protokollan versio halutun monisteessa esitetyn palvelun ympäristölleen?

   Piilotetaan ensin edellisen tehtävän siirtymäsysteemissä toiminnot d0, d1, a0, a1, t ja minimoidaan se sitten käyttäen heikkoa bisimulaatioekvivalenssia. Tulokseksi saadaan seuraava siirtymäsysteemi, joka on sama kuin haluttu palvelu.



3. Edellisissä harjoituksissa spesifioit Lotoksella Hymanin poissulkemisprotokolan kahden prosessin tapauksessa ja muodostit yhteistilaverkon CADP-ohjelmistoa käyttäen. Piilota yhteistilaverkossa kaikki tapahtumat lukuunottamatta kriittistä aluetta mallintavia. Minimoi nyt yhteistilaverkko heikon bisimulaation suhteen ja tutki tulosta. Ovatko prosessit koskaan yhtäaikaa kriittisellä alueella?

   Minimoidusta verkosta löytyy polku, jossa tapahtumat *incs*1 ja *incs*2 ovat peräkkäin eli molemmat prosessit pääsevät yhtäaikaa kriittiselle alueelle.

   (HUOM! Saat vääränlaisen verkon ja minimointi antaa väärän tuloksen, jos esimerkiksi olet käyttänyt prosessia 1 prosessin 2 määrittelemiseen ja olet parametrisoinut kutsun väärin.)

4. Ominaisuuksien testaus voidaan tehdä spesifioimalla haluttu ominaisuus Lotos-testiprosessina (joka päättyy testin onnistumisesta kertovaan toimintoon `testok`) ja sykronoimalla testiprosessi spesifikaation kanssa. Tällöin sykronointilistassa ovat kaikki testiprosessin toiminnot paitsi toiminto `testok`.

   Jos esimerkiksi prosessi S voi suorittaa joko tapahtumajonon `ev1;ev2;ev3;exit` tai tapahtumajonon `ev1;ev2;ev4;exit` ja haluttaisiin tietää voiko tapahtuma `ev4` seurata tapahtumaa `ev2`, muodostetaan testausta varten seuraavat prosessit ja käyttäytymislauseke:

   process T[ev2, ev4,testok] : **exit** :=
   ev2;ev4;testok;**exit**
   endproc

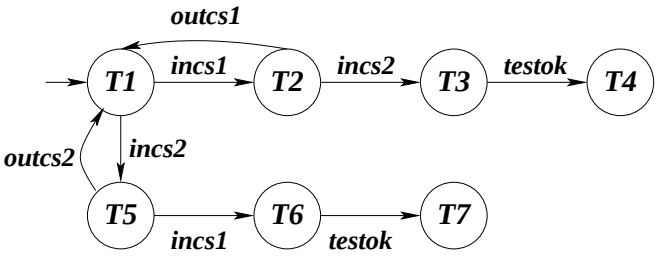
   process S[ev1,ev2,ev3,ev4] : **exit** :=
   ev1;ev2;ev3;**exit** [] ev1;ev2;ev4;**exit**
   endproc

   S[ev1,ev2,ev3,ev4] |[ev2, ev4]| T[ev2,ev4,testok]

   Prosessit T ja S siis sykronoidaan taphtumien `ev2` ja `ev4` avulla. Tämän jälkeen tutkitaan kaikki mahdolliset yhdistetyn prosessin jäljet ja jos testitapahtuma löydetään on testi mennyt läpi.

   Miten voit käyttää yllä olevaa menetelmää selvittääksesi voivatko prosessit olla yhtäaikaa kriittisellä alueella? Kokeile Hymanin algoritmia ja monisteessa esitettyä Dekkerin algoritmia (sivu 97-99).

   Tehdään seuraavanlainen testiprosessi, joka huomaa tilanteen, jossa prosessit ovat yhtäaikaa kriittisellä alueella.



Testiprosessia voidaan käyttää seuraavaalla tavalla testaamaan Hymanin poissulkemisalgoritmia:

```
specification Hyman[incs1, outcs1, incs2, outcs2, testok] : noexit behaviour
hide in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,kr1,kr2,kw1,kw2
in
((P1[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
    kr1,kw1,kr2,kw2]
|||
P2[incs2,outcs2,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2])
|[in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,kr1,kr2,kw1,kw2]|
(in1f[in1rt,in1rf,in1wt,in1wf]
|||
in2f[in2rt,in2rf,in2wt,in2wf]
|||
K1[kr1,kr2,kw1,kw2]))
|[incs1,outcs1,incs2,outcs2]|
T[incs1,outcs1,incs2,outcs2,testok]

where

process T[incs1,outcs1,incs2,outcs2,testok]: noexit :=
  incs1; (outcs1; T [incs1,outcs1,incs2,outcs2,testok]
           [] incs2;testok;stop)
  []
  incs2; (outcs2; T[incs1,outcs1,incs2,outcs2,testok]
           [] incs1;testok;stop)
endproc

process P1[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] : noexit :=
  in1wt;P11[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2]
endproc

process P11[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] : noexit :=
  kr1;P13[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] []
  kr2;P12[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2]
endproc
```

```
process P12[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] : noexit :=
  in2rf;kw1;P11[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] []
  in2rt;i;P12[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2]
endproc

process P13[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] : noexit :=
  incs1;outcs1;in1wf;P1[incs1,outcs1,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2]
endproc

process P2[incs2,outcs2,in1rt,in1rf,in1wt,in1wf,in2rt,in2rf,in2wt,in2wf,
           kr1,kw1,kr2,kw2] : noexit :=
  P1[incs2,outcs2,in2rt,in2rf,in2wt,in2wf,in1rt,in1rf,in1wt,in1wf,
           kr2,kw2,kr1,kw1]
endproc

process in1t[in1rt,in1rf,in1wt,in1wf] : noexit :=
               in1rt;in1t[in1rt,in1rf,in1wt,in1wf] []
               in1wt;in1t[in1rt,in1rf,in1wt,in1wf] []
               in1wf;in1f[in1rt,in1rf,in1wt,in1wf]
endproc
process in1f[in1rt,in1rf,in1wt,in1wf] : noexit :=
               in1rf;in1f[in1rt,in1rf,in1wt,in1wf] []
               in1wf;in1f[in1rt,in1rf,in1wt,in1wf] []
               in1wt;in1t[in1rt,in1rf,in1wt,in1wf]
endproc


process in2f[in2rt,in2rf,in2wt,in2wf] : noexit := in1f[in2rt,in2rf,in2wt,in2wf]
endproc

process K1[kr1,kr2,kw1,kw2] : noexit := in1t[kr1,kr2,kw1,kw2]
endproc

endspec (* Hyman *)
```

Kun generoidaan yhteistilaverkko ja tarkastellaan sitä, niin huomataan, että mukaan

on tullut tapahtuma `testok`. Tällöin siis testi meni läpi ja prosessit voivat olla yhtäaikaa kriittisellä alueella.

Testiprosessia voidaan käyttää seuraavaalla tavalla testaamaan monisteessa esitettyä Dekkerin algoritmia (sivu 97-99):

```
specification dekkertest[enter1, exit1, enter2, exit2,testok]: noexit

library
   BOOLEAN,
   NATURAL
endlib

type COMMAND is
   sorts COMMAND
   opns
      READ  (*! constructor *),
      WRITE (*! constructor *) : -> COMMAND
endtype

behavior

    hide NCS0, NCS1, F0, F1, T in
    ( (
      P [NCS0, enter1, exit1, F0, F1, T] (0)
      |||
      P [NCS1, enter2, exit2, F1, F0, T] (1)
      )
   |[F0, F1, T]|
      (
      FLAG [F0] (false)
      |||
      FLAG [F1] (false)
      |||
      TURN [T] (0)
      ) )
|[enter1,exit1,enter2,exit2]|

T[enter1,exit1,enter2,exit2,testok]

where

process T[enter1,exit1,enter2,exit2,testok]: noexit :=
  enter1; (exit1; T [enter1,exit1,enter2,exit2,testok]
           [] enter2;testok;stop)
  []
```

```
  enter2; (exit2; T[enter1,exit1,enter2,exit2,testok]
           [] enter1;testok;stop)
endproc


process P [NCS, entern, exitn, FJ, FI, T] (J : Nat) : noexit :=
      NCS; (* non critical section *)
         FJ !WRITE !true;
            P_AUX_1 [NCS, entern, exitn, FJ, FI, T] (J)
   endproc

   process P_AUX_1 [NCS, entern, exitn, FJ, FI, T] (J : Nat) : noexit :=
      FI !READ ?VAL_FI:Bool;
         (
         [VAL_FI] ->
            T !READ ?VAL_T:Nat;
               (
               [VAL_T <> J] ->
                  FJ !WRITE !false;
                     P_AUX_2 [NCS, entern, exitn, FJ, FI, T] (J)
               []
               [VAL_T == J] ->
                  P_AUX_1 [NCS, entern, exitn, FJ, FI, T] (J)
               )
         []
         [not (VAL_FI)] ->
            entern; (* critical section *)
              exitn;
               T !WRITE !(J + 1) mod 2;
                  FJ !WRITE !false;
                     P [NCS, entern, exitn, FJ, FI, T] (J)
         )
   endproc

   process P_AUX_2 [NCS, entern, exitn, FJ, FI, T] (J : Nat) : noexit :=
      T !READ ?VAL_T:Nat;
         (
         [VAL_T <> J] ->
            P_AUX_2 [NCS, entern, exitn, FJ, FI, T] (J)
         []
         [VAL_T == J] ->
            FJ !WRITE !true;
               P_AUX_1 [NCS, entern, exitn, FJ, FI, T] (J)
```

```
      )
   endproc

   process FLAG [F] (VAL_FLAG : BOOL) : noexit :=
      F !READ !VAL_FLAG;
         FLAG [F] (VAL_FLAG)
      []
      F !WRITE ?VAL:BOOL;
         FLAG [F] (VAL)
   endproc

   process TURN [T] (VAL_TURN : NAT) : noexit :=
      T !READ !VAL_TURN;
         TURN [T] (VAL_TURN)
      []
      T !WRITE ?VAL:NAT;
         TURN [T] (VAL)
   endproc

endspec
```

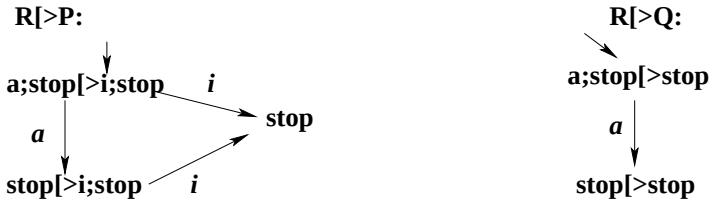
Kun generoidaan yhteistilaverkko ja tarkastellaan sitä, niin huomataan, että mukana ei ole tapahtumaa `testok`. Tällöin siis testi ei mennyt läpi ja prosessit eivät voi olla yhtäaikaa kriittisellä alueella.

HUOM! Kummassakaan tapauksessa ei tarvitse piilottaa prosessin muita toimintoja (siis niitä, jotka eivät ole mukana testiprosessissa). Jos toimintoja ei piilota, niin `testok` tapahtuman löytyessä verkosta voi nähdä suoritusjäljen, joka johtaa testitapahtumaan.

5. Olkoon $P \approx_{wbis} Q$. Tutki päteekö kaikilla prosesseilla R $R[> P \approx_{wbis} R[> Q$. (HUOM! Jos väite ei päde, niin riittää antaa vastaesimerkki.)

   Väite ei päde, koska voidaan antaa seuraava vastaesimerkki. Olkoot prosessit prosessien P, Q ja R toiminta määritelty seuraavasti:
   P: i;stop
   Q: stop
   R: a;stop

   Nyt selvästi $P \approx_{wbis} Q$, mutta $R[> P \approx_{wbis} R[> Q$.



   Siirtymäsysteemit eivät ole heikosti bisimilaarisia. Jos nimittäin nyt yritetään muodostaa heikko bisimulaatiorelaatio $\mathcal{S}$, niin parin, jossa on prosessien alkutilat täytyy kuulua relaatioon. Käytetään seuraavaksi määritelmän ehtoa 2: vasemmanpuoleinen prosessi tekee heikon ε siirtymän alkutilasta tilaan stop. Ainoa tapa simuloida tätä oikeanpuoleisen prosessin osalta on tehdä heikko ε-siirtymä takaisin alkutilaan. Siis parin, johon kuuluvat edellisten ε-siirtymien päätetilat tulee kuulua myös relaatioon $\mathcal{S}$. Käytetään tähän pariin määritelmän ehtoa 3: oikeanpuoleinen prosessi tekee a-siirtymän, mutta vasemmanpuoleinen prosessi ei voi simuloida tätä siirtymää. Siis heikkoa bisimulaatiorelaatiota ei voi olla olemassa prosessien $R[> P$ ja $R[> Q$ välillä, joten prosessit eivät ole heikosti bisimilaariset.

6. (a) Prosessin kutsu ei ole tarkalleen sama, kuin prosessin vartalo, jossa muodolliset parametrit on korvattu kutsuparametreilla. Tarkastele alla olevaa prosessia ja kerro miten voit havainnollistaa asiaa prosessilla.

```
process koe [a,b] :=
  apu[a,a]
 []
  apu[a,b]
where
  process apu[x,y] :=
   x; exit || y; exit
  endproc
endproc
```

Kummastakaan `apu[a,a]`:sta eikä `apu[a,b]`:stä ei ole siirtymiä eteenpäin vaikkakin käyttäytymislausekkeesta `a; exit || a; exit`, jossa parametrit on suoraan korvattu kutsuparametreilla on a-siirtymä.

Jotta `apu[a,a]` $\xrightarrow{a} Q$ täytyisi olla `(x; exit || y; exit)[a/x, a/y]` $\xrightarrow{a} Q$.
Jotta `(x; exit || y; exit)[a/x, a/y]` $\xrightarrow{a} Q$ täytyisi olla `(x; exit || y; exit)` siirtymä, mutta koska synkronointijoukossa on kaikki tapahtumat, niin siirtymiä ei ole ollenkaan. Täten ei ole myöskään siirtymiä `apu[a,a]`:sta.

(HUOM! Caesar/Aldebaran-ohjelmisto toimii tässä kohtaa väärin.)

(b) Tarkastellaan seuraavia täyden Lotoksen prosesseja:

```
P[a,b,c] (n:Nat): noexit :=
 a!n; b?m:Nat !3; c?n:Nat; P[a,b,c](n)

Q[a,b,c]: noexit :=
 a?x:Nat; b!x !3; c!x+1; Q[a,b,c]
  []
 a!1; i; Q[a,b,c]
```

Muodosta yhteistilaverkkoa

```
P[a,b,c](0) || Q[a,b,c]
```

sen verran, että kokonaisrakenne tulee selville.

Alla on generoitu yhteistilaverkkoa sen verran, että kokonaisrakenne tulee selville: