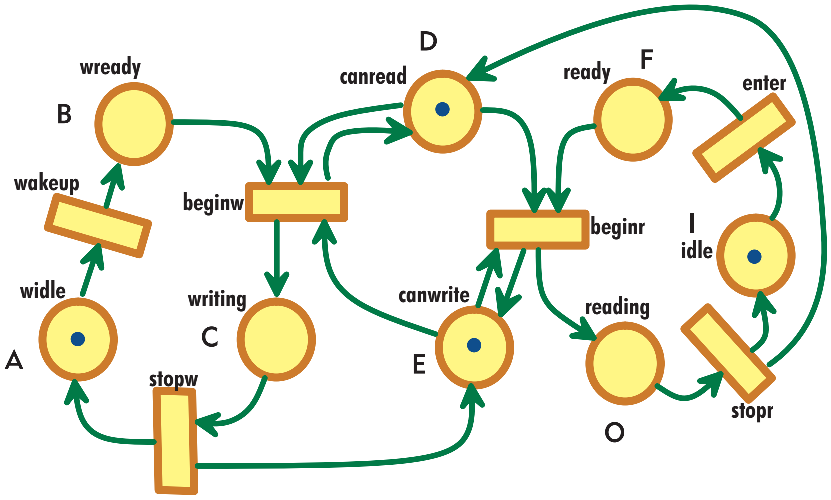
Laatinut: Esa Kettunen

Petri-verkon muuntaminen ohjelmaksi

Ehkä olet jo havainnut, että Petri-verkot ovat aivan omalla abstraktiotasollaan. Petri-verkkospesifikaation *toteuttaminen* onkin oma taiteen lajinsa. Paljon riippuu siitä, mikä on kohdejärjestelmän rinnakkaisuusmalli. Synkroninen kommunikointi on tietyssä mielessä lähempänä Petri-verkkoja, kun taas epäsynkroninen kommunikointi voi aiheuttaa lisätyötä.

Seuraavassa hahmotellaan **tilaa testaavan mutex-algoritmin** toteutus käyttäen sanomanvaihtoon operaatioita `send` ja `receive`, joiden kaltaiset löytyvät mm. TCP/IP:n socket-pohjaisesta ohjelmointirajapinnasta.



Teemme eron seuraavien Petri-verkon primitiivi-ilmiöiden välillä:

- *testataan* onko token paikassa,
- *otetaan* token paikasta tai
- *pannaan* token paikkaan.

Järjestelmässä on kaksi prosessityyppiä, `writer` ja `reader` eikä kummastakaan voi olla enempää kuin yksi instanssi käynnissä kunakin ajan hetkenä. (On luonnollisesti myös muunlaisia ratkaisuja readers/writers-ongelmaan: joissakin sallitaan monta instanssia.)

Kirjoittajaprosessilla on

- 3 prosessin "omaa" paikkaa: `widle`, `wready` ja `writing`. Ne mallitetaan kokonaislukumuuttujina.
- Kaksi paikkaa jaettuna lukijaprosessin kanssa: `canread` ja `canwrite`. Ne mallitetaan kokonaislukumuuttujina, mutta tokenin poistaminen/paneminen paikkaan edellyttää kommunikointia lukijaprosessin kanssa.

Lukijaprosessilla on

- 3 kolme "omaa" paikkaa: `idle`, `ready` ja `reading`. Ne mallitetaan kokonaislukumuuttujina.
- Kaksi paikkaa jaettuna lukijaprosessin kanssa: `canread` ja `canwrite`. Ne mallitetaan kokonaislukumuuttujina, mutta tokenin poistaminen/paneminen paikkaan edellyttää kommunikointia kirjoittajaprosessin kanssa.

Osoittautuu, että `writer` pelkästään testaa tokenin olemassaoloa paikassa `canread`. `Reader` puolestaan testaa tokenin olemassaoloa `canwrite`-paikassa. Operaatiot `send` ja `receive` perustuvat epäsynkroniseen kommunikointiin, mutta luonnollisesti koko järjestelmän saa toimimaan myös synkronoinneilla (jolloin kuittaukset ovat tarpeettomia).

```
process writer
      var widle: integer := 1;
          wready: integer := 0;
          writing: integer := 0;
          canread: integer := 1;
          canwrite: integer := 1;
          enter_crit: boolean := FALSE;

   procedure check_tokens (canwrite_op: canwrite_id)
      var place: place_id;
          oper:  oper_id;     // recv operation

      if canwrite_op = Q_CANWRITE_GET
         send (Q_CANWRITE, Q_GET)
      elseif canwrite_op = Q_CANWRITE_PUT
         send (Q_CANWRITE, Q_PUT)
      endif
      while recv (place, oper) or (not canwrite_op = Q_CANWRITE_NONE)
         if place = Q_CANREAD and oper = Q_GET
```

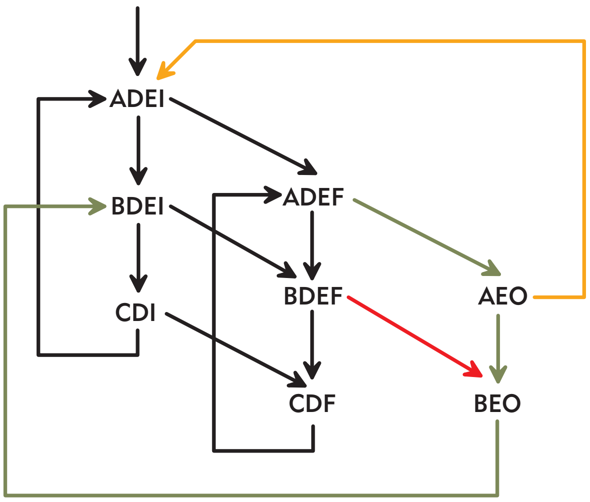
```
         canread := canread - 1
         send (Q_CANREAD, Q_ACK)
         log (widle, wready, writing, canread, canwrite)
      else if place = Q_CANREAD and oper = Q_PUT
         canread := canread + 1
         send (Q_CANREAD, Q_ACK)
         log (widle, wready, writing, canread, canwrite)
      else if place = Q_CANWRITE and oper = Q_ACK
         canwrite_op := Q_CANWRITE_NONE
      endif
   endwhile
endprocedure

begin // writer process
...
while true
   log (widle, wready, writing, canread, canwrite)
   enter_crit := calculate_a_little ()
   check_tokens (CANWRITE_NONE)
   if widle and enter_crit
      widle := 0
      wready := 1
      log (widle, wready, writing, canread, canwrite)
   endif
   if wready and canread and canwrite
      check_tokens (CANWRITE_GET)
      if canread
         canwrite := 0
         wready := 0
         writing := 1
         log (widle, wready, writing, canread, canwrite)
         //
         // now in critical section (writing)
         //
         critical_section (Q_WRITE);
         //
         // now exiting critical section (writing)
         //
         canwrite := 1
         writing := 0
         idle := 1
         enter_crit := FALSE
      else
         check_tokens (CANWRITE_PUT)
```

```
        endif
        log (widle, wready, writing, canread, canwrite)
      endif
    endwhile
endprocess
```

`Reader`-prosessi toteutetaan samaan tapaan. Kukin prosessi tuottaa lokin, jota verrataan Petri-verkon saavutettavuusgraafiin. Tarkkaavainen lukija on jo huomannut, että `writer`-prosessi tuottaa pelkkiä osittaismerkintöjä, koska se ”näkee” ainoastaan viisi paikoista. Tämän vuoksi tyydymme siihen, että lokin ja saavutettavuusgraafin vertailussa löytyy sekvenssi osittaismerkintöjen kanssa yhteensopivia solmuja. Muussa tapauksessa päättelemme, että toteutus on virheellinen.[1]

Jos ohjelmoijan tehtävänä on toteuttaa vain osa spesifikaatiosta. Tällöin vaihtoehtona voi olla testata ”näkyvät” transitiot (eikä siis merkintöjä) ja pitää muita transitioita näkymättöminä. Sen tarkistaminen löytyykö laukaistujen näkyvien transitioiden jälki saavutettavuusgraafista on suhteellisen yksinkertaista. Tämä lähestymistapa on ilmiselvässä yhteydessä prosessialgebrojen teorian kanssa.



Yo. saavutettavuusgraafissa kirjoittajaprosessin tulee generoida merkintäsekvenssit ADE*-BDE*-CDE*-ADE*, ADE*-BDE*-BE*-BDE*, ADE*-AE*-ADE* jne. eikä mitään muita sekvenssejä.

[1] `VAROITUS:` Patologisessa tapauksessa voi käydä niin, että huolimatta `reader`- ja `writer`-prosessin toteutusten oikeellisuudesta *erikseen*, kokonaisuus ei kuitenkaan toimi niinkuin Petri-verkkospesifikaatio määrää. Tämä edellyttää pidemmälle meneviä prosessialgebran ja testauksen teorian opintoja kuin mitä tässä kurssissa voidaan edellyttää. Suurin varmuus saadaan, kun testiloki tuottaa kokonaisia merkintöjä ja koko saavutettavuusgraafin, mikä kieltämättä on varsin kova vaatimus testin kattavuudelle.

Joskus sekä toteutuksessa ja Petri-verkossa on lukkiuma. Hyväksyttävää on vain se, että lukkiumat vastaavat toisiaan.

Tilaa testaavan mutex-algoritmin voi parhaiten käsittää keinona taata keskinäinen poissulkevuus thread-pohjaisessa toteutuksessa (tai yleisemmin: prosessien jakaessa RAM-muistia). Tällöin TCP/IP-tyylinen `send/receive` ei todellakaan ole hyvä idea. Semaforit tai jaetut muuttujat ovat usein tehokkaampi tapa kommunikoida. Kriittisen resurssin voi kuvitella olevan esim. tiedosto. (Keskinäinen poissulkevuus voitaisiin saavuttaa lukitsemalla tiedosto, mutta tämä olisi todennäköisesti tehottomampaa kuin edellä esitetty Petri-verkkoratkaisuun perustuva mutex-lähestymistapa.)

Löydätkö virheitä toteutuksesta?

Onko toteutuksessa mitään järkeä?