T-79.179 Rinnakkaiset ja hajautetut digitaaliset järjestelmät

Johdatus MARIA-työkaluun

Marko Mäkelä

10. helmikuuta 2003

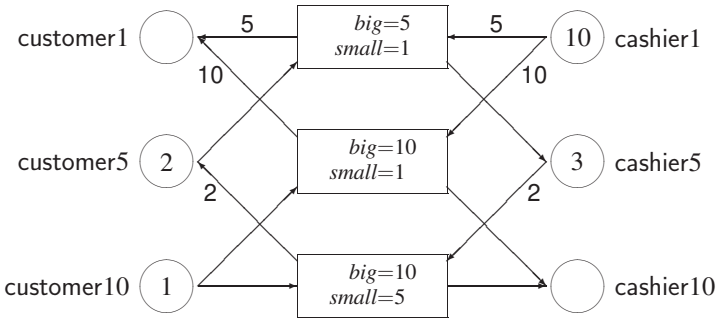
MARIA

MARIA (Modular Reachability Analyser for Algebraic System Nets) tarkastaa rinnakkaisten järjestelmien loogista yhtenäisyyttä, kuten tiedonsiirtoprotokollien tai jaetun muistin välityksellä keskustelevien rinnakkaisten prosessien toimintaa. Järjestelmät kuvataan tekstimuotoisella kielellä, jonka lausekkeet, tietotyypit, operaatiot ja kielioppi muistuttavat tavallisia ohjelmointikieliä, kuten C:tä ja Javaa.

MARIA voi joko simuloida vuorovaikutteisesti sille annettua järjestelmän mallia tai tutkia tyhjentävästi mallin kaikki saavutettavissa olevat tilat. Tyhjentävän haun aikana se voi tarkistaa, pätevätkö lineaarisen aikalogiikan kaavoilla kuvatut oikeellisuusvaatimukset. Hakua voidaan tehostaa kääntämällä malli ajettaviksi C-kielisiksi funktioiksi.

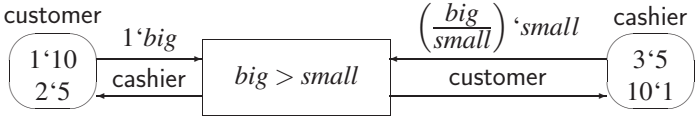
Esimerkki: rahan vaihtaminen (1/4)

Seuraava paikka–siirtymä-järjestelmä kuvaa tilannetta, jossa asiakas haluaa vaihtaa kolikoitaan pienemmiksi. Kuvassa asiakkaalla on kaksi ⑤:n kolikkoa ja yksi ⑩.



Esimerkki: rahan vaihtaminen (2/4)

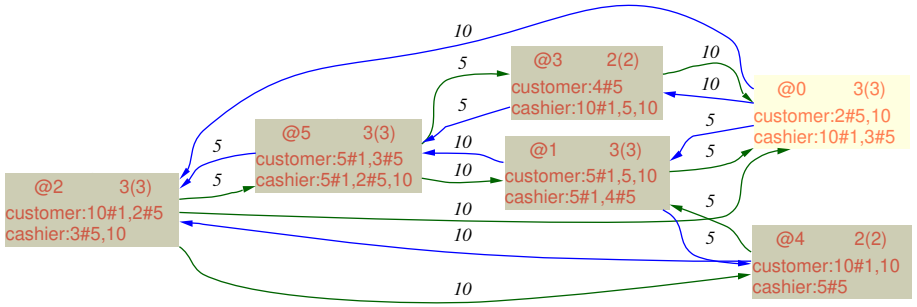
Paikka–siirtymä-järjestelmät toimivat melko matalalla tasolla. Usein on selkeämpää käyttää korkeamman tason formalismia, kuten korkean tason verkkoja:



Huomaa, että ②:n kolikko voitaisiin lisätä muuttamatta siirtymämääritystä.

Esimerkki: rahan vaihtaminen (3/4)

MARIA laski saavutettavuusgraafin seuraavalla kalvolla esitetystä mallista. Tulostetta on muokattu hieman värien lisäämiseksi ja kaarten nimien yksinkertaistamiseksi. Kaarilla olevat luvut ovat vaihtotapahtumaan liittyviä suurempiarvoisia rahoja, ja siniset kaaret kuvaavat asiakkaan kolikon vaihtumista pienemmiksi kolikoiksi.



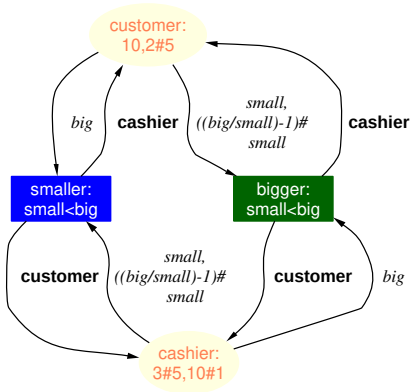
Esimerkki: rahan vaihtaminen (4/4)

```
typedef unsigned (1,5,10,50,100,500) money_t;

place customer money_t: 10,2#5;
place cashier money_t: 3#5,10#1;

trans smaller
in {
  place customer: big;
  place cashier: small, (big/small-1)#small;
}
gate big > small
out {
  place cashier: place customer;
  place customer: place cashier;
};

trans bigger
in {
  place customer: small, (big/small-1)#small;
  place cashier: big;
}
gate big > small
out {
  place customer: place cashier;
  place cashier: place customer;
};
```



MARIAn tietotyypit (1/2)

MARIAn tietotyyppijärjestelmä suunniteltiin korkean tason ohjelmointi- ja kuvauskielten tarpeisiin. Perustyypit ovat:

`bool`	`false,true`	totuusarvot
`char`	`'\0'..'\377'`	8-bittiset merkit
`unsigned`	0, 1, ...	luonnolliset luvut (yleensä 32 tai 64 bittiä)
`int`	..., −1, 0, 1, ...	kokonaisluvut (yleensä 32 tai 64 bittiä)
`enum`		luettelot

Rakenteisia tietotyyppejä voidaan muodostaa seuraavien rakenteiden avulla:

`struct { int a; char b; }`	`{0, 'x'}`	monikko tai tietue
`union { int a; char b; }`	`b = 'x'`	unioni (yksi vaihtoehdoista)
`int(0..2)[bool]`	`{0,1}`	taulukko
`bool[queue 2]`	`{false,false}`	0..2 totuusarvoa sisältävä jono
`char[stack 3]`	`{'z'}`	0..3 merkin pino

MARIAn tietotyypit (2/2)

Kaikkia tietotyyppejä voidaan yhdistellä mielivaltaisesti. Jos tarvitset monikolla indeksoituja jonoja sisältäviä taulukkoja sisältävän jonon, voit määritellä sellaisen.

Tietotyyppien arvoalueita voidaan rajoittaa, kuten edellisellä kalvolla tehtiin kokonaisluvuille: `int(0..2)[bool]`. Rajoitteiden ei tarvitse olla yhtenäisiä lukualueita, vaan ne voivat olla mielivaltaisia luetteloita: `unsigned(..3,10,15..20)`. Myös rakenteisia tietotyyppejä voidaan rajoittaa.

MARIA määrittelee suuruusjärjestyksen kaikille tietotyypeille. Mille tahansa arvolle `x` on määritelty seuraaja ja edeltäjä (`+x` and `|x`) siten, että suurimman arvon seuraaja on pienin arvo ja pienimmän arvon edeltäjä suurin arvo. Jos `x` on tyyppiä `unsigned(0..4)`, voit yksinkertaisesti kirjoittaa `+x` lausekkeen `(x+1)%5` sijaan.

MARIAn peruslausekkeet (1/2)

MARIAn syötekielen lausekkeet perustuvat voimakkaasti ohjelmointikieliiin C, C++ tai Java. Kielessä ei ole osoittimia, viitteitä, metodikutsuja eikä sijoituksiakaan* (ei edes sellaisia kuin `x*=3` tai `x++`).

Edellisen kalvon seuraajan ja edeltäjän lisäksi on joitakin uusia operaattoreita. Jotkut unaarioperaattorit ovat vain lyhennysmerkintöjä: `<int` on sisäänrakennetun `int`-tyypin pienin arvo, `>bool` on `true`, ja `#char` on `256`, erilaisten `char`-arvojen määrä.

C-kielen jos-niin-muutoin-operaattori `ehto?tosilauseke:muutoinlauseke` on yleistetty valintaoperaattoriksi. Jos `x` on tyyppiä `unsigned(0..4)`, lauseke `x?3:2:1:0:4` vastaa lauseketta `|x`.

*Petri-verkkojen lausekkeilla ei ole sivuvaikutuksia. Kaikki vaikutukset mallinnetaan siirtymillä.

MARIAn peruslausekkeet (2/2)

Unioniarvon aktiivinen komponentti voidaan selvittää binäärisen `is`-operaattorin avulla. Jos `x` on unioni, jonka komponenttien nimet ovat `a`, `b` ja `c`, ehto `x is b` pätee silloin, kun unionin toinen komponentti on aktiivinen.

Unaarinen `is` on tietotyyppimuunnos. Etumerkillinen kokonaisluku `c` voidaan muuntaa etumerkittömäksi kirjoittamalla `is unsigned c`. Negatiivinen `c` tuottaa virheilmoituksen.

Jonoille ja pinoille määritellyt unaariset `/` ja `%` kertovat käytetyn ja jäljellä olevan tilan. Binäärinen `+` lisää alkion ja unaarinen `-` poistaa alkion. Unaarinen `*` lukee alkion. Oletusarvoista käsittelykohtaa voidaan muuttaa hakasulkujen `[` ja `]` välisellä indeksillä: `-b[2]` on jonon tai pinon `b` sisältö, josta on poistettu kolmas alkio. Indeksointi alkaa nollasta, joka on indeksin oletusarvo.

Käyttöohjeessa on määritelty kaikki operaattorit yksityiskohtaisesti.

MARIAn monijoukko-operaatiot

Monijoukot ovat muuten kuin joukkoja, mutta monijoukko voi sisältää saman alkion moneen kertaan. Muodollisesti: mikä tahansa A:n monijoukko μ voidaan esittää $\mu : A \to \mathbb{N}$. ”Tavallinen” joukko $A' \subseteq A$ voitaisiin esittää $A' : A \to \{0,1\}$.

Vain pientä osaa MARIAn valmiista monijoukko-operaatioista tarvitaan perusmallintamiseen. MARIA muuntaa yksittäiset arvot monijoukoiksi automaattisesti. Monijoukkoja voidaan yhdistää pilkuilla, kuten lausekkeessa `10,2#5`, joka vastaa lauseketta `10,5,5`. Binäärinen `#`-operaattori monistaa monijoukon kaikki alkiot.

Monijoukkosumma voi olla hyödyllinen alkumerkintöjä esitettäessä. MARIAn kielellä joukko $\bigcup_{d\in\mathcal{D}}\bigcup_{e\in\mathcal{D}\setminus\{d\}}\{\langle d,e\rangle\}$ kirjoitetaan `D d: D e (e != d): {d,e}`.

Muita monijoukko-operaatioita käytetään yleensä vaativissa toiminnoissa, kuten kuvattaessa mallin haluttuja loogisia ominaisuuksia.

Tietorakenteiden määritteleminen: `typedef` ja `place`

Kaikki laskentajärjestelmät käsittelevät tietoa. MARIAssa on kaksi tietorakenteisiin liittyvää määritystä: `typedef` (tietotyyppimääritys) ja `place` (tallennusmääritys).

```
/** filosofeja tai haarukoita kuvaava tietotyyppi */
typedef unsigned (1..4) ph_t;
/** käytettävissä olevat haarukat (0..4); alussa kaikki paikalla */
place fork (0..#ph_t) ph_t: ph_t f: f;
/** ajattelevat filosofit (alussa kaikki) */
place thinking (0..#ph_t) ph_t: ph_t ph: ph;
/** nälkäiset filosofit (alussa ei ketään) */
place hungry (0..#ph_t) ph_t;
/** lounastavat filosofit (alussa ei ketään) */
place eating (0..#ph_t) ph_t;
```

`place`-määreiden hienouksia

Vaikka paikan kapasiteettia (sallittujen merkkien määrää) ei ole pakko määritellä, on hyvä tehdä niin, sillä rajoja voidaan hyödyntää analyysissä, ja rajat voivat auttaa virheiden löytämisessä. Sama pätee tietotyyppimäärityksiin: kaikkia tietotyyppejä ei ole pakko nimetä:

```
/** rajoittamaton paikka, jossa on pareja */
place pairs struct { int a; char b; }: 2#{ 0, 'x' };
```

Toisaalta on mahdollista määritellä vielä tiukempia rajoja. Lounastavien filosofien mallissa vain ne filosofit, jotka eivät ole nälkäisiä tai syömässä, voivat ajatella:

```
place thinking (0..#ph_t) ph_t: (ph_t f: f)
  minus place hungry minus place eating;
```


Siirtymämääritykset (1/3)

Tietomuunnokset ovat minkä tahansa laskentamallin kiinnostavin osa. Korkean tason Petri-verkoissa on vain yhdenlaisia toimintoja, korkean tason siirtymiä, joilla voi olla:

- paikallisia muuttujia (implisiittisesti määriteltyjä MARIAssa)
- syöte- ja tuloskaaria, joilla on monijoukkoarvoisia lausekkeita
- paikallisten muuttujien sidontoja rajoittavia ehtoja
- lisätoimintoja, kuten prioriteetteja, reiluusoletuksia tai epädeterminismiä

Siirtymämääritykset (2/3)

Lounastavien filosofien mallin siirtymät voidaan määritellä seuraavasti

```
trans left
in { place thinking: ph; place fork: ph; }
out { place hungry: ph; };
trans back
in { place eating: ph; }
out { place thinking: ph; place fork: ph, +ph; };
```

Harjoitustehtävä: Kuinka määritellään `trans right`?

Siirtymämääritykset (3/3)

Rahanvaihtoesimerkissä tuloskaarella luki `place customer`. Se on lyhennysmerkintä kyseiseen paikkaan liittyvien syötekaarten lausekkeiden kokonaisarvosta. Samassa mallissa rajoitettiin muuttujasidontoja `gate`-sanan avulla:

```
trans smaller
in { customer: big; cashier: small,(big/small-1)#small; }
out { cashier: place customer; customer: place cashier; }
gate big > small;
```

Miksi toisella syötekaarella ei lue yksinkertaisesti `(big/small)#small`? Suorituskykyään parantaakseen MARIA oikaisee hieman ja käyttää unifiointialgoritmia syötekaarilla. Muuttujan `small` arvoa ei voitaisi määrittää sellaisesta lausekkeesta, koska kerrointa `big/small` ei voi laskea tuntematta muuttujan `small` arvoa.

Epädeterminismi (1/2)

Korkean tason Petri-verkoissa epädeterminismiä mallinnetaan yleensä kaikki mahdolliset arvot sisältävällä vakiopaikalla, josta sidotaan satunnaiseksi tarkoitettu arvo muuttujaan. MARIAssa sellaisiin paikkoihin voidaan liittää `const`-määre:

```
/** kaikki arvonnan kohteena olevat filosofit */
place random (#ph_t-1) ph_t const: ph_t ph (ph != <ph_t): ph;
/** valitse satunnainen filosofi */
trans random
in { place random: p; /* ... */ }
out { place random: p; /* ... */ };
```


Epädeterminismi (2/2)

Jos epädeterminististä arvoa ei tarvita missään tulokaarilausekkeessa, voidaan käyttää tehokkaampaa epädeterminismioperaattoria lähtökaarella:

```
/** vaihda jonkin ajattelevan filosofin henkilöllisyyttä */
trans random_think
in { place thinking: ph; }
out { place thinking: ph_t p! (p != <ph_t && p != ph); };
```

Ehtoa ei ole pakko antaa. Jos muuttujan arvoa tarvitaan vain kerran, myös muuttujan nimi voidaan jättää pois, jolloin lauseke lyhenee muotoon `ph_t!`.

Tilaominaisuuksien kirjoittaminen (1/2)

Laskentajärjestelmän mallista ei ole paljon hyötyä, ellei jotakin sanota järjestelmältä edellytettävistä ominaisuuksista. Automaattinen työkalu voi kyllä tutkia mallin kaikki saavutettavissa olevat tilat ja kertoa laskentavirheistä ja lukkiumista, mutta se ei voi tehdä paljon enempää.

Oletetaan, että halutaan tarkistaa, voiko lounastavien filosofien pöytä tyhjentyä haarukoista. Tarkistus voidaan tehdä joko muuttamalla paikan `fork` kapasiteettirajoitusta tai kirjoittamalla `reject`-kaava:

```
/** käytettävissä olevat haarukat (aina vähintään 1) */
place fork (1..#ph_t) ph_t: ph_t f: f;
/** luettele ne tilat, joissa pöydässä ei ole haarukoita */
reject place fork equals empty;
```


Tilaominaisuuksien kirjoittaminen (2/2)

Oletusarvoisesti MARIA ei kerro havaitsemistaan lukkiumatiloista (joissa yhtään siirtymää ei ole vireessä). Määreen `deadlock true` vaikutuksesta MARIA raportoi kaikki havaitsemansa lukkiumat. Mielenkiinnottomat lukkiumat voidaan suodattaa kirjoittamalla ehto:

```
/** luettele kaikki lukkiumat, joissa pöydässä on haarukoita */
deadlock !(place fork equals empty);
```

Määreisiin `reject` ja `deadlock` liittyvät kaavat voivat olla mitä tahansa tilaehtoja. Käyttöohjeessa on lueteltu kaikki monijoukko-operaatiot.

Jos `reject-` tai `deadlock`-kaavan arvoksi tulee erikoisarvo `fatal`, analyysi pysähtyy. Operaattorien `||`, `&&` ja `?:` oiottu arvotus on hyödyksi tässä:

```
/** pysähdy, jos havaitaan odottamaton lukkiuma */
deadlock !(place fork equals empty) && fatal;
```

MARIA-mallien muokkaaminen

MARIA-malleja voi muokata millä tahansa tekstintoimittimella. Koska sekä `CR` että `LF` tulkitaan tyhjäksi väliksi, Apple- tai Microsoft-järjestelmien kanssa ei pitäisi ilmetä ongelmia.

GNU EMACS 20:n ja 21:n käyttäjille voi olla apua tiedostosta `pn-mode.el`. Se perustuu C:n kaltaisten kielten tilaan `cc-mode` ja tarjoaa älykkään sisennyksen ja syntaksin mukaisen värityksen, mikä helpottaa MARIA-mallien lukemista ja kirjoittamista.

MARIAn käyttöohjeessa on lyhyt ohje EMACS-asetusten muokkaamiseksi.

Jos tarvitaan muunlaisia tunnisteita kuin `[A-Za-z_][A-Za-z0-9_]*`, ne voidaan sisällyttää lainausmerkkeihin: `place "Möbiuksen nauha"`. Ainoa tunnisteissa kielletty merkki on `NUL` eli `"\0"`.

Marko Mäkelä

Mallien simulointi ja tutkiminen (1/3)

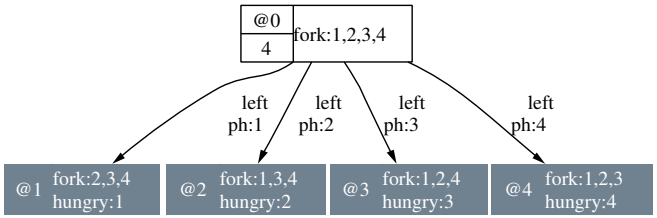
Monet kirjoittavat järjestelmän formaalin kuvauksen mieluiten vähän kerrallaan. MARIAn vuorovaikutteinen simulaattori tukee tällaista työtapaa. MARIA voidaan käynnistää milloin tahansa mallin syntaktisen oikeellisuuden tarkistamiseksi ja alkumerkinnän laskemiseksi. Tarkastellaan esimerkiksi lounastavien filosofien mallia:

```
>maria -m dinner.pn
@0$show
unprocessed state (
 fork:
  1,2,3,4
 thinking:
  1,2,3,4
)
@0$hide thinking; visual show
```

Marko Mäkelä

Mallien simulointi ja tutkiminen (2/3)

Jälkimmäinen komento näyttää tilan graafisesti ilman paikan `thinking` sisältöä. Kun hiiren vasenta näppäintä on painettu alkutilaa esittävän suorakaiteen kohdalla, tilanne näyttää seuraavalta:



Järjestelmän tila-avaruutta voidaan tutkia vuorovaikutteisesti painamalla hiiren vasenta näppäintä kiinnostavien tilojen kohdalla. Hiiren oikean näppäimen painaminen pohjaan tilan, kaaren tai graafin taustan kohdalla näyttää aiheeseen liittyvän valikon.

Marko Mäkelä

Mallien simulointi ja tutkiminen (3/3)

Lähes kaikki komennot voidaan syöttää graafisesti. Jos pidät enemmän tekstimuotoisesta liitännästä tai jos graafinen liitäntä ei ole jostain syystä käytettävissä, voit lukea käyttöohjeesta kyselykielen komentojen kuvauksen.

Eräs asia, jota ei voi helposti tehdä graafisesti, on aikalogiikan ominaisuuksien tarkastaminen. Oletetaan, että haluamme tietää, tyhjeneekö pöytä haarukoista kaikissa alkutilasta lähtevissä suorituksissa:

```
>maria -m dinner.pn -p lbt
@0$<> place fork equals empty
(command line):2:constructing counterexample
(command line):2:counterexample path:
(command line):2: @0 @1 @8 @8 @0
@0$exit
```

Graafinen esitys saadaan lisäämällä kaavan eteen `visual`.

Marko Mäkelä

Tyhjentävä saavutettavuusanalyysi (1/2)

Paras tapa tutkia `reject-` ja `deadlock`-ominaisuuksia ja etsiä kaikki mallin laskentavirheet on tyhjentävä leveys- tai syvyyshaku. Näissä tiloissa MARIA ei tulosta mitään (ellei sitä pyydetä kertomaan tutkittujen tilojen ja kaarten määrä `-E`-valitsimella), kunnes koko tila-avaruus on käsitelty tai tapahtuu vakava virhe.

Tehokkuuden vuoksi saavutettavuusgraafia esittävät levytiedostot on syytä pitää paikallisella levyllä. Luo alihakemisto paikalliselle levylle ja siirry sinne:

```
>mkdir /tmp/maria-$USER; cd /tmp/maria-$USER
>maria -b ~/dinner.pn -e exit
deadlock state @27
"dinner.pn": 34 states, 88 arcs
```

Syntynyttä graafia voidaan tutkia myöhemmin valitsimen `-g` avulla. Temporaaliominaisuuksiakin voidaan tarkastaa suoraan komentoriviltä käsin: `maria -m ~/dinner.pn -e '<> place fork equals empty' -e exit`.

Marko Mäkelä

Tyhjentävä saavutettavuusanalyysi (2/2)

Tutkittaessa pientä mallia, jolla on suuri tila-avaruus,* valitsin `-C` kehottaa MARIAa kääntämään mallin ajettavaksi C-koodiksi. Valitsimella on yksi parametri: sen hakemiston nimi, johon koodi kirjoitetaan.

Suuria malleja voi yrittää tutkia yhdistelmällä `-P` ja `-e breadth` tai `-e depth`. Kokeellinen `-P`-valitsin tekee saavutettavuusgraafin sijaan saavutettavien tilojen joukkoa keskusmuistissa esittävän hajautustaulun. Jos hajautustaulussa tapahtuu törmäys, jotkin tila-avaruuden osat jäävät tutkimatta. Tätä valitsinta käytettäessä ei voi tarkastaa elävyysominaisuuksia.

*Suurella mallilla voi olla pieni tila-avaruus; tässä ei ole nyrkkisääntöä.

Marko Mäkelä

Loppulause

MARIA yrittää olla helppokäyttöinen formaali työkalu rinnakkaisten ja hajautettujen järjestelmien käyttäytymisen tutkimiseen. Mallinnusformalismi on tehokas, koska se suunniteltiin korkean tason ohjelmointi- ja kuvauskielten tarpeisiin.

Analysaattori voidaan liittää sekä korkeamman tason formalismeihin (kuten ITU-T:n SDL-kieleen tai Java-kielen osajoukkoon) että matalamman tason formalismeihin, kuten matalan tason Petri-verkkoihin (LOLA, PEP, PROD) sekä nimettyihin siirtymäjärjestelmiin.

Lisätietoja on saatavissa MARIA-kotisivulta osoitteesta

http://www.tcs.hut.fi/maria/

sekä suoraan tekijöiltä.

Marko Mäkelä